Machine Vision and Applications (2014) 25:1841–1858
DOI 10.1007/s00138-014-0637-y

ORIGINAL PAPER

# Automatic inpainting by removing fence-like structures in RGBD images

**Qin Zou · Yu Cao · Qingquan Li · Qingzhou Mao · Song Wang**

Received: 7 August 2013 / Revised: 26 February 2014 / Accepted: 1 May 2014 / Published online: 8 August 2014


**Abstract** Recent inpainting techniques usually require human interactions which are labor intensive and dependent on the user experiences. In this paper, we introduce an automatic inpainting technique to remove undesired fence-like structures from images. Specifically, the proposed technique works on the RGBD images which have recently become cheaper and easier to obtain using the Microsoft Kinect. The basic idea is to segment and remove the undesired fence-like structures by using both depth and color information, and then adapt an existing inpainting algorithm to fill the holes resulting from the structure removal. We found that it is difficult to achieve a satisfactory segmentation of such structures by only using the depth channel. In this paper, we use the depth information to help identify a set of foreground and background strokes, with which we apply a graph-cut algorithm on the color channels to obtain a more accurate segmentation for inpainting. We demonstrate the effectiveness of the proposed technique by experiments on a set of Kinect images.

**Keywords** Depth map · Image segmentation · Image inpainting · Graph cuts · Morphological watershed

Q. Zou (✉)
School of Computer Science, Wuhan University, Wuhan 430072, People's Republic of China
e-mail: qzou@whu.edu.cn

Y. Cao · S. Wang
Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA
e-mail: cao@cec.sc.edu

S. Wang
e-mail: songwang@cec.sc.edu

Q. Li
Shenzhen Key Laboratory of Spatial Information Smart Sensing and Service, Shenzhen University, Shenzhen, People's Republic of China
e-mail: liqq@szu.edu.cn

Q. Mao
State Key Laboratory of Information Engineering in Surveying, Mapping, and Remote Sensing, Wuhan University, Wuhan, People's Republic of China
e-mail: qzhmao@whu.edu.cn

## 1 Introduction

When we take a picture, it may contain certain structures or objects that are undesirable to us. For example, when we take a photo of a beautiful outdoor scene or a historic building, there may be some fence or fence-like structures located between the camera and the scene or the building to be photographed. Such undesirable structures may affect the esthetics feeling of the picture and in the computer vision society, many image editing or inpainting techniques have been recently developed to remove such undesired structures and fill the holes left behind in a visually plausible way. However, many existing inpainting techniques require human interactions to label the structures to be removed. Such labels may take different forms, including strokes, structural boundaries, specified colors, etc. This human labeling step is usually labor intensive (especially in processing a large collection of images) and may require experience and skills.

In practice, such undesirable structures can usually be separated from the remaining scene by examining their distance to the camera, i.e., the depth information. Recently, Microsoft released Kinect [37] that can provide both RGB color information and depth information, with a price comparable to a widely used RGB camera. While the current Kinect hardware still has limitations such as lower resolution in RGB channels

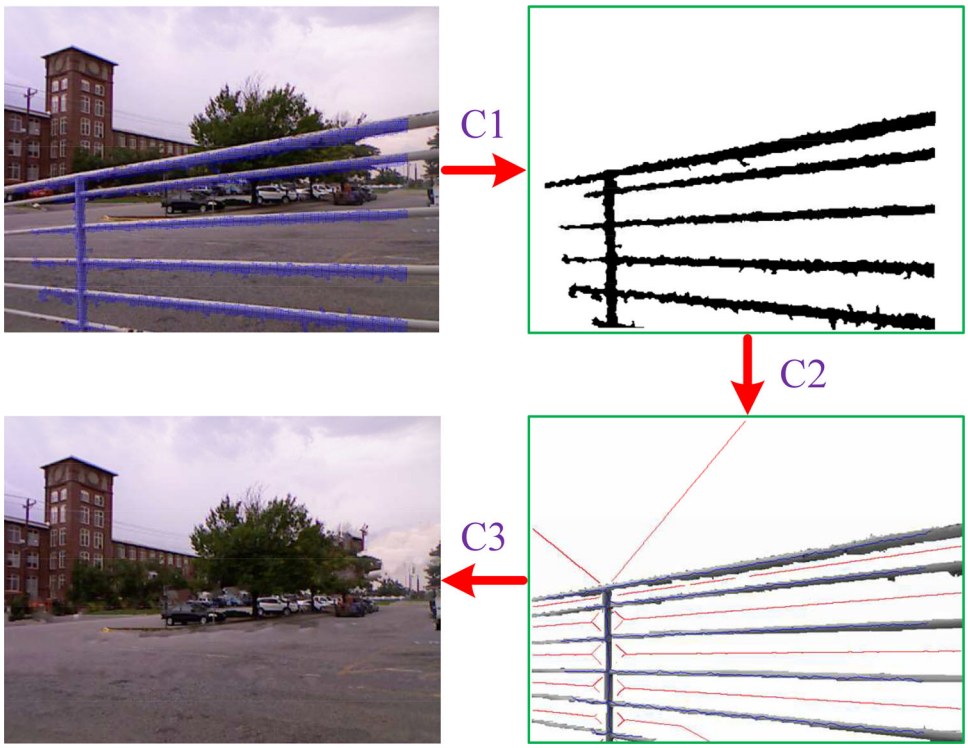



**Fig. 1** The diagram of the proposed inpainting technique. *C1* identifying candidate regions for the fence-like structures using the depth information, *C2* constructing the foreground and background strokes and then performing a foreground–background segmentation, and *C3* completing inpainting by filling the holes after removing the foreground segments

than the classical RGB cameras, dependence on AC power, and limited measurable depth range, it shows us a prospect whereby collecting both RGB and depth information in high resolution would become a default and basic function of cameras in future. Clearly, this prospect will reshape the current computer vision research by incorporating the additional depth information. For example, Kinect has recently been successfully used to address the challenging person-tracking and activity-recognition problems. In this paper, we investigate automatic inpainting on Kinect images by considering both RGB color information and depth information.

In this paper, we focus on automatic inpainting to remove the fence-like structures located between the camera and the desirable scene/structures. As shown in Fig. 1, the proposed techniques consist of the following components: (C1) identifying the candidate regions for the undesired fence-like structures by examining the depth information, (C2) automatically constructing a set of foreground and background strokes and applying graph cut to segment the fence-like structures from the remaining scene, and (C3) performing a constrained exemplar-based inpainting to fill the holes after removing the fence-like structures segmented in (C2).

While it sounds trivial to segment the undesirable structures located between the camera and the remaining scene by only examining the depth information provided by Kinect, it is not the case in practice. We found there are at least three complications that may make this segmentation difficult. First, given the limited resolution of the depth sensors used in Kinect, the depth of very small or thin structures in the image may not be well captured, as shown by an example in Fig. 2a. Second, the depth sensor is usually very sensitive to high reflections. For the structures with highly reflective surfaces, e.g., mirrors or polished surfaces, the depth information provided by Kinect is inaccurate or even lost (setting as the maximum value) when the camera direction is coincident with the reflected direction of strong sunlight, as shown by an example in Fig. 2b. Third, the depth sensor in Kinect has very limited valid range. In practice, we found that the depth information provided by Kinect is usually accurate only in the range from 1.2 to 3.5 m, as shown by an example in Fig. 2c. In this paper, we only use the depth information from Kinect to automatically identify a set of confident foreground and background strokes, which are then used to segment the RGB image for more accurate segmentation of the undesired fence-like structures. Related to our work are the depth-guided inpainting methods developed in [21,44], where depth information is obtained from disparity maps. However, human interactions are needed to define the foreground occluder in these methods, while in this paper we extract the foreground fence-like structures automatically.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents a triangulation-based method to identify the candidate regions for the fence-like structures by using the depth information. Section 4 describes the algorithm for automatically constructing the foreground strokes and background strokes

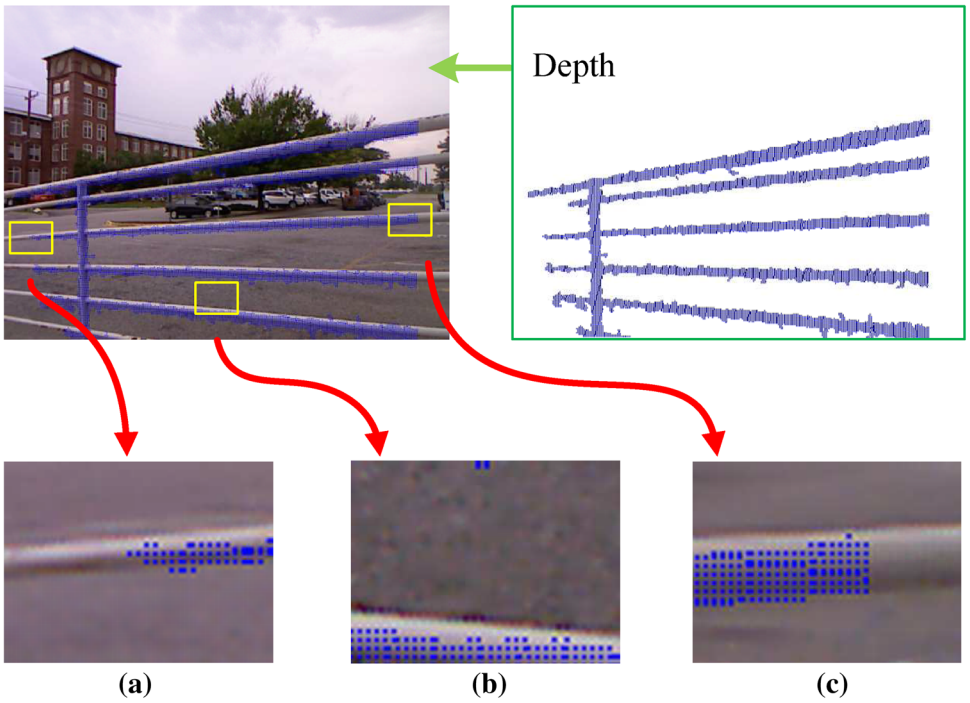



**Fig. 2** Three complications in using the depth information provided by Kinect. The top row shows an RGBD image, in which the *blue dots* denote the points with valid depth information. The *bottom row* shows that **a** the depth of a very thin substructure cannot be captured, **b** the depth of a substructure with high reflections cannot be captured, and **c** the depth of a substructure that is too close to the camera (<1.2 m) cannot be captured

and the segmentation of the undesired fence-like structures. Section 5 introduces the adapted exemplar-based method for the filling-in step in image inpainting. Section 6 reports our experimental results on a set of Kinect images and Sect. 7 concludes our work.

## 2 Related work

Typically, an image inpainting consists of two steps: segmentation of the undesired structure and filling in of the holes after removing the structures. In the previous work, the former is usually achieved by interactive segmentation and the latter is usually achieved by an inpainting algorithm. In this section, we briefly overview the related work on these two steps.

*Interactive segmentation* While the general purpose, automatic RGB image segmentation is still an unsolved problem, interactive segmentation is usually used for the step of segmentation in image inpainting. Many different interactive methods have been developed for foreground–background segmentation, where the foreground indicates the undesired structures to be removed. For example, in [6,19], users draw strokes for foreground and background to segment the foreground. In [36], users drag a box around the structure to be segmented. In [20,29], users label scribbles or trimaps for interactive segmentation. One interesting work for automatic segmentation is the Flash Cut [39], where flash and no-flash image pairs are collected for inferring the depth information and foreground–background segmentation. In this paper, as mentioned in Sect. 1, we attempt to automatically identify the strokes and then use graph cut to achieve a fully automatic step of segmentation for image inpainting.

*Inpainting algorithms* Image inpainting is a useful tool for image editing [2,10] and has a similar objective as image completion [15,40,47] and image restoration [32]. For filling in the holes after removing the undesired structures, there are typically three categories of methods [7,25]. The first is based on statistical models or correspondence maps [1,13,32]. These methods are widely used for texture synthesis. Therefore, if the background around the hole shows highly regular texture and the hole is expected to be filled with the same texture, these methods can produce satisfactory results. The second is based on diffusion with partial differential equations (PDEs) [3–5,8,9,42,43]. The diffusion process smoothly propagates information from the boundary of the hole toward the center of the hole. As PDE-based methods implicitly assume that the expected content of the hole is smooth and nontextured, they often introduce blurring artifacts when they are used to fill in large and textured regions. The third is based on the principle of self-similarity. These methods try to fill in the removed foreground simply by iteratively searching for and copying similar image patches from the background. In [11,12], a strategy considering gradient and confidence was proposed to determine the order of the filling in. In [47], exemplar-based method was extended for video completion. In [24,25], a method for image completion, also for image inpainting, was proposed, where priority scheduling and

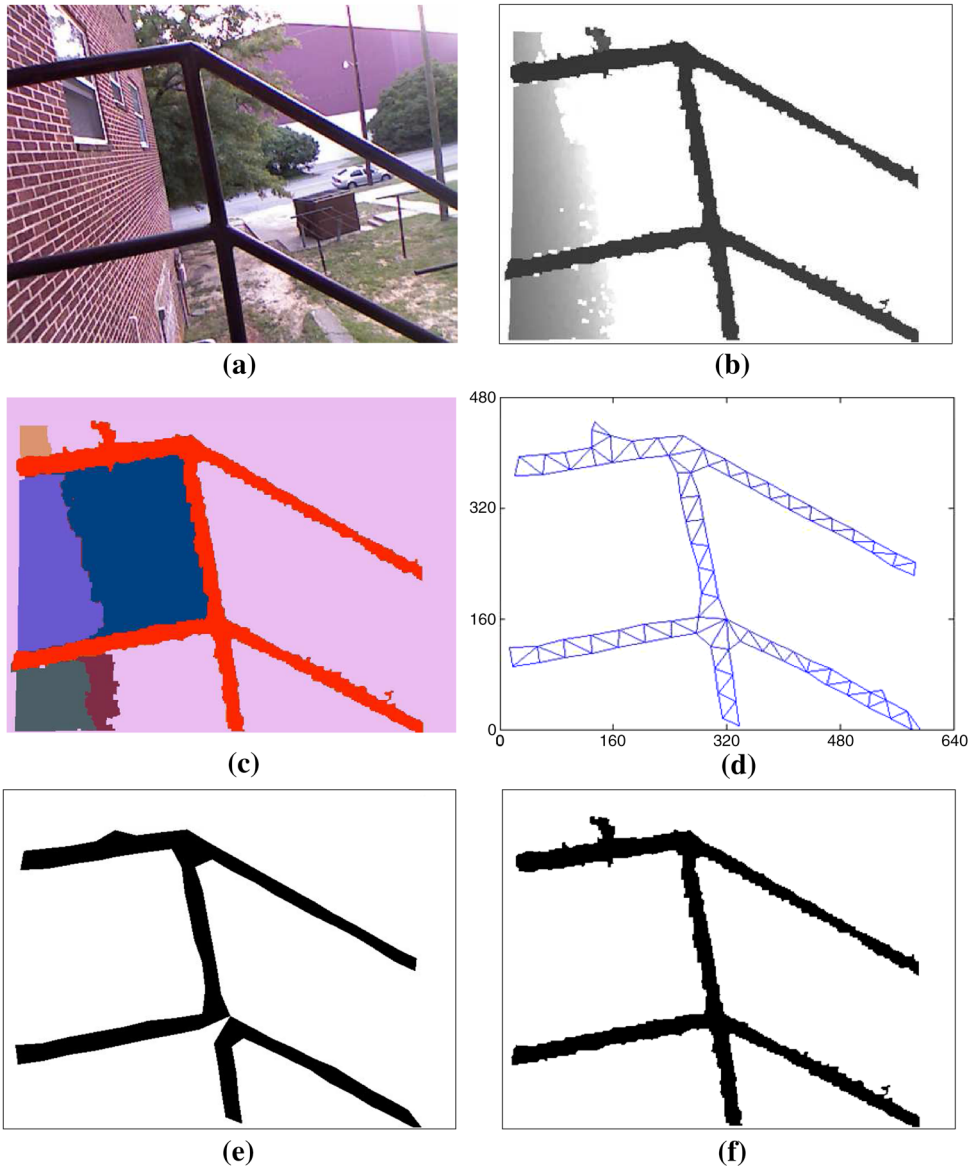



**Fig. 3** An illustration of identifying candidate regions from depth data. **a** An original color image, **b** the corresponding depth map, **c** the clustering result from SRM, **d** the triangulation result on the *red region* in **c**, **e** the extracted elongated branches, and **f** the identified candidate regions

dynamic pruning strategies were used for an efficient belief propagation. Many exemplar-based algorithms are also used for texture synthesis [16,17,26–28,30,46]. However, the matched patches are often searched in a greedy way, which can sometimes lead to visual inconsistencies. To alleviate this problem, some techniques seek human interactions, e.g., in [40], where curves were specified to guide the inpainting of occlusions over certain structures, and in [15], where points of interest were used. In addition, multi-scale analysis such as wavelets [8] and framelet [14] were also used for image inpainting.

Many image inpainting techniques are also studied or used in other image-editing applications, e.g., the image analogies [22], image region merging [33], image summarizing [38], and image authentication [45]. In [35], inpainting was applied to fill in missing parts in the depth maps captured by Kinect, where the color information and depth information were fused in a structure-guided manner.

*Fence removal* Image de-fencing has been studied on RGB images. In [31], a three-step de-fencing approach was proposed. At first, the skeleton structure of a potential fence was automatically searched in the form of a deformed lattice. Then, foreground/background layers were separated by using appearance regularity. At last, the occluded foreground was filled with inpainting algorithms. This work was improved in [34], where in the filling-in step information from multiple views were used and the candidate patches could be found by simulating bilateral symmetry patterns from the source image. The above methods only use the color information and mainly handle lattice-like fences with repeating patterns. In [48,49], occluders such as fences are removed

as foreground objects from multi-focus images. However, these methods require the collection of multiple images with flashlight on and off, and with different focal lengths, while keeping the camera static.

In this paper, we handle the fence-like structures which may not bear highly regular and repeated patterns as in [31, 34], e.g., the fences shown in Fig. 14 (row 1, column 1-4). Furthermore, we use depth information to help obtain more accurate foreground/background segmentation. Finally, we adapt an exemplar-based method [12] for the filling-in step. By searching for similar patches, this method operates on the patch level other than on the pixel level and therefore shows fewer blurring artifacts. In this paper, we further adapt this method by introducing a constrained filling-in strategy to reduce wrong exemplars.

## 3 Identifying candidate regions

In this section, we examine the depth channel of a Kinect image to identify *candidate regions* that are likely to be fence-like structures. These candidate regions may contain many errors (both false positives and false negatives) because of the complications as discussed in Sect. 1. Note the fence-like structures to be removed may not be the closest structure to the camera. In addition, we do not assume that all the points on the fence-like structures in an image have the same depth. In the following, we introduce an image triangulation-based algorithm to identify the candidate regions.

First, we normalize the Kinect depth map in terms of the maximal valid depth value, without considering those pixels with a measured depth that is out of the range of Kinect, as shown in Fig. 3b. Second, since the depth along a fence-like structure changes continuously from one side to another, we cluster the pixels into different regions in terms of their (normalized) depth, using statistical region merging (SRM) [33], as shown in Fig. 3c.

Since a fence-like structure is commonly composed of a number of elongated parts, we take the regions which have an elongated shape as candidate regions. Specifically, for each region after the clustering, we extract its boundary and triangulate the boundary using the algorithm proposed in [18]. By connecting the centers of the neighboring triangles, we can get a shape tree that describes the skeleton of this region. As in [41], we then check the elongation $E$ of each tree branch $t$ using

$$E(t) = L(t) \cdot N(t)/S(t), \tag{1}$$

where $L(t)$, $N(t)$ and $S(t)$ denote the length of, the number of triangles in, and the total area of the triangles in $t$, respectively. If $E(t)$ is larger than a given threshold $T_e$, we take its corresponding region as a candidate region, as shown in Fig. 3f.

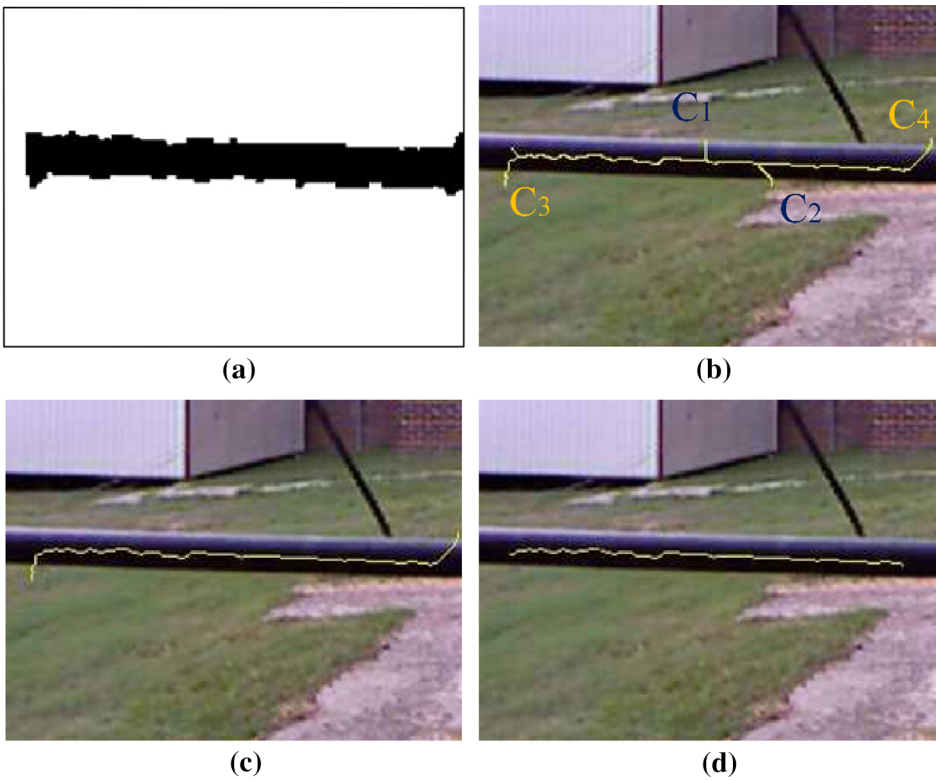



**Fig. 4** An example of foreground stroke construction by edge thinning and pruning. **a** A cropped depth map (corresponding to the image in Fig. 17, row 1, column 5), **b** edge thinning result overlaid on the (cropped) RGB image, **c** result after the branch pruning, **d** result after the end pruning

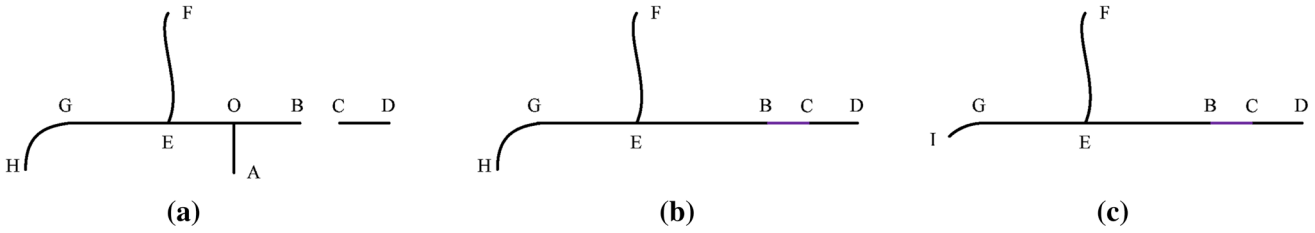



**Fig. 5** An illustration of the pruning algorithm. **a** An example of the thinning result, **b** result after performing the branch pruning on **a**, and **c** result after further performing the end pruning on **b**

## 4 Automatic construction of strokes for segmentation

From Fig. 3f, we can see that the candidate regions contain many errors when compared with the ground-truth fence-like structures. In this section, we develop a new approach to automatically construct a set of foreground strokes (on the fence-like structures) and background strokes (on the remaining scene), with both as constraints, and we then apply the widely used graph-cut algorithm [6] to segment the entire fence-like structures from the scene.

### 4.1 Foreground strokes

As in the human-interaction segmentation, the selected foreground strokes should satisfy two conditions: be fully located inside the foreground and well distributed within the foreground. The proposed foreground stroke construction consists of two steps: edge thinning and edge pruning.

*Edge thinning* While not all the pixels in the candidate regions identified in Sect. 3 lie in the foreground, as shown in Fig. 3f, it is usually safe to say that the medial axes of these elongated candidate regions are surely part of the real foreground. Therefore, we first apply an edge-thinning operation to get the medial axes (skeleton) of the candidate regions. In particular, we use the thinning algorithm proposed in [23]. An example is shown in Fig. 4b, from which we can see that there are still short branches along these medial axes that are not fully located inside the foreground and in the following we design an edge pruning step to remove such branches.

*Edge pruning* From Fig. 4b we can see that there are two kinds of false edges that we need to remove before constructing the foreground strokes. One is the false short edge fragment incident from a "Y" junction, such as edge fragments $C_1$ and $C_2$ in Fig. 4b. The other is the false short edge fragment at the end of the true medial axis of the fence-like structure, such as edge fragments $C_3$ and $C_4$ in Fig. 4b. In this paper, we sequentially perform a "branch pruning" and an "end pruning" operation on the thinning results to remove these two kinds of undesired edge fragments, respectively.

In the "*branch pruning*" operation, we examine every edge fragment by disconnecting the thinning result at junctions and endpoints, as shown in Fig. 5. The general idea is to prune

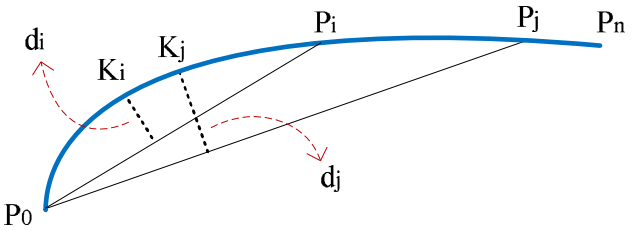



**Fig. 6** An illustration of the end pruning

short branches incident from a "Y" junction, e.g., fragment $OA$ in Fig. 5a, and keep long branches, e.g., fragment $EF$ in Fig. 5a. However, for certain short fragments, e.g., $CD$ in Fig. 5a, we may want to keep them because they are well aligned with the other fragment, e.g., $OB$ in Fig. 5a to form a longer fragment by filling a short gap BC. Such short gaps may be caused by the depth information complications discussed above and needs to be filled.

To achieve this goal, we first construct a forest (a set of disjoint graphs) from the thinning result by treating each pixel on the thinning result as a node and connecting each pair of nodes by a graph edge if the Euclidean distance between the corresponding pixels is less than a pre-given threshold $L_e$. We further define the edge weight using the corresponding Euclidean distance. We then compute a minimum spanning tree (MST) for each graph in this forest.

On these MSTs, we conduct branch pruning by recursively identifying the longest path (the path with the largest total edge weights between leaves within the same MST) as the candidate foreground medial axis, as summarized in Algorithm 1. Note that in this algorithm, we use the edge-length threshold $L_e$ to connect the short gaps, e.g., BC in Fig. 5a and a path-length threshold $L_p$ to remove false short branches. For example, if we set $L_e = |\mathrm{BC}| + \epsilon$, $L_p = |\mathrm{OA}| + \epsilon$ with $\epsilon$ being a very small positive number, in Fig. 5a, we can get the paths HGEBCD and EF by using Algorithm 1. As a result, the branch pruning would produce a result as shown in Fig. 5b.

In the "*end pruning*" operation, we attempt to remove the edge fragments at the end of the edges in the thinning result that may be deviated from foreground into the background, as shown by the edge fragments $C_3$ and $C_4$ in Fig. 4b. Such deviated fragments usually show poor continuity (i.e., higher

**Algorithm 1** Branch-pruning algorithm

```
1: procedure BRANCHPRUNING
2:     input: T: an MST
3:            L_p: path-length threshold
4:     output: R: result edges
5:     // get all leaf nodes from V
6:     V_leaf ← GetLeafnodes(V);
7:     // search a longest path P_max between tree leaves
8:     W_max ← 0;
9:     for each v_i ∈ V_leaf do
10:        P ← LongestPathFrom(v_i);
11:        W ← PathLengthOf(P);
12:        if W > W_max then W_max ← W, P_max ← P;
13:        end if
14:    end for
15:    if W_max < L_p then go to Line 20;
16:    end if
17:    R ← Add P_max to R;
18:    // recursive pruning
19:    T ← (T − P_max), go to Line 6;
20: end procedure
```

**Algorithm 2** End-pruning algorithm

```
1: procedure ENDPRUNING
2:     input: C: thinning edges, L_d: deviation threshold
3:            L_τ: end-length threshold
4:     output: R: result edges
5:     // Push all the endpoints of C into E
6:     E ← GetEndPoints(C);
7:     // Remove all the junction points of C
8:     C' ← RemoveJunctions(C);
9:     for each edge fragment C'_j in C', C'_j ⋂ E≠∅ do
10:        // Push points on C'_i into a point array P successively,
11:        P_0 ← C'_j ⋂ E,
12:        P ← GetPointsFrom(C'_j, P_0);
13:        // Get the first high-curvature point K
14:        IsHighCvt = false;
15:        for i = 1 to n do
16:            (d_i, K_i) ← GetDeviationOf(P_0P_i);
17:            if d_i > L_d then IsHighCvt=true, K=K_i, break;
18:            end if
19:        end for
20:        if (IsHighCvt==true && Length(P_0K)<L_τ) then
21:            C'_j ← C'_j − P_0K;
22:        end if
23:    end for
24:    R ← C' + JunctionPointsOf(C);
25: end procedure
```

curvature) compared to their connected true edge fragments, as illustrated by the edge $GH$ in Fig. 5a.

Let $P = \{P_i | i = 0, ..., n\}$ be a connected edge fragment after the branch pruning. As shown in Fig. 6, we define $d_i$ to be the deviation of the edge $P_0P_i$ by

$$d_i = \max_{0 \le k \le i} |P_k, P_0 \rightarrow P_i|, \tag{2}$$

where $|P_k, P_0 \rightarrow P_i|$ indicates the Euclidian distance between the point $P_k$ to the line that passes through $P_0$ and $P_i$. As summarized in Algorithm 2, we pick the point $P_k$ with large deviation $d_k$ and then remove the edge fragment $P_0P_k$. Such end-pruning operations are conducted at each possible end of the edges in the branch-pruning result. We simply take the thinning results after both branch pruning and end pruning as the foreground strokes.

## 4.2 Background strokes

Similarly, there are also two requirements for the background strokes: fully located in the background and well distributed in the background. Because of the depth information complications, the construct foreground strokes may not cover the entire fence-like structure. As shown in Fig. 12d, part of the fence-like structure may be missing in the middle or at the end of the foreground strokes. We need to avoid constructing background strokes at such location. The proposed background-stroke construction consists of three steps: line fitting, watersheding, and subtraction.

*Line fitting* Most fence-like structures are composed of straight (or very smooth) elongated components. We design a line-fitting algorithm on the foreground strokes to infer the possibly missing components of the fence-like structures. Our line-fitting algorithm consists of the following steps.

i) Decompose the foreground strokes into a set of edge fragments by separating at "Y" junctions" and high-curve points, as illustrated in Fig. 7b.

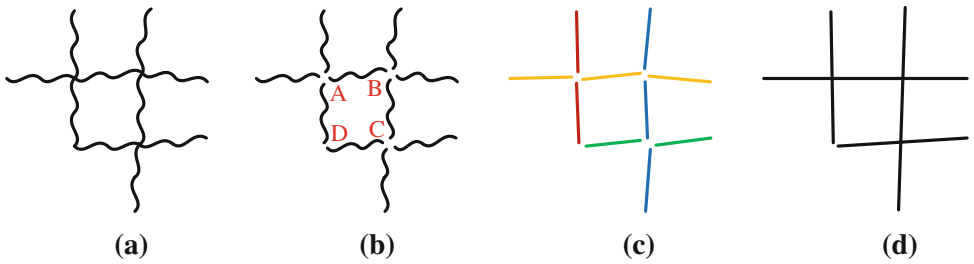



**Fig. 7** An illustration of line fitting. **a** The constructed foreground strokes, **b** edge fragments by separating at "Y" junctions (e.g., *A*, *B*, and *C*) and high-curvature points (e.g., *D*), **c** line-fitting results on individual edge fragments and their grouping results (one color indicates one group), and **d** line-fitting results for each group

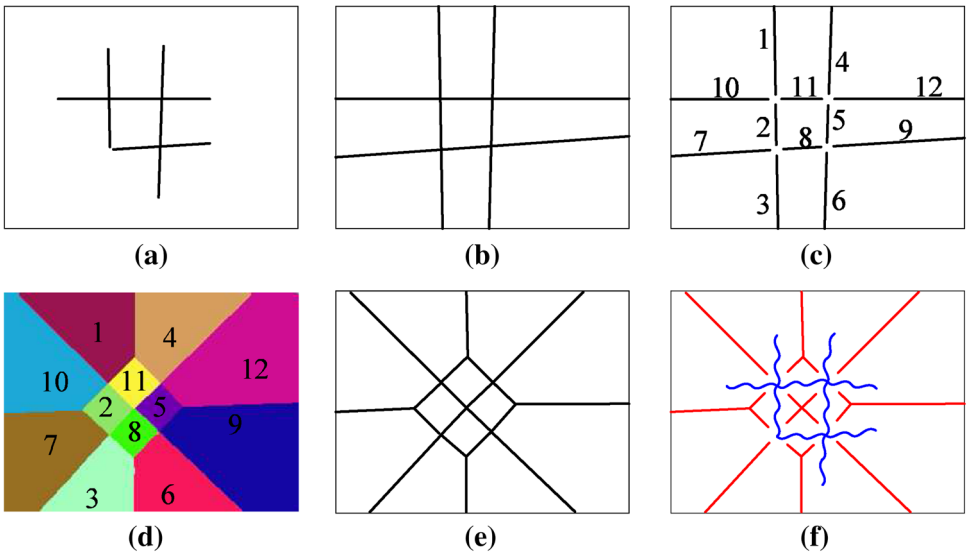



**Fig. 8** An illustration of watersheding and subtraction. **a** Line fitting to the foreground strokes (Fig. 7a), where the rectangle indicates the image boundary, **b** result from line extension, **c** line segments after junction-point removal, **d** result from watersheding, **e** the extracted watershed, **f** the background strokes (in *red*) constructed by subtraction, together with the foreground strokes (in *blue*)

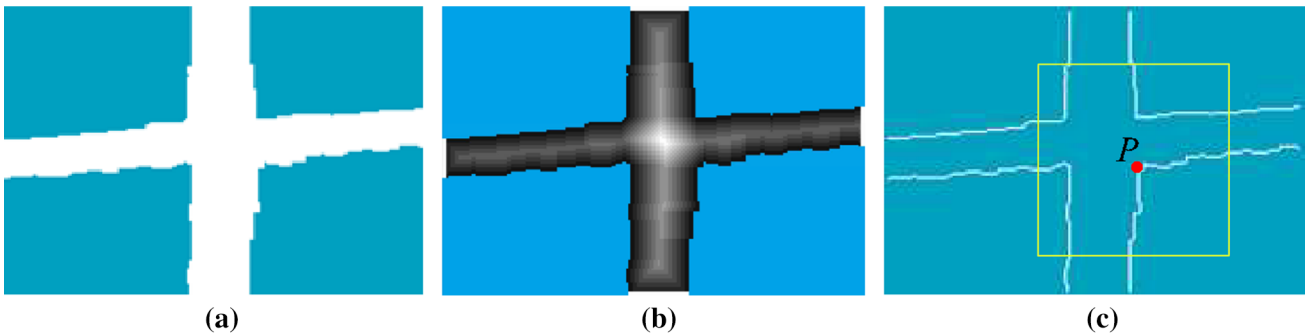



**Fig. 9** Illustration of the new inpainting algorithm. **a** A mask (in write), **b** the geodesic maps, **c** the boundary of the mask, where the *yellow square* denotes the local window centered at point $p$

ii) Perform line fitting to each edge fragment and then group all the edge fragments in terms of the coincidence of their fitted lines, as illustrated in Fig. 7c.
iii) Fit all the edge fragments in each group by a single line, as illustrated in Fig. 7d.

*Watersheding* The pixels with equal distance to two neighboring foreground strokes (i.e., no other foreground strokes in between) have a large probability to fall in the background. In this paper, we identify such pixels by using a watersheding algorithm. First, we extend the above fitted lines to the image boundaries, as illustrated in Fig. 8b. Second, we get rid of all the junction points in the line-extension result and construct a set of line segments, as illustrated in Fig. 8c. Third, we conduct a morphological dilation to these line segments until the dilations from different line segments meet each other and cover the entire image, as shown in Fig. 8d, where the dilations from different line segments are shown in different colors. The pixels where the dilations from different line segments meet each other are identified as watersheds, as shown in Fig. 8e.

*Subtraction* The above watersheds are constructed from the one-pixel-wide foreground strokes. Therefore, the watershed pixels that are very near to the junctions of the foreground strokes may actually be located in the foreground. We employ a subtraction step to further remove such watershed pixels for constructing the final background strokes.

Specifically, let $I_{\mathrm{d}}$ be the binary map that represents the candidate regions identified in the depth map: 1 and 0 indicate the pixels in or out of the candidate regions, respectively. Let $I_{\mathrm{w}}$ be the binary map that represents the watershed: 1 and 0 indicates the watershed pixels and non-watershed pixels, respectively. We construct a binary map $I_{\mathrm{bsk}}$ as the background strokes (1 indicates the background strokes) by

$$I_{\mathrm{bsk}} = I_{\mathrm{w}} - \mathrm{mmDilate}(I_{\mathrm{d}}, s), \tag{3}$$

where $\mathrm{mmDilate}(I_{\mathrm{d}}, s)$ indicates a morphological dilation of $I_{\mathrm{d}}$ by a size $s$. An example of this subtraction step is shown in Fig. 8f. In our experiments, we consistently set $s = 3$ pixels.


**Fig. 10** Inpainting with (*top row*) and without (*bottom*) the geodesic-map and local-window constraints. *Each square* refers to the location of the filled patch in the corresponding iteration

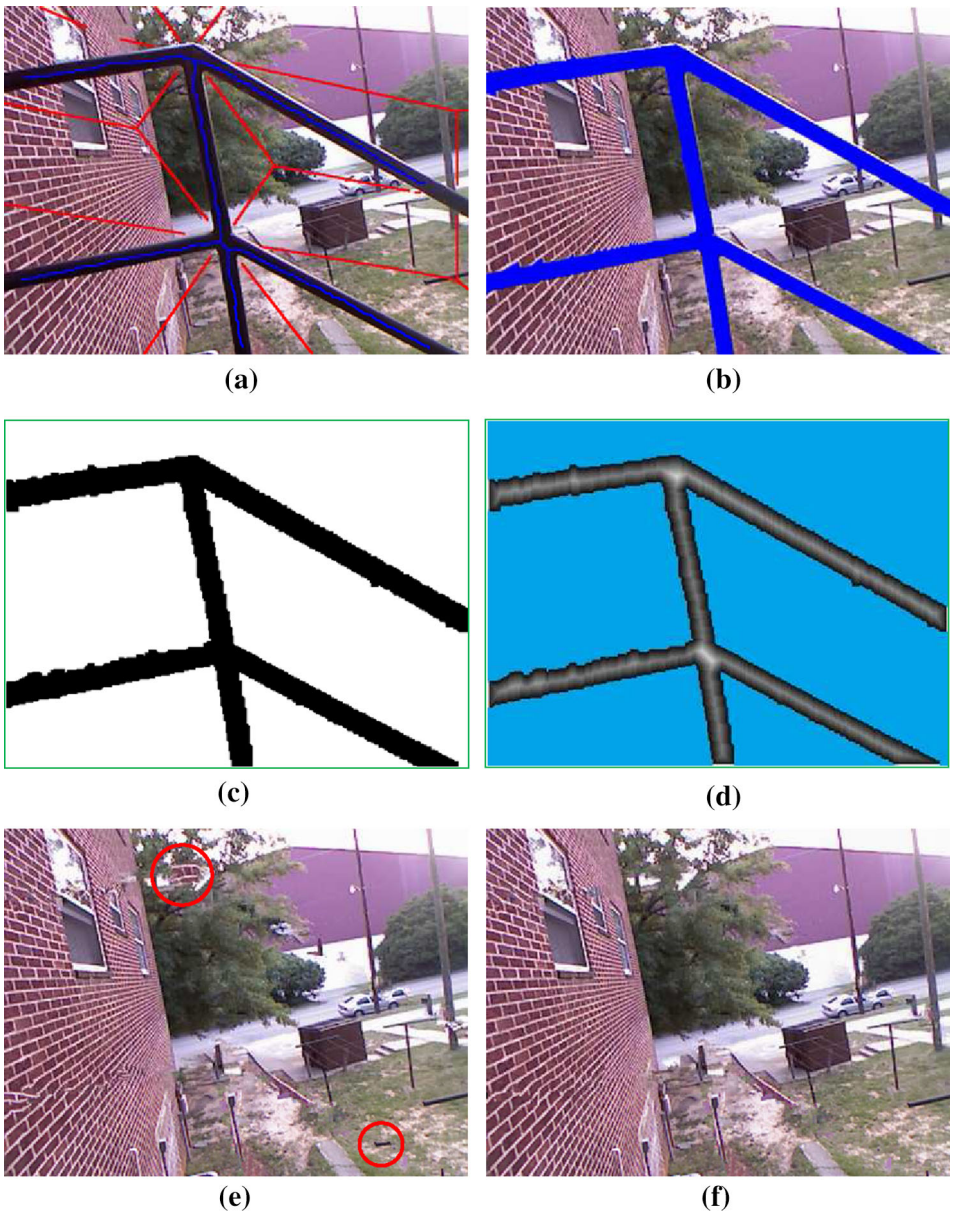



**Fig. 11** An example of the proposed inpainting. **a** An RGB image overlaid with foreground strokes (in *blue*) and background strokes (in *red*), **b** the graph-cut segmentation result (in *blue*), **c** the dilation result on **b**, and **d** the geodesic map generated from **c**, **e** the non-constrained inpainting result, **f** the constrained inpainting result

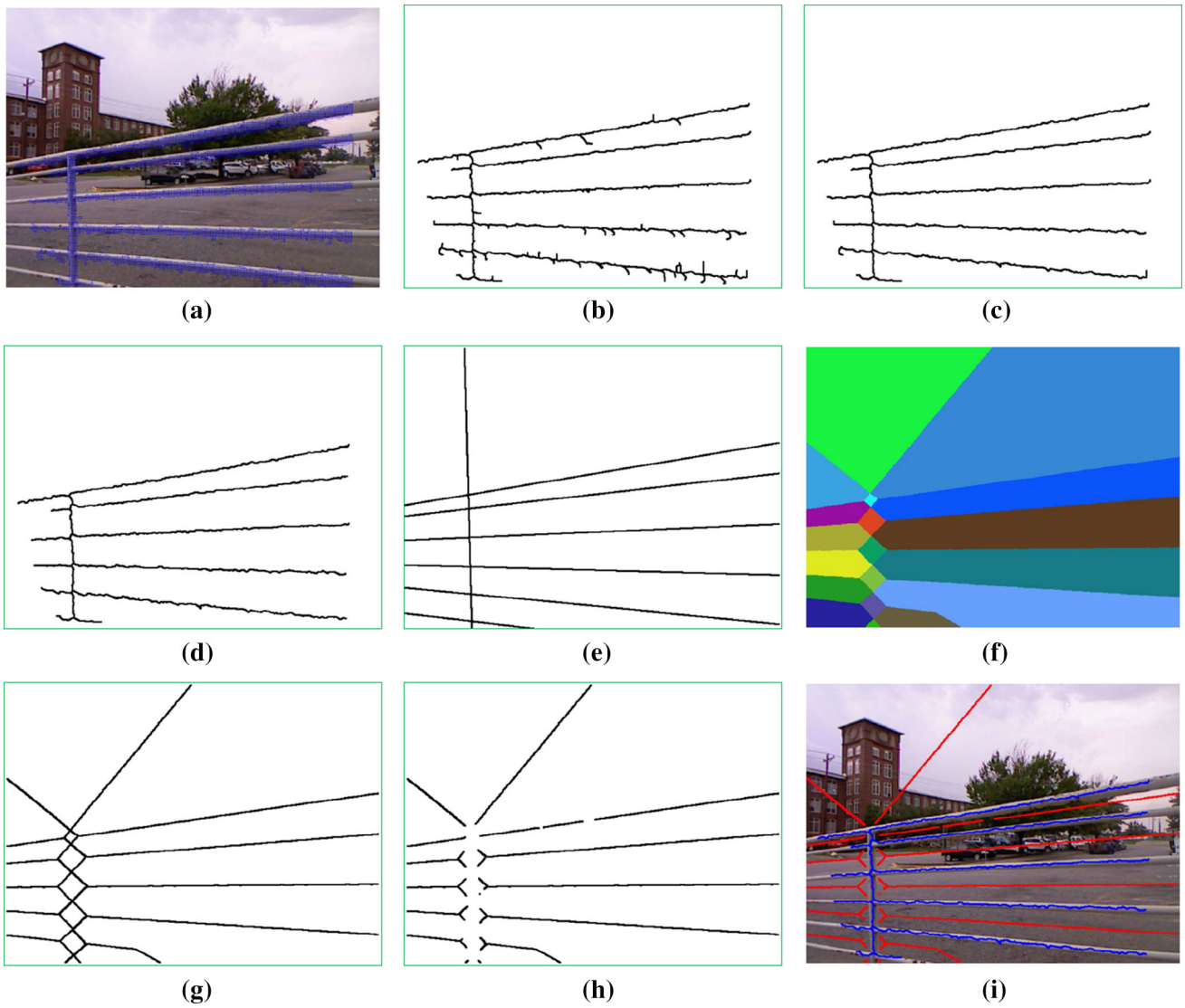



**Fig. 12** A sample result for constructing the foreground and background strokes. **a** A sample RGB image from Kinect, overlaid with depth data (*dots in blue*), **b** the thinning result of the candidate regions, **c** the branch-pruning result on **b**, **d** the end-pruning result on **c**, **e** the line-fitting and line-extension result on **d**, **f** the watersheding result on **e**, **g** the watershed extracted on **f**, **h** the subtraction result on **g**, **i** the constructed foreground (in *blue*) and background (in *red*) strokes

### 4.3 Graph-cut based segmentation

Once we construct the foreground and background strokes, we use the graph-cut method proposed in [6] to segment the fence-like structures. An example is shown in Fig. 11a, b.

## 5 Constrained filling-in for inpainting

In this section, we adapt the exemplar-based inpainting algorithm [12] to fill in the holes after removing the fence-like structures. We take the hole (after foreground removal) as the initial mask and select local points (and the small patches centered at these points) in the mask for filling in. After the filling in at a local point, the mask is updated and we repeat this process until the entire hole is filled. Generally, the order of the points selected for filling in is important for an exemplar-based inpainting algorithm. Given a patch centered at the point $p$ ($p$ is on the boundary of the updated mask), the algorithm proposed in [12] computes the priority $P$ of $p$ by

$$P(p) = C(p) \cdot D(p), \tag{4}$$

where $C(p)$ is the confidence term, defined to be the portion of background pixels in the $p$-centered patch, and $D(p)$ is the

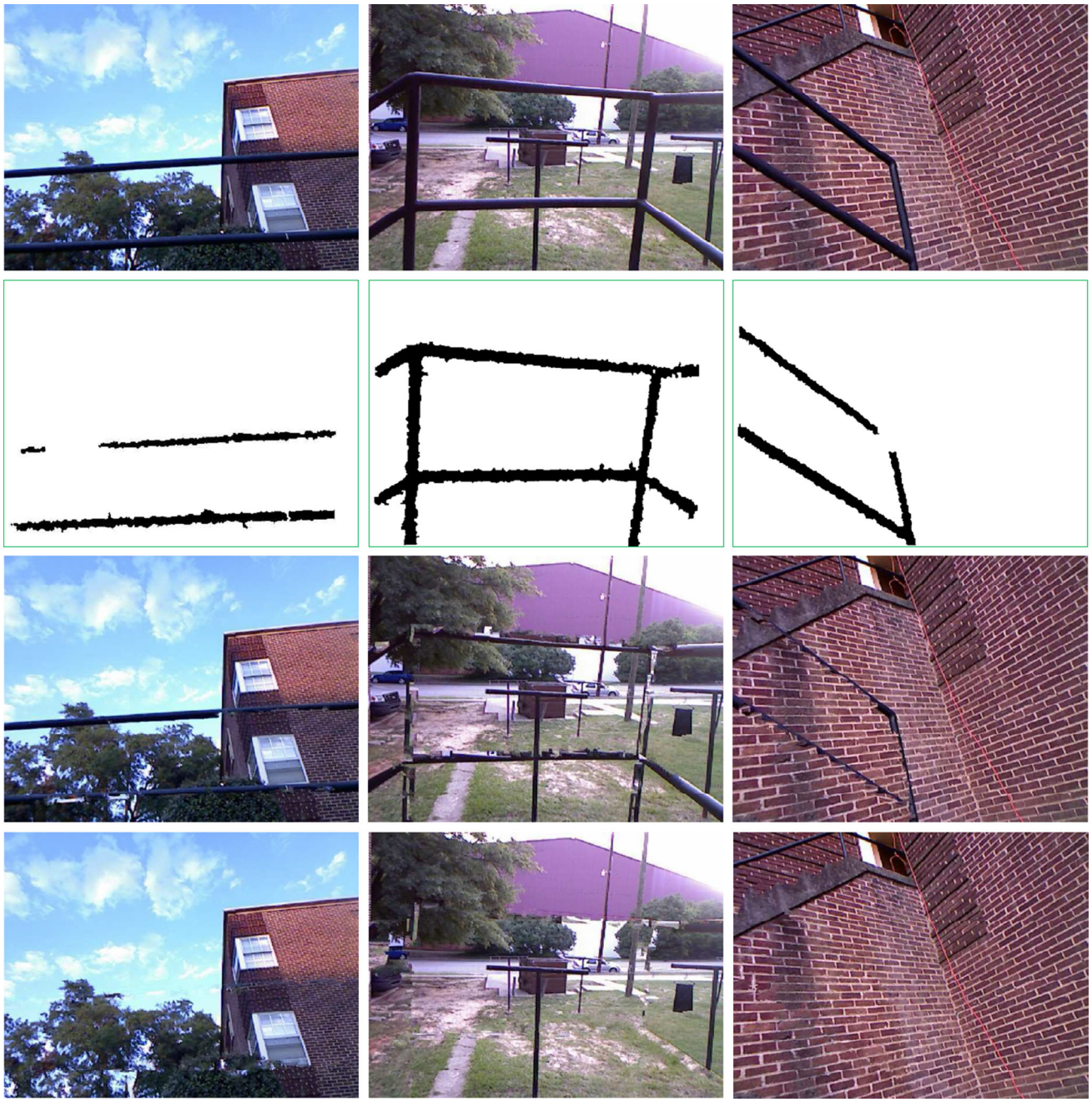


**Fig. 13** Inpainting results by removing different foregrounds. *Row 1* input RGB images. *Row 2* candidate regions. *Row 3* inpainting results by directly using the candidate regions as the foreground. *Row 4* inpainting results by using the proposed segmentation algorithm to extract the foreground

data term, defined to be the projection of 90°-rotated gradient at $p$ to the normal direction of the mask boundary. Iteratively searching a pixel with maximum $P$ on the boundary of the updated mask and filling with the matched patch, the algorithm in [12] can well handle the structures with continuous high gradient.

However, the drawback of this algorithm is that, once a wrong exemplar is taken, more wrong exemplars may be searched and used to fill in the nearby unfilled area. An example is shown in row 2 of Fig. 10. Specifically, since the removed fence-like structures often consist of a number of elongated, sparsely distributed parts, as shown in Fig. 11a, the original exemplar-based algorithm [12] may easily get wrong exemplars when searching a matched patch in the whole image. To address this problem, we adapt the exemplar-based inpainting algorithm by using a constrained filling-in strat-

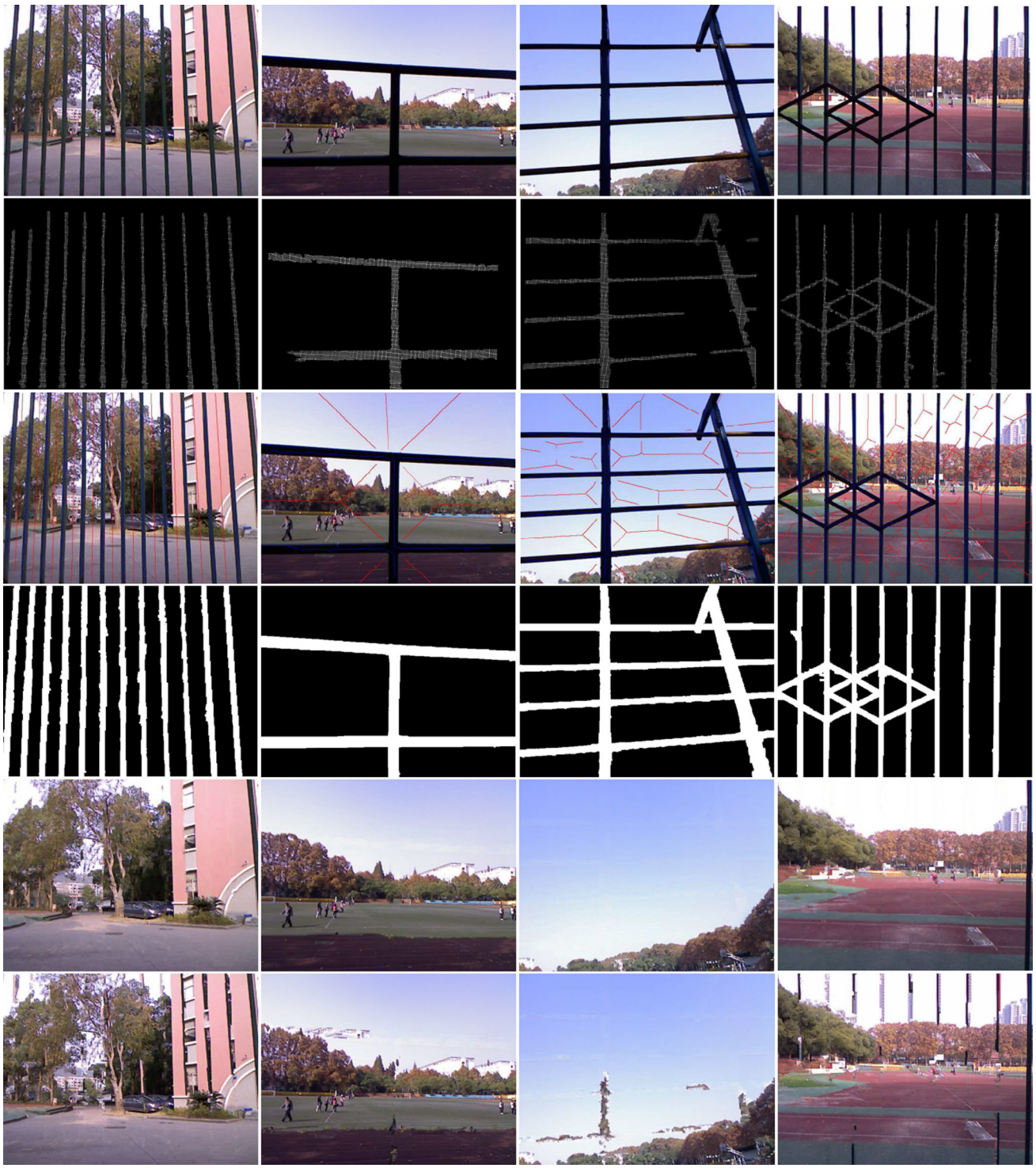


**Fig. 14** Inpainting results on four sample images. *Row 1* input RGB images. *Row 2* candidate regions gained from depth maps. *Row 3* the constructed foreground strokes (in *blue*) and background strokes (in *red*). *Row 4* the segmented fence masks. *Row 5* inpainting results from the constrained inpainting algorithm. *Row 6* inpainting results from the non-constrained inpainting algorithm

egy. More specifically, we incorporate the following two constraints.

*Geodesic-map constraint* We partition the mask into a set of geodesic levels by using a recursive erosion algorithm. In each round of erosion, the removed pixels are assigned with a geodesic value higher than that in the previous round. Therefore, pixels near the mask boundary have lower geodesic values, and pixels near the mask center have higher geodesic

values. An example is shown in Fig. 9b, where brighter pixels have higher geodesic values.

Given $B$ to be the boundary of the current mask and $G_{min}$ to be the minimum geodesic value on $B$, we select the point $p_0$ (i.e., patch center) to be processed by

$$\{p_0\} = \arg\max_{\{p_i\}} \{P(p_i) | p_i \in B, G(p_i) = G_{min}\} \quad (5)$$

where $P(\cdot)$ calculates the priority by using Eq. (4) and $G(p)$ denotes the geodesic value at pixel $p$. With the geodesic-map constraint $G(p_i) = G_{min}$, the modified algorithm will perform the filling in from the boundary toward the center of the mask.

*Local-window constraint* Instead of searching the matched patch in the whole image, we choose to search in a local window. The local window is a square, centered at the point to be processed, with a side length of $L_c$. An example of the local window is shown in Fig. 9c. Since the fence-like structures have a number of elongated parts, the matched patch is likely to locate close to the fence. Thus, the local-window constraint not only improves the searching speed, but also reduces the likelihood of taking wrong exemplars.

In our algorithm implementation, we create the initial mask by dilating the segmented foreground by a size of 2 pixels to make sure the fence-like structures are fully covered. Figure 10 shows a comparison study with intermediate results, where the source image is cropped from the image in Fig. 14, column 3. In Fig. 10, results in the top row are produced under constraints of the geodesic map and the local window (with a side length of 100 pixels). In the bottom row, wrong exemplars are searched at the high-gradient area without incorporating geodesic-

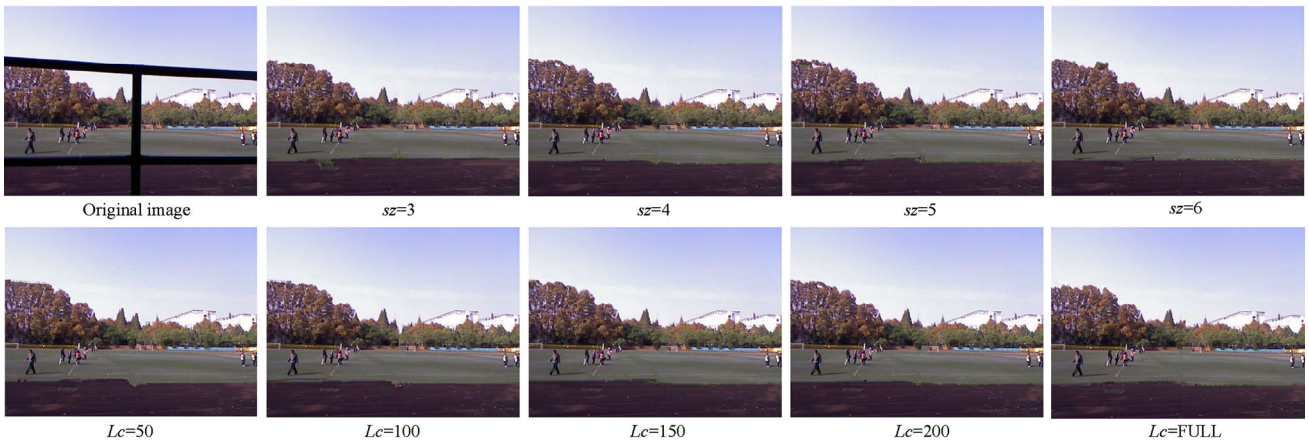



**Fig. 15** Inpainting results at different $sz$ and $L_c$. *Row 1* results at different $sz$ when setting $L_c = 100$. *Row 2* results at different $L_c$ when setting $sz = 4$. This test image is taken from Fig. 14, row 1, column 2

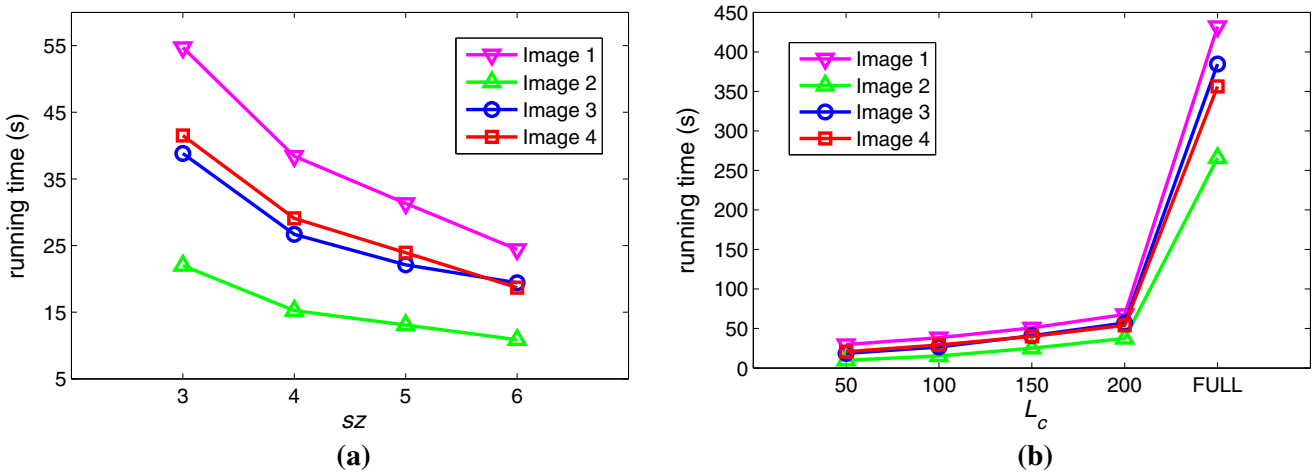



**Fig. 16** Running time at different $L_c$ and $sz$ for four images in Fig. 14. **a** Running time at different $sz$ when setting $L_c = 100$, **b** running time at different $L_c$ when setting $sz = 4$

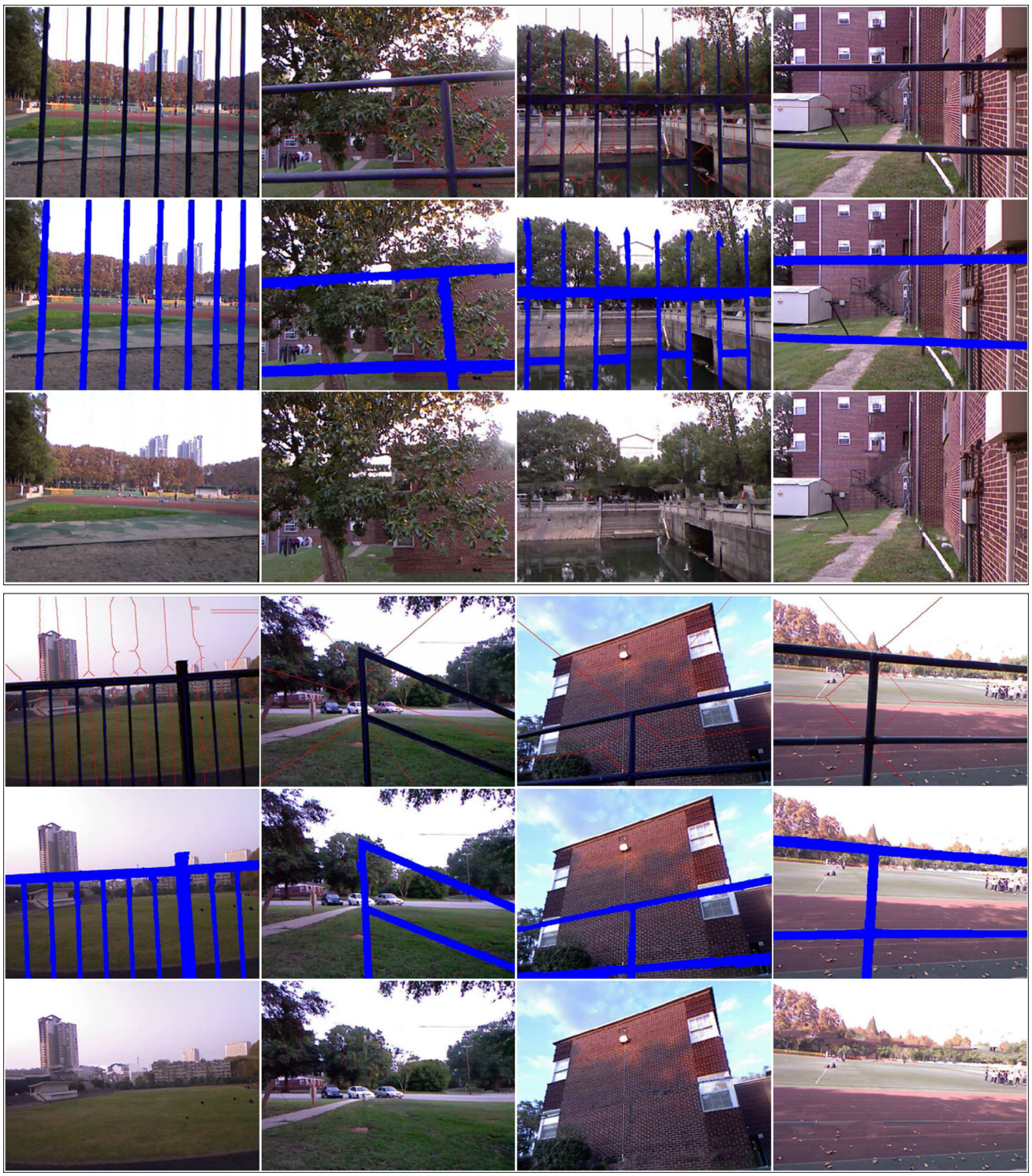


**Fig. 17** Inpainting results on eight more sample images. *Rows 1 and 4* the original RGB images and the constructed strokes. *Rows 2 and 5* the masks produced with graph cut. *Rows 3 and 6* the inpainting results using the proposed algorithm

map and local-window constraints. Clearly, the constrained inpainting performs the filling ins from the boundary to the center of the fence, and produces more visually plausible results. Figure 11 shows the sample inpainting results using the non-constrained (Fig. 11e) and the above constrained exemplar-based algorithms (Fig. 11f), respectively. Two sets of wrong exemplars are indicated in Fig. 11e using red circles.

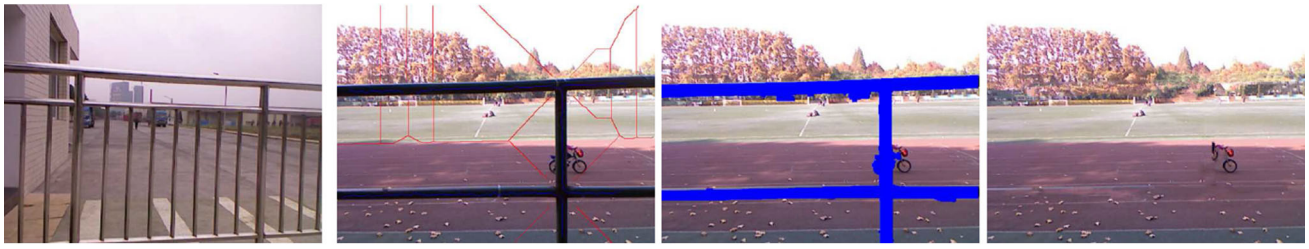



**Fig. 18** Two examples for failed cases. **a** A Kinect image where no valid depth response is produced in the region corresponding to the polished metal surface. **b** A Kinect image with the derived foreground strokes (*blue*) and background strokes (*red*). **c** The segmented foreground from **b**. **d** The inpainting result for the image in **b**

## 6 Experiments and results

To validate the proposed approach, we collected a set of 120 Kinect images[1] with fence-like structures. In these images, the fence structures typically have a distance of around 1.0 m to 2.5 m to the camera. Note that the fence may not be parallel to the Kinect image plane and, therefore, the distance between the fence and image plane may not be a constant value in a same Kinect image, as shown in Fig. 12a. Furthermore, the current Kinect sensors are sensitive to sunlight—using Kinect directly in strong sunlight may incur some invalid depth responses. Thus, in our data collection, we avoided the use of Kinect directly in strong sunlight. The software for data collection is developed based on the Kinect for Windows SDK[2]. In our dataset, the size of the color image is 640 × 480, and the original size of the corresponding depth map is 320 × 240. The alignment of the depth map with the color image is achieved using functions in the SDK which convert depth coordinates to the color space coordinates. After alignment, the depth image has the same size of color image, i.e., 640 × 480. In our experiments, we consistently set $T_e = 5$ for identifying the elongated branches in the shape tree, $L_e = 10$ pixels, and $L_p = 50$ pixels for the branch-pruning algorithm, $L_d = L_\tau = 10$ pixels for the end-pruning algorithm. We implemented the proposed algorithm in C++, and ran it on a 64-bit Windows 7 computer, with one core of a 3.3 GHz CPU and 8 G RAM.

Figure 12 demonstrates the construction of the foreground and background strokes by showing the step-by-step results on a sample Kinect image. Figure 13 shows the necessity of the proposed stroke construction and segmentation step. We can see that by directly removing the candidate regions derived from the depth map and filling in the holes left behind, the inpainting results are poor. The major reason is that the foreground directly using the candidate regions does not cover the whole fence and parts of the fence are missed. In this way, the inpainting process could not handle the uncovered region of the fence—once a wrong exemplar occurs, the exemplar-based inpainting algorithm would propagate more wrong exemplars to the neighboring regions and thus lead to poor result.

Figure 14 shows the stroke construction results and the inpainting results on four Kinect images in our dataset. In Fig. 14, from row 2, we can see that the candidate regions obtained from the depth maps cannot cover the whole fence in the RGB images. Based on the candidate regions in row 2, the proposed approach can construct reasonable foreground and background strokes, as shown in row 3. Based on these strokes, the segmented foregrounds well cover the whole fence-like structures, as shown in row 4. Comparing row 5 and row 6 in Fig. 14, we can see that the adapted constrained exemplar-based algorithm can produce more visually plausible results, while the original exemplar-based algorithm produces many wrong exemplars when using the same masks.

We also conduct a series of experiments to determine the proper parameters for the constrained inpainting algorithm, namely the radius of the patch $sz$[3], and the side length of the local window $L_c$. We find that $sz = 4$ and $L_c = 100$ can handle all the images in the dataset. An example is shown in Fig. 15 by applying different $sz$ and $L_c$ on a Kinect image. We can see from Fig. 15 that, when with a fixed $L_c = 100$, a small $sz = 3$ may produce some wrong exemplars and a big $sz = 6$ may result in sharp mosaic edges between exemplars. When setting $sz = 4$, we can see that a small local window with $L_c = 50$ may incur some visual inconsistencies. If we set the local window to be the whole image ($L_c$ = FULL), the proposed constrained inpainting algorithm still produces better results than the original inpainting algorithm [12], as shown in Fig. 15. It indicates that the constraint from the geodesic map is effective in reducing wrong exemplars.

In addition, we analyze the impact of $sz$ and $L_c$ on the efficiency of the constrained inpainting algorithm. We can

[1] https://sites.google.com/site/qinzoucn/documents

[2] http://www.microsoft.com/en-us/kinectforwindowsdev/downloads.aspx

[3] The size of the patch is $(2{\cdot}sz{+}1){\times}(2{\cdot}sz{+}1)$

see from Fig. 16 that the running time is longer for a smaller patch radius $sz$ and a bigger local window $L_c$. Among the four images in Fig. 14, image 1 and image 2 take the longest and shortest running time, respectively. This is mainly because the size of the foreground to be inpainted is different. In image 1, the segmented foreground accounts for 29 % of the image, while in image 2, the segmented foreground is only accounts for 14 % of the image. We consistently set $sz = 4$ and $L_c = 100$ to produce results in Figs. 14 and 17, and the average running time for one image was about 20 s. Note that the filling-in step dominates the running time in the proposed approach.

Figure 17 shows the stroke construction and the inpainting results on eight more images in our dataset. We can see that the proposed approach can automatically construct reasonable foreground and background strokes to segment the undesired fence-like structures and the adapted constrained inpainting algorithm can fill in the holes in a visually plausible way.

There are several situations which would challenge the proposed approach. Figure 18 shows several examples of failures when performing the proposed approach for fence inpainting. In Fig. 18a, the polished metal surface of the fence structure caused mirror reflection and did not produce any valid depth response. In Fig. 18b–d, the person (riding a bicycle) who is partially occluded by the fence cannot be fully recovered using the exemplar-based inpainting algorithm. In the future, we plan to take a sequence of Kinect images that can provide redundant RGBD information to address this issue.

## 7 Conclusion

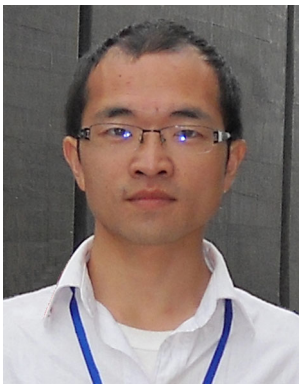
In this paper, we developed a new approach for automatic inpainting by removing fence-like structures in RGBD images provided by Kinect. In this approach, we first identified candidate regions of the fence-like structures using the depth information. Based on these candidate regions, we developed an algorithm to automatically construct a set of foreground and background strokes, which are then fed to the graph-cut algorithm to achieve a more accurate foreground–background segmentation on the RGB image. Finally, we adapted a widely used exemplar-based inpainting algorithm to recursively fill in the hole after removing the foreground fence-like structures. We tested the proposed approach on a set of Kinect images and achieved very promising results.

**Acknowledgments** The authors would like to thank Mr. Shufan Liu and Mr. Liang Zhang for help in collecting the Kinect data. This research was supported, in part, by the China Postdoctoral Science Foundation funded project (2012M521472), National Natural Science Foundation of China (61301277 and 41371431), Hubei Provincial Natural Science Foundation (2013CFB299), National Basic Research Program of China (2012CB725303), and the fund from AFOSR FA9550-11-1-0327, NSF IIS-0951754, NSF IIS-1017199, and ARL W911NF-10-2-0060.

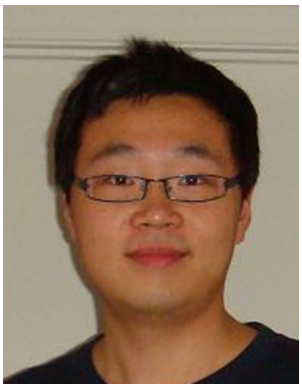
## References

1. Ashikhmin, M.: Synthesizing natural textures. In: ACM Symp. Interactive 3D Graphics (2001)
2. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.: PatchMatch: a randomized correspondence algorithm for structural image editing. In: SIGGRAPH, pp. 1–10 (2009)
3. Bertalmío, M.: Strong-continuation, contrast-invariant inpainting with a third-order optimal pde. IEEE TIP **15**(7), 1934–1938 (2006)
4. Bertalmío, M., Bertozzi, A., Sapiro, G.: Navier–stokes, fluid dynamics, and image and video inpainting. In: CVPR (2001)
5. Bertalmío, M., Sapiro, G., Caselles, V., Ballester, C.: Image inpaintng. In: SIGGRAPH (2000)
6. Boykov, Y.Y., Jolly, M.P.: Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In: ICCV (2001)
7. Bugeau, A., Bertalmío, M., Caselles, V., Sapiro, G.: A comprehensive framework for image inpainting. IEEE TIP **19**(10), 2634–2645 (2010)
8. Cai, J., Chan, R., Shen, Z.: A framelet-based image inpainting algorithm. Appl. Comput. Harmon. Anal. **24**(2), 131–149 (2008)
9. Chan, T.F., Shen, J.: Nontexture inpainting by curvature-driven diffusions. J. Vis. Commun. Image Rep. **12**(4), 436–449 (2001)
10. Cho, T., Butman, M., Avidan, S., Freeman, W.: The patch transform and its applications to image editing. In: CVPR (2008)
11. Criminisi, A., Pérez, P., Toyama, K.: Object removal by exemplar-based inpainting. In: CVPR (2003)
12. Criminisi, A., Pérez, P., Toyama, K.: Region filling and object removal by exemplar-based image inpainting. IEEE TIP **13**(9), 1–13 (2004)
13. Demanet, L., Song, B., Chan, T.: Image inpainting by corresponding dence maps: a deterministic approach. In: UCLA CAM R, Tech. Rep. (2003)
14. Dong, B., Ji, H., Li, J., Shen, Z., Xu, Y.: Wavelet frame based blind image inpainting. Appl. Comput. Harmon. Anal. **32**, 268–279 (2012)
15. Drori, I., Cohen-Or, D., Yeshurun, H.: Fragment-based image completion. In: SIGGRAPH, pp. 303–312 (2003)
16. Efros, A., Freeman, W.: Image quilting for texture synthesis and transfer. In: SIGGRAPH, pp. 341–346 (2001)
17. Efros, A., Leung, T.: Texture synthesis by non-parametric sampling. In: ICCV, pp. 1033–1038 (1999)
18. Felzenszwalb, P.F.: Representation and detection of deformable shapes. IEEE TPAMI **27**(2), 208–220 (2005)
19. Gulshan, V., Rother, C., Criminisi, A., Blake, A., Zisserman, A.: Geodesic star convexity for interactive image segmentation. In: CVPR, pp. 1–8 (2010)
20. He, K., Sun, J., Tang, X.: Fast matting using large kernel matting laplacian matrices. In: CVPR (2010)
21. He, L., Bleyer, M., Gelautz, M.: Object removal by depth-guided inpainting. In: Austrian Association for Pattern Recognition Workshop (AAPRW'11) (2011)
22. Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., Salesin, D.: Image analogies. In: SIGGRAPH, pp. 327–340 (2001)
23. Hilditch, C.J.: Linear skeletons from square cupboards. Mach. Intell. **4**, 403–420 (1969)
24. Komodakis, N., Tziritas, G.: Image completoin using global optimization. In: CVPR (2006)
25. Komodakis, N., Tziritas, G.: Image completion using efficient belief propagation via priority scheduling and dynamic pruning. IEEE TIP **16**(11), 2649–2661 (2007)

26. Kopf, J., Fu, C., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.: Solid texture synthesis from 2d exemplars. In: SIGGRAPH, pp. 1–9 (2007)
27. Kwatra, V., Schodl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. In: SIGGRAPH, pp. 277–286 (2003)
28. Lefebvre, S., Hoppe, H.: Parallel controllable texture synthesis. In: SIGGRAPH, pp. 1–8 (2005)
29. Levin, A., Lischinski, D., Weiss, Y.: A closed form solution to natural image matting. In: CVPR (2006)
30. Liang, L., Liu, C., Xu, Y., Guo, B., Shum, H.: Real-time texture synthesis by patch-based sampling. In: SIGGRAPH (2001)
31. Liu, Y., Belkina, T., Hays, J., Lublinerman, R.: Image de-fencing. In: CVPR (2008)
32. Mairal, J., Elad, M., Sapiro, G.: Sparse representation for color image restoration. IEEE TIP **17**(1), 53–69 (2008)
33. Nock, R., Nielsen, F.: Statistical region merging. IEEE TPAMI **26**(1), 1452–1458 (2004)
34. Park, M., Brocklehurst, K., Collins, R.T., Liu, Y.: Image de-fencing revisited. In: ACCV, pp. 422–434 (2010)
35. Qi, F., Han, J., Wang, P., Shi, G., Li, F.: Structure guided fusion for depth map inpainting. Pattern Recognit. Lett. **34**, 70–76 (2013)
36. Rother, C., Kolmogorov, V., Blake, A.: GrabCut: interactive foreground extraction using iterated graph cuts. ACM Trans. Graph. **23**(3), 309–314 (2004)
37. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: CVPR (2011)
38. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. In: CVPR, pp. 1–8 (2008)
39. Sun, J., Kang, S.B., Xu, Z.B., Tang, X., Shum, H.Y.: Flash cut: foreground extraction with flash and no-flash image pairs. In: CVPR (2007)
40. Sun, J., Yuan, L., Jia, J., Shum, H.: Image completion with structure propagation. In: SIGGRAPH, pp. 868–875 (2005)
41. Temlyakov, A., Munsell, B., Waggoner, J., Wang, S.: Two perceptually motivated strategies for shape classification. In: CVPR (2010)
42. Tschumperlé, D.: Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE's. IJCV **68**(1), 65–82 (2006)
43. Tschumperle, D., Deriche, R.: Vector-valued image regularization with PDEs: a common framework for different applications. IEEE TPAMI **27**(4), 506–517 (2005)
44. Wang, L., Jin, H., Yang, R., Gong, M.: Stereoscopic inpainting: joint color and depth completion from stereo images. In: CVPR (2008)
45. Wang, S.S., Tsai, S.L.: Automatic image authentication and recovery using fractal code embedding and image inpainting. Pattern Recognit. **41**, 701–712 (2008)
46. Wei, L., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: SIGGRAPH (2000)
47. Wexler, Y., Shechtman, E., Irani, M.: Space–time video completion. In: CVPR, pp. 1–8 (2004)
48. Yamashita, A., Matsui, A., Kaneko, T.: Fence removal from multi-focus images. In: International Conference on Pattern Recognition (ICPR'10) (2010)
49. Yamashita, A., Tsurumi, F., Kaneko, T., Asama, H.: Automatic removal of foreground occluder from multi-focus images. In: IEEE International Conference on Robotics and Automation (ICRA'12) (2012)

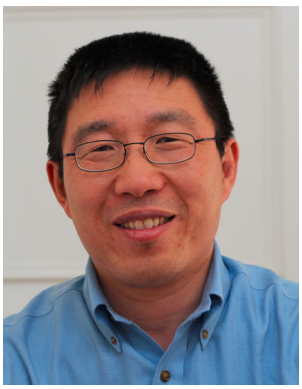
**Qin Zou** received his BE degree in information science in 2004 and PhD degree in photogrammetry and remote sensing (computer vision) in 2012 from Wuhan University. From October 2010 to October 2011, Qin Zou was a visiting student at the Computer Vision Lab, University of South Carolina. Currently, Qin Zou is with the School of Computer Science, Wuhan University. His research activities involve computer vision, machine learning, ubiquitous computing, intelligent transportation systems, etc. He is a member of IEEE.



**Yu Cao** received his BS degree in information and computation science from Northeastern University, Shenyang, China, 2003, MS degree in Applied mathematics from Northeastern University, Shenyang, China, 2007, and PhD degree in computer science from University of South Carolina, SC, 2013. His current research interests include computer vision, machine learning, and pattern recognition. He is a student member of IEEE.

**Qingquan Li** received his PhD degree in GIS and photogrammetry in 1998 from Wuhan Technical University of Surveying and Mapping, China. He is now the president and a professor of Shenzhen University, China. His research areas include three-dimensional and dynamic data modeling in GIS, location-based service, surveying engineering, integration of GIS, GPS and RS, intelligent transportation system, road surface checking, etc.

**Qingzhou Mao** is an associate professor of Wuhan University. He received his PhD degree in photogrammetry and remote sensing from Wuhan University in 2008. His research areas of interest include machine vision, image processing, close-range photogrammetry, mobile mapping, multi-sensors integration, industrial measurement and inspection, etc.



**Song Wang** received his PhD degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (UIUC) in 2002. From 1998 to 2002, he also worked as a research assistant in the Image Formation and Processing Group at the Beckman Institute of UIUC. In 2002, he joined the Department of Computer Science and Engineering at the University of South Carolina, where he is currently a professor. His research interests include computer vision, medical image processing, and machine learning. He is currently serving as the publicity/web portal chair of the Technical Committee of Pattern Analysis and Machine Intelligence, IEEE Computer Society and an associate Editor of Pattern Recognition Letters. He is a senior member of the IEEE and the IEEE Computer Society.