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 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 a modified image. Possible modifications in the patch domain 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 the use of a label pruning method that finds loops of patches that are likely to fit together. In addition, structural misalignment artifacts are supressed through a patch jittering scheme that spatially shifts the assigned patches by a sub-patch size. We demonstrate the patch transform and its effectiveness on natural images.

Index Terms

I. 4. 8. c. Image models, I. 4. 10. d. Statistical, I. 4. 9. Applications, I. 3. 3. e Image-based rendering

✦

EDICS Category: 3-BBND

- *T. S. Cho is with Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, 02139. E-mail: taegsang@mit.edu*
- *S. Avidan is with Adobe Systems Incorporated, Auburndale, 02466.*
- *W. T. Freeman is with Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, 02139, and with Adobe Systems Incorporated, Auburndale, 02466.*

Manuscript received XX XX, 2008; revised XX XX, 2008.

The patch transform

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 on the image grid 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. The inverse patch transform is solved by formulating a Markov random field (MRF) on image nodes, and using loopy belief propagation to solve for the patch label at each image node.

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

Image edits, such as hole filling, texture synthesis, object removal and image retargeting have already been successfully addressed in 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 carefuly 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 and place them in their new location. The inverse patch 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 non-overlapping patches considered in [1], to enhance the compatibility measure and reduce visual artifacts. Secondly, we introduce a new pruning technique, termed the patch loop based label pruning, to reduce the complexity of belief propagation from $O(N^3)$ to sub-quadratic 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 sub-patch size structural alignment error.

We describe related work in Section 2, and introduce the patch transform based image editing framework in Section 3. Then we develop the inverse patch transform algorithm and implementation details in Section 4, and introduce several image editing applications leveraging the patch transform in Section 5. To handle thousands of patches and improve edited image quality, we introduce a label pruning method and a patch jittering scheme in Section 6 and Section 7.

2 Related work

The inverse patch transform is closely related to solving jigsaw puzzles. Some types of jigsaw puzzles were shown to be NP-complete by Demaine and Demaine [2] because they can be reduced to the Set Partition Problem. Nevertheless, there has been much work in the literature to (approximately) solve the problem, and for jigsaws with discriminative shapes, we can prove that a polynomial algorithm solves the puzzle. Image jigsaw puzzles can be solved by exploiting the 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] used 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 relics 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 introduced in the literature 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. A patch based propagation method was introduced by Criminisi *et al.* [14], which was augmented with a global optimization framework by the follow-up papers [15], [16]. However, the global methods allow the same patch to be used multiple times, which limits the amount of control a user has over the synthesis. Also, it may result in tiling the same bland patch across the image, which would look unpleasant. This issue was addressed in Kopf *et al.* [17] by keeping a counter for each patch and reducing the probability of re-using a patch accordingly. We address this issue in a probabilistic framework by introducing a term in the image model that penalyzes 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 side-step 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 retargetting. 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 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's closely related to larger spatial scale versions of that task, including Digital Tapestry [24], Auto Collage [25] and panorama stitching [26], although with different goals. Some techniques employed in Digital Tapestry [24] are similar to this work, although Digital Tapestry focuses more on synthesizing a single summary image from multiple input images and does not require the exclusion constraint. Non-parametric texture synthesis algorithms, such as [27], and image filling-in, such as [14], [28], [29], 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], [30], which also require compatibility measures between patches. Efros and Freeman [30] and Liang *et al.* [31] used overlapping patches to synthesize a larger texture image. Neighboring patch compatibilities were found through squared difference calculations in the overlap regions. Freeman, Pasztor and Carmichael [32] used similar patch compatibilities, and used loopy belief propagation in an MRF to select image patches from a set of candidates. Kwatra *et al.* [33], and Komodakis and Tziritas [34] employed related Markov random field models, solved using graph cuts or belief propagation, for texture synthesis and image completion. The most salient difference from all 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.

The patch loop based label pruning algorithms are related to numerous label pruning algorithms for belief propagation. Label pruning has been useful in accelerating belief propagation in many applications, including pose estimation [35], object recognition [36], image analysis [37], stereo depth map inference [38], and in medical applications [39]. On a theoretical note, Bishop *et al.* [40] and Koller *et al.* [41] proposed effective label pruning algorithms for MRFs with potentials from an exponential family. Freeman *et al.* [32] proposed a method to select 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 highest probability before running belief propagation. While such first order label pruning scheme works well in many applications, it may break down in the inverse patch transform: in inverse patch transform, there should be at least one state label per node that its four neighboring nodes can agree on. We call this a **concensus constraint**, and the first order label pruning does not guarantee a concensus constraint.

Komodakis *et al.* [34] proposed a dynamic label pruning method that reduces the number of state labels as belief propagation runs: a state label is discarded if its marginal probability is lower than a certain threshold. 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 would be too expensive for the inverse patch transform since it should make sure that the concensus constraint is satisfied after each BP iteration.

3 The patch transform image editing framework

We introduce the image editing framework (Fig. 1) leveraging the patch transform. Given an input image, the system samples overlapping patches from a regular grid, each with 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 given the user input 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 on the MRF to solve for the patch assignment that is visually pleasing while satisfying user inputs.

Once patches are assigned to the image nodes, we stitch patches together in a way similar to Efros and Freeman [30]: 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

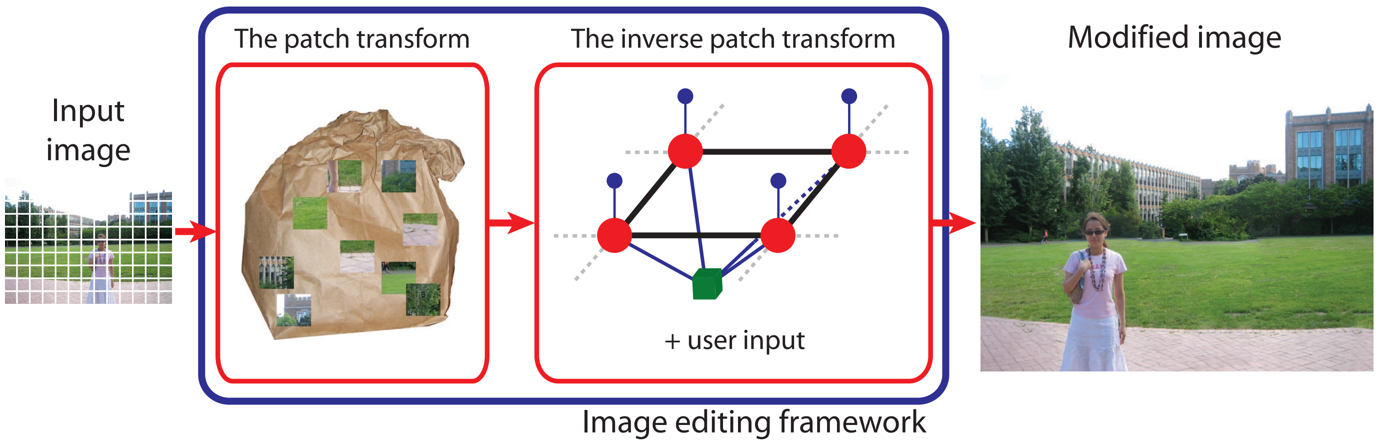

Fig. 1. This figure illustrates the pipeline of the patch transform based image editing scheme. First, we compute the patch transform representation of the input image, and use loopy BP on MRF to solve for the patch labels on the image grid, respecting the user input, compatibility requirement, and the 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 use each patch seldom more than once.

neighboring patches were not adjacent in the original image. Thus, we remove the intensity gradients along the seam if two patches that generated the seam were not adjacent in the original image. We use the poisson solver provided by Agrawal *et al.* [42] to generate an image with suppressed seam artifacts. Section 4 introduces the inverse patch transform algorithm and implementation details.

4 The inverse patch transform

4.1 The image model

The unknown state at the i^{th} image node is the index of the patch x_i to be placed at that position. Based on how plausibly one patch fits next to another, we define a compatibility, ψ. 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 $\psi_{i,j}(k,l)$.

We let $\mathbf{x}$ be a vector of the unknown patch indices x_i at each of the N image positions i. We define the probability of an assignment, $\mathbf{x}$, of patches to image positions to be

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

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. The user's constraints on patch positions are represented by a local evidence term, $\phi_i(x_i)$. We show later that $\phi_i(x_i)$ is also used to aid the image reconstruction.

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 $P(\mathbf{x})$ in Eq. (1).

4.2 Approximate solution by belief propagation

Finding the assignment $\mathbf{x}$ that maximizes $P(\mathbf{x})$ in the MRF of Eq. (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 [43]. For belief propagation applied in networks with loops, different factorizations of the MRF joint probability can lead to different results. Interestingly, we found better results for solving the patch assignments using an alternative factorization of Eq. (1) as a directed graph.

To derive a directed graph image model, we define a normalized compatibility,

$$p_{i,j}(x_i|x_j) \approx \frac{\psi_{i,j}(x_i, x_j)}{\sum_{i=1}^{M} \psi_{i,j}(x_i, x_j)} \tag{2}$$

and the local evidence term $p(y_i|x_i) = \phi_i(x_i)$. We can then approximate the joint probability of Eq. (1) in terms of conditional probabilities as

$$P(\mathbf{x}) \propto \prod_{i=1}^{N} \prod_{j \in \mathcal{N}(i)} p(y_i|x_i) p_{i,j}(x_j|x_i) p(x_i) E(\mathbf{x}) \tag{3}$$

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. This expression is exact for a tree, but only approximate for a MRF with loops. A similar factorization for an MRF was used in [32]. We can manipulate $p(x_i)$ to steer the MRF to favor patches with certain characteristics, but in most cases we model $p(x_i)$ as a uniform distribution.

The approximate marginal probability at node i can be computed by iterating the sum-product message passing scheme until convergence [43]. 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:

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

and the patch assignment at node i is:

$$\hat{x_i} = \underset{l}{\operatorname{argmax}}\, b_i(x_i = l) \quad (5)$$

where the belief at node i is defined as follows:

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

4.3 Message updates with the patch exclusion term

The message passing scheme introduced in the previous section may fail to reconstruct the original image because each patch can be used more than once. In this section, we propose a message passing scheme that integrates the exclusion term so that it favors a solution that uses each patch seldom 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 [43]. The message from the node i to the factor (m_{if}) can be shown to be the belief $b_i(x_i)$ (Eq. (6)), and the message from the factor to the node i can be computed as follows:

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

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 operation becomes $O(M^{(N-1)})$, which is intractable.

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

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

where

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

The full joint potential $\psi_F(x_1, ..., x_N)$ can be zero even if the patch to be placed at node i is not being shared with other image nodes if two other nodes, say $u, w \in S$, share the same patch. However, the product of pair-wise exclusion potential $\prod_{t \in S \setminus i} \psi_{F_t}(x_t|x_i)$ is zero only if the patch to be assigned to node i has already been used by another image node. Combining Eq. (7 - 9),

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

where we have assumed that m_{tf} is normalized to 1. In words, the exclusion factor node f tells the node i to place low probability on claiming patch l if patch l has already been claimed by another node with a high probability, and is intuitively satisfying.

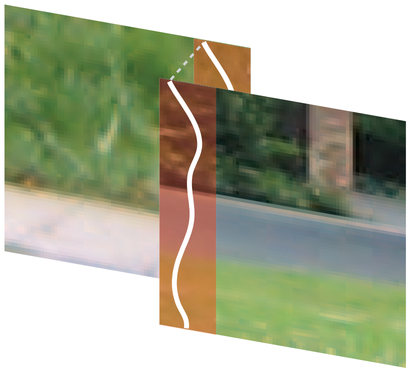

Fig. 2. This figure illustrates how we compute the left-right seam energy for two overlapping patches. The white line is the seam along which the color difference between the patches is minimum. The color difference along the white line is defined as the seam energy.

4.4 Implementation details

4.4.1 *The compatibility measure*

In [1] the patches are assumed to be non-overlapping and the patch-to-patch compatibility is defined in terms of natural image statistics prior. In this paper, we sample overlapping patches from the grid, and compute the pair-wise compatibility using the seam energy. We found that this patch sampling strategy reduces visual artifacts from tiling patches.

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 with the red strip, along which the color difference between the two patches is minimum. We use the dynamic programming method described in Efros and Freeman [30] 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:

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

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

One notable characteristic of E_{seam} is that it's 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 accomodate image editing operations, as MRF based algorithm does.

4.4.2 Computing $\sigma_c(l)$

In Cho *et al.* [1], $\sigma_c(l)$ in Eq. (11) was the same for all pairs of patches, and was set through cross-validation. In this work, we automatically set $\sigma_c(l)$ differently for every reference patch l. The main motivation is to simplify the image completion problem such that for each reference patch l, the algorithm considers only few patches as a plausible neighbor, effectively reducing the combinatorial complexity. This tends to shape our optimization function (Eq. (1)) to have a wide peak around the global maximum. 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 patch l and the best match, and $E_l(2)$ is the seam energy between patch l and the second best match.

4.4.3 The user input in the image model

The user-specified constraint can be incorporated into the image model with the local evidence term. If the user has fixed the patch k at image position i, then $p(y_i|x_i = k) = 1$ and $p(y_i|x_i = l) = 0$, for $l \neq k$. At unconstrained nodes, the mean color of the original patch y_i can serve as an observation:

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

where $m(\cdot)$ is the mean color of the argument, and $\sigma_{evid} = 0.4$ determined through cross-validation. While $m(\cdot)$ denotes the mean color in this work, it can be defined to meet application's needs. The local evidence term (Eq. (13)) steers the algorithm to retain the input image's color structure. For example, if upper part of the input image was blue, upper part of the output image will favor blue patches. Eq. (13) steers the MRF to reconstruct the original image, but the user input will steer the MRF to modify the image. This local evidence term is used in all applications described in the next section unless specified otherwise.

5 IMAGE EDITING APPLICATIONS

We introduce a number of image editing applications that leverage the patch transform. All the applications introduced here follow a unified pipeline introduced in Fig. 1.

5.1 Image reorganization

A user may want to change the location of an object after capturing the picture. We show the image editing example for relocating a person in Fig. 3. Fig. 3(a) is the original image, and the user wants to move the woman

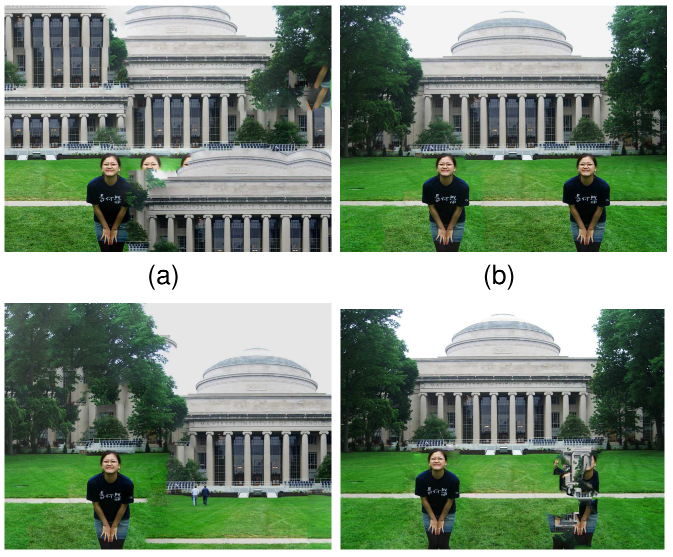

Fig. 4. (a - c) show the inverse patch transform results with and without the local evidence and the exclusion term. (a) Without the local evidence and without the exclusion term. (b) With the local evidence and without the exclusion term. (c) Without the local evidence and with the exclusion term. (d) The image editing result using Photoshop^{TM} and the image completion algorithm [14].

to the left side of the image. Fig. 3(b) 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 get relocated automatically. The local evidence $p(y_i|x_i)$ in image nodes where the woman used to stand is now uniform over all x_i since the inverse patch transform doesn't know apriori what to place there.

With this user input, the inverse patch transform finds the patch configuration, and the reconstructed image, prior to Poisson blending, is shown in Fig. 3(c), with corresponding seams overlaid. 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 did not just swap patches that belonged to the woman with the patches that the woman is placed upon.

The reconstructed image Fig. 3(c) does contain an artifact. As shown in the inset, the inverse patch transform cuts off the head of the man with a white jacket. The optimal seam search algorithm decided that it'd be cheaper to cut off his head so that his white jacket abuts the white building, which is a legitimate approach to reduce the seam energy. Other than that, the overall image looks plausible after supressing the seam artifact using Poisson blending. The resulting image after Poisson blending is shown in Fig. 3(d).

To see how each term in the image model contributes to the output image, we have conducted an experiment

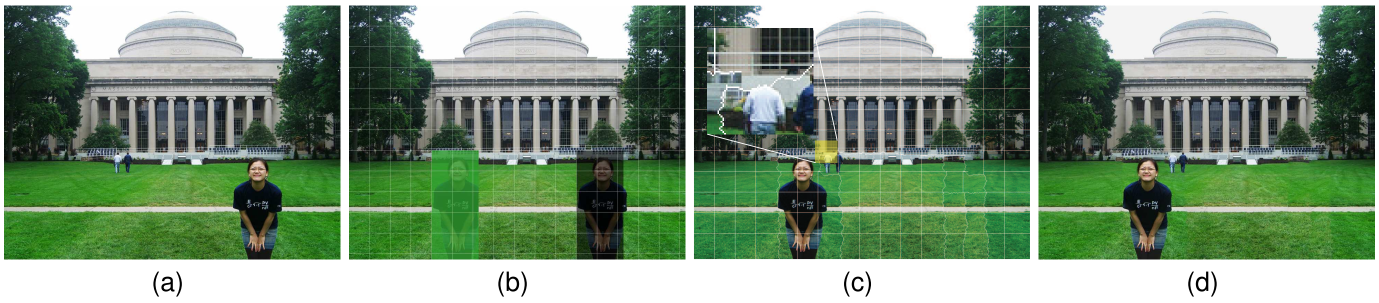

Fig. 3. This figure shows a typical example of a patch transform based image editing. (a) The input image. (b) The user input (c) The inverse patch transform result, prior to Poisson blending, with overlaid seams. (d) The final output image. The **black** bounding box denotes patches that are taken from, and the **green** bounding box denotes where those patches should be moved to.

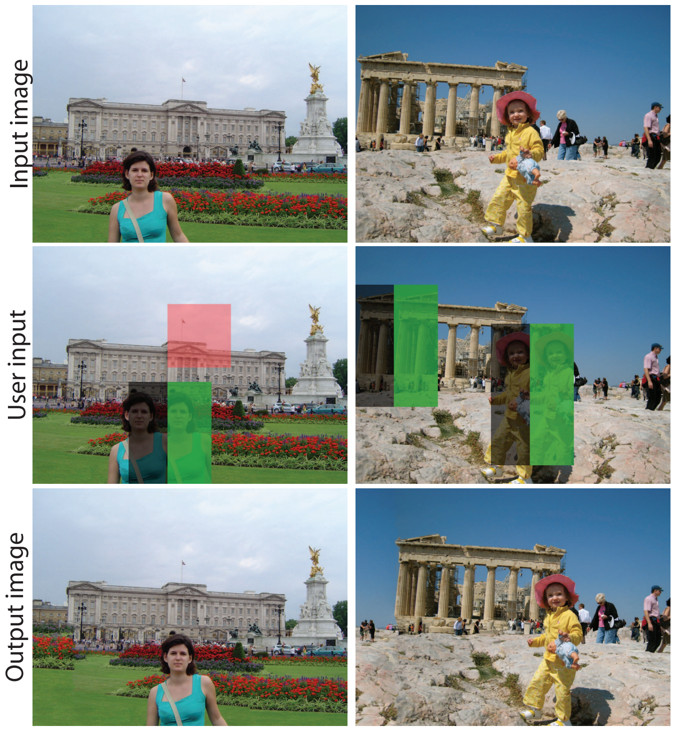

Fig. 5. Subject reorganization examples. The **red** bounding box denotes patches that are fixed at the original location.

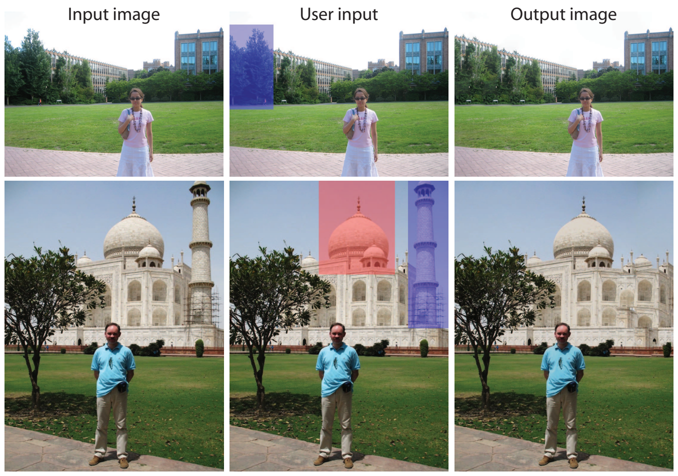

Fig. 6. The object removal examples. The inverse patch transform reconstructs an image discarding the user specified patches. The **blue** bounding box denotes patches that should be removed from the pool of patches.

where the local evidence and the exclusion term are each turned on and off during the inverse patch transform. Fig. 4 (a-c) shows the results. Fig. 4(a) 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 output 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 much better structured image shown in Fig. 4(b), 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. 4(c). While there aren't any repeating patches that generate visual artifacts, the structure of the input image is not maintained. When we incorporate both terms, we can reconstruct a more plausible image.

We compare the patch transform result with that of the conventional pipeline: the user blends the woman at the desired location using Photoshop^{TM}, and fills in the missing hole using the image completion algorithm proposed by Criminisi *et al.* [14]. The editing result is shown in Fig. 4(d). While the woman is blended seemlessly into the background, the hole now has many artifacts. A possible explanation for the failure is that the image size is very large, so the number of pixels the algorithm needs to fill in is correspondingly large. Most image completion algorithms are very effective at handling small regions, but tend to fail in completing such large regions. Also, two men in the background are removed because they are not relocated as in the patch transform framework. Fig. 5 has more examples of relocating subjects.

5.2 Object removal

The patch transform can be used to remove objects quite trivially: the user can simply remove patches that belong to the object of interest, and reconstruct the image with

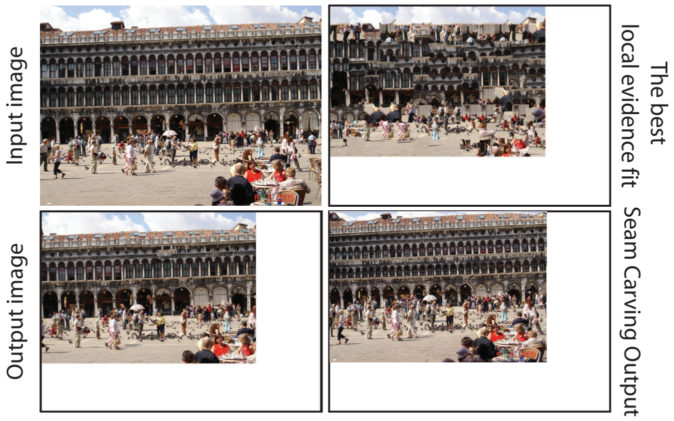

Fig. 7. The image retargeting example. The inverse patch transform reconstructs a resized image by solving for the patch assignment on a smaller canvas.

the reduced set of patches. Since the exclusion term is not a hard constraint, the inverse patch transform will judiciously re-use some patches.

Fig. 6 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, 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 to propagate it.

5.3 Image retargeting

A user may be interested in resizing the 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 [44]. The seam carving method finds a seam along which the energy is minimum, and removes the seam from the image. While it achieves excellent result 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 the image while keeping the structure of the image through the local evidence term. The 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 bi-cubically resized image.

We show an image retargetting example in Fig. 7, in which the user wants to reduce the image size to be 80% 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

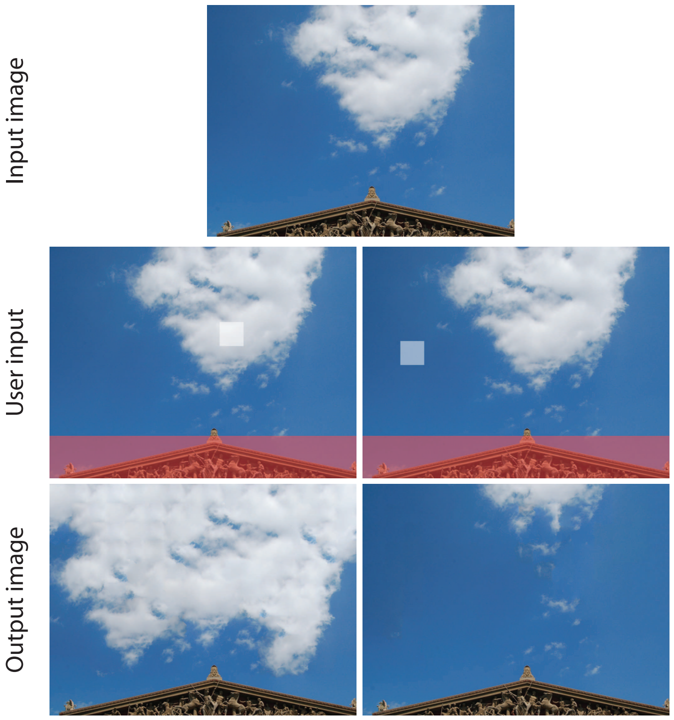

Fig. 8. The user can manipulate the patch statistics of an image through the image prior term, which can be designed to favor user specified patches (shown in **white**.)

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 [44], the scene can be well summarized, but the overall structure of the image is not maintained. Notice that sky occupies a smaller portion of the whole image.

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

5.4 Manipulating patch statistics

The patch transform is well suited for 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 appears in the image. In this section, we present another method to control the patch statistics: by manipulating $p(x_i)$ in the image model (Eq. (3).)

Let's consider the input image in Fig. 8, and suppose that the user wants to have more clouds, say similar to patch x_s, in the output image. The patch preference information can be folded into the $p(x_i)$ we modeled as a constant:

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

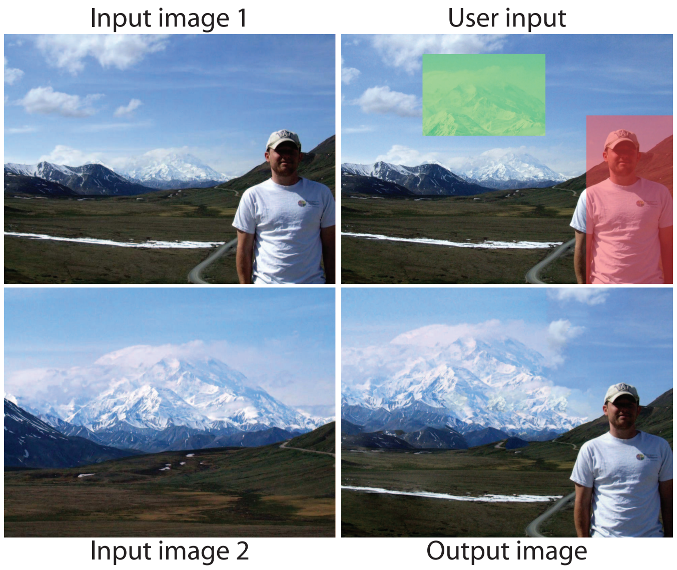

Fig. 9. We generate a photomontage of the two input images. The user wants to copy the large mountain in image 2 to the background of image 1, while keeping the person at the original location.

where σ_{sp} is a specificity parameter, and $f(\cdot)$ is a function that captures the characteristic of the argument. In this work, $f(\cdot)$ is the mean color of the argument, but can be defined to meet 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 by the energy preference specified by the prior Eq.(14). We can also favor sky patches, and generate an image consisting primarily of sky.

We have observed that manipulating $p(x_i)$ introduces a few local minima, so we had to run the inverse patch transform a few times to find a plausible image. Nevertheless, all local minima have more cloud/sky patches than the original image.

5.5 Photomontage

Until now, we have focused on manipulating patches from a single image. In this section, 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 case, the user can take multiple shots using different lenses or zooms, and combine them in the patch domain: in Fig. 9, the user wants to transfer the large mountain from image 2 to the background of image 1. This 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's a region of contention that transitions from one image to the next. The inverse patch transform finds the best region to make such transitions.

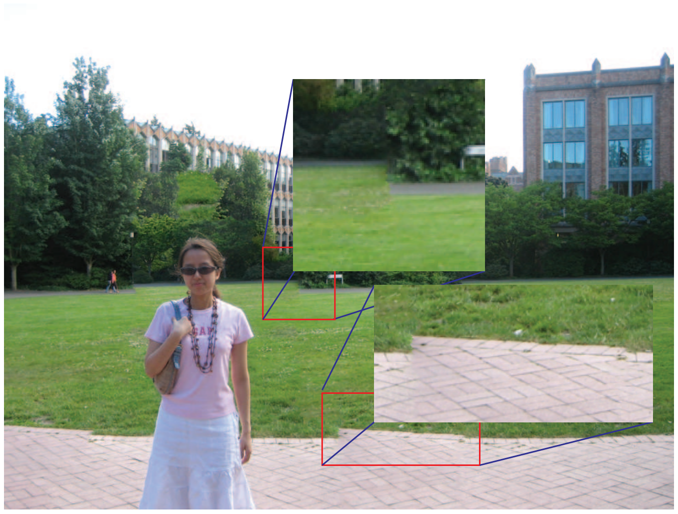

Fig. 10. Structural misalignments cause visible artifacts.

6 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 2-3 randomized initial conditions, and chose visually the most pleasing image. With our relatively unoptimized MATLAB implementation on a 2.66GHz CPU, 3GB RAM machine, the compatibility computation 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. 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 are, in effect, increasing the number of patches as well as number of image nodes in the MRF, 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, which reduces M and K. We show experimentally that the image reconstruction algorithm is sub-quadratic in the total number of patches.

6.1 Finding patch loops

Label pruning algorithms in the MRF energy minimization framework are cast as finding state labels that are potential candidates at each node. The set of potential candidates, termed active labels, is much smaller than the total number of states, thus reducing the complexity of the energy minimization algorithm. Most label pruning algorithms are first order: the algorithm retains top k states that are most likely under the local evidence measure, and does not consider what the active labels are in the neighboring nodes. This tends to increase the

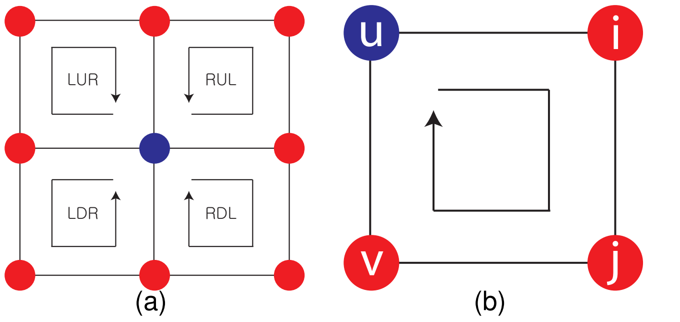

Fig. 11. (a) For a reference patch (shown in blue), there are 4 different types of neighboring loops (not considering the directions.) (b) A RDL loop.

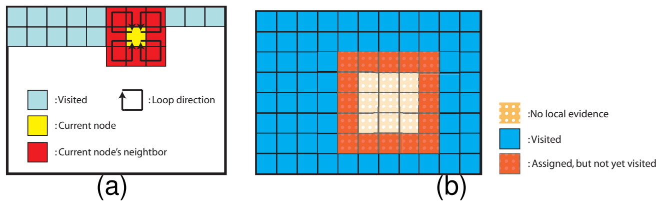

Fig. 12. (a) The active labels are added by scanning the image nodes from the top-left corner to the bottom-right. (b)The active label assignment algorithm initially steps over the region without valid local evidence. Then the algorithm assigns active labels to these regions by propagating the active labels from the boundary nodes.

number of active labels at each node, and sometimes fails to retain active labels with which its neighboring nodes' active labels have high compatibility. The proposed patch loop label pruning algorithm takes into account which labels are active in the neighboring nodes, overcoming some limitations of the first order label pruning algorithms.

A patch loop is a sequence of patches that form a loop on an MRF, as shown in Fig. 11(a). We can form 4 types of loops given a reference patch (shown in blue), ignoring the directionality. We name each loop LUR, LDR, RUL, RDL based on the order in which we traverse the loop. For an image to be visually pleasing, the product of compatibility along the patch loop should be high. Thus, we reduce the number of patch labels at each node by only "activating" patches that form highly compatible loops with patch candidates in the neighboring nodes.

We introduce an algorithm that returns l approximately best patch loops, given a fixed reference patch u, that minimize the seam energy across the loop. The optimal l loops can only be computed by enumerating all N^3 candidates, where N is the total number of patches.

Consider Fig. 11(b). Let $cH(\zeta, \eta)$ be the seam energy incurred by placing ζ to the left of η, and $cV(\zeta, \eta)$ be the seam energy incurred by 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, with the corresponding patch 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 tracking: $j^* = \arg\min_j (E_j(v^*)), i^* = \arg\min_i (E_i(j^*))$

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

6.2 Pruning patch labels with patch loops

We propose a patch label pruning method leveraging the highly compatible patch loops. This label pruning routine serves as a preprocessing step to belief propagation: the active label (i.e. the patch candidates) is fixed during belief propagation.

We assign the active labels to image nodes by scanning the image nodes from the top-left corner to the bottom-right corner (Fig. 12(a)). At each node, say the yellow node in Fig. 12(a), we take k most probable patches according to the local evidence, compute l best patch loops for each of the k chosen patches, and add patches that form highly compatible patch loops as active labels at the corresponding locations in the neighboring nodes (shown in red.) In the case of RDL loop (Fig. 11(b)), 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.

The active label assignment algorithm relies heavily on the local evidence. When the user removes patches, the local evidence is essentially uniform for all patches at those image nodes, and does not have any cue to select the initial k best patch candidates. Therefore, if an image node has a uniform local evidence, the active label assignment algorithm initially steps over it. Once the algorithm finishes the node scanning process, the algorithm propagates the active labels from the boundary of the patch removed region into the core of the patch removed region (Fig. 12(b).) 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 labels (instead of top k patch labels according to the local evidence) we compute l^* patch loops (where l^* can be different from l), and assign patches that participate in patch loops as active labels at the corresponding neighboring nodes. The initial active labels at node i are patches added by its neighbors before the algorithm reached node i. In practice $l^* = 1$ performs well.

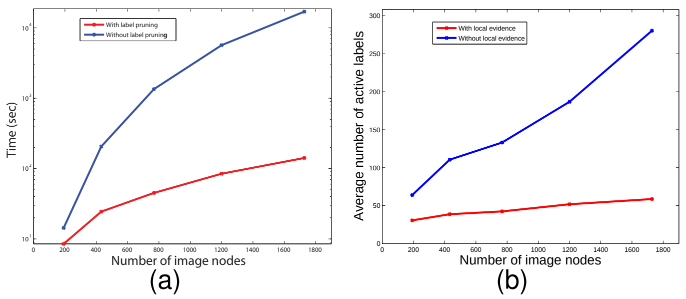

Fig. 13. (a) The run-time comparison with and without label pruning. Note that the y axis is in log-scale. (b) The average number of active labels in nodes with/without local evidence as we increase the total number of patches.

6.3 Evaluation

We experimentally show that reducing the patch size tends to improve the edited image quality. We show that the label pruning step becomes necessary due to excessive computation time as we reduce the patch size. The label pruning scheme reduces the run-time per message pass iteration as well as the total number of required message pass iterations for convergence.

6.3.1 Run-time comparison

We have performed the image reconstruction task by breaking the image into 192, 432, 768, 1200, and 1728 patches. The corresponding patch sizes are 76x76, 56x56, 46x46, 40x40, and 36x36, all with 16 pixels 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.

The blue plot in Fig. 13(a) shows the runtime of 50 message passing iterations without label pruning. Notice that the image reconstruction slows down rapidly as the number of image nodes (thus the number of patches) increases. When there are 1728 patches, it takes nearly 5 hours to run 50 message passing iterations.

The red plot in Fig. 13(a) shows the runtime of 50 message passing iterations with label pruning. Even with 1728 patches, it takes about 2.5 minutes to complete 50 message passing iterations. While the runtime is much less than the case without label pruning, the complexity of BP after label pruning is not perfectly linear in the number of patches. This can be attributed to the fact that the number of active labels in image nodes increases roughly linearly to the number of patches.

Fig. 13(b) 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 increase 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 computing the patch loops. The patch loop computation takes 8 minutes with a mex code implementation in MATLAB. Note that the patch loop is computed only once before running belief propagation, and is not visible to the user during the image editing process.

6.3.2 Image quality comparison after 50 iterations of belief propagation

The benefit of label pruning is not just the per-iteration message passing runtime reduction, but also the faster convergence of messages. In this section, we show that 50 message passing iterations is enough, even with 1728 patches, to reconstruct a plausible image with label pruning, but is not enough to reconstruct a plausible image without label pruning.

Fig. 14 shows reconstructed images with label pruning after 50 iterations of belief propagation using 192, 432, 768, 1200, and 1728 patches. The top row in Fig. 14 shows reconstructed images without label pruning. One observation is that the reconstructed images contain repeating-patch artifacts. Also, the reconstructed images contain severe structural misalignments: belief propagation did not converge to a stable solution after 50 iterations of belief propagation.

In the bottom row of Fig. 14, we show reconstructed images with label pruning, with $k = 6, l = 2$. A notable observation is that as we use smaller patches, artifacts due to structure misalignments are reduced. In the left two columns, the structural misalignment along the black pavement and the red pavement stands out, but such misalignments disappear as we increase the total number of patches. Also, note that artifacts due to patch repetition (which is introduced when belief propagation has not converged) is unnoticeable in all examples. The label pruning not only helps in reducing the runtime per message passing iteration, but also in reducing the number of message passing iterations. Experimenting with k and l revealed that messages converge within 50 iterations of BP, even with 1700 patches, if $4 \leq k \leq 8, 1 \leq l \leq 4$.

Variables k and l balance the fast convergence of belief propagation and the ability to explore the space of images that can be generated with a given set of patches. If k and l are small, the messages will converge faster, but the resulting image would look similar to the original image. If k and l are large, belief propagation requires more message passing iterations to converge, but the algorithm will explore a larger space of plausible images. If $l = 1$, the inverse patch transform with label pruning becomes $O(N)$.

We show some more images edited with the accelerated patch transform. $k = 6$, $l = 2$ is used in these experiments. Each image is broken into patches of size 36x36, with 16 pixel overlap, generating about 1700 patches per image. A rough breakdown of the processing time is

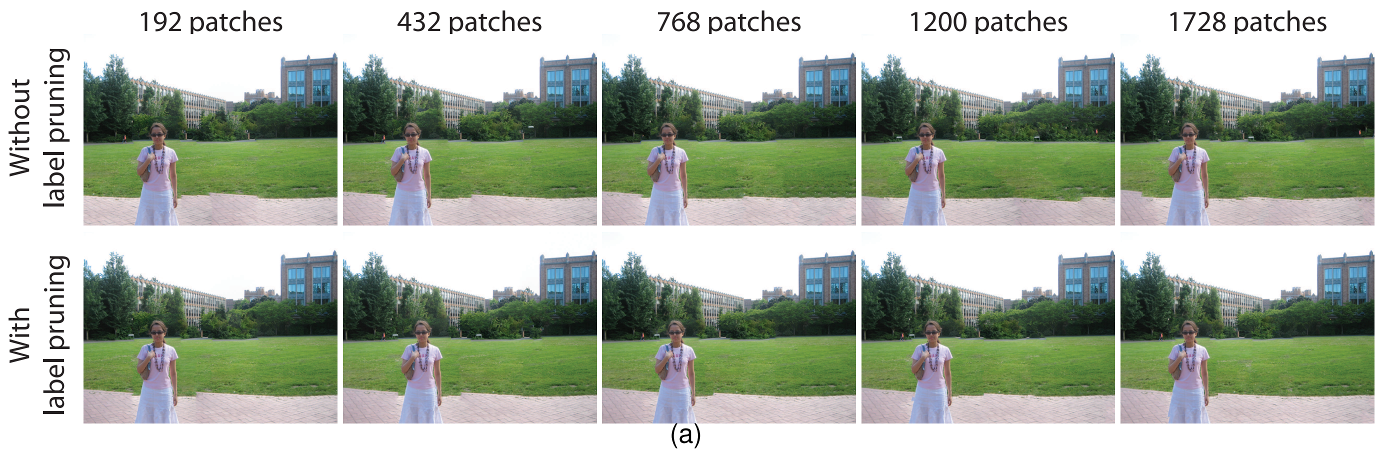

Fig. 14. Reducing the patch size improves the edited image quality. Also, the label pruning helps the faster convergence of the message passing algorithm.

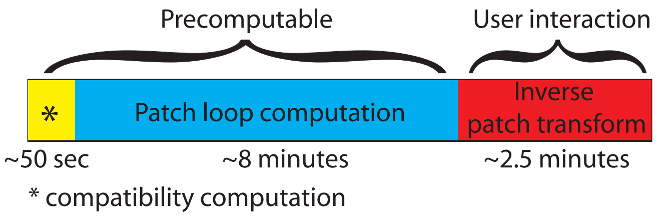

Fig. 15. The run-time breakdown for 1728 patches. The run-time is measured in MATLAB.

shown in Figure 15. Images in Fig. 16 are generated by running the inverse patch transform 3 times and taking the visually most pleasing image.

In Fig. 17 we show some failure examples. Since the patches are smaller, the patch transform sometimes finds it hard to transfer a whole object in tact: on the top row of Fig. 17, 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. 17), the different textures may try to merge.

While the use of small patches drastically improved the edited image quality, sub-patch size structural misalignments still generate visual artifacts. The next section introduces a scheme to suppress such misalignment artifacts.

7 Patch Jittering

We present a method to suppress a sub-patch size structural misalignments through a post processing step called the patch jittering. The patch jittering operation refers to finding an optimal spatial offset to patches that are assigned to image nodes. The patch jittering is an attempt to overcome the rigidity imposed by the rectangular grid. Patches can be jittered without generating a hole in the output image since patches overlap. The maximum amount of possible jitter is determined by the amount of patch overlap. When there is p pixel overlap,

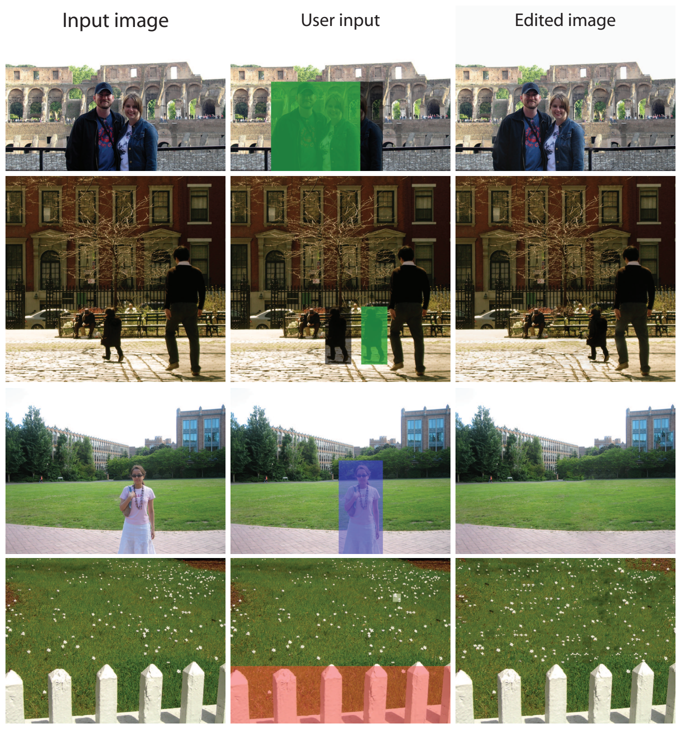

Fig. 16. More image editing examples.

we can jitter each patch $p/2$ pixels in top, down, left, right directions without generating a hole.

7.1 A local jittering scheme and its limitations

To automatically suppress the misalignment artifact, we should first detect it. One method to find two patches with structural misalignment is through pair-wise compatibility. If the pair-wise compatibility is low between two abutting patches, we jitter those patches. This operation is illustrated in Fig. 18(a). The gray overlapping squares denote the underlying patch grid, and the red square is the patch that we jitter. Notice that the red patch perfectly matches the patch on its left. By moving

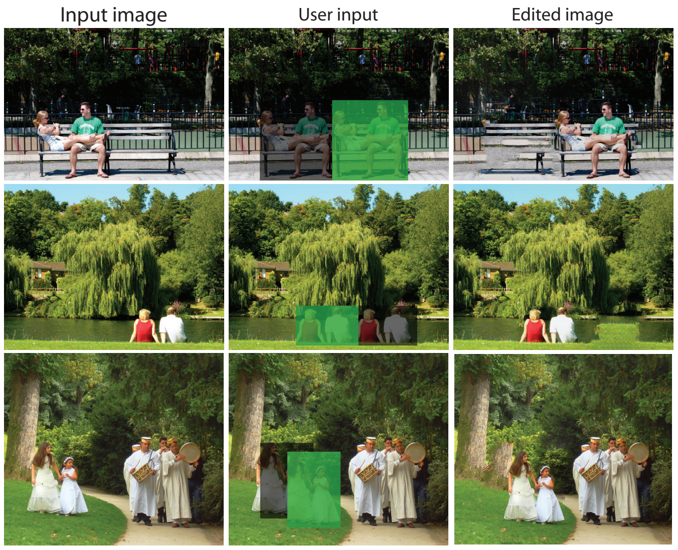

Fig. 17. Some failure examples.

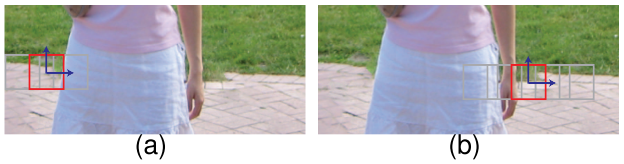

Fig. 18. (a) The misalignment between the red patch and the patch to its right can be reduced by jittering the red patch towards the right. (b) A simple local jitter operation cannot fix the misalignment error for this example. Moving the red patch upward would better fit the side on the right, but it would cause misalignment with a patch on its left.

the red patch to the right, the red patch will better fit 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 the patch has a lateral structure. Essentially, we are re-using some pixels in the overlapped region to mask alignment errors.

Jittering each patch one by one has its own limitations. Consider Fig. 18(b). By jittering the red patch upward, the red patch will better fit the patch to its right. However, the upward jitter causes a misalignment of the red patch with the patch to its left, and increases the total seam energy between the red patch and its neighbors. Therefore, we need to jitter the neighboring patches in unison to find a global optimal jitter at every node in the neighborhood.

7.2 The jitter MRF

Our approach to globally jitter patches is to use an 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 the state labels at each node is the jittered versions of the assigned patches. Then belief propagation on the jitter MRF searches for the globally optimal jittered patch at each node.

One drawback of the jitter MRF is that the number of patch labels per node can be large. When there is p pixel overlap between patches, and if each patch can move $\pm p/2$ pixels in x-y directions, the number of possible translations per patch becomes p^2. Also, we need to compute the compatibility between each jittered patch to all other jittered patches in 4 neighboring nodes. This amounts to $N \times 4 \times p^2 \times p^2$ number of seam energy computation, which can quickly become computationally expensive.

One way to reduce this number is to allow the jitter only in steps of pixels. In our implementation, $p = 16$ and the maximum amount of jitter is 8 pixels. Therefore, we allow the jitter in steps of $[-6, -3, 0, 3, 6]$ pixels. We should note that the structural misalignment occurs only in small regions of the image. In other words, we don't need to jitter patches in the entire image, but only few that are near the misalignment artifacts. While it's easy to spot the misalignment artifacts using pairwise compatibility, it's hard to find a region that should participate in jittering. Since the patch jittering is a post processing step, we require users to specify the region that needs jittering.

The patch jittering operation proceeds as follows:

1) The inverse patch transform assigns patches to image nodes. If there are any structural misalignments, the user selects regions that need jittering.
2) The algorithm computes the compatibility among neighboring patches with all possible jitters.
3) Formulate an MRF where the state variable at each node is the amount of jitter. The MRF is formed only on nodes that the user specified. Run belief propagation to find the optimal jitter at each node.

As we jitter the patch, the overlapped region between patches also changes. To compute the compatibility with different amount of overlap, we find the seam energy between two patches by finding an optimal seam only within the overlapping region, and normalize the sum of color difference along the seam by the length of the overlapped region. This way, we do not unnecessarily penalize the no-jitter state.

7.3 Evaluation

We show the patch jittering operation on our image Fig. 19(a). The inset shows that the edited image before the patch jittering contains visible alignment artifacts. Fig. 19(b) shows how the user specifies the region to be jittered, and Fig. 19(c) shows the final edited image after the patch jittering. It takes about one second to setup the jitter MRF (computing the pair-wise 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 has been removed by gradually jittering patches upward to connect the pavements. During the patch jittering, some grass pixels have been used twice

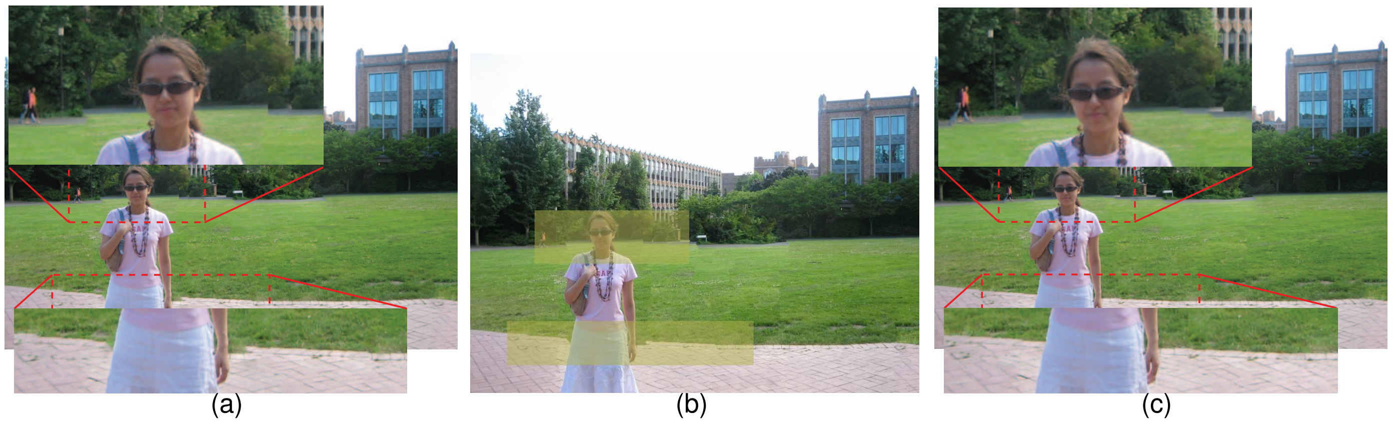

Fig. 19. (a) An edited image before the patch jittering. (b) A user input for the patch jittering. (c) An edited image after the patch jittering.

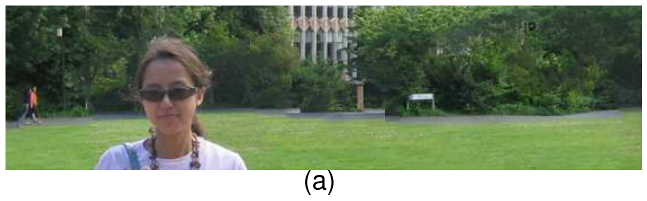

Fig. 20. Two other local minima of patch jittering operation.

and some bush pixels have disappeared. To fix the alignment error along the red pavement to the left side of the woman, the algorithm performed a local patch jittering operation to the patch that abuts the woman's dress. To fix the alignment error to the right side of the woman, the jitter MRF decided to shorten her wrist and reuse some finger pixels.

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

Fig. 21 shows more patch jittering examples on different images. The patch jittering on the first example removed the alignment error along the stair case, and the jittering on the second example removed the residual error from the water bottle.

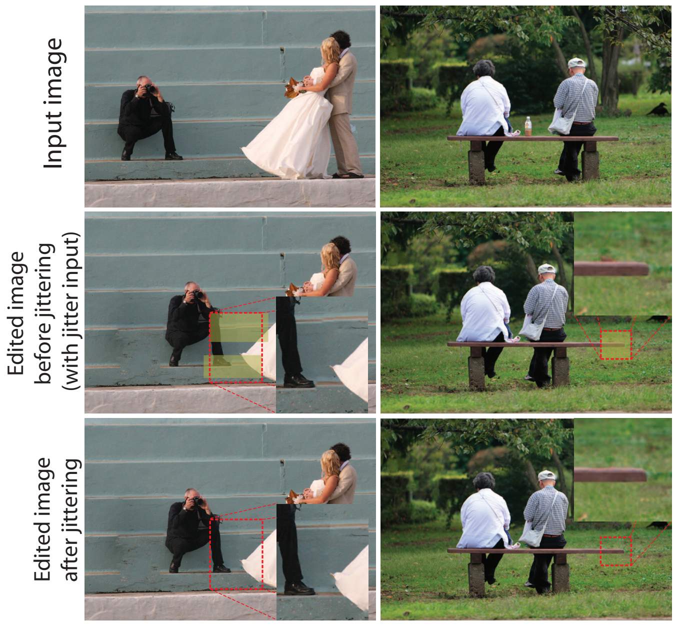
Input image
Edited image before jittering (with jitter input)
Edited image after jittering

Fig. 21. More patch jittering examples.

8 Conclusions

We have presented the patch transform and its applications to image editing. The patch transform allows users to 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: (i) neighboring patches should be compatible to one another so that visible artifacts can be minimized, (ii) each patch should be used only once in generating the output image, (iii) 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 a Markov Random Field (MRF) where each node denotes a location where we can place the patches. We introduced various image editing applications that leverages the patch transform, and verified that the patch transform framework works well with real images. To further improve the edited image quality, we introduced 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.

REFERENCES

[1] T. S. Cho, M. Butman, S. Avidan, and W. T. Freeman, "The patch transform and its applications to image editing," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
[2] E. D. Demaine and M. L. Demaine, "Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity," *Graphs and Combinatorics*, vol. 23, 2007.
[3] M. G. Chung, M. M. Fleck, and D. A. Forsyth, "Jigsaw puzzle solver using shape and color," in *Proceedings of International Conference on Signal Processing*, 1998.
[4] T. R. Nielsen, P. Drewsen, and K. Hansen, "Solving jigsaw puzzles using image features," *Pattern Recognition Letters*, 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," *ACM SIGGRAPH*, 2008.
[6] H. C. da Gama Leitao and J. Stolfi, "A multiscale method for the reassembly of two-dimensional fragmented objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 9, pp. 1239–1251, September 2002.
[7] D. Koller and M. Levoy, "Computer-aided reconstruction and new matches in the forma urbis romae," in *Bullettino Della Commissione Archeologica Comunale di Roma*, 2006.
[8] C.-S. Wang, "Determining molecular conformation from distance or density data," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.
[9] M. Levison, "The computer in literary studies," in *Machine Translation*, A. D. Booth, Ed. Amsterdam: North-Holland, 1967, pp. 173–194.
[10] L. Zhu, Z. Zhou, and D. Hu, "Globally consistent reconstruction of ripped-up documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 1, pp. 1–13, 2008.
[11] Y.-X. Zhao, M.-C. Su, Z.-L. Chou, and J. Lee, "A puzzle solver and its application in speech descrambling," in *Proceedings of the 2007 WSEAS International Conference on Computer Engineering and Applications*, 2007.
[12] N. Jojic, B. J. Frey, and A. Kannan, "Epitomic analysis of appearance and shape," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2003.
[13] A. Kannan, J. Winn, and C. Rother, "Clustering appearance and shape by learning jigsaws," in *Advances in Neural Information Processing Systems 19*, 2006.
[14] A. Criminisi, P. Pérez, and K. Toyama, "Object removal by exemplar-based inpainting," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.
[15] Y. Wexler, E. Shechtman, and M. Irani, "Space-time video completion," in *IEEE CVPR*, 2004.
[16] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, "Image completion with structure propagation," *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," *ACM SIGGRAPH*, 2007.
[18] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *ACM Transactions on Graphics (SIGGRAPH)*, 1995.
[19] J.-F. Lalonde, D. Hoiem, A. A. Efros, C. Rother, J. Winn, and A. Criminisi, "Photo clip art," *ACM Transactions on Graphics (SIGGRAPH)*, 2007.
[20] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," in *ACM Transactions on Graphics (SIGGRAPH)*, 2003.
[21] J. Wang and M. F. Cohen, "An iterative optimization approach for unified image segmentation and matting," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2005.
[22] A. Levin, A. Rav-Acha, and D. Lischinski, "Spectral matting," in *IEEE CVPR*, 2007.
[23] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani, "Summarizing visual data using bidirectional similarity," in *IEEE CVPR*, 2008.
[24] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake, "Digital tapestry," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
[25] C. Rother, L. Bordeaux, Y. Hamadi, and A. Blake, "Autocollage," in *ACM Transactions on Graphics (SIGGRAPH)*, 2006.
[26] M. Brown and D. Lowe, "Recognising panoramas," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2003.
[27] J. D. Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images," in *ACM Transactions on Graphics (SIGGRAPH)*, 1997.
[28] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *ACM Transactions on Graphics (SIGGRAPH)*, 2000.
[29] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 1999.
[30] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *ACM SIGGRAPH*, 2001.
[31] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, July 2001.
[32] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *IJCV*, 2000.
[33] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM SIGGRAPH*, 2003.
[34] N. Komodakis and G. Tziritas, "Image completion using efficient belief propagation via priority scheduling and dynamic pruning," *IEEE Transactions on Image Processing*, 2007.
[35] J. M. Coughlan and S. J. Ferreira, "Finding deformable shapes using loopy belief propagation," in *ECCV*, 2002.
[36] M. P. Kumar and P. Torr, "Fast memory-efficient generalized belief propagation," in *ECCV*, 2006.
[37] J. Lasserre, A. Kannan, and J. Winn, "Hybrid learning of large jigsaws," in *IEEE CVPR*, 2007.
[38] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *IEEE PAMI*, 2003.
[39] M. Pechaud, R. Keriven, T. Papadopoulo, and J.-M. Badier, "Combinatorial optimization for electrode labeling of eeg caps," in *MICCAI*, 2007.
[40] C. M. Bishop, D. Spiegelhalter, and J. Winn, "Vibes: A variational inference engine for bayesian networks," in *NIPS*, 2003.
[41] D. Koller, U. Lerner, and D. Angelov, "A general algorithm for approximate inference and its application to hybrid bayes nets," in *UAI*, 1998.
[42] A. Agrawal, R. Raskar, and R. Chellappa, "What is the range of surface reconstructions from a gradient field?" in *Proceedings of European Conference on Computer Vision (ECCV)*, 2006.
[43] 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, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=779352
[44] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," *ACM Transactions on Graphics (SIGGRAPH)*, 2007.