Creating 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 Computing Laboratory, Comp. & Automation 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 about any ordinary video recorded by any means available. The method is based on the Paintbrush Transformation earlier invented, developed and 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, 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. This output can be converted to any available video format by decompression, frame-reconstruction and recompression.

1. Introduction

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.

A. Hertzmann in [11] presented a method of creating images with painting-like appearance from photographs, and an approach to designing illustration styles. Our paintbrush transformation method [1, 2, 7] differs in many ways from this approach. We paint images with rectangular-shaped stroke-sets in a coarse-to-fine way, optionally following edges on the image, but not trying to establish or mimic different artistic styles or methods, only to approach the visual quality of the input image sequence, being able to represent the images with the strokes used. Trying to near the input quality and following the guidelines of the [1, 2] method we achieve visually pleasant representations of the input images, and at the same time we are able to store the input sequence in reasonably compressed format.

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 information 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 a so called intermediate format, in which the frames of the output video are stored as series of brush-strokes, Huffman-encoded [6] and the motion data is also stored between the frames, run-length-encoded.

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 real scenery. 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 main concepts of the algorithm were: it should have sharp edges at any level of image-construction; there are no fine details below a limit; there are sharp edges at the finest level as well; from a given distance the image must give the same visual scenery as the original

The transformation starts with bigger strokes then refines the actual picture with smaller strokes. During the refining process a stroke with given size, orientation and color gets on the picture if it's placement 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 [2].

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 and 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. 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.

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 $(x(t),y(t))^T$ along which intensity is equal to a constant c must satisfy:

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

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:

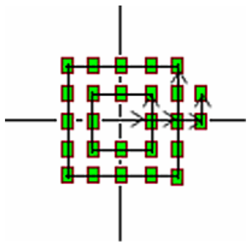
$$E(u_1,u_2) = \sum_{x,y} g(x,y)\left[u_1 f_x(x,y,t) + u_2 f_y(x,y,t) + f_t(x,y,t)\right]^2$$

Here $g(x,y)$ is Gaussian window function. We solve for $u1$ and $u2$.

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

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

$$M = \begin{bmatrix} \sum g f_x^2 & \sum g f_x f_y \\ \sum g f_x f_y & \sum g f_y^2 \end{bmatrix} \qquad u \approx -M^{-1} b$$



In the block matching motion detection [3, 4, 5] used we search for a block with a given radius in an also given radius area around it, along a spiral path (Fig.1).

The optical flow field data obtained with these motion detection methods is used when transforming the motion areas of the frames.

5. The Algorithm

The main steps of the algorithm are as follows:

1. Setting of parameters (key-frame frequency, type of motion detection to be used and other parameters like block radiuses, search ranges, thresholds, etc.).
2. Paintbrush transformation of frame F(i) to F'(i), Huffman-compressing 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.

The steps of the transformation used at step 4 are as follows:

1. Choose stroke-set. If last one, go to step 12.
2. Convolution of motion areas with the strokes of the set (color data estimation).
3. Generation of error image between the actual step of transformation and the original area.
4. Blurring the error image, generation of histogram from absolute values of error image.
5. 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. Optionally take in consideration the morphologically after-processed difference image of the two frames (useful in high-motion old-style cartoons).
6. 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.
7. 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.
8. 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.
9. If there are more strokes in the actual set and there was some improvement at step 8., then go to step 5.
10. Clear those strokes, which are completely covered by other strokes.
11. Jump to step 1.
12. Generate the next transformed frame from the painted motion area and the transformed image obtained from the previous iteration step. The function realizing this algorithms 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 the frame gets written into the intermediate format file, along with the motion data – all required for final video reconstruction.

6. Compression, Comparison and Time Data

For real-life video comparison we chose the Cinepak codec, because it has long been the most popular codec for AVI files, and because it provides good replay speed and moderate quality.

Fig. 1 shows the MSE data from frame to frame of a Cinepak-generated output and our output video (MPV – motion-painted video; it contains the Huffman-coded frames stored by the strokes it consists of, with the RLE-coded motion data between them) relative to the original input video at approx. the same PSNR. But in our case MSE values don't reflect image quality very well.

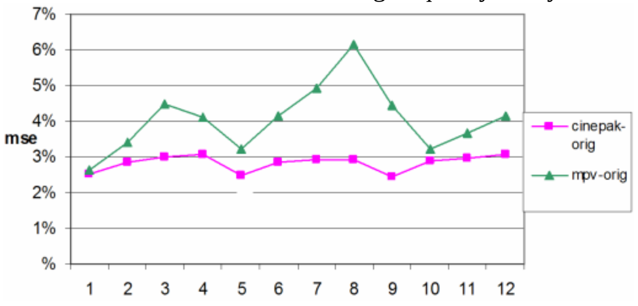


Figure 1

We present some non-photo-realistic video-processing results, comparing our method and output to existing codecs. Some frames from painted videos are presented below (Fig. a1-a9, b1-b3, c1-c3).

Fig. 2 shows the comparison of the output sizes of our processed output format, and some existing video codecs, and some variations of them (parameter modifications). The codecs used were Cinepak (2nd to 5th) Indeo (9th) and some MJPEG codecs with different parameters. The painted output is in the first column of both graphs.

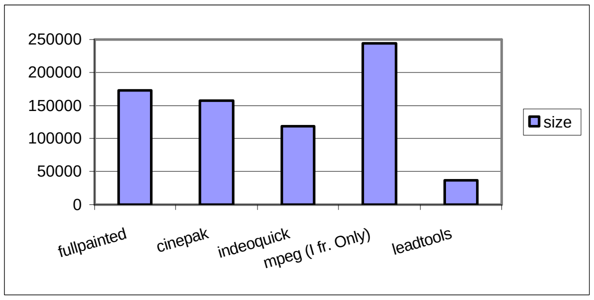


Figure 2

Fig. 3 shows the corresponding MSE values (for approx. identical PSNR values).

Table 1 contains data about the differences of original and processed output video motion fields (MSE between the motion field calculated on the original and on the painted video).

Frames	MSE
0-1	0
1-2	0.00955
2-3	0.00908
3-4	0.00341

Table 1

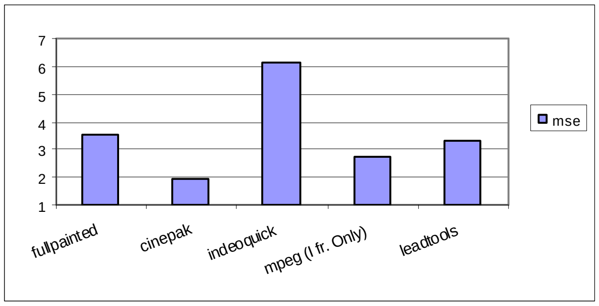


Figure 3

7. Conclusion

One major issue we still have to deal with is speed. We are working on this [2], also planning to build a distributed calculation framework, which is under development, with good prospects.

The method we are developing is a way of generating animations from real-life scenes, and a method for representing and storing cartoons in a way that still has a pleasant visual quality, with the option of moderate compression, in a way that reflects the artistical methods of creating such images. The algorithm resembles fractal-based algorithms: slow encode - fast decode, requiring relatively small bandwidth, good compression ratios (moreover in the way of building the painted image). Our transformation also can be joined to some interesting results of matching pursuit video coding [12]. Possible connections and implications with fractal compression will be considered in the future. Having the mentioned features and prospects, we try to implement our method into practical coding systems to get more reasonable animated image quality.

8. 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 (2001- Springer Verlag, Vol. LNCS 2134, pp.201-215)
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 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. Hungarian Patent Office, No.: P00 00041, 2000
8. H.H.S.Ip, H.T.F. Wong, Generation of Brush Written Characters with Fractal Characteristics from True-Type Fonts (ICCPOL, p. 156-161, 1997)
9. H.H.S.Ip, H.T.F. Wong, Calligraphic Character Synthesis Using Brush Model (CGI'97, p. 13-21, 1997)
10. Y. Fisher, Fractal Image Compression (Springer-Verlag, 1994)
11. Aaron Hertzmann, Painterly Rendering with Curved Brush Strokes of Multiple Sizes (SIGGRAPH, 1998)
12. R. Neff, A. Zakhor: Matching pursuit Video Coding, IEEE Tr. CAS VT, V.12, No.1, pp.13-39, 2002

Figure a1

Figure a2

Figure a3

Figure a4

Figure a5

Figure a6

Figure a7

Figure a8

Figure a9

Figure b1

Figure b2

Figure b3

Figure c1

Figure c2

Figure c3