Computers & Graphics 000 (2018) 1–8

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

PatchSwapper: A novel real-time single-image editing technique by region-swapping☆

Shizhe Zhou[a,*], Chengfeng Zhou[a,b,**], Yi Xiao[a], Guanghua Tan[a]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[b] Key Laboratory of High Performance Computing and Stochastic Information Processing, Ministry of Education of China, China

ARTICLE INFO

Article history:
Received 6 December 2017
Revised 11 February 2018
Accepted 5 March 2018
Available online xxx

Keywords:
Image editing
Patch-based synthesis
Optimal seam

ABSTRACT

While many image composition and synthesis techniques have been proposed, neither existing work nor professional image editing softwares such as Adobe Photoshop© provide a program explicitly for the task of *realistically swapping regions within a single image*. In this paper, we present an easy-to-use image-editing tool explicitly for that purpose, named as PatchSwapper. Users can simply determine the centre and radius of a pair of candidate regions or specify arbitrary borders by sketches; then, the swapping step is automatically and optimally executed. A graph-based approach is designed to find the optimal borders of two irregular regions to avoid generating visible seams. We use this approach to handle both non-transformed and transformed patches. For non-transformed cases, we not only achieve realtime performance with CPU multi-thread implementation, we also provide a location recommendation algorithm to help users find the appropriate exchangeable areas. For transformed patches, our method searches for the optimal transformation to generate interesting appearance changes on both the source and target locations. A Poisson color blending is performed onto the stitched patches. Overall the proposed approach is not only suitable for swapping image objects of any shapes but also for some other applications, e.g., image completion and composition.

The experiments demonstrate that common images often contain potentially exchangeable areas and that large content variations can be obtained by simply swapping their locations.

1. Introduction

In recent years, mach image editing work has proposed solutions for image composition [1–7], image melding among multiple images [8], image retargeting [9–11], image reshuffling [12–15], and image completion [16–21] in single images. In this paper we explore a special patch-based image editing operation—finding two patches with identical boundaries in a single image and swapping their locations to produce a new image. We called it PatchSwapper. Using PatchSwapper, users can produce a natural yet substantial appearance change [22] in a single image (see Fig. 1 and the accompanying video).

Existing methods for patch swapping require a circular patch [23] or pre-segmentation [3] to find a closed cut. To avoid these limitations, we establish a sparse directed graph between two patches based on the polar coordinate system. Since this graph model are built on pixels instead of patches [24], it can easily adapt to irregularly shaped patches—including non-convex and non-symmetric shapes, and it requires no preprocessing. We search a set of closed cuts and exchange the internal pixels in the optimal cut. The entire process is highly parallelisable. Through experiments we verified that our algorithm achieves nearly a linear speed-up ratio (Fig. 9). Besides, we surprisingly find interesting results generated by our method on transformed patches.

Often, a potentially exchangeable pair is difficult to discern with the naked eye. Therefore, we provide a location recommendation algorithm based on regional image similarity in this study. Specifically, we randomly select pairs of exchangeable areas surrounding each pixel in the input image according to their global similarity, and perform non-maximal suppression on each location according to the local similarity of the central region of the respective patches. A high global similarity guarantees the high quality of an optimal cut, while a low local similarity will lead to drastic appearance changes after swap.

We also show other applications of PatchSwapper in image completion and image composition (Figs. 12 and 13). Our method shows the same level of quality as state-of-the-art patch based

☆ This article was recommended for publication by Shi-Min Hu.

* Corresponding author.
** Corresponding author.
E-mail addresses: shizhe@hnu.edu.cn (S. Zhou), joe_chief@hnu.edu.cn (C. Zhou), yixiao1984@gmail.com (Y. Xiao), guanghuatan@gmail.com (G. Tan).

https://doi.org/10.1016/j.cag.2018.03.002
0097-8493/

Fig. 1. Content Swapping. Above two group images demonstrate the effect of PatchSwapper. In each group, the left image is the original image and the right image is the result after PatchSwapper. (a) the result on non-transformed patches after PatchSwapper one time. (b) the result on transformed patches after PatchSwapper 3 times.

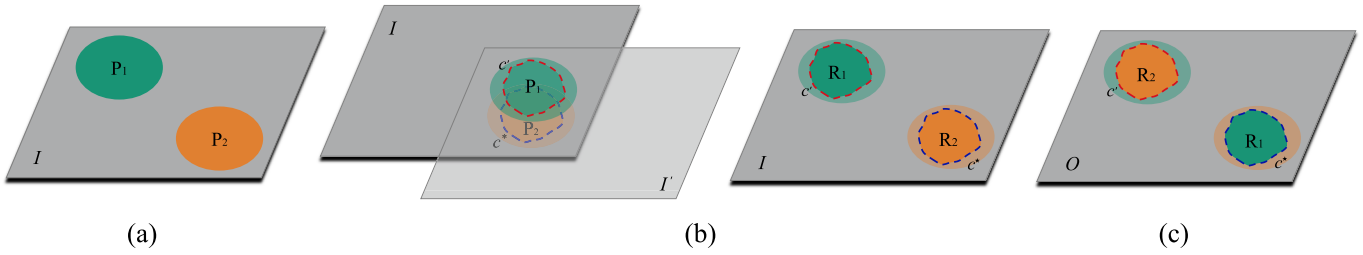

Fig. 2. Overview of the system. (a) Specify two patches P_1 and P_2. (b) Find the optimal cut c^* and $c^{'}$. Overlapping P_1 and P_2, then find the optimal cut c^* on this overlapping region. $I^{'}$ is the copy of I and $c^{'}$ is the duplicate of c^* on P_1. (c) Swap two regions R_1, R_2 to produce the output O. R_1 and R_2 are enclosed by c^* and $c^{'}$ respectively.

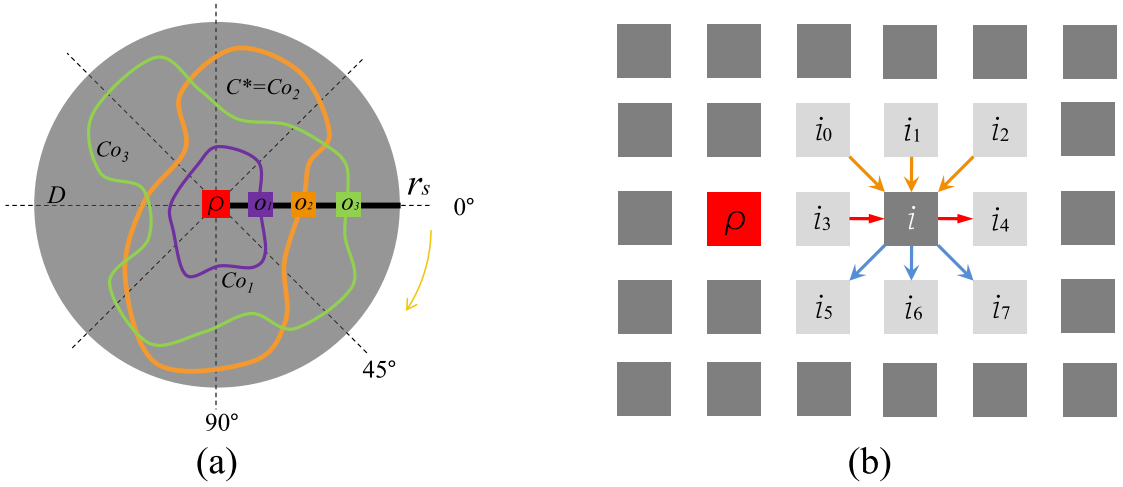

Fig. 3. Optimal cut searching. (a) We establish a counterclockwise polar coordinate system on pixel difference map D centered at pixel ρ. Horizontal radius r_s denote the starting line of our closed path searching. For each node o_j on r_s, we compute an optimal closed path that starts and ends at the same pixel. Then, we select the optimal path c^*, here for instance $c^* = c_{o_2}$ (the thick orange curve), from all these paths. (b) Zoom-in view of D near the pole ρ. Let's consider the connection between node i and its 1-ring neighborhood. i_5, i_6, i_7 have polar angle greater than i, so we define edges emanating from i to them (blue arrows). i_3, i_4 have the same polar angle with i but have different radius, so we define edges $i_3 \rightarrow i$, $i \rightarrow i_4$(red arrows). Specifically we define edges $i_0 \rightarrow i$, $i_1 \rightarrow i$ and $i_2 \rightarrow i$ (orange arrows) to ensure the closure of paths. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

methods [8,18–21,25]. To summarise, our main contributions are as follows.

- We introduce a special and useful patch-based image operation—PatchSwapper. PatchSwapper not only can exchange the location of two objects directly and precisely (Fig. 1(a)), not need to take extra time to fill the area outside the selected area compared to image reshuffling [12–15]; but also can create new structured patterns (Fig. 1(b)).
- We propose a novel graph algorithm to efficiently search for the optimal closed cut. This approach achieves real-time performance even on a consumer-level CPU and is suitable for any patch shape (Fig. 9).
- We present a helper feature consisting of a location recommendation algorithm based on regional image similarity. This feature helps users discover potentially exchangeable patches in an image without having to perform exhausting test-and-evaluate procedures (Figs. 7 and 8).

2. Related work

Texture Synthesis is an important aspect of computer graphics that focus on creating regular or semi-regular textures from small exemplars. Earlier texture synthesis algorithms were based on pixels and utilise pixel adjacency [26–28] or Markov Random Fields (*MRF*) [29] to perform the synthesis. Compared with pixel-based image synthesis, patch-based methods [23,30] are better at capturing and preserving structures because they copy regions rather than only pixels from the exemplar. A highly parallel algorithm

Fig. 4. Swapping results of our method on non-transformed patches. The input (original) images are marked with a small magenta square on their top left corners.

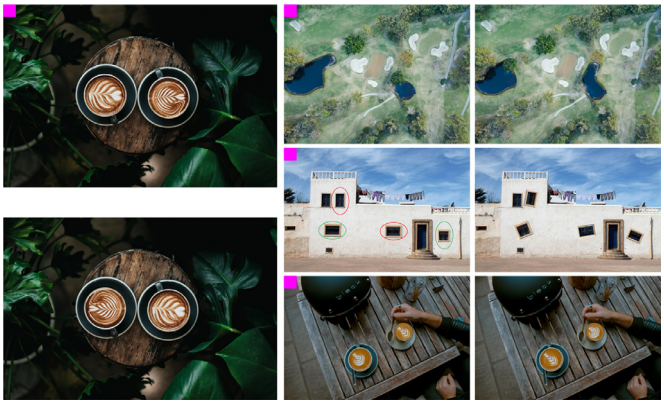

Fig. 5. Swapping results of our method on transformed patches. The patches are allowed to be rotated and scaled. Our algorithm automatically searches for the best transformation by testing all combination with rotation angle from 0 to 360, and the scale ratio 1.0 to 1.3.

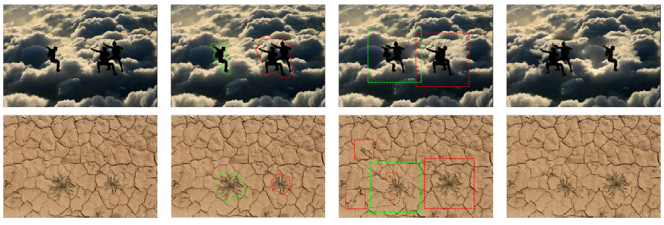

Fig. 6. Swapping result using specified regions. From left to right are the input images, arbitrary boundaries specified by user sketches, the swapping result of the rectangle regions, the swapping result of specified boundaries, in which the integrity of the object is better preserved.

is presented in [23] that iteratively lays out patches and stitches them together using the parallel processing power of a *GPU*. Using polar coordinates, they transform the colour difference map into a Dynamic Programming (*DP*) table. However, this step causes little distortions, especially for pixels near the polar centre. Moreover, this method must perform the coordinate transformation again when the cut is computed. Our method does not require any coordinate transformation or any pixel interpolations to find the cut.

Patch-based Image Processing has been successfully applied for various editing tasks on photos [12], photo collections [24], videos [31], and light fields [32]. Barnes and Zhang [33] has investigated recent papers in this topic since 2009 to 2017. It has summarized the existing methods and divided these into two main stages: *matching* and *blending*. Suitable patches copying from exemplars will be found at the *matching* stage, and be combined and blended together on the output image at the *blending* stage. PatchTable [31] has introduced a data structure to efficiently query the approximate nearest neighbor in large datasets. They precompute a multidimensional hash table which maps a hash bucket to an exemplar patch in datasets, and use Locality-sensitive hashing (LSH) to map the query patch to the hash table cells.

Poisson Blending is a commonly method used in image editing that seamlessly blends colours from two images without visible discontinuities around the boundary by solving a Poisson equation [34]. Therefore, it is often used to enhance the combined results [1,3,5,8]. The effectiveness of Poisson blending depends on the colour-smoothness of the boundary. Drag-and-drop pasting [3] presents a shortest closed-path algorithm to iteratively optimise the location of the best boundary. Their method first removes the interior part of the patch and then computes a closed path on the remaining narrow band. In contrast, we build a graph model directly from the whole patch. In addition, our method involves only a single dynamic programming step and does not require preprocessing, e.g., segmentation or object cutout.

Optimal Seam in general, the optimal seam method uses an energy function defined on the pixels. Then, it finds a vertical or horizontal curve in a pixel-based graph. Optimal seam is a common approach in image retargeting [9] and for applications such as objection insertion [1], texture synthesis [25,35,36] and superresolution [37]. The Seam Carving method proposed by [9] employs a simple content-aware image resizing operation by removing or inserting a seam. This method supports various types of energy functions, such as sum-of-squared-differences (*SSD*), gradient magnitude, entropy, visual saliency, eye-gaze movement, and others. In [35], the authors cast synthesis as a shortest-path graph problem. Each path in the graph implies how to form the result by cutting strips from the source image and reassembling them in a specific order. They use *SSD* as the energy function to achieve a content-aware cut searching mechanism. In [25], the authors introduce the Graph-cut Textures, a graph cut technique to find a minimum cut in a special graph model. Compared to the image quilting [36], this scheme produced better results in many cases.

3. Region swapping

Given a single image, we assume the task is to swap a pair of patches from different locations. Achieving this task faces the following main technical challenges. (1) The candidate regions must have exactly the same shape and the same outline so that no gap will exist after the swapping operation. (2) The colour differences along the boundaries of the candidate patches should be minimal; the goal is to have no visible seam after the swapping operation. (3) To meet real-time requirements, the algorithm must be lightweight and structurally simple. In addition, an implicit requirement exists: because people naturally tend to desire a drastic visual appearance change after editing, the two candidate patches should be as large as possible.

Our approach to seamless swapping is to search for repeated content in the form of circular paths along which similar colours are observed [35], rather than similar objects [11,38–40]. In the traditional optimal seam approach [1,2,35,36], the cut follows an open curve segment. In contrast, to solve our problem, we must find

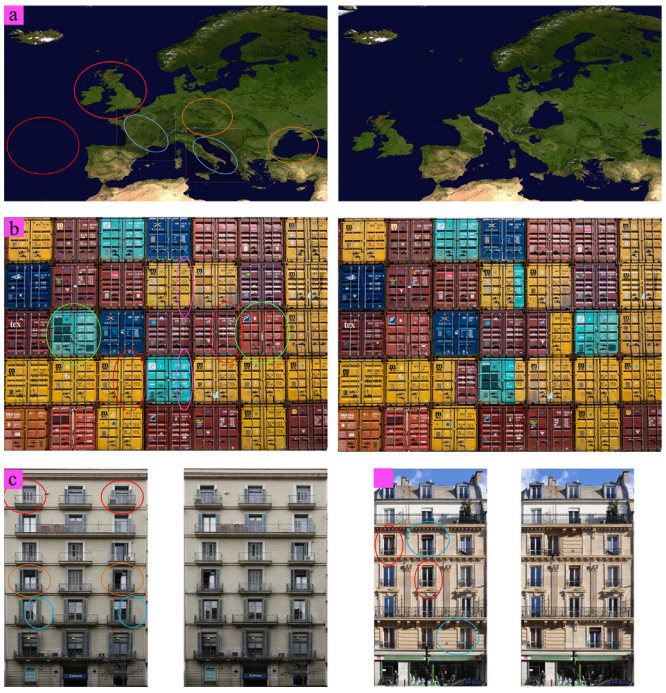

Fig. 7. Results of recommendation algorithm on images with complicated structure. The pairs of exchanged regions are marked by the ellipses with the same colour in the input image (left).

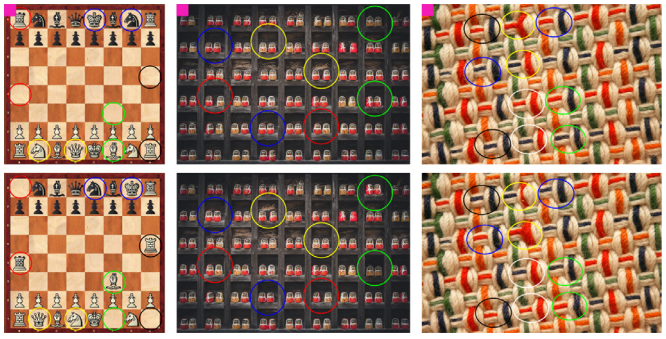

Fig. 8. Results of recommendation algorithm on images with regular structure. The pairs of exchanged regions are marked by ellipses with the same colour in the input images and the output images. Since the patches P_1, P_2 are aligned and have a high Sim_{global}, the results well preserve the original structure in input image.

a closed cut on a difference map of two patches. However, finding a closed path on an undirected graph [3,23] is an intractable problem. Therefore, to make it tractable, we convert the overlapping pixel region into a well-organised directed graph by defining a novel graph-model on the difference map based on the polar coordinate system. Then, a shortest path algorithm [41] is employed at different pixels in parallel to compute a set of closed shortest paths. Finally, we select the best closed shortest path as the optimal cut. This process is highly parallelisable.

The entire process of PatchSwapper is performed in three stages. As shown in Fig. 2, the first stage involves specifying two patches of the same size P_1 and P_2 which should not overlap mutually due to the variation of neighbours after swapping. W.l.o.g, we first describe our algorithm using circular patches. However, the implemented algorithm uses square patches and has no problems. The second step finds an optimal cut c^* on a difference map, which can be regarded as finding two cuts of the same shape c^* and c' in the respective patches P_1, P_2 where c^*, c' enclose the respective regions R_1, R_2. Finally, we swap the paired pixels in R_1 and R_2. Sections 3.1 and 3.2 describe these steps in more detail.

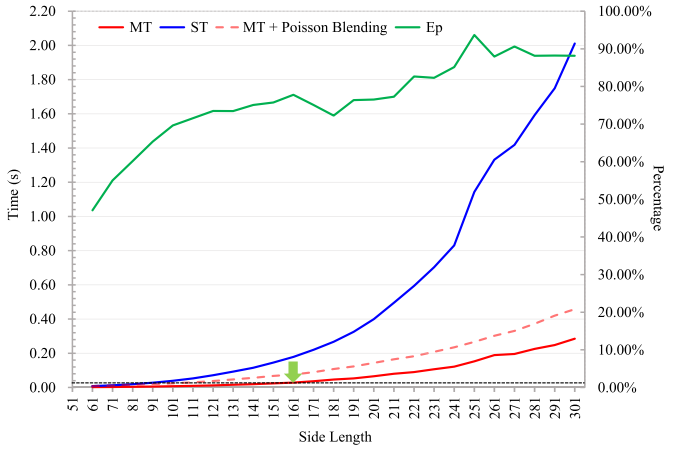

Fig. 9. Performance analysis of the optimal cut search procedure in single threaded and multi-threaded CPU modes. The parallel efficiency E_p of our algorithm is approximately 0.9, while our multi-thread implementation achieves 10 FPS for a patch width of 221 pixels and obtains a realtime performance when the patch width less than 161 pixels. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.1. Graph model

Using a polar coordinate system centred at pixel ρ, a directed graph can be built whose nodes are the pixels in the difference map and whose edges are specifically defined as suitable for our problem. We define weights on the nodes instead of edges. Then, searching for an optimal cut to swap the patches can be reduced to finding a closed shortest path in the directed graph.

First, it is necessary to choose a matching quality measure that has only a small computational cost for the pixels from P_1 and P_2. The sum of squared differences (*SSD*) is an ideal measure because it measures the colour difference between pixel pairs in the simplest way. We compute the difference map D on the overlapping region derived from completely aligning the two patches. D is a disk with radius r equal to the radius of the user-selected patch. Let i denote a pixel position in the overlapping region. The difference $D(i)$ has the following form:

$$D(i) = \|P_1(i) - P_2(i)\| \tag{1}$$

In all our experiments, we compute $D(i)$ in the *RGB* colour space.

The pole and the reference direction are the two most essential concepts of the polar coordinate system. Each polar point on a plane is determined by a distance from the pole and an angle from the reference direction. In Fig. 3(a), we define the centre pixel ρ of D as the pole, and the horizontal radius r_s (black) as the reference direction.

On this basis, we construct the directed graph g, in which the weight of each node i is equal to $D(i)$. We take advantage of the polar coordinate to determine whether the nodes are adjacent. Fig. 3(b) shows the 8-neighbourhood connections of node i to its neighbours. We link node i to the nodes whose polar angle is greater than i (blue arrows) or whose polar angle is equal to i but whose radius is greater than i (red arrows). The latter type of edges extends the searching space, allowing better cuts to be found. The orange edges exist only immediately above the radius r_s. Such a connection is crucial for closed paths to exist in the graph. The gradual increase in polar angle ensures that a closed path is finally formed. The directed graph g is a sparse graph in which the number of edges is approximately 4 times the number of nodes, and it

is applicable for any patch shape. Here, because the topology of our graph remains constant, the directed graph can be pre-computed and stored. Subsequently, each time a path-searching procedure is initiated, we need only recalculate its weights.

Because the procedures of searching for a shortest path starting at different nodes is identical for every node [41], these procedures can be executed without any data-exchange, runtime communication, or synchronization, which makes this procedure highly suitable for parallel execution on multi-core platforms (Fig. 9).

3.2. Optimal cut

Note that we obtained the shortest path set C starting from and ending at each node located in r_s (the thick solid black line in Fig. 3(a)). To achieve a better editing effect, a patch swapping operation should seamlessly exchange regions that contain rich visual content. Achieving this goal requires finding the largest possible image regions enclosed by an optimal cut. To accomplish this, we select one optimal cut from a set of shortest paths obtained from the procedure described in the preceding section.

We filter out paths that contain too few pixels because exchanging small areas will not introduce much variation. Because the operation that counts the number of internal pixels within the irregular closed path is time consuming, we use the path bounding box to estimate the number of pixels it contains and discard paths that contain fewer internal pixels than $|D|/4$. $|D|$ is the number of pixels in the difference map D. The remaining paths are represented by $C_{o_j},\ o_j \in r_s$.

For these remaining shortest paths, simply using the total difference in path length to select the optimal cut is unreasonable. Because each path has a different length in pixels, we introduce the concept of average difference:

$$M(c) = \frac{\sum_{i\in c} D(i)}{|c|} \tag{2}$$

where $|c|$ is the number of nodes on c. We select the shortest path with the minimum $M(c)$ value as the optimal cut $c^* = \arg\min_{c\in C_o}(M(c))$ (as shown in Fig. 3(a)). We obtain the exchange regions R_1, R_2 through c^* and its duplication c' (Fig. 2(b)). Finally, we swap R_1 and R_2 to produce the final output O. The output O can be further enhanced by poisson blending [34] to eliminate the color gap (Figs. 4 and 5). Intuitively, O is more effective when the patches, P_1 and P_2, are larger. However, increasing r excessively tends to filter out promising cuts that can result in plausible appearance changes.

In some cases users wish to swap two specific objects without being cut through by the pathes (Fig. 6). Here we let the user draw the borders of two regions, then the union of two regions is computed by aligning their centroids. Next we perform a *restricted* PatchSwapper by modifying the structure of our graph g, i.e., removing the nodes inside the union(including those lying on the border of the union). Thus the *restricted* PatchSwapper strictly protects the integrity of the object enclosed by the border(Fig. 6) after the seamless swapping. More details are shown as animation in the accompanied video.

4. Swapping location recommendation

Finding potentially non-transformation exchangeable regions that exist in an image is often not easy with the naked eye; consequently, a recommendation algorithm that suggests reasonable candidate swap pairs to users is helpful. This recommendation algorithm is based on regional similarity, global similarity Sim_{global} and local similarity Sim_{local}. The final candidate proposals do not overlap each other and will yield a high-quality output. After multiple recommendations and decisions, the final synthesized image can be formed (see Figs. 7 and 8).

Table 1
Different size of the local portion. Setting the patch size to 161 * 161, we have collected after 1000 times recommendation on the image from Fig. 7. (a), (b), (c). It has shown a good balance between the average difference $M(c)$ and the number of pixels exchanged (denoted by percentage) when local portion size equal to $(r+1)^2$.

Portion size	Fig. 7(a)		Fig. 7(b)		Fig. 7(c)	
	M(c)	Percentage	M(c)	Percentage	M(c)	Percentage
41 * 41	0.001532	28.36%	0.010013	29.93	0.001356	39.22
81 * 81	0.001881	29.65%	0.010108	33.83	0.001358	46.25
121 * 121	0.002421	31.67%	0.011542	34.02	0.001637	47.33

Here we define global similarity as follows:

$$Sim_{global}(P_1, P_2) = \frac{1}{|D|}\sum_{i\in D} D(i) \tag{3}$$

where D is the difference map of P_1, P_2. We compute the global similarity for every possible pairable location (usually set the step to 10) and select the pair of swapping locations stochastically for each location l in the input image according to following probability function:

$$P(P_l, P^*) \propto e^{-\frac{Sim_{global}(P_l, P^*)}{k\sigma^2}} \tag{4}$$

where P_l is the patch centering in l, P^* denote the corresponding patches of P_l, σ is the standard deviation of the pixel values in the input image and k controls the randomness in pair selection. A low k value picks similar regions whereas a high k value selects pairs more randomly. In our experiments, we set k to 0.1. Global similarity ensures that the cut is more likely to pass through low-difference nodes and implicitly aligns the main structure of the input image (see Fig. 8).

After picking candidate pairs for each location l, non-maximal suppression is performed on those pairs according to the local similarity, which rewrote Eq. (3):

$$Sim_{local}(P_1, P_2) = \frac{1}{|D_p|}\sum_{i\in D_p} D(i) \tag{5}$$

where D_p is the difference map of local portion p. We define the "local portion" in the patch to be the area centred around the pole whose radius is equal to $r/2$. This radius given a good balance between the average difference $M(c)$ and the number of pixels exchanged in our experiment (Table 1). When this process is complete, the remaining pairs do not overlap each other and their potential exchangeable regions have distinct appearances. We then search for the optimal cut only among the remaining pairs to provide final recommendations to the user.

In our implementation, we set P_1 and P_2 to be squares of size $(2r+1)^2$ due to the convenience on follow steps. Thus, the local portion is a square with a width of $r+1$. We can rewrote $D(i)$ (1) to obtain the follow equation:

$$\sum_{i\in D} D(i) = \sum_{i\in D} P_1^2(i) + \sum_{i\in D} P_2^2(i) - \sum_{i\in D} P_1(i)P_2(i) \tag{6}$$

when calculating global similarity Sim_{global}. The first two terms in (6) are the sum of squares of the pixel values over P_1 or P_2. These values can be computed efficiently in $O(m)$ time using summed-area tables. The sum of the third term is a convolution of the input with the output and can be computed in $O(m\log m)$ time using an *FFT*. Here, m is the number of pixels in the difference map D. We also use this method [25] to calculate the local similarity Sim_{local}, and achieved a huge performance acceleration.

We validated the effectiveness of our recommendation algorithm in two aspects: the average difference $M(c)$ (Eq. (2)) normalized to range [0,1] and the number of internal pixels. The data

Table 2
Comparison between recommended positions and random picked positions. There are significant improvements in the *M(c)* and the number of internal pixels.

Patch size	Recommended position		Random picked position	
	M(c)	Pixel number	*M(c)*	Pixel number
121*121	0.002312	5945.92	0.007329	5707.96
141*141	0.002270	8343.26	0.006521	7677.32
161*161	0.002118	10990.75	0.005897	10188.81

Table 3
Percentages of unproductive cuts in our approach and [23]. Most of results produced by [23] enclose very few pixels, even no pixels. Therefore, it can not bring an obvious visual change in the input image after swapping.

Pixel number	[23]	Ours
0	16.10	0.00
Below 10	42.66	8.51
Below 100	55.60	22.70

in Table 2 was obtained on 5 different images, randomly picking 1000 locations or selecting 1000 recommended locations on each image. Since a drastic decrease on *M(c)* and a significant increase on internal pixels number have emerged on the table, we can find out that the recommended algorithm can greatly improve the quality of results. Furthermore, that algorithm is well-suited for a grid structure image (Fig. 8).

5. Application and results

We tested our method for non-transformed patches on a wide range of images. Here we present the results and analyse the method's performance in both single and multi-threaded modes. Furthermore, we compared our method to the state-of-the-art method [23] and applied it to image completion and image composition. All the results were obtained running on an Intel I7 7700K 4.2 GHz using 4 cores and 8 threads for computations.

For the sake of implementation simplicity and runtime efficiency in practice, as mentioned earlier, we used square patches rather than circular patches. We denote the width of the square patch by ω.

Time consumption. Through the experiments, we observed that the parameter with the greatest impact on both image quality and performance is the patch width ω. We tested the proposed method on 5 images while varying ω *from 61 to 301 at a step size of 10*. For each ω we randomly selected 1000 locations to compute the optimal cut. Fig. 9 shows the performance results of the experiments. As shown in Fig. 9, we achieved interactive swapping for patches up to 221 * 221 in size. As we mentioned before, oversized patches may filter out promising cuts. A patch width of $\omega = 221$ provides an acceptable balance between plausible results and the interactive runtime performance (see the accompanying video). Furthermore, we plotted the parallel efficiency of the algorithm (the green line shown in Fig. 9), which shows that the speed-up ratio of the algorithm is approximately linear as the patch width increases.

Comparison with state-of-the-art methods. The authors of [23] also proposed a fast approximate closed-cut algorithm using polar coordinates and dynamic programming. There are some fundamental differences between their method and ours: (1) Their method transforms the circular selected region to a rectangular patch. Consequently, a pixel in the selected region may correspond to multiple pixels in the rectangular areas. So their results on two patches with similar central areas will enclose very few pixels (Usually less than 10, see Table 3 and Fig. 10(b) and (c)). (2) their method can not be straightforwardly generalized to non-circular irregular shapes. Forcing the mapping from an irregular shape patch to a circle using a mask wastes *DP* table space (Fig. 10(a)).

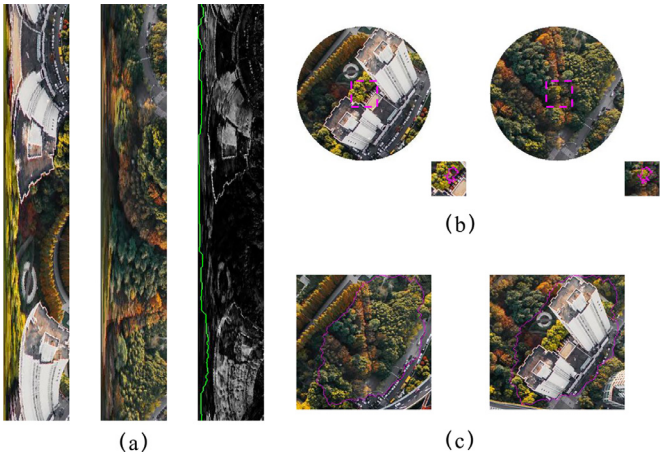

Fig. 10. Results of the patch swapping method in [23]. (a) The $\mathcal{P}_{polar}$ and its difference map in [23]; (b) Because method in [23] has to transform the circular patches to a rectangular *DP* table, in many cases (e.g., the area in the difference map near the polar point (left side), which has low error) the periodic path in the *DP* table tends to enclose a very small image region. (c) Our method produces a significantly larger patch for swapping.

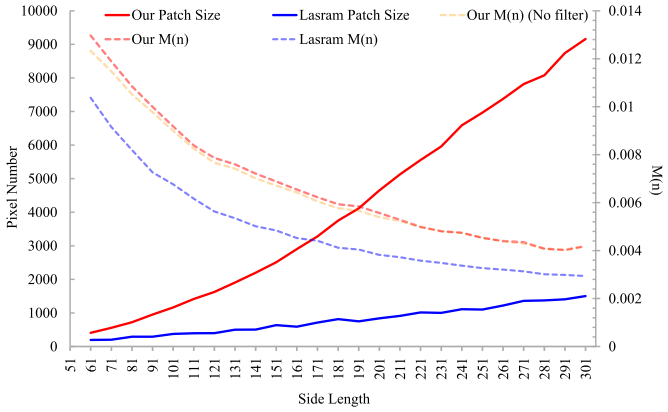

Fig. 11. Performances of our approach and that in [23] estimated by the number of internal pixels and *M(c)*. In our experiment, we compare our method (no filter) with the method in [23] in above criterions. The former (blue dotted line) is slightly better than the latter (red dotted line) on *M(c)*, which is merely about 0.0014. However, there is a considerable gap on pixels number between their method (blue solid line) and our method without filter (red solid line), reaching the peak at 7658 pixels in patch width of 301 pixels. In addition, we provide the data (orange dotted line) of our whole method (Section 3.2) in *M(c)* on the chart. From the chart, we can observe that the difference in *M(c)* between the whole version and the no filter version is quite small. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We implemented the method in [23] using the same energy function (Eq. (1)) and compare with our method. Here, we set the radius of the selected region to $\omega/2$ for the method in [23]. While varying *the* ω *from 61 to 301 at a step size of 10*, we obtained the respective results on same locations which are randomly picked in 1000 positions per image on 5 different images. We compare our method which does not filter small exchangeable regions with the method in [23] rather than the whole approach due to the impact of filter on the number of internal pixels. Although [23]'method has a faster speed and a small gap on *M(c)*, we may reasonably conclude that our method is more suitable on the task of exchanging two regions compared to the method in [23] because of the high proportion of unproductive cuts which enclose few pixels with small *M(c)* in their results (see the Table 3). In fact, our result still looks authentic and evidently demonstrates the change of appearance after swapping in spite of the narrow margin of *M(c)*.

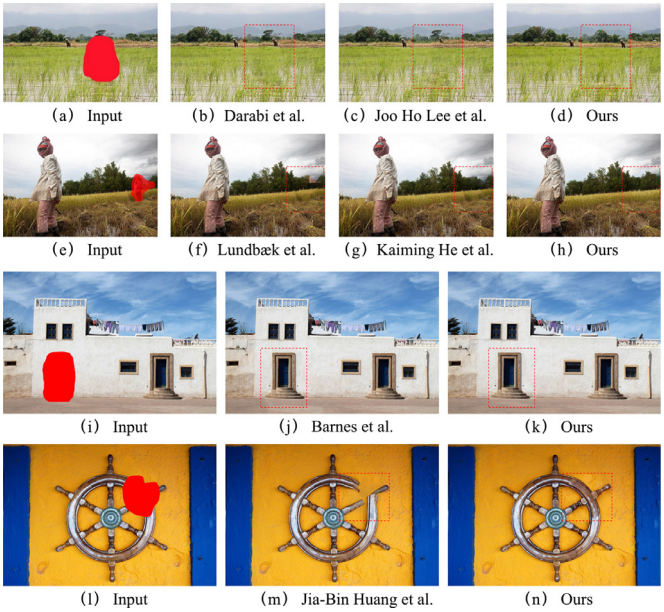

Fig. 12. Image completion (a) Input image in which the red region needs to be completed; (b) Darabi et al. [8]; (c) Joo Ho Lee et al. [21]; (d) our method with only 1 swap on non-transformed patch; (e) input image; (f) Lundbæk et al. [18]; (g) Kaiming He et al. [19];(h) our method with only 1 swap on non-transformed patch; (i) input image; (j) Barnes et al. [31],the run time=38.91s; (k) our method with only 1 swap on transformed patch,the run time=1.082s + time for interaction; (l) input image; (m) Jia-Bin Huang et al. [20] and (n) our method with only 1 swap on transformed patch. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

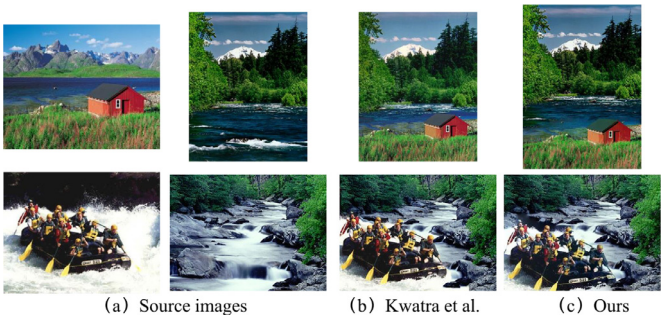

Fig. 13. Examples of interactively combining two source images. (a) Sources images; (b) Kwatra et al. [25]; and (c) our method.

In Fig. 10(b) and (c) we show the comparison results on identical patch locations of the same image.

Applications. Our approach can also be used in other image processing contexts. Fig. 12 shows the application of image completion. Unlike the fully automatic image completion methods [8,18–21,31] our algorithm requires user intervention to obtain satisfactory results, but the whole process is not time-consuming (about 20 to 40 seconds) and the results will have less artifacts compared with state-of-the-art methods in some cases give a correct user intervention (Fig. 12). However it consumes more time to do multiple trial-and-error when the missing region requires largely-transformed patches to fill (Fig. 12 (l)). Also we do not consider explicitly the perspective transformation and synthesis new content from small patches [31], so our method could fail in the images with strong perspective effect [20] and the images with few available examplars.

For image composition, unlike the approach in [13] and [6], users can interactively drag a large selected patch into a new location in the target image in a WYSIWYG (what you see is what you get) way, without any computation for a new background or adjustment for the location of the patch. Here, we compare our method with that of [25] in Fig. 13. Although the results are very similar, the near realtime performance of our algorithm results in a better interactive experience.

Fig. 14. Incorrect patch locations can lead to implausible results. In this example, the forest cut off the overpass.

6. Discussion and limits

In this study, we revealed that common images contain many possible exchangeable areas, and we proposed a concise algorithm to perform such exchanges. In addition, to aid users, we proposed a recommendation algorithm based on regional similarity. This algorithm recommends exchangeable region candidates to the user. These recommendations offer possibilities for appealing synthesized images without users having to perform tedious editing and image-manipulation tasks.

The limits of PatchSwapper are 2-fold. One is that each node of our directed graph-model represents only the colour differences of one pixel: neither any neighbouring structure nor any semantic information is considered. This approach can produce artefacts (Fig. 14) when the boundaries of the swapped patch match but the inner structures of the two patches do not match. This problem can be improved easily by using a higher dimensional appearance space. The other is that our synthesis quality is highly dependent on the location of the patch centre. Unrealistic results could be generated if the patch location is incorrect. For example, in Fig. 14, the forest is replaced by a highway segment. The underlying reason for this problem is that searching for the best swapping location (as well as the rotation angle and the scaling ratio) is computationally intensive; therefore, in future work, we plan to design an efficient search procedure using the *GPU*. We will also plan to continue exploring how to apply this technique to video editing.

Acknowledgement

The authors would like to thank the reviewers for their positive and constructive comments.Shizhe Zhou and Chengfeng Zhou is supported by the grant of National Science Foundation of China (No. 61303147), Science Foundation of Hunan Province (No. 2018JJ3064) and HPCSIP Key Laboratory, Ministry of Education. Yi Xiao is supported by the grant of National Science Foundation of China (No. 61502158), Science Foundation of Hunan Province (No. 2017JJ3042).Guanghua Tan is supported by the grant of National Science Foundation of China (No. 61602165), Science Foundation of

Hunan Province (No. 2018JJ3074). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cag.2018.03.002

References

[1] Agarwala A, Dontcheva M, Agrawala M, Drucker S, Colburn A, Curless B, Salesin D, Cohen M. Interactive digital photomontage. ACM Trans Graph 2004;23(3):294–302.
[2] Jia J, Tang C-K. Eliminating structure and intensity misalignment in image stitching. In: Proceedings of the tenth IEEE international conference on computer vision - volume 2. ICCV '05. Washington, DC, USA: IEEE Computer Society; 2005. p. 1651–8. doi:10.1109/ICCV.2005.87. ISBN 0-7695-2334-X-02
[3] Jia J, Sun J, Tang CK, Shum HY. Drag-and-drop pasting. ACM Trans Graph 2006;25(3):631–7.
[4] Jia J, Tang C. Image stitching using structure deformation. IEEE Trans Pattern Anal Mach Intell 2008;30(4):617–31.
[5] Sunkavalli K, Johnson MK, Matusik W, Pfister H. Multi-scale image harmonization. ACM Trans Graphs 2010;29(4):1–10.
[6] Pritch Y, Poleg Y, Peleg S. Snap image composition. In: Proceedings of international conference on computer vision / computer graphics collaboration techniques and applications; 2011. p. 181–91.
[7] Diamanti O, Barnes C, Paris S, Shechtman E, Sorkine-Hornung O. Synthesis of complex image appearance from limited exemplars. ACM Trans Graphs 2015;34(2):1–14.
[8] Darabi S, Shechtman E. Image melding: combining inconsistent images using patch-based synthesis. ACM Trans Graphs 2012;31(4):82.
[9] Avidan S, Shamir A. Seam carving for content-aware image resizing. ACM Trans Graphs 2007;26(3):10.
[10] Wang Y-S, Tai C-L, Sorkine O, Lee T-Y. Optimized scale-and-stretch for image resizing. ACM Trans. Graph. (TOG) 2008;vol. 27:118. ISBN 978-1-4503-1831-0
[11] Dong W, Zhou N, Lee TY, Wu F, Kong Y, Zhang X. Summarization-based image resizing by intelligent object carving. IEEE Trans Vis Comput Graph 2014;20(1):111–24.
[12] Barnes C, Shechtman E, Finkelstein A, Dan BG. Patchmatch: a randomized correspondence algorithm for structural image editing. ACM Trans Graphs 2009;28(3):1–11.
[13] Pritch Y, Kav-Venaki E, Peleg S. Shift-map image editing. In: Proceedings of IEEE international conference on computer vision; 2009. p. 151–8.
[14] Cho TS, Butman M, Avidan S, Freeman WT. The patch transform and its applications to image editing. In: Proceedings of 2008 IEEE conference on computer vision and pattern recognition; 2008. p. 1–8. doi:10.1109/CVPR.2008.4587642.
[15] Simakov D, Caspi Y, Shechtman E, Irani M. Summarizing visual data using bidirectional similarity. In: Proceedings of IEEE conference on computer vision and pattern recognition; 2008. p. 1–8.
[16] Sun J, Yuan L, Jia J, Shum HY. Image completion with structure propagation. ACM Trans Graphs 2005;24(3):861–8.
[17] Komodakis N, Tziritas G. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. IEEE Trans Image Process 2007;16(11):2649–61.
[18] Lundbk K, Malmros R, Mogensen EF. Image completion using global optimization. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition; 2006. p. 442–52.
[19] He K, Sun J. Statistics of patch offsets for image completion. In: Proceedings of European conference on computer vision; 2012. p. 16–29.
[20] Huang JB, Kang SB, Ahuja N, Kopf J. Image completion using planar structure guidance. ACM Trans Graphs 2014;33(4):1–10.
[21] Lee JH, Choi I, Kim MH. Laplacian patch-based image synthesis. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR); 2016. p. 2727–35. doi:10.1109/CVPR.2016.298.
[22] Ma LQ, Xu K, Wong TT, Jiang BY, Hu SM. Change blindness images. IEEE Trans Vis Comput Graph 2013;19(11):1808–19.
[23] Lasram A, Lefebvre S. Parallel patch-based texture synthesis. In: Proceedings of ACM SIGGRAPH / Eurographics conference on high-performance graphics; 2012. p. 115–24.
[24] Hu SM, Zhang FL, Wang M, Wang J, Wang J. Patchnet: a patch-based image representation for interactive library-driven image editing. ACM Trans Graphs 2013;32(6):196.
[25] Kwatra V, Schdl A, Essa I, Turk G, Bobick A. Graphcut textures: Image and video synthesis using graph cuts. ACM Trans Graph SIGGRAPH 2003 2003;22(3):277–86.
[26] Efros AA, Leung TK. Texture synthesis by non-parametric sampling. In: Proceedings of the seventh IEEE international conference on computer vision; 2000. p. 1033.
[27] Wei LY, Levoy M. Fast texture synthesis using tree-structured vector quantization. In: Proceedings of conference on computer graphics and interactive techniques; 2000. p. 479–88.
[28] Zelinka S, Garland M. Jump map-based interactive texture synthesis. ACM Trans Graphs 2004;23(4):pgs.930–962.
[29] Paget R. Strong Markov random field model. IEEE Trans Pattern Anal Mach Intell 2004;26(3):408–13.
[30] Kaspar A, Neubert B, Lischinski D, Pauly M, Kopf J. Self tuning texture optimization. In: Proceedings of computer graphics forum; 2015. p. 349–59.
[31] Barnes C, Zhang FL, Lou L, Wu X, Hu SM. Patchtable: efficient patch queries for large datasets and applications. ACM Trans Graphs 2015;34(4):97.
[32] Zhang FL, Wang J, Shechtman E, Zhou ZY, Shi JX, Hu SM. Plenopatch: patch-based plenoptic image manipulation. IEEE Trans Vis Comput Graph 2017;23(5):1561–73.
[33] Barnes C, Zhang F-L. A survey of the state-of-the-art in patch-based synthesis. Comput Vis Med 2017;3(1):3–20. doi:10.1007/s41095-016-0064-2.
[34] Rez P, Gangnet M, Blake A. Poisson image editing. ACM Trans Graphs 2003;22(3):313–18.
[35] Lefebvre S, Hornus S, Lasram A. By-example synthesis of architectural textures. ACM Trans Graphs 2010;29(4):84.
[36] Efros AA, Freeman WT. Image quilting for texture synthesis and transfer. In: Proceedings of conference on computer graphics and interactive techniques; 2001. p. 341–6.
[37] Freeman WT, Pasztor EC, Carmichael OT. Learning low-level vision. Int J Comput Vis 2000;40(1):25–47.
[38] Cheng M-M, Zhang F-L, Mitra NJ, Huang X, Hu S-M. Repfinder: finding approximately repeated scene elements for image editing. ACM Trans Graphs 2010;29(4) 83:1–8.
[39] Huang H, Zhang L, Zhang H. Repsnapping: efficient image cutout for repeated scene elements. Comput Graph. Forum 2011;30(7):2059–66.
[40] Kong Y, Dong W, Mei X, Zhang X, Paul JC. Simlocator: robust locator of similar objects in images. Vis Comput Int J Comput Graph 2013;29(9):861–70.
[41] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. In: Proceedings of symposium on foundations of computer science, 1984.; 1987. p. 338–46.