Inverse Image Editing: Recovering a Semantic Editing History from a Before-and-After Image Pair

Shi-Min Hu[1] Kun Xu[1] Li-Qian Ma[1] Bin Liu[1] Bi-Ye Jiang[1] Jue Wang[2]
[1]TNList, Tsinghua University, Beijing [2]Adobe Research

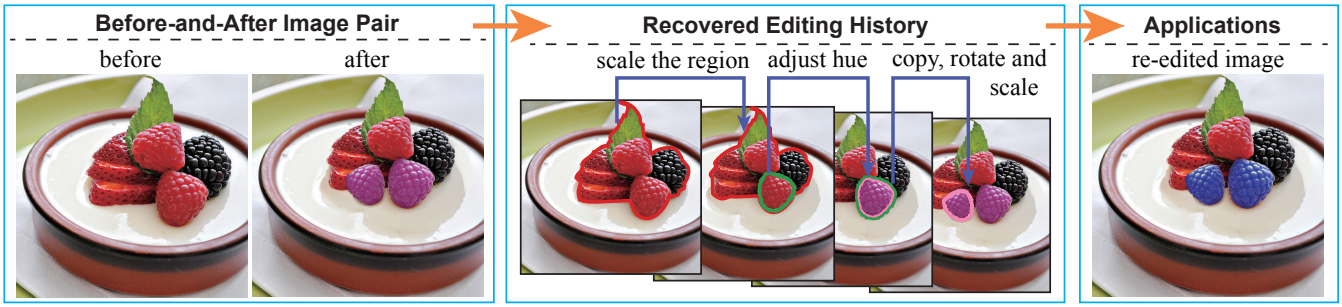


Figure 1: *Given a source image and an edited copy (left), our system automatically recovers a semantic editing history (middle), which can be used for various applications, such as re-editing (right). In this case, the second editing step of the recovered history, involving hue modification, is altered to change the berries to a difference color. Image courtesy of Andrea Lein.*

Abstract

We study the problem of *inverse image editing*, which recovers a semantically-meaningful editing history from a source image and an edited copy. Our approach supports a wide range of commonly-used editing operations such as cropping, object insertion and removal, linear and non-linear color transformations, and spatially-varying adjustment brushes. Given an input image pair, we first apply a dense correspondence method between them to match edited image regions with their sources. For each edited region, we determine geometric and semantic appearance operations that have been applied. Finally, we compute an optimal editing path from the region-level editing operations, based on predefined semantic constraints. The recovered history can be used in various applications such as image re-editing, edit transfer, and image revision control. A user study suggests that the editing histories generated from our system are semantically comparable to the ones generated by artists.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors; I.4.9 [Image Processing and Computer Vision]: Applications

Keywords: image editing history, inverse image editing, history recovery, region matching

Links: DL PDF WEB

1 Introduction

In image editing, a series of operations are often performed to accomplish an editing task. For instance, the user may first select an object in the image, apply geometric transforms to adjust its shape and position, and then apply various color adjustments to enhance its appearance. This process is repeated if multiple image regions need to be touched up to achieve an editing goal, resulting in a long and sometimes complicated editing history.

Having a clean and complete editing history available is required in many graphics and file management applications, such as image editing revision control [Chen et al. 2011], automatic tutorial generation [Grabler et al. 2009] and editing visualization [Heer et al. 2008]. Retention of an editing history also leads to new possibilities in image editing, such as: adjusting the history to produce results with different variations from the source image; transferring the history to another image, etc. These editing goals are much harder to achieve without having the editing history.

Unfortunately, although existing software provides powerful editing tools, there is no universal, efficient solution to encode, store, transmit and re-use an editing history. For instance, Adobe Photoshop [1] only allows a *partial* history to be saved in either the image file or the command log file. It is often the case that a *complete* history is not available after the editing task is accomplished. Furthermore, image editing is a trial and error process especially for amateurs. It can easily take dozens of operations to explore different ideas, fix errors and fine tweak parameters before producing a desired output. As a result, the raw editing history produced by a novice user is often long, redundant, less understandable, and may not be directly applicable in some applications that requires a *clean* history.

In this work, we study the problem of recovering a *clean* and *semantically meaningful* editing history given a source image and an edited version, which we call *inverse image editing*. This task involves several technical challenges. Firstly, we need to discover which objects or regions have been edited. Secondly, we need to determine how each object or region has been edited. Finally, a semantically meaningful editing path needs to be generated from

[1]http://www.photoshop.com/

recovered local editing operations. Our system provides a set of solutions to these problems. In particular, we improve the state-of-the-art region matching methods to handle large appearance differences between an original object and its edited version (see Sec. 4). We propose new methods to recover semantic appearance operators for each edited region (see Sec. 5), and then merge them to form a meaningful editing path under predefined semantic constraints (see Sec. 6). We also demonstrate how the recovered history can be applied to various applications (see Sec. 7).

It is impossible in practice to recover all editing operations that may have been applied to an image. Our system supports a set of commonly used linear and non-linear geometric and color adjustments, and arbitrary combinations of them. Specifically, each edit step may contain (but not necessarily) the following operations: (1) select an object (region); (2) apply a geometric transform that may involve scaling, rotation, translation, and flipping; and (3) apply per-region color adjustments that may involve brightness, exposure, hue, saturation, and tone adjustments, with a *spatially-varying* per-pixel strength map for simulating local paint brushes. These operators, when stacked together, provide vast editing possibilities, and cover many popular adjustment tools in modern image editing software.

To evaluate the proposed system, we construct a test dataset that contains before-and-after image pairs, along with the original editing steps performed by artists. A user study on this dataset revealed that the editing histories generated by our system are in general comparable to the original ones in terms of semantic meaningfulness (see Sec. 8).

2 Related Work

Region matching between images. A core component of our system is to build correspondences between regions in the original and edited images. Sparse matching methods such as SIFT matching [Lowe 2004] are not applicable, as dense correspondences are required in our application. Conventional optical flow methods [Brox et al. 2009; Zimmer et al. 2011] cannot be applied either, since an edited region may undergo large appearance transformations that violate the apparent motion assumption of optical flow. The recent SIFT flow method [Liu et al. 2008] and the dual bootstrap method [Yang et al. 2007] combine sparse features with dense matching through a seed-and-grow scheme, but they still have difficulty in matching textureless objects where features are hard to extract.

Recently, a family of patch-based fast image correspondence methods have been proposed. PatchMatch [Barnes et al. 2009] is a randomized algorithm for finding approximate nearest neighbors between image patches; it has been further extended to include geometric and color transformations during matching [Barnes et al. 2010], or to find exact nearest patches [Xiao et al. 2011]. Built upon the generalized PatchMatch algorithm [Barnes et al. 2010], HaCohen et al. [2011] propose a non-rigid dense correspondence (NRDC) algorithm for matching regions between images with shared content. We adopt the framework of this algorithm and extend it for our application, as described in Sec.4.

Editing process management. Kurlander and Feiner [1988] present an early work on generating editable graphical visualizations for long user sessions. More recently, Heer et al. [2008] propose a visualization system for interactively generating graphical histories. Su et al. [2009] provide another interactive visualization approach for managing operation histories for vector graphics. Grabler et al. [2009] propose a system to automatically create step-by-step, visually appealing tutorials of complicated image editing processes. Chen et al. [2011] propose a nonlinear revision control method for image editing, using a directed acyclic graph for managing and visualizing the editing process. Chen et al. [2012] suggest an *adaptive history*, which automatically segments and groups a lengthy sequence of editing commands for easy navigation. The Delta system [Kong et al. 2012] can help users identify the tradeoffs between workflows using visual comparisons. All these systems require the editing process to be available. Fu et al. [2011] propose a system to estimate a reasonable drawing order from a static line drawing, which is in spirit similar to our proposed system, but is limited to line art rather than natural images.

Edit transfer. Berthouzoz et al. [2011] propose content adaptive macros to transfer complex image manipulations applied on source images to new target images, by learning the relationships between image features and the parameters of editing procedures. Similarly, Bychkovsky et al. [2011] learn global tone adjustment models from training images, which can then be automatically applied to new images. Again, these systems require the editing procedure to be known.

Image Analogies [Hertzmann et al. 2001] provides a classic example of edit transfer *without* recovering the editing process. However, it cannot handle geometric or object-level edits. The RepFinder system [Cheng et al. 2010] finds repeated scene elements in an image so that edits made to one element can be transferred to others. As one of many applications, our recovered history from one image pair can be applied to new images to achieve semantic editing transfer. The ImageAdmixture system [Zhang et al. 2012] finds object-level grouped elements in images, and allows object mixing and appearance transferring between different images. Yücer et al. [2012] propose *transfusive* image manipulation, an automatic approach to transfer edits made to one image to others containing the same object or scene.

3 Algorithm Overview

The pipeline of our algorithm is shown in Fig. 2. It consists of three main steps: (1) region matching; (2) recovering semantic appearance operators for each matched region pair; and (3) generating the editing history.

Specifically, we first find all matched region pairs between the source and edited images (Sec. 4). This is achieved by using a matching method extended from the non-rigid-dense-correspondence (NRDC) algorithm [HaCohen et al. 2011] to accommodate a wider range of appearance difference between a pair of regions. Secondly, we recover semantic appearance operations for each matched region pair (Sec. 5), which may include both global linear color transforms such as brightness, exposure, hue and saturation adjustments, as well as non-linear tone mapping and local brushes with spatially-varying strength. Finally, given the matched region pairs and their recovered editing operations, we use an optimization approach to generate a compact, semantically-meaningful editing history, according to a set of predefined editing rules (Sec. 6).

4 Region Matching

In the first step of the algorithm, we seek to reliably recover region pairs between the source and edited images that share the same content. Comparing two matched regions gives us the means to discover whether the source region has been edited, and if so how. Our region matching approach is extended from the NRDC [HaCohen et al. 2011] algorithm to have better capabilities of handling large appearance transforms.

Specifically, we adopt the coarse-to-fine framework in the original

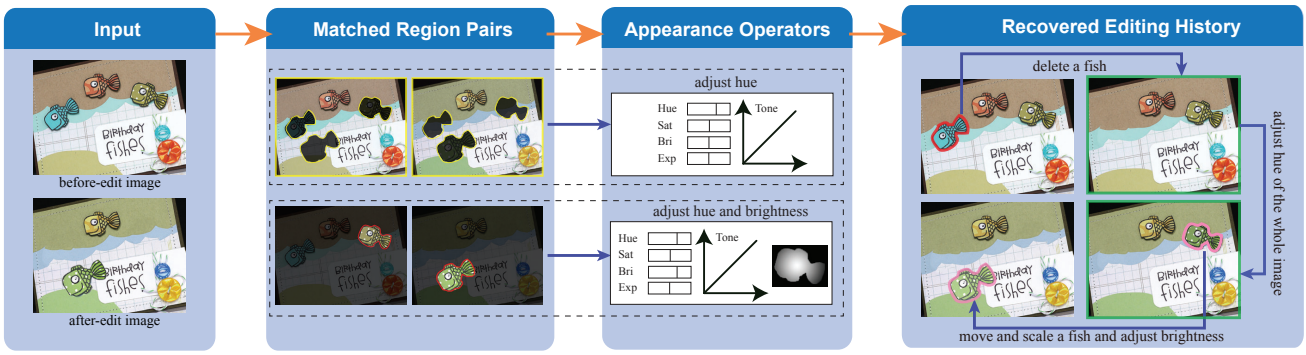


Figure 2: *Pipeline of the proposed inverse image editing system.*

NRDC approach, which initially downsamples the image to a low resolution, finds matches, and uses them as constraints for finding matches at the next finer resolution. At each level, the following four steps are applied sequentially: (1)nearest neighbor search; (2) patch merging; (3) color transform estimation; and (4) coarse-to-fine propagation. After the above steps finish at the finest level, a boundary refinement step is applied to produce an accurate boundary for each matched region. Next we will describe these steps and their differences from the original NRDC approach in more detail.

4.1 Nearest neighbor search

Given a source image A and an edited image A', for each patch u' in A', our goal is to find the closest source patch u in A that minimize a distance measure $D(u, u')$. A 4D geometric transformation G (i.e. representing 2D translation, rotation, scaling, and flipping) is used to represent the geometric relationship between u and u'.

The above problem can be efficiently solved by generalized PatchMatch [Barnes et al. 2010], which we now briefly review. Initially, a few random transformations are assigned to each patch. The algorithm improves the transformations by iterating between a *propagation* and a *random search* step. The propagation step proceeds in scan-line order, replacing the transforms of each patch by that of its neighbor if appropriate. In the random search step, each patch randomly finds several transforms for further evaluation. The transforms are randomly chosen from windows of exponentially decreasing sizes.

We utilize generalized PatchMatch, but improve upon it in several ways. Firstly, like NRDC, we use floating-point coordinates to support sub-pixel precision. Secondly, to account for cross-channel color changes between patches, we define the matching cost $D_m(u \to u')$ from patch u to patch u' as:

$$D_m(u \to u') = \min_{C_u} \left\| C_u I(u) - I(u') \right\|_2, \qquad (1)$$

where $I(\cdot)$ represents the color values of pixels in a patch. C_u is a 3×3 color transform matrix, obtained by least square minimization of the above equation. It accounts for cross-channel color transforms such as a hue change. This measure, however, do not work well with degenerated cases, e.g., a patch u' with a uniform color. Hence, we define the final distance between u' and u to be:

$$D(u, u') = \max(D_m(u \to u'), D_m(u' \to u)). \qquad (2)$$

Thirdly, as done in the PatchMatch Stereo method [Bleyer et al. 2011], we add an additional *local refinement* step in each iteration. In this step, for each patch, we locally optimize the stored geometric transforms using the gradient descent method to further reduce the distance measure in Eqn. 2. This is because the 4D transformation space is too large for random search to efficiently find optimal transforms.

4.2 Patch merging

After the best transform has been identified for each patch u', adjacent patches which are consistent are merged into larger regions. To achieve this we define the *consistency error* between two patches in the edited image as:

$$\Delta(u', v') = \frac{\|G_v(v'_c) - G_u(v'_c)\|_2}{\|G_u(u'_c) - G_u(v'_c)\|_2} + \lambda \left\| (C_u - C_v) \cdot I(u') \right\|_2,$$

where u', v' represent two patches in A' centered at u'_c, v'_c, respectively, and (G_u, C_u) and (G_v, C_v) are estimated geometric and color transforms of them. The two terms measure differences in geometric and color transforms between u' and v' respectively, and λ is a balancing weight which we fix at 0.1 when using normalized color and spatial coordinates in $[0, 1]$. Note that the corresponding definition in the NRDC approach [HaCohen et al. 2011] does not include a color difference term, which is essential to distinguish patches with only color transformation.

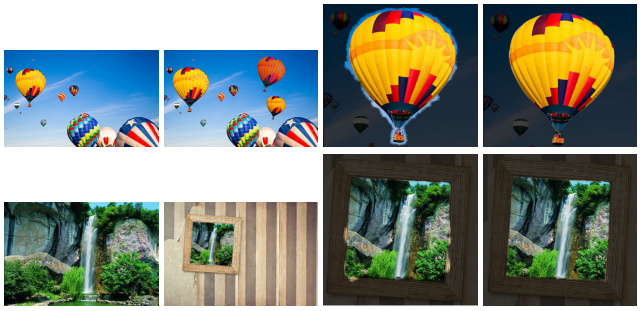
Next, we adopt a greedy scheme to merge neighboring patches based on the proposed consistency error. Specifically, we randomly select a patch, and greedily grow the region by merging it with its neighboring patches whose consistency error with respect to the selected patch is low (the threshold is 10), until no more neighbors can be included. We then select another patch to start another growing process. Following NRDC, we prune regions that are too small (less than 1% image size). This process results in a set of merged regions in A', each corresponds to a matched region in A.

4.3 Color transform estimation

For each matched region pair R and R', we assume a *per-region* cross-channel cubic color transform C_R between them, along with a *per-pixel* smoothly-varying strength map w. This allows us to handle a rich set of possible color editing operations, including hue, contrast, saturation, tone, and brightness adjustments. It is also important to note that introducing a spatially-varying strength map w allows the system to support various paint brushes for local editing. Mathematically the color transform is formulated as:

$$\begin{aligned} I_m(p') =& w(p) \cdot \sum_{i,j,k \geq 0; i+j+k \leq 3} a_{i,j,k,m} (I_0(p))^i (I_1(p))^j (I_2(p))^k \\ &+ (1 - w(p)) \cdot I_m(p), \quad m = 0, 1, 2; p \in R \end{aligned} \qquad (3)$$

where p denotes a pixel in region R, and p' is the corresponding pixel in R', $I_m(\cdot)$ denotes the color value of the m-th channel of



(a) (b) (c) (d)

Figure 3: *Boundary refinement for matched regions. (a,b) Source and edited images. (c,d) Region matching results before and after boundary refinement.*

a pixel; $a_{i,j,k,m}$ are the coefficients of the per-region cross-channel cubic color transform C_R, and $w(p)$ is the per-pixel editing strength whose value is between 0 and 1.

An iterative optimization approach is applied to recover both the per-region cross-channel cubic color transform C_R, and the per-pixel edit strength map w. We start with a constant strength map $w(p) = 1$. At each iteration, we first fix w and obtain C_R using a least square solver, by taking into account all pixels in two matched regions. We then fix C_R to recover w. We assume that it varies very smoothly, and the values are roughly the same in a small neighborhood such as a 7×7 window. Hence, for each pixel p, $w(p)$ is also obtained using a least square solver, by taking into account the neighboring pixels in a local 7×7 window. In our implementation, we find usually 3-5 iterations is usually enough to get good estimates of both C_R and w that are accurate enough for our purposes.

4.4 Coarse-to-fine propagation

Similar to the NRDC approach, in our coarse-to-fine strategy, we use the results obtained at the current level to constrain the solution at the next level with a higher resolution. Specifically, for faster convergence, the stored transform of a patch leading to a matched region have its search space at the next level limited to be within a small range (e.g. shift within 5 pixels; scale, rotation within 0.2) of the final transform at the current level. Besides, the estimated cross-channel cubic color transform C_R of a region pair is applied to reduce the appearance difference between the two regions, before evaluating the distance (Eqn. 2) between patches.

4.5 Boundary refinement

Because we use patches of size 7×7 in the region matching process, the boundary of each matched region is typically not located accurately, as shown in the examples in Fig. 3. To further improve the accuracy of region boundaries, we first over-segment both the source and the edited images using the Mean Shift algorithm [Comaniciu and Meer 2002], and assume that each segment should belong to a single region based on color consistency. Hence, we check each segment for conflict. If 80% of a segment is inside a matched region or the unmatched region U (i.e. pixels not belonging to any matched region), we assign the whole segment to the same region.

Next, in the edited image, we create a refinement band B' around each matched region R' by expanding and shrinking its boundary by 5 pixels, and then calculate an alpha matte in the refinement band using an existing alpha matting technique [Levin et al. 2008]. Once the alpha matte is computed, we also refine the transforms for pixels in B'. We use a similar algorithm to that in Sec. 4.1 to propagate the transforms from inside the region R' to the refinement band B', the only difference being that we modify Eqn. 1 to use the alpha values of pixels (α_p) as weights in calculating patch distance:

$$D_m(u \to u') = \min_{C_u} \sum_{p \in u} \alpha_p \left\| C_u I(p) - I(p') \right\|^2, \qquad (4)$$

where p is a pixel in u, and p' is the corresponding pixel in u'. In this way we avoid the influence of background colors when calculating matching distances for patches in the foreground objects. Examples of boundary refinement are shown in Fig. 3.

Finally, we remove those region pairs whose color and geometric transforms are both identity transforms (i.e. correspond to no edit). The remaining matched region pairs will be used in subsequent steps for generating the editing history. Fig. 4 shows several examples of the matched region pairs.

5 Semantic Appearance Operator Recovery

Recall that we use a per-region cross-channel cubic transform (Eqn. 3) to describe appearance changes between a matched region pair. Such a transformation is powerful for reconstruction, but lacks semantic meanings. In this section we describe how to further recover semantic editing operations from it.

A wide variety of global color and appearance adjustment tools exist in modern image editing software. By consulting professionals and studying representative software packages such as GIMP[2] , Adobe Lightroom[3] and Apple Aperture[4] , we have identified five of the most commonly used basic appearance operators: brightness, exposure, hue, saturation, and non-linear tone curve adjustments. Brightness and hue operators shift the values x of their corresponding channels by a constant f, to $x + f$. Saturation and exposure operators scale the value of each channel x to $x \cdot (1 + f)$. Inspired by existing tone curve adjustment interfaces, we use a cubic spline to represent a non-linear tone curve, parameterized by five control points: $(0, 0)$, $(0.25, f_1)$, $(0.5, f_2)$, $(0.75, f_3)$, $(1, 1)$, where f_1, f_2, f_3 are three values in $[0, 1]$. In total, we have 7 parameters for appearance editing (i.e. 1 each for brightness, exposure, hue and saturation, 3 for the tone curve). Although this is a short list of basic operators, a wide range of adjustment effects can be achieved by composing multiple operators. For instance, commonly used contrast, shadow and highlight adjustments can all be achieved by non-linear tone curve manipulation.

It is important to note that, besides the 7 per-region editing parameters, each pixel inside the region still maintains a per-pixel editing strength w, such that: $x' = w \cdot f(x) + (1 - w) \cdot x$. This allows the system to support local adjustment brushes with spatially-varying strength. Similar to Sec. 4.3, we optimize the per-region parameters and the per-pixel editing strength alternatively, as detailed below.

Parameter initialization. For initializing w, we use the editing strength map obtained in Sec. 4.3. We then obtain the initial estimations of per-region parameters by applying the following steps:

1. Convert images to HSV color space, and compute initial hue and saturation parameters by subtracting the mean hue values and dividing the mean saturation values of the two regions, respectively.

2. Convert images to grayscale, and obtain initial brightness and exposure parameters f_b and f_e by fitting a linear model $x' =$

[2] http://www.gimp.org/

[3] http://www.adobe.com/products/photoshop-lightroom.html

[4] http://www.apple.com/aperture/

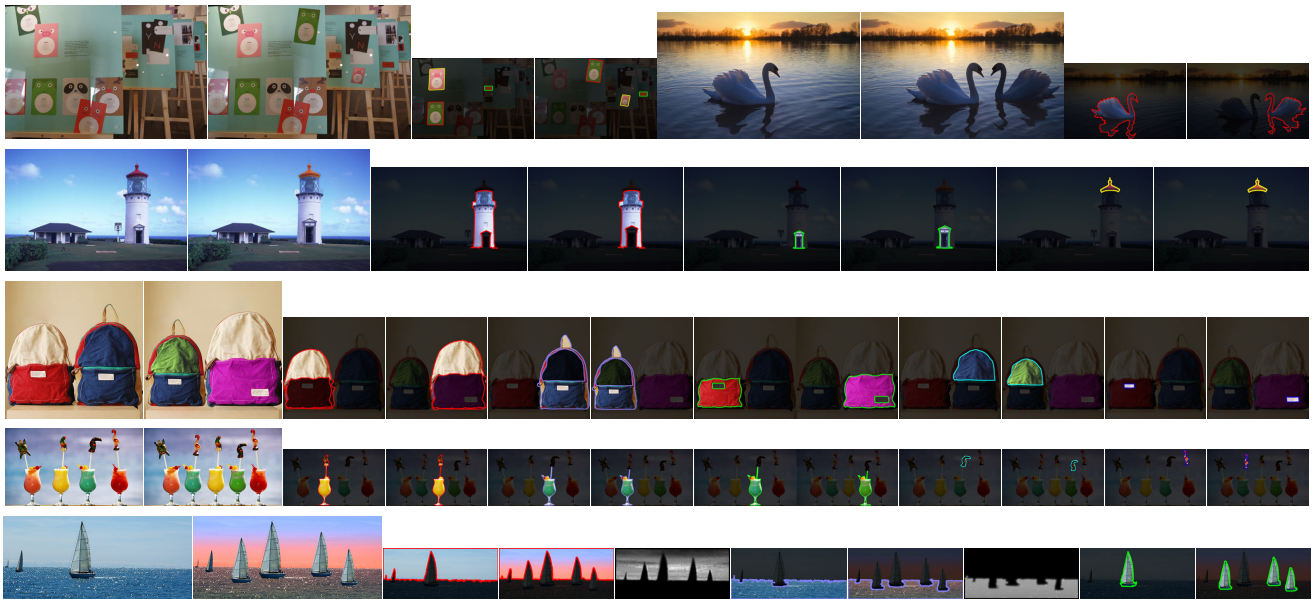


Figure 4: *Matched region pairs. Left to right: original and edited images, and recovered matched region pairs. In the last example, the recovered spatially varying strength maps are also shown.*

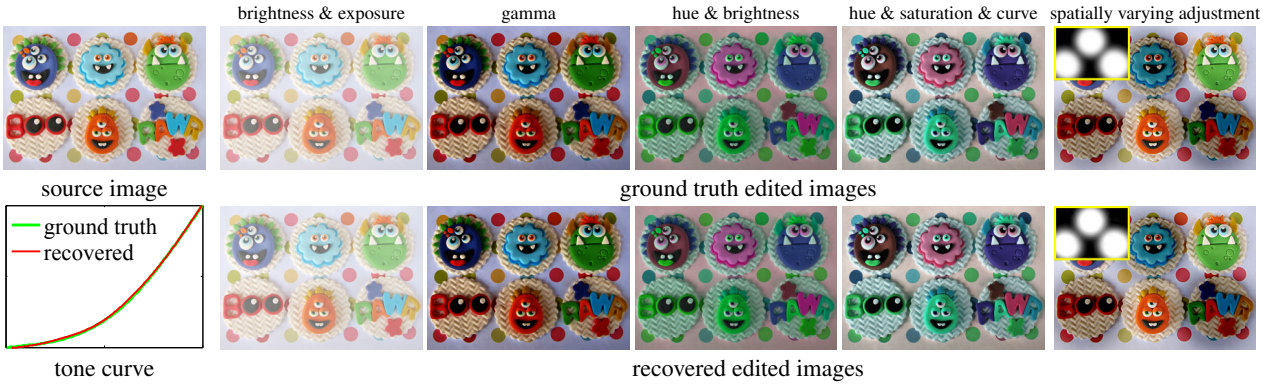


Figure 5: *The accuracy of the appearance operator recovery. Top: ground truth edited images generated by changing: brightness and exposure, gamma (tone) curve, hue and brightness, hue and saturation and tone curve, spatially varying adjustment of brightness and exposure, respectively. Bottom: results rendered using recovered operators are visually indistinguishable from the ground truth. Errors in estimated parameters are within* 1%*. Bottom-left: ground truth and recovered tone curves for the 4-th "hue and saturation and tone curve" example. Notice that in the rightmost example, spatially varying adjustments of brightness and exposure have been applied, and the ground truth and recovered spatially-varying strength maps are shown in the top left corner of corresponding images, respectively.*

$f_e \cdot x + f_b$, where x and x' are intensities of corresponding pixels in the two regions.

3. After compensating for brightness and exposure differences, use least square fitting to obtain initial tone curve parameters.

Parameter refinement. After initialization, we iteratively optimize the per-pixel strength map w and the per-region parameters. we first fix w and use a gradient descent method to search for optimal parameter values, by minimizing the L_2 color difference between the two regions after applying the color adjustments. Since the parameters are also influenced by the order of operations, we assume the following fixed order for color adjustments: (1) hue, (2) saturation, (3) brightness, (4) exposure, (5) tone curve. Next, we fix the per-region parameters and adjust w, using the same local constancy assumption and least square solvers as described in Sec. 4.3. We usually apply 5 iterations in this process.

Operation removal. After the above optimization process, we examine the parameter values to see if they are close to the default values that correspond to no change. If some of them are, we then assume that the corresponding edits have not been applied. We then fix these parameters to their default values and re-apply the optimization process to update others.

Fig. 5 shows some examples of recovered appearance operators. It suggests that the recovered operators are accurate and can generate high fidelity rendered results when compared with the ground truth edited images. Fig. 6 gives another example on spatially varying adjustment. From left to right, we give the source image, edited image, and recovered per-pixel editing strength map. The result suggests our method is robust to recover such edits. A more thorough evaluation is presented in Sec. 8.

6 Recovering the Editing History

So far we have generated a list of matched region pairs, and appearance and geometric transforms between each pair. We now explain how to further process the matching result to generate a compact

source image edited image recovered strength map

Figure 6: *Spatially varying adjustment example.*

editing history.

6.1 Layered editing

Before introducing the proposed editing path recovery method, it is important to note that our system intrinsically does layered editing, where each pair of matched region is on a different layer. Note that the same object in the source image A (denoted as $O(A)$) could match to multiple objects in the edited image A' (denoted as $O_i(A')$), often caused by cloning. In this case there will be multiple copies of $O(A)$ and each one pairs with an $O_i(A')$ and lives in a different layer.

6.2 The optimal editing path

The problem of generating an editing path from matched regions does not have a unique solution, as many different paths can lead to the same editing result. To find a reasonable path, we first introduce the concept of *state*. Formally, let the list of matched region pairs be $\{R_k \xrightarrow{c_k, g_k} R'_k\}$ ($1 \le k \le n$, n is the number of matches), where c_k, g_k denote the estimated per-region appearance and geometric transforms between each pair. We denote the state of each region R_k by the edits that have already been applied. Hence, the beginning state of an editing path is: $\Phi_{start} = \{\phi_1 = (0,0), \phi_2 = (0,0), ..., \phi_n = (0,0)\}$, and the final state is: $\Phi_{end} = \{\phi_1 = (c_1, g_1), \phi_2 = (c_2, g_2), ..., \phi_n = (c_n, g_n)\}$. Our goal is to find a stack of editing steps that change the state from Φ_{start} to Φ_{end}.

In this framework, a single editing step involves selecting one, or multiple spatially connected regions $\{R_j\}$ simultaneously, and modifying their states in the same way: $\phi_j = \phi_j + (c_e, g_e)$ for all j, where (c_e, g_e) indicates the particular operation applied. (c_e, g_e) are not allowed to be both zero (i.e., no edit). Furthermore, either c_e or g_e should satisfy the constraint that $c_e = c_j - c_j^{now}$ or $g_e = g_j - g_j^{now}$ for at least one selected region R_j; (c_j^{now}, g_j^{now}) denotes the current state of region R_j. In other words, at least one region reaches its final color or geometric transform in a single editing step. Given these constraints, at any step, the number of possible editing paths to be taken is limited. For example, given a region R_1 whose current state is $\phi_1 = (0,0)$, the allowed editing steps are $\{(0, g_1), (c_1, 0), (c_1, g_1)\}$.

We also need to define semantically what is the *optimal* path from Φ_{start} to Φ_{end}. By consulting with experienced artists, we have determined four principles that the majority of people agree:

1. Layers are edited in the coarse-to-fine order. Large, visually dominant edits are applied before fine-tuning the appearance of small objects;
2. If the same edits are applied on multiple objects, then they are applied either together at once, or sequentially without interruption (temporal focus);
3. Nearby objects are edited sequentially before moving to far away objects (spatial focus);
4. Downsampling is deferred as much as possible.

Specifically, Principle 2 and 3 reflects the spatio-temporal local editing focus, meaning that the users are more likely to focus on the same editing operations for a while before moving to other tools, and focus on nearby objects in a local region first before moving to far away ones.

Based on the above principles, we define the total *editing cost* as:

$$\sum_{1 \le i \le m} E_i + \lambda_p \sum_{1 \le i \le m-1} |t_{i+1} - t'_i|, \tag{5}$$

where m is the number of editing steps, E_i denotes the cost of the i-th editing step (to be explained next), and t_i, t'_i denote the centroids of the selected region(s) of the i-th editing step before and after applying the edits, respectively.

The first term is the summed cost of all editing steps, and the second term measures the switching costs between adjacent editing steps according to Principle 3 (i.e. $|t_{i+1} - t'_i|$ approximates the spatial movement from the i-th step to the $(i+1)$-th step). λ_p is a controlling weight set at 0.5 in our experiments.

The cost of the i-th editing step E_i is defined as:

$$E_i = \sqrt{\|S_i\|} \cdot (d_c(i) + d_g(i)) \cdot (1 + \lambda_s i) \cdot P_i, \tag{6}$$

where $S_i = \{R_j\}$ denotes the selected region(s) of the i-th editing step, $\|S_i\|$ is its size and $\sqrt{\|S_i\|}$ is used to approximate its visual dominance. $d_c(i)$ and $d_g(i)$ measure the amount of color and geometric transforms applied in this step, their exact formulation will be given later. λ_s is a constant set to 0.01 in our system. The term $(1 + \lambda_s i)$ has two purposes: (i) it penalizes longer editing paths: a shorter path is preferred for the same result, and (ii) it favors edits on dominant objects (i.e. $\|S_i\|$ is large) being applied earlier (i.e. when i is smaller), thus satisfying Principle 1.

The final term P_i in Eqn. 6 penalizes for early down-sampling (satisfying Principle 4). It is defined as the average scaling factor that has already been applied:

$$P_i = \frac{\sum_j \|R_j\| \min(1/t_{scale}(R_j), 1)}{\sum_j \|R_j\|},$$

where j iterates over all selected regions in this editing step, $t_{scale}(R_j)$ is the scale factor that has already been applied to that region R_j. Note that we completely avoid enlarging after downsampling the same region, by setting $P_i = +\infty$ for such edits.

Finally, the amount of color and geometric transforms $d_c(i)$ and $d_g(i)$ in Eqn. 6 are defined as:

$$d_c(i) = \sum_{p \in S_i} \|I(p') - I(p)\| / (\sigma_c \|R_s\|),$$

$$d_g(i) = \sqrt{T_x^2 + T_y^2}/\sigma_{\text{xy}} + T_{\text{rot}}/\sigma_{\text{rot}} + |\log T_{\text{scale}}|/\sigma_{\text{scale}},$$

where p is a pixel in S_i and p' is its corresponding pixel, and σ_c controls the influence of the color transform. (T_x, T_y), T_{rot} and T_{scale} denote the average translation, rotation angle and scaling of the region, respectively. σ_{xy}, σ_{rot} and σ_{scale} are weights controlling the influence of each term. In our system, the weights are empirically set to $\sigma_c = 1, \sigma_{xy} = 1, \sigma_{rot} = \pi, \sigma_{scale} = \ln 3$.

With above definitions, we seek the editing path from Φ_{start} to Φ_{end} with minimal total editing cost (as defined in Eqn. 5) among all possible paths. This can be efficiently solved using dynamic programming.

Figure 7: *Recovered editing histories from before-and-after image pairs.*

6.3 Handling no-match regions

There may exist regions in both A and A' that cannot find good matches on the other image. This may be caused by operations such as cropping, object insertion or removal. We first examine if the edited image A' has been derived from the original A by cropping, by finding two bounding boxes $B(A)$ an $B(A')$ containing all the matched regions in A and A', respectively. Cropping is identified if $B(A')$ covers the entire image while $B(A)$ covers only a portion of it, and $B(A)$ is treated as the cropping window.

We then look for object insertion and removal. An unmatched region in A' may be the result of inserting an object into A, or removing one from A and filling the hole using image completion techniques. If the region is caused by image completion, then there should be no obvious seams between the unmatched region and its surrounding ones. Otherwise if the region is caused by inserting a new object, then we expect the region to be surrounded by a strong object boundary. We thus check how smooth the transition is between the region and its surroundings. Specifically, we check how well the region boundary agrees with the over-segmentation boundaries obtained by the mean shift algorithm (see Sec. 4.5). If they agree with each other well (e.g. there is more than 50% overlap), this region is considered to be a newly inserted object. Otherwise we further check the same region in A, and if the source region is also an unmatched region, then we decide that the object in the source region has been removed and the hole has been filled by image completion techniques.

If such operations are identified, we incorporate them into the editing path computed in Sec. 6.2. Specifically, we add cropping and object removal at the beginning of the history, and add object insertion at the end of it to create a completed editing path.

7 Applications and Results

We have implemented our method on a PC with an Intel Xeon 2.4GHz CPU and 8GB memory. For an image of size 640×480, region matching takes 2–3 minutes, recovering appearance operators takes about 10–15 seconds, depending on the number of regions being edited, and finding the optimal editing path takes about 2–5 seconds.

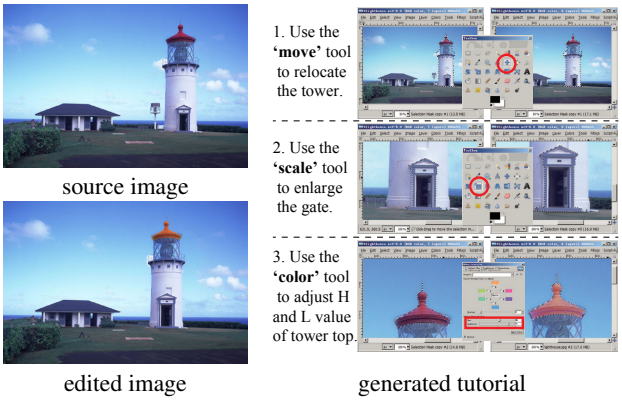


Figure 8: *Generating a tutorial from the recovered edit history.*

Fig. 7 shows a number of input before-and-after image pairs and the recovered editing histories using our approach. It suggests that our system works reliably even for large geometric and appearance transforms, and the appearance transform is allowed to be spatially varying (e.g. the first example). Furthermore, the recovered editing steps are semantically-meaningful, and can easily be used to drive tools in image editing software. Note how the system generates a compact and reasonable editing history for the 2nd and 3rd examples, from the low-level region matching results shown in the 3rd and 4th rows of Fig. 4. Next, we illustrate how the recovered history can be used in various applications.

Automatic tutorial generation. A straightforward application is to use the recovered editing history to automatically generate an image editing tutorial. Fig. 8 shows a simple example of turning the history into a step-by-step tutorial. Our method can be combined with

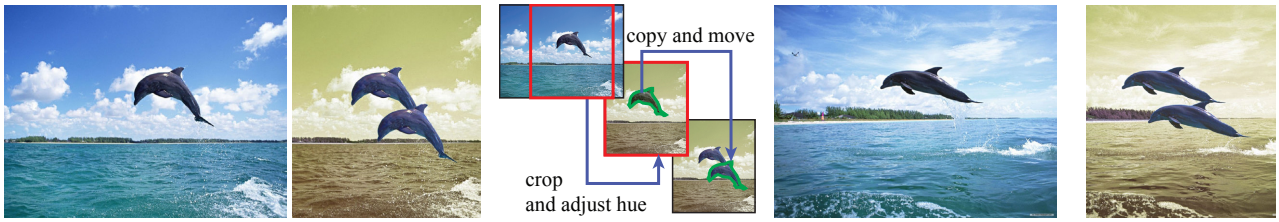


Figure 10: *Edit transfer. Given the image pair A and A', we first recover the editing history (middle), then apply it to image B to generate a result B' in a similar way. Object segmentation in B is obtained by GrabCut.*

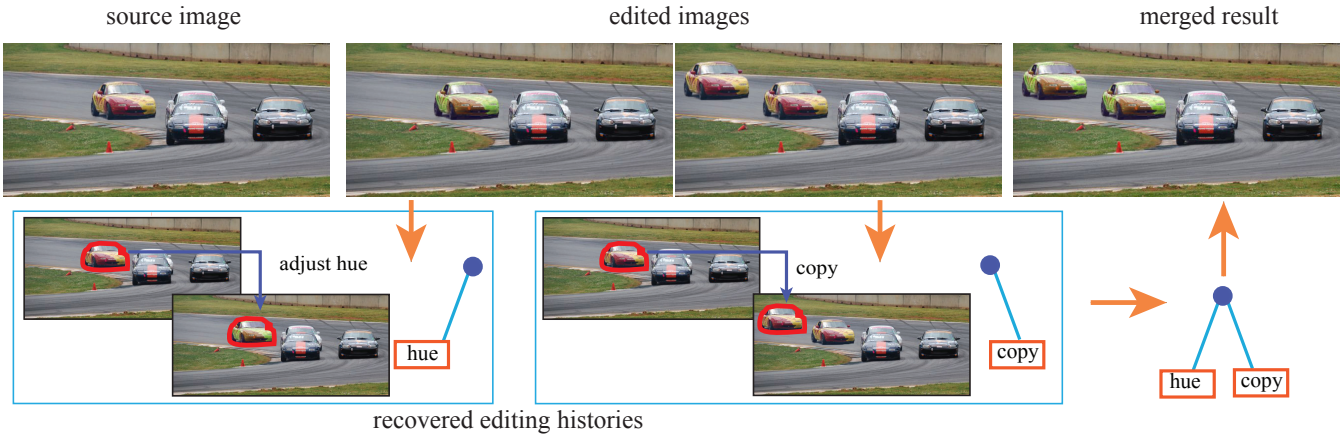


Figure 11: *Merging multiple editing paths from the same source image. The image on the right is rendered using the merged editing history. Image courtesy of Flickr user Hans J E.*

Figure 9: *Image re-editing results generated from recovered editing histories (see original examples in Fig. 4 top right and the second row of Fig. 7). Left: increasing (instead of decreasing in the original edit) the brightness of the swan on the right. Right: the first editing step (move and scale) is modified to only include move, and the fifth editing step (move) is changed to removal.*

more powerful tutorial generating systems [Grabler et al. 2009] to produce more visually appealing results.

Re-editing. Users can modify the recovered history, e.g. to remove steps, or change parameters of some steps, and re-apply the modified history to the source image A to generate a new edited image A''. Some examples are shown in Fig. 1 and Fig. 9. This is easier than directly editing the original edited image A' to achieve A'', as the user may have to carry out object selection and use other editing tools again in the latter case; the user may also no longer remember the parameters for the parts of the edit which are to remain unchanged.

Edit transfer. The recovered editing history from one pair of images can be transferred to a new image to achieve edit transfer. An example is shown in Fig. 10, where the recovered editing history from images A and A' is applied to image B. To do this, we assume that A and B have similar composition, so that objects in B are roughly at the same locations as those in A. To automatically extract the object mask in B, we first identify a rough bounding box in B which is 1.5 times larger than the actual object bounding box in A, then apply GrabCut [Rother et al. 2004] for segmentation.

Other dedicated approaches have already been proposed for edit transfer, such as image analogies [Hertzmann et al. 2001] and content-adaptive macros [Berthouzoz et al. 2011]. Compared to image analogies, our method has more constraints on the input, requiring similar composition of images A and B. However, on the other hand, it is capable of object-level editing, and also handles geometric editing such as cropping, which are clear advantages over image analogies and other appearance-based transfer approaches. Content-adaptive macros requires the editing history to be known, so our system can be potentially used to generate such editing macros.

Merging editing paths. Our approach can be combined with the image revision control system [Chen et al. 2011] to merge different editing paths. An example is shown in Fig. 11, where our system recovers multiple different editing paths and merges them to create a single final rendering result.

8 Evaluation

To objectively evaluate the proposed system, we create an evaluation dataset that contains 21 image editing examples produced by several artists. To produce these examples, we explicitly explained the range of supported operations to the artists and asked them to perform editing using supported operations. The input images and the performed editing paths were chosen or created by the artists

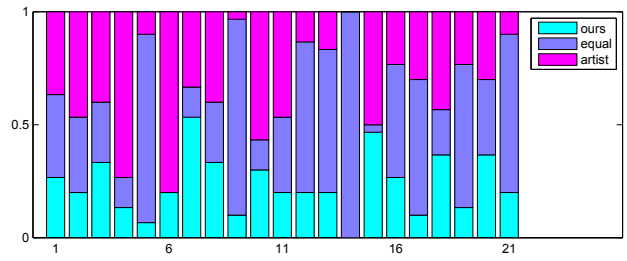


Figure 12: *Percentage in favor of our and artist generated edit sequences.*

without any supervision. We then selected 21 successful or moderately successful examples out of 25 that we received from the artists for user study. The original editing steps are recorded for all examples. We generate editing histories for all these examples using our system, and compare them to the original ones.

We first evaluate the representation ability of the recover histories, by measuring the PSNR values of the reconstructed edited images. The average PSNR value of this dataset is 26.1dB, and the standard deviation is 6.2dB. Visually, there is no noticeable difference between the reconstructed and the ground truth edited images for most examples. These results suggest that our recovered histories can faithfully reproduce the edited images in high fidelity.

Secondly, we conducted a qualitative evaluation on the semantic meaningfulness of the recovered histories. We invited 30 participants who are image editing enthusiasts and are familiar with Adobe Photoshop to participate in the study. In each user session, for each editing example, we first showed the before-and-after image pair to the subject, followed by the two editing histories: the original one and the recovered one. The display orders of the two histories were randomly determined. Each history was illustrated as an automatic slideshow, and the subject was allowed to switch back and forth to see each step more clearly, and there is no time limit in the study. After viewing a pair of editing histories, the subject was then asked to judge which one is semantically more natural, or they are about equal. The results of the user study are given in Fig. 12. In total, our method achieved an averaged score of 0.45 (i.e. "ours is better", "about equal", "artist's is better" are scored at 1, 0.5, and 0, respectively), and the one tail p-value is 0.039 [5]. This study demonstrates that our recovered histories are comparable to the original ones produced by artists in terms of semantic meaningfulness. In 17 out of 21 examples (81.0%), a majority of users rated that our recovered histories are equal or better than the original ones. All examples used in the user study are included in the supplemental material.

9 Limitations and Discussion

9.1 Failure cases

To better understand the limitation of the proposed system, we studied the low rating examples (e.g. the 2nd and 6th examples) in the above user study to see why they are less successful. For the 2nd example (top row in Fig. 13), although our recovered editing steps are the same as that of the artist, participants rated it lower due to its visually noticeable reconstruction error. This is because the saturated color values in the sun region lead to errors in the reconstructed color transforms. For the 6th example (bottom row in Fig. 13), the original editing patch produced by the artist has a more natural spatial layout (i.e. change the color of the leaves from bottom to top, given the bottom one is visually more dominant), while our method first edits the leaves in the middle since their regions are large (see Eqn. 6). This example suggests that one could potentially use a better algorithm to rank the visual dominance of different objects to improve the semantic meaningfulness of the recovered history.

Our system can fail more dramatically when the individual technical components are incapable of handling more difficult cases. Firstly, the region matching algorithm in Sec. 4 may fail in the following cases: (1) when the size of the matched regions is too small to be classified as reliable; (2) when the applied geometry/color transformations are too dramatic; and (3) when some regions are purely textureless. Two such examples are shown in Fig. 14. In the first row of Fig. 14, the matching of the head of giraffe failed due to the large deformation. In the second row of Fig. 14, our region matching missed the left and right mushroom stems since they are too thin. Secondly, appearance operations that cannot be well approximated using the ones in Sec. 5 will lead to large reconstruction error when re-applying the history. Such an example is shown in the third row of Fig. 14, where our method failed to recover the complex appearance changes.

9.2 Supported operations

As discussed earlier, our system currently only supports a limited range of operations. We believe that with dedicated solutions, other editing operations could potentially be supported. For instance, to support Gaussian blur, we can optionally add it as a new transform dimension in the region matching step. By allowing a blurred region to match against multiple versions of the original image with different amount of blur, we can match the blurred region to its original sharp region, and at the same time produce an estimation of the size of the blur. In the appearance operator recovery step, we can first blur the source region using the recovered blur, then downsize both regions to a lower resolution to remove the effect of blurring for estimating other color operators. We implemented the above procedure in the system and in Fig. 15, we provide an example involving two operations: foreground move, and background Gaussian blur. Our method successfully recovered both operations.

Other more complicated operations are not supported in our current system, such as bilateral filtering, alpha matting, Poisson blending, etc. In general, it is relatively easy to support global operations. However, if the operation is "content aware", i.e., the color of a pixel is changed adaptively according to its local neighborhood appearance, such as bilateral filtering, then it is much harder to recover. Nevertheless, we have demonstrated that our current system is already widely useful as it supports a wide range of commonly used editing operations.

10 Conclusion

We present a novel system for recovering a semantically meaningful editing history from a source image and an edited version of it. To achieve this, we use a dense correspondence method which extends the NRDC approach to find all edited regions, and recovers appearance operations applied to each region. From all possible edit paths, we recover an optimal one based on semantic constraints. Experimental and user study results show that our system can recover clean and meaningful editing histories involving large geometric and appearance transformations. We further show that the recovered histories can be useful in a wide range of applications.

[5] This p-value measures the average scores of all examples, i.e. compute averaged scores for each example, and measure the p-value of 21 scores.

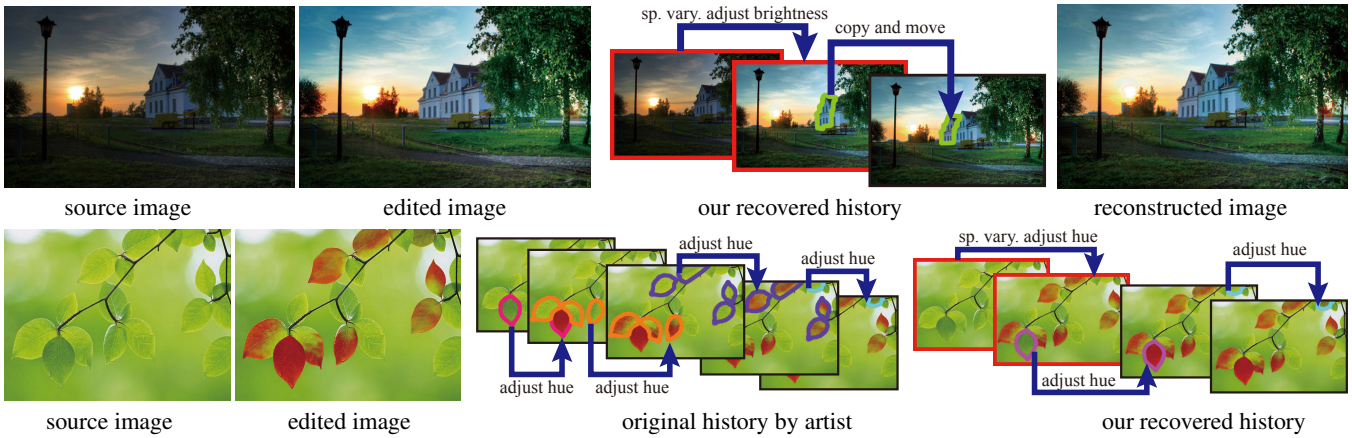


Figure 13: *Less successful examples. Top: noticeable color reconstruction error due to the saturated sun region. Bottom: the recovered history has less ideal operation order compared with the original one. Image courtesy of Flickr user äquinoktium (top).*

As future work, we would like to improve the robustness of the proposed system, by replacing some technical components with newly developed, more advanced methods. For example, we could potentially improve the robustness of the region matching step by using recently proposed higher-order deformation models [Yücer et al. 2012], after obtaining the initial matching by our proposed method. Another way to improve system robustness is to combine our method with techniques proposed in previous photo manipulation detection systems [O'Brien and Farid 2012; Kee et al. 2013]. We are also interested in combining our system with the work of [Ma et al. 2013] to automatically generate change blindness images, and in extending our method to handle vector images [Lai et al. 2009; Liao et al. 2012]. Another potential extension is to apply our method to large image libraries [Hu et al. 2013], for analyzing image correlations and/or dependencies within a large dataset.

Acknowlegements. We thank the anonymous reviewers for their valuable comments. This work was supported by National Basic Research Project of China (2011CB302205), Natural Science Foundation of China (61120106007 and 61170153), National High Technology Research and Development Program of China (2012AA011802), PCSIRT and Tsinghua University Initiative Scientific Research Program.

References

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3, 24:1–24:11.

BARNES, C., SHECHTMAN, E., GOLDMAN, D. B., AND FINKELSTEIN, A. 2010. The generalized patchmatch correspondence algorithm. In *Proc. of ECCV*, 29–43.

BERTHOUZOZ, F., LI, W., DONTCHEVA, M., AND AGRAWALA, M. 2011. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM Trans. Graph. 30*, 5, 120:1–120:14.

BLEYER, M., RHEMANN, C., AND ROTHER, C. 2011. Patchmatch stereo - stereo matching with slanted support windows. In *Proceedings of the British Machine Vision Conference*, 14.1 – 14.11.

BROX, T., BREGLER, C., AND MALIK, J. 2009. Large displacement optical flow. In *Proc. of CVPR*, 41 –48.

BYCHKOVSKY, V., PARIS, S., CHAN, E., AND DURAND, F. 2011. Learning photographic global tonal adjustment with a database of input / output image pairs. In *Proc. of CVPR*, 97–104.

CHEN, H.-T., WEI, L.-Y., AND CHANG, C.-F. 2011. Nonlinear revision control for images. *ACM Trans. Graph. 30*, 4, 105:1–105:10.

CHEN, H.-T., WEI, L.-Y., HARTMANN, B., AND AGRAWALA, M. 2012. Data-driven adaptive history for image editing. *Technical Report*.

CHENG, M.-M., ZHANG, F.-L., MITRA, N. J., HUANG, X., AND HU, S.-M. 2010. Repfinder: finding approximately repeated scene elements for image editing. *ACM Trans. Graph. 29* (July), 83:1–83:8.

COMANICIU, D., AND MEER, P. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 5, 603–619.

FU, H., ZHOU, S., LIU, L., AND MITRA, N. J. 2011. Animated construction of line drawings. *ACM Trans. Graph. 30*, 6 (Dec.), 133:1–133:10.

GRABLER, F., AGRAWALA, M., LI, W., DONTCHEVA, M., AND IGARASHI, T. 2009. Generating photo manipulation tutorials by demonstration. *ACM Trans. Graph. 28*, 3 (July), 66:1–66:9.

HACOHEN, Y., SHECHTMAN, E., GOLDMAN, D. B., AND LISCHINSKI, D. 2011. Non-rigid dense correspondence with applications for image enhancement. *ACM Trans. Graph. 30*, 4, 70:1–70:10.

HEER, J., MACKINLAY, J., STOLTE, C., AND AGRAWALA, M. 2008. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics 14*, 6, 1189–1196.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proc. of Siggraph*, 327–340.

HU, S.-M., CHEN, T., XU, K., CHENG, M.-M., AND MARTIN, R. R. 2013. Internet visual media processing: a survey with graphics and vision applications. *The Visual Computer 29*, 5, 393–405.

source image edited image reconstructed image

Figure 14: *Failure examples due to large geometric deformation (top), small image structures (middle) and complex color change (bottom). Image courtesy of Flickr user Michael Li (third row).*

source image edited image recovered history

Figure 15: *A gaussian blur example. Image courtesy of Flickr user LarryBiker.*

KEE, E., O'BRIEN, J., AND FARID, H. 2013. Exposing photo manipulation with inconsistent shadows. *ACM Transactions on Graphics 32*, 3, 28:1–28:12.

KONG, N., GROSSMAN, T., HARTMANN, B., AGRAWALA, M., AND FITZMAURICE, G. W. 2012. Delta: a tool for representing and comparing workflows. In *Proc. of CHI*, 1027–1036.

KURLANDER, D., AND FEINER, S. 1988. Editable graphical histories. In *IEEE Workshop on Visual Languages*, 127–134.

LAI, Y.-K., HU, S.-M., AND MARTIN, R. R. 2009. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph. 28*, 3, 85:1–85:8.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 2, 228–242.

LIAO, Z., HOPPE, H., FORSYTH, D., AND YU, Y. 2012. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics 18*, 11, 1858–1867.

LIU, C., YUEN, J., TORRALBA, A., SIVIC, J., AND FREEMAN, W. T. 2008. Sift flow: Dense correspondence across different scenes. In *Proc. of ECCV*, 28–42.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60*, 2, 91–110.

MA, L.-Q., XU, K., WONG, T.-T., JIANG, B.-Y., AND HU, S.-M. 2013. Change blindness images. *IEEE Transactions on Visualization and Computer Graphics*, to appear.

O'BRIEN, J. F., AND FARID, H. 2012. Exposing photo manipulation with inconsistent reflections. *ACM Transactions on Graphics 31*, 1, 4:1–4:11.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. 23*, 3, 309–314.

SU, S. L., PARIS, S., ALIAGA, F., SCULL, C., JOHNSON, S., AND DURAND, F. 2009. Interactive visual histories for vector graphics. *Tech Report, MIT-CSAIL-TR-2009-031*.

XIAO, C., LIU, M., YONGWEI, N., AND DONG, Z. 2011. Fast exact nearest patch matching for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics 17*, 8, 1122–1134.

YANG, G., STEWART, C., SOFKA, M., AND TSAI, C.-L. 2007. Registration of challenging image pairs: Initialization, estimation, and decision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 11, 1973–1989.

YÜCER, K., JACOBSON, A., HORNUNG, A., AND SORKINE, O. 2012. Transfusive image manipulation. *ACM Trans. Graph. 31*, 6, 176:1–176:9.

ZHANG, F.-L., CHENG, M.-M., JIA, J., AND HU, S.-M. 2012. Imageadmixture: Putting together dissimilar objects from groups. *IEEE Transactions on Visualization and Computer Graphics 18*, 11, 1849–1857.

ZIMMER, H., BRUHN, A., AND WEICKERT, J. 2011. Optic flow in harmony. *Int. J. Comput. Vision 93*, 3, 368–388.