Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition Letters 25 (2004) 619–629
Pattern Recognition Letters
www.elsevier.com/locate/patrec

# Clustering and blending for texture synthesis

Jasvinder Singh, Kristin J. Dana *

*Department of Electrical and Computer Engineering, Rutgers University, 94 Brett Road, Piscataway, NJ 08854, USA*

Received 21 August 2003; received in revised form 17 November 2003

**Abstract**

We consider texture modeling techniques which are exemplar-based, i.e. techniques which use several example texture images to learn pixel arrangements. The motivation for these techniques is that the structure of a texture can be characterized by the spatial distribution of pixels in a neighborhood of the image or the multiresolution pyramid. To get another instance of the same type of texture, one can rearrange the pixels as long as certain spatial neighbor relationships are enforced. In this work, we investigate two components of exemplar-based modeling: (1) grouping examples for computational savings during analysis and (2) blending for artifact removal during synthesis. First, we employ clustering in order to group example features and this method provides a significant computational savings without compromising the quality of the texture characterization. Second, we implement techniques for blending to remove border artifacts during the placement stage of texture synthesis. We show that for many textures, the pixel rearrangements can be done without filtering or neighborhood constraints, as long as the blending of borders is done well. Specifically, random rearrangement of texture patches generates a new texture instance of the same texture type when artifacts are removed with blending. We investigate existing blending approaches and introduce a blending method based on image fusion.


*Keywords:* Texture recognition; Texture synthesis; Clustering; Blending

## 1. Introduction

Exemplar-based texture modeling has become increasingly popular because of the large variety of textures that can be accurately modeled with simple texture representations. In exemplar-based modeling, sample images of texture are used to build statistical models for texture recognition (Debonet and Viola, 1998) or for regenerating new instances (synthesis) (Heeger and Bergen, 1995). Texture synthesis plays an important role, not only in computer graphics, but also in verifying the validity of the texture representation. A correct synthesis results gives credibility to a given computational representation.

In this paper we investigate the existing methods of exemplar-based texture modeling for synthesis and present two improvements. The first improvement is the addition of clustering to identify characteristic primitives and to increase compactness and efficiency. We demonstrate the

* Corresponding author. Tel.: +1-732-445-5253; fax: +1-732-445-0593.
*E-mail address:* kdana@ece.rutgers.edu (K.J. Dana).

0167-8655/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2003.12.015

advantage of clustering by modifying the Debonet algorithm (Debonet, 1997). Adding clustering to this algorithm is sensible since many texture features are repetitive and identification of canonical primitives can greatly simplify the texture representation. We show that there are significant computational advantages when employing clustering in texture modeling.

The second improvement to exemplar-based models is not in the primitive itself but rather in the image-processing technique of blending that is needed at the placement stage of synthesis. We show that the quality of blending is often more important than the placement rule. The importance of blending algorithms in the context of texture synthesis has been understated in the literature. In our study, we evaluate existing approaches to blending including feathering and multiresolution splines. In addition, we introduce an approach using image fusion that blends borders better than prior techniques. While image fusion has been used for blending and merging in prior work (Burt and Adelson, 1985; Ogden et al., 1985), its specific application in exemplar-based texture synthesis is new.

While the modeling methods discussed here are applied to texture synthesis, they also have direct implications for texture recognition. Exemplar-based modeling using clusters can clearly be used for recognition, and the blending used for synthesis has potential for blending representative models at borders in segmentation/classification tasks.

## 2. Clustering for texture synthesis

### 2.1. Method

Following Heeger and Bergen's pioneering work (Heeger and Bergen, 1995), recent texture synthesis algorithms try to learn feature statistics across various scales by using oriented edge filter banks. A white noise image is transformed to look like the input texture image by iteratively matching its feature histograms to the example feature histogram at each level in a multiscale pyramid framework. Heeger and Bergen's method works well for highly stochastic textures but fails for more structured textures. Debonet presents an algorithm (Debonet, 1997) belonging to the same class as above which decomposes the input texture image into a multiscale pyramid and modifies it based on parent feature similarity constraints. The modified multiscale pyramid is used to reconstruct an image which represents the same texture class as the original input image. In the following sections, we describe a modification of the texture representation in the original Debonet algorithm which makes use of clustering and attains useful computational advantages.

In Debonet's algorithm, a candidate set for a given pixel is a set of pixels with similar parents in the pyramid structure. The parent relationship in the image pyramid is illustrated in Fig. 1. A pixel in the texture image is synthesized by sampling from its candidate list. In this algorithm, most of the computational effort is expended in finding candidates for each pixel at each level of the pyramid. The search for candidates involves doing distance computations between multidimensional feature vectors. Furthermore, the resultant distances must be thresholded to determine a set of similar features.

With a one-time offline feature clustering, the costly repetitive distance computations are eliminated. Also, similar features can simply be defined as features within a cluster. This scheme removes the threshold parameter from the texture generation procedure. In the original synthesis algorithm, threshold parameters need to be tuned *exclusively*

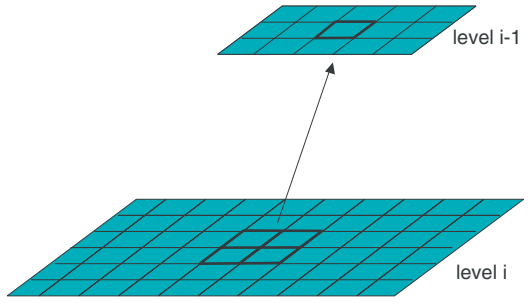


Fig. 1. Illustration of the parent–child relationship among pixels in adjacent scales of a multiresolutional pyramid. For each pixel in level $i-1$, there are four pixels in level $i$ which are called children.

for different texture images. However, in the case of offline clustering, only *one time tuning* is required when choosing the number of clusters and after that the same library of primitives can be used for synthesizing different texture images. The threshold functions used in Debonet's algorithm now manifest themselves as the number of clusters chosen for offline clustering. This method saves the user from going through the time consuming process of choosing suitable thresholds for each texture image during synthesis. Instead, all parameter tuning is done at a prior stage.

Feature clustering is done using a large and diverse set of images with representative local features such as edges, lines, spots, etc. We intuitively expect the feature vectors to form clusters in a multidimensional space. As such, feature vectors from all the images at the same scale are used as input to the standard $k$-means clustering algorithm. The resultant mean vectors are computed at each scale and stored as representatives of all commonly occurring features.

For generating a new texture, an image pyramid (e.g. laplacian, Burt and Adelson, 1983 or steerable, Freeman and Adelson, 1991) is built from the example texture. Feature vectors are computed for each pixel. All pixels at the current pyramid level are labeled with an integer label depending upon the index of the mean vector which falls closest to the feature vector at that pixel. This procedure is repeated for all levels of the pyramid. When searching for pixels with similar parents in the pyramids, clustering labels are used for comparing parent structures. Two pixels are considered similar and can be treated as candidates for each other only if they have the same label vector corresponding to their parent vectors. In this manner, costly floating distance computations between feature vectors get replaced by single integer label comparisons, achieving the desired speedup.

### 2.2. Computational advantages of clustering

Consider a computational comparison of two methods of texture synthesis: Debonet's algorithm and the clustering method. Consider an $N \times N$ image. The maximum height $L$ of a pyramid is given by $L = [\log_2 N + 1]$. Label the pyramid levels from 0 to $L-1$ starting from the finest (original image) going up to the coarsest level $L-1$.

Synthesis at $i$th level requires computation of $\frac{N^4}{4^{2(i+1)}}$ distance functions between parent feature vectors at the $(i+1)$th level. This computation may be followed by further distance computations between grandparents at $(i+2)$th level and so on, which is ignored here. Then, the total time required $T$ is given by

$$T = T_{\mathrm{f}} \sum_{i=0}^{L-2} \left[ \frac{qN^4}{4^{2(i+1)}} \right], \tag{1}$$

where $T_{\mathrm{f}}$ is the time for a floating point distance calculation and $q$ is the dimension of the local feature vector. For example, if steerable pyramids are used then $q$ is the number of subbands and if laplacian pyramids are used then $q = 1$.

Now let us consider the modified algorithm that uses offline clustering. In this case, clustering is done offline and is not considered while computing the synthesis cost. Assume the total number of clusters at all levels $= K$. After clustering, only the cluster centers need to be stored (length $q$ each). Assigning labels to each of the pixels in the image pyramid involves comparisons of the feature vector of each pixel at a level to the mean vectors of all clusters at that level. These comparisons involve $K\left(\frac{N}{2^i}\right)^2$ floating point computations at each level. After this one time fixed cost, only integer valued labels are to be compared to determine candidates at each level, so

$$T \propto T_{\mathrm{i}} N^2, \tag{2}$$

where $T_{\mathrm{i}}$ denotes the time for an integer comparison.

Let the cost of a distance computation be $F$ times that of an integer comparison, where $F = T_{\mathrm{f}}/T_{\mathrm{i}}$. If $F$ or $N$ is large, then the speedup achieved by offline clustering compared to Debonet's original algorithm is substantial.

Our implementation of the Debonet algorithm (with and without clustering) uses candidate propagation across successive levels of the pyramid which gives added times savings. Candidate propagation means that candidate sets at any level are inherited from prior levels. Therefore at

a given level, the candidate search for a pixel is performed only among the children of candidates for that pixel's parent at the previous level. This candidate propagation is motivated by the fact that if the grandparents of two pixels are not similar, then they cannot be similar across scale even if their immediate parents have matching features.

### 2.3. Clustering results

Figs. 2 and 3 show representative results produced by our algorithm and the original Debonet algorithm. The synthesis results are quite similar indicating the validity of this clustering approach. Note that the images are processed by a histogram matching stage where the intensities of the synthesized images are matched to the intensities of the exemplar image to remove any global brightness and contrast mismatch.

## 3. Patch-based sampling

The Debonet algorithm (Debonet, 1997) as well as Heeger and Bergen (1995) and Portilla and Simoncelli (2000) represent a class of texture synthesis algorithms which compute global texture statistics. In contrast, a second approach considers local properties and synthesizes pixels incrementally. Several successful algorithms following the second approach have been suggested recently (Efros and Freeman, 2001; Ashikhmin, 2001; Liang et al., 2001). Here we first implement one such algorithm, namely the patch-based sampling algorithm (Liang et al., 2001), which serves as an illustrative example for demonstrating the importance of blending.

The patch-based sampling algorithm uses the texture patches of the input sample texture $I_{\text{in}}$ as the building blocks for constructing the synthesized texture $I_{\text{out}}$. In each step, a patch $B_k$ of the input sample texture $I_{\text{in}}$ is pasted into the synthe-

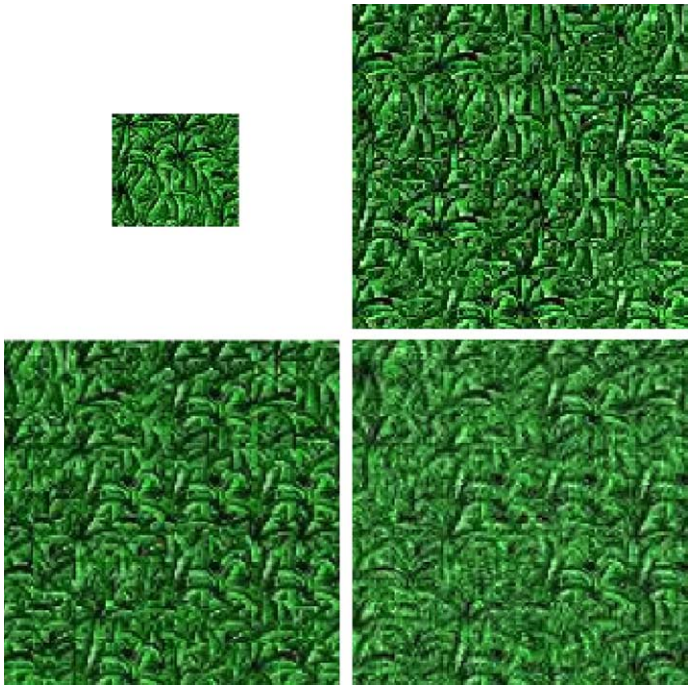

Fig. 2. Comparison of Debonet's results with those obtained using the our modified clustering-based algorithm. Input texture is on top left ($64\times64$ pixels). Debonet's results are on top right. On the bottom left, we have results from our modified algorithm when reconstruction is done using a laplacian pyramid. The synthesis result on the the bottom right was obtained when a steerable pyramid is used for reconstruction.

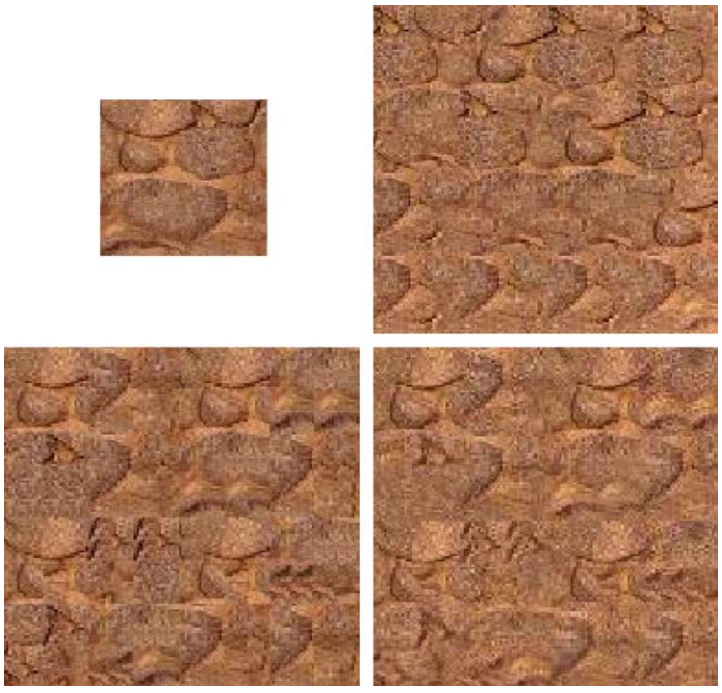

Fig. 3. Comparison of Debonet's results with those obtained using the our modified clustering-based algorithm. Input texture is on top left (64×64 pixels). Results from the Debonet algorithm are on top right. On the bottom left, we have results from our modified algorithm when reconstruction is done using a laplacian pyramid. The synthesis result on the bottom right was obtained when a steerable pyramid is used for reconstruction.

sized texture $I_{\text{out}}$. To avoid mismatching features across patch boundaries, $B_k$ is carefully selected based on the patches already pasted in $I_{\text{out}}$, $(B_0, \ldots, B_{k-1})$. The set $\psi_B$ is created with all texture patches of $I_{\text{in}}$ which have similar neighborhood as the area where the next patch should be pasted. For finding out the texture patches which will form the set $\psi_B$, an exhaustive search of the whole image needs to be done by choosing all distinct possible texture patches from the image. However in our implementation, in order to reduce computational costs, we instead choose texture patches uniformly from the image and form a patch population. The size of the patch population is a parameter that can be controlled. Each patch from this population is then checked to see if it could be added to the set $\psi_B$. An element is randomly selected from $\psi_B$ as the $k$th texture patch $B_k$. The output texture is updated by inserting $B_k$ into $I_{\text{out}}$. Then the value of $k$ is incremented and the processing is repeated until $I_{\text{out}}$ is fully covered. The texture patches are pasted in a scanline order and blending is done across the boundary zones.

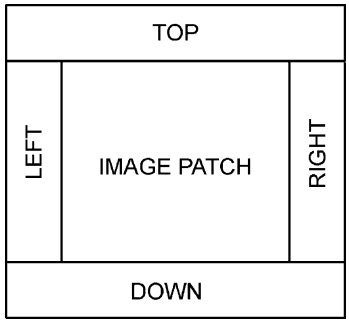


Fig. 4. The four boundaries of a texture patch.

The boundary zones of a square texture patch are shown in Fig. 4. There are four boundary zones of a square texture patch, namely, top, down, left and right. Define $P'_{\text{l}}$ and $P'_{\text{t}}$ as the texture patches to the left and top of the area where the next patch should be pasted. Then, a texture

patch $P$ from $I_{\text{in}}$ is added to the set $\psi_B$ if its left and top boundary zones match with the right and down boundary zones of $P'_{\text{l}}$ and $P'_{\text{t}}$, respectively. Define the match measure between any two boundary zones $Z_1$ and $Z_2$ as

$$D(Z_1, Z_2) = \left[ \frac{1}{A} \sum_{i=1}^{A} (p_{1i} - p_{2i})^2 \right]^{1/2}, \tag{3}$$

where $p_{1i}$ denotes the $i$th pixel in $Z_1$, $p_{2i}$ denotes the $i$th pixel in $Z_2$, and $A$ is the total number of pixels. Let $Z_{\text{r}}(P)$, $Z_{\text{d}}(P)$, $Z_{\text{l}}(P)$ and $Z_{\text{t}}(P)$ denote the right, down, left and top boundary zones of $P$, respectively. Then, patch $P$ from $I_{\text{in}}$ is added to $\psi_B$ if

$$D(Z_{\text{l}}(P), Z_{\text{r}}(P'_{\text{l}})) < (1+\tau) \times \arg\min_X D(Z_{\text{l}}(X), Z_{\text{r}}(P'_{\text{l}})) \tag{4}$$

and

$$D(Z_{\text{t}}(P), Z_{\text{d}}(P'_{\text{t}})) < (1+\tau) \times \arg\min_Y D(Z_{\text{t}}(Y), Z_{\text{d}}(P'_{\text{t}})), \tag{5}$$

where $\tau$ is the border similarity threshold.

## 4. Blending for texture synthesis

Texture synthesis algorithms such as Debonet's algorithm essentially reposition patches of the image. Indeed much emphasis in the research literature has been placed on rearranging the image in a smart way that results in the correct feature distribution. There is the potential for border artifacts and blending at the patch borders is a critical part of the algorithm because blending greatly reduces the appearance of seams. While experimenting with texture synthesis algorithms, we made an interesting and surprising observation. Except for highly structured textures, most textures could be satisfactorily synthesized by putting together patches randomly and doing blending carefully. Fig. 5 shows synthesis results when patches were chosen randomly from the input image and placed arbitrarily in the output image without imposing any neighborhood similarity constraints. The patch size chosen was $96 \times 96$ pixels so as to accommodate the largest features present in the input images. Because of blending, no visible artifacts are present in the synthesized

Fig. 5. Synthesized images obtained by choosing patches randomly and applying multiresolution splining (size: $356 \times 356$ pixels).

images and the quality of synthesis result for unstructured textures is quite reasonable. This observation suggests that blending plays a more important role than the placement rule in texture synthesis.

In the literature, the importance of blending has been underemphasized. Here we list and compare the types of blending methods currently used for texture synthesis. We show results with the popular blending techniques, feathering and multiresolution spline blending. Then we introduce a new method of blending during texture synthesis that uses image fusion. We demonstrate the results using patch-based sampling.

### 4.1. Prior blending methods

#### 4.1.1. Feathering

The problem that blending attempts to solve is to remove the edge discontinuity which arises due to non-matching features at the boundary where two images are joined together. The basic idea in feathering (Szeliski and Shum, 1997) is to replace each pixel in the overlapping boundary zones of two patches by the weighted average of pixels from the two patches at that location. The weight assigned to a pixel is a monotonic function of its distance from the end of the overlapping area between the patches. The choice of width of overlap or transition region between adjacent images is a matter of concern. If the width is chosen too small compared to the image features, the boundary may still appear as a step in image color level, albeit a somewhat blurred step. If, on the other hand, the width is large compared to image features, features from both the images may appear superimposed within the overlap region, as in a photographic double exposure.

#### 4.1.2. Multiresolution splining

Multiresolution splining (Adelson and Burt, 1993) is a blending technique which proceeds by splitting the images to be joined into a set of bandpass images and choosing the proper transition width for each band. It has been shown (Adelson and Burt, 1993) that decomposing the images to be joined into gaussian pyramids and reconstructing the image corresponding to a new pyramid formed by copying nodes from both the pyramids is equivalent to choosing a smooth weight function (gaussian) and appropriate transition widths for different bands. The algorithm which uses multiresolution splining as the blending step in patch-based sampling scheme is as follows:

(1) Employ the patch-based sampling algorithm given in the previous section with an additional data structure to keep track of the positions of patches in $I_{\text{in}}$ which are being filled in the synthesized texture image $I_{\text{out}}$.
(2) Construct the gaussian pyramids $P_{\text{out}}$ and $P_{\text{in}}$ for the synthesized texture image $I_{\text{out}}$ and the original input texture image $I_{\text{in}}$, respectively.
(3) Start with the finest level of $P_{\text{out}}$ and traverse through all the nodes (pixels at all levels). For each node $N_{\text{out}}$ at location $(x_{\text{out}}, y_{\text{out}})$, find which patch $p_k$ it belongs to and the location $(x_k, y_k)$ of that patch $p_k$ in the input image $I_{\text{in}}$ (location of a patch is defined by the position of its leftmost-topmost pixel). If the relative position of $N_{\text{out}}$ within patch $p_k$ is $(u_{\text{out},k}, v_{\text{out},k})$, then replace $N_{\text{out}}$ with a node $N_{\text{in},k}$ from the finest level of $P_{\text{in}}$ at position $(x_k + u_{\text{out},k}, y_k + v_{\text{out},k})$. Repeat this at all coarser levels until the patches get reduced to a single node. In our implementation, we stop after the first three levels which is sufficient since after that all patches start looking very similar.
(4) Reduce modified $P_{\text{out}}$ to get new $I_{\text{out}}$.

### 4.2. Image fusion for blending

Blending works quite well and smooth out all the edge discontinuities present in the synthesized image after the sampling step. However, this technique has drawbacks because weighted averaging can cause undesirable modification to appearance of features. Features present at the patch boundaries lose contrast when they are averaged with featureless regions. A technique is needed which would preserve the strength of important features in the overlapping boundary region.

Image fusion (Burt, 1992) is such a technique which is widely used by the multisensor fusion community to combine images of different

modalities in such a way so as to preserve maximum amount of perceptually important information. This preservation is accomplished by forming the laplacian pyramids for the two images and comparing them node by node. Whichever node has a higher absolute value, signifies a stronger feature and gets copied into a new pyramid which is finally reduced to give the fused image containing information from both images. Here we describe an algorithm which uses image fusion in place of blending in the patch-pasting scheme. Instead of simply copying the nodes for each patch from the input pyramid to the output pyramid (as was done in the multiresolution spline case), adjacent patches are allowed to overlap with each other. In regions of overlap, multiple nodes are present at the same location which correspond to different patches (two at the boundaries and four at the corners). Out of these multiple nodes, the node which has the highest absolute value is selected and copied into the output pyramid. In regions of no overlap, only one node is present at

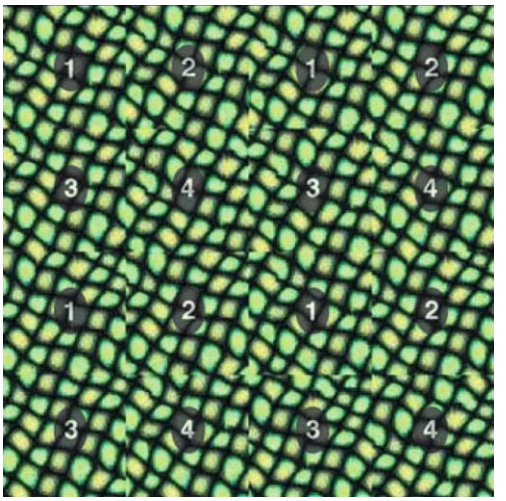


Fig. 6. Illustration of the patch labeling procedure. This image shows a synthesized texture before blending. For multiresolution blending, four new images $I_1$, $I_2$, $I_3$, $I_4$ will be created. These four images are the same size as the synthesized texture. $I_i$ is constructed by copying all patches with label $i$ from this image and putting zeroes everywhere else.

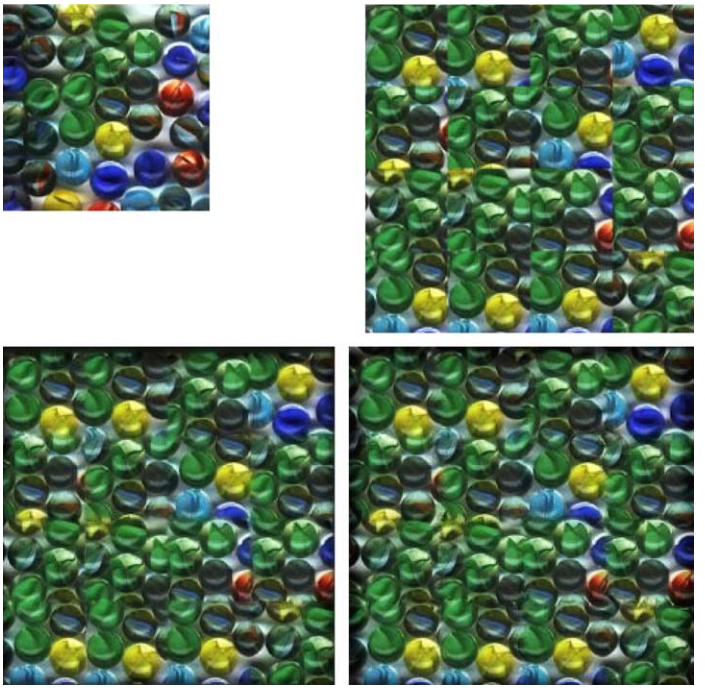

Fig. 7. (Top left) Original image; (top right) synthesized image without blending, (bottom left) synthesized image with multiresolution spline blending, (bottom right) synthesized image with fusion.

each location and is copied directly from the input image.

Label the texture patches in the synthesized texture $I'_{out}$ (not blended) as shown in Fig. 6. Construct four different images ($I_1$, $I_2$, $I_3$ and $I_4$) each one of which contains patches with a unique label (1, 2, 3 or 4). The four boundary zones of each patch (as depicted in Fig. 4) are copied along with that patch. These boundaries are found in the original image by making use of the location information of each patch which we stored in the first step. Pixels at all the other locations in the four images are filled with zeroes. Note that adjacent patches have different labels and thus fall into separate images.

Construct laplacian pyramids corresponding to each of the four images ($I_1$, $I_2$, $I_3$ and $I_4$) and call them ($P_1$, $P_2$, $P_3$ and $P_4$). Let $I_{out}$ be the synthesized output texture image which is initially the same as $I'_{out}$. Let $P_{out}$ be the laplacian pyramid for $I_{out}$. Now, for each node (pixel at any pyramid level) $N_{out}$ in $P_{out}$, compare the absolute value of the four corresponding nodes in $P_1$, $P_2$, $P_3$ and $P_4$ and replace $N_{out}$ with the node having maximum absolute value. There are three cases to be considered. In the first case, $N_{out}$ lies at a patch corner which overlaps with boundaries from three other patches. In this situation, these four overlapping values will be present in the nodes corresponding to $N_{out}$ in ($P_1$, $P_2$, $P_3$ and $P_4$) and a comparison will be made among these values. In the second case, $N_{out}$ is at a boundary which overlaps with just one more boundary. Here, two out of the four nodes corresponding to $N_{out}$ in ($P_1$, $P_2$, $P_3$ and $P_4$) will contain zeroes and effectively only the other two will be compared. Finally, if $N_{out}$ is inside a patch with no overlap, then three of the four nodes corresponding to $N_{out}$ in ($P_1$, $P_2$, $P_3$ and $P_4$) will contain zeroes and the fourth node will be same as $N_{out}$ which will be copied into $P_{out}$. Reducing $P_{out}$ back into $I_{out}$ gives the synthesized texture image.

Image fusion preserves the strongest features that fall in the overlap region. However, since the selection rule in the fusion procedure compares pixels, we sometimes see high frequency artifacts where continuity of larger features is destroyed by smaller but stronger features. One heuristic which helps avoid this situation to some extent is to attach weights with the features in the transition zone such that features which are nearer to their parent patches get more weight than overlapping features from the adjacent patch. At the midpoint of the transition zone, features from both the patches get equal weight. We call this scheme *weighted image fusion*.

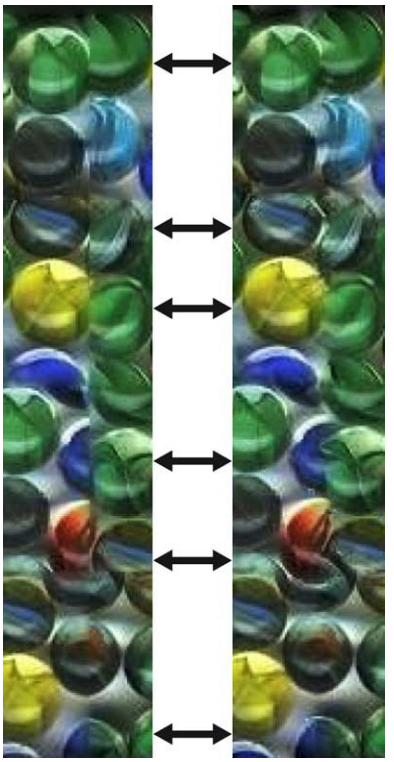

Fig. 8. A larger view of one section from the bottom left and bottom right of Fig. 7. Notice that in general there is a line down the center that is more visible in the multiresolution spline blending result. In the fusion result, this line is less apparent. The arrows highlight areas where the fusion of image structure shows a marked improvement over blending.

### 4.3. Results

The synthesis results presented in this section were produced by running the algorithm with the following parameters: patch size = 96×96 pixels; patch border width = 16 pixels; patch population = 400; border similarity threshold is 0.1; all

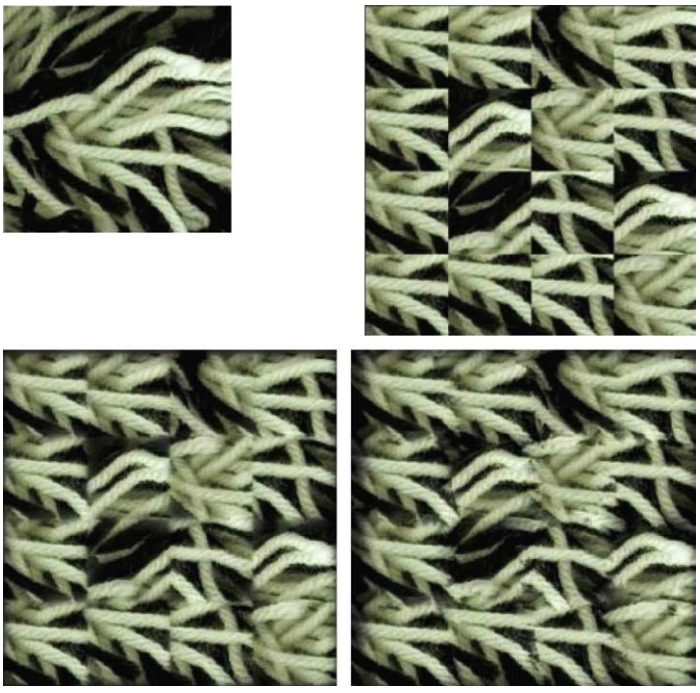

Fig. 9. (Top left) Original image; (top right) synthesized image without blending; (bottom left) synthesized image with multiresolution spline blending and (bottom right) synthesized image with fusion.

input images [1] are $256 \times 256$ pixels. For the fusion results, weighted fusion is employed.

*4.3.1. Comparison between standard blending and fusion*

Figs. 7–9 provide a comparison between multiresolution spline blending and fusion procedures. The synthesized images are created using patch-based sampling followed by either multiresolution spline blending or image fusion. Comparison of the results shows that multiresolution spline blending sometimes introduces undesirable blurring and dulls the features near boundaries. Fusion however, gives preference to the stronger feature so there is no blurring or contrast reduction. This observation is emphasized in Fig. 8 where the arrows indicate portions of the image where the fusion result is superior to the multiresolution spline result. Fusion preserves the visual appearance of features in the boundary and as a result the border is less apparent.

Fig. 9 presents a challenging image texture to synthesize with patch-based methods. The long linear features in the image are broken by the smaller patches used for synthesis. The fusion method completes the truncated linear features so that the visual artifact is less apparent than the other results shown. Increasing patch size can help reduce the problem, but at the cost of making the synthesized image look too similar to the original. In general, the algorithm works best when the texture is less regular and when the size of the characteristic features is smaller than the patch.

## 5. Conclusion and summary

Texture representations for a variety of algorithms can be evaluated by confirming the synthesized texture based on the representation is

[1] The input images were downloaded from texture research webpages including www.cns.nyu.edu/~lcv/texture/color.

similar to the original texture. Two improvements of examplar-based texture representations show promising results. Clustering enables computational savings by avoiding distance calculations and blending with fusion allows seamless joining of patch borders of neighboring texture elements.

## Acknowledgements

This work has been partly supported by the National Science Foundation under Grant Nos. 0092491 and 0076585.

## References

Adelson, E.H., Burt, P.J., 1993. A multiresolution spline with application to image mosaics. ACM Trans. Graph. 2 (4), 217–236.

Ashikhmin, M., 2001. Synthesizing natural textures. In: The Proc. 2001 ACM Symp. on 3D Interactive Graphics, March, pp. 217–226.

Burt, P.J., 1992. A gradient pyramid basis for pattern-selective image fusion. Soc. Inform. Display Dig., 467–470.

Burt, P.J., Adelson, E.H., 1983. The laplacian pyramid as a compact image code. IEEE Trans. Commun. 31 (4), 532–540.

Burt, P.J., Adelson, E.H., 1985. Merging images through pattern decomposition. Proc. SPIE 575, 173–181.

Debonet, J.S., 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. ACM SIGGRAPH (August), 361–368.

Debonet, J.S., Viola, P., 1998. Texture recognition using a non-parametric multi-scale statistical model. In: Proc. IEEE Conf. on Comput. Vision Pattern Recognition, pp. 641–647.

Efros, A.A., Freeman, W.T., 2001. Image quilting for texture synthesis and transfer. ACM SIGGRAPH (August), 341–346.

Freeman, W.T., Adelson, E.H., 1991. The design and use of steerable filters. IEEE Trans. Pattern Anal. Machine Intell. 13 (9), 891–906.

Heeger, D.J., Bergen, J.R., 1995. Pyramid-based texture analysis/synthesis. ACM SIGGRAPH (August), 229–238.

Liang, L., Liu, C., Xu, Y., Guo, B., Shum, H., 2001. Real-time texture synthesis by patch-based sampling. ACM Trans. Graphics 20 (3), 127–150.

Ogden, J., Adelson, E., Bergen, J., Burt, P., 1985. Pyramid-based computer graphics. RCA Eng. 30, 4–15.

Portilla, J., Simoncelli, E.P., 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. Int. J. Computer Vision 40 (1), 49–70.

Szeliski, R., Shum, H., 1997. Creating full view panoramic image mosaics and environment maps. ACM SIGGRAPH (August), 251–258.