Learning A Stable Structure To Describe Dynamic Texture

Chuan Li and Peter Hall
Department of Computer Science
University of Bath
{cl249, pmh}@cs.bath.ac.uk

Abstract

We are interested in descriptors for dynamic textures that supports both further analysis and resynthesis applications. These tasks demand that the description encodes appearance and motion separably. This paper shows that a tree hierarchy which is built from nested image regions can be acquired via analysis of the first few frames of a video sequence. The tree is stable over space and time and leads to a measurably improved performance on the generic application of tracking the dynamic texture itself. The claim is supported by experimental data taken from a range of dynamic textures including trees and flowers. We conclude that both appearance and motion are better described using the stable structure rather than by a sequence of equivalent hierarchies each optimised for a single frame, because motion data helps to reduce clutter artefacts from trees built for static images.

1 Introduction

Dynamic textures, such as trees, water and smoke offer significant challenges to computer vision. They move quickly, change shape, parts appear and disappear. Within this two general categories can be recognised: those used for classification and those used for resynthesis — corresponding to the well know discriminative / generative model divide. Discriminative models are designed for recognition of the dynamic texture and include work such as [1]; In contrast, most of the generative models are used for image based rendering applications, [17] [10] or for editing [4], they give impressive results. There are also models such as [7] [8] which can synthesis for a variety of dynamic textures and can also be used for recognition.

Nonetheless, so far as we know, little of this prior art has given an explicit description for both the appearance and the motion of a dynamic texture. Yet works such as [21] [13] and [6] show separating appearance from motion is a powerful way to represent moving objects. The separation means that the video can be classified on either appearance or motion or both, or edited in a flexible way. For example, simplifying an object's appearance into a cartoon style, or magnifying motion for use by, say, an engineer studying the effects of stress. But such work assumes rigid bodies or, at least, soft bodies which makes it ill suited to the problems dynamic textures pose. For dynamic texture, the idea of making appearance and motion separable has been explored by [22] [23]. The authors introduce a generative model which represents an image as a superposition of linear bases such as Gabor or Laplacian; each moving element is represented as a group of several spatially adjacent bases; these elements are then tracked through the image sequence to get the dynamic motion. The elements tracked are very simple and occur often in an image. Consequently the tracker can become easily confused as is likely to give poor performance on, say, textures such as trees. In this paper, we provide a structure-based descriptor for dynamic texture. Our method does not only model appearance and motion as separable but also benefits users from a hierarchical coarse-to-fine description and a more stable acquisition of motion.

The appearance of texture is often described by statistical texton-based models. Textons were initially introduced as feature vectors that encode local appearance information [12] [20], and more recently [11] [19]as visually salient features. Usually the first and second order statistics are studied for texture recognition and classification. Although these models are designed for discrimination tasks, they are essentially very good simplifications of texture appearance.

Our representation is based on the well known maximally stable extremal region (MSER) descriptors [14]. Given a static image, an MSER tree will decompose its appearance into a hierarchy of nested regions. MSER trees have been used in many applications such as wide-baseline stereo matching and content based retrieval, painting too has been suggested as a possible application [9]. Unfortunately MSER trees are not stable over a sequence of frames that depict a dynamic texture, such as a tree blowing in the wind. In such a case the number of nodes, the appearance of nodes, and most particularly the structure of the hierarchy are all subject to temporal change.

In this paper, we show our adaption of MSER to motion, which is unique so far as we know. We track MSERs through video using standard methods for feature tracking — MSERs being features in our case. The problem of feature based tracking has been comprehensively studied. Both gradient descent searching [2] [3] and appearance matching [18] [13] are widely used. In order to improve the length of the tracking, probabilistic frameworks are also applied to predict motion in coming frames [5]. Compared to tracking moving objects in ordinary scenes, dynamic textures have their own challenges (Figure 3). Firstly, tracking features individually is extremely difficult because there is high similarity between their appearances. Additionally difficult tracking conditions such as occlusion happen more often. Secondly, MSER trees (and we conjecture other hierarchies) are unstable because similar features may move differently. Considering a tree in the wind, different branches may move differently — features appear to merge and split — so that overall global perception is complicated, approaching chaotic.

In this paper, we introduce two ideas that improve tracking performance for dynamic textures and which, simultaneously, enables us to build the kind of descriptor we seek. The first idea is a structure based tracking scheme based on a simple observation: large regions are usually more stable but less accurate to track, while small regions tend to be more accurate but less stable. We use the MSER hierarchy to get the advantages of both: we track from larger parent regions to smaller child regions. The child's motion will first be predicted by the parent, then corrected by local updating.

The instability of MSER over time leads to our second idea — we refine the MSER tree by splitting and merging nodes based on motion correlation, as defined in "motion magnification" [13]. The refined tree structure is more stable to motion, as shown by the empirical fact that it significantly improves the tracking performance. Moreover, the refining process also benefits describing appearance: false unions of regions which only exist for a short time will be deleted via splitting; while reasonable new unions of regions which are not detected in a particular frame(s) will be created by merging. In the end, we have a stable feature-based structure that gives us potentially fine control over dynamic textures by modelling their appearance and motion separately.

2 Describing Appearance of Dynamic Textures

Given an image (a frame), we construct a tree of MSERs using the method introduced by [14]. Broadly, a gray level image, I is thresholded with many threshold values to create a set of binary images $B(v)$, which are ordered using the value v. The “foreground” of any of these binary images are those regions below the threshold. Typically these regions are initially small in area, but in any case they all grow as the threshold rises and will merge together, eventually creating a single region. By connecting regions in images with neighbouring v values a tree is constructed. This tree can be pruned by identifying regions that change little with respect to threshold — these long runs of *maximally stable regions* can be replaced by a single representative. The result is a tree that describes the appearance of the image. Moreover, smaller regions are naturally nested into larger ones.

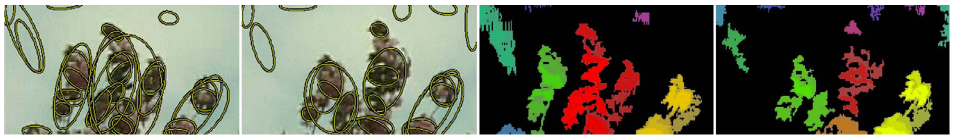

Figure 1: MSER tree for a dynamic texture (a cherry tree). The left two pictures show frames (the 1st and the 64th) with MSERs detected. The right two pictures show the MSER tree generated from these two frames. Nodes on a same MSER branch are coloured in the same hue while children have higher saturation and value. We can see that children are fully contained by their parent. We can also see the tree topologies are different.

We see the MSER tree from the point of view of moving from the larger to the smaller regions; that is the way in regions split rather than merge. We adopt standard terminology: a region that splits is here called a *parent* node, while those regions it’s splits into are *children*. We say that a child node is on a lower level of the tree than its parent. We take a *root* node to be any node that has no parent other than the whole image; root nodes exist on the top-most level of the tree. The *leaves* of a tree have no children, and can potentially exist on any level. Figure 1 shows the hierarchy of an MSER tree for a dynamic texture.

3 Describing Motion of Dynamic Textures

Now we have a description of appearance we wish to embellish it with a description of motion. In doing so we will be forced to alter the description of appearance. The result will be a tree structure that is stable in space and time. Each node contains both the appearance of some image region, and also the way in which it moves. The structure of the tree reflects the observed structure of the real object, using evidenced from over a time window.

To understand the problems facing us a little more, consider building an MSER tree for every frame of a video sequence. Because of the nature of dynamic textures the MSER tree in each frame will differ significantly. There will be no dynamic content to any of the trees, but the shape of regions will change between frames: some regions may merge and others split, which changes tree topology. Any tree drawn from a single frame will give a poor description of the dynamic texture over all, and may even be misleading within the frame it represents — since distinct real-world objects may appear to be one MSER region.

Our aim now is to stablise a single MSER that fits all frames in a sequence. We do this by tracking, correcting, and correlating. First we track the individual features that are MSERs. Next we use the MSER tree to predict and correct the motion of child regions, given their parent. Finally we correlate the motion of regions to automatically edit the MSER into a stable form. This final form is takes both appearance and motion into account. All of this is done over the first ten or so frames of a sequence, and the tree we obtain is shown to improve tracking in subsequent frames. Moreover, the true structure of the underlying object is likely to be better described because objects that are accidentally merged into a single region in one frame may become separated in another. We begin our account with a description of individual feature tracking.

3.1 Tracking Features Individually

The goal of tracking is to follow objects in a video. We are given a video sequence $I^t, t = 1 : n$, and wish to output the time dependent transform of a region in frame one. A region in any frame covers pixels in the set of locations $\mathbf{X}$ and has an appearance denoted $I^t(\mathbf{X}^t)$, these are MSER regions. They are visually salient features, the superscript t indicates a feature can change in time.

There are at least two ways to track features, which for brevity we will call *landmark* and *transform*, and are analogous to land-mark matching and transform searching, respectively. They have different characteristics. *Landmarking* requires two features in different frames be matched, which is fast but can be unreliable if the feature changes shape significantly. *Transforming*, on the other hand, more readily allows for shape changes but copes less well if the feature is fast moving. We say an object “moves fast” if its observed displacement is large compared to its observed size.

Dynamic textures contain features that change shape and which move fast. Consequently it is not clear *a priori* which method to use when tracking individual features. Rather, the method of choice is likely to be content dependent. Our general response is to track individual features using both methods and choose which ever is the better. Yet as we will explain below, there is good reason to use *transforming* when tracking a feature which is known to be a child in an MSER tree. Now, though we will describe each tracking method in turn.

The *landmark* method matches region $I^t(\mathbf{X}_i^t)$ and another in the next frame $I^t(\mathbf{X}_j^{t+1})$. We follow [5] in using a bounding rectangular window; but unlike them we measure error using normalised cross correlation. We match to the region giving the smallest error, thus

$$\widehat{\mathbf{X}_i^t} = \arg\underset{\mathbf{X}_j^{t+1}}{\underline{\min}}\left(NCC(Wnd(I^{t+1},\overline{\mathbf{X}_j^{t+1}}), Wnd(I^t,\overline{\mathbf{X}_i^t}))\right) \tag{1}$$

is our matched region. In this, $\overline{\mathbf{X}_i^t}$, $\overline{\mathbf{X}_j^{t+1}}$ are region centers. The function $Wnd(I,\overline{\mathbf{X}})$ extract a square window from a colour image I centered at location $\overline{\mathbf{X}}$.

Our second way to track features is to *transform* an initial template. This method has been comprehensively studied by [2], whom we follow here. We suppose that the region $I^t(\mathbf{X}_i^t)$ is a template, T_i^t , which is transformed to a new position in the successive frame. This transform can be searched for by the *Kanada Lucas Tracker* (KLT):

$$\mathbf{P}_i^t = \arg\min_{\mathbf{P}} \sum_{\mathbf{X}_i^t} [I^{t+1}(W(\mathbf{X}_i^t;\mathbf{P}) - I^t(\mathbf{X}_i^t))]^2 \tag{2}$$

in which $W(\mathbf{X}_i^t;\mathbf{P}_i^t)$ is the deformed region $I^t(\mathbf{X}_i^t)$ from I^t to I^{t+1}, and $\mathbf{P}_i^t = (p_1^t, p_2^t, p_3^t ... p_n^t)^T$ is the vector of parameters of the deformation. As before, we select the minima.

As mentioned above, features in dynamic textures change shape, which suggests we should update the template in each frame. This is true for both methods. When landmarking the length of a successful track can be increased by using the feature matched to, that is $\widehat{\mathbf{X}_i^t}$ as the next probe feature. When transforming, the newly deformed region becomes the next probe. But updating the appearance of the probe feature risks the unfortunate consequence of (what we call) “target drift”: the probe sooner or later changes so much that an entirely new real-world object becomes the subject of the tracker.

We resolve this problem (for both methods) by updating the appearance of the feature $I^t(\mathbf{X}_i^t)$, but as described in [15] in which the update is always made using the original feature. To explain briefly, suppose $\mathbf{P}$ is a transform (which for *landmarking* is readily estimated after matching as a translation). We compute an ”incremental transform” as

$$\Delta\mathbf{P}_i^t = gd \min_{\mathbf{P}=\mathbf{P}_{n-1}} \sum_{\mathbf{X}_i^t} [I^{t+1}(W(\mathbf{X}_i^t;\mathbf{P}) - T_i^1)]^2 \tag{3}$$

in which $gd\min_{\mathbf{P}=\mathbf{P}_{n-1}}$ means “ perform a gradient descent minimization ” starting at $\mathbf{P} = \mathbf{P}_{n-1}$. We apply this incremental transform to the template T_i^i to create a new template.

Before leaving the tracking of individual features we should note that in many works such as [18] the authors match regions represented by SIFT descriptors. The reason we do not use affine invariant descriptors here is that our aim is to track region in a time coherent frame sequence, rather than search for objects though a long movie. In our case, the regions are unlikely to have large variance in scale, and using an image window keeps information that can help track.

3.2 Structure based Tracking

In the previous section we explained how to track features independently of one another. Yet features in MSER trees are not all independent: children depend on their parents. This dependency can be used to advantage. The basic idea of structure based feature tracking is that large, parent regions are usually more reliable but less accurate to track, while small child regions tend to be more accurate but less reliable.

Under the assumption of dependency, it makes sense to track the large features first, then move down through the tree towards the smaller features. In particular, a child’s motion will be predicted by its parent. The default prediction we use is that the motion of child and parent are identical. This is corrected by local updating using the child’s appearances. In this case we need only use template tracking — because the motion of children relative to parents is usually small. In our experience, template tracking tends to be more accurate and reliable than landmark matching. So, this method yields gains in accuracy and reliability when compared to tracking individual features.

The situation is different for root nodes, because there is no parent to give a predicted motion. For root regions we use both the *landmark* and *transform* tracking methods. Thus for each root node we generate two possible motions, one for each tracking method — including the template update.

If there is only one root region, we must choose between the two solutions. We select the result which has a non-singular transform, and if both are non-singular that with the smaller RMS error between the template and the target; the target suffers an inverse transform prior to RMS measurement.

If there is more than one root region, we can improve tracking by considering neighbouring roots. This is based on the assumption that regions close to each other might move in a similar way. In this case, we build a graph to connect root regions. The track of a given node is then weakly supported by its neighbours. In this case, each neighbour will have two possible motions associated with it ("landmark" and "transform" solutions). Hence for a root node with N neighbours there is now a pool of $2(N+1)$ possible motions. We simply choose the best of these by minimising RMS distance in the manner of a single root node.

The result of structure tracking is an improved tracking result (Figure 3 and 4)— we can track features more reliably and more accurately for a longer period of time. But this relies on the MSER tree constructed for the first frame only. We can do much better if we reconfigure the tree to be optimal for a sequence of video frame, not just one frame. In the next section we explain how to do this.

3.3 Temporally Stable MSER Trees

So far we have used the MSER tree from a single frame to improve tracking. But if we modify the MSER to be stable over time we not only improve tracking performance, but are likely to develop a structure that is a closer reflection of the real underlying object, because accidental appearances tend to be unstable. To see this consider the fact that MSERs to merge and split, when in fact occlusion and dis-occlusion events are occurring. This is because MSER trees are appearance based, so accidental appearances cause the topology of MSER tree to change over time.

There is another reason, to update the MSER tree related to that above but a little more subtle. Closer analysis of the tracker so far shows that because root regions are large they appear to move slowly and can be tracked well. Child regions create error for one of two main reasons: (1) the gradient descent search for the child has fallen into a local minimum; (2) the motion of the child is not well predicted by its (putative) parent. The first case is recoverable, provided the child is truly attached to its parent. But the assumption that the observed motion of a child region follows the observed motion of its MSER parent can be violated if the child and parent are not connected in the real world.

Here we propose a method to refine the structure to be stable. A short sequence of video will be used for learning. An MSER tree is used to track the features in the video clip. Once tracking is complete the MSER is updated on the basis of motion correlation, which is measured as explained below. In particular, updating separates children from parents, if they are weakly correlated by motion. This creates a new tree with new roots — the separated children are given root status. Any children of the new roots follow their parent. The updated tree is used to track the same sequence once again, all previous data being discarded. This track-update sequence is iterated over until there is no further change to the MSER tree.

In a refinement, this iterative process is carried out level by level. That is, the first set of tracking and tree reconfigurations is limited to level one, just below roots. When level one is stable we then move to level two and so on. Of course, the level index of a region changes whenever a child is promoted to root status — but this change is trivially carried into the next iteration because each iteration is independent of the last. In this way we refine the tree with a coarse to fine strategy.

The resulting MSER tree is finally updated by grouping subtrees and deleting poorly tracked features. Tracking quality is once again measured using RMS error, as with single root nodes. Regions with sufficiently similar motion are grouped. See Figure 2 for an example. Motion similarity is measured using the same technique as when splitting, and is explained next.

The motion correlation measure we use is taken from "motion magnification" [13]. For each region, its instantaneous velocity, (u,v) , is represented as a complex number; $z = u + jv$, with $j = \sqrt{-1}$. The motion correlation between regions n and m is then given by

$$\rho_{nm} = \frac{|\sum_k z_{nk} z^*_{mk}|}{(\sum_k |z|^2_{nk} \sum |z|^2_{mk})^{1/2}} \tag{4}$$

in which the k subscripts indexes frames and z^* is the complex conjugate of z.

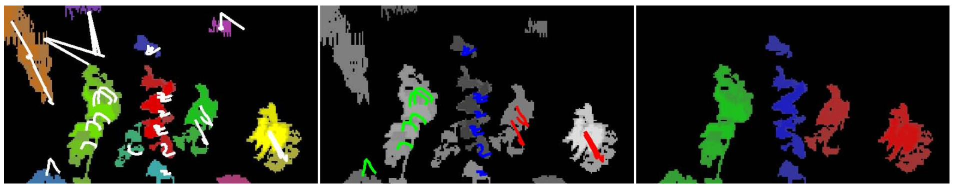

Figure 2: The temporally stable MSER tree. The left picture shows the result of splitting. At this stage, the tree has been split to be stable and the trajectory of each MSER region is drawn in white. The middle picture shows the result of grouping motion via correlation. Trajectories of grouped motions are drawn in the same colour. Regions whose motion correlates badly to all others are ignored — they are usually background noise. The right picture shows the result of merging — new, stable roots are created.

4 Experiments

In this section we empirically show the value of temporally stable trees to tracking. To do so we measure the quality of the tracker by the ratio between the number of successfully tracked features and the total number of features. A track is judged successful if the transform is non-singular and RMS between the initial feature and the (inversely) transformed target is small enough, our threshold is 0.015. The temporally stable tree may have a different number of features than the number of features in the initial MSER tree,

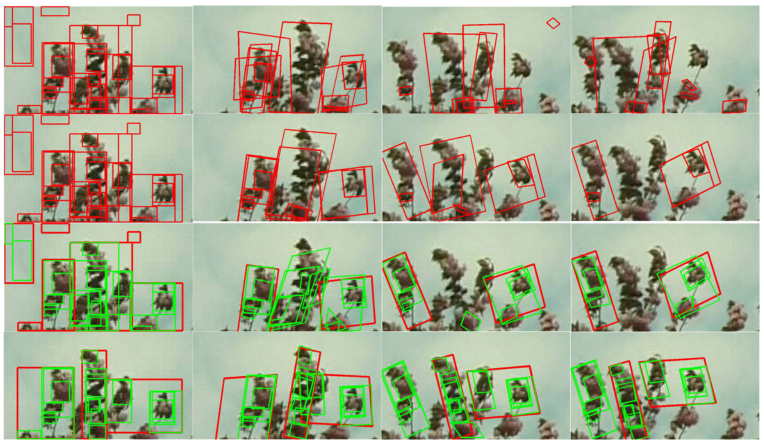

Figure 3: Tracking through a video of a cherry tree. Here we show the frame 1, 64, 128, 192. The top row shows tracking features individually by landmark matching. The second row shows tracking features individually by transform searching. The third row shows structure based tracking using an MSER tree generated from the first frame. Root regions are marked in red, their children are marked in green. The fourth row shows structure based tracking using a temporally stable tree.

so taking the ratio of the number of features in these two gives a normalised measure of success. This measure can be seen in Figure 4. For each dynamic scene in this paper, we use 10–20 frames to learn the temporally stable tree and each complete scene contains over 220 frames.

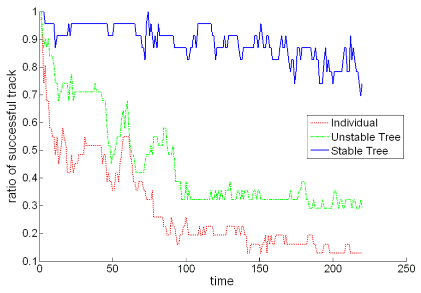

Figure 4: A quantitative comparison of the quality of different trackers. Quality is measured as the ratio of successfully tracked features. The red line is given when tracking features individually. The green line is given by tracking based on MSER tree from the first frame. The blue line is given by tracking using a temporally stable tree.

Figure 3 and 4 compares result of feature tracking for the Cherry tree scene. Figure 5 we show some experiment results on different dynamic textures. Some of the textures are from [16]. For each texture (the left column) shown here, we demonstrate the learned stable structure in the middle. The tracking results are compared on the right: individually feature tracking (red line) performs very unreliable; structure based tracking gives better results and learned stable structure significantly improves the overall track length.

Our method works well on dynamic textures such as trees and flower. Because their feature are relatively stable, and features are naturally structured. For example, leaves moves in the similar way as they are associated with the same branch. The method works not so well in two cases: 1) textures that have poorly defined features, such as smoke, flame, clouds; 2) textures with very unstable features for example, water.

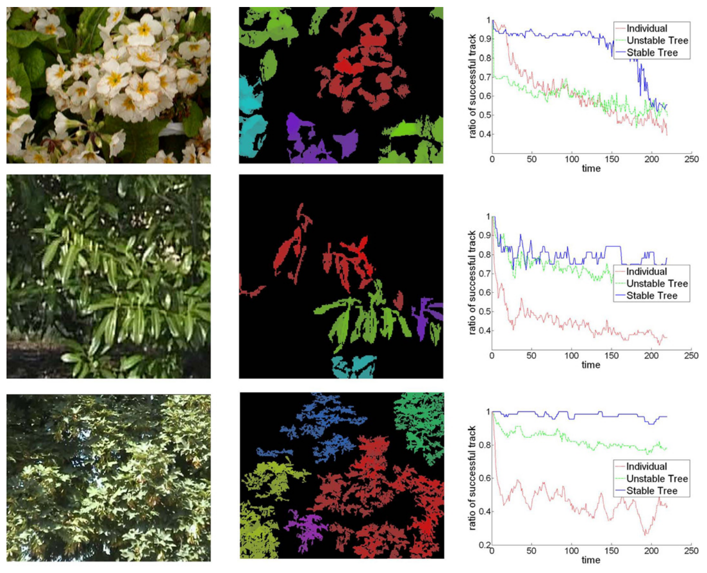

Figure 5: Experiments on different dynamic textures. Region groups are colour coded (Middle). Tracking based on temporally stable structure gives the best result (Blue lines in the right column). The method is seen to fail for the flowers (top row), when the motion becomes too chaotic and fast. In this case it degenerates to the performance of tracking individual features.

5 Conclusion

In conclusion, we introduced temporally stable trees as a descriptor for dynamic textures. The appearance is modelled by a MSER tree, and the motion by transforms from a parent to a child. Our descriptor encodes appearance and motion separately in the sense that applications can work with either one independently of the other.

This temporally stable tree is shown to benefits the performance of feature based tracking. We have shown experiment results on different dynamic textures including trees and flowers. Currently, our method does not work well on other forms of dynamic texture, such as water; other dynamic texture methods can handle some of these cases but, as we observed in the introduction, are less versatile than a representation like ours. We expect our representation to benefit not just tracking but a wide range of other applications too. Future work includes extending the range of dynamic textures we can cope with, and using a more sophisticated tracking framework, for example, a probabilistic framework. Nonetheless, the main contribution is the idea of a temporally stable tree descriptor that separates motion from appearance; it has already proven useful — we expect it will be useful elsewhere too.

References

[1] T. Amiaz, S. Fazekas, D. Chetverikov, and N. Kiryati. Detecting regions of dynamic texture. In *SSVM*, pages 848–859, 2007.

[2] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. In *IJCV*, volume 56, pages 221–255, 2004.

[3] V.G. Bellile, A.E. Bartoli, and P. Sayd. Feature-driven direct non-rigid image registration. In *BMVC*, 2007.

[4] K. Bhat, S. Seitz, J.K. Hodgins, and P. Khosla. Flow-based video synthesis and editing. In *SIGGRAPH*, volume 23, August 2004.

[5] A. Buchanan and A. Fitzgibbon. Combining local and global motion models for feature point tracking. In *CVPR*, pages 1–8, 2007.

[6] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. In *IEEE TVCG*, volume 11, pages 540–549, 2005.

[7] G. Doretto, D. Cremers, P. Favaro, and S. Soatto. Dynamic texture segmentation. In *ICCV*, pages 1236–1242, 2003.

[8] G. Doretto and S. Soatto. Editable dynamic textures. In *CVPR*, 2003.

[9] P. Forssén. Maximally stable colour regions for recognition and matching. In *CVPR*, pages 1–8, 2007.

[10] V. Kwatra, A. Schodl, I.A. Essa, and A.F. Bobick. Graphcut textures: image and video synthesis using graph cuts. In *SIGGRAPH*, volume 22, pages 277–286, 2003.

[11] S. Lazebnik, C. Schmid, and J. Ponce. Affine-invariant local descriptors and neighborhood statistics for texture recognition. In *ICCV*, page 649, 2003.

[12] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. In *IJCV*, volume 43, pages 29–44, 2001.

[13] C. Liu, A. Torralba, W. T. Freeman, F. Durand, and E. H. Adelson. Motion magnification. In *SIGGRAPH*, pages 519–526, 2005.

[14] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *BMVC*, volume 1, pages 384–393, 2002.

[15] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. In *IEEE Transactions on PAMI*, volume 26, pages 810–815, 2004.

[16] M. Huskies R. Pteri and S. Fazekas. Dyntex: A comprehensive database of dynamic textures. In *http://www.cwi.nl/projects/dyntex*, 2006.

[17] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *SIGGRAPH*, pages 489–498, 2000.

[18] J. Sivic, F. Schaffalitzky, and A. Zisserman. Object level grouping for video shots. In *IJCV*, volume 67, pages 189–210, April 2006.

[19] P. Southam and R. Harvey. Towards texture classification in real scenes. In *BMVC*, 2005.

[20] M. Varma and A. Zisserman. Texture classification: are filter banks necessary? In *CVPR*, pages II: 691–698, 2003.

[21] J. Wang, Y.Q. Xu, H.Y. Shum, and M. F. Cohen. Video tooning. In *SIGGRAPH*, pages 574–583, 2004.

[22] Y.Z. Wang and S.C. Zhu. A generative method for textured motion: Analysis and synthesis. In *ECCV*, pages 583–598, 2002.

[23] Y.Z. Wang and S.C. Zhu. Analysis and synthesis of textured motion: Particles and waves. In *IEEE Transactions on PAMI*, volume 26, pages 1348–1363, October 2004.