PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing

Connelly Barnes[1] Eli Shechtman[2,3] Adam Finkelstein[1] Dan B Goldman[2]
[1]Princeton University [2]Adobe Systems [3]University of Washington

(a) original (b) hole+constraints (c) hole filled (d) constraints (e) constrained retarget (f) reshuffle

Figure 1: *Structural image editing. Left to right: (a) the original image; (b) a hole is marked (magenta) and we use line constraints (red/green/blue) to improve the continuity of the roofline; (c) the hole is filled in; (d) user-supplied line constraints for retargeting; (e) retargeting using constraints eliminates two columns automatically; and (f) user translates the roof upward using reshuffling.*

Abstract

This paper presents interactive image editing tools using a new randomized algorithm for quickly finding approximate nearest-neighbor matches between image patches. Previous research in graphics and vision has leveraged such nearest-neighbor searches to provide a variety of high-level digital image editing tools. However, the cost of computing a field of such matches for an entire image has eluded previous efforts to provide interactive performance. Our algorithm offers substantial performance improvements over the previous state of the art (20-100x), enabling its use in interactive editing tools. The key insights driving the algorithm are that some good patch matches can be found via random sampling, and that natural coherence in the imagery allows us to propagate such matches quickly to surrounding areas. We offer theoretical analysis of the convergence properties of the algorithm, as well as empirical and practical evidence for its high quality and performance. This one simple algorithm forms the basis for a variety of tools – image retargeting, completion and reshuffling – that can be used together in the context of a high-level image editing application. Finally, we propose additional intuitive constraints on the synthesis process that offer the user a level of control unavailable in previous methods.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques; I.4.9 [Computing Methodologies]: Image Processing and Computer Vision—Applications

Keywords: Approximate nearest neighbor, patch-based synthesis, image editing, completion, retargeting, reshuffling

1 Introduction

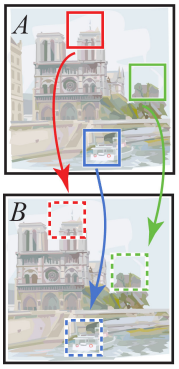
As digital and computational photography have matured, researchers have developed methods for high-level editing of digital photographs and video to meet a set of desired goals. For example, recent algorithms for *image retargeting* allow images to be resized to a new aspect ratio – the computer automatically produces a good likeness of the contents of the original image but with new dimensions [Rubinstein et al. 2008; Wang et al. 2008]. Other algorithms for *image completion* let a user simply erase an unwanted portion of an image, and the computer automatically synthesizes a fill region that plausibly matches the remainder of the image [Criminisi et al. 2003; Komodakis and Tziritas 2007]. *Image reshuffling* algorithms make it possible to grab portions of the image and move them around – the computer automatically synthesizes the remainder of the image so as to resemble the original while respecting the moved regions [Simakov et al. 2008; Cho et al. 2008].

In each of these scenarios, user interaction is essential, for several reasons: First, these algorithms sometimes require user intervention to obtain the best results. Retargeting algorithms, for example, sometimes provide user controls to specify that one or more regions (e.g., faces) should be left relatively unaltered. Likewise, the best completion algorithms offer tools to guide the result by providing hints for the computer [Sun et al. 2005]. These methods provide such controls because the user is attempting to optimize a set of goals that are known to him and not to the computer. Second, the user often cannot even articulate these goals *a priori*. The artistic process of creating the desired image demands the use of trial and error, as the user seeks to optimize the result with respect to personal criteria specific to the image under consideration.

The role of interactivity in the artistic process implies two properties for the ideal image editing framework: (1) the toolset must provide the flexibility to perform a wide variety of seamless editing operations for users to explore their ideas; and (2) the performance of these tools must be fast enough that the user quickly sees intermediate results in the process of trial and error. Most high-level editing approaches meet only one of these criteria. For example, one family of algorithms known loosely as *non-parametric patch sampling* has been shown to perform a range of editing tasks while meeting the first criterion – flexibility [Hertzmann et al. 2001; Wexler et al. 2007; Simakov et al. 2008]. These methods are based on small (e.g. 7x7) densely sampled patches at multiple scales, and are able to synthesize both texture and complex image structures that qualitatively resemble the input imagery. Because of their ability to preserve structures, we call this class of techniques *structural image editing*. Unfortunately, until now these methods have failed the second criterion – they are far too slow for interactive use on all but the smallest images. However, in this paper we will describe an algorithm that accelerates such methods by at least an order of magnitude, making it possible to apply them in an interactive structural image editing framework.

To understand this algorithm, we must consider the common components of these methods: The core element of nonparametric patch sampling methods is a repeated search of all patches

in one image region for the most similar patch in another image region. In other words, given images or regions A and B, find for every patch in A the nearest neighbor in B under a patch distance metric such as L_p. We call this mapping the *Nearest-Neighbor Field* (NNF), illustrated schematically in the inset figure. Approaching this problem with a naïve brute force search is expensive – $O(mM^2)$ for image regions and patches of size M and m pixels, respectively. Even using acceleration methods such as *approximate nearest neighbors* [Mount and Arya 1997] and dimensionality reduction, this search step remains the bottleneck of non-parametric patch sampling methods, preventing them from attaining interactive speeds. Furthermore, these tree-based acceleration structures use memory in the order of $O(M)$ or higher with relatively large constants, limiting their application for high resolution imagery.



To efficiently compute approximate nearest-neighbor fields our new algorithm relies on three key observations about the problem:

Dimensionality of offset space. First, although the dimensionality of the patch space is large (m dimensions), it is sparsely populated ($O(M)$ patches). Many previous methods have accelerated the nearest neighbor search by attacking the dimensionality of the patch space using tree structures (e.g., *kd*-tree, which can search in $O(mM \log M)$ time) and dimensionality reduction methods (e.g., PCA). In contrast, our algorithm searches in the 2-D space of possible patch offsets, achieving greater speed and memory efficiency.

Natural structure of images. Second, the usual independent search for each pixel ignores the natural structure in images. In patch-sampling synthesis algorithms, the output typically contains large contiguous chunks of data from the input (as observed by Ashikhmin [2001]). Thus we can improve efficiency by performing searches for adjacent pixels in an interdependent manner.

The law of large numbers. Finally, whereas any one random choice of patch assignment is very unlikely to be a good guess, some nontrivial fraction of a large field of random assignments will likely be good guesses. As this field grows larger, the chance that no patch will have a correct offset becomes vanishingly small.

Based on these three observations we offer a randomized algorithm for computing approximate NNFs using incremental updates (Section 3). The algorithm begins with an initial guess, which may be derived from prior information or may simply be a random field. The iterative process consists of two phases: *propagation*, in which coherence is used to disseminate good solutions to adjacent pixels in the field; and *random search*, in which the current offset vector is perturbed by multiple scales of random offsets. We show both theoretically and empirically that the algorithm has good convergence properties for tested imagery up to 2MP, and our CPU implementation shows speedups of 20-100 times versus *kd*-trees with PCA. Moreover, we propose a GPU implementation that is roughly 7 times faster than the CPU version for similar image sizes. Our algorithm requires very little extra memory beyond the original image, unlike previous algorithms that build auxiliary data structures to accelerate the search. Using typical settings of our algorithm's parameters, the runtime is $O(mM \log M)$ and the memory usage is $O(M)$. Although this is the same asymptotic time and memory as the most efficient tree-based acceleration techniques, the leading constants are substantially smaller.

In Section 4, we demonstrate the application of this algorithm in the context of a structural image editing program with three modes of interactive editing: image retargeting, image completion and image reshuffling. The system includes a set of tools that offer additional control over previous methods by allowing the user to constrain the synthesis process in an intuitive and interactive way (Figure 1).

The contributions of our work include a fast randomized approximation algorithm for computing the nearest-neighbor field between two disjoint image regions; an application of this algorithm within a structural image editing framework that enables high-quality interactive image retargeting, image completion, and image reshuffling; and a set of intuitive interactive controls used to constrain the optimization process to obtain desired creative results.

2 Related work

Patch-based sampling methods have become a popular tool for image and video synthesis and analysis. Applications include texture synthesis, image and video completion, summarization and retargeting, image recomposition and editing, image stitching and collages, new view synthesis, noise removal and more. We will next review some of these applications and discuss the common search techniques that they use as well as their degree of interactivity.

Texture synthesis and completion. Efros and Leung [1999] introduced a simple non-parametric texture synthesis method that outperformed many previous model based methods by sampling patches from a texture example and pasting them in the synthesized image. Further improvements modify the search and sampling approaches for better structure preservation [Wei and Levoy 2000; Ashikhmin 2001; Liang et al. 2001; Efros and Freeman 2001; Kwatra et al. 2003; Criminisi et al. 2003; Drori et al. 2003]. The greedy fill-in order of these algorithms sometimes introduces inconsistencies when completing large holes with complex structures, but Wexler et al. [2007] formulated the completion problem as a global optimization, thus obtaining more globally consistent completions of large missing regions. This iterative multi-scale optimization algorithm repeatedly searches for nearest neighbor patches for all hole pixels in parallel. Although their original implementation was typically slow (a few minutes for images smaller than 1 MP), our algorithm makes this technique applicable to much larger images at interactive rates. Patch optimization based approaches have now become common practice in texture synthesis [Kwatra et al. 2005; Kopf et al. 2007; Wei et al. 2008]. In that domain, Lefebvre and Hoppe [2005] have used related parallel update schemes and even demonstrated real-time GPU based implementations. Komodakis and Tziritas [2007] proposed another global optimization formulation for image completion using Loopy Belief Propagation with an adaptive priority messaging scheme. Although this method produces excellent results, it is still relatively slow and has only been demonstrated on small images.

Nearest neighbor search methods. The high synthesis quality of patch optimization methods comes at the expense of more search iterations, which is the clear complexity bottleneck in all of these methods. Moreover, whereas in texture synthesis the texture example is usually a small image, in other applications such as patch-based completion, retargeting and reshuffling, the input image is typically much larger so the search problem is even more critical. Various speedups for this search have been proposed, generally involving tree structures such as TSVQ [Wei and Levoy 2000], *k*d-trees [Hertzmann et al. 2001; Wexler et al. 2007; Kopf et al. 2007], and VP-trees [Kumar et al. 2008], each of which supports both exact and approximate search (ANN). In synthesis applications, approximate search is often used in conjunction with dimensionality reduction techniques such as PCA [Hertzmann et al. 2001; Lefebvre and Hoppe 2005; Kopf et al. 2007], because ANN methods are much more time- and memory-efficient in low dimensions. Ashikhmin [2001] proposed a *local propagation* technique exploiting local coherence in the synthesis process by

limiting the search space for a patch to the source locations of its neighbors in the exemplar texture. Our propagation search step is inspired by the same coherence assumption. The *k-coherence* technique [Tong et al. 2002] combines the propagation idea with a precomputation stage in which the k nearest neighbors of each patch are cached, and later searches take advantage of these precomputed sets. Although this accelerates the search phase, k-coherence still requires a full nearest-neighbor search for all pixels in the input, and has only been demonstrated in the context of texture synthesis. It assumes that the initial offsets are close enough that it suffices to search only a small number of nearest neighbors. This may be true for small pure texture inputs, but we found that for large complex images our random search phase is required to escape local minima. In this work we compare speed and memory usage of our algorithm against kd-trees with dimensionality reduction, and we show that it is at least an order of magnitude faster than the best competing combination (ANN+PCA) and uses significantly less memory. Our algorithm also provides more generality than kd-trees because it can be applied with arbitrary distance metrics, and easily modified to enable local interactions such as constrained completion.

Control and interactivity. One advantage of patch sampling schemes is that they offer a great deal of fine-scale control. For example, in texture synthesis, the method of Ashikhmin [2001] gives the user control over the process by initializing the output pixels with desired colors. The image analogies framework of Hertzmann et al. [2001] uses auxiliary images as "guiding layers," enabling a variety of effects including super-resolution, texture transfer, artistic filters, and texture-by-numbers. In the field of image completion, impressive guided filling results were shown by annotating structures that cross both inside and outside the missing region [Sun et al. 2005]. Lines are filled first using Belief Propagation, and then texture synthesis is applied for the other regions, but the overall run-time is on the order of minutes for a half MP image. Our system provides similar user annotations, for lines and other region constraints, but treats all regions in a unified iterative process at interactive rates. Fang and Hart [2007] demonstrated a tool to deform image feature curves while preserving textures that allows finer adjustments than our editing tools, but not at interactive rates. Pavic et al. [2006] presented an interactive completion system based on large fragments that allows the user to define the local 3D perspective to properly warp the fragments before correlating and pasting them. Although their system interactively pastes each individual fragment, the user must still manually click on each completion region, so the overall process can still be tedious.

Image retargeting. Many methods of image retargeting have applied warping or cropping, using some metric of saliency to avoid deforming important image regions [Liu and Gleicher 2005; Setlur et al. 2005; Wolf et al. 2007; Wang et al. 2008]. Seam carving [Avidan and Shamir 2007; Rubinstein et al. 2008] uses a simple greedy approach to prioritize seams in an image that can safely be removed in retargeting. Although seam carving is fast, it does not preserve structures well, and offers only limited control over the results. Simakov et al. [2008] proposed framing the problem of image and video retargeting as a maximization of bidirectional similarity between small patches in the original and output images, and a similar objective function and optimization algorithm was independently proposed by Wei et al. [2008] as a method to create texture summaries for faster synthesis. Unfortunately, the approach of Simakov et al. is extremely slow compared to seam carving. Our constrained retargeting and image reshuffling applications employ the same objective function and iterative algorithm as Simakov et al., using our new nearest-neighbor algorithm to obtain interactive speeds. In each of these previous methods, the principal method of user control is the ability to define and protect important regions from distortion. In contrast, our system integrates specific user-directable

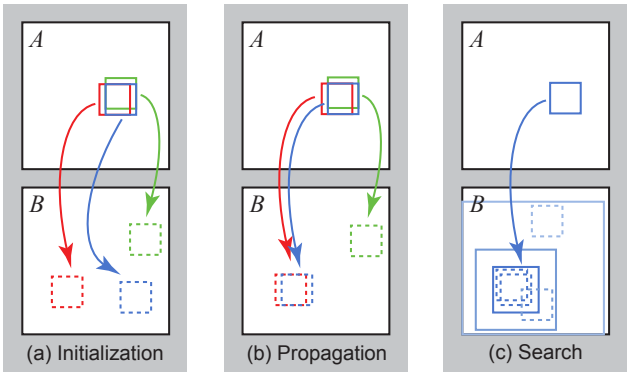


Figure 2: *Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods.*

constraints in the retargeting process to explicitly protect lines from bending or breaking, restrict user-defined regions to specific transformations such as uniform or non-uniform scaling, and fix lines or objects to specific output locations.

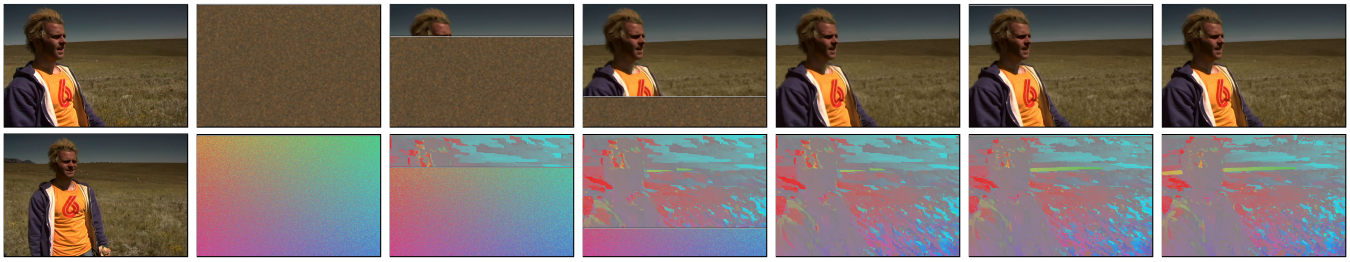
Image "reshuffling" is the rearrangement of content within an image, according to user input, without precise mattes. Reshuffling was demonstrated simultaneously by Simakov et al. [2008] and by Cho et al. [2008], who used larger image patches and Belief Propagation in an MRF formulation. Reshuffling requires the minimization of a global error function, as objects may move significant distances, and greedy algorithms will introduce large artifacts. In contrast to all previous work, our reshuffling method is fully interactive. As this task might be particularly hard and badly constrained, these algorithms do not always produce the expected result. Therefore interactivity is essential, as it allows the user to preserve some semantically important structures from being reshuffled, and to quickly choose the best result among alternatives.

3 Approximate nearest-neighbor algorithm

The core of our system is the algorithm for computing patch correspondences. We define a *nearest-neighbor field* (NNF) as a function $f : A \mapsto \mathbb{R}^2$ of offsets, defined over all possible patch coordinates (locations of patch centers) in image A, for some distance function of two patches D. Given patch coordinate $\mathbf{a}$ in image A and its corresponding nearest neighbor $\mathbf{b}$ in image B, $f(a)$ is simply $\mathbf{b} - \mathbf{a}$. We refer to the values of f as *offsets*, and they are stored in an array whose dimensions are those of A.

This section presents a randomized algorithm for computing an approximate NNF. As a reminder, the key insights that motivate this algorithm are that we search in the space of possible offsets, that adjacent offsets search cooperatively, and that even a random offset is likely to be a good guess for many patches over a large image.

The algorithm has three main components, illustrated in Figure 2. Initially, the nearest-neighbor field is filled with either random offsets or some prior information. Next, an iterative update process is applied to the NNF, in which good patch offsets are propagated to adjacent pixels, followed by random search in the neighborhood of the best offset found so far. Sections 3.1 and 3.2 describe these steps in more detail.



(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration (e) 1 iteration (f) 2 iterations (g) 5 iterations

Figure 3: *Illustration of convergence. (a) The top image is reconstructed using only patches from the bottom image. (b) above: the reconstruction by the patch "voting" described in Section 4, below: a random initial offset field, with magnitude visualized as saturation and angle visualized as hue. (c) $1/4$ of the way through the first iteration, high-quality offsets have been propagated in the region above the current scan line (denoted with the horizontal bar). (d) $3/4$ of the way through the first iteration. (e) First iteration complete. (f) Two iterations. (g) After 5 iterations, almost all patches have stopped changing. The tiny orange flowers only find good correspondences in the later iterations.*

3.1 Initialization

The nearest-neighbor field can be initialized either by assigning random values to the field, or by using prior information. When initializing with random offsets, we use independent uniform samples across the full range of image B. In applications described in Section 4, we use a coarse-to-fine gradual resizing process, so we have the option to use an initial guess upscaled from the previous level in the pyramid. However, if we use only this initial guess, the algorithm can sometimes get trapped in suboptimal local minima. To retain the quality of this prior but still preserve some ability to escape from such minima, we perform a few early iterations of the algorithm using a random initialization, then merge with the upsampled initialization only at patches where D is smaller, and then perform the remaining iterations.

3.2 Iteration

After initialization, we perform an iterative process of improving the NNF. Each iteration of the algorithm proceeds as follows: Offsets are examined in scan order (from left to right, top to bottom), and each undergoes *propagation* followed by *random search*. These operations are interleaved at the patch level: if P_j and S_j denote, respectively, propagation and random search at patch j, then we proceed in the order: $P_1, S_1, P_2, S_2, \ldots, P_n, S_n$.

Propagation. We attempt to improve $f(x,y)$ using the known offsets of $f(x-1,y)$ and $f(x,y-1)$, assuming that the patch offsets are likely to be the same. For example, if there is a good mapping at $(x-1,y)$, we try to use the translation of that mapping one pixel to the right for our mapping at (x,y). Let $D(\mathbf{v})$ denote the patch distance (error) between the patch at (x,y) in A and patch $(x,y)+\mathbf{v}$ in B. We take the new value for $f(x,y)$ to be the arg min of $\{D(f(x,y)), D(f(x-1,y)), D(f(x,y-1))\}$.

The effect is that if (x,y) has a correct mapping and is in a coherent region R, then all of R below and to the right of (x,y) will be filled with the correct mapping. Moreover, on *even* iterations we propagate information *up and left* by examining offsets in reverse scan order, using $f(x+1,y)$ and $f(x,y+1)$ as our candidate offsets.

Random search. Let $\mathbf{v_0} = f(x,y)$. We attempt to improve $f(x,y)$ by testing a sequence of candidate offsets at an exponentially decreasing distance from $\mathbf{v}_0$:

$$\mathbf{u}_i = \mathbf{v}_0 + w\alpha^i \mathbf{R}_i \tag{1}$$

where $\mathbf{R}_i$ is a uniform random in $[-1,1]\times[-1,1]$, w is a large maximum search "radius", and α is a fixed ratio between search window sizes. We examine patches for $i = 0,1,2,\ldots$ until the current search radius $w\alpha^i$ is below 1 pixel. In our applications w is the maximum image dimension, and $\alpha = 1/2$, except where noted. Note the search window must be clamped to the bounds of B.

Halting criteria. Although different criteria for halting may be used depending on the application, in practice we have found it works well to iterate a fixed number of times. All the results shown here were computed with 4-5 iterations total, after which the NNF has almost always converged. Convergence is illustrated in Figure 3 and in the accompanying video.

Efficiency. The efficiency of this naive approach can be improved in a few ways. In the propagation and random search phases, when attempting to improve an offset $f(\mathbf{v})$ with a candidate offset $\mathbf{u}$, one can do *early termination* if a partial sum for $D(\mathbf{u})$ exceeds the current known distance $D(f(\mathbf{v}))$. Also, in the propagation stage, when using square patches of side length p and an L_q norm, the change in distance can be computed incrementally in $O(p)$ rather than $O(p^2)$ time, by noting redundant terms in the summation over the overlap region. However, this incurs additional memory overhead to store the current best distances $D(f(x,y))$.

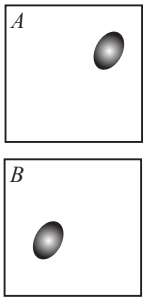
GPU implementation. The editing system to be described in Section 4 relies on a CPU implementation of the NNF estimation algorithm, but we have also prototyped a fully parallelized variant on the GPU. To do so, we alternate between iterations of random search and propagation, where each stage addresses the entire offset field in parallel. Although propagation is inherently a serial operation, we adapt the jump flood scheme of Rong and Tan [2006] to perform propagation over several iterations. Whereas our CPU version is capable of propagating information all the way across a scanline, we find that in practice long propagations are not needed, and a maximum jump distance of 8 suffices. We also use only 4 neighbors at each jump distance, rather than the 8 neighbors proposed by Rong and Tan. With similar approximation accuracy, the GPU algorithm is roughly 7x faster than the CPU algorithm, on a GeForce 8800 GTS card.

3.3 Analysis for a synthetic example

Our iterative algorithm converges to the exact NNF in the limit. Here we offer a theoretical analysis for this convergence, showing that it converges most rapidly in the first few iterations with high probability. Moreover, we show that in the common case where only approximate patch matches are required, the algorithm converges even faster. Thus our algorithm is best employed as an approximation algorithm, by limiting computation to a small number of iterations.

We start by analyzing the convergence to the exact nearest-neighbor field and then extend this analysis to the more useful case of con-

vergence to an approximate solution. Assume A and B have equal size (M pixels) and that random initialization is used. Although the odds of any one location being assigned the best offset in this initial guess are vanishingly small $(1/M)$, the odds of at least one offset being correctly assigned are quite good $(1-(1-1/M)^M))$ or approximately $1-1/e$ for large M. Because the random search is quite dense in small local regions we can also consider a "correct" assignment to be any assignment within a small neighborhood of size C pixels around the correct offset. Such offsets will be corrected in about one iteration of random search. The odds that at least one offset is assigned in such a neighborhood are excellent: $(1-(1-C/M)^M)$ or for large M, $1-exp(-C)$.



Now we consider a challenging synthetic test case for our algorithm: a distinctive region R of size m pixels lies at two different locations in an otherwise uniform pair of images A and B (shown inset). This image is a hard case because the background offers no information about where the offsets for the distinctive region may be found. Patches in the uniform background can match a large number of other identical patches, which are found by random guesses in one iteration with very high probability, so we consider convergence only for the distinct region R. If any one offset in the distinct region R is within the neighborhood C of the correct offset, then we assume that after a small number of iterations, due to the density of random search in small local regions (mentioned previously), that all of R will be correct via propagation (for notational simplicity assume this is instantaneous). Now suppose R has not yet converged. Consider the random searches performed by our algorithm at the maximum scale w. The random search iterations at scale w independently sample the image B, and the probability p that any of these samples lands within the neighborhood C of the correct offset is

$$p = 1-(1-C/M)^m \tag{2}$$

Before doing any iterations, the probability of convergence is p. The probability that we did not converge on iterations $0,1,...,t-1$ and converge on iteration t is $p(1-p)^t$. The probabilities thus form a geometric distribution, and the expected time of convergence is $\langle t\rangle = 1/p-1$. To simplify, let the relative feature size be $\gamma = m/M$, then take the limit as resolution M becomes large:

$$\langle t\rangle = [1-(1-C/M)^{\gamma M}]^{-1}-1 \tag{3}$$

$$\lim_{M\to\infty}\langle t\rangle = [1-\exp(-C\gamma)]^{-1}-1 \tag{4}$$

By Taylor expansion for small γ, $\langle t\rangle = (C\gamma)^{-1}-\frac{1}{2} = M/(Cm)-\frac{1}{2}$. That is, our expected number of iterations to convergence remains constant for large image resolutions and a small feature size m relative to image resolution M.

We performed simulations for images of resolution M from 0.1 to 2 Megapixels that confirm this model. For example, we find that for a $m = 20^2$ region the algorithm converges with very high probability after 5 iterations for a $M = 2000^2$ image.

The above test case is hard but not the worst one for exact matching. The worst case for exact matching is when image B consists of a highly repetitive texture with many distractors similar to the distinct feature in A. The offset might then get "trapped" by one of the distractors, and the effective neighborhood region size C might be decreased to 1 (i.e., only the exact match can pull the solution out of the distractor during random search). However in practice, for many image analysis and synthesis applications such as the ones we show in this paper, finding an approximate match (in terms of patch similarity) will not cause any noticeable difference. The chances of finding a successful approximate match are actually higher when many similar distractors are present, since each distractor is itself an approximate match. If we assume there are Q distractors in image B that are similar to the exact match up to some small threshold, where each distractor has approximately the same neighborhood region C, then following the above analysis the expected number of iterations for convergence is reduced to $M/(QCm)-0.5$.

	Time [s]		Memory [MB]	
Megapixels	Ours	*k*d-tree	Ours	*k*d-tree
0.1	0.68	15.2	1.7	33.9
0.2	1.54	37.2	3.4	68.9
0.35	2.65	87.7	5.6	118.3

Table 1: *Running time and memory comparison for the input shown in Figure 3. We compare our algorithm against a method commonly used for patch-based search: kd-tree with approximate nearest neighbor matching. Our algorithm uses $n = 5$ iterations. The parameters for kd-tree have been adjusted to provide equal mean error to our algorithm.*

3.4 Analysis for real-world images

Here we analyze the approximations made by our algorithm on real-world images. To assess how our algorithm addresses different degrees of visual similarity between the input and output images, we performed error analysis on datasets consisting of pairs of images spanning a broad range of visual similarities. These included inputs and outputs of our editing operations (very similar), stereo pairs[1] and consecutive video frames (somewhat similar), images from the same class in the Caltech-256 dataset[2] (less similar) and pairs of unrelated images. Some of these were also analyzed at multiple resolutions (0.1 to 0.35 MP) and patch sizes (4x4 to 14x14). Our algorithm and ANN+PCA *k*d-tree were both run on each pair, and compared to ground truth (computed by exact NN). Note that because precomputation time is significant for our applications, we use a single PCA projection to reduce the dimensionality of the input data, unlike Kumar et al. [2008], who compute eigenvectors for different PCA projections at each node of the *k*d-tree. Because each algorithm has tunable parameters, we also varied these parameters to obtain a range of approximation errors.

We quantify the error for each dataset as the mean and 95th percentile of the per-patch difference between the algorithm's RMS patch distance and the ground truth RMS patch distance. For 5 iterations of our algorithm, we find that mean errors are between 0.2 and 0.5 gray levels for similar images, and between 0.6 and 1.5 gray levels for dissimilar images (out of 256 possible gray levels). At the 95th percentile, errors are from 0.5 to 2.5 gray levels for similar images, and 0.9 to 6.0 gray levels for dissimilar images.

Our algorithm is both substantially faster than *k*d-tree and uses substantially less memory over a wide range of parameter settings. For the 7x7 patch sizes used for most results in the paper, we find our algorithm is typically 20x to 100x faster, and uses about 20x less memory than *k*d-tree, regardless of resolution. Table 1 shows a comparison of average time and memory use for our algorithm vs. ANN *k*d-trees for a typical input: the pairs shown in Figure 3. The rest of our datasets give similar results. To fairly compare running time, we adjusted ANN *k*d-tree parameters to obtain a mean approximation error equal to our algorithm after 5 iterations.

The errors and speedups obtained are a function of the patch size and image resolution. For smaller patches, we obtain smaller speedups (7x to 35x for 4x4 patches), and our algorithm has higher error values. Conversely, larger patches give higher speedups (300

[1] `http://vision.middlebury.edu/stereo/data/scenes2006/`

[2] `http://www.vision.caltech.edu/Image_Datasets/Caltech256/`

times or more for 14x14 patches) and lower error values. Speedups are lower at smaller resolutions, but level off at higher resolutions.

In our comparison, we also implemented the fully parallelized algorithm from Section 3.2 (proposed for GPU usage) on a multi-core CPU architecture. In this context, its errors are comparable with the original CPU algorithm, and although it is roughly 2.5x slower than the CPU algorithm on a single core, the run-time scales linearly with the number of cores.

Recent work [Kumar et al. 2008] indicates that vp-trees are more efficient than *k*d-trees for exact nearest-neighbor searches. We found this to be the case on our datasets. However, exact matching is far too slow for our applications. When doing approximate matching with PCA, we found that vp-trees are slower than *k*d-trees for equivalent error values, so we omitted them from our comparison above.

4 Editing tools

In this section, we discuss some of the novel interactive editing tools enabled by our algorithm. First, however, we must revisit the bidirectional similarity synthesis approach [Simakov et al. 2008]. This method is based on a bidirectional distance measure between pairs of images - the source (input) image S and a target (output) image T. The measure consists of two terms: (1) The *completeness* term ensures that the output image contains as much visual information from the input as possible and therefore is a good summary. (2) The *coherence* term ensures that the output is coherent w.r.t. the input and that new visual structures (artifacts) are penalized. Formally, the distance measure is defined simply as the sum of the average distance of all patches in S to their most similar (nearest-neighbor) patches in T and vice versa:

$$d_{BDS}(S,T) = \overbrace{\frac{1}{N_S}\sum_{s\subset S}\min_{t\subset T} D(s,t)}^{d_{complete}(S,T)} + \overbrace{\frac{1}{N_T}\sum_{t\subset T}\min_{s\subset S} D(t,s)}^{d_{cohere}(S,T)} \qquad (5)$$

where the distance D is the SSD (sum of squared differences) of patch pixel values in L*a*b* color space. For image retargeting, we wish to solve for the image T that minimizes d_{BDS} under the constraints of the desired output dimensions. Given an initial guess for the output image, T_0, this distance is iteratively minimized by an EM-like algorithm. In the E step of each iteration i, the NN-fields are computed from S and T_i and "patch-voting" is performed to accumulate the pixel colors of each overlapping neighbor patch. In the M step, all the color "votes" are averaged to generate a new image T_{i+1}. To avoid getting stuck in bad local minima, a multi-scale "gradual scaling" process is employed: T is initialized to a low-resolution copy of S and is gradually resized by a small factor, followed by a few EM iterations after each scaling, until the final dimensions are obtained. Then, both T and S are gradually upsampled to finer resolutions, followed by more EM iterations, until the final fine resolution is obtained.

In addition to image retargeting, Simakov et al. [2008] showed that this method can be used for other synthesis tasks such as image collages, reshuffling, automatic cropping and the analogies of these in video. Furthermore, if we define a missing region (a "hole") in an image as T and the rest of the image as S, and omit the *completeness term*, we end up with exactly the image and video completion algorithm of Wexler et al. [2007]. Importance weight maps can be used both in the input (e.g., for emphasizing an important region), and in the output (e.g., for guiding the completion process from the hole boundaries inwards).

The randomized NNF algorithm given in Section 3 places no explicit constraints on the offsets other than minimizing patch distances. However, by modifying the search in various ways, we can introduce local constraints on those offsets to provide the user with more control over the synthesis process. For image retargeting, we can easily implement *importance masks* to specify regions that should not deform at all [Avidan and Shamir 2007; Wolf et al. 2007; Rubinstein et al. 2008; Simakov et al. 2008] or scale uniformly [Wang et al. 2008], as in previous work. However, we can also explicitly define constraints not well supported by previous methods, such as straight lines that must remain straight, or objects and lines that should move to some other location while the image is retargeted. We can clone objects ("copy and paste") or define something else that should replace the resulting hole. Objects can be scaled uniformly or non-uniformly (e.g, for "growing" trees or buildings vertically) in the context of structural image editing. All these are done naturally by marking polygons and lines in the image. A simple bounding box for objects often suffices.

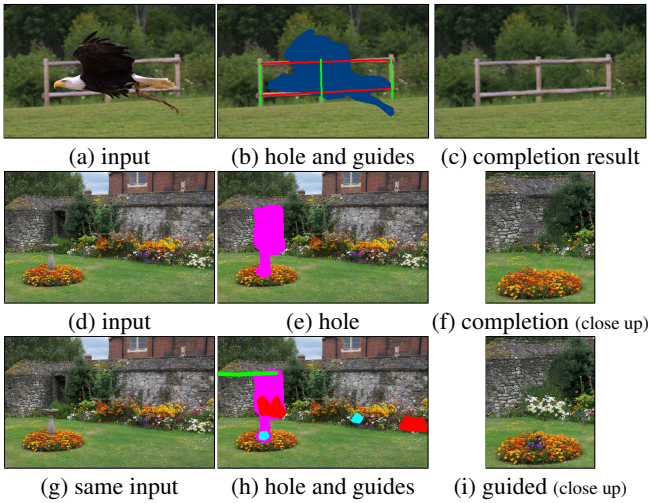


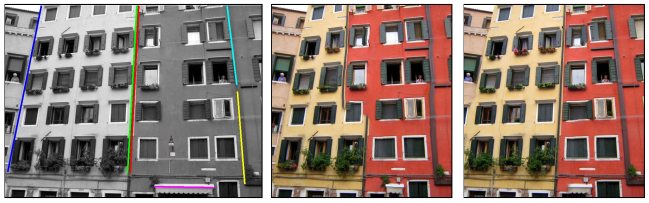
Figure 4: *Two examples of guided image completion. The bird is removed from input (a). The user marks the completion region and labels constraints on the search in (b), producing the output (c) in a few seconds. The flowers are removed from input (d), with a user-provided mask (e), resulting in output (f). Starting with the same input (g), the user marks constraints on the flowers and roofline (h), producing an output (i) with modified flower color and roofline.*

Some of these image editing tools are new and others have been used before in a limited context. However, we believe that the collection of these tools – in conjunction with the interactive feedback provided by our system – creates new powerful image editing capabilities and a unique user interaction experience.

4.1 Search space constraints

Image completion of large missing regions is a challenging task. Even the most advanced global optimization based methods [Wexler et al. 2007; Komodakis and Tziritas 2007] can still produce inconsistencies where structured content is inpainted (e.g., a straight line that crosses the missing region). Furthermore, in many cases the boundaries of the missing region provide few or no constraints for a plausible completion. Sun et al. [2005] proposed a guided structure propagation approach, in which the user draws curves on top of edges that start outside the hole and defines where they should pass through it. This method propagates structure nicely along the curves, but the overall process is still slow (often in the order of a few minutes for a 0.5 megapixel image) and sometimes requires further manual intervention.

In our work, we have adopted the same user interaction approach, allowing the user to draw curves across the missing region. The



(a) original (b) retargeted (c) with constraints

Figure 5: *Model constraints. The original image (a) is retargeted without constraints, producing broken boundaries in (b). When constraints shown as colored lines are added, the boundaries remain straight (c).*

(a) original (b) retargeted (c) with constraints

Figure 6: *Free lines and uniform scale constraints. The original image (a) is retargeted without constraints (b). Constraints indicated by colored lines produce straight lines and the circle is scaled down to fit the limited space (c).*

curves can have different labels (represented in our interface using different colors) to indicate propagation of different structures. However unlike Sun et al. [2005], which utilizes separate completion processes for curves and textured regions, our system synthesizes both simultaneously in the same unified framework. This is accomplished by limiting the search space for labeled pixels inside the hole to the regions outside the hole with the same label. (Paradoxically, adding these extra constraints accelerates the convergence properties by limiting the search space.) Figure 1 illustrates the effect of these search space constraints for image completion. In addition to curve and edge structures, the same tool can be used to specify particular contents for some portions of the hole. This type of interaction is similar to the "texture by numbers" approach [Hertzmann et al. 2001] when applied to image completion. We show examples of these cases in Figures 1 and 4.

4.2 Deformation constraints

Many recent retargeting methods allow the user to mark *semantically important* regions to be used with other automatically-detected cues (e.g., edges, faces, saliency regions) [Avidan and Shamir 2007; Simakov et al. 2008; Wang et al. 2008]. One important cue that has been overlooked by previous methods are the lines and objects with straight edges that are so common in images of both man-made scenes (indoor photos, buildings, roads) and in natural scenes (tree trunks, horizon lines). Keeping such lines straight is important to produce plausible outputs. However, marking a line as an important *region* in existing techniques usually forces the line to appear in its entirety in the output image, but does not guarantee that the line will not bend or break (see Figures 11). Moreover often we do not care if the line gets shorter or longer in the output, as long as it remains straight. We show additional examples of straight-line constraints in Figures 5 and 6.

Model constraints. To accomplish this, we extend the BDS formulation from a free optimization to a *constrained* optimization

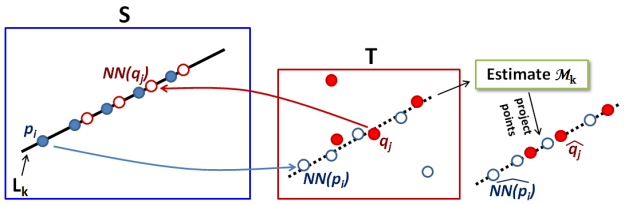
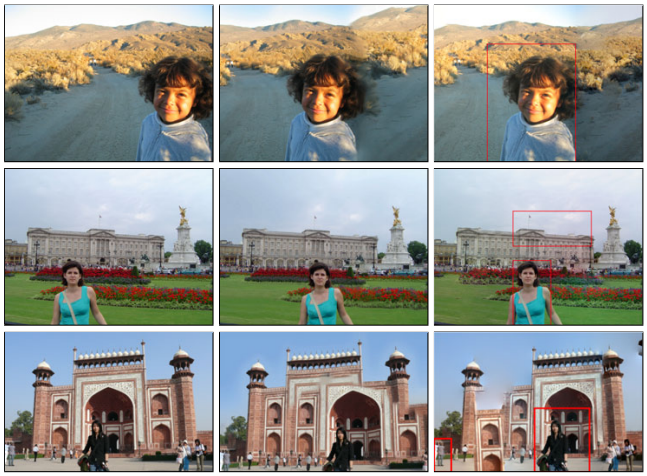


Figure 7: *Model constraints. L_k is a straight line in the source image S. For point p_i on L_k, the nearest neighbor in T is $NN(p_i)$. A point q_i in T has nearest neighbor $NN(q_i)$ that lies on L_k. We collect all such points $NN(p_i)$ and q_i and robustly compute the best line M_k in T, then project the points to the estimated line.*



(a) input (b) our reshuffling (c) Patch Transform

Figure 8: *Examples of reshuffling. Input images are shown in the first and fourth column. The user annotates a rough region to move, and the algorithm completes the background in a globally consistent way. "Patch Transform" results are shown from Cho et al. [2008], which takes minutes to run, and our algorithm, which runs interactively.*

problem, by constraining the domain of possible nearest neighbor locations in the output for certain subsets of input points. Following the notation of equation (5), let $\{p_i\} \in L_k$ refer to all the pixels belonging to the region (or line) L_k in the input image S, and let $\{q_j\} \in T | NN(q_j) \in L_k$ refer to all the points q_j in the output T whose nearest neighbors lie in the region L_k (see notations in Figure 7). The objective from equation (5) then becomes:

$$\arg\min d_{BDS} \quad \text{s.t.} \quad \mathcal{M}_k(p_i, NN(p_i), q_j, NN(q_j)) = 0 \qquad (6)$$

where we have K models $\mathcal{M}_k$ ($k \in 1 \ldots K$) to satisfy. The meaning of $\mathcal{M}_k() = 0$ is as follows: in the case of lines, satisfying a model means that the dot product of the line 3-vector (l, in homogeneous coordinates) and all the points in the output image should be zero, i.e., $NN(p_i)^T l = 0$. In the case of regions, we limit ourselves here to 2D homography transformations (H) and therefore satisfying the model means that the distance of all projected points and the corresponding NN points in the output image should be zero, i.e., $H_k p_i - NN(p_i) = 0$.

Now the question remains how best to impose these constraints on the solution. We observed that during the "gradual scaling" process lines and regions deform only gradually due to lack of space. This gives us an opportunity to correct these deformations using small adjustments after each EM iteration. So, in order to satisfy the additional constraints, we apply an iterative correction scheme using the RANSAC [Fischler and Bolles 1981] robust estimation method after each iteration. We assume that in each iteration

most of the locations of $NN(p_i)$ as well as q_j *almost* satisfy the desired model, and we estimate this model robustly by discarding outliers. The estimated model is used to project the output points onto $\widehat{NN(p_i)}$ and $\widehat{q_j}$, and to correct the NN fields accordingly. For regions, outlier points are corrected but we obtained better results for lines by excluding outliers from the voting process.

We found the following models to be useful for constrained retargeting and reshuffling applications: *Free lines*, with output points constrained to lie on a straight line with unconstrained translation and slope (see Figure 6); *Fixed-slope lines*, with slope identical to the input, but free translation (see Figure 5); *Fixed-location lines*, with fixed slope and user-defined translation, for which there is no model to estimate, but the points are still projected onto the line allowing its length to change (see Figure 12(n) where the waterline was dragged down with a line constraint); *Translating regions*, with free translation but fixed scale (see Figure 11(right) where the car and bridge were marked as a translating region); and *Scaled regions*, with user-defined uniform scale and free translation (see Figure 6).

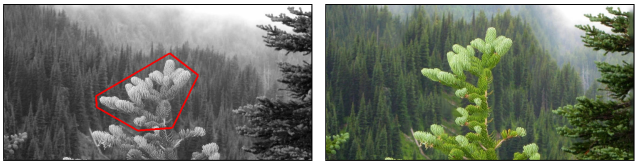
Hard constraints (a.k.a. "reshuffling"). The model constraints described in the previous section usually succeed in preserving lines and regions. However, in difficult cases, with large scale factors – or when there are contradictions between the various constraints – they cannot all be well satisfied automatically. In other cases, such as in image *reshuffling* [Simakov et al. 2008; Cho et al. 2008], the user may want to strictly define the location of a region in the output as a hard constraint on the optimization. This can be easily done in our framework, by fixing the NN fields of the relevant region points to the desired offsets according to these hard constraints. After each EM iteration we simply correct the offsets to the output position, so that the other regions around the objects gradually rearrange themselves to align with these constrained regions. For an object that moves substantially from its original output location, we give three intuitive options to the user to determine the initialization of the contents of the hole before the optimization starts: *swap*, in which the system simply swaps the pixels between the old and new locations; *interpolate*, in which the system smoothly interpolates the hole from the boundaries (as in Wexler et al. [2007]); and *clone*, in which the system simply clones the object in its original location. For small translations these all generate similar outputs, but for large objects and large motions these options lead to entirely different results (see Figure 12).

Local structural scaling. A tool shown in Figure 9 allows the user to mark an object and rescale it while preserving its texture and structure (unlike regular scaling). We do this by gradually scaling the object and running a few EM iterations after each scale at the coarse resolution, just as in the global retargeting process.

4.3 Implementation details

Small changes of orientation and scale in some of the deformation models (e.g., *free lines* and *scaled regions*) can be accomplished by simply rearranging the locations of existing patches. For larger angle and scale changes that may be required for extreme retargeting factors, one may have to rotate/scale the patches as well. In each of the above cases we use a weighted version of equation (5) (see [Simakov et al. 2008]) and we increase the weight of patches in the important regions and lines by 20%. We also note that finer levels of the pyramid have better initial guesses, and therefore the search problem is easier and fewer EM iterations are needed. We thus use a high value (typically 20-30) of EM iterations at the coarsest level, and at finer levels we use a number of EM iterations that decreases linearly with the pyramid level. For the last few levels of the pyramid, global matching is less necessary, so we find that reducing the random search radius to $w = 1$ does not significantly affect quality.

(a) building marked by user (b) scaled up, preserving texture



(c) bush marked by user (d) scaled up, preserving texture.

Figure 9: *Examples using local scale tool. In both examples, the user marks a source polygon, and then applies a nonuniform scale to the polygon, while preserving texture.*

5 Results, discussion, and future work

As we have discussed, our nearest-neighbor framework can be used for retargeting, completing holes, and reshuffling content in images. We have compared performance and quality with several other competing methods in these domains.

Figures 10 and 11 illustrate differences between the results produced by our method and those of Rubinstein et al. [2008] and Wang et al. [2008]. In Figure 10, both existing retargeting methods deform one of the two children in the photograph, whereas our system allows us to simply reshuffle one of the children, thus gaining more space for retargeting, and the background is automatically reconstructed in a plausible manner. In Figure 11, we see that "seam carving" introduces unavoidable geometric distortions in straight lines and compresses repeating elements. In contrast, our method allows us to explicitly preserve perspective lines and elide repetitive elements.

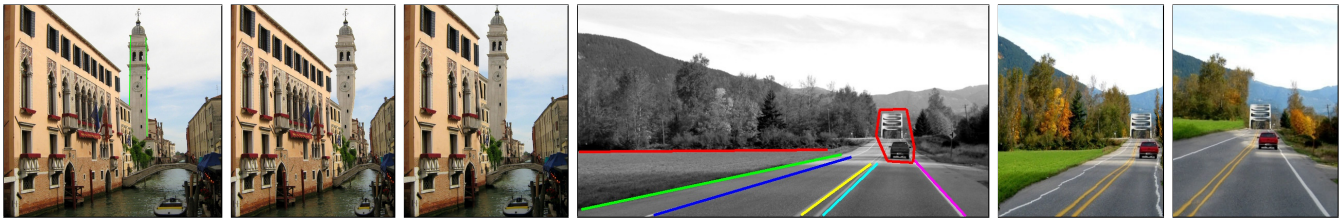
By marking out mask regions, users can interactively fill nontrivial holes. For example, in Figure 1, we use our system to perform some long-overdue maintenance on an ancient Greek temple.

Our reshuffling tools may be used to quickly modify architectural dimensions and layouts, as shown in Figure 12. Visually plausible buildings – albeit occasionally fantastical! – can be constructed easily by our algorithm, because architecture often contains many repeating patterns. Reshuffling can also be used to move human or organic objects on repeating backgrounds, as previously illustrated in Figure 8. However, in some cases line constraints are needed to fix any linear elements of the background that intersect the foreground object, as demonstrated in Figure 10, in which a line constraint is used to constrain the shadow in the sand.

The nature of our algorithm bears some superficial similarity to LBP and Graph Cuts algorithms often used to solve Markov Random Fields on an image grid [Szeliski et al. 2008]. However, there are fundamental differences: Our algorithm is designed to optimize an energy function without any neighborhood term. MRFs often use such a neighborhood term to regularize optimizations

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

Figure 10: *Retargeting. From left: (a) Input image, (b) [Rubinstein et al. 2008], (c) [Wang et al. 2008], (d) Our constraints, (e) Our result.*



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

Figure 11: *Retargeting: (a) Input image, annotated with constraints, (b) [Rubinstein et al. 2008], (c) Our output.*

with noisy or missing data, based on coherency in an underlying generative model. In contrast, our algorithm has no explicit generative model, but uses coherency in the *data* to prune the search for likely solutions to a simpler parallel search problem. Because our search algorithm finds these coherent regions in early iterations, our matches err toward coherence. Thus, even though coherency is not explicitly enforced, our approach is sufficient for many practical synthesis applications. Moreover, our algorithm avoids the expensive computations of the joint patch compatibility term and inference/optimization algorithms.

As with all image synthesis approaches, our algorithm does have some failure cases. Most notably, for pathological inputs, such as the synthetic test case of Section 3.3, we have poor convergence properties. In addition, extreme edits to an image can sometimes produce "ghosting" or "feathering" artifacts where the algorithm simply cannot escape a large local minimum basin. However, we point out that the speed of our algorithm makes it feasible to either introduce additional constraints or simply rerun the algorithm with a new random seed to obtain a different solution. Although such repeated trials can be a burden with a slower algorithm, in our experiments we occasionally enjoyed such explorations of the space of creative image manipulations!

Among the many exciting avenues for future work in this domain, we highlight several important ones:

Extending the matching algorithm. By using a queue at every pixel, rather than computing a single nearest neighbor, one could compute k nearest neighbors. This may allow the k-coherence strategy to be used with our algorithm. In general, we find that the optimal random sampling pattern and halting criteria are functions of the inputs. For some inputs, such as small highly regular textures, less or no random search is required. For other inputs with large structures, such as in Figure 12, the full random search is needed. By exploring these tradeoffs and further investigating GPU implementations, additional speed gains may be realized, opening up new applications in real-time vision and video processing.

Other applications. Although we have focused on creative image manipulation in this paper, we also hope to implement completion, retargeting, and reshuffling for video. Fully automatic stitching of photographs to form a collage was demonstrated by Rother et al. [2006], and it was shown that bidirectional similarity [Simakov et al. 2008] can be used to generate image collages, so our method is extensible to collages as well. Fitzgibbon et al. [2003] obtained state-of-the-art results for new view synthesis by using a patch-sampling objective term and a geometry term. Dense nearest neighbor patch search was also shown to be useful for image denoising [Buades et al. 2005] and learning based superresolution [Freeman et al. 2002]. These are all excellent candidates for acceleration using our algorithm. We also hypothesize that by operating in the appropriate domain, it may be possible to apply our techniques to perform retargeting, hole filling, and reshuffling on 3D geometry, or 4D animation or volumetric simulation sequences. Finally, although the optimization we perform has no explicit neighborhood term penalizing discontinuous offsets (as required in stereo and optical flow), we believe its speed may be of use as a component in vision systems requiring fast, dense estimates of image correspondence, such as object detection and tracking.

In conclusion, we have presented a novel algorithm for quickly computing approximate nearest-neighbor fields between pairs of images or image regions. This algorithm makes such nearest-neighbor matching fast enough that a wide variety of patch-based optimization approaches for image synthesis can now be applied in a real-time interactive interface. Furthermore, the simplicity of the algorithm makes it possible to introduce a variety of high-level semantic controls with which the user can intuitively guide the optimization by constraining the nearest-neighbor search process. We believe we have only scratched the surface of the kinds of interactive controls that are possible using this technique.

Acknowledgements

We would like to thank the following Flickr users for Creative Commons imagery: *Sevenbrane* (Greek temple), *Wili* (boys), *Moi of Ra* (flowers), and *Celie* (pagoda). We thank the paper reviewers, and our internal reviewers at Tiggraph and the Adobe CTL retreat. This work was sponsored in part by Adobe Systems and the NSF grant IIS-0511965.

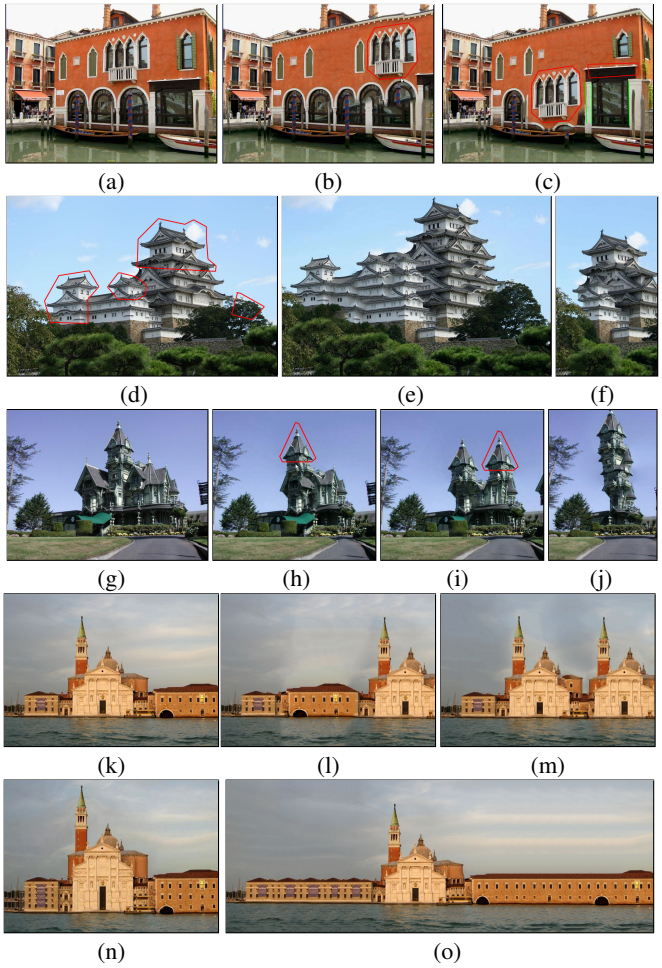


Figure 12: *Modifying architecture with reshuffling. (a-c) The window position for (a) an input image is (b) moved to the right or (c) to the lower story of the building. (d-f) Hard constraints on a building are used to make various renovations. (g) Another house and (h-j) combinations of retargeting and reshuffling produce more extreme remodeling variants. (k-n) The buildings in (k) the input image are (l) horizontally swapped (m) cloned, (n) vertically stretched, (o) and widened.*

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *I3D '06 Proc.*, ACM, 217–226.

AVIDAN, S., AND SHAMIR, A. 2007. Seam carving for content-aware image resizing. *ACM SIGGRAPH 26*, 3, 10.

BUADES, A., COLL, B., AND MOREL, J. M. 2005. A non-local algorithm for image denoising. In *CVPR*, vol. 2, 60–65.

CHO, T. S., BUTMAN, M., AVIDAN, S., AND FREEMAN, W. 2008. The patch transform and its applications to image editing. *CVPR*.

CRIMINISI, A., PÉREZ, P., AND TOYAMA, K. 2003. Object removal by exemplar-based inpainting. *CVPR 2*, 721.

DRORI, I., COHEN-OR, D., AND YESHURUN, H. 2003. Fragment-based image completion. *ACM SIGGRAPH 22*, 303–312.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH*, ACM, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. *ICCV 2*, 1033.

FANG, H., AND HART, J. C. 2007. Detail preserving shape deformation in image editing. *ACM Trans. Graphics 26*, 3, 12.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6, 381–395.

FITZGIBBON, A., WEXLER, Y., AND ZISSERMAN, A. 2003. Image-based rendering using image-based priors. In *ICCV*, 1176.

FREEMAN, W., JONES, T., AND PASZTOR, E. 2002. Example-based super-resolution. *Computer Graphics and Applications, IEEE 22*, 2, 56–65.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. 2001. Image analogies. In *ACM SIGGRAPH*, 327–340.

KOMODAKIS, N., AND TZIRITAS, G. 2007. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE Transactions on Image Processing 16*, 11, 2649–2661.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. *ACM SIGGRAPH 26*, 3, 2:1–2:9.

KUMAR, N., ZHANG, L., AND NAYAR, S. K. 2008. What is a good nearest neighbors algorithm for finding similar patches in images? In *European Conference on Computer Vision (ECCV)*, II: 364–378.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM SIGGRAPH 22*, 3 (July), 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM SIGGRAPH 24*.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM SIGGRAPH 24*, 3, 777–786.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graphics 20*, 3, 127–150.

LIU, F., AND GLEICHER, M. 2005. Automatic image retargeting with fisheye-view warping. In *UIST*, ACM, 153–162.

MOUNT, D. M., AND ARYA, S., 1997. ANN: A library for approximate nearest neighbor searching, Oct. 28.

PAVIC, D., SCHONEFELD, V., AND KOBBELT, L. 2006. Interactive image completion with perspective correction. *The Visual Computer 22* (September), 671–681(11).

RONG, G., AND TAN, T.-S. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *I3D '06 Proc.*, 109–116.

ROTHER, C., BORDEAUX, L., HAMADI, Y., AND BLAKE, A. 2006. Autocollage. *ACM SIGGRAPH 25*, 3, 847–852.

RUBINSTEIN, M., SHAMIR, A., AND AVIDAN, S. 2008. Improved seam carving for video retargeting. *ACM SIGGRAPH 27*, 3.

SETLUR, V., TAKAGI, S., RASKAR, R., GLEICHER, M., AND GOOCH, B. 2005. Automatic image retargeting. In *MUM*, ACM, M. Billinghurst, Ed., vol. 154 of *ACM International Conference Proc. Series*, 59–68.

SIMAKOV, D., CASPI, Y., SHECHTMAN, E., AND IRANI, M. 2008. Summarizing visual data using bidirectional similarity. In *CVPR*.

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. In *ACM SIGGRAPH*, 861–868.

SZELISKI, R., ZABIH, R., SCHARSTEIN, D., VEKSLER, O., KOLMOGOROV, V., AGARWALA, A., TAPPEN, M., AND ROTHER, C. 2008. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE PAMI 30*, 6, 1068–1080.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM SIGGRAPH 21*, 3 (July), 665–672.

WANG, Y.-S., TAI, C.-L., SORKINE, O., AND LEE, T.-Y. 2008. Optimized scale-and-stretch for image resizing. In *ACM SIGGRAPH Asia*, ACM, New York, NY, USA, 1–8.

WEI, L. Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *ACM SIGGRAPH*, 479–488.

WEI, L.-Y., HAN, J., ZHOU, K., BAO, H., GUO, B., AND SHUM, H.-Y. 2008. Inverse texture synthesis. *ACM SIGGRAPH 27*, 3.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2007. Space-time completion of video. *IEEE Trans. PAMI 29*, 3 (March), 463–476.

WOLF, L., GUTTMANN, M., AND COHEN-OR, D. 2007. Non-homogeneous content-driven video-retargeting. In *ICCV*.