The Patch Transform

Taeg Sang Cho, *Student Member*, *IEEE*, Shai Avidan, *Member*, *IEEE*, and William T. Freeman, *Fellow*, *IEEE*

Abstract—The patch transform represents an image as a bag of overlapping patches sampled on a regular grid. This representation allows users to manipulate images in the patch domain, which then seeds the inverse patch transform to synthesize modified images. Possible modifications include the spatial locations of patches, the size of the output image, or the pool of patches from which an image is reconstructed. When no modifications are made, the inverse patch transform reduces to solving a jigsaw puzzle. The inverse patch transform is posed as a patch assignment problem on a Markov random field (MRF), where each patch should be used only once and neighboring patches should fit to form a plausible image. We find an approximate solution to the MRF using loopy belief propagation, introducing an approximation that encourages the solution to use each patch only once. The image reconstruction algorithm scales well with the total number of patches through label pruning. In addition, structural misalignment artifacts are suppressed through a patch jittering scheme that spatially jitters the assigned patches. We demonstrate the patch transform and its effectiveness on natural images.

Index Terms—Image models, statistical, applications, image-based rendering.

1 Introduction

The patch transform represents an image as a bag of patches sampled on a regular grid and treats each patch as a basic element of an image. Image editing can be formulated as shuffling patches to generate a visually pleasing image while conforming to user constraints. Reconstructing images from a set of patches is called the "inverse patch transform." One can think of the inverse patch transform as solving a jigsaw puzzle subject to user constraints. When no user constraints are specified, the inverse patch transform reduces to solving a standard jigsaw puzzle. We solve the inverse patch transform by formulating a Markov random field (MRF) on image nodes and using loopy belief propagation to find the patch label at each image node.

The patch transform framework enables users to manipulate images in the "patch domain": Users can constrain patch positions and add or remove patches from an image. Such a characteristic gives rise to many useful image editing operations. For example, a user can relocate objects by specifying the desired location of certain patches. Also, a user can "retarget" an image by specifying the desired size of a modified image. Alternatively, a user can modify the amount of certain textures by adding or removing patches that belong to a particular class (say, blue sky or clouds). A user can also mix patches from multiple images to generate a collage. All of these image editing operations follow a unified pipeline with a coarse, simple user input.

- T.S. Cho and W.T. Freeman are with the CSAIL, Massachusetts Institute of Technology, 32 Vassar Street, 32D-466 Cambridge, MA 02139. E-mail: {taegsang, billf}@mit.edu.
- S. Avidan is with Tel-Aviv University, Israel. E-mail: shai.avidan@gmail.com.

Manuscript received 15 Aug. 2008; revised 5 Nov. 2008; accepted 5 May 2009; published online 22 June 2009.
Recommended for acceptance by Q. Ji, A. Torralba, T. Huang, E. Sudderth, and J. Luo.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMISI-2008-08-0515.
Digital Object Identifier no. 10.1109/TPAMI.2009.133.

Image edits, such as hole filling, texture synthesis, object removal, and image retargeting, have been previously addressed in the literature. Yet, there are two important benefits to our method. First, we propose a unified framework that addresses many editing operations. Any improvement to our algorithmic engine can, therefore, improve all image editing operations that rely on it. Second, our approach simplifies user interactions. For example, with current techniques, moving an object in the image from one location to another involves carefully segmenting the object, making room for it in its target location, pasting it to its new location, and filling the hole created in the process. In contrast, our algorithm only requires the user to roughly select a small number of patches that belong to the desired object and place them in their new location. The inverse patch transform algorithm will automatically make room for the newly located patches, fill in the hole, and rearrange the image to comply with user constraints.

This paper extends the work of Cho et al. [1] in a number of aspects. First, we sample overlapping patches from the grid, instead of the nonoverlapping patches considered in [1], to enhance the compatibility measure and reduce visual artifacts. Second, we introduce a new pruning technique, named the patch-loop-based label pruning, to reduce the complexity of belief propagation from $O(N^3)$ to subquadratic in N, where N is the total number of patches. Using label pruning, we can edit images with thousands of patches. We also introduce a patch jittering technique to reduce subpatch size structural alignment error.

2 Related Work

The inverse patch transform is closely related to solving jigsaw puzzles. Solving some types of jigsaw puzzles is NP-complete [2] because it can be reduced to a set partition problem. Nevertheless, there has been much work in the literature to (approximately) solve the problem, and for jigsaws with discriminative shapes, a polynomial algorithm can solve the puzzle. Most jigsaw puzzle solvers exploit the

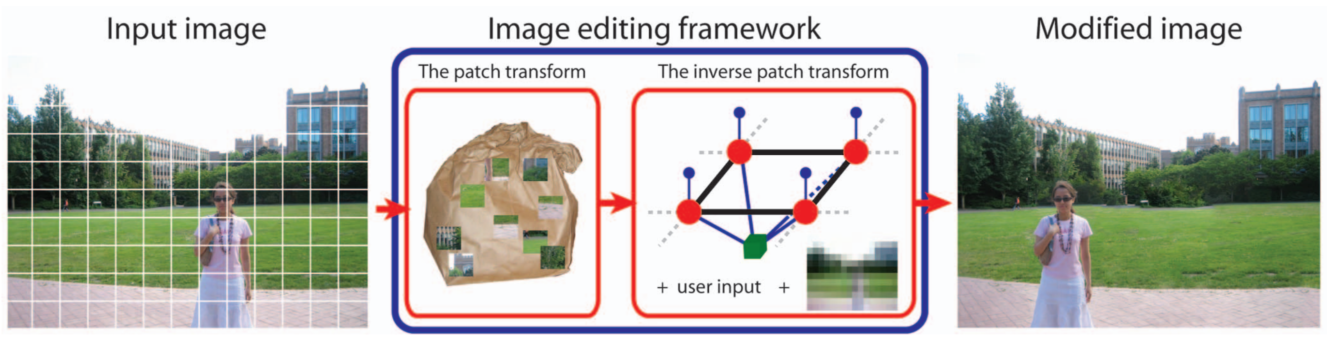

Fig. 1. Illustration of the pipeline of the patch transform-based image editing scheme. First, we compute the patch transform representation of the input image and use the inverse patch transform to solve for the patch labels on the image grid, respecting the user input, compatibility requirements, the low-resolution image, and the exclusion constraint. In the inverse patch transform block, the red ball denotes the image node and the green block denotes the exclusion factor node that steers the solution to seldom use each patch more than once.

shape and content of the jigsaw. In a shape-based approach, the boundary shape of the jigsaw is used to find valid neighbors. Even if valid neighbors can be found using shape, the problem is still NP-complete because finding the correct order of the boundary jigsaws can be reduced to the traveling salesman problem. Chung et al. [3] use both shape and color to reconstruct an image and explore several graph-based assignment techniques. To our knowledge, the largest jigsaw puzzle solved by computer is with 320 jigsaw pieces [4].

Many scientific problems have been formulated as solving a jigsaw puzzle as well: reconstructing a relic from its fragments [5], [6], [7], fitting a protein with known amino acid sequence to a 3D electron density map [8], reconstructing documents from its fragments [9], [10], and reconstructing a speech signal from its scrambles [11].

Patch-based image representations have been used in the form of "epitomes" [12] and "jigsaws" [13], where an image is represented by a small source image and a transformation map. While these models can generate an image with overlapping patches from the source image, they are applied primarily for image analysis.

There has been an evolution in the patch-based image editing community. Criminisi et al. [14] introduce a patch propagation method which is augmented with a global optimization framework by the follow-up papers [15], [16]. These global methods allow the same patch to be used multiple times, which limits the amount of control a user has over the synthesized image. Also, it may tile the same patch across the image, which may look unpleasant. Kopf et al. [17] address this issue by keeping a counter for each patch and adjusting the probability of reusing a patch accordingly. We address this issue in a probabilistic framework by introducing a term that penalizes the use of the same patch multiple times.

While the patch transform is also closely related to existing image editing frameworks, the patch transform tries to sidestep other typical image editing tasks, such as region selection [18] and object placement or blending [19], [20], [21]. Techniques on image matting and image composition [21], [22] work at a pixel-level accuracy, and were shown to perform well in extracting foreground layers in images and placing them on a new background, thus avoiding the difficult tasks of hole filling, image reorganization, or image retargeting. The patch transform inherently works with patch-level accuracy—thus, not requiring the user to provide very accurate input—and it *adjusts* the image to the user input so as to make the output image as plausible as possible. Related functionalities have been obtained using the notion of bidirectional similarity, which is described in [23].

The patch transform stitches patches together to synthesize new images, thus it is closely related to larger spatial scale versions of that task, including Auto Collage [24] and panorama stitching [25], although with different goals. Nonparametric texture synthesis algorithms, such as [26], and image filling-in, such as [14], [27], [28], can involve combining smaller image elements and are more closely related to our task. Also related, in terms of goals and techniques, are the patch-based image synthesis methods [14], [29], which also require compatibility measures between patches. Efros and Freeman [29] and Liang et al. [30] use overlapping patches to synthesize a larger texture image. Neighboring patch compatibilities were found through squared difference calculations in the overlap regions. Freeman et al. [31] use similar patch compatibilities and used loopy belief propagation in an MRF to select image patches from a set of candidates. Kwatra et al. [32], Ramanarayanan and Bala [33], and Komodakis and Tziritas [34] employ related Markov random field models, solved using graph cuts or belief propagation, for texture synthesis and image completion. The most salient difference from texture synthesis methods is the patch transform's constraint against multiple uses of a single patch. This allows for the patch transform's controlled rearrangement of an image.

3 The Inverse Patch Transform

3.1 The Image Editing Framework

We introduce an image editing framework (Fig. 1) leveraging the patch transform. Given an input image, the system samples overlapping patches from a regular grid, each of the same size and the same amount of overlap, and computes the compatibility among all possible pairs of patches.

The inverse patch transform reconstructs an image by first formulating an MRF in which nodes represent spatial positions, where we place the patches. We call these nodes the "image nodes." The inverse patch transform runs loopy belief propagation to solve for the patch assignment that is visually pleasing, is similar to the original image at low resolution, and satisfes user inputs.

Once patches are assigned to the image nodes, we stitch patches together in a way similar to the algorithm introduced by Efros and Freeman [29]: Image nodes are scanned in a horizontal-first manner, and at each image node, the patch is stitched to the image thus far blended by finding the seam that results in minimum artifacts. The stitched image can still contain artifacts due to luminance difference if neighboring patches were not adjacent in the original image. We remove the intensity gradients along the seam if the adjacent two patches were not adjacent in the original image. We use the Poisson solver provided by Agrawal et al. [35].

3.2 The Image Model

The unknown state at the ith image node is the index of the patch x_i to be placed at that position. We let $\mathbf{x}$ be a vector of unknown patch indices x_i at each of the N image positions i. We define a compatibility Ψ that measures how plausibly one patch fits next to another. Each image node has four neighbors (except at the image boundary), and we write the compatibility between patch k and patch l, placed at neighboring image positions i and j, to be $\psi_{i,j}(k,l)$. A "patch exclusion" function, $E(\mathbf{x})$, is zero if any two elements of $\mathbf{x}$ are the same (if any patch is used more than once) and is otherwise one. We can represent user constraints using local evidence terms $\phi_i(x_i)$. We use $\phi_i(x_i)$ to also ensure that the final image has a similar structure to the original image: $\phi_i(x_i)$ favors patch assignments $\mathbf{x}$ whose low-resolution version looks similar to the low-resolution version of the original image. We define the probability of an assignment $\mathbf{x}$ of patches to image positions to be

$$P(\mathbf{x}) \propto \left\{ \prod_i \phi_i(x_i) \prod_{j \in \mathcal{N}(i)} \psi_{ij}(x_i, x_j) \right\} E(\mathbf{x}). \tag{1}$$

By maximizing $P(\mathbf{x})$, we seek a solution that matches compatible patches locally while ensuring that each patch is used only once. In the next section, we introduce a message passing scheme to find the patch assignment $\mathbf{x}$ that approximately maximizes $P(\mathbf{x})$.

3.3 Approximate Solution by Belief Propagation

Finding the assignment $\mathbf{x}$ that maximizes $P(\mathbf{x})$ (1) is NP-hard, but approximate methods can nonetheless give good results. One such method is belief propagation. Belief propagation is an exact inference algorithm for Markov networks without loops, but can give good results even in some networks with loops [36]. For belief propagation applied in networks with loops, different factorizations of the MRF joint probability can lead to different results. We factorize (1) as a directed graph:

$$P(\mathbf{x}) \propto \prod_{i=1}^{N} \prod_{j \in \mathcal{N}(i)} p(y_i|x_i) p_{i,j}(x_j|x_i) p(x_i) E(\mathbf{x}), \tag{2}$$

where $\mathcal{N}(i)$ is the neighboring indices of x_i, y_i is the patch at location i in the original image, and $p_{i,j}$ is the appropriate normalized compatibility determined by the relative location of j with respect to i. A normalized compatibility $p_{i,j}(x_i|x_j)$ is defined as follows:

$$p_{i,j}(x_i|x_j) \approx \frac{\psi_{i,j}(x_i, x_j)}{\sum_{i=1}^{M} \psi_{i,j}(x_i, x_j)}, \tag{3}$$

and the local evidence term $p(y_i \mid x_i) = \phi_i(x_i)$. In most cases, we model $p(x_i)$ as a uniform distribution, but we can manipulate $p(x_i)$ to steer the MRF to favor patches with certain characteristics. We use this factorization, instead of (1), to ensure that textureless patches are not overly favored compared to other patches. Freeman et al. [31] use a similar factorization. Equation (2) is exact for a tree, but only approximate for an MRF with loops.

We can find the approximate marginal probability at node i by iterating the sum-product message passing scheme until convergence [36]. Ignoring the exclusion term $E(\mathbf{x})$ for now, the message update rules for this factorization are as follows: Let us suppose that j is in the neighborhood of i. The message from j to i is

$$m_{ji}(x_i) \propto \sum_{x_j} p_{i,j}(x_i|x_j) p(y_j|x_j) \prod_{l \in \mathcal{N}(j) \setminus i} m_{lj}(x_j), \tag{4}$$

and the patch assignment at node i is

$$\hat{x_i} = \operatorname*{argmax}_{l} b_i(x_i = l), \tag{5}$$

where the belief at node i is defined as follows:

$$b_i(x_i) = p(y_i|x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(x_i). \tag{6}$$

3.4 Message Updates with the Patch Exclusion Term

The message passing scheme in the previous section may fail to reconstruct the original image because each patch can be used more than once. In this section, we introduce a message passing scheme that integrates the exclusion term so that it favors a solution that seldom uses each patch more than once.

Since the exclusion term is a global function involving all x_i, we represent it as a factor node (shown in Fig. 1) that is connected to every image node i [36]. The message from a node i to a factor node (m_{if}) is the belief $b_i(x_i)$ (6), and the message from a factor node to a node i is as follows:

$$m_{fi}(x_i) = \sum_{\{x_1,\ldots,x_N\} \setminus x_i} \psi_F(x_1, \ldots, x_N) \prod_{t \in S \setminus i} m_{tf}(x_t), \tag{7}$$

where S is the set of all image nodes. If any of the two nodes in S share the same patch, $\psi_F(\cdot)$ is zero, and is one otherwise. The message computation involves marginalizing $N-1$ state variables (i.e., the image nodes) that can take on M different values (i.e., the number of patches), so the complexity of the marginalization becomes $O(M^{(N-1)})$, which is often intractable.

We propose an approximate solution to (7). Instead of marginalizing variables over a joint potential $\psi_F(x_1, \ldots, x_N)$, we approximate $\psi_F(x_1, \ldots, x_N)$ as a product of pairwise exclusion potentials. For computing the message from the exclusion factor node to an image node i,

$$\psi_{F_i}(x_1, \ldots, x_N) \approx \prod_{t \in S \setminus i} \psi_{F_t}(x_t|x_i), \tag{8}$$

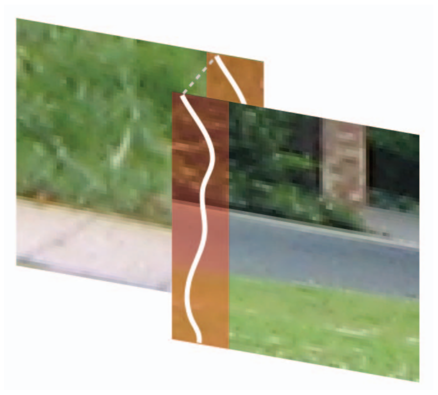

Fig. 2. Illustration of how we compute the left-right seam energy for two overlapping patches. The white line is the seam along which the color difference between the patches is minimum. The color difference along the white line is defined as the seam energy.

where

$$\psi_{F_j}(x_j|x_i) = 1 - \delta(x_j - x_i). \tag{9}$$

The product of pairwise exclusion potentials $\prod_{t \in S \setminus i} \psi_{F_t}(x_t|x_i)$ is zero only if the patch to be assigned to node i has already been used by another image node. This is different from the full joint potential $\psi_F(x_1, \ldots, x_N)$, which can be zero even if the patch to be placed at node i is not being shared with other image nodes. Combining (7)-(9), it gives

$$\begin{aligned} m_{fi}(x_i = l) &\approx \prod_{t \in S \setminus i} \sum_{x_t = 1}^{M} \psi_{F_t}(x_t|x_i = l) m_{tf}(x_t) \\ &= \prod_{t \in S \setminus i} (1 - m_{tf}(x_t = l)), \end{aligned} \tag{10}$$

where we assume that m_{tf} is normalized to 1. This is intuitively satisfying: The exclusion factor node f steers the node i to place low probability on claiming a patch l if that patch has already been claimed by another node with a high probability.

3.5 Implementation Details

3.5.1 The Compatibility Measure

In Cho et al. [1], sampled patches are nonoverlapping and the patch-to-patch compatibility is defined in terms of a natural image statistics prior. In this paper, we sample overlapping patches and compute the pairwise compatibility using the seam energy. using overlapping patches reduces visual artifacts.

Fig. 2 illustrates how we compute the left-right seam energy for two overlapping patches k and l. We first find the seam within the overlapped region—denoted by the red strip—along which the color difference between the two patches is minimum. We use a dynamic programming method described by Efros and Freeman [29] to find the optimal seam. The color difference along the optimal seam is the seam energy $E_{seam}(k, l)$ and we exponentiate it to compute the compatibility ψ:

$$\psi_{i,j}(k, l) \propto \exp\left(-\frac{E_{seam}(k, l)}{\sigma_c(l)^2} \right), \tag{11}$$

where $\sigma_c(l)$ is a parameter that controls how much we penalize finite seam energy with a reference patch l.

Note that E_{seam} is zero for two patches that were adjacent in the original image. This characteristic allows a greedy polynomial-time algorithm to reconstruct the original image. However, such a greedy algorithm does not generalize well to accommodate image editing operations, as an MRF-based algorithm does.

3.5.2 Setting $\sigma_c(l)$

In Cho et al. [1], $\sigma_c(l)$ in (11) is fixed for all pairs of patches and was set through cross validation. In this work, we adaptively set $\sigma_c(l)$ for every reference patch l. We define $\sigma_c(l)$ as follows:

$$\sigma_c(l) = E_l(2) - E_l(1), \tag{12}$$

where $E_l(1)$ is the seam energy between a patch l and its best match and $E_l(2)$ is the seam energy between a patch l and its second best match.

The motivation for this choice of $\sigma_c(l)$ is to simplify the image completion problem such that the algorithm considers only few patches as a plausible neighbor to a reference patch l, effectively reducing the combinatorial complexity.

3.5.3 User Input in the Image Model

We can incorporate the user input into the image model using the local evidence term. If a user fixes a patch k at an image node i, $p(y_i \mid x_i = k) = 1$ and $p(y_i \mid x_i = l) = 0$, for $\forall l \neq k$. At unconstrained nodes, the mean color of the original patch y_i serves as an observation:

$$p(y_i \mid x_i = l) \propto exp\left(-\frac{(m(y_i) - m(l))^2}{\sigma_{evid}^2} \right), \tag{13}$$

where $m(\cdot)$ is the mean color of the argument and $\sigma_{evid} = 0.4$ is determined through cross validation. Equation (13) allows the algorithm to keep the scene structure correct (i.e., sky at the top and grass at the bottom) and is used in all applications described in this section unless specified otherwise. While $m(\cdot)$ denotes the mean color in this work, we can modify this function to meet an application's needs.

4 Image Editing Applications

We introduce a number of image editing applications that leverage the patch transform. All the applications follow the same pipeline as shown in Fig. 1.

4.1 Image Reorganization

A user may want to change the location of an object after capturing an image. We show an example for relocating a person in Fig. 3. Fig. 3a is the original image, and the user wants to move the woman to the left side of the image. Fig. 3b shows how the user specifies the input. The user grabs patches that belong to the woman (the black bounding box) and snaps them at the desired location (the green bounding box). We have deliberately placed the woman to occlude two men in the background to show how two men are relocated automatically. The local evidence $p(y_i \mid x_i)$ for nodes where

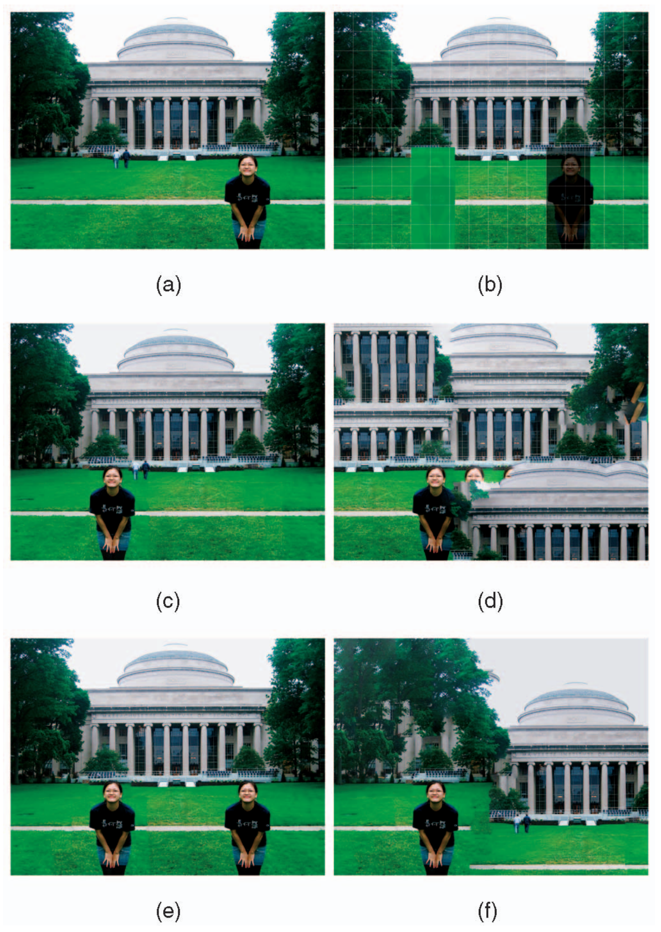

Fig. 3. Illustration of the contribution of each term in the image model. (a) The input image. (b) The user input. (c) The reconstructed image respecting the local evidence and the exclusion term. (d)-(f) Reconstructed images with and without the local evidence and the exclusion term. (d) Without the local evidence and without the exclusion term. (e) With the local evidence and without the exclusion term. (f) Without the local evidence and with the exclusion term.

the woman used to stand is now uniform over all x_i since the algorithm doesn't know a priori what to place there.

With this user input, the inverse patch transform finds a pleasant patch configuration and reconstructs an edited image (Fig. 3c). One observation from the reconstructed image is that two men in the background "got-out-of-the-way" and placed themselves at a new location. The inverse transform does not just swap patches that belong to the woman with patches that the woman is placed upon.

To see how each term in the image model contributes to the output image, we have conducted an experiment where the local evidence and the exclusion term are each turned on and off in the inverse patch transform. Fig. 3d shows the reconstructed image when the local evidence term is uniform (except in nodes with fixed patches) and the exclusion term is turned off. While the patch assignments are locally compatible, the final image does not have the same structure as the input image. If we incorporate the local evidence term while keeping the exclusion term off, we get a better structured image, shown in Fig. 3e, but the reconstructed image has duplicates of the woman in the foreground. If we only turn on the exclusion term and keep the local evidence uniform, we get the result shown in Fig. 3f. While there aren't any

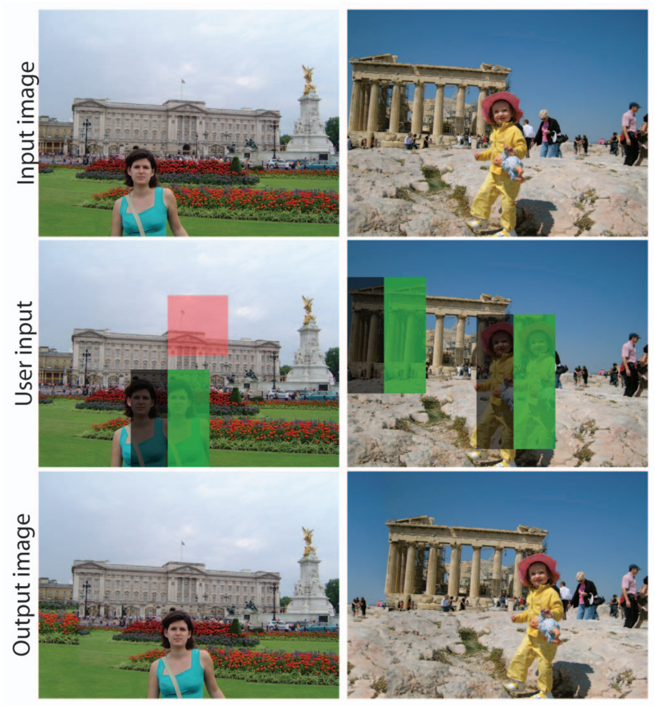

Fig. 4. Subject reorganization examples. (**Color codes:** red—fix, green—move to, black—move from.)

repeating patches that generate visual artifacts, the structure of the input image is not maintained. Fig. 4 has more examples of object reorganization.

4.2 Object Removal

The patch transform can be used to remove objects from an image: A user can reconstruct an image without patches that correspond to the object of interest. Since the exclusion term is not a hard constraint, the inverse patch transform will judiciously reuse some patches.

Fig. 5 shows some examples of object removal. In the first row, the user wants to remove large trees on the left side of the building. To make up for the missing tree patches, the algorithm chooses to reuse some patches of the building, propagating the building structure. In the second example,

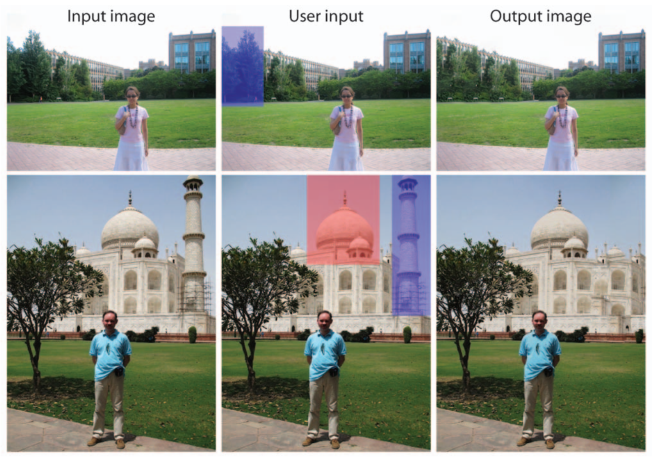

Fig. 5. The object removal examples. The inverse patch transform reconstructs an image discarding the user-specified patches. (**Color codes:** red—fix, blue—remove.)

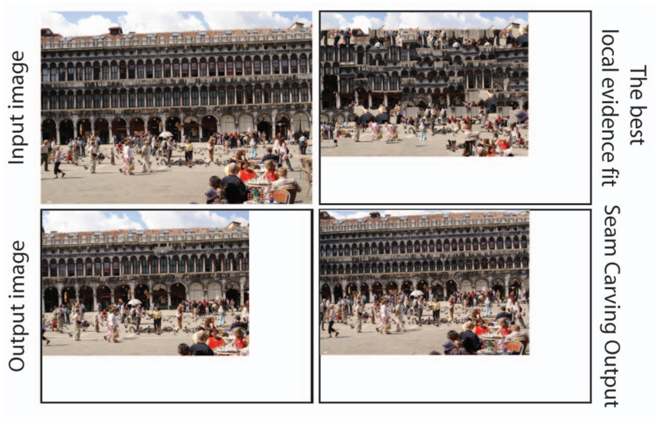

Fig. 6. The image retargeting example. The inverse patch transform reconstructs a resized image by solving for the patch assignment on a smaller canvas.

the user wants to remove the long tower under construction while keeping the dome at its current location. To complete the missing region, the inverse patch transform reuses some patches from the building.

4.3 Image Retargeting

A user may be interested in resizing an image while keeping as much content of the original image as possible. One effective method to retarget an image is "seam carving" by Avidan and Shamir [37]. A seam carving method finds a seam along which the energy is minimum, and removes the seam from an image. While it achieves excellent results on many images, the algorithm is inherently based on low-level cues, and sometimes it fails to retain the overall structure of the image.

We argue that the patch transform allows users to resize an image while keeping the structure of the image through the local evidence term. An image retargeting in the patch transform framework can be thought of as solving a jigsaw puzzle on a smaller canvas (leaving some patches unused). The local evidence on a smaller canvas is the low-resolution version of the bicubically resized image.

We show an image retargeting example in Fig. 6 in which the user wants to reduce the image size to be 80 percent in height and width. The patch assignment result just with the local evidence term is shown on the right. While the image contains many artifacts, the overall structure of the original image is maintained. After running belief propagation, we can generate a resized image: A whole floor of the building has been removed to fit the vertical size of the image, and some pillars have been removed to fit the lateral size of the image. At the same time, some pavement patches as well as some people have disappeared. When we retarget the image using Seam Carving [37], the scene can be well summarized, but the overall structure of the image is not maintained. Note that sky occupies a smaller portion of the whole image.

What makes retargeting work is that while the compatibility term tries to simply crop the original image, the local evidence term competes against that to retain as much information as possible. The inverse patch transform balances these competing interests to generate a retargeted image.

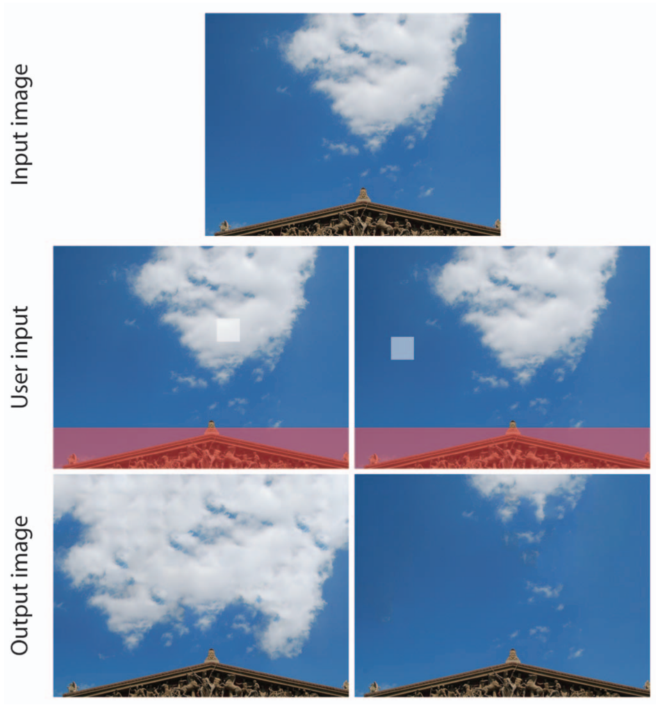

Fig. 7. The user can manipulate the patch statistics of an image through the image prior term, which can be designed to favor user specified patches. (**Color codes:** red—fix, white—favor.)

4.4 Manipulating Patch Statistics

The patch transform is well suited to controlling the amount of textures, or the patch statistics, in an image. One method of manipulating the patch statistics is by explicitly controlling how many patches of a certain class appear in an image. In this section, we present another method to control the patch statistics: by manipulating $p(x_i)$ in the image model (2).

Consider an input image in Fig. 7, and suppose that the user wants to have more clouds similar to a patch x_s. We can fold the patch preference information into $p(x_i)$:

$$p(x_i; x_s) \propto exp\left(-\frac{(f(x_i) - f(x_s))^2}{2\sigma_{sp}^2}\right), \tag{14}$$

where σ_{sp} is a specificity parameter and $f(\cdot)$ is a function that captures the characteristic of the patch. In this work, $f(\cdot)$ is the mean color of the argument, but can be defined to meet the application's needs.

The reconstructed image, with $\sigma_{sp} = 0.2$, is shown in the last row. Cloud patches are used multiple times: The energy penalty paid for using these patches multiple times is compensated for by the energy preference specified in the prior (14). We can also favor sky patches and generate an image consisting primarily of sky.

4.5 Photomontage

We introduce a photomontage application, where we mix patches from multiple images to generate a single image. A photographer may find it hard to capture the person and the desired background at the same time at a given shooting position. In this scenario, the user can take multiple images using different zooms and combine them in the patch domain. In Fig. 8, the user wants to transfer the large mountain from image 2 to the background of image 1. This

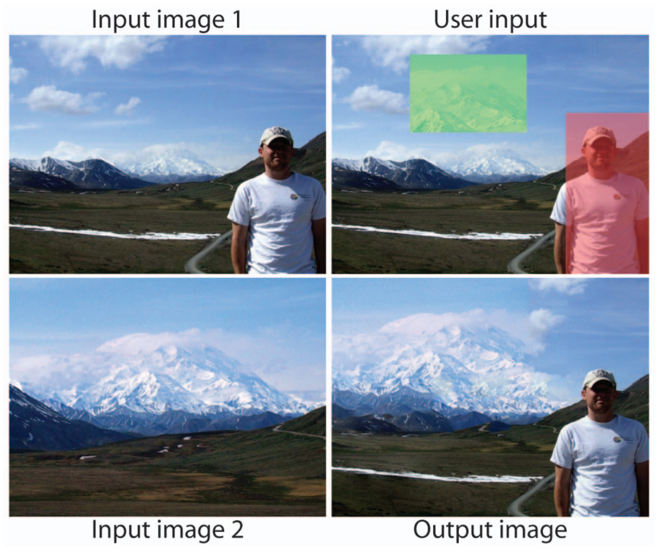

Fig. 8. We generate a photomontage of the two input images. The user wants to copy the large mountain in image 2 to the background of image 1, while keeping the person at the original location. (**Color codes:** red—fix, green—insert.)

operation is simple in the patch transform framework. The user specifies which portion of image 2 should be in the background of image 1 and what region should be affixed in image 1. Then the inverse patch transform reconstructs a plausible image using patches from both images. There is a region of contention that transitions from one image to the next. The inverse patch transform finds a good region to make such transitions.

5 The Fast Inverse Patch Transform

We have introduced a number of image editing applications that leverage the patch transform. In all examples, the input image was broken into 192 patches. We ran BP with two to three randomized initial conditions and chose the most visually pleasing result. With our relatively unoptimized MATLAB implementation on a 2.66 GHz CPU, 3 GB RAM machine, the compatibility computation

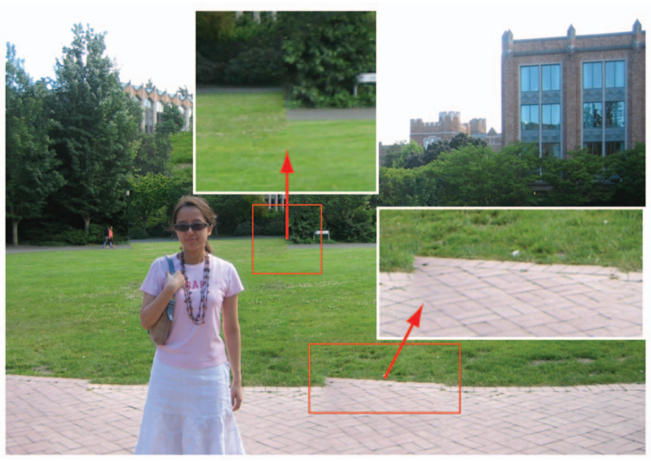

Fig. 9. Structural misalignments cause visible artifacts, motivating the use of smaller patches (Section 5) and the patch jittering algorithm (Section 6).

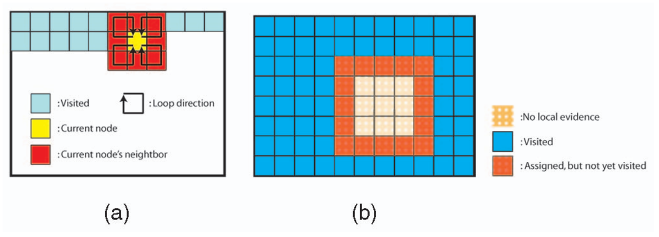

Fig. 10. Methods to manage the growth of node state dimensions. (a) The active labels are added by scanning the image nodes from the top-left corner to the bottom-right. (b) The active label assignment algorithm initially steps over the region without valid local evidence. Then the algorithm assigns active labels to these regions by propagating the active labels from the boundary nodes.

takes about 3 seconds, and belief propagation takes about 7 seconds for 50 message passing iterations.

One source of artifacts in the edited image is the structural alignment error, as shown in Fig. 9. We could reduce the patch size to suppress the structural misalignments. We cannot, however, indefinitely reduce the patch size because of the belief propagation complexity. The complexity of belief propagation algorithm for discrete probabilities is $O(NMK)$, where N is the number of nodes, M is the number of state labels per node, and K is the number of candidate neighbors per state. By reducing the patch size, we increase the number of patches as well as the number of image nodes, which slows down belief propagation significantly.

We present a patch-loop-based label pruning method that reduces the number of patch candidates per image node (i.e., M and K).

5.1 Pruning Patch Labels with Patch Loops

A patch loop is a sequence of patches that forms a loop on an MRF, as shown in Fig. 11a. We can form four types of loops given a reference patch (shown in blue), ignoring directionality. We name each loop LUR, LDR, RUL, RDL based on the order in which we traverse the neighboring nodes. For an image to be visually pleasing, the product of compatibility along the patch loop should be high. The patch loop label pruning "activates" patches that form highly compatible loops with patch candidates in neighboring nodes. This label pruning routine serves as a preprocessing step to belief propagation.

5.1.1 *Active Label Assignment Algorithm*

We assign active labels to image nodes by scanning them from the top-left corner to the bottom-right corner (Fig. 10a). At each node, say, the yellow node in Fig. 10a, we take the k most probable patches according to the local evidence, find l patch loops with maximal compatibility for each of the k chosen patches, and add patches that form patch loops as active labels at the corresponding locations in the neighboring nodes (shown in red). In the case of RDL loop (Fig. 11b), the algorithm finds k patches for node u, and adds active labels to i, j, and v in the order they form the patch loop.

When the user removes the patch at a node i, the local evidence at a node i is uniform for all patches, and the algorithm does not have any cue to select the initial k candidates. Therefore, if an image node has a uniform local evidence, the active label assignment algorithm initially steps over it and does not assign any active labels. Once the

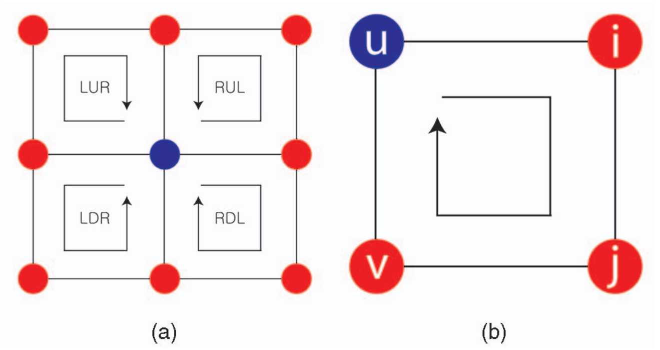

Fig. 11. (a) For a reference patch (shown in blue), there are four different types of neighboring loops (not considering the directions). (b) An RDL loop.

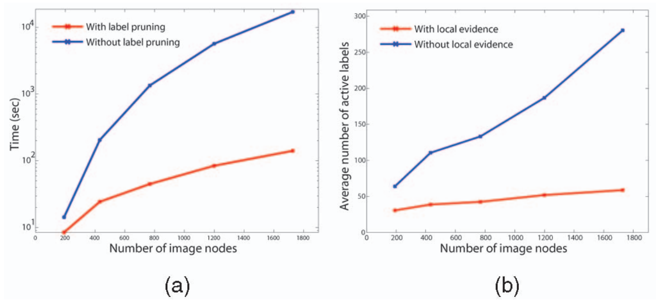

Fig. 12. (a) The runtime comparison with and without label pruning. Note that the y-axis is in log scale. (b) The average number of active labels in nodes with/without local evidence as we increase the total number of patches.

algorithm finishes the node scanning process, the algorithm propagates the active labels from the boundary of the patch removed region into the center of the patch removed region (Fig. 10b). We scan the uniform local evidence region from the top-left corner to the bottom-right corner with RDL loops, and reverse scan from the bottom-right corner to the top-left corner with LUR loops. At each node, for *every* active label (instead of the top k patch labels according to the local evidence), we use l^* patch loops (where l^* can be different from l) to assign active labels at the corresponding neighboring nodes. The initial active labels at node i are patches added by its neighbors before the node scanning reaches the node i. In practice, $l^* = 1$ performs well.

5.1.2 *Finding l Best Patch Loops*

We could find the optimal l patch loops for a reference patch u by enumerating all N^3 loop candidates, where N is the total number of patches. We introduce an $O(N^2)$ algorithm that finds l approximately best patch loops for a fixed reference patch u.

Consider Fig. 11b. Let $cH(\zeta, \eta)$ be the seam energy for placing ζ to the left of η, and $cV(\zeta, \eta)$ be the seam energy for placing ζ to the top of η. The following algorithm finds patch labels i^*, j^*, and v^* that minimize the seam energy across the loop:

1. Find the patch label i that minimizes $E_i(j) = \min_i(cH(u,i) + cV(i,j))$ for each j.
2. Find the patch label j for fixed i that minimizes $E_j(v) = \min_j(E_i(j) + cH(v,j) + cV(u,v))$ for each v.
3. Find l patches v^* that minimize $E_j(v)$ with the corresponding j.
4. Find patches i^*, j^* for each v^* through back tracing: $j^* = \arg\min_j(E_j(v^*))$, $i^* = \arg\min_i(E_i(j^*))$.

In other words, for each reference patch u, the algorithm returns l approximately best patch loops by first sorting $E_j(v)$, picking the l best patches v^* that minimize $E_j(v)$, and returning i^*, j^* with the corresponding v^* through back tracing. We can precompute the patch loops. The algorithm is applicable for all other loops (i.e., RUL, LUR, LDR) with minor modifications.

5.1.3 *Discussions*

The patch-loop-based label pruning algorithm is closely related to label pruning algorithms for belief propagation. Label pruning has been successful in accelerating belief propagation in many applications, including pose estimation [38], object recognition [39], image analysis [40], stereo depth map inference [41], and in medical applications [42]. On a theoretical note, Bishop et al. [43] and Koller et al. [44] propose effective label pruning algorithms for MRFs with potentials from an exponential family. Freeman et al. [31] propose a method to retain state labels that have high potential to be the final solution. The state label selection can be thought of as maintaining only the top k state labels with the highest probability before running belief propagation. While such a first-order label pruning scheme works well in many applications, it breaks down in the inverse patch transform. In the inverse patch transform, there should be at least one state label at each node that its four neighboring nodes can agree on. We call this a consensus constraint and the first-order label pruning method may not satisfy the consensus constraint. The patch loop label pruning algorithm addresses this issue by taking into account the active labels in the neighboring nodes.

The patch loop-based label pruning scheme is static: The number of patch labels does not change as belief propagation proceeds. A dynamic label pruning method, introduced by Komodakis and Tziritas [34], reduces the number of state labels as belief propagation runs. This would be too expensive for the inverse patch transform since it should make sure that the consensus constraint is satisfied after each BP iteration.

5.2 Evaluation

We experimentally show that reducing the patch size improves the image editing quality. We show that the label pruning step is important to reduce the computation time. The label pruning scheme reduces the runtime per iteration as well as the total number of required iterations for convergence.

5.2.1 *Runtime Comparison*

We have performed the image reconstruction task by breaking the image into 192, 432, 768, 1,200, and 1,728 patches. The corresponding patch sizes are 76×76, 56×56, 46×46, 40×40, and 36×36, all with 16 pixel overlap. We recorded the time it takes to run 50 iterations of belief propagation with these patches, with and without the label pruning. $k = 6, l = 2$ in these experiments. The number of patch loops for nodes without valid local evidence l^* was 1.

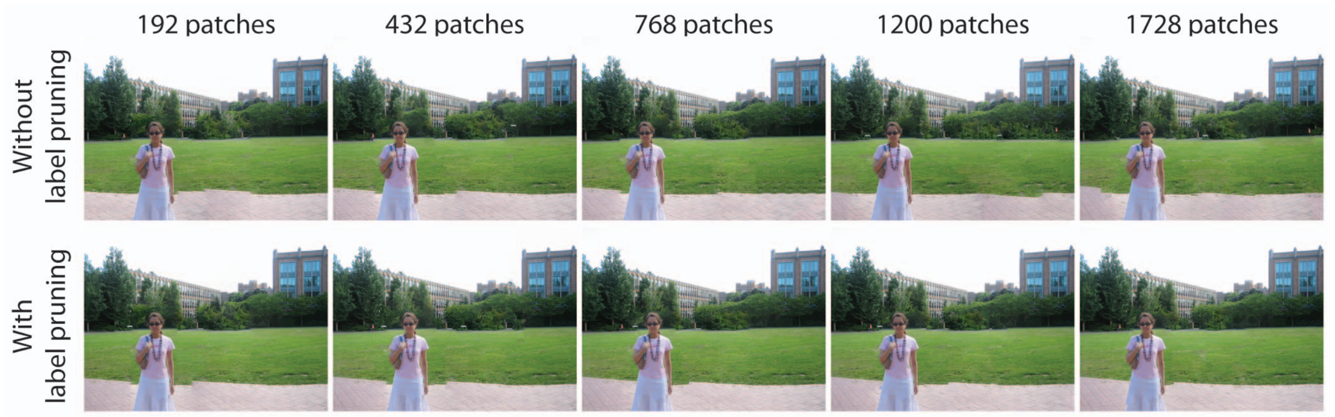

Fig. 13. Reducing the patch size improves the edited image quality. The label pruning helps find visually pleasing local minima with a small number of message passing iterations.

The blue plot in Fig. 12a shows the runtime of 50 message passing iterations without label pruning. Note that the image reconstruction slows down rapidly as the number of image nodes (thus, the number of patches) increases. When there are 1,728 patches, it takes nearly 5 hours to run 50 message passing iterations.

The red plot in Fig. 12a shows the runtime of 50 message passing iterations with label pruning. Even with 1,728 patches, it takes about 2.5 minutes to complete 50 message passing iterations. While the runtime is less than the case without label pruning, the complexity of BP after label pruning is not linear in the number of patches because the number of active labels in image nodes increases roughly linearly to the number of patches.

Fig. 12b shows the average number of active labels in nodes with and without local evidence as we increase the total number of patches. These plots show that the number of active labels increases approximately linearly in the total number of patches. In the case of nodes with local evidence, the proportionality constant is roughly 0.01, and in the case of nodes without local evidence, the proportionality constant is roughly 0.3. This shows that the bulk of the BP runtime is spent in regions without local evidence.

The belief propagation runtime reduction comes at a cost of precomputing patch loops. The patch loop computation takes 8 minutes with a mex code implementation in MATLAB. Note that we only need to compute the patch loop once before running belief propagation, and this delay is not visible to the user during the image editing process.

5.2.2 Image Quality Comparison After 50 Iterations of BP

The benefit of label pruning is not just the per-iteration message passing runtime reduction, but also the reduction in the number of message passes for convergence. We show that 50 message passing iterations are enough, even with 1,728 patches, to reconstruct a plausible image with label pruning, but is not enough to reconstruct a plausible image without label pruning.

Fig. 13 shows reconstructed images with/without label pruning after 50 iterations of belief propagation using 192, 432, 768, 1,200, and 1,728 patches. Without label pruning, reconstructed images contain repeating-patch artifacts. Also, reconstructed images contain severe structural misalignments because belief propagation did not converge to a stable solution after 50 iterations of belief propagation.

With label pruning, with $k = 6, l = 2$, artifacts due to the patch repetition are unnoticeable, and the structural misalignments are less salient. The label pruning not only helps in reducing the runtime per message passing iteration, but also in reducing the number of message passing iterations.

k and l are knobs to balance the fast convergence of belief propagation and the ability to explore the possible space of images that can be generated with a given set of patches. If k and l are small, messages will converge faster, but the resulting image would look similar to the original image. If k and l are large, belief propagation would require more message passing iterations to converge, but the algorithm will explore more space of plausible images. The algorithm is insensitive to the exact k, l values. We experimentally verified that the algorithm performs well if $4 \leq k \leq 8, 1 \leq l \leq 4$.

We show some more images edited with the accelerated patch transform. Each image is broken into patches of size 36×36, with 16 pixel overlap, generating about 1,700 patches per image. A rough breakdown of the processing time is shown in Fig. 14. Images in Fig. 15 are generated by running the inverse patch transform three times and taking the visually most pleasing image.

In Fig. 16, we show some failure examples. Since the patches are smaller, the patch transform sometimes finds it hard to transfer a whole object intact. In the top row of Fig. 16, the right bench should have reappeared on the left side of the image, but it reappeared only partially. Also, if different regions have small color difference (the middle row of Fig. 16), the different textures may try to merge.

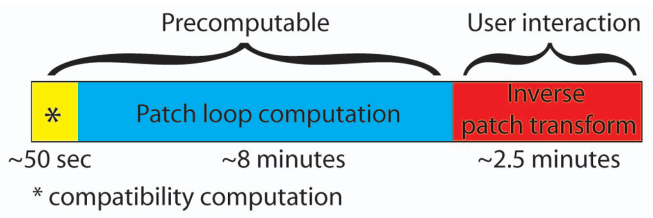

Fig. 14. The runtime breakdown for 1,728 patches. The runtime is measured in MATLAB.

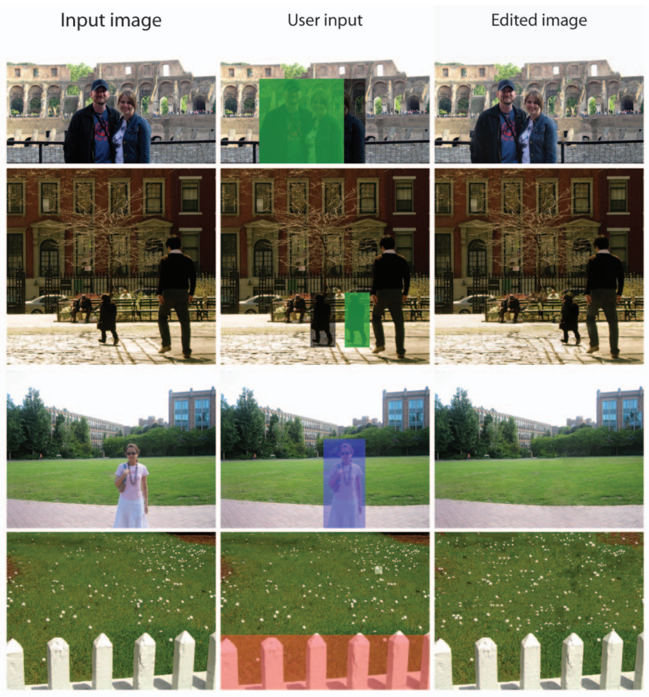

Fig. 15. More image editing examples. (**Color codes:** red—fix, green—move to, black—move from, blue—remove, white—favor.)

6 Patch Jittering

While the use of small patches, enabled by the label pruning method described above, drastically improves the edited image quality, subpatch size structural misalignments still generate visual artifacts. We present a method to suppress subpatch size structural misalignments through a postprocessing step called the patch jittering. The patch jittering operation refers to finding an optimal spatial offset to patches assigned to image nodes by the inverse patch transform. We can jitter patches without generating a hole in the output image since patches overlap. The maximum amount of jitter is determined by the amount of patch overlap.

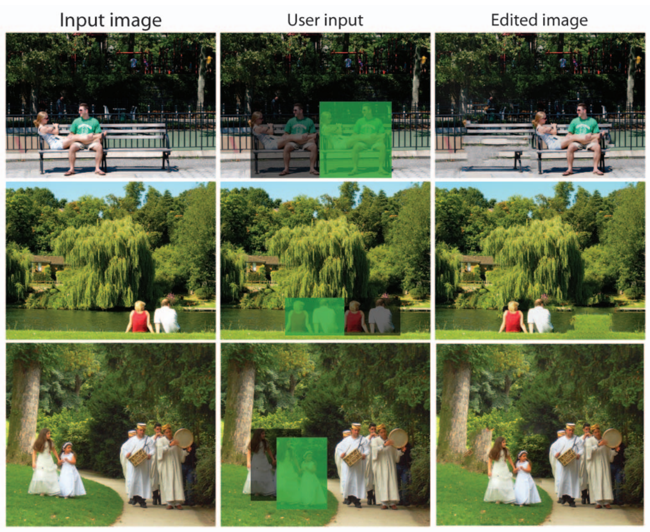

Fig. 16. Some failure examples. The patch transform sometimes fails to preserve structures. (**Color codes:** green—move to, black—move from.)

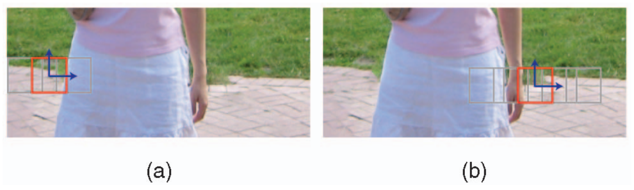

Fig. 17. A local jittering scheme can fix some types of misalignments, but has limitations, motivating the use of an MRF to jitter patches globally. (a) The misalignment between the red patch and the patch to its right can be reduced by jittering the red patch toward the right. (b) A simple local jitter operation cannot fix the misalignment error for this example. Moving the red patch upward would better fit the side on the right, but it would cause misalignment with a patch on its left.

6.1 A Local Jittering Scheme

Fig. 17a illustrates the jittering operation. The gray overlapping squares denote the underlying patch grid and the red square is the patch that we jitter. By moving the red patch to the right, the red patch better fits the patch to its right: The "dip" between the red tile and the white dress will disappear. The increase in the seam energy between the red patch and the patch to its left will not be drastic since they are self-similar. Essentially, we are reusing some pixels in the overlapped region to mask alignment errors.

Jittering each patch one by one has a limitation. Consider Fig. 17b. By jittering the red patch upward, the red patch will better fit the patch to its right. However, the upward jitter causes a misalignment with the patch to its left, and increases the total seam energy between the red patch and its four neighbors. Therefore, we need to jitter neighboring patches in unison to find a global optimal jitter at every node.

6.2 The Jitter MRF

Our approach to globally jitter patches is to use another MRF. We call this new MRF the jitter MRF. Once the inverse patch transform assigns patches to image nodes, we formulate a jitter MRF, where state labels at each node are jittered versions of the assigned patches. Then belief propagation on the jitter MRF searches for the globally optimal jittered patch at each node. When computing the seam energy, we normalize the sum of color difference along the seam by the length of the overlapped region because the size of the overlapped regions between patches changes as we jitter them. This allows us not to unnecessarily penalize the no-jitter state.

In many cases, only a small portion of the image contains structural misalignment artifacts that need to be fixed through jittering. While it is easy to spot the misalignment artifacts using pairwise compatibility, it is hard to find a region that should be jittered. Since the patch jittering is a postprocessing step, we ask users to specify regions that need jittering. In our implementation, each patch is jittered in steps of $[-6, -3, 0, 3, 6]$ pixels to reduce the computational overhead.

The patch jittering operation proceeds as follows:

1. The user selects regions that need jittering.
2. The algorithm computes the compatibility among neighboring patches with all possible jitters.

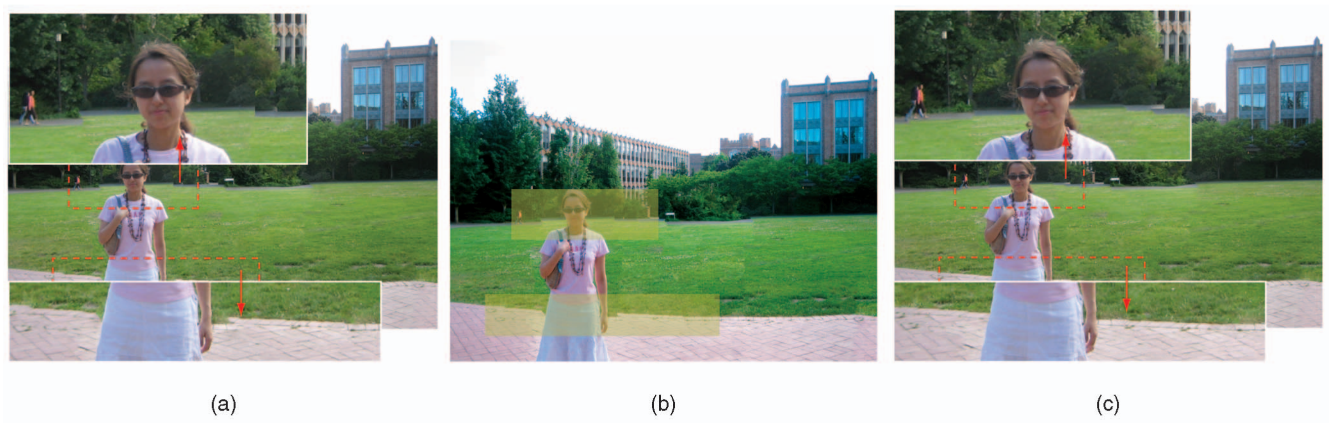

Fig. 18. Illustration of the effectiveness of the patch jittering operation. (a) An edited image before the patch jittering. (b) A user input for the patch jittering. (c) An edited image after the patch jittering.

3. The algorithm formulates an MRF, where the state variable at each node is the amount of jitter, and runs belief propagation to find the optimal jitter at each node. The MRF is formed only on nodes that the user selected.

6.3 Evaluation

We show the patch jittering operation in Fig. 18a. The inset shows that the edited image before the patch jittering contains visible alignment artifacts. Fig. 18b shows how the user specifies the region to be jittered and Fig. 18c shows the final edited image after the patch jittering. It takes about one second to set up the jitter MRF (computing the pairwise compatibility) and less than 0.5 second to run 20 iterations of belief propagation.

The jittering operation removes much of the structural alignment artifact. The alignment error along the black pavement is removed by gradually jittering patches upward to connect the pavements. Some grass pixels are used twice and some bush pixels have disappeared. To fix the alignment error along the red pavement to the left side of the woman, the algorithm jitters the patch that abuts the woman's dress. To fix the alignment error to the right side of the woman, the jitter MRF shortens her wrist and reuses some finger pixels.

The jittering operation has a few local minima since most of the jittered patches are compatible with its neighbors. We explore some local minima by randomly initializing BP messages. Fig. 19 shows two other local minima of the patch jittering operation to user input Fig. 18b. Since running 20 iterations of belief propagation takes less than 1 second, the user could try different message initializations to explore different local minima, and choose the best one.

Fig. 20 shows more patch jittering examples. The patch jittering on the first example removes the alignment error along the staircase, and the jittering on the second example removes the residual error from the water bottle.

7 Conclusions

We have presented the patch transform and its applications to image editing. The patch transform allows users to

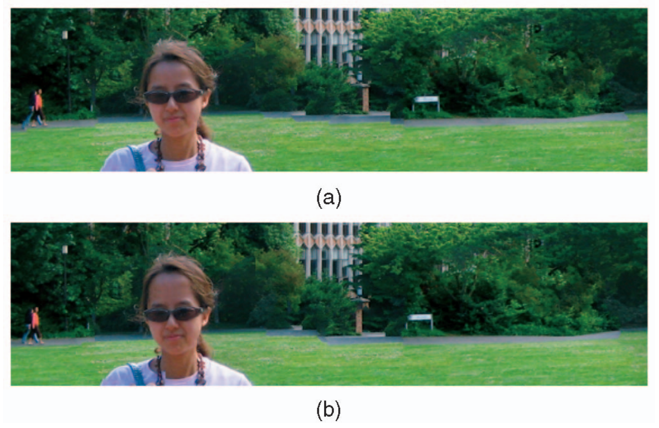

Fig. 19. Two other solutions to the jittering operation in Fig. 18. In (a), tree pixels are reused to align the background, whereas in (b), the forehead pixels are reused.

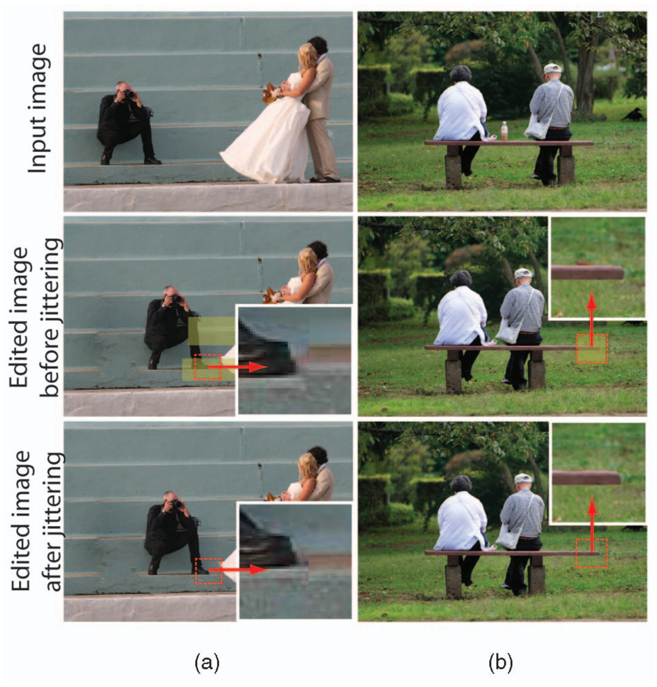

Fig. 20. More patch jittering examples. Note that in (a), the alignment error in staircases is fixed, and in (b), the residual error from the bottle is fixed.

manipulate images in the patch domain, and the inverse patch transform reconstructs an image that conforms to the user input. The inverse patch transform reconstructs an image from a bag of patches by considering three elements: 1) Neighboring patches should be compatible to one another so that visible artifacts can be minimized, 2) each patch should be used only once in generating the output image, and 3) the user-specified changes in the patch domain should be reflected in the reconstructed image. We model the inverse patch transform as solving for patch labels on an MRF, where each node denotes a location where we can place patches. We introduce various image editing applications that leverage the patch transform, and verify that the patch transform framework works well with real images. To further improve the edited image quality, we introduce the patch loop-based label pruning and the patch jitter-based structure misalignment correction. These improvements give the appearance of making object-level manipulations, while using the low-level computations of the patch transform.

ACKNOWLEDGMENTS

This research is partially funded by ONR-MURI grant N00014-06-1-0734 and by Shell Research. This work was partially completed while Taeg Sang Cho was an intern at Adobe Systems. Taeg Sang Cho is partially supported by the Samsung Scholarship Foundation. The authors would like to thank Myung Jin Choi, Ce Liu, Anat Levin, and Hyun Sung Chang for fruitful discussions. They would also like to thank Flickr for images.

REFERENCES

[1] T.S. Cho, M. Butman, S. Avidan, and W.T. Freeman, "The Patch Transform and Its Applications to Image Editing," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.
[2] E.D. Demaine and M.L. Demaine, "Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity," *Graphs and Combinatorics,* vol. 23, pp. 195-208, 2007.
[3] M.G. Chung, M.M. Fleck, and D.A. Forsyth, "Jigsaw Puzzle Solver Using Shape and Color," *Proc. Int'l Conf. Signal Processing,* 1998.
[4] T.R. Nielsen, P. Drewsen, and K. Hansen, "Solving Jigsaw Puzzles Using Image Features," *Pattern Recognition Letters,* vol. 29, pp. 1924-1933, 2008.
[5] B.J. Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. Dobkin, A. Vlachopoulos, C. Doumas, S. Rusinkiewicz, and T. Weyrich, "A System for High-Volume Acquisition and Matching of Fresco Fragments: Reassembling Theran Wall Paintings," *Proc. ACM SIGGRAPH,* 2008.
[6] H.C. da Gama Leitao and J. Stolfi, "A Multiscale Method for the Reassembly of Two-Dimensional Fragmented Objects," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, no. 9, pp. 1239-1251, Sept. 2002.
[7] D. Koller and M. Levoy, "Computer-Aided Reconstruction and New Matches in the Forma Urbis Romae," *Bullettino Della Commissione Archeologica Comunale di Roma,* 2006.
[8] C.-S. Wang, "Determining Molecular Conformation from Distance or Density Data," PhD dissertation, Massachusetts Inst. of Technology, 2000.
[9] M. Levison, "The Computer in Literary Studies," *Machine Translation,* pp. 173-194, 1967.
[10] L. Zhu, Z. Zhou, and D. Hu, "Globally Consistent Reconstruction of Ripped-Up Documents," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 30, no. 1, pp. 1-13, Jan. 2008.
[11] Y.-X. Zhao, M.-C. Su, Z.-L. Chou, and J. Lee, "A Puzzle Solver and Its Application in Speech Descrambling," *Proc. 2007 WSEAS Int'l Conf. Computer Eng. and Applications,* 2007.
[12] N. Jojic, B.J. Frey, and A. Kannan, "Epitomic Analysis of Appearance and Shape," *Proc. IEEE Int'l Conf. Computer Vision,* 2003.
[13] A. Kannan, J. Winn, and C. Rother, "Clustering Appearance and Shape by Learning Jigsaws," *Advances in Neural Information Processing Systems 19,* MIT Press, 2006.
[14] A. Criminisi, P. Pérez, and K. Toyama, "Object Removal by Exemplar-Based Inpainting," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2003.
[15] Y. Wexler, E. Shechtman, and M. Irani, "Space-Time Video Completion," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2004.
[16] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, "Image Completion with Structure Propagation," *Proc. ACM SIGGRAPH,* 2005.
[17] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T.-T. Wong, "Solid Texture Synthesis from 2D Exemplars," *Proc. ACM SIGGRAPH,* 2007.
[18] E.N. Mortensen and W.A. Barrett, "Intelligent Scissors for Image Composition," *ACM Trans. Graphics,* 1995.
[19] J.-F. Lalonde, D. Hoiem, A.A. Efros, C. Rother, J. Winn, and A. Criminisi, "Photo Clip Art," *ACM Trans. Graphics,* 2007.
[20] P. Pérez, M. Gangnet, and A. Blake, "Poisson Image Editing," *ACM Trans. Graphics,* vol. 22, pp. 313-318, 2003.
[21] J. Wang and M.F. Cohen, "An Iterative Optimization Approach for Unified Image Segmentation and Matting," *Proc. IEEE Int'l Conf. Computer Vision,* 2005.
[22] A. Levin, A. Rav-Acha, and D. Lischinski, "Spectral Matting," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.
[23] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani, "Summarizing Visual Data Using Bidirectional Similarity," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.
[24] C. Rother, L. Bordeaux, Y. Hamadi, and A. Blake, "Autocollage," *ACM Trans. Graphics,* 2006.
[25] M. Brown and D. Lowe, "Recognising Panoramas," *Proc. IEEE Int'l Conf. Computer Vision,* 2003.
[26] J.D. Bonet, "Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images," *ACM Trans. Graphics,* 1997.
[27] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image Inpainting," *ACM Trans. Graphics,* 2000.
[28] A.A. Efros and T.K. Leung, "Texture Synthesis by Non-Parametric Sampling," *Proc. IEEE Int'l Conf. Computer Vision,* 1999.
[29] A.A. Efros and W.T. Freeman, "Image Quilting for Texture Synthesis and Transfer," *Proc. ACM SIGGRAPH,* 2001.
[30] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-Time Texture Synthesis by Patch-Based Sampling," *ACM Trans. Graphics,* vol. 20, no. 3, pp. 127-150, July 2001.
[31] W.T. Freeman, E.C. Pasztor, and O.T. Carmichael, "Learning Low-Level Vision," *Int'l J. Computer Vision,* vol. 40, pp. 25-47, 2000.
[32] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut Textures: Image and Video Synthesis Using Graph Cuts," *Proc. ACM SIGGRAPH,* 2003.
[33] G. Ramanarayanan and K. Bala, "Constrained Texture Synthesis via Energy Minimization," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, no. 1, pp. 167-178, Jan./Feb. 2007.
[34] N. Komodakis and G. Tziritas, "Image Completion Using Efficient Belief Propagation via Priority Scheduling and Dynamic Pruning," *IEEE Trans. Image Processing,* vol. 16, no. 11, pp. 2649-2661, Nov. 2007.
[35] A. Agrawal, R. Raskar, and R. Chellappa, "What Is the Range of Surface Reconstructions from a Gradient Field?" *Proc. European Conf. Computer Vision,* 2006.
[36] J.S. Yedidia, W.T. Freeman, and Y. Weiss, "Understanding Belief Propagation and Its Generalizations," *Exploring Artificial Intelligence in the New Millennium,* pp. 239-269, http://portal.acm.org/citation.cfm?id=779352, Morgan Kaufmann Publishers, 2003.
[37] S. Avidan and A. Shamir, "Seam Carving for Content-Aware Image Resizing," *ACM Trans. Graphics,* vol. 26, 2007.
[38] J.M. Coughlan and S.J. Ferreira, "Finding Deformable Shapes Using Loopy Belief Propagation," *Proc. European Conf. Computer Vision,* 2002.
[39] M.P. Kumar and P. Torr, "Fast Memory-Efficient Generalized Belief Propagation," *Proc. European Conf. Computer Vision,* 2006.
[40] J. Lasserre, A. Kannan, and J. Winn, "Hybrid Learning of Large Jigsaws," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.

[41] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo Matching Using Belief Propagation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 25, no. 7, pp. 787-800, July 2003.
[42] M. Pechaud, R. Keriven, T. Papadopoulo, and J.-M. Badier, "Combinatorial Optimization for Electrode Labeling of Eeg Caps," *Medical Image Computing and Computer-Assisted Intervention,* Springer, 2007.
[43] C.M. Bishop, D. Spiegelhalter, and J. Winn, "Vibes: A Variational Inference Engine for Bayesian Networks," *Proc. Conf. Neural Information Processing Systems,* 2003.
[44] D. Koller, U. Lerner, and D. Angelov, "A General Algorithm for Approximate Inference and Its Application to Hybrid Bayes Nets," *Proc. Ann. Conf. Uncertainty in Artificial Intelligence,* 1998.

Taeg Sang Cho received the BS degree in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology in 2005, and the SM degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 2007, where he is currently working toward the PhD degree in the field of computer vision and computer graphics. His current research focuses on developing computational cameras and algorithms to enhance photographic images. He is the recipient of the 2007 AMD/CICC Student Scholarship Award, the 2008 DAC/ISSCC Student Design Contest Award, and the 2008 IEEE CVPR Best Poster Paper Award. He is a recipient of the Samsung scholarship. He is a student member of the IEEE.

Shai Avidan received the PhD degree from the School of Computer Science at the Hebrew University, Jerusalem, Israel, in 1999. He is a professor at Tel-Aviv University, Israel. He was a postdoctoral researcher at Microsoft Research, a project leader at MobilEye, a start-up company developing camera-based driver-assisted systems, a research scientist at Mitsubishi Electric Research Labs (MERL), and s senior research scientist at Adobe Systems. He has published extensively in the fields of object tracking in video sequences and 3D object modeling from images. Recently, he has been working on the Internet vision applications such as privacy preserving image analysis, distributed algorithms for image analysis, and media retargeting, the problem of properly fitting images and video to displays of various sizes. He is a member of the IEEE.

William T. Freeman received the PhD degree from the Massachusetts Institute of Technology in 1992. He is a professor of electrical engineering and computer science at the Massachusetts Institute of Technology (MIT), working in the Computer Science and Artificial Intelligence Laboratory (CSAIL). He has been on the faculty at MIT since 2001. He is also a principal scientist at Adobe's Advanced Technology Lab, where he works part-time. From 1992 to 2001, he worked at Mitsubishi Electric Research Labs (MERL), Cambridge, Massachusetts. Prior to that, he worked at Polaroid Corporation, and in 1987-1988, was a foreign expert at the Taiyuan University of Technology, China. His research interests include machine learning applied to problems in computer vision and computational photography. He is a fellow of the IEEE.