Analyzing the layout of composite textures

G. Caenen[1], V. Ferrari[2], A. Zalesny[2], and L. Van Gool[1,2]
[1] D-ITET/BIWI, ETH Zurich, Switzerland, lastname@vision.ee.ethz.ch
[2] ESAT/PSI Visics, Univ. of Leuven, Belgium, lastname@esat.kuleuven.ac.be

Abstract— **The synthesis of many textures can be simplified if they are first decomposed into simpler subtextures. Such bootstrap procedure allows to first consider a 'label texture', that captures the layout of the subtextures, after which the subtextures can be filled in. A companion paper focuses on this latter aspect. This paper describes an approach to arrive at the label texture. Pairwise pixel similarities are computed by matching simple color and texture features histograms in pixel neighbourhoods, using efficient mean-shift search. A graph-based, unsupervised algorithm segments the image into subtextures, based on the similarities.**

Keywords—**segmentation, composite textures, texture synthesis**

I. Introduction

Many textures are so complex that for their analysis and synthesis they can better be considered a composition of simpler subtextures. A good case in point are landscape textures. Open pastures can be mixed with patches of forest and rock. The direct synthesis of the overall texture would defy existing methods. The whole only appears to be one texture at a very coarse scale. In terms of intensity, colors, and simple filter outputs such scene can not be considered 'homogeneous'. The homogeneity rather exists in terms of the regularity (in a structural or stochastic sense) in the layout of simpler subtextures.

Texture can be thought of as a regular layout of simpler structures, which themselves can be considered in such way. This process of decomposition can end at the point where the remaining subtextures are homogeneous in terms of very simple features, like color or simple filter outputs. This recursive definition suggests a hierarchical texture modeling scheme. This paper discusses an approach to analyze one layer of this hierarchy, directly above the level where the subtextures can be discovered on the basis of simple features. An unsupervised segmentation scheme is proposed, that decomposes a texture into such subtextures.

Section II discusses the simple texture features that we use, and how they are compared to group pixels into the different subtextures. Section III describes the clique partitioning method that lies at the heart of this grouping process. Section IV shows some segmentation results. Section V concludes the paper.

II. Pixel similarity scores

For the description of the subtextures, both color and structural information is taken into account. Local statistics of the (L, a, b) color coordinates and the wavelet detail coefficients (h, v, d) (horizontal, vertical and diagonal) are derived. We used a Haar-wavelet, but another wavelet family or filterbank could be used to optimize the system.

The initial (L, a, b)-color and (h, v, d)-structural feature vector of an image pixel i are both referred to as $\mathbf{x_i}$. The local statistics of the vectors $\mathbf{x_j}$ near the pixel i are captured by a local histogram p_i.

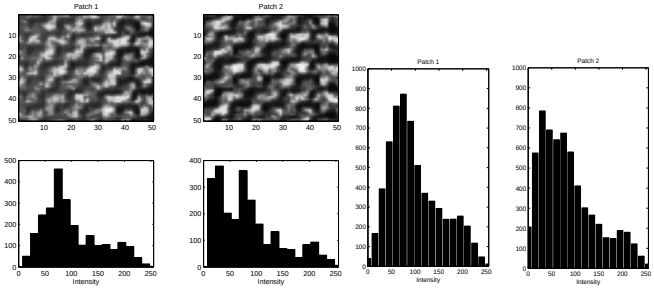

Fig. 1. *(top left) patches with identical texture and different illumination. (bottom left) traditional intensity histograms of the paches. (right) weighted histograms of the patches.*

A. Weighted Feature histograms.

To avoid problems with sparse high-dimensional histograms, we first quantize the feature space, in the same vein as the texton analysis in [7]. The bin centers are obtained by hierarchically clustering the vectors $\mathbf{x_i}$ until a fixed number of bins is reached (we chose 8) or an error is exceeded.

Instead of assigning a pixel to a single bin, each pixel is assigned a vector of weights that express its affinity to the different bins (textons). The weights are based on the euclidean distances to the bin centers. If $d_{ik} = \|\mathbf{x_i} - \mathbf{b_k}\|$ is the distance between a feature value $\mathbf{x_i}$ and the k-th bin center, we compute the corresponding weight as

$$w_{ik} = e^{-(d_{ik}^2/2\sigma^2)} \tag{1}$$

The resulting local *weighted histogram* p_i of pixel i is obtained by averaging the weights over a region R_i:

$$p_i(k) = \frac{1}{|R_i|} \sum_{j \in R_i} w_{jk} \tag{2}$$

In our experiments, R_i was chosen a circular region, with a radius of 8 pixels.

The resulting weighted histogram can be considered a smooth version of the traditional histogram. The weighting causes small changes in the feature vectors (e.g. due to non-uniform illumination) to result in small changes in the histogram. In traditional histograms this is often not the case as pixels may suddenly jump to another bin. Figure 1 illustrates this by computing histograms of two rectangular patches from a single Brodatz texture. The patches have similar texture, only the illumination is different. Clearly the weighted histograms (right) are less sensitive to this difference. This is reflected in a higher Bhattacharyya score (3).

Color and structural histograms are computed separately. In a final stage, the color and structure histograms are simply con-

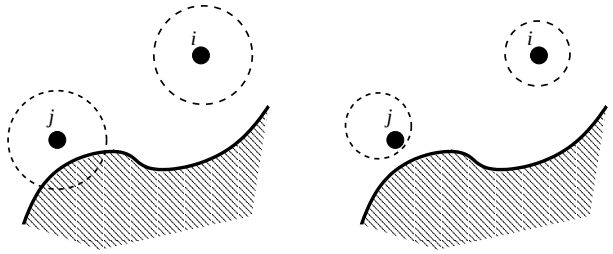

Fig. 2. *(left) normal comparison between two pixels, the dashed lines indicate the supports R_i and R_j of the histograms. (right) comparison with shifts, avoiding problems near texture borders.*

catenated into a single, longer histogram and the $p_i(k)$ are scaled to ensure that $\sum_k p_i(k) = 1$.

In order to compare the feature histograms, we have used the Bhattacharyya coefficient ρ. Its definition for two frequency histograms $p = (p_1, \ldots, p_n)$ and $q = (q_1, \ldots, q_n)$ is

$$\rho(p, q) = \sum_i \sqrt{p_i \cdot q_i} \tag{3}$$

This coefficient is proven to be more robust [8] in the case of zero count bins and non-uniform variance than the chi-squared statistic. In fact, after a few manipulations one can show the following relation:

$$\rho(p, q) \approx 1 - \frac{1}{8} \sum_i \frac{(q_i - p_i)^2}{p_i} = 1 - \frac{1}{8}\chi^2(p, q) \tag{4}$$

Another advantage of the Bhattacharyya coefficient over the χ^2-measure is that it is symmetric, which is more natural when similarity has to be expressed.

B. Shifted Matching.

In order to evaluate the similarity between two pixels, their feature histograms are not simply compared. Rather, the comparison of the histogram for the first pixel is made with those of all pixels in a neighbourhood of the second. The best possible score is taken as the similarity $S'(i, j)$ between the two pixels. This allows the system to assess similarity without having to collect histograms from large regions R_i. The advantage is that boundaries between subtextures are better located, as is shown in figure 2.

The search for the location with the best matching histogram close to the second pixel is based on the mean shift gradient to maximize the Bhattacharyya measure [6]. This avoids having to perform an exhaustive search. Comparisons are in fact also carried out over a number of different scales. This is to cater for perspective effects and the like that may exist within a single subtexture. A final refinement is by defining a symmetric similarity measure S: $S(i, j) = S(j, i) = \max\{S'(i, j), S'(j, i)\}$.

As shifted matches cause neighbouring pixels to have an exact match, the similarity scores are only computed for a subsample (a regular grid) of the image pixels, which also yields a computational advantage. Yet, after segmentation of this sample, a high-resolution segmentation map is obtained as follows. The histogram of each pixel is first compared to each entry in the list of neighbouring sample histograms (at true scale only). The pixel is then assigned to the best matching class in the list.

Our particular segmentation algorithm requires a calibrated similarity matrix S with entries ≥ 0 indicating that pixels are likely to belong together and entries < 0 indicating the opposite. The absolute value of the entry is a measure of confidence. So far, all the similarities S have positive values. We subtract a constant value, which in all our experiments was kept the same. With this fixed value images with different numbers of subtextures could be segmented successfully. Hence, the number of subtextures was not given to the system, as would e.g. be required in k-means clustering. Having this threshold in the system can be an advantage, as it allows the user to express what he or she considers to be perceptually similar.

III. Pixel grouping

A. Clique partitioning

In order to achieve the intended, unsupervised segmentation of the composite textures into simpler subtextures, pixels need to be grouped into disjoint classes, based on their pairwise similarity scores. Taken on their own, these similarities are too noisy to yield robust results. Pixels belonging to the same subtexture may e.g. have a negative score (false negative) and pixels of different subtextures may have positive scores (false positives). Nevertheless, taken altogether, the similarity scores carry quite reliable information about the correct grouping. The transitivity of subtexture membership is crucial: if pixels p_i, p_j are in the same class and p_j, p_k too, then p_i, p_j and p_k *must* belong to the same class. Even if one of the pairs gets a falsely negative score, the two others can override a decision to split. Next, we formulate the texture segmentation problem so as to exploit transitivity to detect and avoid false scores. We present a time-optimised adaptation of the grouping algorithm we first introduced in [9]. We construct a complete graph G where each vertex represents a pixel and where edges are weighted with the pairwise similarity scores. We partition G into completely connected disjoint subsets of vertices (cliques) so as to maximize the total score on the remaining edges (Clique Partitioning, or CP). The transitivity property is ensured by the clique constraint: every two vertices in a clique are connected, and no two vertices from different cliques are connected. The CP formulation of texture segmentation is made possible by the presence of positive *and* negative weights: they naturally lead to the definition of a *best solution* without the need of knowing the number of cliques (subtextures) or the introduction of artificial stopping criteria as in other graph-based approaches based on strictly positive weights [3], [1]. On the other hand, our approach needs the parameter t_0 that determines the splitting point between positive and negative scores. But, as our experiments have shown, the same parameter value yields good results for a wide range of images. Moreover, the same value yields good results for examples with a variable number of subtextures. This is much better than having to specify this number, as would e.g. be necessary in a k-means clustering approach.

CP can be solved by Linear Programming [2] (LP). Let w_{ij} be the weight of the edge connecting (i, j), and $x_{ij} \in \{0, 1\}$ indicate whether the edge exists in the solution. The following LP can be established:

$$\begin{array}{ll} \text{maximize} & \sum_{1\le i<j\le n} w_{ij}x_{ij} \\ \text{subject to} & x_{ij}+x_{jk}-x_{ik}\le 1, \quad \forall\, 1\le i<j<k\le n \\ & x_{ij}-x_{jk}+x_{ik}\le 1, \quad \forall\, 1\le i<j<k\le n \\ & -x_{ij}+x_{jk}+x_{ik}\le 1, \quad \forall\, 1\le i<j<k\le n \\ & x_{ij}\in\{0,1\}, \quad \forall\, 1\le i<j<k\le n \end{array} \tag{5}$$

The inequalities express the transitivity constraints, while the objective function to be maximized corresponds to the sum of the intra-clique edges.

B. A fast approximation

Unfortunately CP is an NP-hard problem [2]: LP (5) has worst case exponential complexity in the number n of vertices (pixels), making it impractical for large n. The challenge is to find a practical way out of this complexity trap. The correct partitioning of the example in figure 3 is $\{\{1,3\},\{2,4,5\}\}$. A simple greedy strategy merging two vertices (i,j) if $w_{ij}>0$ fails because it merges $(1,2)$ as its first move. Such an approach suffers from two problems: the generated solution depends on the *order* by which vertices are processed and it looks only at *local* information.

We propose the following iterative heuristic. The algorithm starts with the partition

$$\Phi=\{\{i\}\}_{1\le i\le n}$$

composed of n singleton cliques each containing a different vertex. The function

$$\mathrm{m(c_1,c_2)}=\sum_{\mathrm{i\in c_1,j\in c_2}}\mathrm{w_{ij}}$$

defines the cost of merging cliques c_1, c_2. We consider the functions

$$\begin{array}{rcl} \mathrm{b(c)} & = & \max_{t\in\Phi}\mathrm{m(c,t)} \\ \mathrm{d(c)} & = & \arg\max_{\mathrm{t}\in\Phi}\mathrm{m(c,t)} \end{array}$$

representing, respectively, the score of the best merging choice for clique c and the associated clique to merge with. We merge cliques c_i, c_j if and only if the three following conditions are met simultaneously

$$\mathrm{d(c_i)=c_j},\quad \mathrm{d(c_j)=c_i},\quad \mathrm{b(c_i)=b(c_j)>0}\ .$$

In other words, two cliques are merged only if each one represents the best merging option for the other and if merging them increases the total score. At each iteration the functions $\mathrm{b(c)}, \mathrm{d(c)}$ are computed, and all pairs of cliques fulfilling the criteria are merged. The algorithm iterates until no two cliques can be merged. At each iteration, the function m can be progressively computed from its values in the previous iteration. The basic observation is that for any pair of merged cliques $c_k = c_i \cup c_j$, the function changes to $\mathrm{m(c_l,c_k)=m(c_l,c_i)+m(c_l,c_j)}$ for all $c_l \notin \{c_i, c_j\}$. This strongly reduces the amount of operations needed to compute m and makes the algorithm much faster than in [9].

Fig. 3 shows an interesting case. In the first iteration $\{1\}$ is merged with $\{3\}$ and $\{4\}$ with $\{5\}$. Notice how $\{2\}$ is, correctly, not merged with $\{1\}$ even though $m(\{1\},\{2\})=3>0$.

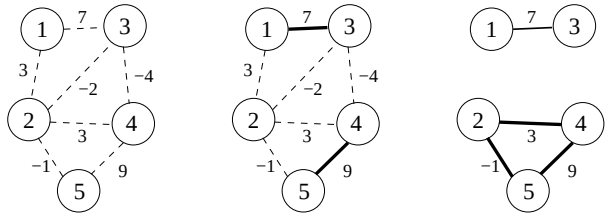

Fig. 3. *An example graph and two iterations of our heuristic. Not displayed edges have zero weight.*

In the second iteration $\{2\}$ is correctly merged with $\{4,5\}$, resisting the (false) attraction of $\{1,3\}$ ($b(\{1,3\},\{2\})=1$, $d(\{1,3\})=\{2\}$). The algorithm terminates after the third iteration because $m(\{1,3\},\{2,4,5\})=-3<0$. The second iteration shows the power of CP. Vertex 2 is connected to unreliable edges (w_{12} is false positive, w_{25} is false negative). Given vertices $\{1,2,3\}$ only, it is not possible to derive the correct partitioning $\{\{1,3\},\{2\}\}$; but, as we add vertices $\{4,5\}$, the *global information* increases and CP arrives at the correct partitioning.

The proposed heuristic is *order independent*, takes a more global view than a direct greedy strategy, and resolves several ambiguous situations while maintaining polynomial complexity. Analysis reveals that the exact amount of operations depends on the structure of the data, but it is at most $4n^2$ in the average case. Moreover, the operations are simple: only comparisons and sums of real values (no multiplication or division is involved).

In the first iterations, being biased toward very positive weights, the algorithm risks to take wrong merging decisions. Nevertheless our merging criterion ensures this risk to quickly diminish with the size of the cliques in the correct solution (number of pixels forming each subtexture) and at each iteration, as the cliques grow and increase their resistance against spurious weights.

C. Performance of the approximation

The practical shortcut for the implementation of CP may raise some questions as to its performance. In particular, how much noise on the edge weights (i.e. uncertainty on the similarity scores) can it withstand? And, how well does the heuristic approximation approach the true solution of CP? We tested both

Vert.	Cliques	Diff %	Err % LP	Err % Approx
15	3	0.53	6.8	6.93
12	2	0.5	2.92	3.08
21	3	0.05	2.19	2.14
24	3	0.2	1.13	1.33

TABLE I

Comparison of LP and our approximation. The noise level is 25%. **Diff %** *is the average percentual difference between the partitionings produced by the two algorithms. The two* **Err** *columns report the average percentage misclassified vertices for each algorithm.*

LP and the heuristic on random instances of the CP problem. Graphs with *a priori* known, correct partitioning were generated. Their sizes differed in that both the number of cliques and the total number of vertices (all cliques had the same size)

Vertices	Cliques	Noise level	Err % Approx
40	4	25	0.33
60	4	25	0.1
60	4	33	2.1
120	5	36	1.6
1000	10	40	0.7

TABLE II

Performance of the CP approximation algorithm on various problem sizes.

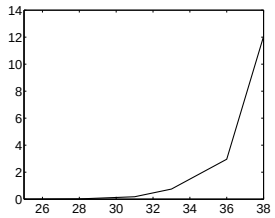

Fig. 4. *Relationship between noise level and error, for a 100 vertices, 5 cliques problem. The average percentage of misclassified vertices (X-axis) is still low with as much as 36% noise level.*

were varied. Intra-clique weights were uniformly distributed in $[-a, 9]$ with a real number, while inter-clique weights were uniformly distributed in $[-9, a]$, yielding an ill-signed edge percentage of $\frac{a}{a+9}$. This *noise level* could be controlled by varying the parameter a. Let the *difference between two partitionings* be the minimum amount of vertices that should change their clique membership in one partitioning to get the other. The quality of the produced partitionings is evaluated in terms of *average percentage of misclassified vertices*: the difference between the produced partitioning and the correct one, averaged over 100 instances and divided by the total number of vertices in a single instance.

Table II reports the performance of our approximation for larger problem sizes. Given 25% noise level, the average error already becomes negligible with clique sizes between 10 and 20 (less than 0.5%). In problems of this size, or larger, the algorithm can withstand even higher noise levels, still producing high quality solutions. In the case of 1000 vertices and 10 cliques, even with 40% noise level ($a = 6$), the algorithm produces solutions which are closer than 1% to the correct one. This case is of particular interest as its size is similar to the typical texture segmentation problems. Table I shows a comparison between our approximation to CP and the optimal solution computed by LP on various problem sizes, with constant noise level set to 25% ($a = 3$). In all cases the partitionings produced by the two algorithms are virtually identical: the average percentual difference is very small as shown in the third column of the table. Due to the very high computationals demands posed by LP, the largest problem reported here has only 24 vertices. Beyond that point, computation times run into the hours, which we consider as too impractical. Note that the average percentage of misclassifications quickly drop with the size of the cliques.

The proposed heuristic is fast: it completed these problems in less than 0.1 seconds, except for the 1000 vertices one, which took about 4 seconds on the average. The ability to deal with thousands of vertices is particularly important in our application, as every pixel to be clustered will correspond to a vertex.

Fig. 4 shows the average error for a problem with 100 vertices and 5 cliques as a function of the noise level (a varies from 3 to 5.5). Although the error grows faster than linearly, and the problem has relatively small size, the algorithm produces high quality solutions in situations with as much as 36% of noise.

These encouraging results show CP's robustness to noise and support our heuristic as a good approximation. Cliques in these experiments were only given the same size to simplify the discussion. The algorithm itself deals with differently sized cliques.

IV. Experimental results

Performance of the algorithm was tested on both composite textures and textured landscapes. Figure 5 shows a texture collage. The image was processed in three ways. In (a) we applied histogram matching without allowing "shifted" matches (i.e. a similarity match for pair (i, j) is simply $\rho(p_i, p_j)$). (c) shows the segmentation using mean shift to achieve an optimal match, which clearly solves the problems at boundaries between subtextures. Segmentation (d) was computed with the normalized cuts algorithm [3], available at `http://www.cs.berkeley.edu` along with a standard set of parameters. Figures 6 and 8 show results of our algorithm on some textures and landscapes. Finally figure 7 shows a full synthesis obtained from its segmentation map. For more details, we refer to the paper "Parallel Composite Texture Synthesis", elsewhere in the proceedings.

V. Conclusions

The paper described the segmentation part of a method that synthesizes textures by first segmenting them into simpler subtextures. An effective method, based on a very time-efficient approximation of clique partitioning, was proposed. This builds on robust color and structure related similarity scores. Future work will include texture hierarchies of more than two levels.

References

[1] J. Aslam, A. Leblanc and C. Stein A new approach to clustering In *Workshop on Algorithm Engineering*, 2000.

[2] R. Graham, M. Groetschel and L. Lovasz (eds.) Handbook of Combinatorics vol,2, Elsevier, pp. 1890-1894, 1995

[3] J. Shi and J. Malik Normalized Cuts and Image Segmentation In *IEEE CVPR*, pp. 731-737, 1997..

[4] J. Puzicha, T. Hofmann, J. Buhmann *Histogram Clustering for Unsupervised Segmentation and Image Retrieval*, Pattern Recognition Letters 20(9), 899-909, 1999.

[5] J. Puzicha, S. Belongie *Model-based Halftoning for Color Image Segmentation*, 2000.

[6] D. Comaniciu, P. Meer *Real-Time Tracking of Non-Rigid Objects using Mean Shift*, ICPR 2000, pp. 629-32 vol.3.

[7] J. Malik, S. Belongie, T. Leung and J. Shi *Contour and Texture Analysis for Image Segmentation*, Perceptual Organization for Artificial Vision Systems, Boyer and Sarkar, eds. Kluwer, 2000.

[8] F. Aherne, N. Thacker, P. Rockett *The Bhattacharyya Metric as an Absolute Similarity Measure for Frequency Coded Data* Kybernetika 34 (1998), No. 4, pp.363-368.

[9] V. Ferrari, T. Tuytelaars, L. Van Gool *Real-time affine region tracking and coplanar grouping*, in Proc. IEEE Comp. Vis. and Patt. Rec., Vol II, pp. 226-233, 2001.

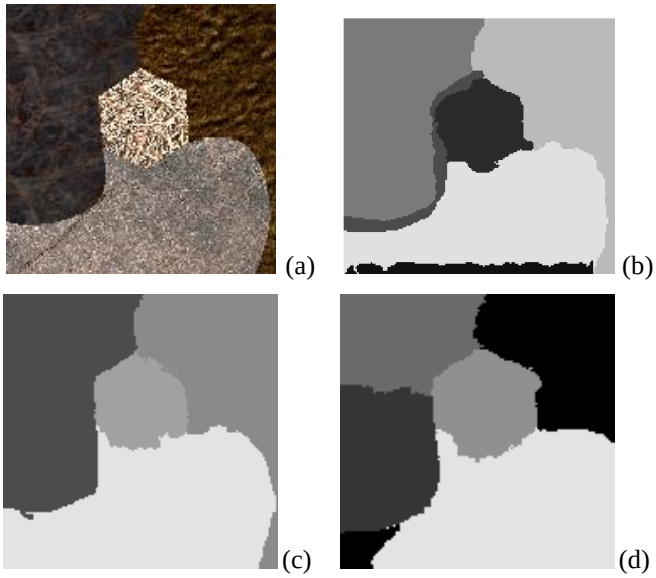

Fig. 5. *(a) original image (texture collage) and segmentations obtained (b) with CP and no shifted matching. (c) using CP and shifted matching (mean shift optimization) and (d) using Normalized Cuts.*

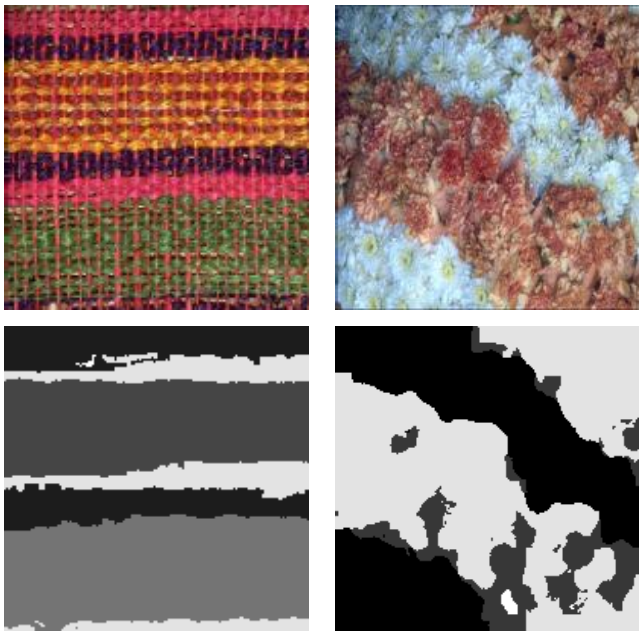

Fig. 6. *Segmentation of composite textures using our algorithm (CP and shifted matching).*

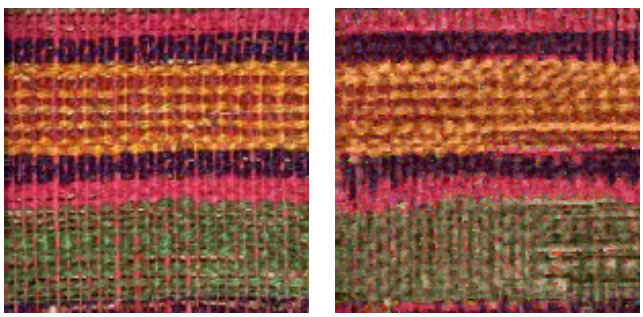

Fig. 7. *Original image and synthesis. The synthesis was obtained by separately synthesizing the subtextures from the labelmap in figure 6 (bottom left).*

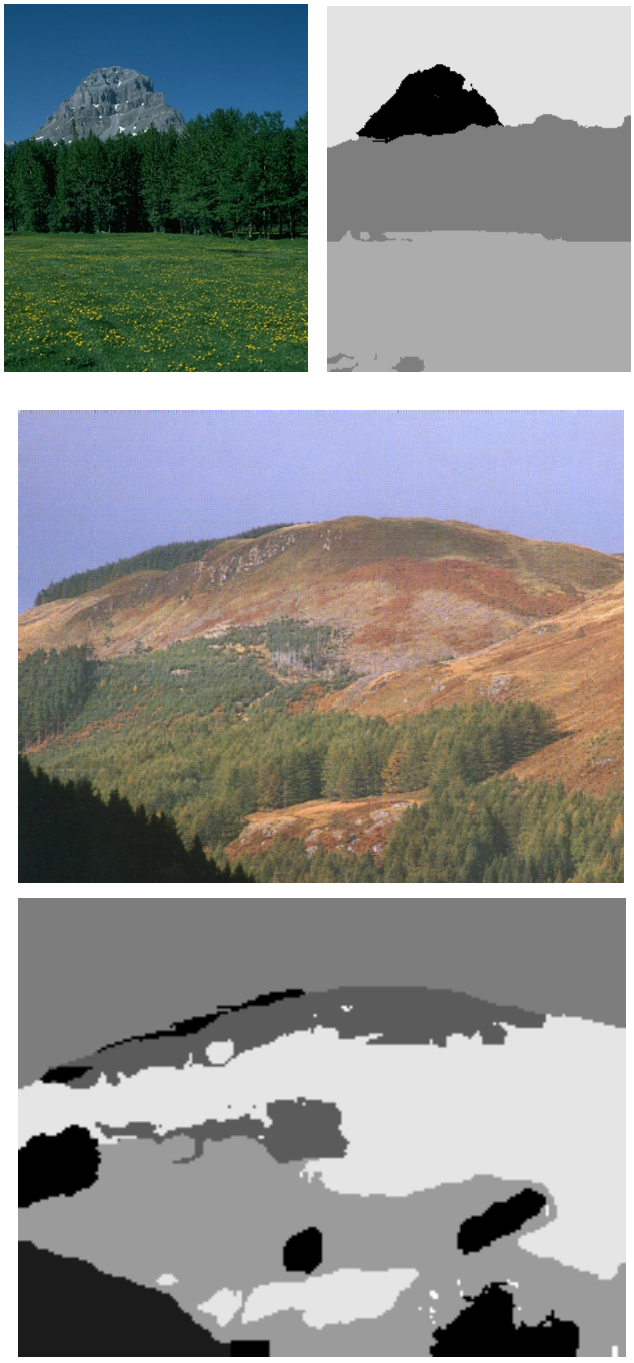

Fig. 8. *Landscape segmentations (CP and shifted matching)*