Teaching People and Machines to Enhance Images

Floraine Berthouzoz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-168
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-168.html

October 10, 2013

Teaching People and Machines to Enhance Images

by

Floraine Berthouzoz

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Maneesh Agrawala, Chair
Professor Jitendra Malik
Professor Tapan Parikh

Fall 2013

Teaching People and Machines to Enhance Images

Abstract

Teaching People and Machines to Enhance Images

by

Floraine Berthouzoz

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Maneesh Agrawala, Chair

Procedural tasks such as following a recipe or editing an image are very common. They require a person to execute a sequence of operations (e.g. chop onions, or sharpen the image) in order to achieve the goal of the task. People commonly use step-by-step tutorials to learn these tasks. We focus on software tutorials, more specifically photo manipulation tutorials, and present a set of tools and techniques to help people learn, compare and automate photo manipulation procedures. We describe three different systems that are each designed to help with a different stage in acquiring procedural knowledge.

Today, people primarily rely on hand-crafted tutorials in books and on websites to learn photo manipulation procedures. However, putting together a high quality step-by-step tutorial is a time-consuming process. As a consequence, many online tutorials are poorly designed which can lead to confusion and slow down the learning process. We present a demonstration-based system for automatically generating succinct step-by-step visual tutorials of photo manipulations. An author first demonstrates the manipulation using an instrumented version of GIMP (GNU Image Manipulation Program) that records all changes in interface and application state. From the example recording, our system automatically generates tutorials that illustrate the manipulation using images, text, and annotations. It leverages automated image labeling (recognition of facial features and outdoor scene structures in our implementation) to generate more precise text descriptions of many of the steps in the tutorials. A user study finds that our tutorials are effective for learning the steps of a procedure; users are 20–44% faster and make 60–95% fewer errors when using our tutorials than when using screencapture video tutorials or hand-designed tutorials.

We also demonstrate a new interface that allows learners to navigate, explore and compare large collections (i.e. thousands) of photo manipulation tutorials based on their command-level structure. Sites such as `tutorialized.com` or `good-tutorials.com` collect tens of thousands of photo manipulation tutorials. These collections typically contain many different tutorials for the same task. For example, there are many different tutorials that describe how to recolor the hair of a person in an image. Learners often want to compare these tutorials to understand the different ways a task can be done. They may also want to identify common strategies that are used across tutorials for a variety of tasks. However, the large number of tutorials in these collections and their

inconsistent formats can make it difficult for users to systematically explore and compare them. Current tutorial collections do not exploit the underlying command-level structure of tutorials, and to explore the collection users have to either page through long lists of tutorial titles or perform keyword searches on the natural language tutorial text. We present a new browsing interface to help learners navigate, explore and compare collections of photo manipulation tutorials based on their command-level structure. Our browser indexes tutorials by their commands, identifies common strategies within the tutorial collection, and highlights the similarities and differences between sets of tutorials that execute the same task. User feedback suggests that our interface is easy to understand and use, and that users find command-level browsing to be useful for exploring large tutorial collections. They strongly preferred to explore tutorial collections with our browser over keyword search.

Finally, we present a framework for generating content-adaptive macros (programs) that can transfer complex photo manipulation procedures to new target images. After learners master a photo manipulation procedure, they often repeatedly apply it to multiple images. For example, they might routinely apply the same vignetting effect to all their photographs. This process can be very tedious especially for procedures that involve many steps. While image manipulation programs provide basic macro authoring tools that allow users to record and then replay a sequence of operations, these macros are very brittle and cannot adapt to new images. We present a more comprehensive approach for generating content-adaptive macros that can automatically transfer operations to new target images. To create these macro, we make use of multiple training demonstrations. Specifically, we use automated image labeling and machine learning techniques to to adapt the parameters of each operation to the new image content. We show that our framework is able to learn a large class of the most commonly-used manipulations using as few as 20 training demonstrations. Our content-adaptive macros allow users to transfer photo manipulation procedures with a single button click and thereby significantly simplify repetitive procedures.

Together these tools provide an automated system to create high quality step-by-step tutorials, allow users to explore and compare large collections of online tutorials, and facilitate repetitive procedures by automating the transfer of photo manipulations. As more and more instructional material appears online, we believe that providing such tools for learning, comparing and automating procedures will be essential to help people work efficiently with software tools.

To Marie-Paule Berthouzoz,
and in loving memory of
Kanya and Udal

Contents

List of Figures

List of Tables

Acknowledgments

First of all, I would like to thank my advisor, Maneesh Agrawala, for his support and mentorship throughout my graduate studies. I met Maneesh as an undergraduate exchange student. His energy and creativity appealed to me from the beginning and I have been fortunate to work with him ever since. I appreciate that Maneesh always treated me as a peer and allowed me to define and drive my own PhD. He encouraged me to use my creativity wisely on projects that combine my personal interests with research. This approach was not always an easy one to follow as it forced me to step away from many of the traditional research directions in graphics. However, I am now very grateful that Maneesh set me on this path. It taught me how to think outside the box and allowed me to pursue my personal interests as part of work, while having an impact on real-world problems I cared about. During my graduate studies, I have always sought new experiences, which led me to spend a lot of time visiting other research labs. I am thankful that Maneesh supported this choice and allowed me to travel as much as I did. Maneesh has been instrumental to my PhD; he has acted as a fantastic mentor, a loyal supporter, a thoughtful and reliable collaborator, and a good friend.

I am also grateful to my committee members, Jitendra Malik and Tapan Parikh, and all my collaborators, Mira Dontcheva, Raanan Fattal, Bjoern Hartmann, Takeo Igarashi, Wilmot Li, Gautham Mysore, Amy Pavel, and Steve Rubin. I especially would like to thank Wilmot Li. Wil has served as a mentor over the past five years and the completion of my PhD would not have been possible without his considerable contributions. He was always someone I could count on and his positive energy made every project fun.

During my graduate studies, I have been fortunate to spend some time at the Tokyo University, the Hebrew University, Columbia University and Adobe's Creative Technology Lab. I would like to acknowledge Takeo Igarashi, Raanan Fattal, Eitan Grinspun and David Salesin for welcoming me in their labs, and sharing their knowledge, expertise and time with me. In particular, I would like to thank Raanan Fattal who also served as a mentor. I felt drawn to Raanan's constant level of excitement, his rigorous approach, and his relentless optimism that every problem has a solution. In many ways, Raanan's attitude set an example for the researcher I would like to be.

Being the only woman in the computer graphics group for most of my PhD, I became very aware of how underrepresented women and other minorities are in computer science. I made it my mission to make a difference and encourage undergraduate women to major in computer science. I was fortunate to co-found CS KickStart with Colleen Lewis and the Berkeley-Stanford Research Meet-Up with Te-Yuan Huang. Leading these organizations was one of the most rewarding experiences of my PhD. I am grateful to so many EECS professors (in particular Ras Bodik, David Culler, Joe Hellerstein, Tsu-Jae King Liu, Claire Tomlin, David Wagner) for sharing our vision and believing in this program. I thank all the undergraduate organizers, Huda Khayrallah and Amy Tsai in particular, for making CS KickStart their own. I feel fortunate to have been part of this experience and I hope CS KickStart will continue to draw many women into computer science.

I would like to thank all my friends at Berkeley with whom I was fortunate to share this experience. In particular, I would like to acknowledge Maryam Kamgarpour, Gireeja Ranade, and Mobin Javed who have been my greatest supporters and loyal companions through the up and downs of graduate school. Finally, I would like to acknowledge my mother, Marie-Paule Berthouzoz, who always encouraged me to dream big and gave me the confidence that I could do anything.

Chapter 1

Introduction

Many of the tasks that we perform are procedural tasks. Examples include cooking, driving, reading or even manipulating an image. Procedural tasks involve a sequence of operations that one has to execute in order to achieve the goal of the task. For example, making a caprese salad is a procedural task that involves three consecutive steps. People first cut the tomatoes and mozzarella, then layer alternating slices of tomatoes and mozzarella on a plate while adding a leaf of basil between each, and finally drizzle the salad with oil and season it. People share instructions in the form of step-by-step tutorials for many different kinds of procedural tasks. For example, the site `food.com` has collected almost half a million cooking recipes. People are also sharing numerous tutorials for assembling and repairing physical objects, for making crafts, and to teach people how to use software. Table 1.1 lists a few example sites and the number of tutorials shared on each site.

Given the importance of procedural tasks in our everyday life and the large quantity of instruc-

Domain	Number of Tutorials
Recipes	
Food.com	475,000
Foodnetwork.com	49,000
Assembly and Repair	
Wikihow.com	150,000
Instructables.com	60,000
Crafts	
Cutoutandkeep.com	59,000
Craftgossip.com	2,000
Software	
Good-tutorials.com	33,000
Psd.tutsplus.com	20,000

Table 1.1: Tutorial sharing sites and number of tutorials shared on each site.

tional material shared online, it is crucial to understand the learning process, and the challenges that people encounter when acquiring procedural skills. Educators and psychologists have studied this question since the 1960s. They have developed several different models for how people acquire procedural skills [35, 37, 25, 79, 9, 6, 87, 101]. We consider one of these models that involves three consecutive learning stages.

1.1 Three Stages of Learning Procedural Tasks

Acquiring a new procedural skill requires a lot of practice. However, simply repeating the same task over and over again, does not ensure learning. Skill acquisition is a mental process that requires learners to fully comprehend the underlying structure of the procedure. For example, to master a new skill learners need to understand the effect that different types of inputs have on the result, and be able to adapt the operations of the procedure to new input. They also need to be able to recognize common strategies used when the procedure is applied in different contexts.

Several researchers have studied the process that learner's go through when acquiring a new procedural skill. The model proposed by Fitts et al. [35] distinguishes between three consecutive learning stages: the cognitive, associative and autonomous stages. Several other researchers [1, 2, 91, 101] further developed this theory. We define each of these three stages and illustrate them using cooking as an example of a procedural skill.

Cognitive Stage. In the cognitive stage, learners follow step-by-step instructions that describe the sequence of actions necessary to complete the task. Typically, the instructions are in the form of a static step-by-step tutorial that combines text and images. Alternatively, a person might also learn from a video tutorial or a life demonstration. Figure 1.1a shows instructions for making a carrot soup. In this recipe, each step necessary to make the soup is clearly described. Following these steps requires conscious effort and leads to high levels of cognitive activity. When reading the recipe, learners examine how the steps come together as a whole so as to correctly execute the procedure. Learners organize their newly acquired knowledge into parts known as schemas [8]. Schemas can be seen as "units" of knowledge, each relating to one particular aspect or concept of the procedure. In well-designed tutorials the schemas often correspond to the steps of the tutorial.

Associative Stage. In the associative stage, learners start comparing and synthesizing information across multiple recipes. In our cooking example, learners make a conscious effort to understand the similarities and differences between multiple carrot soup recipes (Figure 1.1b). They also start identifying common patterns or strategies in the procedures. For example, all soup recipes first heat up the carrots with butter and onions before blending them. However, some recipes add curry powder, others add ginger or even coconut milk. In this stage, people learn which steps of the recipe are optional and how parameters (e.g. cooking time) affect the result. Learners build a model in their head of the different ways the task can be done.

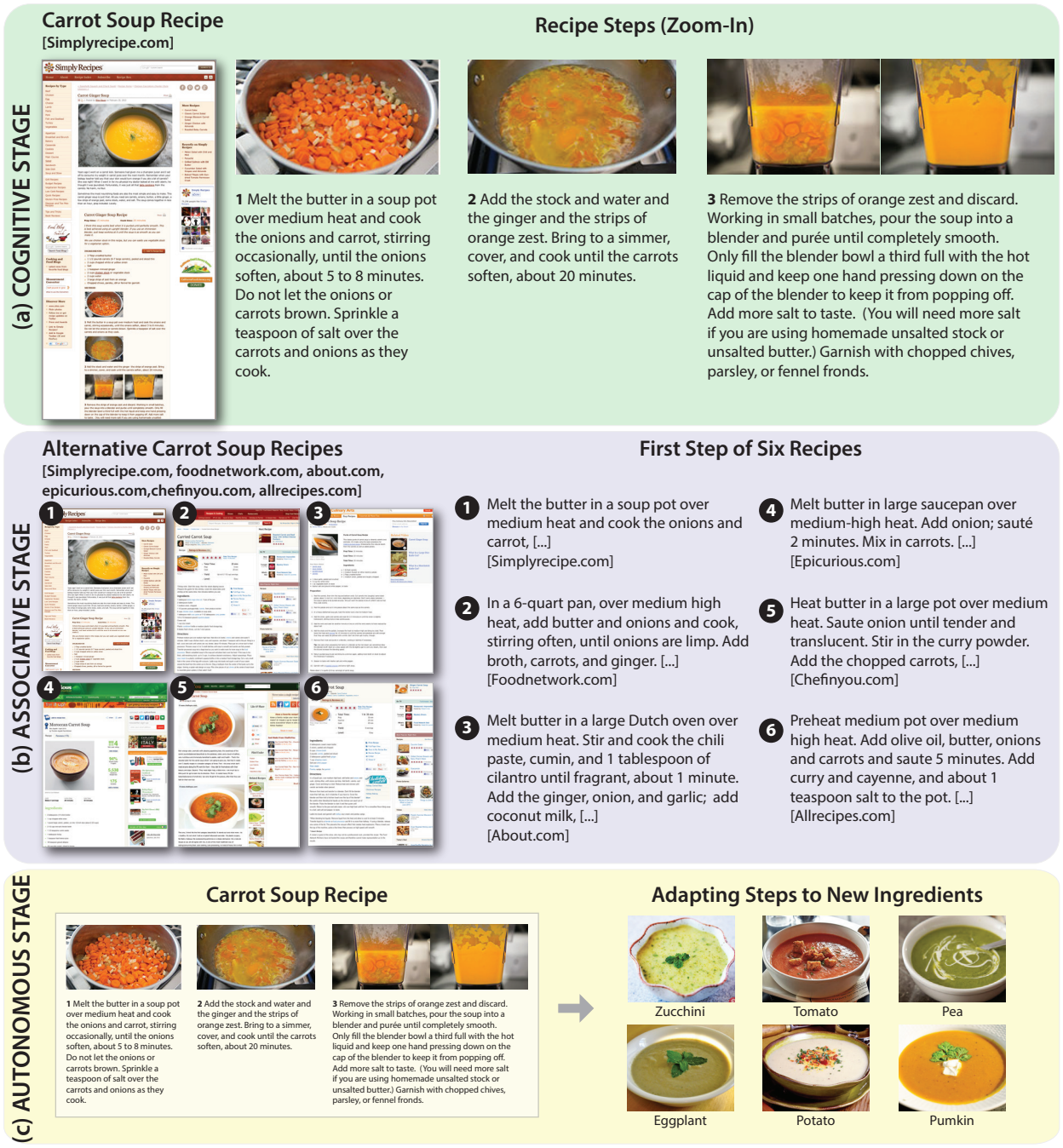


Figure 1.1: ***(a) Cognitive Stage.*** Learners follow step-by-step the instructions described in the carrot soup recipe. ***(b) Associative Stage.*** Learners compare six tutorials for cooking carrot soup. Step 1 of all six recipes first describes how to heat up the carrots with butter and onions. However, some recipes also add ginger (2,3), curry powder (5,6), chili paste and coconut milk (3), or cayenne (6). ***(c) Autonomous Stage.*** Learners do the task fluently, and can adapt the steps of the recipe to new ingredients such as zucchinis, tomatoes, peas etc.

Autonomous Stage. In the autonomous stage, learners have acquired the procedure and can transfer their knowledge to different tasks. For example, after learning how to cook a carrot soup, they can apply the same sequence of operations to other ingredients like tomatoes, peas or zucchinis (Figure 1.1c). In this stage, learners also become experts. They can perform the task fluently, without effort, and can adapt it as necessary to new ingredients. For example, they know how to adapt the recipe if they have a larger or smaller quantity of vegetables, and they know how to make the soup spicier or milder.

This three stage learning process applies to all types of procedural tasks. However, for each of these three stages, there are some open challenges that should be addressed to further facilitate the learning process.

- ***Challenge 1: Facilitating the Authoring of High Quality Instructions.*** For the cognitive stage, a major challenge is to facilitate the authoring of high quality instructions. Step-by-step visual tutorials are currently hand-designed and are very time consuming to create. Tutorial authors often do not take the time necessary to craft high quality tutorials. As a consequence, many of the tutorials shared online are poorly designed which can slow down or even prevent learning.
- ***Challenge 2: Enabling the Comparison and Browsing of Multiple Instructions.*** For the associate stage, a challenge is to enable the comparison and browsing of multiple procedural instructions (e.g. two different recipes for carrot soup). Currently there are no interfaces that help learners compare sequences of operations. Doing this comparison in one's head can be very difficult or even impossible if the procedure involves many steps. As a consequence, learners need a lot more practice to recognize patterns in similar procedures or may never be able to generalize the task to different types of inputs.
- ***Challenge 3: Automating the Transfer of Procedures.*** After learning a procedure, people often apply it over and over again. Repeatedly applying the same steps can be tedious for procedures that are complex and time-consuming. So a challenge for the autonomous stage is to automate the transfer of procedures. This challenge requires teaching the steps of the procedure to a machine and learning how to automatically adapt each step to new input.

The first two challenges arise when people are learning procedural knowledge. The third challenge arises when we try to teach a procedure to a computer. In this work, we address all three challenges in the context of photo manipulation procedures.

1.2 Photo Manipulations

Digital cameras, cell phones, or head-mounted cameras like the Google Glass make it easy to capture the world around us. About 380 billion photos are taken each year and 300 million photos are uploaded to Facebook every day [38]. As cameras become more and more ubiquitous and versatile, this number is predicted to further increase. Yet the images we capture directly in the camera often

fail to depict our intended interpretation of the scene. Good photographs usually require some editing of the raw images to highlight important elements and de-emphasize unimportant features.

As a result, more and more people use image processing software such as Adobe Photoshop or the GNU Image Manipulation Program (GIMP) to edit their raw imagery – to adjust contrast and color, blur backgrounds, remove blemishes and wrinkles, etc. Such photo manipulations are examples of procedural tasks; users execute a sequence of image manipulation operations to achieve the desired effect. Although Photoshop and GIMP provide access to many powerful image manipulation techniques, their interfaces are complex and difficult to learn and use effectively [40]. Therefore, users commonly turn to procedural instructions to learn how to use the software [90, 17].

As we saw in the previous section, learning procedural tasks, including photo manipulation procedures, is a three stage process where learners first follow a single tutorial (cognitive stage), then compare multiple tutorials (associative stage), and finally become fluent in the task and repeat it over and over again (autonomous stage). This learning process is often slow and difficult, because the instructions are poorly designed, learners need to compare tutorials in their head, and learners also need to manually apply repetitive procedures. In this thesis, our goal is to improve the learning process for photo manipulation procedures by addressing the three core challenges, i.e. by (1) facilitating the authoring of high quality tutorials, (2) enabling the comparison and browsing of multiple tutorials, and (3) automatically transferring procedures.

We address the challenges (1) and (3) in the context of face, landscape, and global manipulations, because when users take pictures, they often capture people and outdoor scenes. We informally analyzed 151 tutorials from three popular online photo tutorial sites (`chromasia.com`, `tripwiremagazine.com`, `good-tutorials.com`) and found that 30% (45/151) of the tutorials were global manipulations (i.e., manipulations that affect the entire image and are not tied to a specific object in the scene), 26% (40/151) were face manipulations, and 18% (27/151) were landscape manipulations. The remaining 26% (39/151) of the tutorials were applied to a wider variety of objects (e.g hair, body parts, cars). Therefore, face, landscape and global manipulations indeed make out most of the photo manipulations described in online tutorials. We address challenge (2) as part of a more general framework that is not dependent on any particular type of manipulation.

1.2.1 Facilitating the Authoring of High Quality Tutorials

In the cognitive stage, many users rely on *visual tutorials* to help them learn and work with photo manipulation applications. Visual tutorials are step-by-step depictions of the sequence of image processing operations required to produce specific effects. Figure 1.2 shows an example tutorial that teaches users how to correct for red eyes in an image as well as recolor eyes. Previous work [15, 59, 4, 81, 45] suggests that several characteristics make these visual tutorials particularly effective and thereby facilitate the cognitive stage of the learning process. For example, well-designed tutorials use screenshots to visually show the relevant interface widgets, the parameters required to invoke each operation and the resulting image after each step. In Figure 1.2a, the eye recoloring tutorial shows a screenshot of the toolbar in step 1 to help the user select the Red Eye Tool. Text descriptions, highlights, arrows and other annotations further guide users through

Figure 1.2: A tutorial taken from a book by Huggins [49] that teaches users how to correct for red eyes in an image as well as recolor eyes. The tutorial is well designed. It organizes the operations into steps, combines text descriptions with annotated screenshots of the application, and merges repetitive steps.

the tutorial. Good tutorials are also succinct; they eliminate unnecessary steps and condense repetitive steps. For example, step 2 (Figure 1.2b) groups two repetitive steps where the user first has to select the "Midtones" radio button and change the Cyan/Red slider, and then select the "Shadows" radio button and change the Cyan/Red slider again. As users manually perform each step from the tutorial, they learn the underlying image processing operations, and they can develop new techniques that adapt the operations to other instances of similar problems.

A variety of books [49, 102, 56] and websites contain thousands of such tutorials. Yet authoring tutorials is a tedious process in which the author must remember to save the relevant screenshots, add in the arrows and highlighting, write the text descriptions, and then lay out the screenshots in a step-by-step manner. As a result, many websites provide poorly designed tutorials with few or no screenshots and annotations. The eye recoloring tutorial of Figure 1.3 that we downloaded from the Web only shows the before and after images of the manipulation, but does not show images for the intermediary steps or the widgets that a person has to manipulate in each step. Alternatively, many websites simply provide screencapture videos of the manipulation instead of static tutorials. However, prior studies have shown that video-based instructions are far less effective than static tutorials because they force users to work at the pace of the video rather than working at their own pace [83, 42]. Users typically need to rewind the tutorial several times and do not retain their learnings a week later [83].

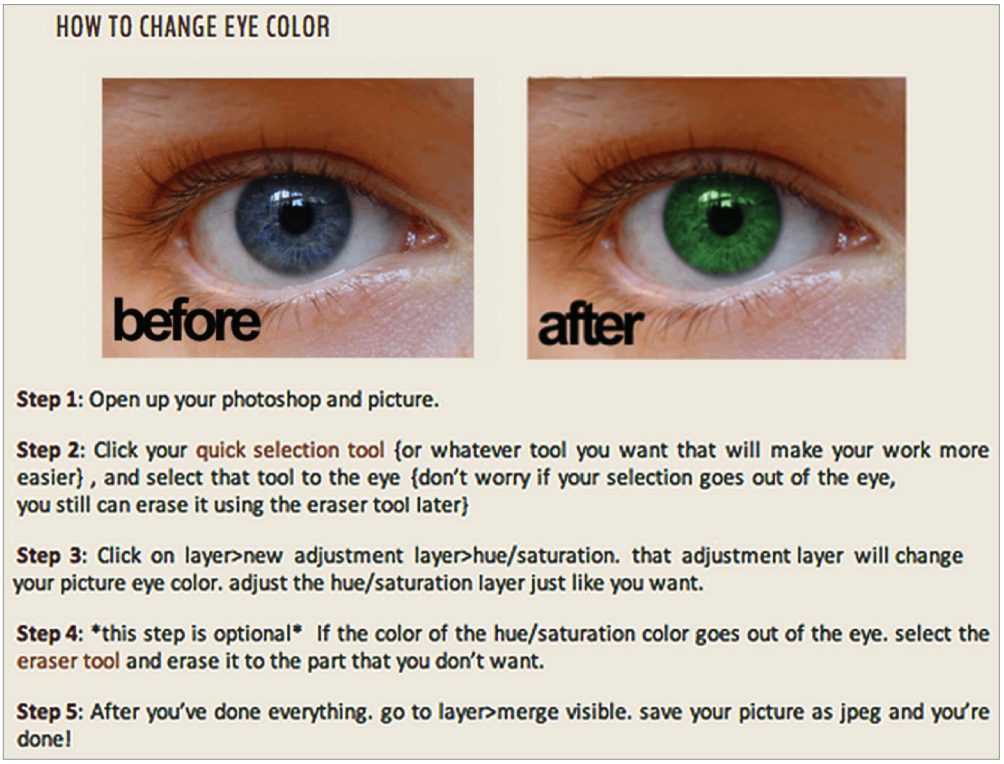


Figure 1.3: An eye recoloring tutorial that we downloaded from the web [100]. This tutorial is poorly designed. It does not show images for the intermediary steps or the widgets that a person has to manipulate in each step.

In Chapter 3, we present a system for automatically generating high-quality static tutorials by demonstration. We design the tutorials based on guidelines in the domain of instruction design [15, 59, 4] and cognitive science [81, 45], so as to facilitate the cognitive stage of the learning process. We generate the tutorials from a single demonstration and thereby make them as easy to create as screencapture video tutorials.

1.2.2 Comparison and Browsing of Multiple Tutorials

Photo manipulation programs often provide many different alternative procedures to achieve the same end goal. For example, there are many different ways to recolor the hair of a person in an image using Photoshop. In the associative stage, learners compare the procedures of multiple tutorials. After users learn one photo manipulation procedure for hair recoloring, they compare their learned procedure to other ones that accomplish the same task. As they understand more and more about their similarities and differences, they build a model in their head of the different ways the task can be completed. Learners may also compare procedures across different tasks (e.g. hair

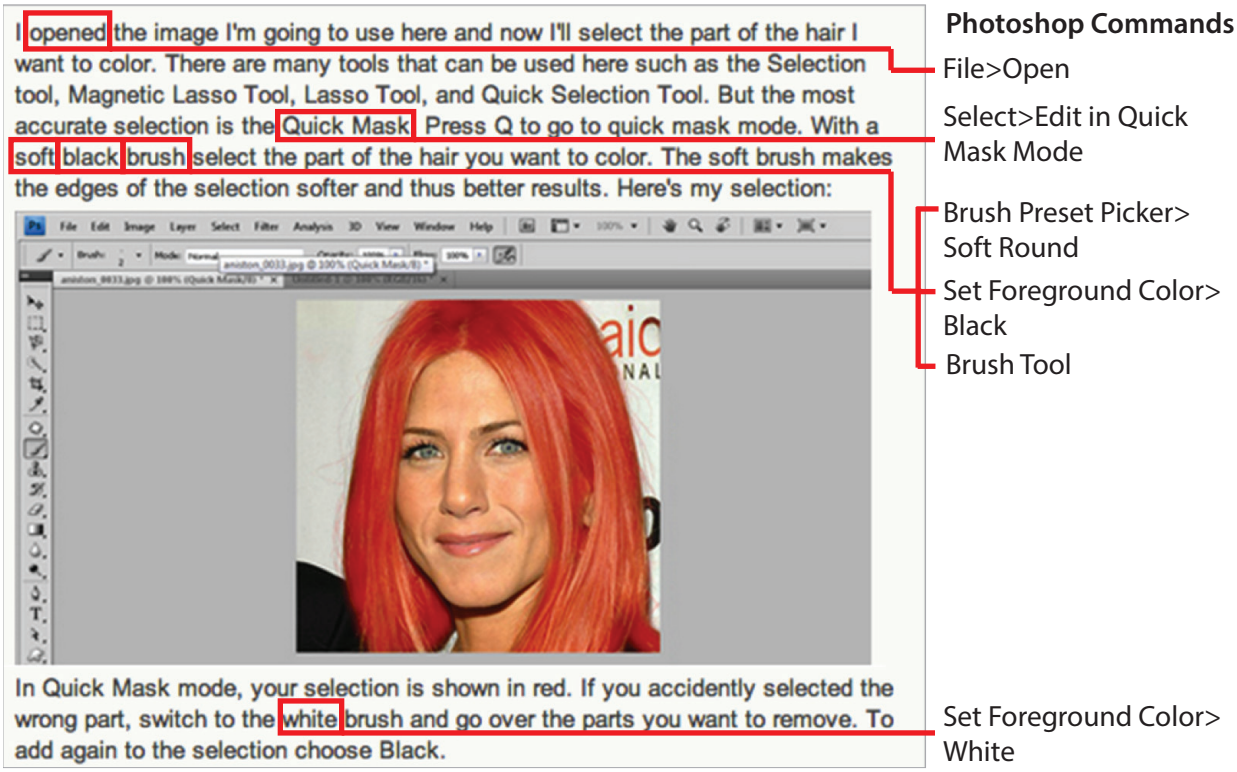


Figure 1.4: Step 1 from an online tutorial on hair recoloring, and a list of the corresponding Photoshop commands described in this step. The user has to very carefully read the text to identify the six Photoshop commands that he/she has to execute to accomplish the task. Furthermore, some commands such as the reference to the "soft black brush" command in the text description, actually correspond to three different Photoshop commands that a user has to execute in the application. Identifying the commands in such online tutorials can be very difficult.

recoloring and eye recoloring) to identify common strategies that are useful for many different types of manipulations. For example, the sequence of Photoshop operations "Quick Mask, Brush Tool, Invert, Hue-Saturation" can be used to recolor hair, but also appears in tutorials for car and garment recoloring. It is therefore a common strategy used to recolor many different types of objects. Identifying these strategies allows learners to generalize procedures beyond the specific task described in the tutorial.

Users often search online photo manipulation tutorial collections such as `tutorialized.com` or `good-tutorials.com` to find multiple tutorials to compare. These sites collect tens of thousands of tutorials and provide access to a very large and diverse collection of photo manipulation procedures. However, the large number and inconsistent formats can make it difficult for users to systematically explore and compare these tutorial collections. Current websites simply organize tutorials by high-level categories such as Photo Effects or Web Layouts. To explore the collection

users must either page through long lists of tutorial titles within each category or perform keyword searches on the natural language tutorial text.

Furthermore, each photo manipulation procedure is defined by its sequence of commands. These commands are crucial; properly following the tutorial requires executing them correctly. However, even identifying the sequence of commands in a single online tutorial can be challenging. Online tutorials use narrative text along with interface screenshots to describe the commands necessary to achieve a specific manipulation (Figure 1.4). But the user has to very carefully read the natural text to identify the commands. For example, in Figure 1.4 the first step of recoloring the hair of a person describes six different commands within two paragraphs. Similarly, when comparing multiple tutorials, the user also has to carefully identify the command sequence of each tutorial and then manually align the sequences. Current websites simply do not exploit the underlying command-level structure (i.e., the sequence of software commands) of the tutorials to help users navigate a single tutorial, explore collections of tutorials, or compare multiple tutorials.

In Chapter 4, we present a new browser interface to help learners explore and compare collections of photo manipulation tutorials based on their command-level structure. Our browser indexes tutorials by their commands, identifies common strategies within the tutorial collection, and highlights the similarities and differences between sets of tutorials that execute the same task. These tools facilitate the associative stage of learning by helping users better understand the underlying structure of photo manipulation programs.

1.2.3 Automatically Transferring Procedures

In the autonomous stage, after users master a photo manipulation procedure, they often repeatedly apply it to multiple images. For example, a photographer might routinely apply the same procedure to enhance the orange hues of sunsets, create a lomography-style vignetting effect, or improve skin-tones. This process is very tedious and time-consuming especially for photo manipulation procedures that involve many steps.

To facilitate repetitive procedures, Photoshop and GIMP provide basic macro authoring tools that allow users to record and then replay a sequence of operations. Yet, the macros authored today are extremely brittle; they are limited to executing exactly the same operations as in the recording and cannot adapt to new target images. Thus they are inappropriate for many common photo manipulations. For example, a macro designed to correct skin-tone for one image will likely fail for new images simply because the skin is in a different location. In addition, the parameter values of the skin-tone color adjustment operation depend on both the color of the skin and the overall color cast of the target image. The parameters used to correct the skin-tone of a dark-skinned person in a photograph with a blue cast ruin the skin-tone of a light-skinned person in a photograph with an orange cast (Figure 1.5).

In Chapter 5, we present a more comprehensive approach for generating content-adaptive macros that can automatically transfer the operations to new target images. We use machine learning to adapt the parameters of each operation to the new image content. In the example of Figure 1.5, our macro automatically adapts the skin-tone correction manipulation to the new location of the skin, and accounts for the light skin of the subject when adjusting the parameters.

Figure 1.5: A user demonstrates a skin-tone correction manipulation on a photograph of a dark-skinned person with a blue cast (a-b). Directly copying the same color adjustment parameters to correct the skin-tone of a light-skinned person turns his skin orange (c-d). Our content-adaptive macro learns the dependency between skin color, image color cast and the color adjustment parameters to successfully transfer the manipulation (e).

Our content-adaptive macros allow users to transfer photo manipulation procedures with a single button click and thereby significantly simplify repetitive procedures.

1.2.4 Computerized Tools for Facilitating the Learning Process

Together these tools facilitate the learning process and address the three core challenges by allowing users to automatically create high quality step-by-step tutorials by demonstration, providing an interface for exploring and comparing large collections of online tutorials, and facilitating repetitive procedures by automating the transfer of photo manipulations. As more and more people share software instructions online, providing tools for learning, comparing and automating procedures will be essential to teach people new procedures and help them work efficiently with software tools. We believe that there are many possible directions for future work that build on the ideas presented in this thesis. In Chapter 6, we describe how computerized tools can play a large role in facilitating

the learning process for many domains beyond photo manipulation procedures and many different types of instructions beyond procedural instructions. The focus of this thesis, however, is to describe computerized tools that teach people and machines to enhance images

Chapter 2

Related Work

We facilitate the learning process by automatically generating step-by-step tutorials by demonstration, and by building new interfaces to help learners compare multiple photo manipulation procedures. We also present a framework for automatically transferring photo manipulation procedures to new images. For each of these three components, we build on several areas of related work.

2.1 Generating Tutorials

The step-by-step tutorials that are featured in books and online are typically hand-designed. Creating high-quality instructions is a very tedious process, where the tutorial author has to take all the relevant screenshots, annotate them, write the text descriptions and lay out the tutorial. Only very few prior approaches attempt to automatically generate such instructions. The prior work in this domain is limited to generating graphical histories and interactive tutorials.

2.1.1 Graphical Histories

Graphical histories depict the operation history as a user is demonstrating a task. Although they are not designed to be used as instructions, they provide a step-by-step representation of the procedure. Early work on the visual depiction of these histories visualizes each operation using a screenshot and text in a comic-strip style [63, 70, 77, 76]. These techniques present snapshots of the main application window in each panel of the strip. Some of these systems crop and compose together multiple screenshots into a single panel to further emphasize the important operations in the history. More recent work has added richer annotations, such as arrows and highlights, to the visual depictions of the application state [10, 98, 78]. Such visualizations and annotations are also crucial for instructional material as they illustrate the task the user must perform. We build on these techniques and visualize procedures using storyboard-style annotations when automatically generating tutorials for photo manipulations.

Many of the approaches [63, 76, 98] used to generate graphical histories are designed to help users selectively undo their work. As a result they include rules for grouping operations into semantically meaningful undo units. This grouping also reduces the length of the comic-strip style history visualizations. In the context of learning, it is important to group similar steps to provide learners with succinct instructions that omit unnecessary operations. We therefore use similar grouping rules to generate step-by-step photo manipulation instructions that are easy to understand and follow. However, unlike graphical histories our end goal is to produce instructional material. We thus also include text descriptions and other semantic information to help users follow the instructions.

2.1.2 Interactive Tutorials

Interactive tutorials such as Kelleher and Pausch's Stencils [57] can be very effective, as they allow users to stay in the context of the application and execute steps directly from the tutorial. But generating effective interactive tutorials like Stencils can be a very manual and tedious process. Bergman et al. [10] solve this problem with a demonstration-based approach for authoring interactive tutorials or wizards. However, their wizards are designed for the Eclipse programming environment and are presented primarily as text scripts with limited support for graphical highlighting. In contrast, our work focuses on facilitating the learning process of photo manipulation procedures. Our goal is to generate static instructional tutorials for novice users. Our tutorials are generated fully automatically and combine text descriptions with visual annotations to guide a user through the steps of the tutorial.

2.2 Comparing Instructions

Online tutorial collections provide access to thousands of photo manipulation tutorials. Each tutorial describes the sequence of operations required to complete the manipulation. To compare multiple tutorials, users need to compare their command sequences. However, online tutorials are typically written in natural text and therefore the command sequences are not readily available. Prior work explores several approaches to extract the command-level structure from tutorial. Once the command sequences are available, researches have also suggested new interfaces for tutorial search and comparison that make use of this underlying structure.

2.2.1 Extracting Command-Level Structure

To help learners compare multiple tutorials and understand the similarities and differences in their steps, our goal is to extract the command-level structure from pre-existing text-based tutorials. Our extraction algorithm is inspired by earlier work on using string matching [88], grammar processing [67], and machine learning techniques [36, 65] to identify text references to commands in software tutorials. Laput et al. [65] achieve the best precision/recall rates using a Conditional Random Field classifier. However, their method requires a relatively large set of 400 training tuto-

rials. Fourney et al. [36] focus on command extraction with sparse training data. They use a Naïve Bayes classifier with Witten-Bell smoothing and require only 35 training tutorials while still maintaining good precision/recall. Both methods require exact string matches between the command name (as specified in the Photoshop interface) and the text description in the tutorial. However, many tutorials use colloquial references to commands. For example, a tutorial might say "Draw a stroke" instead of naming the actual Paint Brush command. When extracting the sequence of commands from text-based online tutorials, our work extends the approach of Fourney et al. [36] to identify colloquial references with sparse training data. Having access to this command-level structure allows us to build new interfaces for tutorial comparisons that are designed to facilitate the associative stage of learning.

Users also share software tutorials as screencapture video recordings. Sikuli [104], Prefab [23] and Pause-and-Play [86] apply computer vision techniques to reconstruct GUI-level interface operations (e.g. menu navigation, button presses, widget clicks, etc.) from such screenshots. However, methods that rely solely on visual GUI output may not be able to robustly identify commands like keyboard shortcuts that do not have visual effects. Our work focuses on text-based tutorials. In contrast to screencapture videos, text-based tutorials usually describe every command that must be executed.

2.2.2 Using Command-Level structure to Improve Tutorial Search and Comparison

Several previous methods use command execution histories to provide more effective application help and tutorial comparison. Ekstrand et al. [30] use the execution history and currently active tools to augment Web search and help users find more relevant tutorials. IP-QAT [74] and LemonAid [20] use recent commands and application context to recommend entries in application-specific question answering systems. CommunityCommands [75] recommends commands for design software based on collaborative filtering algorithms. Unlike our work, these tools do not extract information from existing text-based tutorials and they do not enable browsing or comparing large collections of tutorials.

Closest to our work is Delta [60], an interactive tool that summarizes and compares tutorials based on their command-level structure. Delta does not extract the commands automatically and relies instead on manual transcription. It is also explicitly designed for a small corpus (30 tutorials) and can therefore not handle the large tutorial collections on the Web. In contrast, our work provides automatic command extraction and includes a browsing and tutorial comparison interface that scales to a much larger corpus (2500 tutorials). Unlike previous methods, our interface allows users to directly compare the command-level structure of a set of tutorials and identify patterns within the command sequences of the entire tutorial collection. Our interfaces are designed to facilitate the associative stage of learning by allowing learners to better understand the similarities and differences between multiple tutorials.

2.3 Automating Procedures

Many different techniques exist to transfer procedures to new input. They range from general purpose programming by demonstration techniques that can transfer a demonstrated software procedure to new input, to methods specifically designed for transferring photo manipulation procedures. We review the most relevant transfer algorithms.

2.3.1 Programming by Demonstration

Researchers have developed demonstration-based techniques for creating scripts or programming macros for a variety of application contexts [21, 69] including desktop [77, 66], web [71, 14], and 2D graphics [63, 70]. A challenge common to many programming by demonstration techniques is to generalize the resulting program so that it can automatically work with new inputs. While many of these systems force users to manually describe how program parameters and arguments should generalize, there are some notable exceptions. Chimera [63] uses rules and heuristics to automatically generalize demonstrations, while DocWizards [10] learns the generalization from multiple example demonstrations. However, these techniques are not designed to help users transfer photo manipulation procedures.

2.3.2 Image-Based Transfer of Visual Properties

A recent trend in image manipulation research has been to develop algorithms for transferring certain visual properties of an example image to a target image. For example, image analogy methods [46, 28, 26] take a neighborhood-based approach to transferring low-level texture properties and can imitate non-photorealistic rendering styles. Bae et al. [7] use a histogram-based approach to transfer contrast and thereby replicate the overall look of the example image. Reinhard et al. [89] adjust the statistics of the color distributions of the target image to match those of the example. Flash/no-flash techniques [29, 84, 62] transfer lighting and detail between an image taken with a flash and one without. While these techniques inspire our work, each of them was designed for one specific type of photo manipulation procedure. Our goal is to build a more general framework that can automatically transfer most photo manipulation procedures. Furthermore, these approaches deal with pixel-level models of the images and do not have access to higher-level information such as the content of the image or the set of commands an author used to create the example image. As a result they cannot adapt operation-level macros to work with new content.

2.3.3 Content-Specific Transfer Algorithms

Researchers have also developed techniques to transfer visual properties for specific types of images. For example, human faces are important elements of many photographs, and researchers have explored image-based techniques to transfer expressions and lighting [72], makeup [41], beards [80] and entire faces [13] from one image to another. These techniques typically rely

on identifying corresponding facial regions (e.g. mouth, eyes, skin, etc.) between the two images either manually or using face-recognition algorithms, and then applying manipulation specific pixel-level transfer algorithms to these regions. Our work also takes advantage of automatic image labeling, including face and outdoor scene recognition in our implementation, to transfer photo manipulation procedures to new content. However, our goal is to build a framework that is designed to work with generic labels and landmark points, so that users can transfer any manipulation that they have learned. Our algorithmic techniques are therefore not specific to a particular type of object such as the face.

2.3.4 Operation Transfer Algorithms

A number of researchers have developed algorithms that transfer operations rather than pixel-level properties. Kang et al. [54] present a technique to transfer color and contrast adjustment operations. Given a new target image, they find the nearest neighbor amongst a set of 5000 training images and then apply the corresponding adjustment parameters to the new target. A drawback of their approach is that learning a new manipulation requires a new set of 5000 training examples. Hasinoff et al. [43] use an image-based approach to transfer clone brush operations. They rely on finding pixel-level matches between the training and target images. Thus, their approach is designed primarily for use with image collections from the same photo shoot. Compared with these methods, our framework is more comprehensive as it learns to adapt a wide variety of photo manipulations comprised of selections, strokes and image adjustment operations. Our approach usually requires about 20 training demonstrations and can transfer local edits such as partial selections and brush strokes across images from different photo shoots. Thus, our content-adaptive macros are more suitable for sharing and reuse by other photographers.

Chapter 3

Authoring Tutorials

In the cognitive stage, learners acquire new skills by following step-by-step the instructions that they are given. They primarily rely on visual tutorials to learn photo manipulation procedures. Tutorials in books and on websites are typically hand-crafted by a tutorial author. However, putting together a high quality step-by-step tutorial is a time-consuming process that requires a lot skill. As a consequence, many online tutorials are poorly designed.

We present a system for automatically generating high-quality static visual tutorials by demonstration. We design the tutorials based on guidelines in the domain of instruction design [15, 59, 4] and cognitive science [81, 45], so as to facilitate the cognitive stage of the learning process. Figure 3.1 shows an eye recoloring tutorial generated with our system. We generate these tutorials from a single demonstration and thereby make them as easy to create as screencapture video tutorials. As users demonstrate a manipulation technique, our system records their actions in the interface (e.g., clicking buttons, moving sliders, etc.) and the corresponding image processing operations in the application (e.g., blur region, set brush width parameter, etc.). In addition, our system applies automated image labeling to identify semantically important regions (e.g., eyes, mouth, sky, ground, etc.) in the photograph. Finally our system combines information from the recordings and the image labeling to produce succinct, step-by-step, visual tutorials that include images, annotations, and content-specific text descriptions.

One limitation of our tutorials is that they cannot explain why users must perform each operation. Our automatically generated tutorials are closer to annotated descriptions or histories of the operations the users must conduct to complete the manipulation. In contrast, hand-designed tutorials found in books often explain why each step must be performed. Nevertheless, a user study comparing our automatically generated tutorials to hand-designed tutorials and screencapture video recordings with voiceover explanations, finds that users are 20–44% faster and make 60–95% fewer errors using our tutorials. Moreover, our tutorials are generated in a fraction of the time required to manually capture, annotate and layout screenshots, or record voiceovers. Thus, our automatically generated tutorials could also serve as a first step that authors could supplement with further explanations describing the reasoning behind each step.

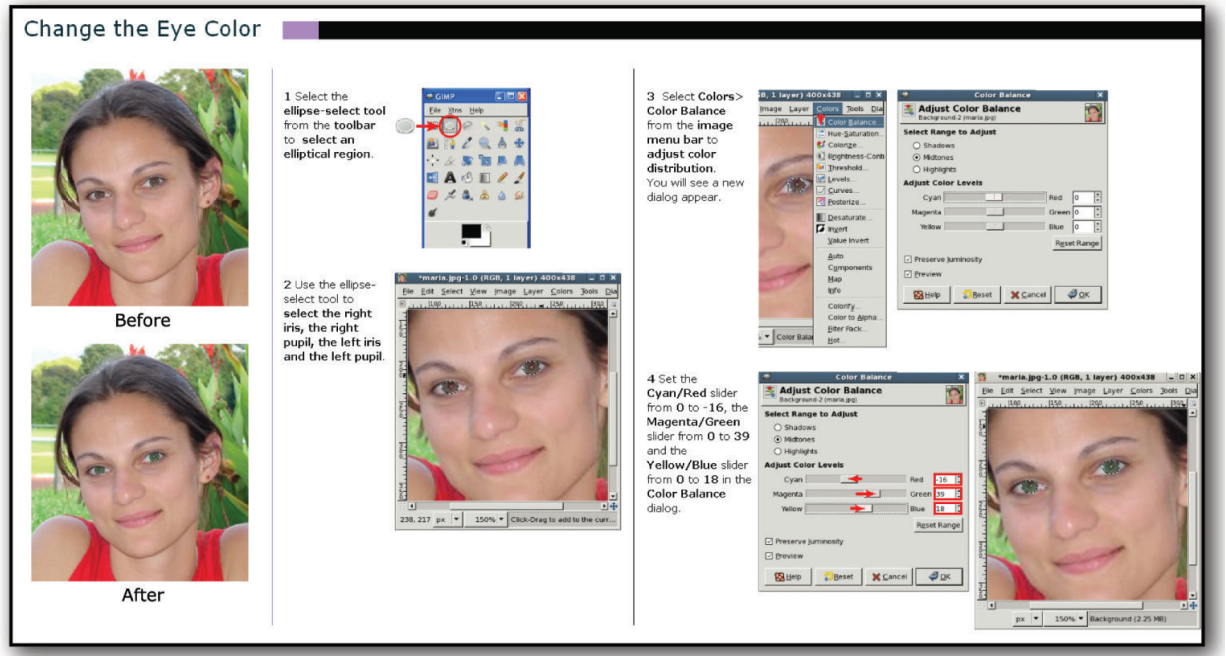


Figure 3.1: As a user demonstrates an eye recoloring portrait manipulation technique, our system records his actions in the interface and the corresponding image processing operations in the application. It then combines information from the recordings with image recognition to automatically generate a succinct, step-by-step visual tutorial that others can use to learn the technique.

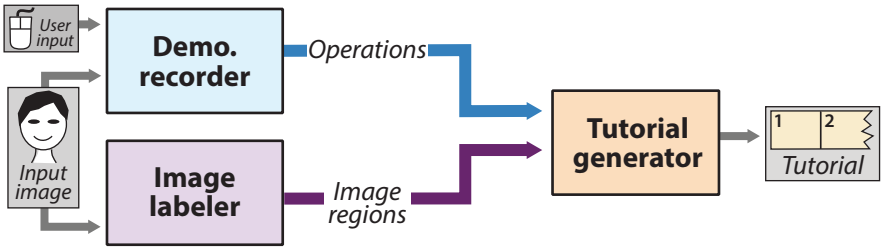


Figure 3.2: System overview.

3.1 System Overview

Our system consists of three main components shown in Figure 3.2:

Demonstration recorder. We have instrumented the GIMP open source photo manipulation application to record the state of both the interface and the application during an image manipulation session. While the ingimp project [99] also instruments GIMP to record such actions and

operations, we could not adopt ingimp as our recorder because ingimp is explicitly designed to prevent reconstruction of user work sessions from the usage data it records.

Image labeler. Many photo manipulation techniques involve localized edits to semantically important regions of an image. We use automated image labeling techniques to detect such regions and combine this information with the recorded sequence of user interactions to generate descriptive text instructions for each step of our tutorials.

Tutorial generator. Using both the recorder and labeler output, our tutorial generator automatically creates step-by-step tutorials of the demonstrated manipulation techniques. Each step includes text descriptions, screenshots of the user interface, and in some cases, visual annotations such as highlights, arrows, and callouts.

Although our implementation is based on GIMP, our approach builds on a higher-level representation that treats photo manipulations as a sequences of selection, image processing and brush stroke operations. In Section 3.2 we describe this general representation that is common to many photo manipulation applications. We then describe each component of our system in Sections 3.3–3.5.

3.2 Photo Manipulation Software Structure

In most photo manipulation applications, including Photoshop and GIMP, users can apply a wide variety of operations to select a subregion of the image and then filter, transform, or edit the selection. We distinguish between two types of operations. *Global operations* affect the entire selection, or the entire image if no region is selected. They consist of image processing operations such as adjusting the hue-saturation or applying a Gaussian filter. *Local operations* are all operations that users apply directly on the active layer by using a tool and clicking and dragging the tool in the desired image area. They consist of brush stroke operations such as the paint brush or the healing brush, and selection operations such as the lasso select or by color select. Users can change the application state in three ways and we must record all three to capture a complete representation of a photo manipulation demonstration.

Set operations. To operate on an image, users must first set the operation. Setting a global operation usually requires traversing the menu hierarchy, while setting a local operation usually requires clicking on a tool in the toolbar.

Set parameters. After setting the operation, users can adjust its parameters via interface widgets that appear in dialog windows spawned when the operation is first set or in toolbars located around the edges of the main window. The primary difference between global operations and local operations is that local operations treat the pointer position as a parameter that is set whenever the pointer is in the selection region and the mouse-button is down.

Commit operations. Committing a global operation usually requires clicking the OK button in the operation dialog. In contrast, a local operation is committed whenever the pointer is in the selection region and the mouse-button is down. Thus, dragging a tool across the active layer corresponds to both setting the position parameter for the local operation and simultaneously committing the local operation. Performing an *undo* operation cancels the previous commit operation.

3.3 Demonstration Recorder

Our demonstration recorder is designed to record all changes in the *interface state* (e.g., moving sliders, clicking menu options, entering text, etc.), as well as the resulting changes in the *application state* (e.g., setting the hue-saturation operation, setting the saturation parameter, etc.) during an image manipulation session. Our system requires both types of state information to properly interpret and depict the demonstrated photo manipulation. We capture interface state in a GIMP specific manner and use it to show users how to adjust the widgets in GIMP to produce the desired effect. We capture the application state using the higher-level representation presented in Section 3.2 and interpret this representation to generate succinct tutorials. As the user demonstrates a manipulation, we record the following information on every mouse-click, mouse-drag, mouse-release or keypress:

Interface state. We record information about the widget the user is manipulating, including the *window name* of the window that has focus, *widget type* (e.g. slider, button, radio button, checkbox, drop-down list, menu item, toolbar item, etc.), *widget position*, *widget size* and the *widget state*. We also record a screenshot of the window that has focus. Because taking screenshots is relatively slow, we only capture them at the beginning and end of a mouse-drag and not at every mouse position during the drag.

Application state. Whenever a new operation is set, we record the *name* and a *description* of the operation. For the description, we use the tooltip text associated with toolbar buttons and menu items. Whenever an operation parameter is set, we record the *parameter value*. Finally, whenever an operation is committed, we record a few important application-level state variables including the *active layer* the user is working with, the *selection mask* within the layer (by default, the selection mask is the entire image), the *foreground paint color*, the *background erase color*, etc.

Figure 3.3 shows an example recording. After the demonstration is complete, our system immediately cleans the raw recording. First, it associates the changes in interface state with the corresponding changes in application state. Usually, there is a one-to-one correspondence between interface- and application-level state changes. For example, moving the saturation slider in lines 3, 6 and 10 in the interface corresponds to setting the saturation parameter of the hue-saturation operation. However, some changes in application state, such as setting an operation via the menubar, require multiple interface actions. In lines 1–2 and 8–9, the user performs two interface actions to navigate to the hue-saturation menu item. The first action, clicking on the color item in the menubar, does not change the application state. However, our tutorial must depict such navigational interface actions so that the user can see how to set the hue-saturation operation. Thus, our system associates both of these navigational interface actions with setting the hue-saturation operation.

Our system handles undo operations by eliminating the previous commit operation. Finally it removes any changes in application state that do not affect a committed operation. Users often make mistakes, such as setting the wrong operation when demonstrating a manipulation. For example, in line 12 the user accidentally sets the pencil tool operation by clicking on the pencil tool in the toolbar. However, he never commits a pencil stroke. Instead, he selects the paintbrush

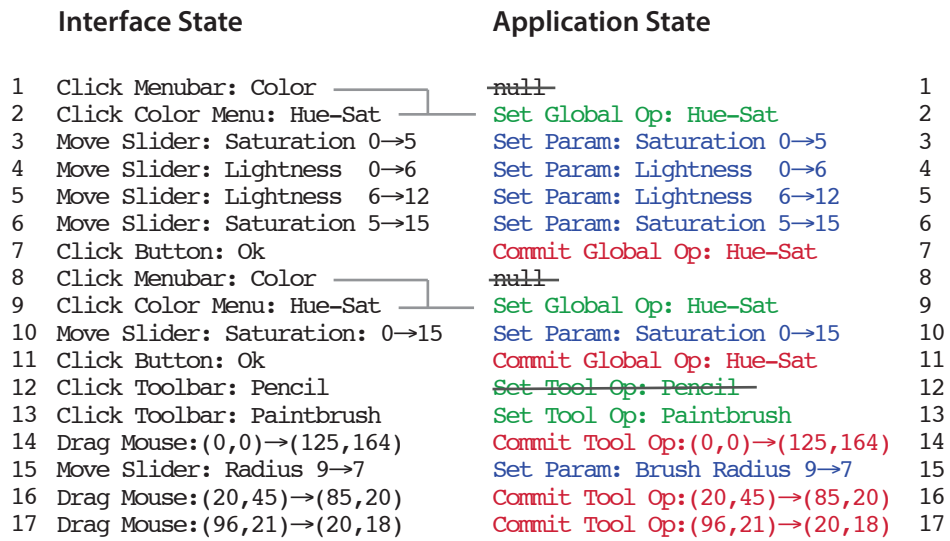


Figure 3.3: Our demonstration recorder captures and associates interface-state information (column 1) with application-state information (column 2). Gray lines indicate wherever more than one interface-level state change is associated with a single application-level state change. The application-level state changes are color-coded: green for changes that set operations, blue for parameter setting changes and red for commit operations.

in line 13 and commits a paint stroke in line 14. As a result, our system eliminates the pencil operation in line 12.

3.4 Image Labeler

We leverage existing computer vision-based techniques to automatically label semantically important regions in images. We currently apply two recognition techniques to every input image: one for face detection and one for sky detection. Labeling facial areas (lips, eyes, etc.) and the sky allows us to generate descriptive text descriptions for many global, face and landscape manipulations. While our results depend on the quality of the labeling, our approach is designed to work in general with any labeled images and could work with other detectors or even hand-labeled images.

Faces are important features in many photographs, and portraits are commonly edited in photo manipulation tools. To detect facial features we use the Bayesian Tangent Shape Model of Zhou et al. [105]. This algorithm detects the 2D positions of 83 landmark points that define the eyes, eyebrows, nose, lips, and face contour (Figure 3.4a,b). Based on these landmark points, we compute the positions of the iris, pupil, and teeth. To detect the iris, we search within the eye for a circular region with minimum average luminance. We approximate the pupil as a circle half the diameter of the iris and co-located at the center of the iris. To find the teeth, we look for pixels within

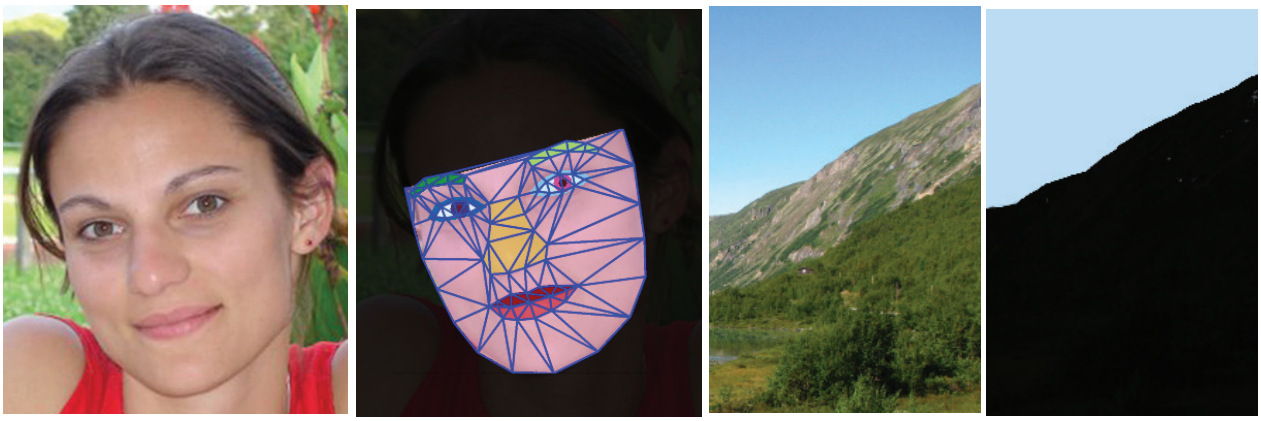


Figure 3.4: For a given input image (a) we detect the following facial features: eyes, iris, pupil, eyebrows, nose, lips, teeth, and face contour (b). For outdoor scenes (c), we automatically detect the sky and the ground (d).

the boundary of the lips that are nearly white. For outdoor scenes, we use the implementation of Hoiem et al. [48] to automatically detect the sky and the ground (Figure 3.4c,d).

Our system uses the image labels to produce more precise, human-understandable text descriptions of the spatial regions in which users must perform the operations in each step (Figure 3.7 in Section 3.5.2). For example, in Figure 3.1(left) step 2 explains that the user should select "the right iris, the right pupil, the left iris and the left pupil" in order to recolor the eyes. Similarly, suppose the user is demonstrating how to add blush to a face and paints a red tinted brushstroke on a cheek to the right of the nose. Since our face recognizer cannot identify cheeks, if the stroke is close enough to the nose the text description will explain that user should "paint a stroke to the right of the nose". If no labeled feature is near the selected region or stroke our system produces a more generic text description such as "paint a stroke" and relies on the annotated screenshot to provide enough context for the user to perform the operation properly.

We compute the spatial context for a local operation by first generating a mask corresponding to the selection or the brush strokes. We then intersect this mask with the labeled segmentation generated by both the face and outdoor scene recognizers. For selection operations, if the selection mask covers at least 70% of the pixels of a labeled area, we set the location of the selection to the name of the labeled region. For brush strokes, if the stroke mask falls entirely within a labeled area, we set the location of the stroke to the name of the labeled region. If there is little or no overlap between either mask and a labeled area but they are close to one another, we use the relative locations of the bounding boxes of the labeled area and the mask to specify if the stroke or selection is above, below, to the side, or surrounding the labeled area. In some cases the recognizers may give the same pixel multiple labels (e.g. face contour and sky). In such cases our system uses all of the labels to describe the location of the operation and relies on the screenshot to help the user further disambiguate where the operation should be performed. However, we have

never encountered such multi-labeled pixels using our current combination of face and outdoor scene recognizers.

3.5 Tutorial Generator

Our goal in generating photo manipulation tutorials is to produce a set of instructions that people can easily understand, and that facilitate the cognitive stage of the learning process. We base the visual design of our tutorials on previous work in instruction design [15, 59, 4] and cognitive science [81, 45] that suggest guidelines for creating effective instructional material. These guidelines are based on human-subject experiments studying the way people understand diagrammatic depictions of how to assemble furniture, open packaging, or prepare a recipe. We summarize the most relevant design guidelines.

Step-by-step. People mentally break processes into a sequence of steps, and well-designed instructions present each major step in a separate panel.

Succinct. People prefer succinct instructions. Unnecessary steps should be eliminated and repetitive sequences of steps should be condensed and shown only once.

Annotations. People better understand the operations required in each step when arrows, highlights, callouts and other annotations are used to visually show how the step should be performed.

Text and images. People prefer instructions that combine text descriptions of each step with images visually depicting the steps. Images alone are less preferable and harder to use. Text descriptions alone are least preferable and most difficult to use.

Grid-based layout. People prefer grid-based tutorial layouts that clearly indicate the sequence of steps and place text and annotated images describing each step near one another.

To create a tutorial from a demonstration, the user specifies the title of the tutorial. Our system then processes the demonstration in three stages that are based on the design guidelines. First, it groups together consecutive changes to application state to build a succinct sequence of steps, each involving one major image processing operation. Next, it annotates the screenshots that correspond to each step with arrows and highlights that indicate the interface-level actions required for the user to perform each step. It also composes text descriptions describing each step. Finally, our system lays out the annotated screenshots and the corresponding text for each step using a simple grid-based design to form a complete tutorial that is designed to be printed or accessed via the web.

3.5.1 Grouping Operations Into Steps

When demonstrating a photo manipulation technique our experience is that users invariably tweak operation parameters repeatedly until they achieve the desired result. In order to produce clear and succinct step-by-step tutorials, our grouping stage processes the raw recordings to merge repetitive operations. Such grouping is similar to previous work on aggregating input events into higher level actions [63, 61, 78]. In our system, grouping requires two passes through our recordings (columns 2 and 3 in Figure 3.5).

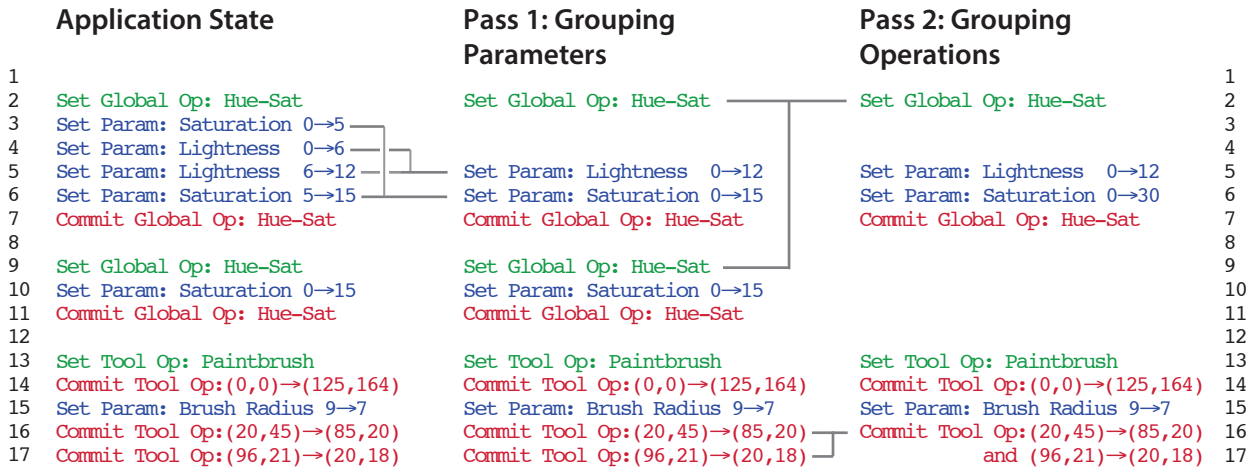


Figure 3.5: The first column shows the application-state information after clean-up (Section 3.3). Our tutorial generator groups parameter changes within an operation (column 2) and then groups multiple commit operations for the same operation (column 3).

Pass 1: Parameter grouping. Within an operation, parameter changes are order independent until the operation is committed. We therefore group together all changes to the same parameter that occur between setting the operation and each commit of the operation. The second column of Figure 3.5 shows how this pass merges changes to the saturation and lightness parameters in the first execution of the hue-saturation operation (lines 3–6).

Pass 2: Operation grouping. Users sometimes commit the same operation multiple times consecutively to test the effects of various parameters. In the example of Figure 3.5, the user committed the global hue-saturation operation twice consecutively (lines 7,11). The operation grouping pass merges such consecutive sequences of the same operation. For local operations we only group together consecutive commits if all operation parameters other than the pointer position remain fixed (lines 16–17). We cannot group the paintbrush commit operations in lines 16–17 with the first paintbrush commit in line 14 because of the change in brush radius between them.

When grouping parameter changes we must distinguish between *absolute* and *relative* parameter changes. Absolute changes replace the old value of a parameter with the new value. For example, when using the color picker tool to set the foreground color, we only save the last color that was picked and discard intermediate values of the color parameter. Relative changes add a new value to the current value of the parameter. When grouping together relative parameters, we accumulate the values of all intermediate changes. For example, consider the image rotation operation. If we execute a 30° rotation followed by a 60° rotation, we must accumulate the rotation angle parameter to 90°. In GIMP, many parameters are absolute when they are changed within an operation dialog before a commit but become relative when changed across commits. In our example, saturation is an absolute parameter within an operation (lines 3, 6) that becomes a relative parameter across the commit of the hue-saturation operation (lines 6, 10).

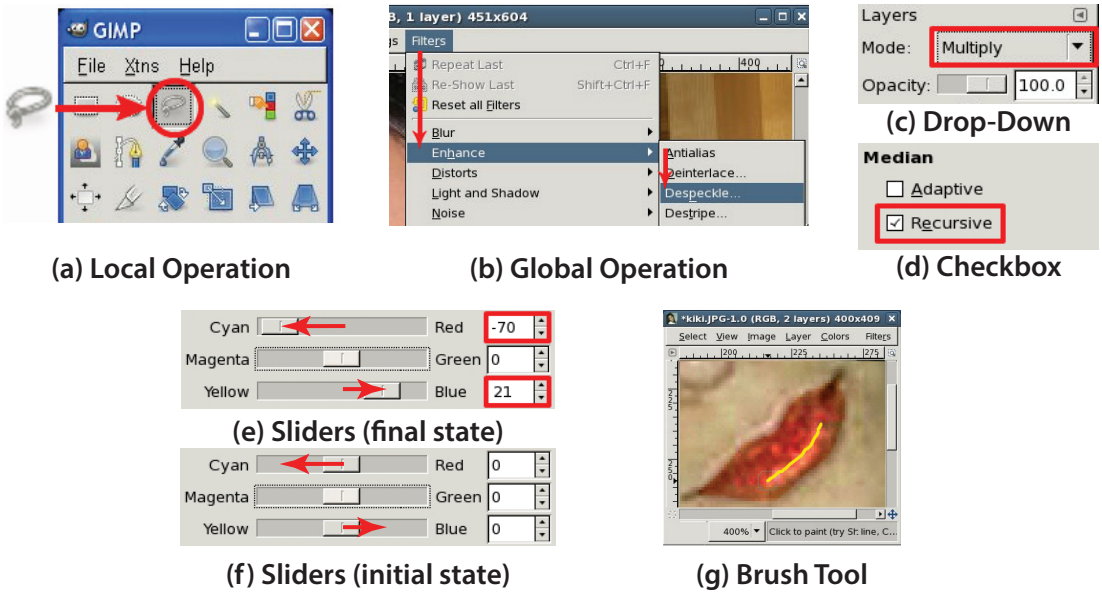


Figure 3.6: We annotate screenshots to highlight the tasks the user must perform. The style of the annotation depends on whether a user sets an operation (a,b), sets a parameter (c-f) or commits an operation (g).

After operation grouping, each line of the final representation (3rd column of Figure 3.5) can be rendered as a step in the output tutorial. Our system can also more aggressively group together all parameter changes for the same operation, which would merge the changes to the saturation and lightness parameters into a single step rather than treating them as two separate steps. We leave this aggressive grouping as an option to the user.

3.5.2 Screenshot Annotations and Text Generation

Our tutorials are comprised of annotated screenshots and descriptive text illustrating the tasks the user must perform. We render the annotations based on the position and size information of the interface elements we record with each interface state. The style of the annotation depends on whether a user sets an operation, sets a parameter or commits an operation.

Set operations. Navigating through the menu hierarchy or the toolbar can be challenging, especially for novice users. Therefore we annotate operation setting steps with arrows, circular highlights and larger icons that indicate how users should traverse the interface (Figure 3.6a,b). After adding annotations we crop the screenshots to further emphasize the location of the operation. For example, the GIMP toolbar window includes a set of local tool operations at the top and an area for setting the operations' parameters below. When depicting how to navigate to a local operation we crop the toolbar image to only show the relevant operation area at the top of the window.

Set parameters. We annotate parameter updates by highlighting the change in state of the corresponding interface widget so that users can better understand how to set the parameter (Fig-

Set Operations

Select <tool name|menu path> from the <toolbar|image menubar> to <description>.

Set Parameters

Slider: Set the <param name> slider from <param value before> to <param value after> in the <dialog name> dialog.

Radiobuttons, Drop-down lists: Select <param value> in the <param name> field in the <dialog name> dialog.

Checkboxes: [Select | Unselect] <param value> in the <dialog name> dialog.

Commit Operations

Paint: Paint using the <op name> [on <location>].

Selection: Use the <op name> tool to select [region |<location>].

Figure 3.7: Templates used to generate text instructions. Green variables are based on interface- and application-level information. Blue location variables are optional and refer to labeled image areas that are produced by our image labeler.

ure 3.6c,d,e). After grouping relative parameters, we usually do not have a screenshot that shows the accumulated state of the parameters. In this case, we show a screenshot of the initial state and use arrows to indicate the final state (Figure 3.6f).

Commit operations. When a user commits a global operation, there is usually a noticeable change within the selected region of the image. As a result, we simply present a screenshot of the window containing the image without any annotation. However, we commit local operations whenever the tool is dragged across the image. For *brush strokes* (paintbrush, pencil, healing-brush, clone-brush, etc.) the path of mouse-drag affects the local operation, so we annotate the exact path (Figure 3.6g). For *selections* (free-select, ellipse-select, by-color-select, etc.) the outline of the selection region is more important than the location of the mouse-drag so we do not annotate the screenshot so that users can see the selection outline.

We use a template-based approach to generate the text descriptions for each step (Figure 3.7). Our approach is designed for extensibility so that additional templates can be easily added to the system as necessary to support other widgets. As explained in Section 3.4, we use automated image labeling techniques and a set of heuristics to identify the location of selection regions and brush stroke operations.

Figure 3.8: Night from day tutorial (first of 3 pages). This tutorial shows users how to make a daytime photograph look as if it was taken at night. Operations are organized into steps that are stacked into columns. Each step includes text descriptions and annotated screenshots.

3.5.3 Layout

The final stage is to lay out the tutorials so that users can easily follow the sequence of steps. We surveyed a variety of books containing photo-manipulation tutorials and chose to adopt the grid-based layout of Huggins [49]. For each step in the tutorial we place the text description on the left and the annotated screenshots next to it on the right. If there is an immediate visual consequence to the action/operation, we add a second screenshot depicting the consequence. Some operations, such as selections with the path-tool, require two interface actions: drawing the path and turning the path into a selection. For such operations we show a screenshot of both actions instead of the action and its consequence.

Given a target page size, we stack the steps vertically in columns starting at the top of the page and add as many columns as possible to fill the horizontal space on the page. To help users immediately see the effect of the manipulations shown in the tutorial, we present the image before and after the manipulation in the first column of the tutorial (Figure 3.1 and 3.8). While this simple layout algorithm allows some adaptation to different page sizes, we do not provide any controls for resizing annotated screenshots or the text areas. As a result, some pages may contain more

white space than necessary. We leave it to future work to explore the use of more sophisticated grid-based layout algorithms [33, 50].

3.6 Results and Discussion

We have used our system to generate eight visual tutorials[1] including six portrait retouching techniques – changing eye color (Figure 3.1), whitening eyes, whitening teeth, removing eye bags, blemish removal, and adding lip gloss – and two methods for editing outdoor scenes – enhancing sunsets, converting day to night (Figure 3.8). We have also asked first-time users in our lab to produce these eight tutorials using our system. Each manipulation technique required 5-10 minutes to demonstrate. Both the face and outdoor scene recognizers took under a second on each input image and our tutorial generator required between 1 and 3 seconds to produce the tutorials, depending on the number of steps.

As shown in Figures 3.1 and 3.8, our tutorials succinctly describe the major operations required to perform the manipulation. Our system omits unnecessary operations and condenses repetitive ones. For example, Step 5 of Figure 3.8 summarizes several slider updates in a single panel. Our tutorials also include annotations that clarify how to perform individual steps, such as finding specific operations (Figure 3.8, Step 1) and navigating menus (Figure 3.8, Step 3).

Our image labeler identified all of the facial features and sky regions in the demonstration images of our eight tutorials. As a result the text descriptions often contain good location information describing where a selection or brush stroke operation should be performed. However, image recognizers can produce two kinds of errors; false-negatives and false-positives. Our tutorials are less sensitive to false-negatives. If a feature is not found in the image our system simply produces a more generic text description for the operation such as "select a region" rather than "select the lip". The system relies on the annotated screenshot to provide enough information for users to properly perform the selection. In contrast if recognition falsely finds a feature, the text description will mislead users by asking them to "select the lip" even though there is no lip at that location in the image. While it may not be possible to eliminate false-positives, many recognition techniques include a parameter that allows users to tradeoff more false negatives for fewer false positives. We leave it to future work to explore good settings for managing this tradeoff.

A limitation of our tutorials is that they cannot explain why users must perform each operation. In some cases the explanations are especially important for users to understand how to apply the technique to their own images. For example, one of the steps in the *creating black and white from color* tutorial in the Photoshop Retouching Cookbook [49] involves multiplying the image by its red color channel. The reason for this step is that red is the highest contrast color channel in the example image and the multiplication increases the contrast for the conversion. While our system would include a step showing that the red channel should be multiplied, it would not be able to explain that the user should choose the highest contrast color channel to multiply. Without the explanation the user would likely pick the red channel instead of the highest contrast color channel. In most cases however, such explanations are not critical for users to properly follow the

[1] Additional tutorials generated with our system can be found at http://vis.berkeley.edu/papers/tutgen/

tutorial and adapt it to their own images. Users can often figure out how to adapt the tutorial to their own image from knowing the goal of the tutorial which is often stated in the title and looking at the screenshots.

We have manually examined all 116 tutorials in Huggins' [49] book and determined that our system could fully reproduce all of the relevant information including images and text for 85 of them. Of the remaining 31 tutorials, 23 required labeling of objects (e.g. hair, foreground objects, light sources) or image features (e.g. highlights, shadows) that our current recognizers could not provide. Our system would produce a more generic description of the location of an operation as in the false-negative case for labeling errors. We found that 8 of the book tutorials including *creating black and white from color* provided critical explanations for one or more steps that our system would not be able to infer.

3.7 User Study: Tutorial Effectiveness

We conducted a user study to evaluate the effectiveness of our automatically generated tutorials compared to hand-designed tutorials from a book and video-based tutorials. We hypothesized that users would be faster and make fewer errors using our tutorials than the other types, for the following reasons. Book tutorials often leave out a number of low-level steps required to successfully complete a manipulation as they usually assume users are experienced with the photo manipulation software. Novice users often have to spend time navigating menus and trying several options to complete a step. Video-based tutorials on the other hand force users to work at the pace of the video and users typically have to rewind several times to replay steps they missed or did not understand.

3.7.1 Study Design

Our experiment investigates the effects of two independent variables on user performance: *tutorial type* (book, video, ours) and *tutorial content* (eye recoloring, adding lip gloss and converting day to night). The tutorial content increased in complexity from eye recoloring which contains 4 steps to converting day to night which contains 17 steps. We initially found these tutorials in a book containing Photoshop tutorials [49]. To produce GIMP-specific versions of these book tutorials we manually replaced the Photoshop-specific screenshots and text instructions with GIMP equivalents. To produce video tutorials we screencaptured GIMP while demonstrating the manipulation and then manually added an audio voiceover explaining each step. Finally, we used our tutorial generation system to generate our tutorials. Thus, our study included a total of nine conditions (3 types $\times$ 3 contents).

Using Craigslist we recruited 18 subjects (10 women, 8 men) ranging in age from 20 to over 55 years old. Ten subjects had some experience with Photoshop and eight subjects had never used a photo-editing software before. None of the subjects had used GIMP before. We used a within subjects study design. We first gave the subjects a 10 minute introduction to GIMP showing them how to use the software. Each subjects then used our instrumented version of GIMP to

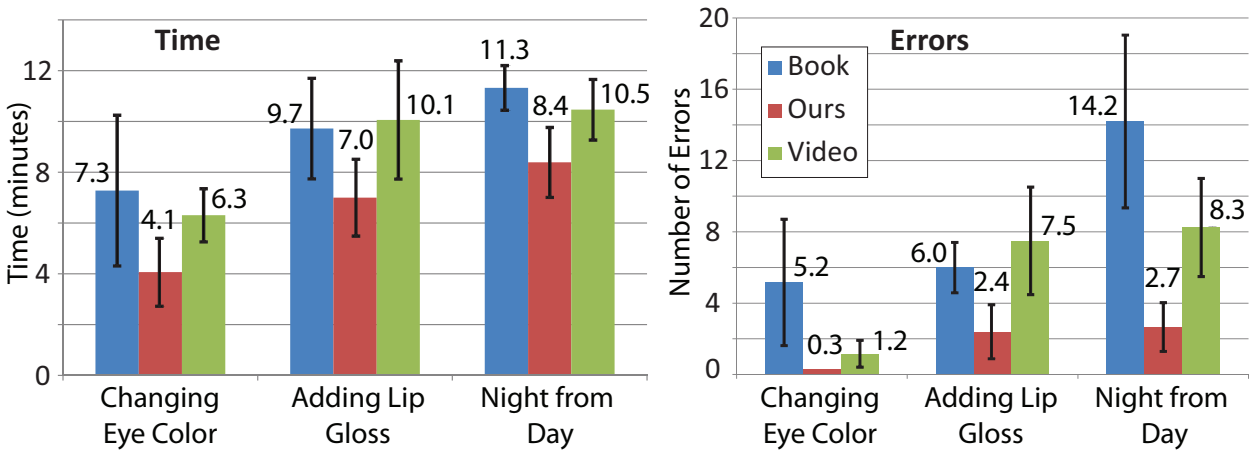


Figure 3.9: Average time (left) and average number of errors (right) required for each tutorial content in our user study. All differences between our tutorials and the book and videos are significant.

perform three different photo manipulations. To mitigate learning effects we fixed the ordering of the content across all subjects giving them the simplest eye recoloring manipulation first, then the more complex lip gloss manipulation, and finally the most complex day to night manipulation. Thus, each subject experienced three combinations of tutorial type $\times$ tutorial content, where the ordering of the tutorial type was fully counterbalanced across subjects and the ordering of tutorial content was always the same. Each of the nine conditions was performed by six subjects.

We recorded the time it took the subjects to complete each manipulation. To determine the number of errors made by each subject we manually examined the traces produced by our demonstration recorder. We counted an error each time the subjects selected the wrong tool, wrong menu item, manipulated the wrong parameter or missed a step. We also counted an error each time a subject selected an incorrect region in the image. For example, if the tutorial asked the subject to select the lips and the subject selected an area that did not include any part of the lips we marked it as an error. We did not count alternative ways to perform an operation as errors. For example, some subjects preferred to use shortcut keys to set an operation. We also did not penalize subjects that adjusted the parameters to better match the input image instead of using exactly the parameter settings indicated in the tutorial.

3.7.2 Study Results

The average performance time and errors for each tutorial content are shown in Figure 3.9. Overall we find that our tutorials are 20-44% faster and reduce errors by 60-95% compared to the book and video. We allowed subjects up to 20 minutes to complete each manipulation. While most subjects finished in much less time, 3 subjects failed to complete the converting day to night manipulation and 1 subject failed to complete the adding lip gloss manipulation using either the book or the video. No subject failed to complete a manipulation using our tutorials. We have excluded the data

from the failed manipulations from our analysis.

Considering each content separately, an ANOVA across tutorial type reveals a significant main effect for both time and errors ($p < 0.02$ in all six cases). Subsequent pairwise T-tests show significant differences between our tutorials and both the book and video tutorials with respect to time and errors across the three content types ($p < 0.04$ in all cases). However, the only significant difference for the book-video pair is for number of errors in the eye recoloring content ($p < 0.043$). All other differences in time and errors between the book and video tutorials were not significant.

These results indicate that our automatically generated tutorials are more effective than hand-designed book and video tutorials at helping users complete the depicted photo manipulation tasks. However, our study did not examine how well users could generalize the operations they used to new manipulation tasks. Hand-designed book or video tutorials may be more effective for such generalization because they usually explain why the user needs to perform each step in the tutorial. We leave it to future work to study how well users can generalize from automatically generated tutorials.

3.8 Next Steps

In this chapter, we have presented a demonstration-based system for automatically generating succinct step-by-step visual tutorials that facilitate the cognitive stage of the learning process. Our user study shows that our tutorials are effective for learning the steps of a procedure; users are faster and make fewer errors when using our tutorials than when using screencapture videos or hand-designed tutorials. As future work, we plan to add additional editing and feedback interfaces to our system, and study if learners can use our automatically generated tutorials for transfer.

Interfaces for additional explanations. One limitation of our tutorials is that they do not provide any explanations for the steps. As we saw in Section 3.6, these explanations can be critical to help users properly follow the tutorial and adapt it to their own images. However, our tutorials are generated in a fraction of the time required to manually capture, annotate and layout screenshots, or record voiceovers. Therefore, tutorial authors could use our automatically generated tutorials as an initial draft, and supplement them with further explanations describing the reasoning behind each step. We plan to add such interactive authoring interfaces to our system, so that tutorial authors can further edit the resulting text descriptions.

Using automatically generated tutorials for transfer. We plan to study how people can use our tutorials for transfer. For example, we would like to give people an eye whitening tutorial and study if they can then complete a similar task like teeth whitening. Doing this type of transfer with our automatically generated tutorials may be challenging, as our text descriptions do not provide any explanations for why a user needs to perform each step. We would like to study for what type of manipulations this transfer is possible, despite the lack of explanations in the text descriptions.

Feedback mechanism. We plan to implement a feedback mechanism, so that we can let users know if they deviate from the instructions. Such a feedback mechanism would require us to track the users as they are performing the task, and use the operation history to then decide on the fly whether each step was executed correctly or incorrectly. It may also require modeling the kind of errors that users typically make when performing the task, so that we can recognize each error when it occurs. Our feedback mechanism could then help users get back on track.

Chapter 4

Comparing Tutorials

In the associative stage, learners compare different procedures for executing the same task, and identify sequences of commands that are useful in different contexts (e.g. the command sequence "Quick Mask, Brush Tool, Invert, Hue-Saturation" is useful for recoloring hair, eyes, cars, garments, etc.). Sites such as `tutorialized.com` collect tens of thousands of photo manipulation tutorials. However, exploring and comparing tutorials within these collections is very difficult, because these sites do not exploit the underlying command-level structure (i.e. sequence of software commands) of the tutorials. To explore the collection users must either read through long lists of tutorial titles or perform keyword searches on the natural language tutorial text. Even identifying the sequence of commands in a single online tutorial can be challenging. Online tutorial use narrative text and often refer to commands using colloquial terms rather than the actual Photoshop command name (Figure 4.1).

We present a new browser interface that allows users to navigate, explore, and compare image manipulation tutorials based on their command-level structure. Our system includes a Command Extractor that identifies command sequences in natural language software tutorials. To build this tool we extend the machine learning approach of Fourney et al. [36] to automatically identify direct and colloquial references to commands (e.g. the tutorial says "make a mask" instead of referencing the command name "Layer Mask", Figure 4.1). We apply this command extractor to a collection of 2500 Photoshop tutorials we obtained from the Web and achieve an average precision of 90% and recall of 99%.

Our browser provides an interface for exploring tutorial collections based on three different views of the command-level structure (Figure 4.2). The *(1) Faceted Browser View* allows users to organize and filter the collection. Filtering facets include high-level tutorial categories (e.g. Photo Effects, 2D Drawing, Web Layout), the length of command subsequences (N-Gram length) and command names of interest. The Category facet allows users to only explore tutorials within their domain of interest. Using the N-Gram facet, users can quickly identify command sequences that are useful in different contexts. For instance, a user can find the most common multi-command strategies in the Photo Effects category by selecting N-Gram lengths 1 through 4 and the Photo Effects facet (Figure 4.2a). Finally, the Command Name facet allows users to familiarize themselves with how the same Photoshop command is used for different tasks. When a user selects a tutorial

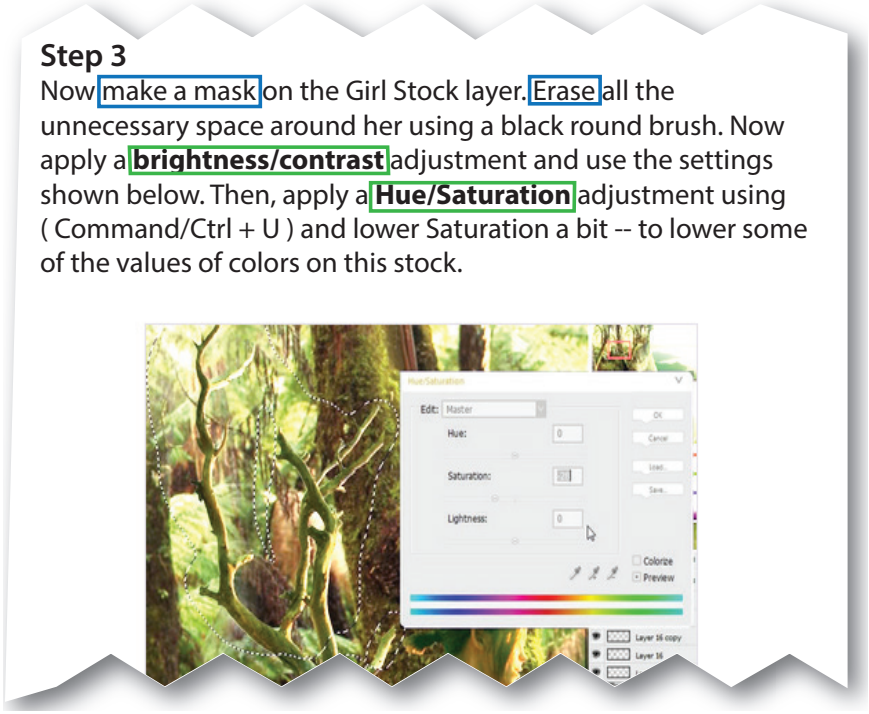


Figure 4.1: A step from an example online tutorial. Tutorials use direct references to commands (i.e. exact string matches to the command name, shown in green) and colloquial/indirect references to commands (e.g. tutorial says "make a mask" rather than writing the command name "Layer Mask", shown in blue)

in the faceted browser the tutorial webpage appears in the *(2) Tutorial View* along with a *table of commands* that serves as both a summary of the tutorial and an index into the webpage. The *(3) Alignment View* graphically depicts the similarities and differences in the command structure of a subset of tutorials. It can be used to compare multiple alternative procedures that achieve the same goal. Together these views allow users to explore tutorial collections and better understand the underlying structure of photo manipulation procedures.

In an informal evaluation of our browser interface we asked nine first-time participants to complete a series of tutorial browsing and comparison tasks using our interface as well as standard keyword search. Qualitative user feedback indicates that our browser is easy for newcomers to learn and understand and that users find faceted command-level browsing to be useful for exploring large tutorial collections. Moreover, participants successfully completed a variety of novel browsing and comparison tasks. We show that these tasks are difficult to accomplish with keyword search. Participants strongly preferred to explore tutorial collections with our browser over keyword search. These results suggest that our interface is an effective tool for browsing and comparing large collections of image manipulation tutorials. It facilitates the associative stage of learning by helping

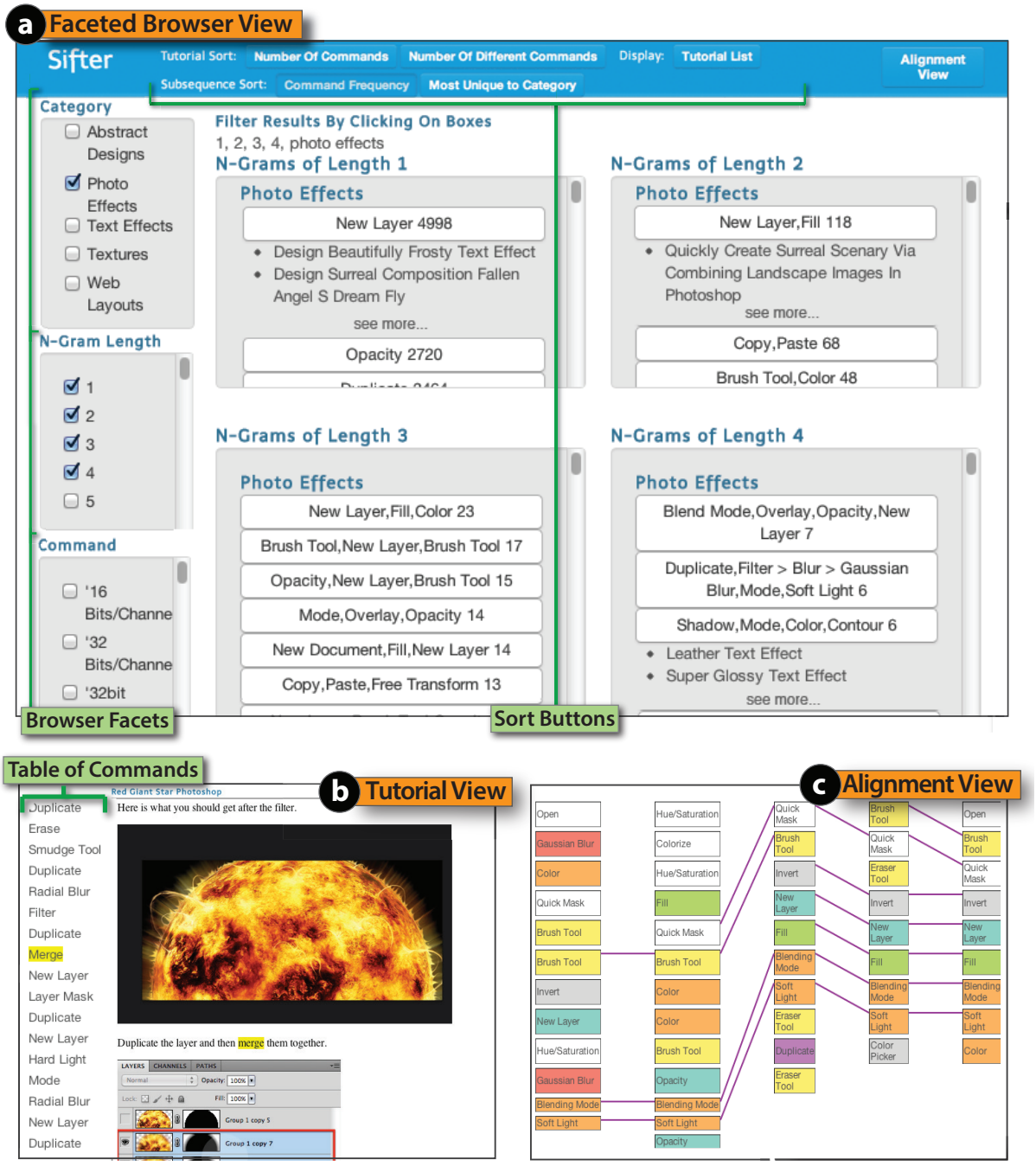


Figure 4.2: Our browser interface allows users to explore our collection of 2500 Photoshop tutorials based on their command-level structure. In the *(a) Faceted Browser View* users can organize tutorials using the Category, the N-Gram length (i.e. command sequences of length N) and the Command Name facets. Clicking on a tutorial title loads the tutorial webpage and its table of commands into the *(b) Tutorial View*. Users can also align the command sequences of multiple tutorials to see the similarities and differences between them in the *(c) Alignment View*.

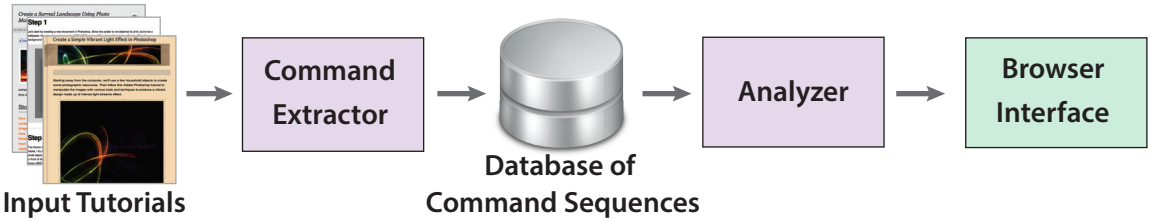


Figure 4.3: System Overview.

users better understand the underlying structure of photo manipulation programs.

4.1 System Overview

We collected a corpus of tutorials by scraping 2500 Photoshop tutorials from eight websites. We then process every tutorial from this collection through the three consecutive components of our system (Figure 4.3). The ***command extractor*** takes a set of text-based tutorials as input and outputs the command-level structure of each one into a database of command sequences. The ***analyzer*** provides algorithmic tools to reveal the underlying structure within the database of command sequences. More specifically, it compares command sequences in the database and identifies frequently occurring subsequences. Finally, the ***browser interface*** builds on the analyzer and gives users access to the algorithmic analysis tools. It provides three different views that allow users to navigate, explore and compare the photo manipulation procedures stored in the database. The Faceted Browser View allows users to filter the collection and identify commonly used subsequences of commands within the tutorial collection. The Tutorial View highlights the commands used in each tutorial and provides a table of commands that indexes and summarizes the tutorial. The Alignment View aligns command sequences from multiple tutorials and highlights their similarities and differences.

In Section 4.2, we describe the features of the browser interface and its three views. We then describe the two technical components, the extractor and analyzer, in Section 4.3.

4.2 Browser Interface

We designed a browser interface that allows users to navigate, explore, and compare the tutorials in our collection using three different views: the Faceted Browser View, the Tutorial View and the Alignment View.

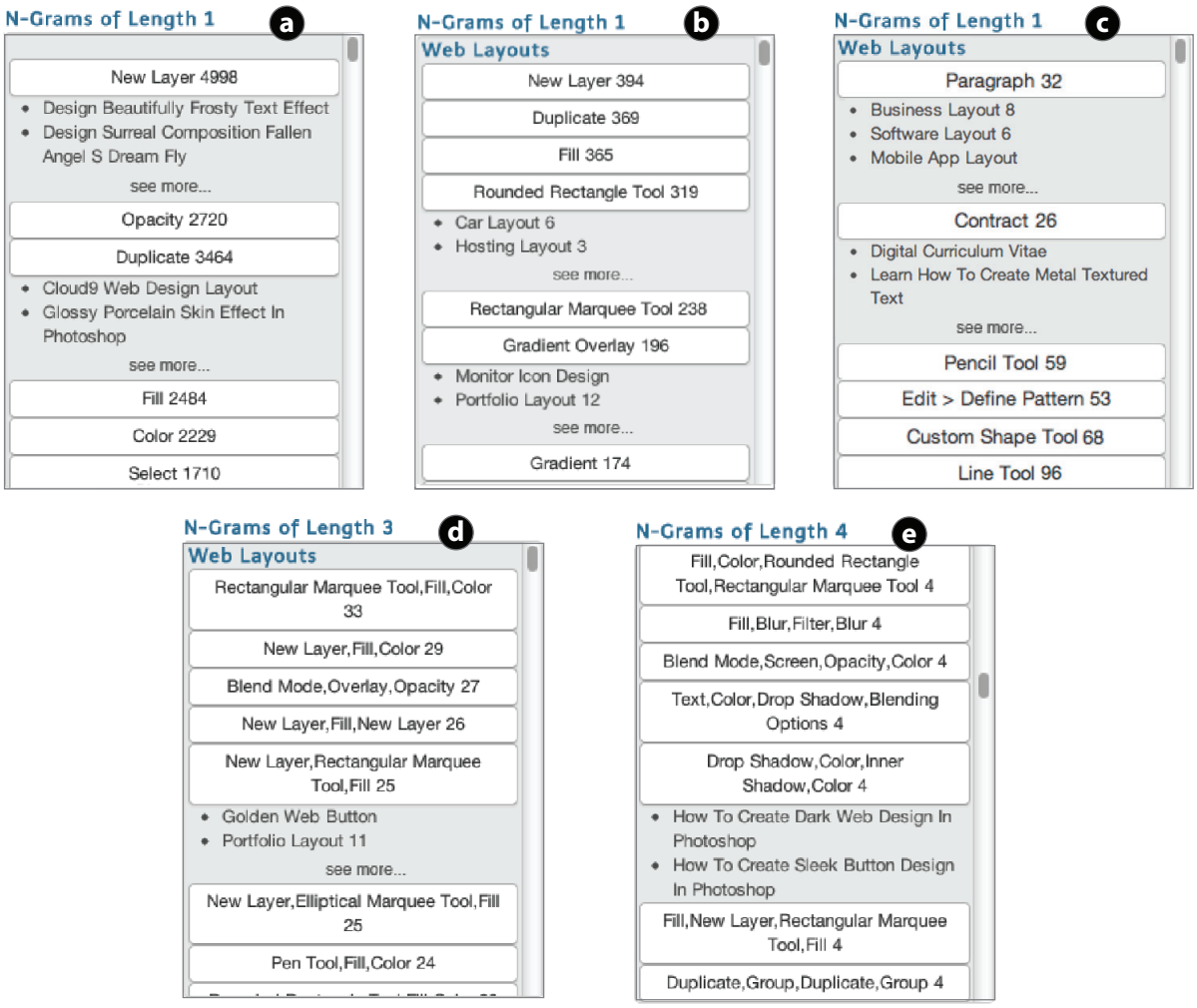


Figure 4.4: Frequently occurring subsequences in our database of command sequences: (a) 1-grams across the entire collection of tutorials, (b) 1-grams within Web Layouts category, (c) 1-grams that are unique to the Web Layouts Category, (d) 3-grams and (e) 4-grams for Web Layouts.

4.2.1 Faceted Browser View

The Faceted Browser View (Figure 4.2a) is designed to help learners explore tutorial collections. To achieve this goal, it provides three facets that allow users to organize and filter tutorials. The Category facet filters the collection by a high-level tutorial category (e.g. Photo Effects, Illustrations, Web Layouts etc.). These categories describe the higher-level purpose of the tutorial and are given by the original tutorial website. The N-Gram facet filters the collection by the length of command subsequences. It sorts the N-Grams by their occurrence frequency, so that the most common command subsequences appear first. Such frequent subsequences often represent higher-

level strategies that are used in many different procedures. Optionally, users can also sort command subsequences by uniqueness to a selected tutorial category. This sorting criterion allows users to identify frequently occurring subsequences of commands that are specific to a tutorial category. Finally, the Command Name facet allows users to filter the collection to only include tutorials that contain specific commands as chosen from a list. The order in which the user selects facets determines the hierarchical organization of the tutorials that appear in the faceted browser view.

Figure 4.4 shows examples of different filters that a learner might apply to the tutorial collection. When setting the N-Gram facet to select single commands (N-Gram length=1), the browser shows the most frequently used commands across all photo manipulation procedures in the database (Figure 4.4a). The top three commands are "New Layer", "Opacity" and "Duplicate Layer". The "New Layer" command appears almost fifty thousand times and is thereby the most versatile and perhaps useful Photoshop command.

Photoshop is commonly used to mock up web layouts. When filtering all the N-Grams of length 1 within the Web Layouts tutorial category, the list now includes the "Rounded Rectangle Tool" and "Gradient Overlay" (Figure 4.4b). In the context of Web Layouts, these commands are often used to draw buttons and give widgets the appearance of 3D depth. We can also specifically list all commands that are frequently used in Web Layouts, but infrequently in the rest of the corpus by selecting the "Most unique to category" sort option. Figure 4.4c shows this list. The "Paragraph" command appears first and is commonly used in web layout tutorials to format text while the "Contract" command is often used to shrink selections and create borders on text boxes (Figure 4.4c). These commands are far less common in the other Photoshop tutorial categories.

Figure 4.4d-e filters for Web Layouts N-Grams of lengths 3 and 4. Many of the frequent three-command sequences involve a selection command followed by a fill command such as "Rectangular Marquee Tool, Fill, Color", "New Layer, Rectangular Marquee Tool, Fill" or "New Layer, Elliptical Marquee Tool, Fill" (Figure 4.4d). When examining some of the tutorials for each of these three-command sequences, we find that they all describe ways to create buttons of various shapes. In the list of four-command sequences, there are several sequences that include styling commands such as "Drop Shadow", "Blending Mode", and "Blur" (Figure 4.4e). These command sequences all apply specific styles to buttons. For example the 4-gram "Drop Shadow, Color, Inner Shadow, Color" adds a drop shadow and inner shadow to a button to give it a 3D appearance.

Therefore, by listing frequently used commands within and across tutorial categories, the faceted browser view allows learners to identify patterns in the procedures and explore how the same higher-level strategies can be used in different contexts.

4.2.2 Tutorial View

The Tutorial View is designed to provide learners with a command-level overview of the photo manipulation procedure described in the tutorial, and give them quick access to any particular step of the procedure. It shows the original webpage for any selected tutorial as well as a table of commands that lists the commands in the tutorial (Figure 4.2b). The table and webpage are linked so that clicking on a command in the table scrolls the tutorial webpage to the corresponding location, while clicking a command in the webpage highlights it in the table.

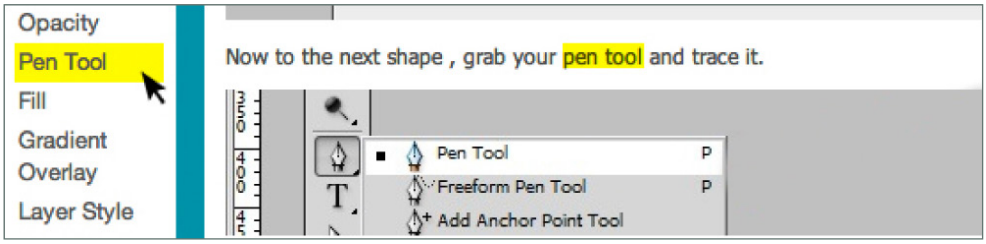


Figure 4.5: Clicking on the "Pen Tool" command in the table of commands scrolls the tutorial to this command and highlights it in the tutorial.

Figure 4.5 shows an example of clicking on the Pen Tool command in the table of commands. It scrolls the tutorial webpage to show how the command is used in context. In this case, the tutorial describes how to use the Pen Tool to draw paths. Thus, the table of commands can serve as a reference as users work through the tutorial and allows them to quickly access more information about any commands as necessary.

Users can also select a subsequence of commands in the table and perform a selection search to retrieve all tutorials containing the chosen subsequence. For example, in the tutorial "How to Design a Web Layout in Photoshop" shown in Figure 4.6a, the tutorial uses the "Rounded Rectangle Tool, Fill, Color" commands to create a box for holding pictures and text within the page layout. When selecting this sequence of commands and clicking on the Selection Search button, the browser returns other tutorials that also contain the sequence (Figure 4.6b). Each of these tutorials has examples of additional styles for content boxes (Figure 4.6c). Thus, the selection search allows learners to better understand how the parameters of command sequences affect the resulting style of the drawn object.

4.2.3 Alignment View

The Alignment View is designed to help learners understand the similarities and differences in the command-level structure of a subset of tutorials. Users can select a set of tutorials in the faceted browser and generate an Alignment View in which each column represents the sequence of commands for a single tutorial and lines connect matching commands (Figure 4.2c). Sibling commands in the Photoshop menu hierarchy are given the same color (e.g. Colorize and Hue/Saturation are both colored white because they both appear within the set of Image>Adjustment commands).

To align the tutorials, we first establish an ordering between them. The user may select the first tutorial in this ordering which we call the *base tutorial*. If the user does not select a base then we automatically set the most representative tutorial as the base, which we compute as the centroid of the tutorial set based on the edit distance described in Section 4.3.2. Once the base tutorial is set, a user can choose between the *one-to-all layout* that orders the tutorials (left-to-right) by their similarity to the base tutorial, or the *pairwise layout* that builds the ordering by incrementally choosing the next tutorial as the one that is most similar to the previous tutorial

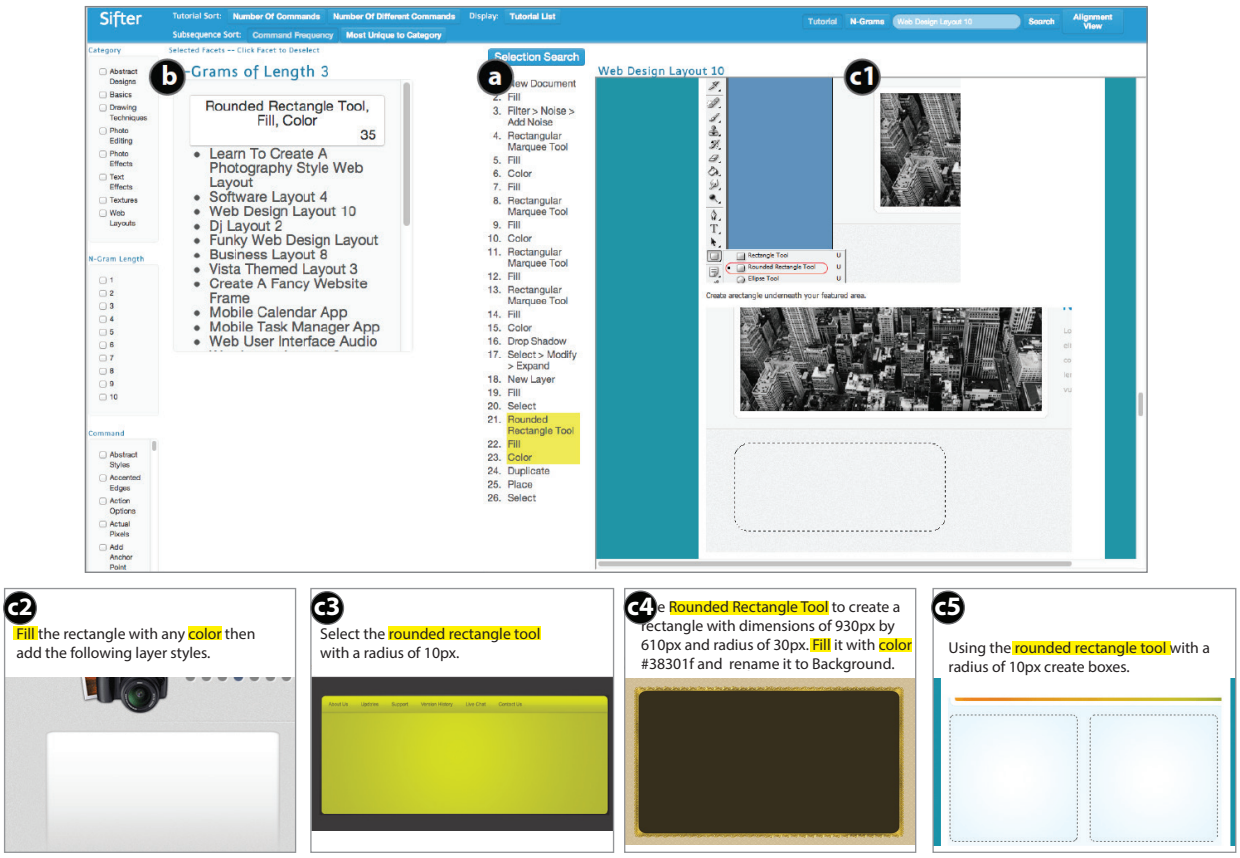


Figure 4.6: The command sequence "Rounded Rectangle Tool, Fill, Color" is used to create a content box. After highlighting this sequence and clicking on the Selection Search button in the Tutorial View (a), the browser returns a list of tutorials that contain this sequence of commands (b). This list of tutorials can be used to find additional styles for content boxes (c1-c5).

starting from the base tutorial. Figure 4.2c shows an example of the pairwise layout. The edit distance (Section 4.3.2) also provides correspondences between commands (either exact matches or substitutions). Our visualization connects matching commands with lines.

In Figure 4.7, we use the alignment view to compare seven different hair recoloring tutorials in our database. We set the "Hair Recoloring in Photoshop" tutorial as the base tutorial and show both, the one-to-many and pairwise layouts. The pairwise layout immediately reveals that there are three different ways to recolor hair. Tutorials a and b primarily use the "Brush Tool" and "Hue/Saturation" to complete the task. Tutorials e, f and d all use a similar four-command subsequence of "Invert, New Layer, Fill, Blending Mode, Soft Light". Tutorial c is a combination of these two approaches, while tutorial g is completely different from the others. This view directly highlights the command-level similarities and differences between the set of tutorials and thereby allows learners to quickly build a model in their head of the different ways a task can be done.

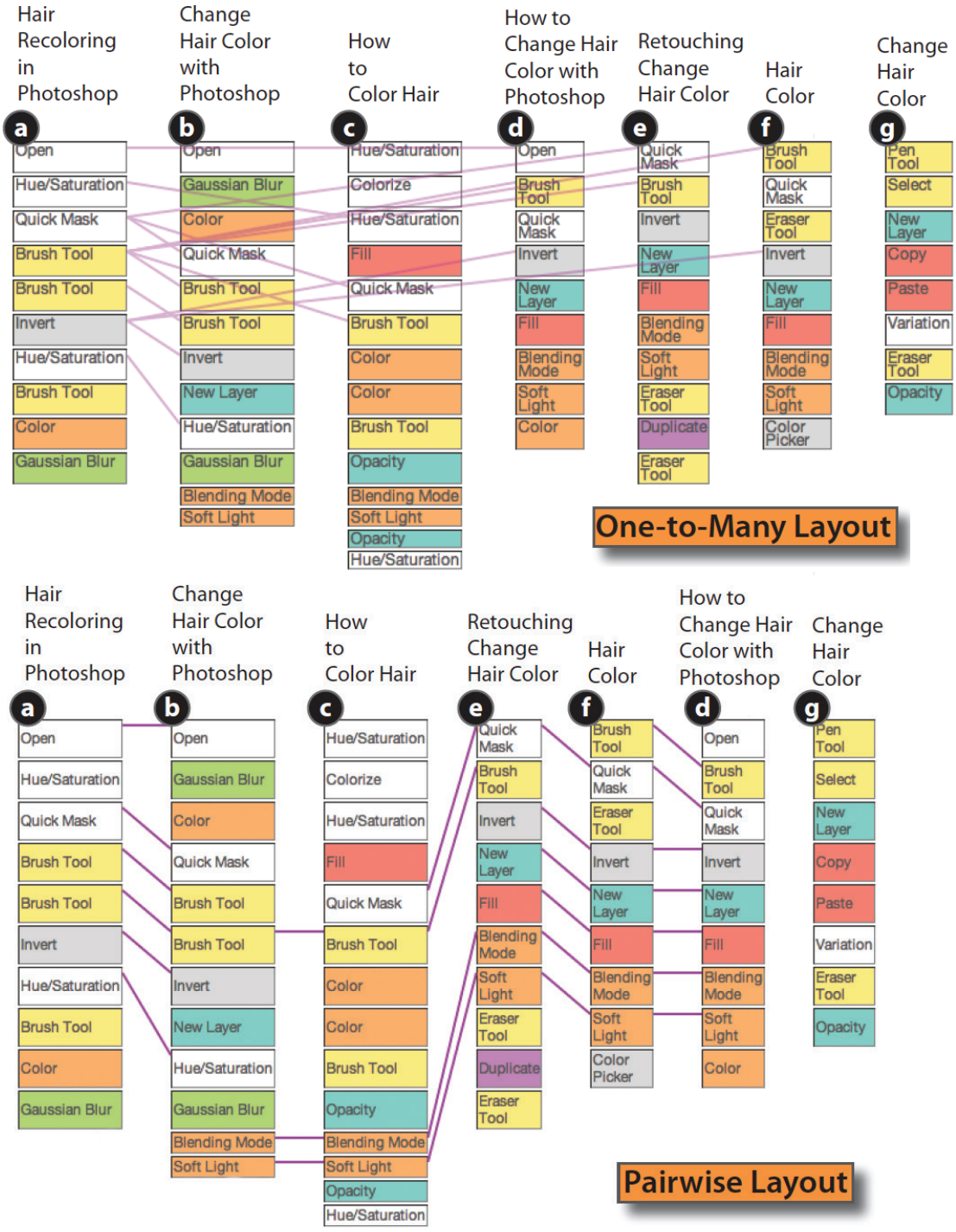


Figure 4.7: Comparison of seven hair recoloring tutorials using the Alignment View, which displays correspondence lines between the matching commands. The *one-to-all layout* (top) orders the tutorials (left-to-right) by their similarity to the first tutorial. The *pairwise layout* (bottom) builds the ordering by incrementally choosing the next tutorial as the one that is most similar to the previous tutorial.

4.3 Implementation

The browser interface relies on two technical components, the command extractor and the analyzer. The extractor takes all the downloaded tutorials as input and outputs the command-level structure of each one into a database of command sequences. The analyzer provides algorithmic tools to compare the command sequences in the database and identify common command subsequences within the tutorial collection.

4.3.1 Command Extractor

To extract commands from natural language tutorials we build on the classifier of Fourney et al. [36]. Their approach is designed to identify direct references to commands (i.e., exact string matches to the command name). We extend their approach to also handle colloquial references to commands (e.g., the tutorial says to "make a mask" rather than writing the command name "Create Layer Mask", Figure 4.1). In a survey of 60 randomly selected tutorials we manually identified about 18% of the command references as colloquial rather than direct. Thus, accounting for such colloquial references is essential for correctly extracting commands from text-based tutorials.

Fourney et al.'s algorithm first compares each word in a tutorial to a list of command names in the Photoshop menu hierarchy using exact string matching. Photoshop directly provides this list via Edit>Keyboard Shortcuts>Summarize. We then manually extend the list to include the colloquial references we identified in the survey of 60 tutorials. We also apply word stemming to automatically derive indirect references to commands. For example, applying stemming to the command name "Scale" produces the derived words "Scaling", "Scaled", etc. The final list contains direct, colloquial and indirect references to commands.

After the first stage of exact string matching, we follow the approach of Fourney et al. and apply a Naïve Bayes classifier with Witten-Bell smoothing to further eliminate false positive matches. As training data we again use the 60 tutorials for which we manually marked all true command references (direct, colloquial and indirect). Using leave-one-out cross validation we obtain an average precision of 90% and recall of 99%. Applying Fourney et al.'s approach without the additional entries for colloquial and indirect references, causes the recall to drop by 14%.

We use the command extractor to identify the sequence of commands used in each of the 2500 Photoshop tutorials in our collection. We save the extracted command sequences and their corresponding tutorials in a database.

4.3.2 Analyzer

The analyzer provides algorithmic tools to compare command sequences and reveal patterns within the database of photo manipulation procedures. More specifically, it provides a distance metric for comparing command sequences, and techniques for identifying common subsequences of commands.

Comparing Command Sequences

To reveal the underlying structure of photo manipulation procedures, we need to compare the command sequences stored in our database and determine their similarities and differences. We use the Needleman-Wunsch edit distance [73] to compare two sequences of tutorial commands. Informally, it computes an alignment score between the two sequences and measures the number of commands that need to be inserted, deleted or substituted to change one sequence into another one. Therefore, it requires a scoring matrix that encodes the penalties for inserting, deleting and substituting commands. We use a constant penalty for insertion and deletion, and build a specialized penalty matrix for substitutions based on two objectives; (1) we increase the penalty for substituting commands that differ significantly in functionality and (2) we increase the penalty for substituting commands that are most important to the tutorial.

To identify commands with similar functionality we adopt the approach of Kong et al. [60] who observed that the Photoshop menu hierarchy groups together similar commands (e.g., all filters appear in the same part of the hierarchy). Therefore, we set the functional similarity penalty P_{FS} to the distance between commands in the menu hierarchy. To compute the importance of a command we use term-frequency-inverse document frequency (tf*idf) which is commonly used in information retrieval to determine the most descriptive words within a document (i.e., words that appear multiple times in the document but rarely appear in other documents). We set the importance penalty P_I to its tf*idf value where we treat each tutorial as a document. Then given any two commands we compute the total penalty as $P_{FS} \cdot P_I$.

Computing Frequency of Subsequences

Subsequences of commands that occur frequently within the whole tutorial collection or within tutorials of a certain category (e.g. Photo Effects, Design, Painting) often encode higher-level strategies that users employ widely across tutorials and in many different contexts. To identify these frequently occurring subsequences, we first collect every command subsequence N-Gram of length 1 to 15 for each tutorial in the collection. We then build a frequency table by hashing each resulting N-Gram and incrementing the count each time we visit the same bin. However, some Photoshop commands can be executed in arbitrary order and produce the same results. We use distance-based matching to account for such swaps. For each subsequence of length N we compute the edit distance to all other subsequences of length $N-2$ to length $N+2$ (we limit the computation to this range for efficiency). If the distance is less than a small threshold we increment the frequencies of both subsequences.

To identify subsequences that occur frequently within a specific tutorial category, but less frequently when considering the whole collection, we compute the term-frequency-inverse document frequency (tf*idf) on the subsequence frequency. More specifically, we compute the number of times a subsequence occurs within a category, normalized by the total number of times it occurs across the collection.

Evaluation Tasks

1. Find a common command in the category of Web Layouts.
   (Find a common command)
2. Find a command used more often in Photo Editing than in Web Layouts.
   (Compare single commands between categories)
3. Find a command unique to the category of Photo Editing.
   (Find a unique command)
4. Find two uses for the Gradient Overlay command.
   (Find different uses of a command)
5. Given one tutorial with the subsequence *Round Rectangle, Fill, Color* for creating a content box, find two more examples of this task.
   (Find multiple examples of a 3-Gram command sequence)
6. Given a task, list tutorials that follow the most common method for this task."Method" refers to a series of commands shared by one or more tutorials as shown in the Alignment View.
   (Find the most common method to perform a task)
7. Given a task, list tutorials with the three most different command flows.
   (Find different methods for a task)

Table 4.1: The evaluation tasks asked users to perform a variety of browsing and analysis tasks based on the command-level structure of tutorials.

4.4 Evaluation

We conducted an informal user evaluation of our browser interface with three goals: (1) to gain feedback on the usability and utility of its features, (2) to gauge user interest in exploring the underlying command-level structure of tutorial collections and (3) to compare task performance and user preference between our browser interface and keyword search – today's status quo technique for exploring online tutorial collections.

4.4.1 Method

We recruited nine participants (age range 20-35), who self-rated their Photoshop expertise on a 5-point scale from novice to expert. Three rated themselves as Photoshop novices, three as intermediates and three as in-between. We first led a 7-minute walk-through of the interface to briefly explain each feature to the participants. Then, we asked the participants to complete a series of exploration and analysis tasks using our browsing interface. We told them that they could abandon any task if they estimated it would take over 10 minutes to complete. We designed the tasks to exercise the four main interface elements: browser facets, the table of commands, the selection search, and the alignment view. The complete set of tasks is listed in Table 4.1.

We asked participants to answer 5-point Likert-scale questions on their understanding of the

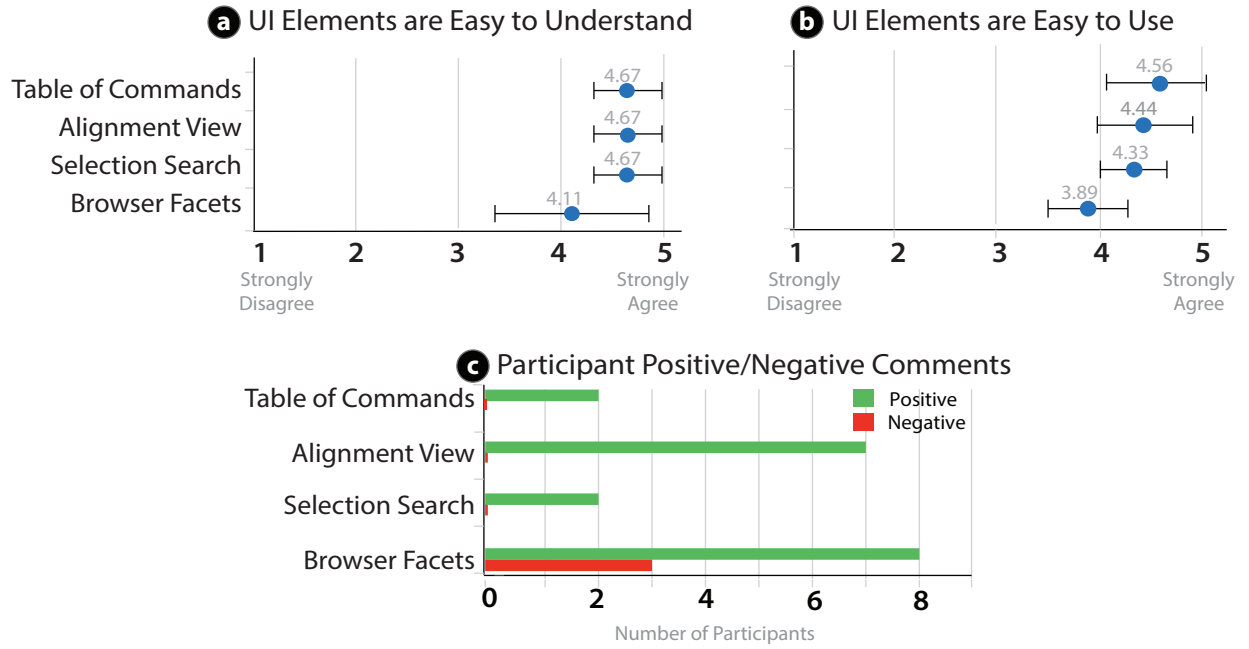


Figure 4.8: (a) Participants were asked to rate if they understood each interface element, and (b) if each element was easy to use on a scale from 1 - Strongly Disagree to 5 - Strongly Agree. (c) Number of participants leaving positive or negative comments in the open responses.

interface elements and the ease of use of these elements. We also asked about the usefulness of a variety of exploration tasks enabled by our browser. Finally, participants could provide longer open-ended feedback and we manually categorized each response about an interface element as positive or negative.

4.4.2 User Feedback

Browser Interface is Easy to Understand and Use. Participants understood the interface elements and found them easy to use, giving relatively high ratings to all four of them (Figure 4.8a-b). However, the browser facets ranked lower than the other three elements in both understanding and ease of use. Nevertheless, eight of nine participants mentioned positive aspects of the faceted browser in the open-ended feedback (Figure 4.8c). The three participants who left negative comments in the open feedback mentioned low-level usability concerns; they thought the term "N-Gram" was too technical and found that filtering by facets was somewhat slow. This data suggests that participants found using facets more complex than the other interface elements. However, they recognized that facets also enable more powerful analysis and exploration of the tutorial collection.

Tasks Enabled by Browser Interface are Useful. Participants generally rated the exploration

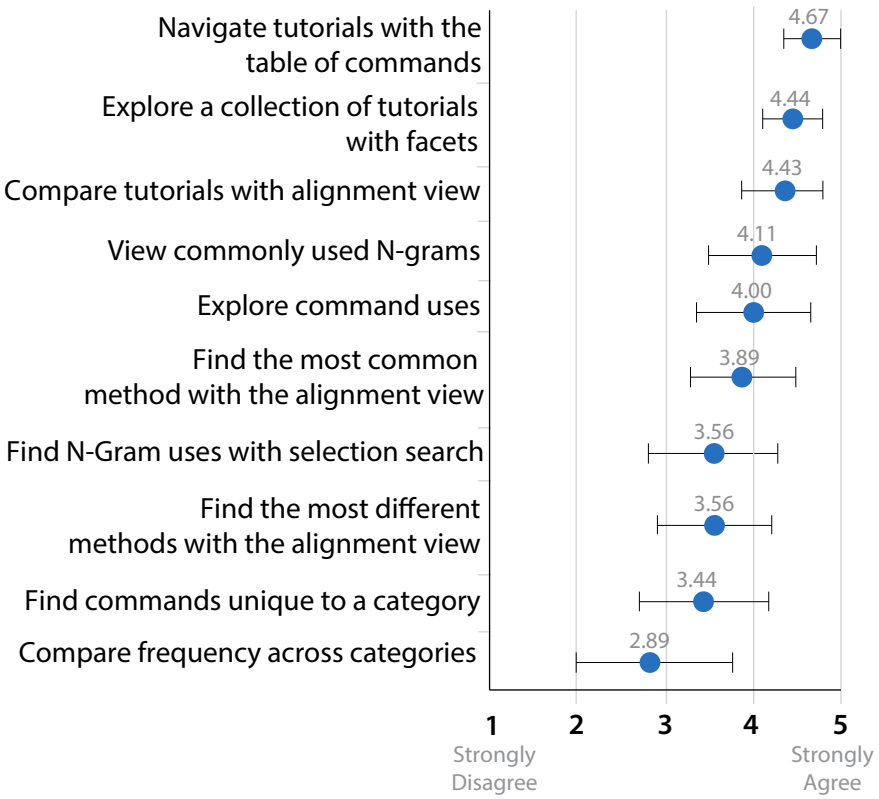


Figure 4.9: Participants were asked to rate usefulness of different browsing tasks on a scale from 1 - Strongly Disagree to 5 - Strongly Agree.

and comparison tasks enabled by our browser to be useful (Figure 4.9). In the open responses many participants mentioned that analyzing and comparing tutorials based on command structure gave them a new way to approach and learn Photoshop. Seven users mentioned they would like to use the Alignment View for assessing the similarities and differences between tutorials, finding the most common command flow to complete a task, and previewing the command flows to gauge familiarity. Five users wanted to use the browser facets to find more examples of unfamiliar commands and to refine tutorial search by command or category. As one user suggested, "Sometimes I just want to know how to use one tool and I can't find a good tutorial just by Googling it". A single user was not interested in command structure as they only wanted to search by task. Nevertheless eight of the nine participants explicitly stated that they would use our browser if they could.

4.4.3 Comparison of Browser Interface and Keyword Search

The evaluation of our browser was designed to assess performance on tasks we believe to be difficult with current tutorial browsing tools. To verify this intuition, we also asked our participants to

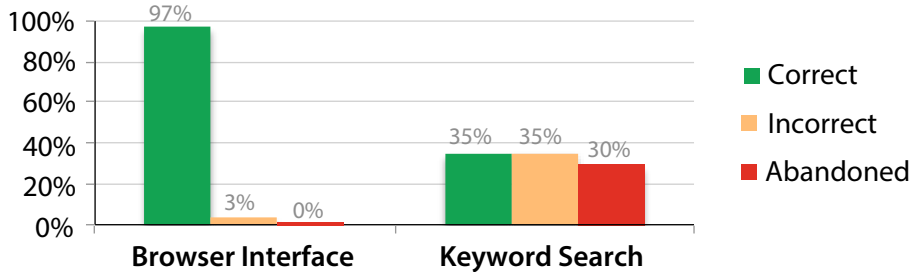


Figure 4.10: Correct, incorrect, and abandoned task percentages for our browser and keyword search tasks.

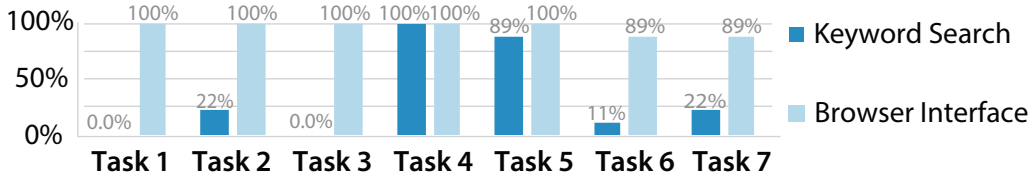


Figure 4.11: Percentages of correct answers for our browser and keyword search tasks.

execute each task (Table 4.1) with a keyword search interface limited to the tutorials in our corpus. By comparing task accuracy between interfaces, we evaluated whether our browser provides additional functionality over keyword search. To create a keyword search interface we used the Google Custom Search tool [39]. We ran a within subject evaluation so that each participant performed the same set of tasks using both our browser and keyword search interfaces. However we changed either the concrete tutorial category or the command name mentioned in the task so that subjects could not directly reuse their work. To measure task accuracy we first enumerated all possible correct answers for each task. If the participant gave an answer in this set, we marked it as correct. We again told users that they could abandon tasks that would take longer than 10 minutes. Finally, we asked users to rate interface preference for each task.

Participants are More Successful and Accurate with our Browser. Participants completed all tasks with our browser, but only 70% of the tasks with keyword search (Figure 4.10). They were also far more accurate with our browser (97% vs 35% using keyword search). A one-way ANOVA finds that this difference is significant ($F(1,12) = 15.12, p = 0.0029$).

In the keyword search condition, participants only performed well on two of the tasks: *Find different uses of a command (Task 4)* and *Find multiple examples of a 3-Gram command sequence (Task 5)* with 100% and 89% accuracy, respectively (Figure 4.11). For these tasks participants directly entered either a single command or a command subsequence as the query and keyword search usually returned a set of matching tutorials. Participants then opened each tutorial and used

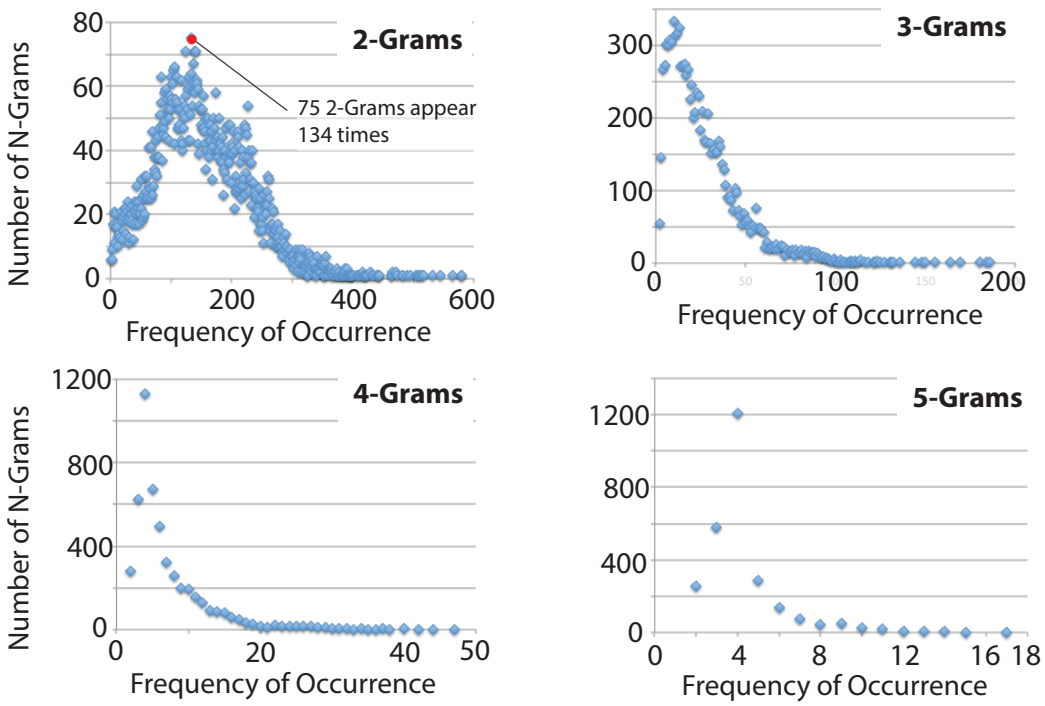


Figure 4.12: Number of N-Grams that appear at each frequency greater than one for N equals two to five.

text search within the page to find the command locations in the tutorial. All but one user preferred to use our browser over keyword search for the evaluation tasks, even for tasks where keyword search performed well.

In aggregate, our results suggest that our browser's interactions are accessible, useful, and provide the preferred interface for exploration and comparison tasks such as the ones used in the evaluation. However, we believe a larger-scale longitudinal evaluation would be necessary to rigorously verify these findings for daily work.

4.5 Corpus Analysis and UI Design Implications

While our analysis tools and browser interface are best used for exploration starting from a specific tutorial, a tutorial category, an N-Gram length or a set of commands, we have also used them to more globally analyze the command-level patterns in our database. Our analysis leads to several implications for designing image manipulation software.

We have found that each tutorial category contains some commands that appear more frequently within that category (Table 4.2). For example, 79% of the uses of the "Clone Stamp Tool"

Command(s)	Drawing/ Painting	Web Layouts	Photo Manipulation	Text Effects
Clone Stamp Tool	9	9	79	3
Perspective	22	14	61	4
Adjustment Layers, Clipping Mask, Black & White	0	0	100	0
Copy, Paste, New Layer	15	18	55	12
Select, Rounded Rectangle Tool, Fill	0	100	0	0
New Layer, Pen Tool, Fill	10	40	50	0

Table 4.2: Examples command sequences and their distribution across the different tutorial categories.

appear in tutorials within the Photo Manipulation category, but only 9% appear within the Web Layouts category. This tool enables photo retouching and is therefore less useful in the context of web design. Similarly, some command subsequences occur primarily within particular categories. For example, the three-command subsequence "Adjustment Layers, Clipping Mask, Black & White" occurs only within the Photo Manipulation category and does not appear in any other category. This subsequence selectively desaturates the image and thereby highlights the remaining colored region. This data suggests that it may be beneficial to group the most important commands for each high-level category in the Photoshop interface. Alternatively, one could also design custom UI panels for each category. Since users typically only work within one category per Photoshop session, quick access to these common commands and command strategies could speed up and simplify their workflows.

We have also analyzed the frequency of all N-Grams of length 2 to 5 across the complete tutorial corpus. In Figure 4.12 we plot the number of N-Grams that appear at each frequency greater than one. For example, 75 different 2-grams appear 134 times. Not surprisingly, the rightmost tails of these plots are low, indicating that there are few command subsequences that appear at high frequency. Since these are the most common N-Grams and there are relatively few of them, learners could focus on these command subsequences first to quickly become proficient with Photoshop. We also find that the number of N-Grams that appear at the lowest frequencies is relatively small, indicating that if an N-Gram appears in a tutorial, it is likely to appear many times in other tutorials. Since many command patterns occur frequently, it may be possible to watch users as they execute commands in Photoshop and infer the command-level strategies they are applying. The Photoshop interface could then highlight the next commands in the sequence, or notify users if they deviate from a sequence.

4.6 Next Steps

We have presented a new browser interface that allows users to navigate, explore and compare the command-level structure of a large collection of image manipulation tutorials. This interface helps learners better grasp the underlying structure of photo manipulation procedures and thereby facilitates the associative stage of learning. We plan to further extend our command extraction algorithm and browser interface.

Extracting parameters in text-based tutorials. We have applied supervised machine learning to identify the command sequence described in each tutorial. A challenging task is to also extract the parameter values for these command sequences. Identifying parameter values is difficult as they are often not specified, or only available within the screenshots of the application that were used to illustrate the step. However, access to these parameters would allow us to extend our interface so that users can compare the range and effect of possible values for a particular operation. Furthermore, for certain manipulations, it may be possible to also automatically apply the sequence of operations with their corresponding parameter values to new input images.

Naming multi-command strategies. Our faceted browser allows users to identify subsequences of commands that appear frequently in our database. These subsequences often represent successful higher-level strategies. However, to identify the higher-level strategy that corresponds to a particular subsequence, users need to browse through tutorials that contain this N-Gram. While our browser provides quick access to all such tutorials, a user still has to read through them to identify and name them. This process can be tedious and time-consuming. As future work, we plan to use plan recognition techniques [18, 12, 103, 16] to automatically name and describe multi-command strategies. Users could then more quickly gauge their interest in learning a particular strategy, and they could also use keyword search to find relevant strategies.

Suggesting related tutorials. We plan to track users as they are executing a task and compare the operation history with the command sequences in our database. We may then be able to recognize the task the user is performing. Using our command sequence edit distance, we may also be able to suggest related tutorials for the task. Having access to related tutorials would allow learners to better understand how they can use similar sequences of operations in different contexts.

Chapter 5

Transferring Procedures

In the autonomous stage, after users master a photo manipulation procedure, they often repeatedly apply it to collections of images. Current macro authoring tools facilitate this process by allowing users to record and replay sequences of operations. Such sequences are composed of global and local operations (as defined in Section 3.2). Together, they allow users to perform three types of low-level operations: 1) select a region, 2) draw a brush stroke and 3) apply an image processing operation within the selected region or along the brush stroke. To properly apply photo manipulations, users must adapt the *location* of the selection region, the *paths* of the brush strokes, and the *parameters* of the image processing operations to the content of the underlying image. Current macros cannot automatically adapt such content-dependent operations to new target images and are thus inappropriate for many common photo manipulations.

We present a more comprehensive approach for generating content-adaptive macros that can automatically transfer selection regions, brush strokes and operation parameters to new target images. Our approach is a framework in the sense that we can apply these transfer mechanisms independently of the class of photo manipulations that we consider. We demonstrate our framework on the three classes of the most commonly-used photo manipulations (Section 1.2): face, landscape and global manipulations, but also consider extensions to other classes of manipulations.

Our key idea is to learn the dependencies between image features (e.g. color, gradients, labels from object recognition, etc.) and the locations of selection regions, the paths of brush strokes and the parameters of image processing operation. To learn a photo manipulation, we require multiple operation-level training demonstrations of the manipulation. We exploit knowledge of the low-level operations to more robustly adapt the manipulation to new target images.

One important advantage of our framework is that it supports a practical workflow for authoring and applying content-adaptive macros. Users create our macros the same way that they create traditional macros in Photoshop – by recording a demonstration of a sequence of image editing operations. While this recorded macro can automatically apply the manipulation to new target images after just one demonstration, the results usually require some corrections or modifications. Often the errors are due to either incorrect labeling from our automated image labelers or insufficient data for our learning algorithms. To help users detect and correct such errors, we provide visual feedback and interactive correction interfaces. Our system then uses each corrected transfer as an

Figure 5.1: After demonstrating a snow manipulation on 20 landscape images, our content-adaptive macro transfers the effect to several target images.

additional demonstration that improves the quality and robustness of the macro. Thus, authoring a content-adaptive macro is a continuous, incremental process that is simply a by-product of applying the recorded manipulation to new images. We show that after about 20 training demonstrations, corrections are rarely required and our framework has enough training data to successfully transfer many common face, landscape and global manipulations. Figure 5.1 shows an example of automatically transferring a snow manipulation to two new target images. The macro correctly added snow to the new mountain and desert scenes. By allowing users to create macros in this incremental fashion and to easily modify or correct macro transfers, our framework enables a significantly more complete and usable workflow compared to purely automated techniques.

To evaluate the quality of our macro results, we ask viewers to compare images generated using our content-adaptive macros with and without interactive corrections to manually generated ground-truth images. We find that they consistently rate both our automatic and corrected results as close in visual appearance to the ground-truth. We also conduct a small informal study with professional photographers, to evaluate our proposed macro authoring workflow. The results from this study suggest that the visual feedback and interactive correction features of our user interface are effective and practical in the context of real world image editing workflows. While our framework does not facilitate the learning process for a person, it successfully teaches photo manipulation procedures to computers and thereby removes most of the tedium of having to manually execute the same sequence of operations over and over again.

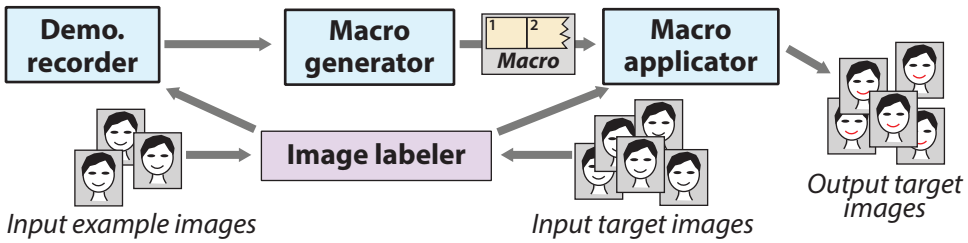


Figure 5.2: System overview.

5.1 System Overview

Our framework for generating content-adaptive macros contains three main components (Figure 5.2). The ***demonstration recorder*** captures example manipulations as authors demonstrate them in Photoshop (Section 5.2). The ***macro generator*** takes one or more example demonstrations as input and produces a content-adaptive macro as output (Section 5.3), while the ***macro applicator*** executes the macro on new target images. Our framework also provides visual feedback and interactive controls for correcting errors that may arise as the macro is demonstrated or applied. These interfaces enable an incremental workflow where each time a macro is applied and corrected it becomes a new training demonstration (Section 5.4).

Like the tutorial generator presented in Chapter 3, our framework for generating macros also relies on automated image labeling to segment input images (both example and target images) into labeled regions. Our ***image labeler*** currently applies the same face recognizer [105] and outdoor scene recognizer [48] as described in Section 3.4 to automatically detect such regions. In addition, the labeler also applies a skin recognizer [51]. We use the default parameter values for these recognizers, and together they label a 600×900 image in about 10 seconds. Our framework is extensible and can incorporate new recognizers as they become available (see Section 5.7).

5.2 Demonstration Recorder

When a person demonstrates a manipulation, we record the sequence of operations and the parameters for each operation. As described in Section 3.2, image manipulation applications provide local and global operations. Together, they allow users to perform three low-level operations: selections, brush strokes and image processing operations. Each low-level operation includes user-specified parameters that affect how they operate. We define in more detail the low-level operations and give example parameters in parentheses.

- ***Selections:*** These operations allow users to select a region of the image. The default selection is the entire image. Subsequent operations affect only the pixels within the selection region. Examples: free-select (location of the selection), by-color-select (range of colors to select).

- ***Brush strokes:*** These operations affect all pixels that lie within a brush region surrounding a stroke path. The stroke path and diameter parameters control the location and size of the stroke. Brushes may also include parameters specific to the adjustment. Examples: color brush (brush color), blur brush (blur strength).

- ***Image processing operations:*** These operations modify the pixels within the selection region by applying a filter, a color adjustment, or a spatial transform to them. Examples: sharpening filter (sharpening radius), contrast adjustment (strength), and rotating the selection region (angle).

The recorder used for generating content-adaptive macros is similar to the one presented in Chapter 3. However, our macro framework only requires recording changes to the application state, i.e. the sequence of operation settings and their parameters. We thus simply leverage the action recording capabilities of Photoshop's built-in *ScriptListener* plug-in. We then apply the clean-up and grouping techniques described in Section 3.3 and 3.5.1 to simplify the raw recordings by eliminating unnecessary operations and merging repetitive operations. Our framework requires that users perform all of the example demonstrations for a particular manipulation using the same sequence of operations, so that after clean-up and grouping the sequences are in one-to-one correspondence with one another. In Section 5.4 we present an interface designed to help users demonstrate the operations in the same order.

5.3 Macro Generator

To generate content-adaptive macros, our framework learns the dependencies between image features and the parameters of the selections, brush strokes and image processing operations comprising a manipulation. We first present the set of image features our framework considers (Section 5.3.1) and then describe the algorithms for transferring selection regions (Sections 5.3.2), adapting the locations of brush strokes (Section 5.3.3) and learning non-spatial adjustment parameters for the operations (Section 5.3.4).

5.3.1 Features

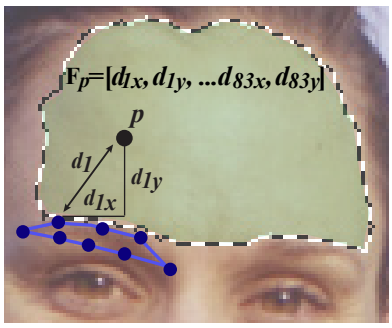
For machine learning algorithms to adapt photo manipulation macros, it is essential to identify features that correlate with the parameters of the selections, brush strokes and image processing operations. We analyzed the relationship between parameter settings and image features in common photo manipulations [49, 56] and found that users often set parameters based on the following features.

Pixel-level features. These features are local descriptors of the image that are based on pixel values. Such features include color, contrast, saturation, luminosity, texture, etc. Selections, brush strokes and image processing operations frequently depend on such features. For example, a user might increase the contrast of a region based on its current contrast or apply the healing brush to a dark (low luminosity) region under the eyes to remove bags. Many pixel-level features encode

redundant information since they are all based on pixel values. We have found that *color features* (L*a*b* space pixel value), *contrast features* (L* gradient magnitude) and *luminosity histogram features* (range, peak and median of histogram of L* values) are sufficient for learning pixel-level dependencies. Although the luminosity histogram features are partially redundant with the L* color feature, we include them because many image processing operations (e.g. color curve adjustment, luminance levels) are designed to directly modify aspects of the luminosity histogram. Therefore these features are better predictors of many image processing parameters than the L* color feature alone.

Label-based features. They describe the semantic content of an image. The location of selections and brush strokes often depend on such image content. For example, a user might select the forehead of a person to reduce its shininess or brush along the horizon of the sky to intensify a sunset. Similarly, parameters like the brush diameter often depend on the size of the object that the user brushes over. To capture these dependencies, we use *label features*, *landmark offset features* and *size features*. Our image labeler (Section 5.1) applies recognizers to compute a set of labels (e.g. eyes, skin, sky, ground, null) for every pixel in the image. The *label feature* is a bit vector representing the labels associated with each pixel. Some recognizers also provide landmark correspondence points that can serve as references for location parameters. For example, Zhou et al.'s [105] face recognizer detects the 2D positions of 83 landmark points (e.g. the outside corner of the left eye) that correspond across images of different faces. However, our skin [51] and outdoor scene [48] recognizers do not provide landmark points. Since the shapes of skin and outdoor regions like sky, ground, etc. can vary drastically from one image to another, it is unclear how to put such shapes in correspondence. We therefore use a weak shape descriptor and treat the 4 corner vertices of the axis-aligned bounding box of each labeled region as landmarks. We leave it as future work to identify more appropriate shape descriptors when landmarks are not available.

We compute *landmark offset features* as the offset $x-$ and $y-$coordinates between the pixel and each landmark point returned by our labeler (see inset). To better capture spatial relationships between selection or brush stroke locations and the image, we also include the 4 corner vertices of the image and of the axis aligned bounding box of the selection region as landmarks in computing the landmark offset features. We normalize the landmark offsets by the width and height of the labeled region, selection region or the entire image depending on the region the landmark point belongs to. Although the offset distance to a single landmark point may be weakly correlated with location, the offset distances to a collection of landmark points can be very discriminative. We compute the *size features* as the width and height of the bounding box of each labeled region.



Operation-based features. These features are characteristics that are specific to a given operation. We include two such features that are characteristic of brush stroke operations; the *stroke length feature* describes the length of the stroke in pixels and is stored as a single value for each stroke, while the *stroke orientation feature* encodes the local orientation of the strokes as a discrete value from 1 to 8 indicating for each pixel on a stroke, the position of the next pixel on the stroke. Together these features enable our framework to transfer strokes that have approximately the same

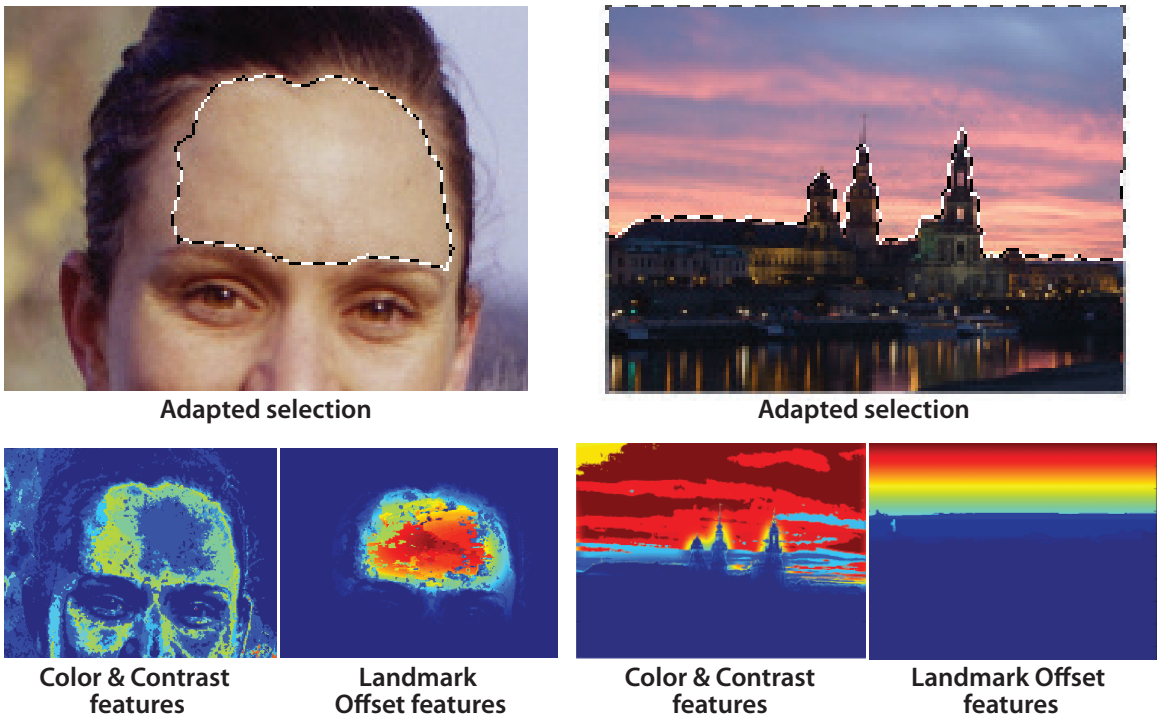


Figure 5.3: ***(Left) Forehead Selection.*** The color and contrast features cannot discriminate between the forehead and skin area on the face. However, landmark offset features accurately predict the location of the selection. ***(Right) Sky Selection.*** In this case the color and contrast features are better predictors of the selection than the landmark offset features.

length and shape as the strokes in the demonstration.

5.3.2 Adapting Selections

Users typically select a set of pixels in the image that have a common property; e.g., they belong to the same object, they are similar in color, etc. To adapt selections to new target images, we model the selection regions of the training images using a feature vector $\mathbf{F_j} = [F_{j,1}, ...F_{j,N}]$ where j denotes a selected pixel in a demonstration image and $1...N$ are the features computed for each such pixel. We then apply this model to classify each pixel i of the target image into two categories, selected ($Sel = 1$) or not selected ($Sel = 0$), based on its feature vector $\mathbf{F_i} = [F_{i,1}, ...F_{i,N}]$.

We use Naïve Bayes for this classification task, because it is simple and flexibly allows adding new features. Although Naïve Bayes assumes features to be conditionally independent given a class, it is known to work surprisingly well in object categorization and information retrieval, even when this assumption is violated [68]. We determine whether pixel i is part of the selection region by computing $P(Sel = 1|\mathbf{F_i}) =$

$$\frac{P(\mathbf{F_i}|Sel=1)P(Sel=1)}{P(\mathbf{F_i}|Sel=1)P(Sel=1)+P(\mathbf{F_i}|Sel=0)P(Sel=0)}. \tag{5.1}$$

Because of the independence assumption, the likelihood of a feature vector for a selected pixel is given by

$$P(\mathbf{F_i}|Sel=1)=\prod_{k=1}^{N}P(F_{i,k}|Sel=1). \tag{5.2}$$

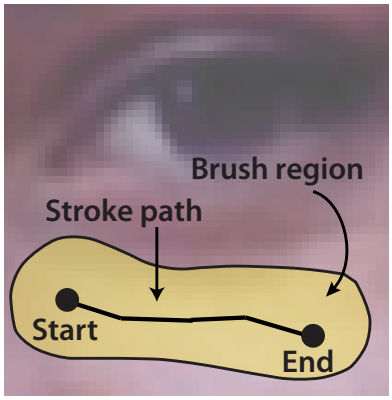
Rewriting equation 5.1 we obtain $P(Sel=1|\mathbf{F_i})=$

$$\frac{\prod_k P(F_{i,k}|Sel=1)P(Sel=1)}{\prod_k P(F_{i,k}|Sel=1)P(Sel=1)+\prod_k P(F_{i,k}|Sel=0)P(Sel=0)}. \tag{5.3}$$

To compute $P(F_{i,k}|Sel=1)$ and $P(F_{i,k}|Sel=0)$ we build histograms of all observed values of the k^{th} feature within either the selected or unselected pixels of the demonstration images. The prior probability $P(Sel=1)$ corresponds to the percentage of selected pixels across all training examples. We then include all pixels of the target image where $P(Sel=1|\mathbf{F_i})>t_{sel}$ in the selection region for that image. We set the selection threshold $t_{sel}=0.27$ for all examples in this chapter. Finally, we apply morphological operators to the selection mask to eliminate thin strips and isolated pixels.

The feature vector $\mathbf{F_i}$ is a $5+2N+L$-dimensional vector composed of the 5 pixel-level features (3 for color, 2 for contrast), $2N$ landmark offset features where N is the number of landmarks in the image and L label features, where L is the number of detected labels in the demonstrations. Figure 5.3 shows how all the features are necessary to adapt selections to new target images.

5.3.3 Adapting Brush Strokes



A brush stroke is composed of a brush region and a stroke path (see inset). Brush regions are equivalent to selection regions, and we use the same kind of Naïve Bayes classifier as in Section 5.3.2 to adapt the brush region to a new target image. A stroke path is a curve representing the centerline of each brush stroke. We compute stroke paths using a Markov Chain model in which the next point on the stroke depends only on the state of the previous point on the stroke $P(S_i|S_{i-1})$. Markov Chains are commonly used to model time-varying processes and have been successfully applied to example-based synthesis problems in many domains including music [92], text [22], and curves [47, 52, 94]. While our approach is similar to previous example-based curve synthesis techniques, unlike the earlier methods which only consider geometric features of the example curves, our approach takes advantage of both geometric features as well as image-based features of the underlying example images. In our stroke model, we define three types of states: $S_{start}=(x,y)$ is the stroke start point in image coordinates,

$S_i = (x,y)$ is the i-th point on the stroke and S_{end} is a state marking the end of a stroke. We proceed in three steps:

Step 1: Initialize stroke start point. For each brush region we determine the position of the stroke start point. Since the start point is not preceded by any other state, we pick the point $p = (x,y)$ within the brush region that maximizes $P(S_{start} = p)$. In our model a point p is fully characterized by its feature vector $\mathbf{F_p} = [F_{p,1}, ...F_{p,N}]$ and $P(S_{start} = p) = P(\mathbf{F_p})$. We assume that each feature $F_{p,k}$ in the feature vector is independent so that $P(\mathbf{F_p}) = \prod_{k=1}^{N} P(F_{p,k})$. To compute $P(F_{p,k})$ we build a histogram for all observed values of the k^{th} feature across the set of known stroke start points in the demonstration brush strokes.

Step 2: Choose next stroke point. Once we have computed the start position, we treat the stroke path as a process where the location of the next stroke point is based on the location of the previous stroke point. The state S_i of the i-th point on the stroke can either specify the position of the point in image coordinates $S_i = (x,y)$, or specify that the stroke should end $S_i = S_{end}$. To determine the next state, we compute

$$\underset{S_{i+1}, q \in N(p)}{\operatorname{argmax}} \{P(S_{i+1} = q | S_i = p), P(S_{i+1} = S_{end} | S_i = p)\} \tag{5.4}$$

where $p = (x,y)$ is the position of the previous point on the stroke and q is chosen from the set of points in the neighborhood $N(p)$ of p. In practice we set $N(p) = \{(x \pm 1, y \pm 1)\}$, which is the 8-connected pixel neighborhood of $p = (x,y)$.

Since points are characterized by their feature vectors, we compute $P(S_{i+1} = q | S_i = p)$ as $P(\mathbf{F_q} | \mathbf{F_p})$. We assume that the individual features $F_{p,k}$ of each feature vector $\mathbf{F_p} = [F_{p,1}, ...F_{p,N}]$, are conditioned only on themselves and independent of the other features, so that $P(\mathbf{F_q} | \mathbf{F_p}) = \prod_{k=1}^{N} P(F_{q,k} | F_{p,k})$. To compute $P(F_{q,k} | F_{p,k})$ we build histograms of the transition probabilities for each feature k in the training demonstrations. Similarly we compute $P(S_{i+1} = S_{end} | S_i = p) = \prod_{k=1}^{N} P(S_{end} | F_{p,k})$ by building a histogram of the feature vectors for the last point of each stroke in our training data.

To reduce the number of training examples needed, it is often assumed that the markov chain is time homogeneous, such that $P(S_{i+1} | S_i) = P(S_{i+t+1} | S_{i+t})$ for all $t > 0$. To better capture the variation of each feature along the path, we assume that the process is time-homogeneous within a limited number of timesteps. We have found that $t = 5$ works well for color, contrast and landmark offset features, while $t = 1$ is necessary for stroke orientation and stroke length, since these features vary more quickly along the stroke path. While our approach requires more training data than setting $t = 0$, we have found it to result in better transfers because it more accurately captures the evolution of each feature along the path.

The resulting stroke path gives a greedy estimate of maximizing the likelihood. There are a variety of algorithms that could be used to refine the curve if desired [55, 5], but we have found our approach to work well in practice.

Step 3: Update brush region. After computing the stroke path, we dilate the path by the brush diameter d (we compute d using the approach of Section 5.3.4) and subtract the dilated stroke from the brush region mask. We then repeat this process and start a new stroke, until the area of

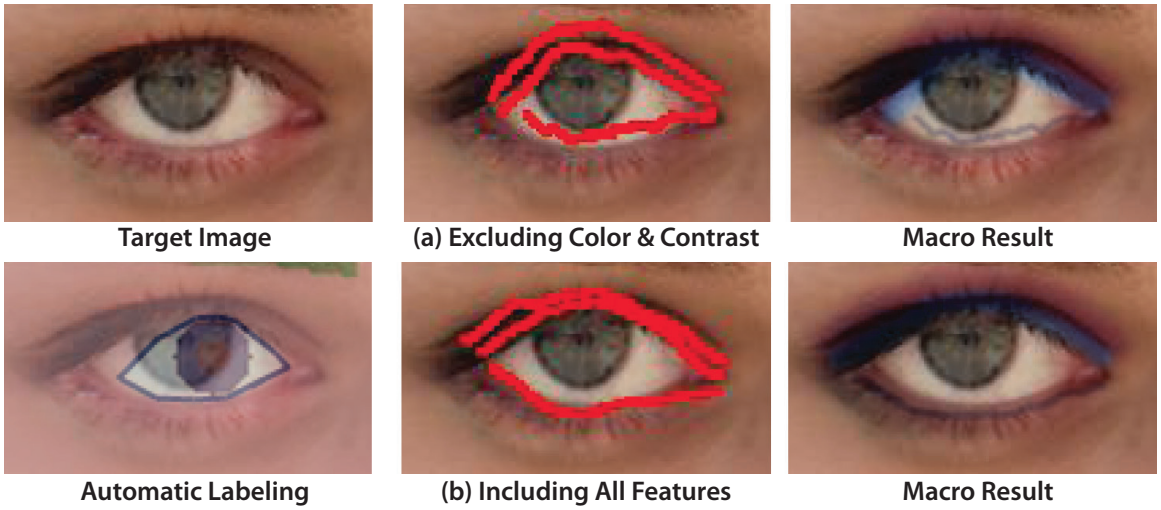


Figure 5.4: Landmark offset features provide a rough estimate of the location of the stroke path (a). But because the image labeler slightly mislabeled the eye, color and contrast features are necessary to correctly adapt the strokes to the contour of the eye (b).

the brush region mask is too small ($< 10\%$ of the brush area of the smallest stroke in the training data).

We use slightly different feature vectors $\mathbf{F_p}$ in steps 1 and 2. In step 1, $\mathbf{F_p}$ includes only three image-based features; color, contrast and landmark offset. For the color and contrast features, we use the median feature values across all pixels that lie within the brush diameter d of stroke point p in the training data. In step 2, $\mathbf{F_p}$ includes two additional geometric features; stroke orientation and stroke length. Through informal tests we have found that all features provide useful information to the model. For example, Figure 5.4 shows a brush stroke transfer for an eye makeup manipulation. Because of the color and contrast features, we can correctly transfer strokes even when the eye is slightly mislabeled.

5.3.4 Adapting Adjustment Parameters

To adapt the numerical image processing parameters to new target images, we must learn how the parameter values depend on the underlying image features. Our goal is to learn a function that maps the image features to a parameter value. Suppose y_i are observations of a parameter y we wish to learn and $\mathbf{F} = [F_1, ..., F_N]$ are a set of image features. Under a linear model, $y_i = c_1 * F_1 + ... + c_N * F_N$, we must compute the set of regression coefficients c_i that best explain our observations. Linear regression is a simple technique for computing these coefficients, but it can lead to overfitting. To avoid such overfitting, we use Least Angle Regression (LARS) [27] which is a variant of linear regression that constrains the sum of the absolute regression coefficients c_i and thereby causes many coefficients to be zero. This property ensures that only a small set of

Figure 5.5: Adapting the skin-tone manipulation (illustrated in Figure 1.5). The 20 training examples contain many images that require shifting the color balance towards yellow. Least Squares overfits the data and exhibits a yellow cast. LARS avoids overfitting and produces a better result.

image features are used for the prediction of a parameter. With 20 training examples we observe that LARS produces better results than least squares for many manipulations including the skin tone manipulation (Figure 5.5).

Image processing operations are applied to an active selection region or brush region. Often the intent is to strengthen or weaken the characteristics of this region with respect to the surrounding pixels. We therefore compute the pixel-level features for the active region and its complement. We use the median color and contrast features computed over the corresponding regions. However, several adjustment parameters (e.g. brush diameter, scale factor) depend on the size of objects in the image and so we include size features for all labeled regions that overlap the active region. To summarize, $\mathbf{F}$ is a $18+2N$-dimensional vector composed of 18 pixel-level features (9 for the selection region and 9 for its complement, consisting of 3 for color, 2 for contrast and 4 for the luminosity histogram) and $2N$ size features, where N is the number of labeled regions that overlap with the active region.

5.4 Workflow: Feedback and Correction

In our framework, the user can create a content-adaptive macro by recording a single demonstration of a manipulation. However, with only one training example, the macro cannot robustly adapt to new target images. Typically, the user needs to demonstrate the manipulation using the same sequence of steps on about 20 different examples to produce a robust macro. To help the user demonstrate and apply the manipulation to new target images, our framework provides two visual feedback and interactive correction panels. The *macro application panel* applies the macro to a new image and then allows the user to correct the transferred selections, brush strokes and adjustment parameters as necessary (Figure 5.6). The *labeling correction panel* helps macro authors correct

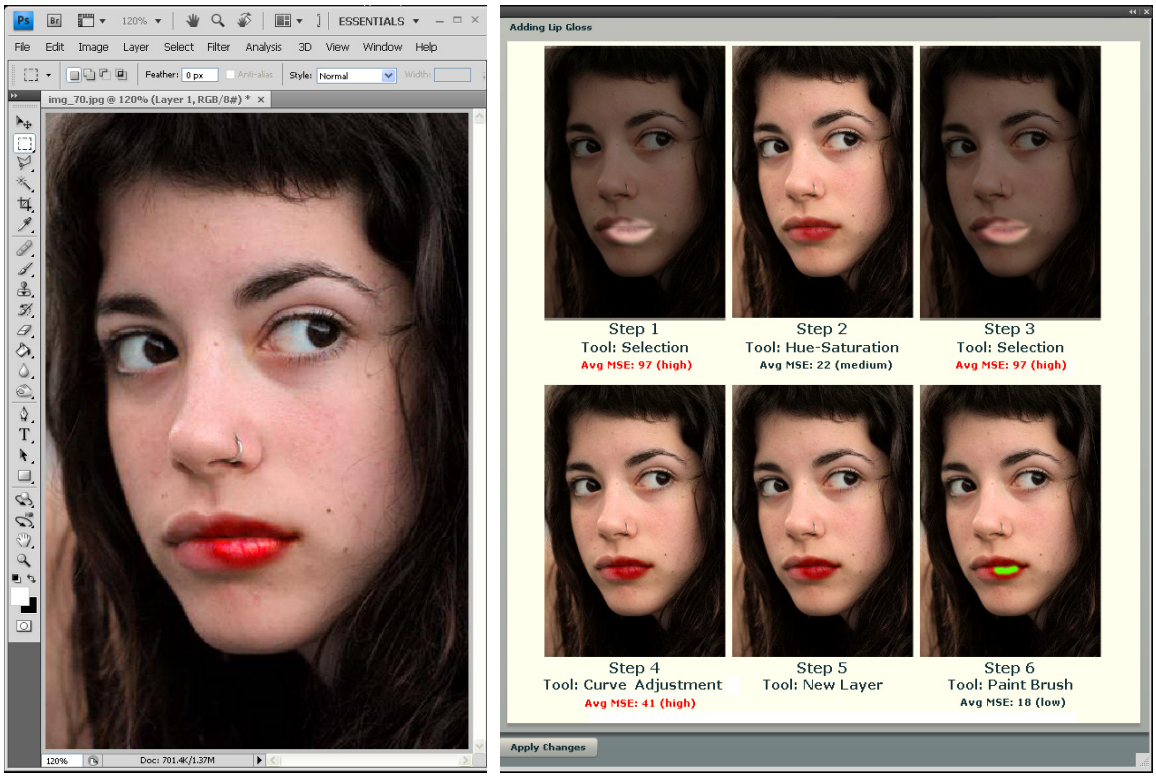


Figure 5.6: ***Macro Application.*** The panel shows the target image after each step of the macro. Selection steps show the selection region in gray. Brush stroke steps show the brush stroke in green. Users can click on any step in the panel to load the corresponding images in Photoshop and modify parameters as necessary and then re-executing the remaining steps.

errors due to our automated image labelers (Figure 5.7).

5.4.1 Macro Application Panel

Suppose an author demonstrates a lip gloss manipulation. After the first demonstration (see inset), our framework generates a macro application panel. Given a new target image, the panel presents the six steps comprising the manipulation and the image that results after applying each step to the new target (Figure 5.6). The visual feedback allows the author to quickly identify and correct errors in individual steps of the macro. Since the sequence of steps is fixed in the panel, it also prevents the author from forgetting steps or changing the order of steps when adding additional demonstrations.

In the example of Figure 5.6, the panel immediately reveals an error in adapting the selection region in the first step – our image labeler incorrectly labeled the lips. The user can click on any step in the panel and our framework executes all of the operations up to that step. The user can then take over and fix any incorrect parameters. In this case, the user clicks on the first selection step, manually selects the correct lip region and then uses our *labeling correction panel* to mark his new selection as lips (Figure 5.7). Although such manual labeling is not strictly necessary and our framework can learn a macro without it, more accurate labels yield more accurate macro adaptation. Similarly if the automatically generated red lip color were undesirable, the user could adjust the parameters of the color curve that reddens the lips by clicking on step four. After each such correction, our framework automatically executes and re-evaluates the parameters of all remaining steps. Once the user has finished correcting the image, our framework treats the target image with the corrected parameter settings as an additional demonstration.

5.4.2 Labeling Correction Panel

The labeling correction panel has three components. The top left image shows the current selection region in green, overlayed on the target image (Figure 5.7a). The top right image visualizes the labeled regions detected by our image labelers (Figure 5.7b). Below, the list of checkboxes indicates which of the detected labels correspond to the current selection (Figure 5.7c). Figure 5.7 shows the state of the panel after the user has noticed the error in step 1of Figure 5.6 and selected the correct lip region using Photoshop's selection tool. To label the current selection as lips, the user simply marks the upper and lower lip checkboxes; our framework then associates these labels with the current selection. Some labelers, such as our face labeler, also provide landmark correspondence points. Although our labeling correction panel does not automatically update such landmark points, we allow users to manually update the location of these points.

5.4.3 Macro Robustness

One challenge for a macro author is to evaluate when the macro is robust enough to distribute to other users. Thus, our macro application panel provides feedback on the robustness of the current macro. After every new demonstration, we run a leave-one-out cross-validation on the existing demonstration dataset. That is, we use one demonstration as the test data and the remaining demonstrations as the training data. We repeat this process for several rounds until every demonstration has been used once as test data.

For each round, we estimate how well the content-adaptive macro is able to adapt the parameters for the test image. For each step of the manipulation, we compute the mean squared error (MSE) between the image generated by our adapted macro and the ground truth image that we obtained with the original demonstration. Since many steps modify only a small region of the image, we avoid excessively low MSEs by including only the set of pixels modified in either the adapted image or ground-truth image. Furthermore, selections do not modify the image and we simply compute MSE on the selection mask for selection steps.

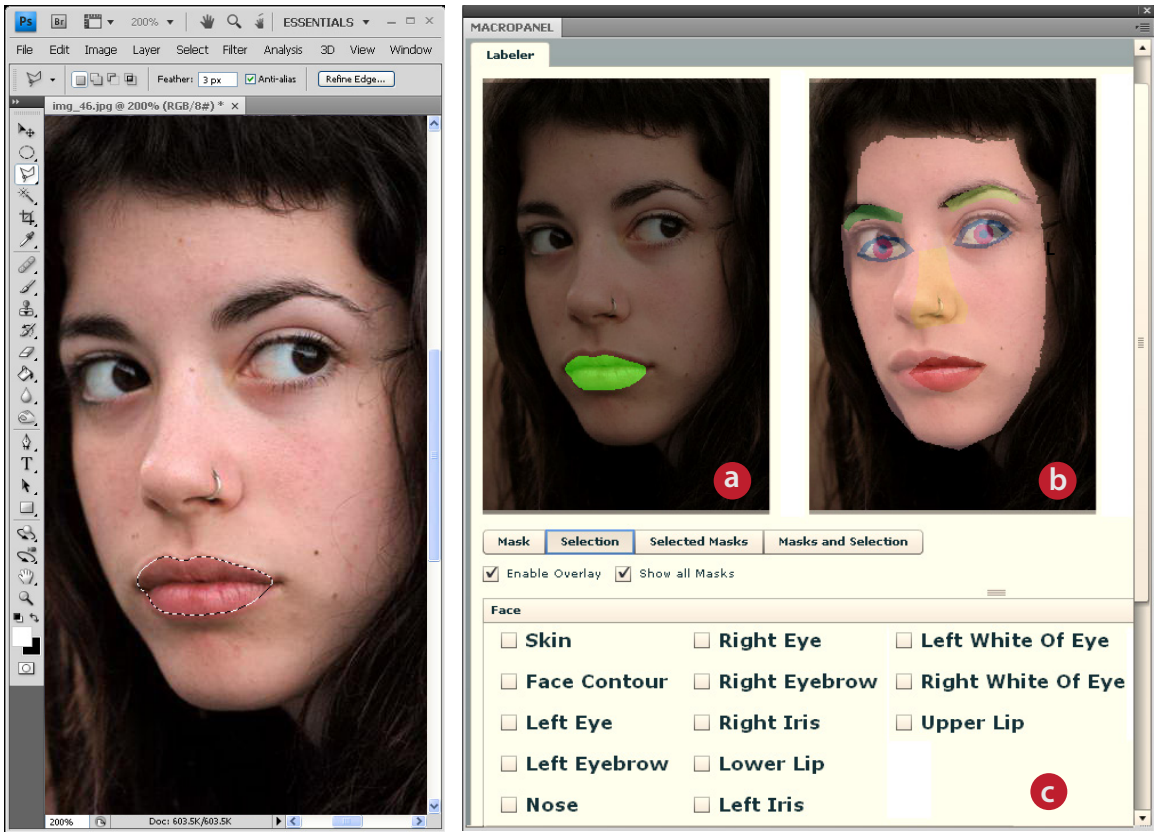


Figure 5.7: ***Labeling Correction.*** The panel shows the set of facial features detected. The eyes, nose and lips are clearly mislabeled (b). After the user selects the lips, the current selection is drawn in green (a). To correct the lip labels, the user can check the upper and lower lip in the list below; our framework associates the selection region with all checked labels (c).

The macro application panel displays the average MSE across all rounds for each step. Although MSE does not capture the perceived difference of pixel values, it indicates the level of similarity between the learned result and the ground truth demonstration. Uniformly low MSE values for all steps suggest that the macro is relatively robust and can be shared with other users. We typically observe low MSEs with 15-20 demonstrations. Note that this MSE measure only indicates robustness with respect to the demonstration images. It remains the task of the author to collect a training set that contains enough variety.

Manipulations	Number of Sel.	Strokes	Params.	Total	Dataset Size
Bag Removal	0	2-4	7	9-11	100/19
Contrast	1	0	4	5	100/18
Eye Makeup	0	6	6	12	100/49
Film Noir	0	5	23	28	100/15
Lip Gloss	1	1	5	7	50/4
Mustache	0	3-10	0	3-10	35/4
Skin Tone	1	0	6	7	50/14
Dark Sky	1	1	4	6	50/14
HDR realistic	1	5-9	26	32-36	50/7
HDR artistic	1	5-9	26	32-36	50/7
Landscape Enhancement	0	5-7	16	21-23	40/8
Rainbow	0	8-12	8	16-20	60/17
Smooth Waterfall	1-5	3-5	11	15-21	50/0
Snow	0	2	24-28	26-30	50/11
Sunset Enhancement	2	2-6	18	22-26	100/62
Black & White	0	0	6	6	100/0
Cross-processing	0	0	17	17	50/0
Lomo	0	0	11	11	50/0
Reflections	1-4	1-2	16	18-22	25/0
Typographic	1	0	2	3	35/0

Figure 5.8: Twenty manipulations (face – gray, landscape – green, global – pink) and the number of selections, brush strokes and adjustment parameters adapted for each one. The dataset size column reports the number of images used in the evaluation, followed by the number of images for which we hand-corrected the labeling.

5.5 Results

We have used our framework to generate content-adaptive macros for the 20 manipulations described in Figure 5.8, some of which are shown in Figures 5.1, 5.9, 5.10 and 5.12. All of these results were generated using our fully automated approach without any corrections through our application or labeling panels. We chose 7 face manipulations, 8 landscapes adjustments, and 5 global manipulations from popular photo editing books [49] and websites. The manipulations range from adding eye makeup to smoothing waterfalls to making the image look as if it were taken with a Lomo camera. In total the manipulations require adapting between 3 and 36 selections, brush strokes and adjustment parameters. For example, the smooth waterfall manipulation

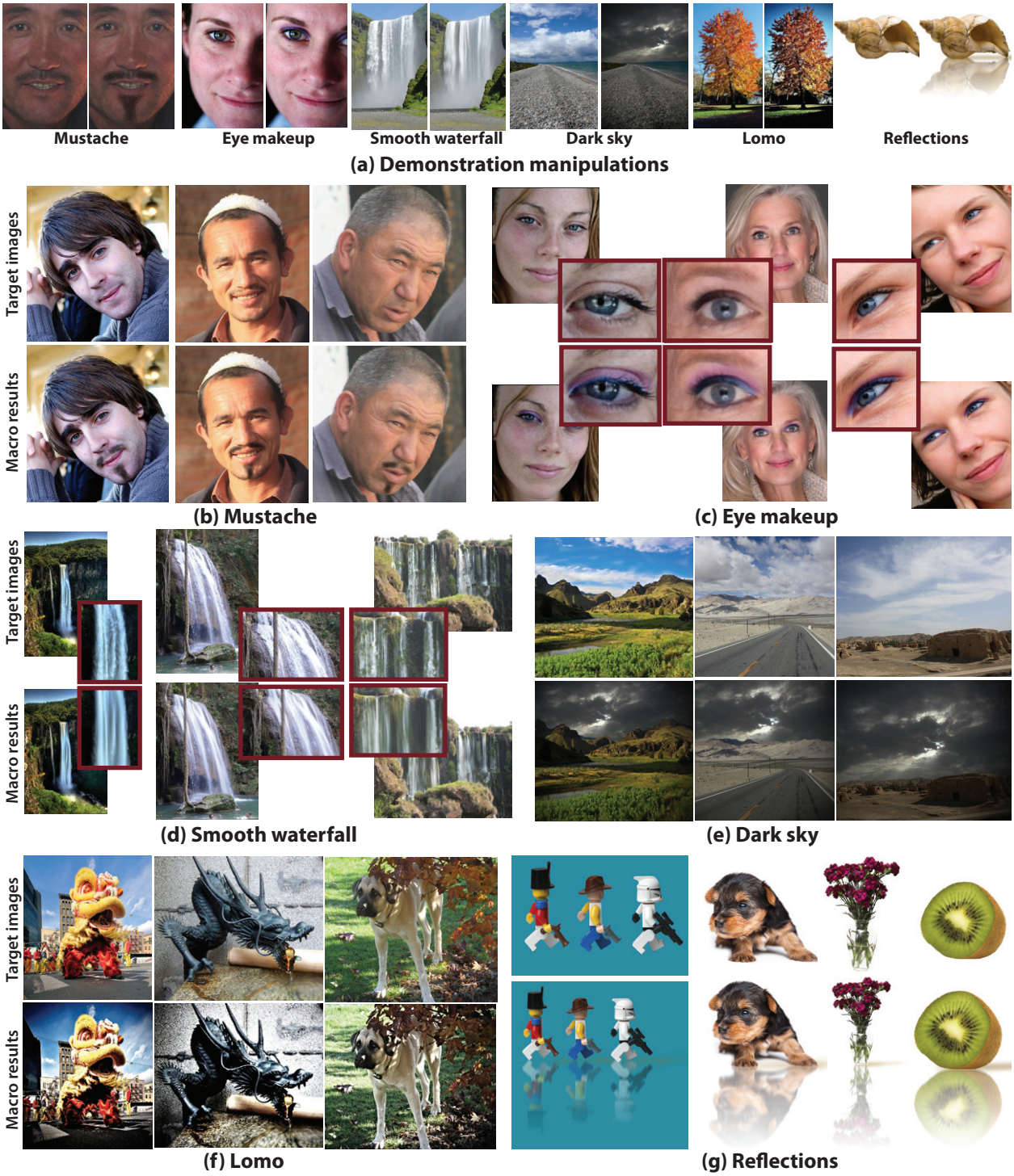


Figure 5.9: Six example manipulations. (a) Representative demonstration input image and result pairs. (b)–(g) Our content-adaptive macro results on new target images.

has the following steps (see project website[1] for descriptions of the other manipulations):

1. Select waterfalls and copy them onto a new layer (1-5 selections).
2. Apply motion blur filter on waterfall layer (2 parameters).
3. Create a layer mask and paint over areas where blur effect goes outside the waterfall regions (3-5 brush strokes).
4. Adjust opacity of waterfall layer (1 parameter).
5. Use warp tool to emphasize shape of waterfall (8 parameters).

To find suitable images for each manipulation we asked photographers and searched the Web (`flickr.com`) to build a dataset of high-quality images that photographers would want to manipulate.

As shown in Figures 5.1, 5.9 and 5.10, in most cases our content-adaptive macros successfully adapt the manipulation to new target images. Figure 5.9 includes 2 face manipulations (b-c), 2 landscape manipulations (d-e) and 2 global manipulations (f-g). The makeup and mustache manipulations (b-c) demonstrate the precise transfer of brush strokes when our labeler recognizes the faces. The waterfall manipulation (d) edits a semantic region of the image – namely the water. Although our framework does not include a water detector, it is still able to successfully transfer the manipulation to many images using the white color of the water (color feature) and the relative location of the water below the sky (landmark offset feature). The snow manipulation (Figure 5.1) includes up to 28 adjustment parameters. When correctly transferred, these parameters cause the ground, trees and even the reflected trees in the lake to appear as if they have snow on them. We include two versions of the same HDR manipulation, one trained with examples that produce an "artistic" look and the other trained to produce a "realistic" look. Figure 5.10 shows that our framework can learn the appropriate parameters for both looks.

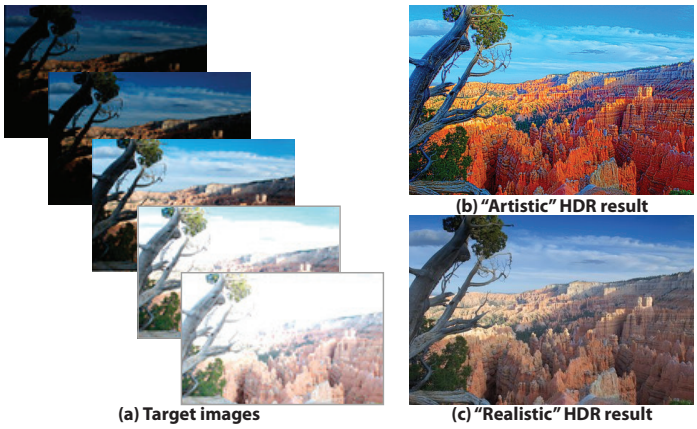


Figure 5.10: High dynamic range (HDR) results. Our system successfully transfers both "artistic" (b) and "realistic" (c) HDR manipulations based on the training demonstrations.

All of these results were generated using 20 training demonstrations, except for the mustache manipulation. Because adding a mustache only involves brush strokes and does not include adjustment parameters, we trained the macro using just 10 demonstrations. Figure 5.11 shows a few less

[1] All supplemental materials are at http://vis.berkeley.edu/papers/macros/

Figure 5.11: Less successful adaptations. Poor parameter estimation results in a washed out image (a). Incorrect selection transfer causes blurring in the region below the waterfall (b). Incorrect labeling results in misplaced brush strokes for the mustache (c) and eye makeup (d) manipulations.

successful examples that include misplaced brush strokes and poor parameter estimation. Both types of errors could be corrected using our application panel.

Timings. Each manipulation initially takes about 5-15 minutes to demonstrate, but with practice and using our macro application panel this authoring time usually reduces to about 1-5 minutes per demonstration. With 20 training demonstrations and images of size 600×900, our system requires about 3 minutes to learn how to adapt each selection, and about 1 minute to adapt each brush stroke or adjustment parameter. Thus, the training time for our manipulations is between 5 and 40 minutes depending on the number of operations. Applying the resulting macro to a new target image requires a few seconds to adapt each selection region and brush stroke operation, and significantly less than a second to adapt each adjustment parameter. These timings are based on our unoptimized MATLAB implementation.

5.6 Evaluation

To determine the effectiveness of our framework as a whole, we evaluate both the quality of the images generated by our content-adaptive macros, as well as the utility of our proposed macro authoring workflow.

5.6.1 Macro Results

To evaluate the quality of our macro results, we chose seven representative manipulations: bag removal, contrast, dark sky, eye makeup, film noir, lomo, and sunset enhancement. For each manipulation, we compare four different methods for adapting the manipulation with a dataset of 50–100 images:

1. ***Ground-truth.*** We generate *ground-truth* images by adapting the manipulation manually for each image in the dataset.

2. ***Average.*** As a baseline, we generate *average* images by averaging adjustment parameters across all training demonstrations and copying any selection regions and brush strokes (re-normalized to account for differences in image size) from the demonstration image closest in size to the target image[2].

3. ***Automatic.*** We generate *automatic* images using our content-adaptive macros without any manual correction using our correction interfaces.

4. ***Corrected.*** To factor out the effect of errors due to incorrect image labeling, we also compare against *corrected* images that we generate by manually correcting poor labeling from our automated labelers using our labeling correction panel. Note that we did not use the macro application panel to correct any other parameter adaptations.

Figure 5.12 shows these adaptations for the film noir manipulation. We do not show our *corrected* result because the image labels were correct and thus the *corrected* and *automatic* results are identical.

As described in Section 5.3.4, the regression technique we use for learning adjustment parameters requires 20 demonstrations to work robustly. Thus, we train our macros on a random subset of 20 images. In addition, to investigate how our macro results vary with the number of training demonstrations, we choose five of the manipulations (bag removal, contrast, eye makeup, film noir, and sunset enhancement) and compare results with random subsets of 1, 10, 20 and 30 training demonstrations. We use the average parameter values to generate our *automatic* and *corrected* results when we have only 1 or 10 training demonstrations.

We compare the four adaptation methods using an Amazon Mechanical Turk study in which participants rate the differences between the manipulated images. We chose to run a user-based evaluation because most standard image difference metrics such as mean squared error (MSE), are not perceptual measures. We also compute the absolute difference in parameter space between *ground-truth* and our macro results. Across 11 manipulations we find that 79 out of 89 parameters generated by our *automatic* and *corrected* methods are closer to the *ground-truth* parameters than those generated by the *average* method.

Finally, to get a sense for the overall quality of our macro results, we manually counted the number of successful and less successful *automatic* results for all 20 manipulations. With 20

[2]We designed this baseline as a relatively straightforward extension to existing macro systems such as Adobe Photoshop's *Actions*.

Figure 5.12: Three adaptation methods for the film noir manipulation. We do not show our *corrected* result, because the image labels were correct and therefore it is identical to the *automatic* result.

training demonstrations, our overall success rate is 82%, ranging from 95% (for lip gloss, skin tone, black & white, and lomo) to 60% (for eye makeup). Our eye makeup macro is less successful because the face labeler did not find any face in 15 of the 80 test images. We report the MSE and parameter difference results on the project page. Our project website[3] also includes success rates and sample macro results for all 20 manipulations.

5.6.2 Mechanical Turk Study Design

We used the Amazon Mechanical Turk to test how well the *automatic, average* and *corrected* images match the *ground-truth*. We simultaneously showed Mechanical Turk workers four images marked A, B, C, and D; we asked them to rate the differences between images B, C and D, and image A on a scale of 1 (”indistinguishable from image A”) to 5 (”very different from image A”). In our first study, we labeled the *ground-truth* as A and counterbalanced the ordering of the *ground-truth*, *average* or *automatic* images with respect to labels B, C, and D. In a second study, we replaced the *automatic* images with the *corrected* images. Each task was completed by 5 different workers, and we paid 3 cents per task.

Findings

There are four notable findings from our user-based evaluation:

1. Macro results are better than *average* results. The mean difference ratings for all seven manipulations indicate that, in general, our *automatic* images (mean: 2.2 range: 1.8-2.6) and *cor-*

[3] All supplemental materials are at http://vis.berkeley.edu/papers/macros/

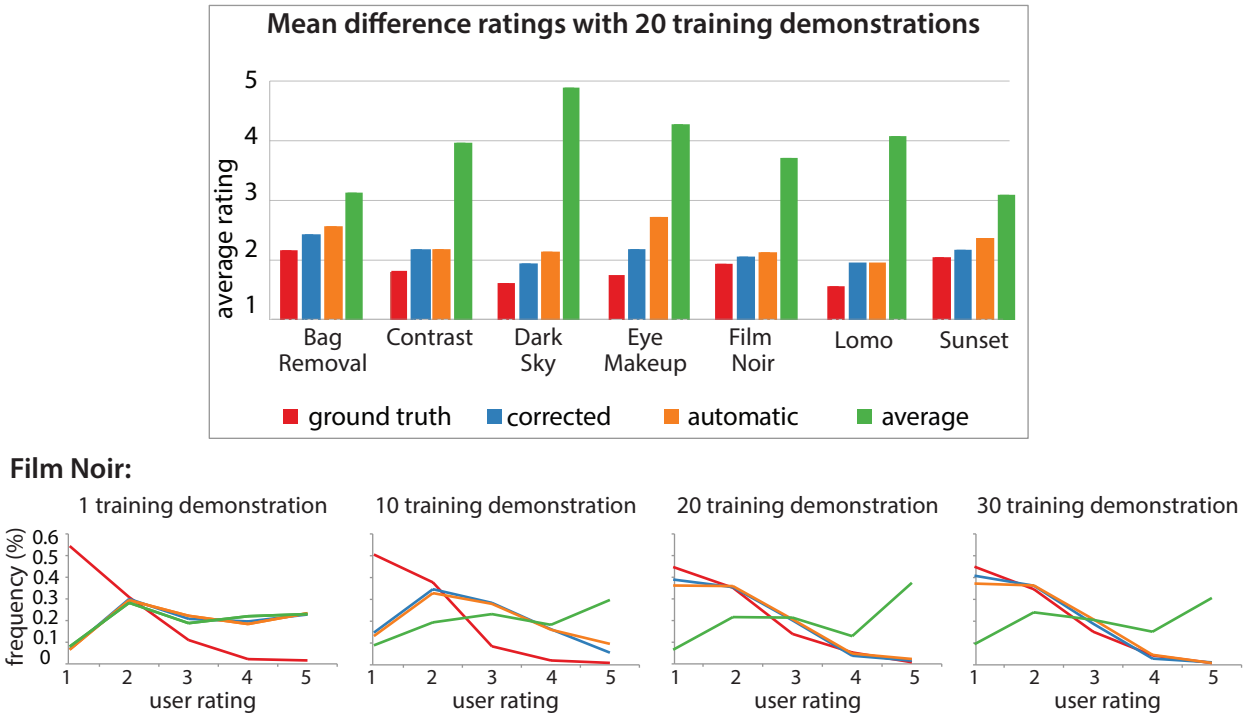


Figure 5.13: ***(Top)*** The mean difference ratings from the Mechanical Turk experiments, where a low difference rating indicates greater similarity to ground truth. With 20 training demonstrations, our automatic and corrected macro results consistently received lower difference ratings than the average images and were close to ground-truth. ***(Bottom)*** The distribution of difference ratings for the film noir manipulation as the number of training demonstrations increases. With one demonstration, the distributions for automatic and corrected closely match the distribution for average. As the number of training demonstrations increase, the distributions for automatic and corrected shift to match that of the ground truth, while the average distribution remains unchanged.

rected images (mean: 2.0 range: 1.8-2.3) match the *ground-truth* (mean: 1.7 range: 1.5-2) more closely than the *average* images (mean: 3.7 range: 2.9-4.7) (Figure 5.13 top). We find all of the differences in ratings across the four conditions to be significant ($p < 0.0001$) using Friedman's nonparameteric test for differences in ranks. Subsequent pairwise comparisons also find significant differences for all pairs of conditions ($p < 0.0001$).

2. Macro results improve with up to 20 training examples. The difference ratings for our *automatic* and *corrected* results decrease as the number of training demonstrations increases. Furthermore, the most significant decrease in the difference ratings occurs at 20 training demonstrations for the two manipulations that primarily involve adjustment parameters – contrast and film noir (Figure 5.13 bottom). For the other manipulations, our results are already noticeably better than the *average* images after 1 demonstration and show little improvement after 10 demonstra-

tions. This data suggests that 20 demonstrations are sufficient for learning adjustment parameters and that 10 demonstrations are enough to adapt most selection and brush stroke operations.

3. *Corrected* results are slightly better than *automatic* results. The automated labelers produced poor labels for 23% of faces and 31% of skies in our image datasets. However, the mean difference ratings for our *corrected* results are only slightly better (i.e., smaller) than the ratings for our fully *automatic* results; with 20 training demonstrations, the discrepancy between the *corrected* and *automatic* ratings ranges from 0 (for the contrast manipulation) to 0.53 (for eye makeup), with an average discrepancy of 0.18 across the six manipulations that required corrections (lomo did not require any corrections).

4. Macro results often indistinguishable from *ground-truth*. Because we include the *ground-truth* image as one of the images in the difference rating task, workers could rate our macro results as a closer match to *ground-truth*, than the *ground-truth* image itself. We count the number of times at least 3 out of 5 workers gave our result a rating less than or equal to their rating for the *ground-truth* image. In such cases it is likely that workers could not visually distinguish our images from *ground-truth*. We also count the number of images for which at least 3 out of 5 workers gave our result a rating greater than or equal to their rating for the average image. These images are the ones for which our approach performs poorly. With 20 training demonstrations, 47% of the automatic and 56% of the corrected images were rated better than or equal to *ground-truth* while just 6% of the *automatic* and 2% of the *corrected* were rated either no better or worse than *average*.

5.6.3 Macro Authoring Workflow

To evaluate the utility of our proposed macro authoring workflow, we conducted a small comparative lab study with three serious photographers who routinely use Photoshop to edit their images. We brought each participant in for a one hour session in which we first described our framework and then asked him to manipulate 5–6 images in Photoshop as if he was generating training demonstrations for our system (i.e., by performing the same sequence of manipulation steps for each image). We chose the lip gloss manipulation as our test case because it includes selections, brush strokes, and adjustment parameters (see Figure 5.8). Each participant performed the demonstrations under two different conditions: first, using Photoshop without our labeling and macro application panels, and then using Photoshop with our panels. Finally, we asked several questions to elicit feedback about our proposed workflow.

Feedback

All three participants agreed that the panels were a clear improvement over the no-panel workflow. In particular, there was consensus that the macro application panel made the demonstration process much easier by enforcing the correct sequence of steps and automatically applying each step to new images, even if some of those steps required corrections. Participants specifically mentioned the

Figure 5.14: Two car manipulations. Red boxes indicate cars found by Felzenswab et al.'s [2008] detector. (a) We demonstrate 10 tilt-shift manipulations and our framework successfully learns to blur the region above and below the car. (b) We demonstrate recoloring of 5 red cars and 5 green cars to blue. Our content-adaptive macro correctly recolors new target cars, without recoloring red/green elements that fall outside the car bounding box. But, it fails when red/green background elements, like grass, fall within the bounding box.

visualization of steps and the ability to edit intermediate steps as important benefits. In addition, they appreciated that training occurs incrementally every time they apply the macro to a new image and correct the results. The feedback about the per-step macro robustness scores was more mixed. While some found the MSE numbers a bit difficult to interpret, most agreed that it was useful to have the system indicate which steps were likely to require corrections.

To get some feedback on the potential limitations of our workflow, we asked how participants felt about performing the same sequence of steps for each image and whether they would be willing to perform 15–20 demonstrations to train a content-adaptive macro. In response, none of the participants were bothered by having to perform the same sequence of steps (especially given the macro application panel), and two of three respondents said they would be willing to perform 15–20 training demonstrations to use our system — the third respondent felt that the number of required demonstrations was reasonable but explained that he does not typically perform the same manipulations on many different images.

Overall, the feedback from this study suggests that our proposed workflow is useful and that our labeling correction and macro application panels make it significantly easier for users to author content-adaptive macros.

5.7 Extensions to Other Image Labelers

Although we have demonstrated our framework on common face, landscape and global manipulations, we have designed a general framework that can easily incorporate new image recognizers as they become available. To test this extensibility we have experimented with adding Felzenszwalb et al's [34] car detector to our image labeler. This detector identifies a bounding box for each

car and does not provide landmark correspondence points. We transfer two car-specific manipulations: a tilt-shift manipulation that leaves the car in focus while blurring its surroundings and a car recoloring manipulation. The tilt-shift manipulation (Figure 5.14a) performs very well for our target images because this manipulation only requires an approximate position and size for the car. It does not require a pixel-accurate boundary. The car recoloring manipulation (Figure 5.14b) performs well if the pixel-level features of the car are sufficiently discriminative. For example, if we train the macro to change red and green cars to blue it learns to recolors only pixels within the car bounding box and does not affect pixels outside the box. However, if there are red/green pixels within the bounding box that are not part of the car (e.g. grass in the third example) it recolors those pixels as well. More accurate landmark points on the boundary of the car would mitigate such problems. We leave it for future work to identify such landmark points when they are not directly provided by the labeler.

5.8 Next Steps

While our framework is able to adapt many different types of photo manipulation macros to new target images, we have observed a few limitations that we plan to address in future work.

Poor quality image labels. Our approach relies on high quality image labeling with consistent landmark points. While our visual feedback and correction interfaces allow users to manually fix incorrect labeling or landmarks, they also incur additional manual work. Moreover, many kinds of recognizers, including our outdoor scene recognizer, do not provide landmark points. In such cases, our approach of using the bounding box vertices as landmarks yields acceptable results for manipulations like sunset enhancement or tilt shift because they require coarse spatial information about the sky or car. But manipulations like car recoloring (Section 5.7) that require precise placement of selection regions or brush strokes are unlikely to transfer well using such landmarks.

Indirect dependencies. For some image manipulations the adjustment parameters depend on non-local image features. Our framework is only able to learn dependencies between adjustment parameters and image features of the active selection or brush region and its complement. For example, when applying the mustache macro to a light haired person, the result looks unnatural even though the location of the mustache is correct (see inset). In this case our framework is not able to learn the dependency between the color parameter of the mustache brush and the person's hair color because the hair is not the main element in the brush region or its complement. An interactive machine learning approach where users guide the algorithm by specifying relevant features or image locations containing those features could mitigate such problems. Such an interactive extension could also reduce the number of demonstrations necessary to learn the macro, accelerate the learning

process, and extend the number of dependencies the system could recognize.

Unaligned demonstrations. Our system requires that users perform all of the example demonstrations for a particular manipulation using roughly the same sequence of operations. While this operation alignment condition is satisfied for most demonstrations, some input images might require additional steps or the user might choose to demonstrate the steps in a different order. One approach may be to use sequence alignment or tree alignment techniques to automatically align operation sequences that differ significantly from one another. Using such alignment techniques, it may also be possible to combine demonstrations from multiple authors and thereby distribute the work of generating example demonstrations.

Chapter 6

Future Work

We have presented several systems that facilitate each of the three stages of the learning process by 1) automatically generating step-by-step tutorials by demonstration, 2) providing new interfaces to help learners compare multiple photo manipulation procedures, and 3) automatically transferring photo manipulation procedures to new images. We believe there are many possible directions for future work that build on the ideas presented in this thesis.

6.1 Teaching versus Automating Procedures

Chapter 3 describes a system to automatically generate photo manipulation tutorials by demonstration. Tutorials teach learners all the necessary steps for the manipulation, but the learners still have to perform the operations themselves. In contrast, the content-adaptive macros presented in Chapter 5 are fully automated. The computer executes and adapts all the operations automatically, but the user also does not learn the procedure. Tutorials and macros are both useful depending on the goals of the user and the context in which the tutorial/macro is used.

But there is also a tension between teaching someone a new procedure versus simply automating it. If the transfer of a procedure can be fully automated in a content-adaptive manner and it always produces successful transfer results, teaching the procedure to users may not be necessary. The software could simply provide the one-button-click macro for this procedure and the user would never have to make corrections. However, our content-adaptive macros rely on image recognition and machine learning techniques. These techniques are not fully robust or error-free, and our framework generates undesirable results for a non-negligible number of transfers. While advances in computer vision and machine learning will further improve our results, it is likely that there will always be cases for which our content-adaptive macros produce less successful transfers.

So one research question that remains is, can we build a semi-automated system that automatically identifies the steps it can transfer robustly, and provides correction interfaces or allows the user to perform the steps that are less robust. The correction interface presented in Section 5.4 offers an initial approach by allowing users to quickly correct any particular operation in the macro and then automatically execute the remaining steps of the procedure. However, this interface

does not provide explanations for any of the steps and assumes the user knows how to make the necessary corrections. One direction for future work is to augment the correction panels with instructions on how to correctly perform each step. These additional explanations could help users detect and correct macro errors. This direction would also require studying the usefulness of such semi-automatic systems and the effect that they have on the learning process. One would need to compare a user's performance when applying a photo manipulation to new target images with such semi-automated approaches to fully automatic macros and manual tutorials.

6.2 Instructions for Physical Procedures

Procedural tasks are very common and people share instructions for procedures within lots of different domains (Table 1.1). However, many of the procedural tasks are physical tasks that occur in the real world such as cooking, assembling furniture or driving. It is still very difficult to automatically generate effective instructions for such tasks. While it is straightforward for people to capture a video of someone performing a physical task like cooking, a major challenge in generating instructions lies in automatically recognizing and segmenting the physical motion into operations. Current computer vision approaches [31, 85, 32] can only recognize a very limited set of these egocentric activities and often assume that the objects and the environment remain constant throughout the task demonstration. These constraints are not fulfilled for most physical tasks.

Furthermore, there is very little research on what type of tutorial format is most effective for such tasks. While several studies [83, 42] suggest that static tutorials are better than video tutorials for learning software procedures, some complicated physical tasks such as dancing may require video instructions to show a 3D movement or interaction. However, video-based instructions force users to work at the pace of the video and may thereof cause learners to make more mistakes. A challenge is to automatically segment such videos into steps and provide tools to help users follow and navigate these video tutorials. If on the other hand the goal is to make a static tutorial a related challenge is to choose the most representative frames from the video recording to show for each step in the tutorial.

An important future research direction is to classify these physical procedural tasks according to the complexity of their 3D interactions, and study what tutorial type (static versus video) is most effective for each category. Furthermore, developing tools to help tutorial authors create static and video tutorials for physical tasks could significantly facilitate their production and improve their quality. In addition to using computer vision techniques to recognize activities in the scene, one could use cues such as the video narration or the speed of the motion in the video to segment the procedure into steps. Finally, it is also important to explore ways to annotate the motion in the video (equivalent to the annotation in Section 3.5.2 for static tutorials) so as to highlight the task the user must perform.

6.3 Automatically Parsing Instructions

One of the recent trends in graphics has been to exploit big datasets. For example, sites such as `flickr.com` contain tens of thousands of images. Researchers have used these image databases for automatic scene completion [44], photo tourism [97] and many other image manipulation applications [64, 95, 96, 3, 93, 24]. More recently, sites like Google Trimble 3D Warehouse have put very large collections of 3D models online. Similarly, in graphics, we have used these databases to develop many new applications for part-based modeling [19, 53, 58] as well as parametrizing and clustering similar shapes [82]. Not surprisingly, having access to large collections enables new research. However, building such collections can be difficult.

As we saw in Chapter 1, people share hundreds of thousands of instructions for many different tasks: recipes for cooking, assembly instructions for building furniture and sewing patterns for clothing. These instructions are designed by humans and for humans. The tutorials are often written in natural text and use inconsistent formats. As a consequence, computers do not know how to interpret them. However, an important research direction is to develop algorithms that can automatically parse these documents, identify the commands, and apply them. Such algorithms would enable computers to automatically follow instructions, and allow researchers to exploit online collections of tutorials to build large databases of structured data. As with images and 3D models, analyzing these new collections could lead to new advances in graphics. For example, it could allow us to synthesize new elements within these domains (e.g. food, furniture, clothes), generate instructions from an end result (rather than a demonstration), or provide feedback to a user who is following the instructions.

We have recently started pursuing this goal by interpreting sewing patterns to build a large collection of 3D garment models [11]. Sewing patterns are composed of a set of panels and implicitly describe the cutting, folding and stitching operations that a human tailor must perform to physically fabricate clothing (Figure 6.1a). Thousands of these patterns are now available online on sites like `burdastyle.com`. We have developed an algorithm that processes the patterns to extract the panels, and automatically applies the assembly operations to generate a 3D model of the garment (Figure 6.1b). The challenge is that the assembly operations are not explicitly stated. Human tailors apply their domain experience to correctly interpret them. We have worked with tailors to identify the rules they use, and have developed a parser that applies these rules automatically.

As future work, we will build a large database of sewing pattern/3D model pairs. We can apply data-driven techniques to this collection to pursue our goals of synthesizing new garments, generating sewing patterns from 3D models, and providing users with feedback when they are following a pattern. The challenge in generating new garments is to combine the panels from different patterns so as to create a valid garment. Parameterizing the space of garments could allow us to achieve this goal. In animation and game studios, modelers often create surface models of garments. They may want to fabricate the corresponding physical garment for commercial purposes. However they cannot easily convert the 3D models into physical garments because they do not have the underlying patterns. We plan to build a mapping between the 3D model to a set of 2D panels while preserving common panel design characteristics (e.g. correct usage of darts). Finally, we could help users create the physical garment. Sewing is not an easy skill and it can

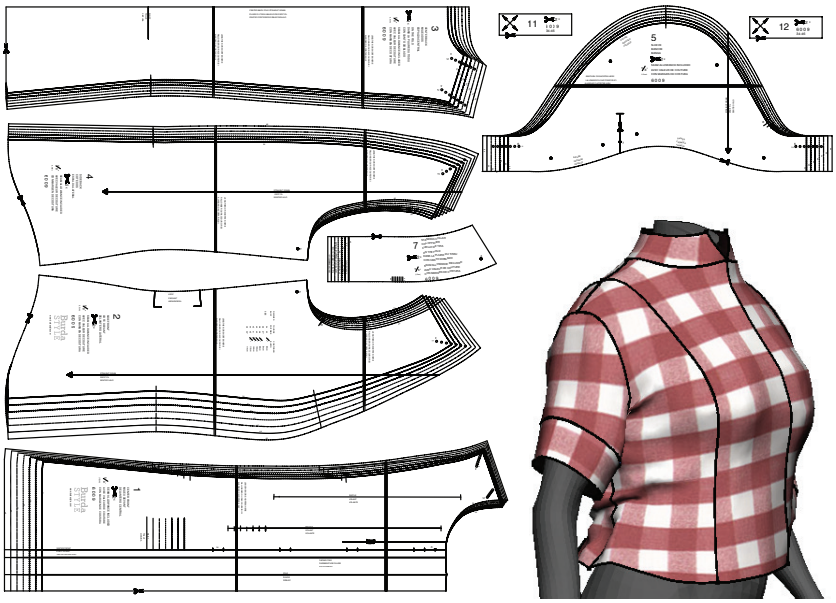


Figure 6.1: We parse sewing patterns (a) to generate the corresponding 3D models (b).

be difficult for a person to follow the patterns. We could physically extend the sewing machine to provide step-by-step instructions and feedback when the person makes mistakes. We could possibly even go so far as correct the mistake. For example, when the person is deviating from the seam line, we could develop a motorized mechanism that readjusts the needle position so that it snaps back in place. Therefore, by instantiating domain-specific design principles in algorithms and interfaces, we ultimately hope to create physical tools that are smarter and safer.

6.4 Beyond Procedural Instructions

Over the last couple of years, people have started sharing many types of instructions online. In particular, sites like the Khan Academy or other platforms for Massive Open Online Courses (MOOCs) offer thousands of online courses and millions of students take them. These sites have fundamentally changed the way people use instructional material by providing free access to a very large collection of high quality video instructions. A well known benefit of MOOCs is that they make education accessible all around the world.

These online courses are designed to teach all forms of knowledge. They combine procedure knowledge with other types of knowledge such as factual or conceptual knowledge. Similar to procedural step-by-step tutorials, these video instructions are very time-consuming to produce. Authors are forced to edit them at the frame level rather than based on their semantic content. Furthermore, it is difficult to make a class engaging for an online viewer. Most videos simply

show a teacher in front of a whiteboard, but the teacher's whiteboard writing may be difficult to read or not use space effectively.

An important direction for future work is to study how to create more effective online instructions beyond procedural tasks. In particular, a challenge is to help teachers author video instructions like the ones on the MOOC platforms. One could apply the same approach that we have used for step-by-step procedural tutorials to online courses. This approach would require studying what characteristics make online classes effective and then developing new tools and techniques to help people create video instructions that facilitate the learning process. For example, one could build a special-purpose video editing tools that allow teachers to change the layout of the notes, add highlights, and add or remove content from the video.

Chapter 7

Conclusion

In this work, we have presented three different systems that are each designed to help with a different stage in acquiring procedural knowledge, more specifically the knowledge of how to manipulate an image.

In Chapter 3, we have presented a demonstration-based system for automatically generating succinct step-by-step visual tutorials that facilitate the cognitive stage of the learning process. An author first demonstrates the manipulation using an instrumented version of GIMP that records all changes in interface and application state. From the example recording, our system automatically generates tutorials that illustrate the manipulation using images, text, and annotations. A key feature of our approach is that we combine application-level information about the operations conducted by the user with image recognition techniques (recognition of facial features and outdoor scene structures in our implementation) that label semantically meaningful elements in the photographs. We leverage the automated labeling to generate more precise text descriptions of many of the steps in the tutorials. One limitation of our tutorials is that they cannot explain why users must perform each operation. Without explanations, it can be more difficult for users to transfer the operations to their images or different tasks. Nevertheless, a user study comparing our automatically generated tutorials to hand-designed tutorials and screencapture video recordings with voiceover explanations, finds that users are 20–44% faster and make 60–95% fewer errors using our tutorials. Moreover, our tutorials are generated in a fraction of the time required to manually design a tutorial. Thus, our automatically generated tutorials could also serve as a first step that authors could supplement with further explanations.

In Chapter 4, we have presented a new browser interface that allows users to navigate, explore and compare the command-level structure of a large collection of image manipulation tutorials. This interface helps learners better grasp the underlying structure of photo manipulation procedures and thereby facilitates the associative stage of learning. To build this interface, we first collect 2500 Photoshop tutorials that we download from the Web. We then apply supervised machine learning to identify the command sequence described in each tutorial from our collection. We also provide algorithmic tools to compare the command sequences in our database as well as identify

common patterns of command sequences within the tutorial collection. Finally, we have presented a new browser interface with three views; a (1) Faceted Browser View for organizing and filtering the tutorials by their commands, a (2) Tutorial View for examining individual tutorials and an (3) Alignment View for comparing the similarities and differences in the command structure between a subset of tutorials. User feedback suggests that our interface is easy to understand and use, and that users find command-level browsing to be useful for exploring large tutorial collections. They strongly preferred to explore tutorial collections with our browser over keyword search.

In Chapter 5, we have presented a framework for generating content-adaptive macros that can transfer complex photo manipulations to new target images. While our framework does not facilitate the learning process for a person, it successfully teaches photo manipulation procedures to computers and thereby removes some of the tedium of having to manually execute the same procedure over and over again. To create a content-adaptive macro, we make use of multiple training demonstrations. Specifically, we use automated image labeling and machine learning techniques to learn the dependencies between image features and the parameters of each selection, brush stroke and image processing operation in the macro. Although our approach is limited to learning manipulations where there is a direct dependency between image features and operation parameters, we show that our framework is able to learn a large class of the most commonly-used manipulations using as few as 20 training demonstrations. Our framework also provides interactive controls to help macro authors and users generate training demonstrations and correct errors due to incorrect labeling or poor parameter estimation. Using our interactive panels, applying a photo manipulation procedure to new images becomes effortless and does not require the user to perform repetitive actions. Finally, our evaluation shows that our framework can produce effective content-adaptive macros for a wide range of image manipulations and that our macro feedback and correction interfaces are both effective and practical in the context of real image editing workflows.

Together these tools allow users to automatically create high quality step-by-step tutorials by demonstration, provide an interface for exploring and comparing large collections of online tutorials, and facilitate repetitive procedures by automating the transfer of photo manipulations. As more and more instructional material appears online, providing such tools for learning, comparing and automating procedures will be crucial to help people work efficiently with software tools. Furthermore, we identify several research challenges for future work. As machine learning algorithms become more robust and many procedures can be automated, we believe that it will be essential to explore the trade-offs between tutorials and macros. We also discuss how researchers may leverage the large online collections of tutorials to automatically build databases of structured data in different domains. Finally, another open research challenge is to build tutorial authoring tools for physical procedures as well as other non-procedural tasks. We believe that computerized tools can play a large role in facilitating the learning process for many domains beyond photo manipulation procedures.

Bibliography

[1] J. A. Adams. “A closed-loop theory of motor learning.” In: *Journal of motor behavior* (1971).

[2] J. A. Adams. “Historical review and appraisal of research on the learning, retention, and transfer of human motor skills.” In: *Psychological Bulletin* 101.1 (1987), p. 41.

[3] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. “Building rome in a day”. In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 72–79.

[4] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, and B. Tversky. “Designing effective step-by-step assembly instructions”. In: *Proc. SIGGRAPH* (2003), pp. 828–837.

[5] A. Amini, R. Curwen, and J. Gore. “Snakes and splines for tracking non-rigid heart motion”. In: *ECCV’96* (1996), pp. 249–261.

[6] J. R. Anderson. “Skill acquisition: compilation of weak-method problem situations.” In: *Psychological review* 94.2 (1987), p. 192.

[7] S. Bae, S. Paris, and F. Durand. “Two-scale tone management for photographic look”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 25.3 (2006), pp. 637–645.

[8] F. F. C. Bartlett and F. C. Bartlett. *Remembering: A study in experimental and social psychology*. Vol. 14. Cambridge University Press, 1995.

[9] P. Benner. “From novice to expert”. In: *AJN The American Journal of Nursing* 82.3 (1982), pp. 402–407.

[10] L. Bergman, V. Castelli, T. Lau, and D. Oblinger. “DocWizards: A system for authoring follow-me documentation wizards”. In: *Proc. UIST*. 2005, pp. 191–200.

[11] F. Berthouzoz, A. Garg, D. M. Kaufman, E. Grinspun, and M. Agrawala. “Parsing sewing patterns into 3d garments”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 85.

[12] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster. “Program understanding and the concept assignment problem”. In: *Communications of the ACM* 37.5 (1994), pp. 72–82.

[13] D. Bitouk, N. Kumar, S. Dhillon, P. N. Belhumeur, and S. K. Nayar. “Face swapping: automatically replacing faces in photographs”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* (2008).

[14] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. "Automation and customization of rendered web pages". In: *Proc. UIST*. 2005, pp. 163–172.

[15] H. Booher. "Relative comprehensibility of pictoral information and printed words in proceduralized instructions". In: *Human Factors*. Vol. 17. 3. 1975, pp. 266–277.

[16] S. Carberry. "Techniques for plan recognition". In: *User Modeling and User-Adapted Interaction* 11.1-2 (2001), pp. 31–48.

[17] J. M. Carroll. *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0-262-0316390.

[18] E. Charniak and R. P. Goldman. "A bayesian model of plan recognition". In: *Artificial Intelligence* 64.1 (1993), pp. 53–79.

[19] S. Chaudhuri and V. Koltun. "Data-driven suggestions for creativity support in 3d modeling". In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 6. ACM. 2010, p. 183.

[20] P. Chilana, A. J. Ko, and J. O. Wobbrock. "Lemonaid: selection-based crowdsourced contextual help for web applications". In: *Proceedings of CHI*. 2012, pp. 1549–1558.

[21] A. Cypher and D. Halbert. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.

[22] A. Dewdney. "A potpourri of programmed prose and prosody". In: *Scientific American* (1989).

[23] M. Dixon and J. Fogarty. "Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure". In: *Proceedings of CHI*. ACM. 2010, pp. 1525–1534.

[24] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. "What makes paris look like paris?" In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 101.

[25] S. E. Dreyfus and H. L. Dreyfus. *A five-stage model of the mental activities involved in directed skill acquisition*. Tech. rep. DTIC Document, 1980.

[26] I. Drori, D. Cohen-Or, and H. Yeshurun. "Example-based style synthesis". In: *In Proceedings of Computer Vision and Pattern Recognition (Madison,USA*. 2003, pp. 143–150.

[27] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. "Least angle regression". In: *Annals of statistics* (2004), pp. 407–451.

[28] A. Efros and W. Freeman. "Image quilting for texture synthesis and transfer". In: *Proc. SIGGRAPH*. 2001, pp. 341–346.

[29] E. Eisemann and F. Durand. "Flash photography enhancement via intrinsic relighting". In: *ACM Transactions on Graphics (TOG)* 23.3 (2004), pp. 673–678.

[30] M. Ekstrand, W. Li, T. Grossman, J. Matejka, and G. Fitzmaurice. "Searching for software learning resources using application context". In: *Proceedings of UIST*. ACM. 2011, pp. 195–204.

[31] A. Fathi, A. Farhadi, and J. M. Rehg. “Understanding egocentric activities”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 407–414.

[32] A. Fathi, Y. Li, and J. M. Rehg. “Learning to recognize daily actions using gaze”. In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 314–327.

[33] S. K. Feiner. “A grid-based approach to automating display layout”. In: *Proc. Graphics interface*. 1988, pp. 192–197.

[34] P. Felzenszwalb, D. McAllester, and D. Ramanan. “A discriminatively trained, multiscale, deformable part model”. In: *Proc. CVPR*. 2008.

[35] P. M. Fitts and M. I. Posner. “Human performance.” In: (1967).

[36] A. Fourney, B. Lafreniere, R. Mann, and M. Terry. “”Then click ’OK!’” extracting references to interface elements in online documentation”. In: *Proceedings of CHI*. 2012, pp. 35–38.

[37] A. M. Gentile. “A working model of skill acquisition with application to teaching”. In: *Quest* 17.1 (1972), pp. 3–23.

[38] J. Good. *How many photos have ever been taken?* `http://blog.1000memories.com/94-number-of-photos-ever-taken-digital-and-analog-in-shoebox`. 2013.

[39] Google, Inc. *Google Custom Search Engine*. `https://www.google.com/cse`. 2013.

[40] T. Grossman, G. Fitzmaurice, and R. Attar. “A survey of software learnability: metrics, methodologies and guidelines”. In: *Proceedings of the 27th international conference on Human factors in computing systems*. Boston, MA, USA: ACM, 2009, pp. 649–658. ISBN: 978-1-60558-246-7. DOI: `10.1145/1518701.1518803`. URL: `http://doi.acm.org/10.1145/1518701.1518803`.

[41] D. Guo and T. Sim. “Digital face makeup by example”. In: *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 0 (2009), pp. 73–79.

[42] S. Harrison. “A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface”. In: *Proc. CHI*. 1995, pp. 82–89.

[43] S. Hasinoff, M. Józwiak, F. Durand, and W. Freeman. “Search-and-replace editing for personal photo collections”. In: *Proc. ICCP*. 2. 2010, p. 8.

[44] J. Hays and A. A. Efros. “Scene completion using millions of photographs”. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 4.

[45] J Heiser, D Phan, M Agrawala, B Tversky, and P Hanrahan. “Identification and validation of cognitive design principles for automated generation of assembly instructions”. In: *Proc. AVI* (2004), pp. 311–319.

[46] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. “Image analogies”. In: *Proc. SIGGRAPH*. 2001, pp. 327–340.

[47] A. Hertzmann, N. Oliver, B. Curless, and S. Seitz. “Curve analogies”. In: *Proc. Eurographics Workshop on Rendering*. 2002, pp. 233–246.

[48] D. Hoiem, A. Efros, and M. Hebert. “Geometric context from a single image”. In: *Proc. ICCV*. 2005, pp. 654–661.

[49] B. Huggins. *Photoshop: Retouching Cookbook for Digital Photographers*. O’Reilly, 2005.

[50] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. “Adaptive grid-based document layout”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 22.3 (2003), pp. 838–847.

[51] M. Jones and J. Rehg. “Statistical color models with application to skin detection”. In: *International Journal of Computer Vision* 46.1 (2002), pp. 81–96.

[52] R. Kalnins, L. Markosian, B. Meier, M. Kowalski, J. Lee, P. Davidson, M. Webb, J. Hughes, and A. Finkelstein. “WYSIWYG NPR: Drawing strokes directly on 3D models”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 755–762.

[53] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun. “A probabilistic model for component-based shape synthesis”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 55.

[54] S. Kang, A. Kapoor, and D. Lischinski. “Personalization of image enhancement”. In: *Proc. CVPR*. 2010.

[55] M. Kass, A. Witkin, and D. Terzopoulos. “Snakes: Active contour models”. In: *International journal of computer vision* 1.4 (1988), pp. 321–331.

[56] S. Kelby. *The Adobe Photoshop CS3 book for digital photographers*. Voices That Matter, 2007.

[57] C. Kelleher and R. Pausch. “Stencils-based tutorials: design and evaluation”. In: *Proc. CHI*. 2005, pp. 541–550.

[58] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser. “Learning part-based templates from large collections of 3d shapes”. In: (2013).

[59] K. Knabe. “Apple guide: A case study in user-aided design of online help”. In: *Proc. CHI*. 1995, pp. 286–287.

[60] N. Kong, T. Grossman, B. Hartmann, G. Fitzmaurice, and M. Agrawala. “Delta: a tool for representing and comparing workflows”. In: *Proceedings of CHI*. 2012, pp. 1027–1036.

[61] D. S. Kosbie and B. A. Myers. “A system-wide macro facility based on aggregate events: A proposal”. In: *Watch what I do: Programming by demonstration*. MIT Press, 1993, pp. 433–444.

[62] D. Krishnan and R. Fergus. “Dark flash photography”. In: *ACM Transactions on Graphics (TOG)* 28.3 (2009), p. 96.

[63] D. Kurlander and S. Feiner. “A history-based macro by example system”. In: *Proc, UIST*. 1992, pp. 99–106.

[64] J.-F. Lalonde, D. Hoiem, A. A. Efros, C. Rother, J. Winn, and A. Criminisi. “Photo clip art”. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 3.

[65] G Laput, E Adar, M Dontcheva, and W Li. “Tutorial-based interfaces for cloud-enabled applications”. In: *Proceedings of UIST*. ACM. 2012, pp. 113–122.

[66] T. Lau, L. Bergman, V. Castelli, and D. Oblinger. "Sheepdog: Learning procedures for technical support". In: *Proc. IUI*. 2004, pp. 109–116.

[67] T. Lau, C. Drews, and J. Nichols. "Interpreting written how-to instructions". In: *Proceedings of IJCAI*. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1433–1438. URL: http://dl.acm.org/citation.cfm?id=1661445.1661675.

[68] D. Lewis. "Naive (Bayes) at forty: The independence assumption in information retrieval". In: *Machine Learning: ECML-98* (1998), pp. 4–15.

[69] H Lieberman. *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2001.

[70] H. Lieberman. "Mondrian: A teachable graphical editor". In: *Watch what I do: Programming by demonstration*. 1993, pp. 341–358.

[71] G. Little, T. Lau, A. Cypher, J. Lin, E. Haber, and E. Kandogan. "Koala: Capture, share, automate, personalize business processes on the web". In: *Proc. CHI*. 2007, pp. 943–946.

[72] Z. Liu, Y. Shan, and Z. Zhang. "Expressive expression mapping with ratio images". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, p. 276.

[73] C. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[74] J. Matejka, T. Grossman, and G. Fitzmaurice. "Ip-qat: in-product questions, answers, & tips". In: *Proceedings of UIST*. Santa Barbara, California, USA: ACM, 2011, pp. 175–184. ISBN: 978-1-4503-0716-1. DOI: 10.1145/2047196.2047218. URL: http://doi.acm.org/10.1145/2047196.2047218.

[75] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. "Communitycommands: command recommendations for software applications". In: *Proceedings of UIST*. Victoria, BC, Canada: ACM, 2009, pp. 193–202. ISBN: 978-1-60558-745-5. DOI: 10.1145/1622176.1622214. URL: http://doi.acm.org/10.1145/1622176.1622214.

[76] C. Meng, M. Yasue, A. Imamiya, and X. Mao. "Visualizing histories for selective undo and redo". In: 1998, pp. 459–464.

[77] F. Modugno and B. Myers. "Pursuit: Graphically representing programs in a demonstrational visual shell". In: *Proc. CHI*. 1994, pp. 455–456.

[78] T. Nakamura and T. Igarashi. "An application-independent system for visualizing user operation history". In: *Proc. UIST*. 2008, pp. 23–32.

[79] A. Newell and P. S. Rosenbloom. "Mechanisms of skill acquisition and the law of practice". In: *Cognitive skills and their acquisition* (1981), pp. 1–55.

[80] M. Nguyen, J. Lalonde, A. Efros, and F. De la Torre. "Image-based shaving". In: *Computer Graphics Forum*. Vol. 27. 2. 2008, pp. 627–635.

[81] L. R. Novick and D. L. Morse. "Folding a fish, making a mushroom: The role of diagrams in executing assembly procedures". In: *Memory and Cognition* 28.7 (2000), pp. 1242–56.

[82] M. Ovsjanikov, W. Li, L. Guibas, and N. J. Mitra. "Exploration of continuous variability in collections of 3d shapes". In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 33.

[83] S. Palmiter and J. Elkerton. "An evaluation of animated demonstrations of learning computer-based tasks". In: *Proc. CHI*. 1991, pp. 257–263.

[84] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama. "Digital photography with flash and no-flash image pairs". In: *ACM Transactions on Graphics (TOG)* 23.3 (2004), pp. 664–672.

[85] H. Pirsiavash and D. Ramanan. "Detecting activities of daily living in first-person camera views". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2847–2854.

[86] S. Pongnumkul, M. Dontcheva, W. Li, J. Wang, L. Bourdev, S. Avidan, and M. F. Cohen. "Pause-and-play: automatically linking screencast video tutorials with applications". In: *Proceedings of UIST*. Santa Barbara, California, USA: ACM, 2011, pp. 135–144. ISBN: 978-1-4503-0716-1. DOI: 10.1145/2047196.2047213. URL: http://doi.acm.org/10.1145/2047196.2047213.

[87] R. W. Proctor and A. Dutta. *Skill acquisition and human performance.* Sage Publications, Inc, 1995.

[88] V. Ramesh, C. Hsu, M. Agrawala, and B. Hartmann. "Showmehow: translating user interface instructions between applications". In: *Proceedings of UIST*. ACM. 2011, pp. 127–134.

[89] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley. "Color transfer between images". In: *IEEE Computer Graphics and Applications* (2001), pp. 34–41.

[90] M. Rettig. "Nobody reads documentation". In: *Communications of the ACM* 34.7 (July 1991), 19–24. ISSN: 0001-0782. DOI: 10.1145/105783.105788. URL: http://doi.acm.org/10.1145/105783.105788.

[91] R. A. Schmidt, D. E. Young, S. Swinnen, and D. C. Shapiro. "Summary knowledge of results for skill acquisition: support for the guidance hypothesis." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 15.2 (1989), p. 352.

[92] D. Schwarz. "Current research in concatenative sound synthesis". In: *Proc. of ICMC*. 2005, pp. 9–12.

[93] A. Shrivastava, T. Malisiewicz, A. Gupta, and A. A. Efros. "Data-driven visual similarity for cross-domain image matching". In: *ACM Transactions on Graphics (TOG)*. Vol. 30. 6. ACM. 2011, p. 154.

[94] S. Simhon and G. Dudek. "Curve Synthesis from Learned Refinement Models". In: *Eurographics* (2003).

[95] I. Simon, N. Snavely, and S. M. Seitz. "Scene summarization for online image collections". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1–8.

[96] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. "Finding paths through the world's photos". In: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, p. 15.

[97] N. Snavely, S. M. Seitz, and R. Szeliski. "Photo tourism: exploring photo collections in 3d". In: *ACM transactions on graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 835–846.

[98] S. Su. "Visualizing, editing, and inferring structure in 2D graphics". In: *UIST 2007 Doctoral Symposium*. 2007.

[99] M. Terry, M. Kay, B. V. Vugt, B. Slack, and T. Park. "Ingimp: Introducing instrumentation to an end-user open source application". In: *Proc. CHI*. 2008, pp. 607–616.

[100] P. Tutorials. *How to change eye color*. `http://fyeahtutorial.tumblr.com/post/14216130212/how-to-change-eye-color`. 2013.

[101] K. VanLehn. "Cognitive skill acquisition". In: *Annual review of psychology* 47.1 (1996), pp. 513–539.

[102] L. Varis. *Skin*. Wiley Publishing, 2006.

[103] S. G. Woods, A. E. Quilici, and Q. Yang. "Program understanding and ai plan recognition". In: *Constraint-Based Design Recovery for Software Reengineering*. Springer, 1998, pp. 117–132.

[104] T. Yeh, T. Chang, and R. Miller. "Sikuli: using gui screenshots for search and automation". In: *Proceedings of UIST*. ACM. 2009, pp. 183–192.

[105] Y. Zhou, L. Gu, and H. Zhang. "Bayesian tangent shape model: Estimating shape and pose parameters via bayesian inference". In: *Proc. CVPR*. 2003, pp. 109–116.