Image Repairing: Robust Image Synthesis by Adaptive ND Tensor Voting*

Jiaya Jia and Chi-Keung Tang
Vision and Graphics Group, Computer Science Department
Hong Kong University of Science and Technology
{leojia,cktang}@cs.ust.hk

Abstract

We present a robust image synthesis method to automatically infer missing information from a damaged 2D image by tensor voting. Our method translates image color and texture information into an adaptive ND tensor, followed by a voting process that infers non-iteratively the optimal color values in the ND texture space for each defective pixel. ND tensor voting can be applied to images consisting of roughly homogeneous and periodic textures (e.g. a brick wall), as well as difficult images of natural scenes which contain complex color and texture information. To effectively tackle the latter type of difficult images, a two-step method is proposed. First, we perform texture-based segmentation in the input image, and extrapolate partitioning curves to generate a complete segmentation for the image. Then, missing colors are synthesized using ND tensor voting. Automatic tensor scale analysis is used to adapt to different feature scales inherent in the input. We demonstrate the effectiveness of our approach using a difficult set of real images.

1 Introduction

Today, powerful photo-editing softwares and large varieties of retouching, painting, and drawing tools are available to assist users to refine or redefine images manually. For instance, we can easily outline and remove a large foreground object by intelligent scissors [12] or other related techniques. However, filling the background hole seamlessly from existing neighborhood information of damaged images is still a difficult problem. A skilled art designer repairs them mostly through his experience and knowledge. Alternatively, some other stereo algorithms [9, 16] use a dense set of images to infer the color of occluded pixels.

Given as few as one image and no additional knowledge, is it possible to automatically repair it? The main issues to be addressed are as follows:

- How much color and shape information in the existing part is needed to seamlessly fill the hole?
- How good can we achieve in order to reduce possible visual artifact when the information available is not sufficient?

Inpainting [3, 1] is an automatic method to fill small holes by exploiting color and gradient information in the neighborhood of the holes. It successfully removes contaminant and restores images with little visual artifact. However, it does not take texture information into account which may introduce blurring in a textured region.

In this paper, we propose a unified approach to gather and analyze statistical information so as to synthesize proper pixel values in image holes. Instead of simply extending neighboring color information, we use the robust, non-iterative ND tensor voting [11] to globally infer the most suitable pixel value in the neighborhood by using the Markov Random Field (MRF) assumption. A texture-based segmentation [4] is adopted to partition images into different region segments. 2D curve connections are voted for, in order to complete the segmentation in image holes. Missing colors are synthesized by ND tensor voting, which adapts to different feature and texture scales.

The rest of this paper is organized as follows: we discuss and compare related work in section 2. In section 3, we review the tensor voting algorithm. Section 4 provides an overview of our approach. In section 5, we describe our image segmentation and curve connection algorithms. Section 6 introduces ND tensor voting and its application in image synthesis. Section 7 presents our results. Finally, we conclude our paper in section 8.

2 Previous work

The contribution of our work is the robust synthesis of missing image or texture information despite insufficient samples. To extrapolate details, the Markov Random Fields (MRF) have been used widely to describe textures [2, 6, 17], including this work. Texture synthesis techniques can be roughly categorized into three classes. The first class uses a parametric model to describe statistical textures. Simoncelli et al. [15] use joint wavelet coefficients to parameterize and synthesize textures. Heeger et al. [7] makes use of Laplacians, steerable pyramids, and histogram analysis of a texture sample to synthesize textures. While impressive results are obtained for highly stochastic textures, these methods are not well suited to represent highly structured textures such as that of a brick wall.

The second class consists of non-parametric sampling and synthesis techniques. Efros and Leung [6] synthesize textures by copying pixels that are matched with the neighborhood of a pixel being synthesized. It works well to reproduce structured textures.

The third class is to synthesize textures by procedural means. Solid textures, proposed independently by Perlin [14]

*This work is supported by the Research Grant Council of Hong Kong Special Administration Region, China: HKUST6193/02E.

and Peachey [13], involve a function that returns a color value at any given 3D point. Reaction-diffusion techniques [18] build up spot and stripe patterns which can be used to synthesize textures.

To accelerate the synthesis process and avoid pixel-wise generation, Xu et al. [19] propose a block-wise synthesis algorithm to seamlessly copy sample patches in an overlapping manner to create a new image. Image quilting [5] extends this algorithm by a minimum error boundary cut so that patches naturally connect to each other. Wei et al. [17] describe another hierarchical texture synthesis algorithm to speed up the per-pixel generation process by matching the neighborhood of lower resolution image pixels with the synthesized ones, and applying TSVQ acceleration. Inpainting [3, 1] is an algorithm to restore damaged 2D images. Instead of performing matching and copying, the basic idea is to smoothly propagate information from the surrounding areas in the isophotes direction. This algorithm generates satisfactory result when holes are small, and the gradient information in holes does not change abruptly. However, it does not take texture information into consideration, resulting loses in precision and obvious artifact in large holes.

3 Review of Tensor Voting

In our method, *Tensor Voting* [11] is used to infer missing curves and pixel values. It makes use of a *tensor* for token representation, and *voting* for communication among tokens. Tensor and voting are brought together by a voting field, which is a dense tensor field for postulating smooth connections among tokens. In this section, we first give a concise review of the stick tensor and the stick voting field, which are used in the following sections.

3.1 Token representation and communication

We are interested in answering the following geometric question in 2D, which can be generalized to higher dimension readily. Suppose there exists a smooth curve connecting the origin O and a point P. Suppose also that the normal $\vec{N}$ to the curve at O is known. What is the most likely normal direction at P? Fig. 1a illustrates the situation. We claim that the osculating circle connecting O and P is the most likely connection, since it keeps the curvature constant along the hypothesized circular arc. The most likely normal is given by the normal to the circular arc at P (thick arrow in Fig. 1a). This normal at P is oriented such that its inner product with $\vec{N}$ is non-negative. The length of this normal, which represents the vote strength, is inversely proportional to the arc length OP, and also to the curvature of the underlying circular arc. The decay of the field takes the following form:

$$\overline{DF}(r, \varphi, \sigma) = e^{-(\frac{r^2+c\varphi^2}{\sigma^2})} \quad (1)$$

where r is the arc length OP, φ is the curvature, c is a constant which controls the decay with high curvature, and σ controls smoothness, which also determines the effective neighborhood size. If we consider all possible locations of P with non-negative x coordinates in the 2D space, the resulting set of normal directions thus produced constitutes the 2D stick voting field, Fig. 1b.

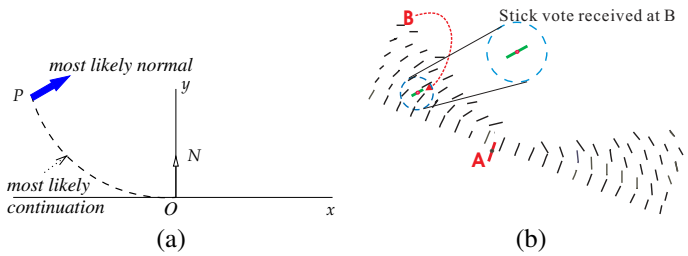

Figure 1: (a) Design of the 2D stick voting field. (b) A casts a stick vote to B, using the 2D stick voting field.

Given an input token A, how can it cast a stick vote to another token B for inferring a smooth connection, assuming that A's normal is known? It is illustrated in Fig. 1b. First, we fix σ to determine the size of the voting field. Then, we align the voting field with A's normal, simply by translation and rotation. If B is within A's voting field neighborhood, B receives a stick vote $[v_x\ v_y]^T$ from the aligned field. Hence, voting is similar to convolution, and the voting field is like a convolution mask, except that the voting result is not a scalar.

Other input tokens cast votes to B as well. Second order tensor sums of all votes received at B are collected into a covariance matrix $\mathbf{S} = \begin{bmatrix} \sum v_x^2 & \sum v_x v_y \\ \sum v_y v_x & \sum v_y^2 \end{bmatrix}$. The corresponding eigensystem consists of two eigenvalues $\lambda_1 \geq \lambda_2 \geq 0$, and two corresponding eigenvectors $\hat{e}_1$ and $\hat{e}_2$. Therefore, $\mathbf{S}$ can be rewritten as $\mathbf{S} = (\lambda_1 - \lambda_2)\hat{e}_1\hat{e}_1^T + \lambda_2(\hat{e}_1\hat{e}_1^T + \hat{e}_2\hat{e}_2^T)$. $\hat{e}_1\hat{e}_1^T$ is a stick tensor, with $\hat{e}_1$ indicating curve normal direction. $\hat{e}_1\hat{e}_1^T + \hat{e}_2\hat{e}_2^T$ is a ball tensor.

3.2 Feature extraction

When we have obtained normal directions at each input site, we can infer a smooth structure that connects the tokens with high feature saliencies (in 2D, curve saliency is represented by $(\lambda_1 - \lambda_2)$[11]). Feature extraction can be achieved by casting votes to all sites (input and non-input sites), using the same voting fields and voting algorithm. Given this dense information, in 2D, we extract true curve points and connect them.

4 Algorithm Framework

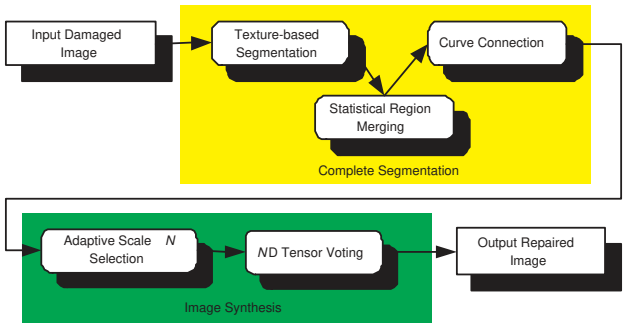

Figure 2: Overview of our image synthesis algorithm.

In this section, we provide a brief outline of our algorithm. Figure 2 gives the overview of the whole process, which uses

a single damaged image as input. To robustly generate pixel colors, a two-phase process is performed (highlighted in the figure). We call the two phases *complete segmentation* and *color synthesis*, respectively.

Complete segmentation Usually, images reveal a cluttered scene with indistinct edge features, leading to a complex image structure. In order not to mix up different information, we perform texture-based segmentation in the damaged image, followed by extrapolating the resulting partitioning curves into an image hole. To this end, 2D tensor voting is used to infer the most likely connection between two curve endpoints. A complete segmentation for the damaged image is achieved.

Color synthesis Let us call a pixel in an image hole (including the one lying on the hole boundary) a *defective pixel*. When an image hole has been segmented, we synthesize the color value of a defective pixel using only existing pixels belonging to the same group as shown in Fig. 3. In other words, we qualify the MRF constraint by complete segmentation. Adaptive scale adjustment further eliminates visual artifact.

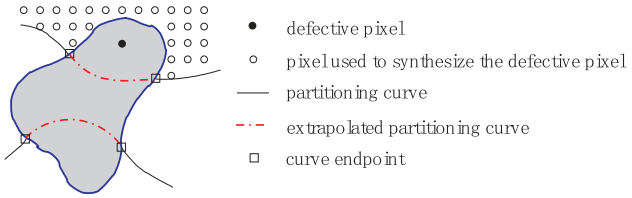

Figure 3: Color synthesis constrained by complete segmentation.

5 Complete segmentation

We adopt the unsupervised segmentation algorithm described in [4] to perform the pattern segmentation. Initially, we assign defective pixels in an image hole to the same segment P_0. Then, an image is constructed to perform seed growing and region merging. In the process, we can assign merge threshold for small regions so as to generate partitions with finer details.

5.1 Region merging

In real images, complex occlusion and object discontinuities are typical. For instance, the building in the flower garden is occluded and partitioned into several parts by the foreground tree as in Fig. 12a. In Fig. 11, the silhouette of the tree is not a smooth blob, but contains many curve orientation discontinuities. We perform region merging to group similar objects together. The grouping is performed in color and gradient space.

To measure color and pattern feature of different regions, the statistics of zero and first order image intensity derivatives are considered. We construct a $(M+1)$D intensity vector V_i^{M+1} for each region P_i $(i \neq 0)$ where M is the maximum color depth in the whole image. The first M components in the intensity vector, i.e., $V_i(1)$ to $V_i(M)$, represent the histogram of region i. The last component is defined as

$$V_i(M+1) = \frac{\alpha}{N_i} \sum_{j \in P_i} \| \nabla j \| \tag{2}$$

where N_i is the number of points in region P_i, $\| \nabla j \|$ is the intensity gradient magnitude of point j. α can be thought as a controlling weight to adjust the significance of the gradient information for measuring intensity similarity. The larger α is, the more important intensity gradient becomes. In our experiments, α is set to normalize $V_i(M+1)$ so that the largest gradient magnitude equals to the color depth, and the smallest equals to zero.

Therefore, we merge two similar regions P_i and P_k as P, or $P = P_i \cup P_k$, if

$$s_{i,k} = \| V_i - V_k \| \leq Threshold \tag{3}$$

where $s_{i,k}$ is the similarity score for the region pair, V_i and V_k are intensity vectors for P_i and P_k respectively. Hence, we match not only the color information but also some nearest neighborhood statistics, i.e., pattern information, to obtain a more reliable measurement.

5.2 Curve connection

After merging regions in the damaged image, we want to extrapolate the partitioning curves into the holes to complete our segmentation, by inferring the most likely connection between curve endpoints. Normally, we do not have any additional knowledge of the missing data. The only available constraint for extrapolating boundary curves is to maintain existing geometrical features, e.g., curvature and curve shape. In order not to over-smooth the synthesized curves, and to preserve the shape of the segmented region, the robust tensor voting algorithm is used. Tensor voting votes for the missing curve elements in image holes, by gathering tensorial support in the neighborhood.

5.2.1 Tensor voting

Fig. 4 shows a simplified hole filling example. First, we place *imaginary points* (the green or lightly shaded dots in Fig. 4) evenly distributed in the hole (the rectangle in Fig. 4). Denote the sampling density by S. Our goal is to identify true curve points from the set of imaginary points in the hole. First, we infer normal directions on the existing curve points by tensor voting [11] (blue or black arrow on the curve, in Fig. 4). These points with normal directions are then encoded into 2D stick tensors, which cast votes to each imaginary point.

After receiving and accumulating the collected votes (section 3.1), each imaginary point obtains a curve saliency value, $\lambda_1 - \lambda_2$ (section 3.2). Assuming that the curve has no intersection, and that it is monotonic along the x-direction (otherwise, the hole can be split to enforce this monotonicity requirement), only one point P_{x_i,z_j} is selected as the curve point for each *anchor* x_i (Fig. 4). The larger the curve saliency the point receives, the more likely the point is actually lying on the curve. For each anchor x_i, the curve saliencies among all

imaginary points are compared. The point P_{x_i,z_j} having the highest curve saliency is selected as the optimal curve point P_{x_i} at anchor x_i. The process can be formulated as follows:

$$P_{x_i} = \max\{P_{x_i,z_j}(\lambda_1 - \lambda_2)\} \quad 1 \leq j \leq S \qquad (4)$$

where x_i is an anchor and S is the sampling density (in pixels). Note that curve normals at P_{x_i}'s have been computed at the same time after tensor voting is run.

Once the discrete curve points are synthesized, we connect them together using a B-spline. In case of a large hole, we perform curve connection at different scales, by warping and upsampling the curve points obtained at successive scales.

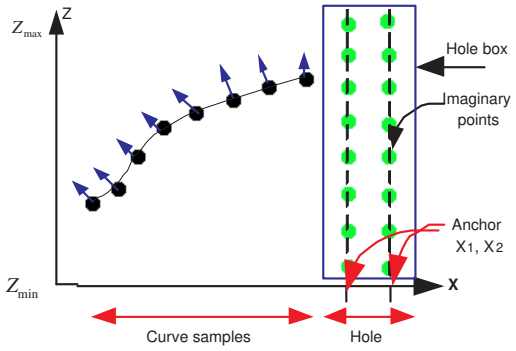

Figure 4: Hole filling by hypothesizing depth values.

Since the voting algorithm is dominated by the design of the voting field, we give a qualitative analysis of the correctness of the above method. Consider a simple situation that two tokens are on a straight line, with three imaginary points: above, on, and below the line respectively (Fig. 5b). Since $|\overline{P_1N_1}| > |\overline{P_2N_1}|$ and $|\overline{P_1N_2}| > |\overline{P_2N_2}|$, the received vote strength of P_2 is stronger than that of P_1. Hence, P_2 has the largest curve saliency and the line segment is extended naturally. If N_1 and N_2 are on a curve, the analysis is similar (Fig. 5b), and P_3 receives the largest saliency, due to the sign of the curvature along the existing curve.

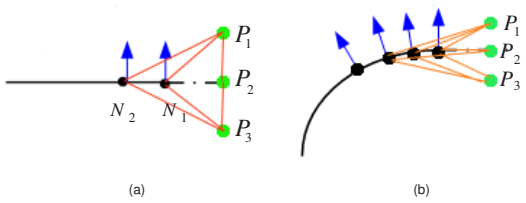

Figure 5: An analysis of 1D hole filling

There are two advantages to apply discrete tensor voting to synthesize optimal curve point. One is that the σ parameter of the voting field can be used to control the smoothness of the resulting curve. Fig. 6 shows one curve connection example. The larger the scale is, the smoother the curve ab is. Thus we can easily set the appropriate σ to avoid over-smoothing.

The other advantage is related to the fact that different shapes of image holes constrain different sets of the imaginary

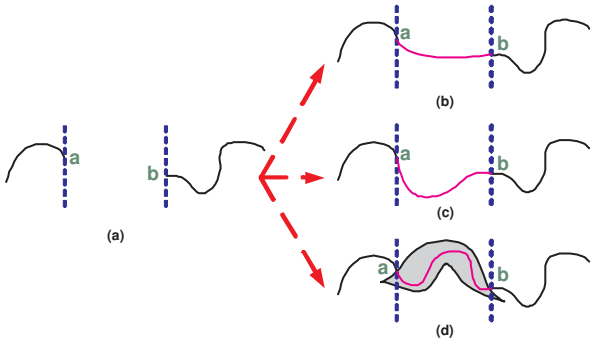

Figure 6: Hole filling examples. (a) The broken curve. (b) Curve connection result with a large σ. (c) Result obtained by using a smaller σ. (d) Curve result, constrained by the shape of the hole.

points. Through the same voting process, we can synthesize various optimal curves accordingly. One example is shown in Fig. 6d. The shaded region represents an image hole. The optimal curve ab synthesized is different from that in Fig. 6b and c without the hole constraint, but it is still a natural connection with respect to the shape of the hole.

5.2.2 Connection sequence

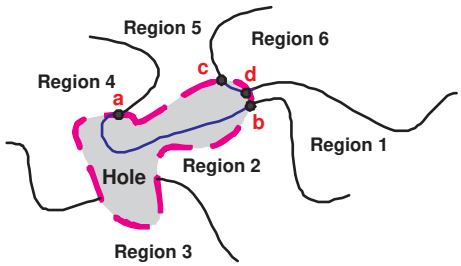

Figure 7: A hole connection example.

We facilitate the discussion in the previous section by using an unambiguous case where only two salient curve endpoints are present. In practice, we have the configuration similar to Fig. 7. To complete the segmentation, we connect all similar but disjoint regions surrounding the hole by inferring the most likely smooth connection curves.

To reduce the instabilities and avoid ambiguities, we propose the following connection scheme, implemented as a greedy algorithm:

1. Find all disjoint region pairs around an image hole which belong to the same merged segment (section 5.1). Sort these pairs by similarity scores in descending order, and put them into a queue. For instance, the three elements in the queue from head to tail in Fig. 7 are Regions 1 and 5, 2 and 4, 3 and 6. It means that Regions 1 and 5 are the most similar.

2. If the queue is not empty, fetch the region pair at the queue head:

- If the two regions are already connected, skip them and back to step 2 (Regions 1 and 5 in Fig. 7).
- If there exists a possible boundary connection which does not intersect any existing curves, infer the connection curve (e.g., Regions 2 and 4 in Fig. 7). Else, skip this region pair and go back to step 2 (e.g. the candidate connection of Regions 3 and 6 in Fig. 7 have to intersect the existing curves ab and cd).

3. If there exists a single region unclosed, infer new curves to close it (by constraining the shape of the hole).

6 Image synthesis

Once the image is completely segmented, we synthesize missing data with existing color and texture information. Since color is a local property at a pixel, and by MRF, texture is defined by a neighborhood, we propose to use ND tensor voting to infer color and texture information, where N indicates neighborhood scale.

6.1 ND tensor representation

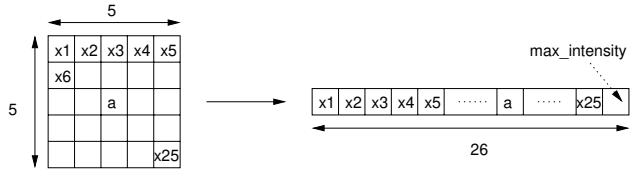

Figure 8: Encoding a sub-image centered at a by a ND vector (N=26 here).

Given a neighborhood window of size $n \times n$ centered at a pixel A, we can translate the sub-image into a stick tensor (section 3.1), by producing a feature vector of dimension $N = n \times n + 1$ (Fig. 8) in a lexicographical ordering. Converted grey levels are used if the input is a color image. Thus, a feature vector is represented by homogeneous coordinates, so that zero intensity can be dealt with uniformly (max_intensity $= M$, the maximum color depth).

Suppose we have two pixels $A = (x_a, y_a)$ and $B = (x_b, y_b)$ with their respective ND feature vectors $\vec{\mathcal{T}}_A$ and $\vec{\mathcal{T}}_B$, which are encoded as described in Fig. 8. Suppose also that we want to synthesize color at B. Therefore, some components in $\vec{\mathcal{T}}_B$ may not have color information. We zero out corresponding components in both vectors before performing matching. We denote the modified vectors by $\vec{\mathcal{T}}_{\mathbf{A}}$ and $\vec{\mathcal{T}}_{\mathbf{B}}$ respectively, and name A as *sample seed*. To avoid too many zero components in vectors, the sample seeds must satisfy the criterion that the number of zero components is less than $\frac{n}{2}$.

The matching between $\vec{\mathcal{T}}_{\mathbf{A}}$ and $\vec{\mathcal{T}}_{\mathbf{B}}$ is translated into tensor voting for a *straight line in the ND space*[1]. First, we need to perform the following ND encoding for A and B into A_N and B_N respectively:

1. Convert A into ND coordinates, denoted by A_N. Without losing generality, we can choose $A_N = (\underbrace{0, \cdots, 0}_{N})$.

2. Choose any $\vec{D}$ such that $||\vec{D}|| \neq 0$ and $\vec{\mathcal{T}}_{\mathbf{A}} \cdot \vec{D} = 0$, that is, $\vec{\mathcal{T}}_{\mathbf{A}}$ and $\vec{D}$ are perpendicular to each other. Then, convert B into ND coordinates, by

$$B_N = A_N + \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \frac{\vec{D}}{||\vec{D}||} \qquad (5)$$

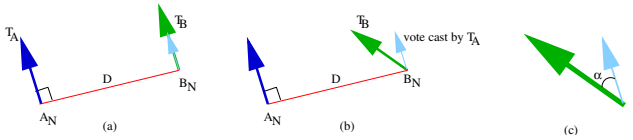

Figure 9: Vote for colors. (a) The ND vectors at A_N and B_N are consistent, indicating that they are lying on the same ND straight line. (b) and (c) Inconsistent normals are indicated by vote inconsistency.

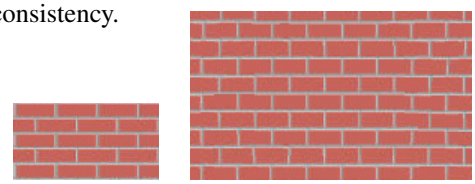

Figure 10: Result on synthesizing structured textures. Left: input texture sample. Right: synthesized texture image.

Therefore, $\vec{\mathcal{T}}_{\mathbf{A}}$ is the normal to the ND straight line connecting A_N and B_N. Now, A_N casts a stick vote to B_N in exactly the same way as described in section 3, except that now a ND stick voting field[2] is used: in ND, an osculating circle becomes an osculating hypersphere. We can define a ND stick voting field by uniform sampling of normal directions in the ND space. The construction is exactly the same as the 2D stick voting field, but now in ND.

When B_N receives a stick vote from A_N, the vote will be matched with $\vec{\mathcal{T}}_{\mathbf{B}}$. Vote consistency is indicated by $s\alpha$, where $s = \lambda_1 - \lambda_2$ is the vote saliency given by the ND stick voting field, and α is the cosine of the angle between $\vec{\mathcal{T}}_{\mathbf{B}}$ and the received stick vote.

The total number of sample seeds which cast votes to B_N depends on the complete segmentation result: the region size of the segment to which B_N belongs. Among all sample seeds, let $\mathcal{A}$ be the $2D$ pixel corresponding to the A_N whose vote to B_N gives the maximum $s\alpha$ at B_N. To synthesize the color at B, we replace the zero components in $\vec{\mathcal{T}}_{\mathbf{B}}$ by corresponding non-zero entries in $\vec{\mathcal{T}}_{\mathcal{A}}$. In practice, not all zero components are replaced. Typically, only zero entries in a small window centered at B are replaced.

The scale N, as depicted in Fig. 8, is the crucial parameter in our method. Its value depends on how complex the neighborhood structure is. If the scale is too large, the synthesis process is slow, which also invalidates the MRF assumption. If the scale is too small, it is inadequate to capture the necessary data. Hence, we propose an automatic and adaptive scale selection method to determine the value of N.

[1] This line defines a family of hyperplanes which are voted for by ND tensor voting.

[2] In implementation, we need not use an ND array to store the ND stick voting field due to its symmetry.

6.2 Adaptive scaling

Normally, texture inhomogeneity in images gives difficulty to assign only one global scale N. In other words, all sample seeds have their own smallest sizes that best capture their neighborhood information. The scale N_i for different region i should vary across the whole image in voting process, as described in section 6.1.

We observe that human eyes are more sensitive to edge discontinuity than to pure color distinctness when synthesis artifact exists. Accordingly, to select an appropriate scale for a sample seed, we compute its edge complexity by accumulating gradients ∇I within its neighborhood window. Simply summing them up will cancel opposite ones. Hence the second order moment matrix for the vectors (tensor) within the window are used [8, 10]:

$$M_\sigma(x,y) = G_\sigma(x,y)((\nabla I)(\nabla I)^T) \tag{6}$$

where G_σ denotes an Gaussian smoothing kernel with variance σ^2 centered at a pixel (x,y). Since the tensor encoding process (section 6.1) treats the window center and the boundary points equally, we set $\sigma = 0$ to make the Gaussian decay an average function to simplify the notation and computation:

$$M_N = AVG_N\{((\nabla I)(\nabla I)^T)\} \tag{7}$$

$$= \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix} \tag{8}$$

where M_N is a function of scale N. $trace(M_N) = q_{11} + q_{22}$ measures the average strength of the square of the gradient magnitude in the window of size N [10]. By observation, inhomogeneity usually results in abrupt change in gradient magnitude. Therefore, we select the scale for each sample seed to be proportional to the local maxima threshold of M_N, as the value of N increases. It gives good estimation in our examples. The detailed scheme is as follows: for each sample seed i, increase its scale N_i from the lower bound (usually set to 3) to the upper bound (depending on the image resolution). For each scale N_i, compute $trace(M_{N_i})$. If $trace(M_{N_i}) < trace(M_{N_i-1}) - \alpha$ where α is a threshold to avoid small perturbation or noise interference, set $N_i - 1 \rightarrow N_i$ and return (i.e., local maxima threshold is reached). Otherwise, we continue the loop by incrementing N_i until a maxima is found, or the upper bound has been reached.

7 Results and limitations

We first show one texture synthesis result on regular texture patterns (Fig. 10 on the previous page). The emphasis of this paper is repairing damaged images of real scenes. We have experimented a variety of such natural images, most of them contain large and complex holes with difficult neighborhood topologies. Fig. 11 on the next page shows two examples. From a single damaged image (middle) with a large amount of missing complex information, we are capable of synthesizing new pixels. By adaptively propagating neighborhood information, our method smoothly generates texture patterns without blurring important feature curves (right). The left images are original ones with the occluding objects. There exists many methods to erase these objects, which are out of the scope of this paper. The original images are provided for comparison.

From a single image of flower garden with the occluding tree removed, we synthesize the result in Fig. 12f. The detailed process is: we first segment the damaged image. It is followed by a merging process described in section 5.1 (Fig. 12d). Then, we complete our segmentation by curve connection using tensor voting described in section 5.2 (Fig. 12e). Fig. 13 shows another result, obtained using the same process. Some intermediate results are also shown.

Fig. 14 compares our method with image inpainting [1], on a relatively large hole. Since our method considers pattern information, it retains texture structure and does not introduce blurring. Another direct comparison is shown in Fig. 12c, which shows that texture synthesis technique such as [6] is not suitable in our task of repairing natural images. Without a proper segmentation, pixels belonging to distinct statistical distributions are mixed together.

(a) (b)

Figure 15: Limitations of our method. Pixels inside the yellow or shaded rectangle cannot be well synthesized by our method. (a) Handrail and stairs. (b) Bird beak.

The running time of our method depends on the image texture and hole complexity. In the experiments, given a 400×300 pixels image and $10,000$ pixels hole area, the method generates output of repaired images in less than 15 minutes on a Pentium III 1GHz PC.

Fig. 15 shows some of the limitations of our method if critical information is missing or insufficient. One example consists of various shapes in Fig. 15a, in which the missing portion of the handrail and stairs cannot be well synthesized due to a complex background and their irregular shapes. The other example is that of an entirely missing bird beak in Fig. 15b. In both cases, additional knowledge on the scene is needed.

8 Conclusion

We have proposed a new method to automatically repair damaged images, which is capable of dealing with large holes, where missing details can be complex and inhomogeneous, and cannot be described by a handful set of statistical parameters. Pure texture synthesis technique will fail on those input images. Real images of natural scenes are typical examples. We address this difficult problem by complete segmentation, robust curve connection, and the use of adaptive scales. ND tensor voting provides a unified basis to implement many of these tasks. We have demonstrated very encouraging results on natural images using our method, and performed some comparison. Our future work focuses on generalizing this image repairing technique to $2\frac{1}{2}$D and 3D data.

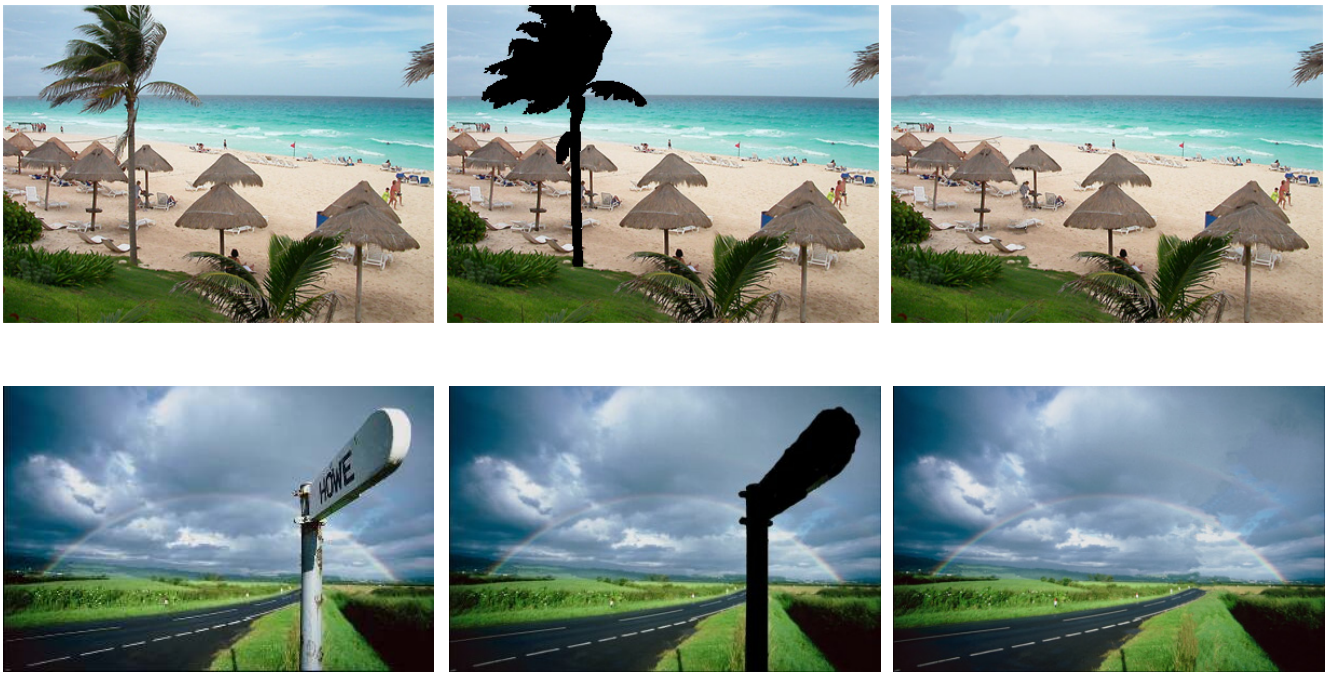

Figure 11: Beach and Moor. Left: original images. Middle: damaged images by erasing some objects. Right: our image synthesis results. Note that we can repair the full rainbow as well as the upper and very faint and rainbow "arc," which is largely occluded by the "HOWE" sign post.

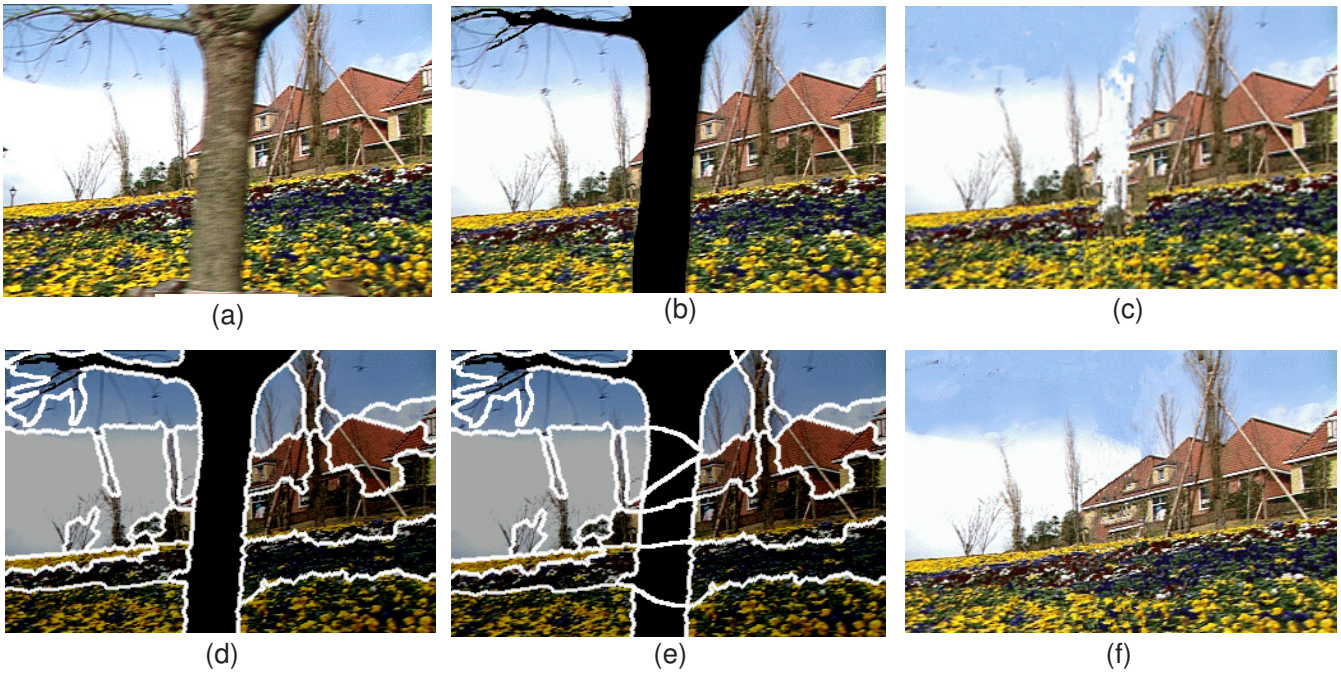

Figure 12: Flower garden example. (a) Only a single image is used. (b) Damaged image by erasing the tree. (c) Result obtained by texture synthesis algorithm [6]. (d) Image segmentation. (e) Curve connection. (f) Our image synthesis result.

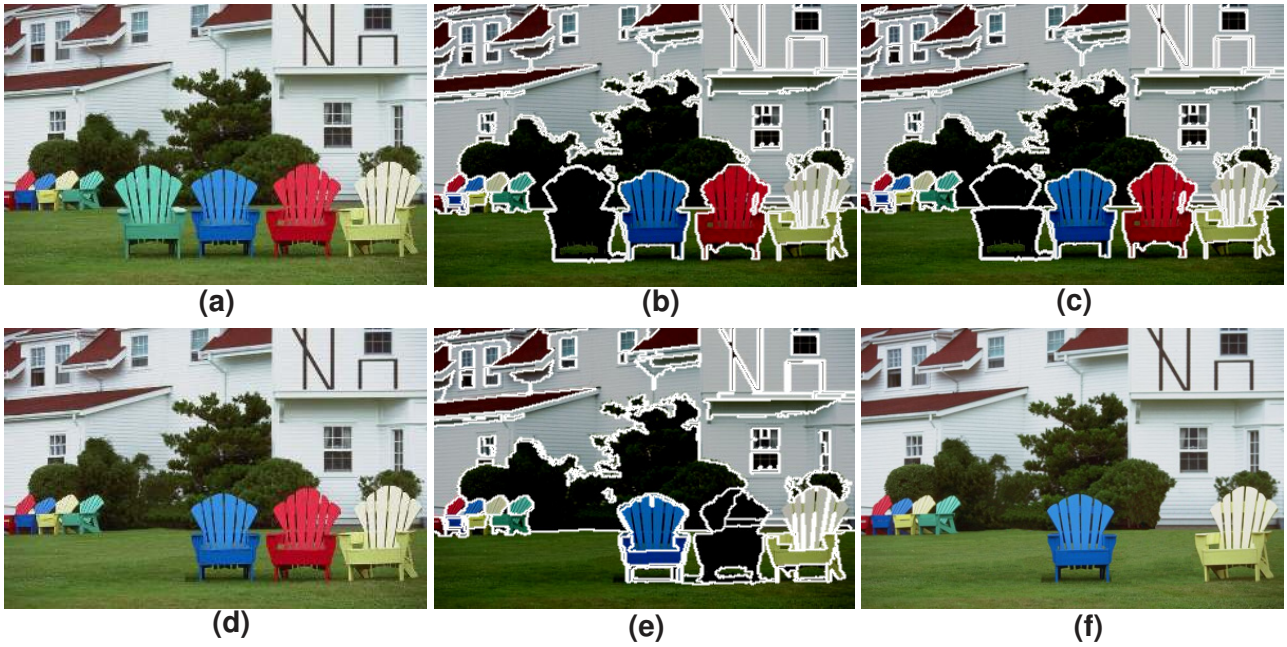

Figure 13: Chair example. (a) Original image. (b) Image segmentation before curve connection, after removing the green chair. (c) Complete segmentation after curve connection. (d) The repaired image. (e) Complete segmentation after removing the red chair. (f) Our result with two chairs left.

Figure 14: Comparison with image inpainting. Left: original image. Middle: image inpainting result by removing the microphone. Right: our result.

References

[1] M. Bertalmio, G. Sapiro, C. Ballester and V. Caselles, "Image inpainting.", *ACM Siggraph 2000*, pp. 417–424 (2000).

[2] J.S.De Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images.", *ACM Siggraph '97*, pp. 361–368 (1997).

[3] T. Chan, J. Shen, "Non-texture inpaintings by curvature-driven diffusions", Technical Report 00-35, Department of Mathematics, UCLA, Los Angeles, (2000).

[4] Yining Deng and B. S. Manjunath, "Unsupervised Segmentation of Color-Texture Regions in Images and Video.", *IEEE TPAMI*, **8**(23), pp. 800–810 (2001).

[5] Alexei A. Efros and Willian T. Freeman, "Image Quilting for Texture Synthesis and Transfer.", *ACM Siggraph '2001*, (2001).

[6] A. Efros and T.K. Leung, "Texture synthesis by non-parametric sampling.", *Proceedings of the Seventh ICCV*, pp. 1033–1038 (1999).

[7] David J. Heeger and James R. Bergen, "Pyramid-based texture analysis/synthesis". *ACM Siggraph '2001*, pp. 229–238 (1995).

[8] J. Bigün, "Local symmetry features in image processing." PhD thesis, Linkoping University, (1988).

[9] Sing Bing Kang, Richard Szeliski and Jinxiang Chai, "Handling Occlusion in Dense Multi-View Stereo", *IEEE CVPR'2001*, (2001).

[10] Jonas Gårding and Tony Lindeberg, "Direct computation of shape cues using scale-adapted spatial derivative operators", *IJCV*, vol. 17(2), pp. 163–191, (1996).

[11] G. Medioni, M.-S. Lee, and C.-K. Tang, *A Computational Framework for Feature Extraction and Segmentation*. Elsevier (2000).

[12] E. N. Mortensen, W. A. Barrett , "Intelligent Scissors for Image Composition.", *ACM Siggraph'95*, pp. 191–198, (1995).

[13] Darwyn R. Peachey, "Solid texturing of complex surfaces.", *ACM Siggragh'85* pp. 279–286, (1985).

[14] Ken Perlin, "An image synthesizer.", *ACM Siggraph'85* pp. 287–296, (1985)

[15] Javier Portilla and Eero P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients.", *IJCV* 40(1), pp. 49–71 (2000).

[16] H.-Y. Shum and L.-W.He, "Rendering with concentric mosaics.", *ACM Siggraph'99*, pp. 299-306 (1999).

[17] Li-Yi Wei and Marc Levoy, "Fast Texture Synthesis Using Tree-Structured Vector Quantization.", *ACM Siggraph'2000*, pp. 479–488 (2000).

[18] Andrew Witkin and Michael Kass, "Reaction-diffusion textures.", *ACM Siggraph'91*, pp. 299–308 (1991).

[19] Y. Q. Xu, S. C. Zhu, B. N. Guo, and H. Y. Shum, "Asymptotically Admissible Texture Synthesis", *Proc. of 2nd Int'l Workshop on Statistical and Computational Theories of Vision*, (2001).