Vis Comput (2010) 26: 157–169
DOI 10.1007/s00371-009-0359-8

ORIGINAL ARTICLE

Lazy texture selection based on active learning

Tian Xia · Qing Wu · Chun Chen · Yizhou Yu

Published online: 16 April 2009

Abstract Interactive selection of desired textures and textured objects from a video is a challenging problem in video editing. In this paper, we present a scalable framework that accurately selects textured objects with only moderate user interaction. Our method applies the active learning methodology, and the user only needs to label minimal initial training data and subsequent query data. An active learning algorithm uses these labeled data to obtain an initial classifier and iteratively improves it until its performance becomes satisfactory. A revised graph-cut algorithm based on the trained classifier has also been developed to improve the spatial coherence of selected texture regions. We show that our system is responsive even with videos of a large number of frames, and it frees the user from extensive labeling work. A variety of operations, such as color editing, compositing, and texture cloning, can be then applied to the selected textures to achieve interesting editing effects.

Keywords Texture descriptors · Segmentation · Supervised classification · Graph cut · Scribbles

T. Xia (✉) · Q. Wu · Y. Yu
Department of Computer Science, University of Illinois, 201 N Goodwin Ave, Urbana, IL, USA
e-mail: tianxia2@illinois.edu

Q. Wu
e-mail: qingwu1@illinois.edu

Y. Yu
e-mail: yyz@illinois.edu

C. Chen
College of Computer Science, Zhejiang University, Hangzhou, China
e-mail: chenc@zju.edu.cn

1 Introduction

Object selection and cutout from images and videos have proven to be a vital technology with many applications in computational photography, image synthesis, and special visual effects for film making. These applications typically require pixel-level accuracy. With today's image processing and computer vision methods, computers still need human assistance in successfully performing this task at such a high level of accuracy. Hence, there has been much work on interactive object selection and cutout [20, 21, 29, 34]. Such work typically cuts out a single connected object which is assumed to have a global color distribution model, such as a mixture of Gaussians.

In this paper, we focus on texture and textured object selection from a video. Here texture refers to a unique local spatial arrangement of colors, such as the stripes on a zebra. There are reasons why existing techniques are not suitable for texture or textured object selection. *First*, since two different texture patterns may observe the same global color distribution, a global color distribution model would be inherently ambiguous in texture discrimination. *Second*, texture regions may be fragmented. Overly emphasizing spatial coherence to compensate the weakness in color distribution models would not work well.

We propose to perform interactive texture and textured object selection using a supervised classifier interactively trained using active learning. A texture classifier demands training examples interactively supplied by the user to improve its performance. Instead of the user searching for new training examples, we adopt the active learning methodology [1, 16, 31] which makes the user and the machine cooperate to find minimal data sufficient to train a good classifier. The machine plays a more active role, trying to figure out which subset of the originally unlabeled data, once labeled,

Fig. 1 Texture selection and editing examples. Regions with cloth textures are first cut out using our texture selection method and then replaced with new textures using texture cloning

would be the best training data. Once such data has been discovered, the user is asked to provide their correct labels. In this mode, the user only needs to answer a small number of queries from the machine. Hence user intervention is much reduced. We have confirmed experimentally that training data automatically chosen by the machine can indeed improve classification performance more significantly than those chosen by the user. Required by interactive performance, we adopt boosted decision trees as the classifier whose training sessions can be quickly performed in an interactive rate.

Powerful classification algorithms still require discriminative data. There has been extensive work on texture descriptors in the image processing and computer vision literature [17, 24]. Instead of a global color distribution model, we adopt local texture descriptors based on responses to an oriented filter bank. Such texture descriptors have proven to be effective in texture discrimination and are actually used as input to our texture classifier.

The interactively trained texture classifier achieves a high success rate. In the event of minor classification errors, we rely on the graph-cut algorithm to remove such errors by minimizing a revised objective function which effectively incorporates intermediate results from the classifier. More specifically, the likelihood function is formulated according to the confidence value returned by the classifier. In our experiments, such a formulation of the objective function has proven to be very effective in the refinement of the intended texture selection.

Our system remains responsive throughout the interactive session, and is not sensitive to an increasing number of frames in video. With the guaranteed scalability, the user can now select complicated texture regions and textured objects throughout the whole video by drawing only a few initial scribbles followed by a few more to label the query data. Afterwards, one can conveniently achieve color editing, compositing, and texture cloning operations on the selected regions or objects.

2 Related work

Unlike video texture selection, there is a large body of literature on image texture classification and segmentation [3, 7, 10, 23, 25, 32]. Since local pixel configurations provide vital information of a texture, one popular method for texture discrimination is based on filtering which can effectively characterize local patterns. Common filters used for this purpose include first and second derivatives of the Gaussian [23], Gabor filters [10], and wavelet decomposition [7]. Both classifier-based [25] and boosting-based [3, 32] techniques have been developed to automatically perform texture classification or segmentation. Nevertheless, these methods have not reached our desired level of accuracy.

Many interactive techniques have been successfully developed for object cutout from still images [2, 13, 21, 26, 27, 29]. These techniques are primarily designed for image regions with spatial coherence rather than textures with much less coherence. The Gaussian mixture model commonly used for color distribution has been extended to have discriminative power for textures in [12, 33]. In particular, a Gaussian mixture model for principal components extracted from texture description has been adopted in [12]. Nevertheless, only three leading principal components were used in [12]. In comparison, since we adopt a supervised classifier with a fast training algorithm, it has become possible for us to have a texture descriptor with more discriminative power by exploiting at least an order of magnitude more principal components.

An important inspiration of our technique came from user-defined scribbles which have been extensively exploited among aforementioned interactive techniques for images. As described in [12, 29], it is promising to have the user explicitly mark only positive or negative points while the rest of the image is implicitly labeled as the opposite. However, one-sided initial scribbles do not guarantee to converge if the color distribution of the background and foreground are somewhat similar. Protiere and Sapiro [27] proposed a novel segmentation technique that computes for each pixel a weighted distance to user scribbles, but the algorithm relies on a relatively large number of initial and subsequent scribbles by the user. Our method does not need a large number of scribbles or deliberate positioning of them, and both positive and negative scribbles are used in order to train a more accurate classifier.

Scribbles have also been utilized in many other topics, including colorization [15, 18, 22, 28] and local color adjustment [19]. Texture descriptors based on Gabor filters were adopted in [28] to assist level-set propagation. Texture patch similarity constraints were taken into account in [22] to solve a labeling problem. Li et al. [19] trains a boosting-based classifier for edge-aware interpolation. Unlike the classification method in [15], we perform active learning directly on the input data and do not need precomputed examples from a database. Note that colorization and segmentation are two related but different problems. While segmentation demands an accurate object boundary to be found in support of object cutout and compositing operations, colorization only alters pixel colors and, in most cases, does not aim at a precise boundary in pixel resolution.

A certain level of user guidance has been incorporated with static and dynamic texture editing. Neighborhood-based self-similarity were exploited in [6] to quickly select features intended for further editing. However, there was no supervised classification except a threshold for neighborhood similarity. User guidance has also been exploited in dynamic texture editing to alter the trajectory [4] or dynamics [9] in the original images.

Video texture classification poses a harder problem. Unfortunately, all interactive cutout methods mentioned above were precisely designed for static images. To our knowledge, no generalization from static image to video is ever clearly mentioned due to the fact that the user's workload may potentially increase in proportion to the number of video frames. Furthermore, in the relatively little literature of interactive video object cutout, [20, 34] were not originally designed for texture or textured object selection. More importantly, these techniques inevitably suffer from the scalability problem, as they treat the costly 3D graph cut as an indispensable part of their interactive session. Our method well handles this problem by freeing the whole interactive session from the graph cut overhead. A revised graph-cut algorithm is only needed as a postprocessing step in our system, hence the interaction is not subject to the growing number of video frames. This is particularly useful for video texture selection with moderate user interaction.

3 Overview

During the preprocessing stage, spatially and temporally nearby pixels sharing similar attributes are grouped into *supervoxels*. In the special case of a static image, each supervoxel is actually a superpixel. For each supervoxel, a vector texture descriptor is obtained by calculating its response to an oriented filter bank.

During the online user interaction stage, the input video is presented to the user as a sequence of frames. The user draws a few scribbles on one frame indicating the desired texture regions. The supervoxels covered by these initial scribbles serve as the set of training data on which the first tentative classifier is trained and applied to the unlabeled data. The classifier also computes a confidence value for each of the unlabeled supervoxel. It then chooses a small number of supervoxels with lowest confidence, and queries the user for their correct labels. The user's feedback is used to obtain an improved version of the classifier, which again chooses the least confident supervoxels to question the user. This iterative procedure normally runs for a few times before the classifier performs sufficiently well on the video.

Usually, the improved classifier is not completely accurate in prediction. It is likely to have minor misclassifications. In the video, these labeling mistakes are observed as noisy pixels inside and outside the intended region, for which further refinement is needed. We refine texture selection results by breaking supervoxels into pixels and applying a revised graph-cut algorithm with an innovative weighting scheme based on the confidence values obtained by our classifier. For videos, graph cut is applied to all frames as an offline batch postprocessing step once the online interactive training session for the classifier is over. For static images, we include graph cut as an additional step in the interactive training session since graph cut has reached interactive performance on single images. We alternate classifier training and graph cut on the input image until convergence. The whole procedure is demonstrated in Fig. 2.

Though mainly designed for video texture selection, our system is a perfect fit for image cutout too. For convenience of presentation, most illustrations shown in the paper are experiments on images. Please refer to supplemental clips for video selection results.

4 Preprocessing

To make run-time texture selection more efficient and responsive, we preprocess the video using the algorithm in [11] so that supervoxels are formed as working units for following operations. A supervoxel corresponds to a small group of geometrically close pixels that share similar color and intensity attributes. Instead of processing every single pixel within the texture video, we only process a representative pixel within each supervoxel and let the rest share the same result. For similar reasons, superpixels have been found useful in [14, 21]. As in [20, 34], we generalized superpixels to supervoxels so nearby pixels across consecutive frames are also incorporated to enhance the temporal coherence of the video. We set an upper bound on the number of pixels within a supervoxel (typically 50) and confine each supervoxel within a cube to prevent long slivers and spiral-like shapes.

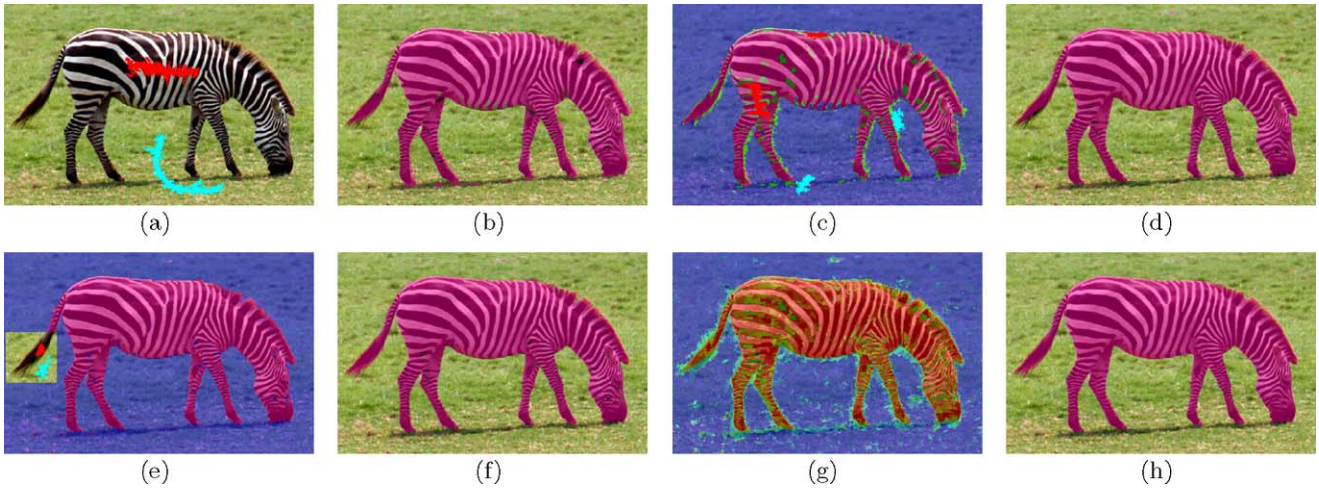


Fig. 2 The texture selection procedure. (**a**) Initial scribbles. (**b**) Tentative classification by the first classifier. (**c**) Query points (*green*) and user provided labels (*solid red* and *cyan*). In most cases, a few scribbles are sufficient to have a tangible improvement of the classifier's performance. The user can answer the query by covering the uncertain area with positive or negative scribbles. Note that the user does not have to label all query points. (**d**) Improved classification after the query. (**e**) and (**f**) A local classifier is trained to refine the tail (optional). (**g**) Signed confidence map of the final classification. A continuous value in $[-1, 1]$ is computed for each pixel. -1 (*blue*) and 1 (*red*) indicate the highest confidence in classification, while 0 (*green*) implies high uncertainty. (**h**) Refinement by the revised graph-cut algorithm. Note that our scribbles have rough and irregular boundaries because they are displayed as the union of covered supervoxels

The second step of preprocessing is to obtain texture descriptors for each supervoxel. Oriented filter banks [17, 24] have proven to be an effective tool to characterize textures. Our learning algorithm primarily uses local statistics of oriented filter responses to differentiate textures. We apply 24 Gabor filters [24] at six orientations and four scales at every pixel, and such filtering is performed for each of the three color channels separately. Thus, every pixel has a 72-component filter response vector after filtering. We compute the mean and standard deviation of each channel over all pixels within every supervoxel to obtain a 144-component vector. Note that the Gabor filters can gather texture information not only within a supervoxel but also in areas surrounding the supervoxel because the support region of the filters extends well beyond the supervoxel.

At the representative pixel of every supervoxel, a color histogram is also extracted from its 7×7 neighborhood. We used 10 bins for each color channel, and the 30-component histogram vector is then concatenated with the 144-component vector. It is reduced to a shorter vector by principal component analysis. This shortened vector, in juxtaposition with the mean and standard deviation of each color channel, forms the texture descriptor for every supervoxel. The inclusion of color information in descriptors effectively alleviates the tendency of blurring the actual boundary when only texture features are used. In our experiments, we have used different descriptor lengths for different examples, normally ranging from 30 to 60. These descriptors are the data points fed to Algorithm 1 during interactive training.

Algorithm 1 The Query-by-Boosting algorithm

Data: $\mathbf{X}$ (unlabeled point set), N (number of trials) and m (number of query candidates)

begin
  Initialize $S_1 = \{(\mathbf{x_1}, l_1)\}$ for random $\mathbf{x}_1$;
  for *i=1 to N* **do**
    Obtain a boosted classifier H_i on S_i;
    Randomly generate a set of m points, $C \subset X \backslash S_i$;
    Pick a point $\mathbf{x}^* \in C$ with the minimum margin:
      $\mathbf{x}^* = \arg\min_{\mathbf{x} \in C} |\sum_t \alpha_t h_t(\mathbf{x})|$;
    Query the label at $\mathbf{x}^*$, $l(\mathbf{x}^*)$;
    Set S_{i+1}=append$(S_i, (\mathbf{x}^*, l(\mathbf{x}^*)))$.
  end
end

Result: the boosted classifier at the last iteration, H_N.

As an option, we have also experimented with the oriented filter bank in [17], another filter bank known for its success in texture discrimination, and achieved comparable results.

5 Active learning based selection

In this section, let us first review the active learning approach. Suppose that the input data to supervised classification consist of a set of tuples, $(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), \ldots, (\mathbf{x}_n, l_n)$, where $\mathbf{x}_i \in \mathbf{X}$ denotes one of the data, and l_i denotes the class label of that datum. $\mathbf{x}_i$ is typically a vector in a multidimensional space $\Re^d$. If one is only concerned

with two classes of data, $l_i \in L = \{-1, +1\}$. Classification is then equivalent to finding a boundary surface in this d-dimensional space so that data with different labels do not lie on the same side. As we can imagine, data points close to the boundary play a much more important role in defining the shape of the boundary than those further away. This is especially true when data with different labels interlace, which gives rise to a jagged decision boundary. A precise definition of the boundary requires a much denser sampling of the space around the boundary than anywhere else.

Active learning [1, 16, 31] is exactly motivated by this observation. It suggests a way for the learning algorithm to proactively choose training data instead of passively receiving them. Once an initial boundary is defined, it keeps refining the boundary automatically by sampling more and more points around it. Obviously, labels for these additional data points need to be supplied by the user. But how can the learning algorithm know which points lie close to the boundary? A typical strategy is to train multiple classifiers and conduct an internal voting to determine the uncertainty of a data point. A point is considered close to the decision boundary when the voting result becomes close to 50–50, which indicates a high uncertainty.

Algorithm 2 The AdaBoost algorithm

Data: $(\mathbf{x}_1, l_1), ..., (\mathbf{x}_n, l_n)$ where $\mathbf{x}_i \in \mathbf{X}$, $l_i \in \{-1, 1\}$
begin
  Initialize distribution, $D_1(i) = 1/m, i = 1, ..., n$;
  for *t=1 to T* **do**
    Using D_t, train base classifier $h_t : \mathbf{X} \to \{-1, 1\}$ with small error $\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq l_i]$ on D_t;
    Set $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t l_i h_t(\mathbf{x}_i))}{Z_t}$, where Z_t is a normalization factor, and $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$.
  end
end
Result: the final classifier, $H(\mathbf{x}) = \text{sign}\left(\sum_t \alpha_t h_t(\mathbf{x})\right)$.

Active learning needs to work with a fast classification algorithm to achieve its goal. In this paper, we adopt decision tree classifiers enhanced with the AdaBoost algorithm [30] (Algorithm 2). Before binary classification, the classifier first computes a margin, $|\sum_t \alpha_t h_t(p)|$, which is also defined as the *confidence*. Boosted decision trees can be trained interactively while achieving a strong discriminative power. Note that other classifiers with a fast training phase can also be adopted, such as the soft-margin linear SVM. A method that integrates boosting with active learning has been developed in [1]. An outline of the method is shown in Algorithm 1. It assumes the knowledge of the entire pool of unlabeled data. Since this pool may be very large, during each iteration, it generates only a random subset of the unlabeled data, and the uncertainty of each point in the subset is indicated by the margin of the weighted votes cast by all the base classifiers trained by AdaBoost. The method finally chooses the point with the highest uncertainty, i.e., minimum confidence, and requests the user to provide its label. The chosen point along with its label is then appended to the set of training data.

5.1 Initial feature selection

Run-time texture selection is cast as a binary supervised classification based on Algorithm 1 with certain revisions. In the current context, the set of unlabeled data, **X**, consist of all supervoxels within the input video. The machine randomly selects an initial video frame to start with and let the user draw one or more scribbles within that frame to mark desired features. Texture descriptors at supervoxels touched by these scribbles (note that a supervoxel might straddle across several frames) are marked as positive training data. The user also draws one or more additional scribbles to mark negative training data. Positive and negative labels are conveniently communicated using left and right mouse buttons, respectively. We apply the K-means algorithm to cluster supervoxels covered by the positive and negative scribbles, respectively, and uniformly sample each cluster. Given both types of sampled training data, an initial boosted classifier can be obtained as in Algorithm 2. Unlike Algorithm 1, we allow multiple initial training data. Note that initial scribbles do not increase with respect to the number of video frames, as it is confined to one frame and requires no more scribbles than a single image does.

5.2 Automatic query

Once an initial classifier has been obtained, as in Algorithm 1, we run multiple subsequent iterations with user query to refine the initial classifier. Each iteration proceeds as follows. Instead of generating a random subset of the unlabeled supervoxels from the entire video as in Algorithm 1, we only generate query candidates from one randomly chosen video frame. All user interactions within that iteration will be focused on the data from that frame only because shuffling among multiple frames would be too distracting. If the number of supervoxels within that frame is sufficiently small, all of them become query candidates; otherwise, a random subset of these supervoxels are generated as query candidates. The classifier from the previous iteration is applied to generate confidence values for all query candidates.

This strategy has important advantages. The entire video consists of a large number of supervoxels, and a random subset of the entire video may fail to include important texture descriptors that lie close to the decision boundary, making active learning less effective. On the other hand, supervoxels from one single video frame provides a much narrowed

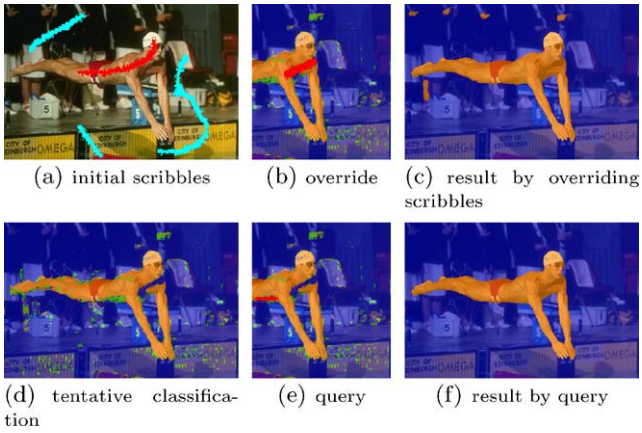


Fig. 3 Initial scribbles shown in (**a**) are used to train a tentative classifier, whose classification result is shown in (**d**). Displayed in *green* is the automatic query by the machine. (**b**) Illustrates the overriding scribble (in *red*) based on the classification in (**d**). It manages to classify the swimmer but also introduces misclassifications in the background as a side effect shown in (**c**). (**e**) Shows the user's feedback (in *red*) based on the query in (**d**). This query significantly improves the accuracy of the classifier without negatively affecting previously correct labels. (**c**) and (**f**) are final results after applying graph cut

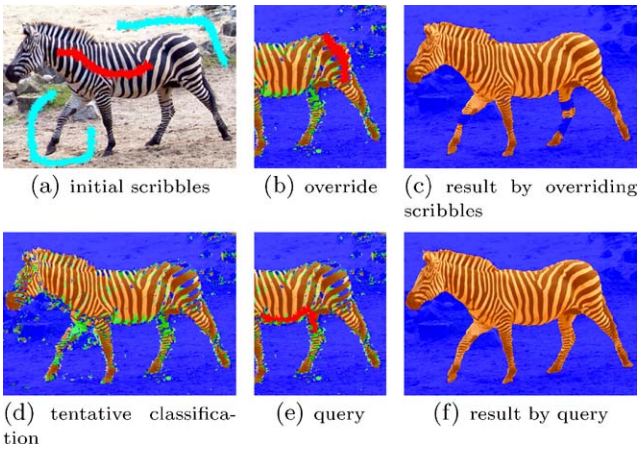


Fig. 4 In (**b**), the overriding scribble (in *red*) on the zebra's rear part have well corrected nearby misclassifications. But it eventually leads to a classifier that fails to recognize similar stripe patterns on the zebra's legs in (**c**). (**e**) In contrast, by feeding correct labels (in *red*) to query points (in *green*), we manage to build a stable classifier with high accuracy. (**c**) and (**f**) are the final results after applying graph cut

scope. Sampling within the much smaller pool of supervoxels significantly increases our chance to choose points close to the decision boundary as query points.

Our algorithm offers a high degree of flexibility to condition the training of classifiers. Instead of choosing only one query point at a time as in Algorithm 1, we allow multiple query points every iteration to speed up the convergence. We choose from the candidate pool all points with a margin smaller than a prescribed threshold (∼0.1–0.15) to inquire the user. Supervoxels covered by these query points are all shown in a special color. Instead of providing the correct label to one query point at a time, the user can draw relatively long scribbles to paint multiple query points with the same label. Painting with labels can be careless because only query points are affected by the scribbles and the rest of the supervoxels are masked.

In addition, the user can directly draw overriding scribbles over incorrectly labeled data in addition to labeling the query data. The input of such overriding scribbles corrects partial classification errors in previous rounds, if any, and leads to a more accurate classification in the next round. Nevertheless, overriding scribbles are applied rather infrequently. In practice, the percentage of such scribbles is typically around 10%.

Once the learning algorithm has obtained the labels of the query points and possibly overriding scribbles, an improved classifier is trained using the newly labeled supervoxels in the current iteration together with a uniformly sampled subset of the labeled supervoxels from previous iterations, and the procedure is repeated.

5.3 Local classifiers

A video sometimes has a region with abundant texture details that requires considerably more training data than other regions do. When this is the case, our algorithm is inefficient because the candidate query points are randomly (and thus uniformly) generated and the region in question tends to be undersampled. We handle the situation by providing the user with an option to define a 3D rectangular region of interest within the video, and train a local classifier for the specified region. The local classifier works exactly the same way as the global one does, only confined within the user specified region. Apparently, the detailed region is much smaller compared to the whole video, resulting in a faster rate of convergence. It is therefore a good practice to refine regions of detailed textures with a local classifier after the classification for the entire video.

5.4 Validation of active queries

In this section, we demonstrate that active queries at data points close to the decision boundary can most effectively and quickly improve the accuracy of the classifier. As a result, by providing the queries automatically and accelerating the convergence of the classifier, active learning can tangibly reduce the amount of user interaction.

We compared the effectiveness of active queries and overriding scribbles in improving the overall classification accuracy. In our experiments, we have observed that active queries in a local image region not only improve the classification results in that local region but also improve the decision boundary on a more global scale. For example, in Figs. 3 and 4, the labels provided to the active query have

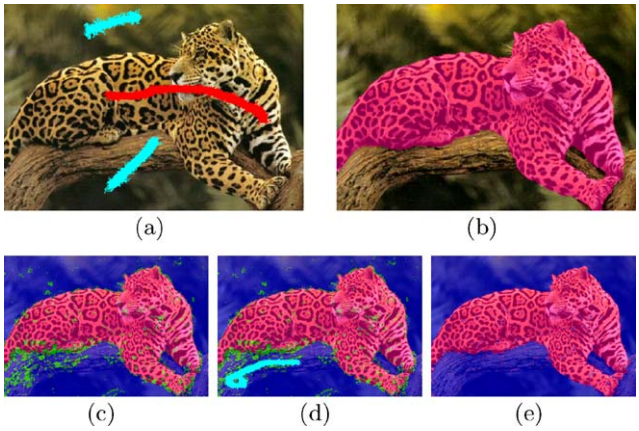


Fig. 5 (**a**) Illustrates initial scribbles used for training the tentative classifier. Its classification result is shown in (**c**), where regions in *green* are displayed as an automatic query by the machine. As we see in (**d**), user-supplied labels (in *cyan*) are only a subset of all query points, avoiding locations not obvious to the eye. An improved classifier is then trained, and the new classification is shown in (**e**). (**b**) Is the final result after applying the graph cut to the classification in (**e**)

succeeded in correcting all misclassifications. Meanwhile, the improved classification results remain stable meaning that active queries in one region tend not to negatively affect previously correct labels in other regions. On the other hand, overriding scribbles sometimes tend to overemphasize textures they cover and result in undesirable side effects elsewhere, as shown in Fig. 3. In addition, as shown in Fig. 4, overriding scribbles also exhibit an inability to simultaneously correct the erroneous labels of similar patterns that may not be spatially close.

Because of the ability of active queries in improving classification accuracy on a global scale, it is not necessary for the user to provide labels at all query points. In fact, tangible improvements within an entire image can be achieved with a very small number of queries. Thus, the user can choose not to label the query points at ambiguous locations, such as those close to the boundary of the intended texture region. As shown in Fig. 5, user-supplied labels at exterior query points actually improve the classification result at the texture boundary.

6 Selection refinement

As has been described in previous sections, a classifier makes the most uncertain decisions on its decision boundary. Another source of misclassification is the presence of outliers. As each data point (supervoxel) is classified according to its own texture descriptor, the relationship among neighboring supervoxels is not taken into account by the classifier. The neglect of the spatial coherence across supervoxels gives rise to occasional outliers in the video. These observations have motivated us to use the graph-cut algorithm as a selection refinement mechanism at the pixel level. In addition, a refinement at the pixel level is able to break misclassified super-voxels that straddle across weak boundaries.

6.1 Graph cut

Graph cut [5] has been used as a powerful image segmentation tool in recent years. The underlying idea is to treat image segmentation as a binary labeling problem. Specifically, the image induces a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. A node corresponds to a pixel in the image. There is an edge between every pair of neighboring pixels, whose weight (cost) encodes the difference of the two pixels. The more they differ, the smaller the cost. Every pixel also has links to two virtual nodes, the source representing the foreground and the sink representing the background. The weight on each link indicates how likely it is for the pixel to belong to the foreground and background. The labeling problem is to assign a unique label l_i for each node $i \in \mathcal{V}$, i.e., $l_i \in$ {foreground(=1), background(=0)}. The solution $L = \{l_i\}$ can be obtained by minimizing the following objective function:

$$E(L) = \sum_{i \in \mathcal{V}} R(l_i) + \lambda \sum_{(i,j) \in \mathcal{E}} B(l_i, l_j), \tag{1}$$

where $R(l_i)$ is a likelihood energy, encoding the cost when the label of node i is l_i, and $B(l_i, l_j)$ is a prior energy for boundary penalty, denoting the cost when the labels of adjacent nodes i and j are l_i and l_j, respectively. If we set the weight of edge (i, j) to $B(l_i, l_j)$, the weight of the link between node i and the source to $R(l_i = 1)$, and the weight of the link between node i and the sink to $R(l_i = 0)$, finding an optimal solution of (1) is equivalent to finding a cut with minimal cost in the graph. More details can be found in [5] and [29].

We would like to make use of the interactively trained texture classifier in the likelihood energy of the graph-cut algorithm. Since the result returned by a classifier is essentially a complicated nonlinear function of the training examples, it can be considered as an accurate example-based parametric model built upon texture descriptors. Therefore, instead of using a global color distribution to set the source/sink link cost as in [29], we opted for the signed confidence value, $\sum_t \alpha_t h_t(p)$, computed in Algorithm 2, to relate to the likelihood energy. Specifically, we set

$$R(l_i = 0) = \max\left(0, -\ln \max\left(0, 0.5 + 0.5 \sum_t \alpha_t h_t(p)\right)\right)$$

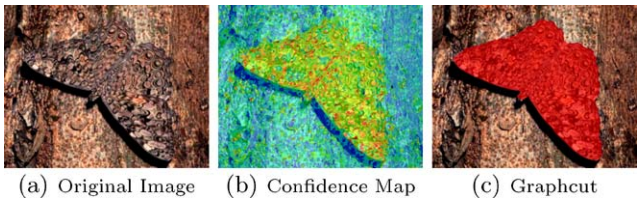


Fig. 6 With difficult examples where foreground and background color distributions are similar, the interactively trained classifier results in misclassifications at random locations as in (**b**), where each pixel is assigned a signed confidence value. (**c**) Graph cut based on signed confidence values computed by the classifier proves to be a powerful refinement method

and

$$R(l_i = 1) = \max\left(0, -\ln\max\left(0, 0.5 - 0.5\sum_t \alpha_t h_t(p)\right)\right).$$

To maintain spatial coherence and remove classification outliers, we adopt the conventional scheme of assigning edge cost where the color difference between adjacent pixels is used, i.e.,

$$B(l_i, l_j) = |l_i - l_j| \exp\left(-\frac{\|\mathbf{c}(i) - \mathbf{c}(j)\|^2}{2\sigma_c^2}\right),$$

where $\mathbf{c}(i)$ represents the color vector at node i. Figure 6 illustrates the result using the graph-cut method.

6.2 Graph cut integration

Considering the difference in computational cost when the graph-cut algorithm is respectively applied to a single image and a video, it is integrated with the rest of the texture selection process in two different schemes.

Since the graph-cut algorithm can achieve interactive performance on a single image, it is interleaved with classification and active learning to iteratively refine texture selection results. More specifically, starting from a tentative classification C_i, we perform active learning and automatic queries to obtain an improved classification C_{i+1}. A graph based on C_{i+1} is then formulated as in Sect. 6.1, followed by a minimum cut. Note that the graph is defined at the pixel level and all pixels within the same supervoxel receive the same value related to the likelihood energy. The foreground and background regions from the minimum cut are then uniformly sampled. These sample pixels, together with pixels on user scribbles and query feedbacks, are collectively used as positive and negative training data, from which a new classifier is trained (Algorithm 2). In the event a supervoxel is broken into foreground and background regions by the graph cut, majority voting is performed on the supervoxel to determine its new label. This procedure is repeated until convergence. Note that for images with relatively simple textures, we can skip the active query step within each iteration and simply alternate between classifier training and graph cut to achieve satisfactory results (Fig. 7).

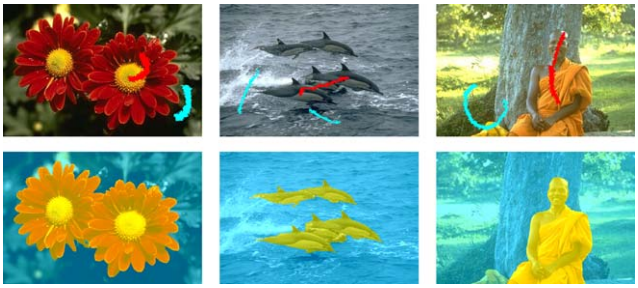


Fig. 7 With initial scribbles illustrated in the *first row*, our iterative method can successfully select desired regions without any further user interaction

However, a graph cut for dozens or even hundreds of video frames is too computationally expensive to achieve interactive update rates. Therefore, we take the graph-cut algorithm out of the interactive session and simply apply it as an offline postprocessing step in video texture selection. More specifically, once multiple iterations of active learning have been performed to obtain a satisfactory classifier and its classification results, the user runs a postprocessing step where a framewise graph cut is applied at the pixel level to further refine selected texture regions. Temporal coherence across consecutive frames is secured by the supervoxel mechanism (Sect. 4) in the classification stage and will not be adversely affected by the framewise graph cut.

7 Experimental results

We have experimented with a series of video sequences and static images on a 3.8 GHz Pentium 4 processor. As described in Sect. 5, our classifier only requires shallow decision trees of moderate accuracy, whose performance are later boosted by AdaBoost and active learning. Building such decision trees (usually three or four levels in depth) using training data with reduced dimensionality is very fast. This gives rise to the computational efficiency shown in the Training column of Table 1, and it normally takes 0.2 to 0.6 seconds to train a completely new classifier for video sequences and static images after each round of interactive query and labeling. Because the training time for each classifier is well below one second, labeling and training stages have thus been seamlessly integrated and have reached an overall interactive rate. Because the total amount of training data collected from initial labeling and subsequent query sessions does not increase significantly for videos, the time needed for each round of training does not have obvious difference between videos and static images.

One major advantage of our video texture selection method over existing video object selection methods

Table 1 Timings for video sequences and static images. Performance measurements were taken on a 3.8 GHz Pentium 4 processor. "Feat. dim" denotes the dimensionality of texture descriptors; "Prep" denotes the preprocessing time; "Training" denotes the training time for classifiers. It has two subcolumns. "Total" means the accumulated training time for a dataset while "Per query" means the average training time for a new classifier after each round of query. "User interaction" denotes the total artist time. Graph cut is performed as offline postprocessing for videos. However, for static images summarized in the *bottom half* of the *table*, time spent on "Graphcut" indicates the accumulated time from multiple iterations

Video (or image)	Size	Feat. dim	Prep	Training		Graphcut	User interaction
				Total	Per query		
Flower	352 × 288 × 30	30	15 min	5.1 sec	0.26 sec	12 sec	6 min
Sea plant	328 × 264 × 104	30	54 min	10.8 sec	0.31 sec	1 min	23 min
Smoke	455 × 355 × 33	30	19 min	3.7 sec	0.37 sec	8 sec	5 min
Bubbles	328 × 264 × 32	45	17 min	6.4 sec	0.34 sec	17 sec	10 min
Jaguar	352 × 288 × 35	60	21 min	7.8 sec	0.52 sec	21 sec	13 min
Zebras	321 × 252 × 107	50	65 min	16.5 sec	0.41 sec	1.1 min	30 min
Curtain	396 × 271 × 30	60	22 min	6.9 sec	0.34 sec	18 sec	8 min
Shirt	312 × 222 × 35	60	25 min	7.9 sec	0.37 sec	21 sec	11 min
Monarch butterflies	529 × 347	60	69 sec	3.5 sec	0.37 sec	3.8 sec	67 sec
Fabrics	442 × 349	60	59 sec	3.2 sec	0.31 sec	4.2 sec	50 sec
Dress	326 × 548	40	52 sec	2.2 sec	0.29 sec	3.0 sec	48 sec
Big cat	371 × 350	40	63 sec	3.1 sec	0.43 sec	3.4 sec	78 sec
Group zebras	457 × 297	55	55 sec	2.7 sec	0.38 sec	2.2 sec	96 sec
Fungi	538 × 339	40	64 sec	0.85 sec	0.21 sec	–	27 sec

[20, 34] is that ours keeps graph cut out of the interactive session and only applies it as an offline postprocessing step. Existing methods integrate graph cut as an indispensable part of the interactive session, making a prompt response vulnerable to the growing size of the video data. In contrast, each iteration of our interactive session only includes fast training and classification, and exhibits very good response time and scalability, which are crucial for video editing. It compares favorably with the performance of each iteration of existing video object selection methods whose performance deteriorates significantly with an increasing number of video frames due to the graph cut within the interactive loop. If we discount computation time within each iteration, the total user interaction time of our texture selection method is comparable to that reported in [20, 34] which was not originally designed for texture selection.

Our algorithm has two essential components, namely, active learning (query) and a supervised classifier on which active learning is based. We first verify the performance of the supervised classifier. To leave active learning out of the pipeline, we adopt the iterative approach mentioned in Sect. 6.2, alternating classification with graph cut without further user interaction (query) until convergence. As shown in Fig. 7, our classifier is capable of identifying relatively simple texture patterns without active queries. To further demonstrate the discriminative power of our texture classifier, we compare it with state-of-the-art object selection algorithms. Both GrabCut [29] and P-Channel [12] use the same iterative framework while relying on the Gaussian mixture model for texture discrimination. As shown in Fig. 8, our classifier produces better results than other methods, especially along object boundaries. Texture discrimination based on similarity of Gabor filter responses fail to precisely recognize distinct texture patterns due to its unsupervised nature.

With a powerful base classifier ensured, we then examine if active learning and query further boost the performance on examples with sophisticated textures and scene composition. We conducted a number of comparisons with GrabCut [29] which is also allowed to accept additional user scribbles in between iterations. Using an equivalent number of scribbles, our active learning based method performs at least as well as GrabCut in all experiments, and consistently produces better results on examples with scattered texture regions. As shown in Fig. 9, GrabCut does poorly in selecting such scattered texture regions because of its specific graph-cut formulation based on the Gaussian mixture model. Notice that GrabCut needs considerably more user scribbles in the presence of discrete objects, and it often mistakes texture variations in the background for the object boundary. Since the video object selection methods in [20, 34] are based on the same graph model adopted in GrabCut, they exhibit the same type of problems with scattered texture regions. Our method handles these situations well, as demonstrated in this comparison and other examples throughout the paper and supplemental materials.

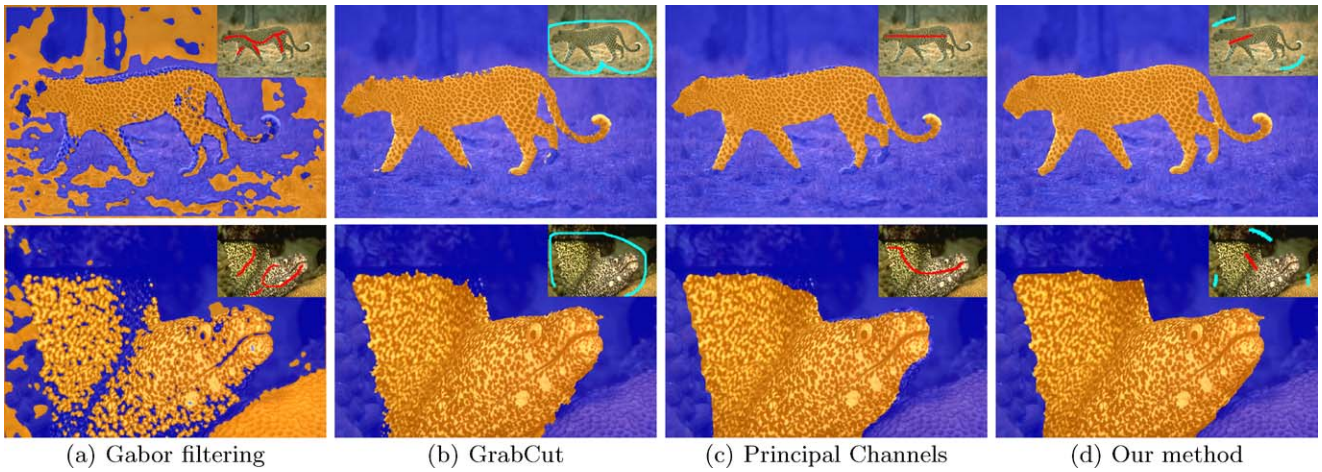
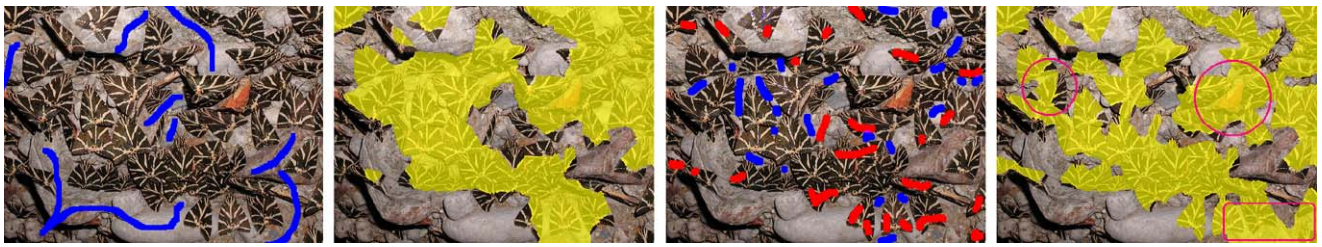
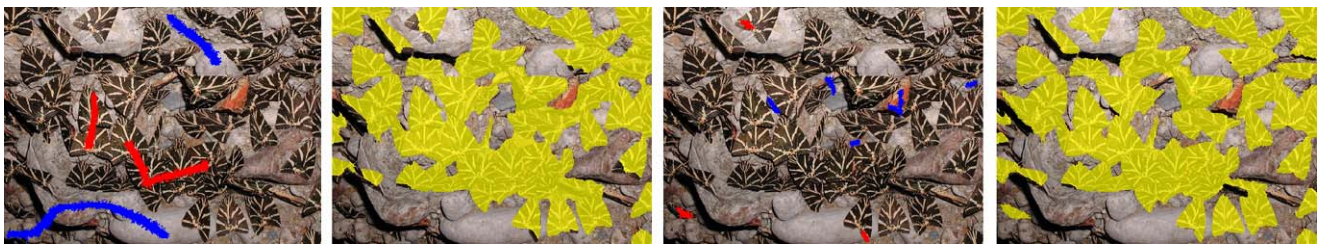


Fig. 8 Comparison among multiple object selection methods (selection based on similarity of Gabor filter responses, grabcut [29], p-channel [12], and our lazy texture selection). The *smaller images* in the *upper right corner* show initial scribbles needed for each method. Our method (the *last column*) excels in precisely identifying object boundaries (e.g., paws of the leopard and the mouth of the fish)



(**a**) Selection using GrabCut. With quite many initial scribbles, the result after the first iteration still tends to have inaccurate boundaries. It yields a moderate foreground selection after additional heavy user interactions. Obvious misclassifications are indicated in the *last image*



(**b**) Results of our lazy texture selection method. The *second image* displays the initial classification after initial scribbles shown in the *first image*. After answering the machine's query by adding few scribbles (shown in the *third image*), we achieve a much improved result without local classifiers

Fig. 9 Comparison of interactive texture selection methods

8 Applications

An accurate texture classification makes it possible to do further editing in user selected regions. Figure 10 is the result of *color transfer*, a simple technique that changes the color of the selected texture. The user picks a few seeding points within the textured object, and our system clusters the rest of the pixels by finding the shortest L_2 distance to any of the seeding points in a color space. In a subsequent step, a color is chosen as the transfer target for each cluster, and the seeding point is taken as the reference. The difference between the target and the reference will be the transfer vector added to every pixel in the cluster. The transfer can be done in any color space specified by the user. In addition, we implemented a weighting scheme for pixels residing on cluster boundaries, i.e., interpolating colors of neighboring clusters, giving the transferred texture a softer and more natural look.

Compositing operations, for example, can be conveniently applied as shown in Fig. 11. To achieve this, a trimap is first constructed from the texture selection result by expanding a transition region from the texture boundary towards the interior of both foreground and background regions, followed by applying an algorithm, such as the one

in [8], to estimate the alpha channel of the transition region.

Texture cloning is another interesting application, in which we compute a neighborhood-based similarity between a seeding pixel and every other pixel within the selected region followed by a texture cloning procedure introduced in [6]. The idea is to blend a new texture with the original one according to opacity values obtained from the neighborhood-based similarity computation (Fig. 1).

9 Discussions and conclusions

In the making of our system, a few questions are open to discussion. For now, our system is yet to handle more intricate and ambiguous texture patterns such as the one shown in Fig. 12. Hence finding a more powerful classifier that works with a broader range of textures will be an interesting topic in future study. In our compositing experiments, existing matting techniques could not give satisfactory alpha estimates when applied to boundary areas with complicated textures such as zebra stripes. In future, we hope to develop and incorporate into our system a matting procedure that better handles textured boundary areas.

In conclusion, we have described a novel interactive approach to the problem of texture selection and editing across both spatial and temporal domains. By using a robust classifier in a supervised fashion, our algorithm is able to quickly distinguish the user-specified textures or textured objects. Meanwhile, it adopts the idea of active learning and puts the user in a more laid back position where only modest marking and labeling work is required. With its achievements in both solution accuracy and efficiency, our system is a convenient tool for texture selection and subsequent editing.

Acknowledgements Four dynamic textures used in this paper, FLOWER, SEA PLANT, SMOKE, and BUBBLES, are from the DynTex database at Center for Mathematics and Computer Science (CWI), The Netherlands. This work was partially supported by National Natural Science Foundation of China (60728204/F020404).

Fig. 10 Color transfer examples in user selected regions

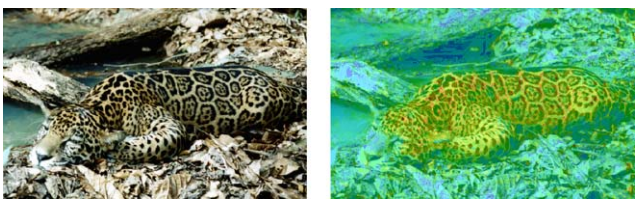


Fig. 12 Failure example. The *greenish* color in the signed confidence map on the *right* indicates that the classifier is uncertain about the classification of most of the pixels

Fig. 11 Compositing examples. *Texture regions* are first extracted as the foreground or background layer, the foreground layer is then composed with a different background using an estimated alpha channel within a transition region

References

1. Abe, N., Mamitsuka, H.: Query learning strategies using boosting and bagging. In: Fifteenth International Conference on Machine Learning, pp. 1–9 (1998)
2. Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., Cohen, M.: Interactive digital photomontage. ACM Trans. Graph. **23**(3), 294–302 (2004)
3. Avidan, S.: Spatialboost: Adding spatial reasoning to adaboost. In: Proceedings of European Conference Computer Vision (ECCV) (2006)
4. Bhat, K., Seitz, S., Hodgins, J., Khosla, P.: Flow-based video synthesis and editing. ACM Trans. Graph. **23**(3), 358–361 (2004)
5. Boykov, Y., Jolly, M.: Interactive graph cuts for optimal boundary and region segmentation of objects in N–D images. In: Int. Conf. Computer Vision, vol. I, pp. 105–112 (2001)
6. Brooks, S., Dodgson, N.: Self-similarity based texture editing. In: SIGGRAPH 2002 Proceedings, pp. 653–656 (2002)
7. Chang, T., Kuo, C.: Texture analysis and classification with tree-structured wavelet transform. IEEE Trans. Image Process. **2**, 429–441 (1993)
8. Chuang, Y.Y., Curless, B., Salesin, D., Szeliski, R.: A Bayesian approach to digital matting. In: Proceedings of IEEE Conf. Computer Vision and Pattern Recognition, pp. 264–271 (2001)
9. Chuang, Y.Y., Goldman, D., Zheng, K., Curless, B., Salesin, D., Szeliski, R.: Animating pictures with stochastic motion textures. ACM Trans. Graph. **24**(3), 853–860 (2005)
10. Dunn, D., Higgins, W., Wakeley, J.: Texture segmentation using 2-d Gabor elementary functions. IEEE Trans. Pattern Anal. Mach. Intell., **16** (1994)
11. Felzenszwalb, P., Huttenlocher, D.: Efficient graph-based image segmentation. Int. J. Comput. Vis. **59**(2), 167–181 (2004)
12. Gavish, L., Wolf, L., Shapira, L., Cohen-Or, D.: Principal-channels for one-sided object cutout. Tech. rep., Tel-Aviv University (2007)
13. Gleicher, M.: Image snapping. In: SIGGRAPH 95 Proceedings, pp. 183–190 (1995)
14. Hoiem, D., Efros, A., Hebert, M.: Automatic photo pop-up. ACM Trans. Graph. **24**(3), 577–584 (2005)
15. Irony, R., Cohen-Or, D., Lischinski, D.: Colorization by example. In: Eurographics Symposium on Rendering (2005)
16. Iyengar, V., Apte, C., Zhang, T.: Active learning using adaptive resampling. In: Sixth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 92–98 (2000)
17. Leung, T., Malik, J.: Representing and recognizing the visual appearance of materials using three-dimensional textons. Int. J. Comput. Vis. **43**(1), 29–44 (2001)
18. Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. ACM TOG **23**(3), 689–694 (2004)
19. Li, Y., Adelson, E., Agarwala, A.: ScribbleBoost: adding classification to edge-aware interpolation of local image and video adjustments. In: Proceedings of Eurographics Symposium on Rendering (2008)
20. Li, Y., Sun, J., Shum, H.Y.: Video object cut and paste. ACM Trans. Graph. **24**(3), 595–600 (2005)
21. Li, Y., Sun, J., Tang, C.K., Shum, H.Y.: Lazy snapping. ACM Trans. Graph. **23**(3), 303–308 (2004)
22. Luan, Q., Wen, F., Cohen-Or, D., Liang, L., Xu, Y., Shum, H.: Natural image colorization. In: Eurographics Symposium on Rendering (2007)
23. Malik, J., Belongie, S., Leung, T., Shi, J.: Contour and texture analysis for image segmentation. Int. J. Comput. Vis. **43**(1), 7–27 (2001)
24. Manjunath, B., Ma, W.: Texture features for browsing and retrieval of image data. IEEE Trans. Pattern Anal. Mach. Intell. **18**(8), 837–842 (1996)
25. Martin, D., Fowlkes, C., Malik, J.: Learning to detect natural image boundaries using brightness and texture. In: Neural Information Processing Systems (NIPS) (2002)
26. Mortensen, E., Barrett, W.: Intelligent scissors for image composition. In: SIGGRAPH 95 Proceedings, pp. 191–198 (1995)
27. Protiere, A., Sapiro, G.: Interactive image segmentation via adaptive weighted distances. IEEE Trans. Image Process. **16**(4), 1046–1057 (2007)
28. Qu, Y., Wong, T.T., Heng, P.A.: Manga colorization. ACM TOG **25**(3), 1214–1220 (2006)
29. Rother, C., Blake, A., Kolmogorov, V.: Grabcut–interactive foreground extraction using iterated graph cuts. ACM Trans. Graph. **23**(3), 309–314 (2004)
30. Schapire, R.: The boosting approach to machine learning: an overview. In: MSRI Workshop on Nonlinear Estimation and Classification (2002)
31. Seung, H., Opper, M., Sompolinsky, H.: Query by committee. In: 5th Annual Workshop on Comput. Learning Theory, pp. 287–294 (1992)
32. Shotton, J., Winn, J., Rother, C., Criminisi, A.: Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In: Proceedings of European Conference Computer Vision (ECCV) (2006)
33. Wang, J.: Discriminative Gaussian mixtures for interactive image segmentation. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2007, pp. 601–604 (2007)
34. Wang, J., Bhat, P., Colburn, A., Agrawala, M., Cohen, M.: Interactive video cutout. ACM Trans. Graph. **24**(3), 585–594 (2005)

Tian Xia received the BE degree in computer science from Zhejiang University, China, in 2004 and the MS degree in computer science from the University of Illinois, Urbana-Champaign, in 2006. He is currently a PhD candidate in computer science at the University of Illinois. His research interests include computer graphics and vision.

Qing Wu received the BE and ME degrees in computer science from Tsinghua University, China, in 1999 and 2001, respectively. He received the PhD degree in computer science from the University of Illinois, Urbana-Champaign in 2007. He is now working with Google.

Chun Chen received the MS and PhD degrees in computer science from Zhejiang University, China, in 1984 and 1990, respectively, and the BS degree in mathematics from Xiamen University, China, in 1981. He is currently a professor in the college of computer science at Zhejiang University. From 1996 to 1997, he was a visiting scholar in the department of computer science at the University of Calgary. He has authored or co-authored 3 books and more than 100 research papers. His research interests include computer vision, image processing, and embedded systems.

Yizhou Yu received the BS degree in computer science and the MS degree in applied mathematics from Zhejiang University, China, in 1992 and 1994, respectively, and the PhD degree in computer science from the University of California, Berkeley, in 2000. He is currently an associate professor in the Department of Computer Science, University of Illinois, Urbana-Champaign. He has developed techniques in the areas of computer graphics and computer vision, including mesh editing based on differential coordinates, inverse global illumination, shape from shadow, feature-based texture synthesis, and controllable fluid simulation.