

Creating Video Animations Combining Stochastic Paintbrush Transformation And Motion Detection

Levente Kovács

University of Veszprém, Department of Image
Processing and Neurocomputing, H-8200
Veszprém, Egyetem u. 10, Hungary
E-mail: levente.kovacs@freemail.hu

Tamás Szirányi

Analogical Comp. Lab., Comp. & Autom.
Inst., Hungarian Academy of Sci., H-1111
Budapest, Kende u. 13-17, Hungary
E-mail: sziranyi@sztaki.hu

Abstract

In this paper we propose a method for creating animation and animation-like video sequences from an ordinary video recorded by any means available. The method is based on the Paintbrush Transformation earlier patented by our Department [7], and on different motion detection algorithms [3,4,5]. One of the goals of the method is to obtain animations, cartoon-like outputs from a usual camera-recorded image sequence. The method inherits the properties of the paintbrush transformation method like well-defined contours, acceptable distortion, and a painting-like view with no fine details below a limit. The resulting output is a series of video frames stored as brush-strokes and motion data between the frames, compressed for size reduction.

1. Introduction

There are numerous image and video coding and compression methods available nowadays. Images can be interpreted in several ways by decomposition into basic functions: strokes [8,9], fractals [10], etc. Each of these is natural in some sense. The idea behind paintbrush transformation techniques was to transform an image so that it gives the sensation of a painting, the painting process being a simulation of the real painting process by using simplified artificial strokes. The parameters of consecutive strokes can then be used for image description and compression as well.

Taking our stochastic paintbrush transformation as a starting point [1,2] we tried to develop a method of motion picture transformation with the goal of obtaining animation-like outputs from ordinary video inputs, which have the properties of the original paintbrush transformation.

The method is based on the following idea: besides the full-frame paintbrush transformation of the key-frames, the transformation is based on the motion in-

formation obtained from motion detection between consecutive frames. By using the optical flow data obtained, the areas where motion occurred are re-transformed and the next transformed frame is obtained from the previous transformed frame and the processed motion areas.

The output of the transformation is an intermediate format, in which the frames of the output video are stored as series of brush-strokes, and the motion data is also stored between the frames, run-length-encoded. The whole intermediate format undergoes a Huffman-encoding [6], obtaining smaller output sizes. By decompression and repainting the frames of the transformed video - using the stored stroke-series and the motion data - can be reconstructed and recoded into any available video format.

2. The Paintbrush Transformation

The development of paintbrush transformation [1] had the goal of achieving an automated method which imitates, simulates the painting process of a real painter, to obtain a picture similar to a real painting, where the purpose of the painter is to portray something which looks like to be a real scenery.

During the transformation process the picture is being constructed randomly with bigger brush-strokes, then it gets refined with smaller and smaller strokes, and the picture increasingly resembles the model.

The transformation starts with bigger strokes then refines the actual picture with smaller strokes. In the original algorithm during the refining process a stroke with given size, orientation and color gets on the picture if it's placement improves it, makes the picture converge more to the model. The coordinates of the next examination were originally picked by exhaustive stochastic search, then an optimized model was developed, in which the exhaustive search was replaced by a decision based on the approximation of the error caused by placing the strokes, where the target density is dynamically

fit during the process and the characterizing densities are changing through the iterations.

An important feature of the transformation is that there are sharp edges and well-defined patterns and areas although we have no a priori information about the textures, contours or shapes on the picture.

The generated image can be described by a series of strokes, which can be used for moderate compression, a stroke being described by seven bytes of data (id, position, color-information).

3. Application On Motion Picture

The first step on the way of producing an animation-like output was to paintbrush-transform every frame of an input video. However, because of the stochastic nature of stroke-placement this method generated vibrating image sequences, and the output was quite large because of the large number of strokes necessary to describe the frames.

It seems that the motion detection-based approach gives the answer. The concept of the method is that we only fully transform the key-frames. Between the key-frames we use the optical flow data obtained from motion detection algorithms, and transform only the motion areas. After eliminating those strokes, which are covered by others, we get much fewer strokes and thus much smaller output – compared to transforming the whole image.

4. Motion Detection

We use basically two motion detection algorithms: gradient-based and block matching motion detections.

The usual starting point for velocity estimation [3,4] is to assume that the intensities are shifted from one frame to the next, and that the shifted intensity values are conserved:

$$f(x, y, t) = f(x + u_1, y + u_2, t + 1)$$

A path $(\mathbf{x}(t), \mathbf{y}(t))^T$ along which intensity is equal to a constant c must satisfy:

$$\frac{d}{dt} f(x(t), y(t), t) = 0 = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} + \frac{\partial f}{\partial t} \frac{dt}{dt} = f_x u_1 + f_y u_2 + f_t$$

Since it's impossible to recover velocity given just the gradient constraint over a single position, we combine constraints over a region. To measure the extent to which the gradient constraints are not satisfied we square the constraints and sum them:

$$E(u_1, u_2) = \sum_{x,y} g(x, y) [u_1 f_x(x, y, t) + u_2 f_y(x, y, t) + f_t(x, y, t)]^2$$

where $g(x, y)$ is window function that is zero outside the neighborhood within we use the constraints.

We solve for u_1 and u_2 :

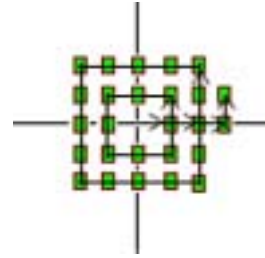
$$\frac{\partial E(u_1, u_2)}{\partial u_1} = \sum_{x,y} g(x, y) [u_1 f_x^2 + u_2 f_x f_y + f_x f_t] = 0$$

$$\frac{\partial E(u_1, u_2)}{\partial u_2} = \sum_{x,y} g(x, y) [u_1 f_x f_y + u_2 f_y^2 + f_y f_t] = 0$$

$$M = \begin{bmatrix} \sum g f_x^2 & \sum g f_x f_y \\ \sum g f_x f_y & \sum g f_y^2 \end{bmatrix}$$

$$u \approx -M^{-1}b$$

In the block matching motion detection [3,4,5], we search for a block with a given radius in an also given radius area around it, along a spiral path. This method is more sensitive to non-translation motion.



The optical flow field data obtained with these motion detection methods are used when transforming the motion areas of the frames. The motion data calculated are stored as a two-dimensional array, each element containing the position where that element will move when the motion occurred.

5. The Algorithm

The main steps of the algorithm are as follows:

1. Setting of parameters (key-frame frequency, type of motion detection to use, other parameters).
2. Paintbrush transformation of frame $F(i)$ to $F'(i)$ and writing of $F'(i)$ into the intermediate format.
3. Motion detection calculation between $F(i)$ and $F(i+1)$.
4. Transformation of the areas where motion occurred using the optical flow field data calculated at the previous step.
5. Writing the partially transformed frame data onto $F'(i)$, generating $F'(i+1)$ and writing it into the intermediate format along with the motion data.

6. If the next frame will be a key-frame then increase i and jump to step 2., otherwise to step 3.
7. The steps of the transformation used at step 4 are as follows:
8. Choose stroke-set. If last one, go to step 12.
9. Convolution of motion areas with the strokes of the set (color data estimation).
10. Generation of error image between the actual step of transformation and the original area.
11. Blurring the error image, generation of histogram from absolute values of error image.
12. Picking the coordinates and orientation of the next proposed stroke over the motion area. Color data taken from step 2., orientation data from the generated motion data.
13. If the error on the actual position isn't lower than a threshold value, and the stroke lowers the error value – more then a threshold -, then place the stroke.
14. Calculating the error image between the original and the image got after placing the stroke, over the motion area. If there is improvement over a threshold value, then accept.
15. If we are over a specified number of tries, then check the difference of the actual image from the image got from the previous iteration. If it's under a threshold, we had no real improvement.
16. If there are more strokes in the actual set and there was some improvement at step 8., then go to step 5.
17. Clear those strokes, which are completely covered by other strokes.
18. Jump to step 1.
19. Generate the next transformed frame from the painted motion area and the transformed image obtained from the previous iteration step. The function realizing this algorithm returns with this frame.

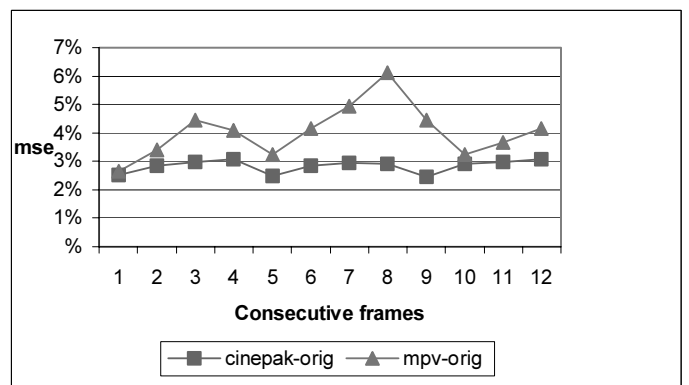
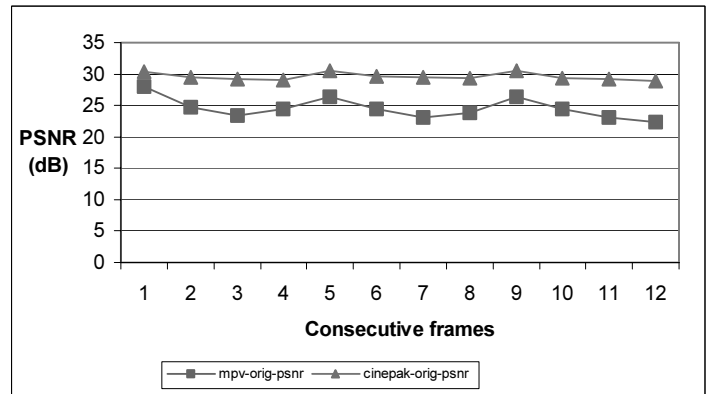
After painting we only have to consider those areas where motion has occurred, and place these areas over the previously generated transformed frame. Then we have to write the frame to the intermediate format file, along with the motion data – all required for final video reconstruction.

6. Comparison of Compression methods

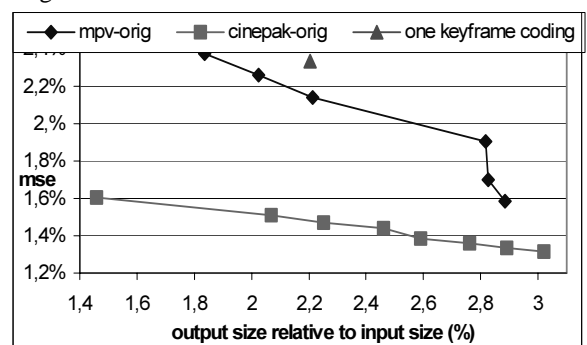
For comparison we choose the Cinepak codec, because it has long been the most popular codec for AVI files, and because it provides good replay speed and image quality. It is a middle-class middle-quality codec, that's why we compared out under-development format

to it. It must be note that our method is in its infancy; a lot can be done to better compare it with professional codecs.

The next two graphs show the MSE data from frame to frame of a Cinepak-generated output and our output video relative to the original input video.



The next graph shows the error between numerous transformed videos and the original input video – and simultaneously the respective Cinepak-coded videos and the original one.



7. Conclusion

Some example frames can be found in Figure 1. The method we are developing is a way of generating anima-

tions from real-life scenes, in motion picture. The algorithm works like fractal-based algorithms (and like the Cinepak codec): slow encoding, fast decoding, requiring relatively small bandwidth, with good compression ratios. There are still problems to be solved, but hopefully the method will turn out to be usable and useful.

References

- [1] T. Szirányi, Z. Tóth: "Random Paintbrush Transformation", *15th ICPR, Barcelona, IAPR & IEEE, V.3*, pp.155-158, **2000**
- [2] T. Szirányi, Z. Tóth: "Optimization of Paintbrush Rendering of Images by Dynamic MCMC methods", *Lecture Notes Comp. Sci.*, Vol. LNCS 2134, pp.201-215, 2001
- [3] Simoncelli, Eero P., "Distributed Representation and Analysis of Visual Motion", 1993
- [4] Heeger, David. J., "Notes on Motion Estimation", *Psych 267/CS 348D/EE* p. 365, 1998
- [5] J. Barron, D.J. Fleet, "Performance of Optical Flow Techniques", *CVPR*, 1992
- [6] Wells, Richard B., *Applied Coding and Information Theory for Engineers*, Prentice-Hall, 1999
- [7] T.Sziranyi, Z. Toth, I. Kopilovic, "Paintbrush Transformation", *Hungarian Patent Office*, No.: P00 00041, 2000
- [8] H. H.S. Ip, H. T. F. Wong, "Generation of Brush Written Characters with fractal Characteristics", *ICCPOL, 17th Int. Conf. on Computer Processing of Oriental Languages*, pp. 156-161, Hong Kong, April, 1997.
- [9] H. H S Ip, H. T F Wong, "Calligraphic Character Synthesis using Brush Model", *CGI'97, Computer Graphics International conference*, pp. 13-21, 1997
- [10] Y. Fisher. (ed.), *Fractal Image Compression*, Springer Verlag , 1994



Figure 1: some example frames from the movies