Research-Document

ViProII

Ali Eghdamian
Jürgen Platzer
Christian Wagner

Content

Introduction and Goal Definition

Over the past years several techniques for non-photorealistic rendering of images have been introduced. In **[1]** a method was presented to compute a painted image from a photograph of the real world. This algorithm can also be applied on videos by rendering each frame separately. Using this approach the correlation between successive images of an image sequence is not considered. Thus noise or changing light conditions in the video material as well as used randomness in the painting algorithm can cause changes between frames of the rendered output that are not intended and yield to minor quality. Therefore adjusted techniques for image sequence rendering are needed to solve these problems. Hertzmann **[3]** proposed several ways to overcome those difficulties on image sequences, so for example areas with minimal changes during the animation were not painted again to eliminate the flickering effect introduced by noise in the source video.

The purpose of the ViProII project is the design of an algorithm that creates a painted version of an image sequence and explores new ways to incorporate the motion and the correlation captured in successive frames of a video. Based on an implementation of the Painterly Rendering algorithm that renders an image sequence frame by frame described in **[1]**, an alternative approach of rendering videos should be realized.

The first part of this document describes the results of the research and analysis phase, where the drawbacks of rendering a video frame by frame are discussed and our ideas to overcome those drawbacks are outlined. At the end of this section the decision for one of the proposed approaches is briefly stated.

The second section explains the different iterations during the implementation of our algorithm. An iteration represents the result of a programming period, after that some test videos are created to find errors in the implementation as well as necessary improvements of our idea. Thus those iterations represent different states of the research process and the discussed results show the problems we came across. The errors and difficulties during the work are also illustrated by sample renderings. Finally the last iteration that represents the completed program is discussed and analyzed.

The report ends with the conclusions attained during the project and a short description of ideas and future prospects that would improve our algorithm.

Research and Analysis

The libraries on which the ViProII project builds up were provided to us in the form of a C++ library. The Painterly Rendering algorithm uses methods from the OpenCV library, works step by step (i.e. frame by frame) and doesn't consider correlations between frames of an image sequence. One of the major tasks of the ViProII project will be the expansion of the algorithm to work on image sequences (i.e. animations) to achieve an increase in the qualitative performance of the algorithm. For this work the interface for the implemented Painterly Rendering algorithm was provided, so that no direct access to the sources was available.

Thus the first task was to analyze the effect(s) of the original algorithm to gain insight about strengths and weaknesses of rendering an image sequence frame by frame. To accomplish the analysis we created test videos. Some of the videos were filmed with a camera others were taken from the internet or from movies. These videos were carefully chosen to examine examples of light changes and fast movements. Also different styles of videos were chosen. The original algorithm was tested on animes, computer generated animations as well as on filmed material.

The main drawbacks of rendering frame by frame were the flickering of static backgrounds and the constantly changing outline of moving objects. The flickering effect is introduced by noise in the original videos as well as by the random painting order of the strokes for each frame. Therefore in consecutive frames different strokes with slightly different shapes and colours are painted last and are thus visible. The changing outline of moving objects is caused by the fact that in each frame a different set of strokes represents the border of the object.

The next task was to look more closely on the provided algorithm and find ways to eliminate the problems we had found in the video analysis. The main interest was to find out by which primitives a painted image is created and which major abstractions were made to imitate a painter's work. For the latter the concept of using layers is introduced. The idea is to paint a rough version of the image with strokes of a large brush. Afterwards the picture is refined by adding details with thinner brushes. The Painterly Rendering algorithm incorporates this concept by overlaying layers, which represent stroke sets with a certain brush size. The first layer contains the thickest strokes. The successive layers contain thinner strokes compared to their predecessors. The layers are then painted in descending order of their stroke size. The painting order within a layer is chosen randomly, since the use of a sequentially painting order leads to poor results.

The strokes are painted using parametric curves, which are defined through control points that influence the shape of the curve. By using those control points the so called curve points are calculated. Curve points are the positions through which the curve passes. The recalculation of the curve points is quite expensive therefore they are cached.

Based on this information four different approaches to overcome the drawbacks of rendering frame by frame were proposed. To try new ways of dealing with the challenges of non-photorealistic rendering of videos, no already known concepts, like the optical flow, were considered. The following four concepts were proposed:

1. Morphing of splines

For this approach the *(k*i + 1)*-th frames (keyframes) of an image sequence are rendered with the Painterly Rendering algorithm, where *i = 0, 1, 2, ...* The k missing frames are generated by morphing the strokes of the neighbouring keyframes to each other. For the morphing a correspondence between strokes of consecutive keyframes has to be established.

Strokes correspond to each other in different frames, if they have nearly the same position and colour. Therefore a similarity measure between strokes of keyframe A and strokes of the successive keyframe B has to be computed. To reduce the computational effort only a subset of strokes of keyframe B is considered for the correspondence calculation of a certain stroke in keyframe A. The subset contains only those strokes that have control points within the bounding box spanned by the control points of the current stroke of keyframe A, for which the correspondence is calculated.

The similarity measure depends on the colour difference and the distance of the control points of the two splines representing the strokes. Therefore the following structure for the similarity measure could be applied:

$$E(a,b) = X * colourdifference(a,b) + Y * distance(a,b),$$

where *a* and *b* represent the two strokes that should be compared, *X* and *Y* are constants that define the importance of the colour difference and the distance of the control points respectively. The colour difference can be evaluated according to the following formula:

$$\begin{aligned} colourdifference(a,b) = ((&colour(a).red - colour(b).red)^2 + \\ &(colour(a).green - colour(b).green)^2 + \\ &(colour(a).blue - colour(b).blue)^2))/((255^2) * 3) \end{aligned}$$

The normalization yields to colour difference values in the range [0,1], so that the difference value between black and white is 1. The distance between two strokes is expressed as follows:

$$distance(a,b) = mean(dist(contrpoint(a_i), contrpoint(b_i))/diag(searcharea)),$$

where *dist* represents the euclidean distance between two coordinates in 2D and *diag(searcharea)* represents the length of the diagonal of the search region for corresponding splines. To assure that every stroke in keyframe A is mapped to a different stroke in keyframe B, only still unused strokes are set in correspondence.

After calculating the correspondence the strokes have to be morphed. Therefore a morphing of the control points is applied. For this task stroke pairs with different numbers of control points have to be considered.

2. Modification of the Painterly Rendering algorithm

In this approach the Painterly Rendering algorithm should be rewritten, so that the creation of the strokes does not depend on a single frame. In the original algorithm the strokes are created by utilizing the gradient information of an image. For the modification the median of the gradients of *n* successive frames in an image sequence could be used instead. This can reduce the noise in the case of small changes from one frame to the next.

One problem for this modification could be the reduced significance of edges. If the stroke creation is based on several frames the borders of moving objects would be split in several borders and thus may tremendously change the appearance of the strokes. Further drawbacks of this approach are jumps in the median function over time. This can be demonstrated with the following example:

Table 1: Number of used frames n = 3

frame	gradient	median	mean
1	5	5	6,66
2	11	10	8,33
3	4		
4	10		

Although the median is robust and neglects outlying values, the changes of the median over time can be very high.

3. Updating heuristic

In the third proposal the first frame is rendered with the Painterly Rendering algorithm. The successive frames are painted by updating the already existing strokes of the preceding frame. The new shape of the stroke is determined by a modification of the stroke generation method of the Painterly Rendering algorithm. Starting with the first control point of a given stroke the gradient of the current frame is examined. As long as the gradient direction is within a tolerance range of the direction from the current control point to the next control point, the latter is updated accordingly. Problems will arise if the tolerance is exceeded. In this case the creation of a new stroke should be considered. The new colour of the stroke is the mean colour of the pixel area that is covered by the stroke.

The major drawback of this method is that all frames depend on the first rendered frame. If this frame is a bad representation for the scene many inconsistencies will be introduced. Furthermore areas will appear where no stroke is placed. To compensate this dissonance a heuristic has to be invented to manage the creation of new strokes.

4. Self organizing strokes

As in the previous idea the first frame is created with the Painterly Rendering algorithm. Afterwards a heuristic is applied so that the strokes organize themselves according to the gradient information in the new frame. Rules define which colour and control point positions the strokes may have. Further it is indicated if a stroke should vanish or it should clone itself. In this abstract idea the main task would be the formulation of the behaviour of these "self organizing" strokes. In addition there should be some parameters the user can tune to get a different appearance of the animated painting.

Determination of the algorithm to implement

As the second two improvement suggestions were very abstract, the choice, which approach should be realized, had to be made between the first two proposals. While the second proposal offered a solution with less complexity for the implementation, the first one had a higher potential for further improvements and in general a much more interesting and mature approach. After judging the assets and drawbacks the team finally decided to take the first improvement as the subject for this work.

Design and Implementation

Iteration 1

Keywords: base program, morphing, property file

This is the first runable version which can read properties from an external file. In this file the name of the image sequence and frame range to be processed is specified. It is possible to specify how many frames of the chosen image sequence have to be rendered with the morphing approach. Keyframes between the morphed frames are rendered with the original Painterly Rendering algorithm. The property file also defines various attributes (→ *see Appendix*) like the number and size of the brushes or the minimum and maximum length of a spline. The variable *Underpaint* provides the opportunity to choose the original image as background for the render process. The coherence between strokes is established, if their first control point has exactly the same 2D coordinates.

To analyze the deficiencies of this iteration **Figure 1** depicts the first frames of two image sequences. **Figure 1(a)** shows the original video material that has been taken from the motion picture "The Matrix". The reason for selecting this part of the movie was to study the behaviour of the algorithm during changes caused by movements of large image elements (fire). **Figure 1(b)** presents one of the first renderings of the morphing approach. The rendering was done using the stroke sizes 8, 4 and 2.

Figure 1(a): Original image sequence taken from the motion picture “The Matrix”

Figure 1(b): Rendered sequence (Morphing approach)

This example points out two major problems of the morphing approach. Because there are big content changes between two neighbouring keyframes the correspondence calculation breaks down and a blending effect is introduced, where strokes of the dark background adjust their colour to the bright foreground. This implies that the morphing in this state is not capable to capture fast movements. The second drawback is that the morphed frames contain black holes. They are created because the coherence between the layers is lost. Thus the small strokes do no longer correct errors of the rough sketch of the frame drawn by the big strokes.

Figure 2(a): Original image sequence taken from the motion picture “Batman”

Figure 2(b): Rendered sequence (Morphing approach)

Besides of the mentioned errors above discovered during the first examination, there were also some good results in the rendered outputs of the morphing procedure, as shown in the next figure. The original images of the sequence **Figure 2(a)** are taken from the motion picture "Batman", the latter sequence **Figure 2(b**) shows the results obtained through the morphing algorithm. The tracking and the overall correspondence in this example is much better as in the previous "Matrix"-sequence because of the minor changes between the consecutive frames.

Iteration 2

Keywords: window size, weighted colour and distance, correspondence calculation, difference images (additional tool)

To reduce the number of stroke pairs that are considered for the correspondence calculation, a search region is introduced. Thus not all strokes in the whole frame are considered for the calculation of the dissimilarity measure. Up to this iteration the search for a corresponding stroke was done in the whole image. In this iteration the correspondence search calculation is done within an area around one control point of the current stroke. This search region has a “windowsize” that can be specified by the property file.

Correspondence Calculation: In the first iteration there was no correspondence calculation. Corresponding strokes were detected by checking if their first control points have the same coordinates. If this was the case the strokes were morphed to each other. Otherwise the stroke had no correspondence and was morphed to a point. In iteration 2 we introduced the first "real" correspondence calculation. A search region is now created. The size of the search region can be specified by the parameter "windowsize" in the property file. For the correspondence calculation only those strokes of the successive keyframe are considered that have their first control point within the region

$$[(cp1.x - windowsize, cp1.y - windowsize),$$
$$(cp1.x + windowsize, cp1.y + windowsize)],$$

where *cp1* indicates the first control point of current stroke in the current keyframe. For all strokes that fulfil this criterion the distances between the control points and the difference in the colours are calculated to get a dissimilarity measure between two strokes.

The control point distance was implemented according to the formula

$$(stroke1cp1.x - stroke2cp1.x)^2 + (stroke1cp1.y - stroke2cp1.y)^2,$$

which corresponds to the squared Euclidean distance between the first control points of the strokes. This distance is normalized to values between 0 and 1 by dividing by the squared diagonal of the search area window.

The colour difference was calculated in RGB space according to the formula

$$((col1.r - col2.r)^2 + (col1.g - col2.g)^2 + (col1.b - col2.b)^2)/(3 * 255^2),$$

where *col1* represents the colour of the first stroke and *col2* the colour of the second stroke. *r, g* and *b* represent the red, green and blue colour component. Thus the colour difference corresponds to the Euclidean distance between two colours in the normalized three-dimensional RGB colour space so that the difference between black and white has a value of 1.

The use of other colour models like CIE Luv or CIE Lab, where colour differences correspond to distances in the colour space, were considered, but never realized, due to the fact that no significant improvement was reported in literature. **[2]**

Both dissimilarity criteria can be weighted by a factor, so that the user can apply different weights for the colour difference and the distance of the strokes. The dissimilarity measure can be expressed by the following formula:

$$dissimilarity = c_1 * distance + c_2 * colourdifference$$

Low dissimilarity values indicate strokes that could be morphed to each other. High values indicate strokes that do not represent the same image content in two different frames of an image sequence.

Figure 3(a): Original image sequence taken from a filmed sequence

Figure 3(b): Rendered sequence (Morphing approach)

Figure 3(b) presents low quality results that were obtained in this iteration. On most of the images the moving arm appears twice. In contrast the new correspondence calculation seems to work quite well for slow movements. Because of the fast motion in this video sequence the strokes are not moving. Additionally the majority of the strokes representing the arm at the former position does not change its colour to the background colour.

Figure 4(a): Original image sequence taken from a filmed sequence

Figure 4(b): Rendered sequence (Morphing approach)

In **Figure 4(b)** surprisingly good outputs are shown. This is caused by the same effect that was already shown in Figure 2. If the changes between two processed images are not too high the correspondence calculation produces better results. The remaining artefacts are "holes" around the contour of the person which always showed up, when the rendering is done with more than one layer.

Iteration 3

Keywords: comparison sequence, secondlayer, allpoints, various splinetypes, bugfixes

This is the first major release where all requested properties are working correctly. This includes the *Correspmode*, where it is possible to switch between “allpoints” and “startpoint”. If “allpoints” is chosen then every control point of the spline is considered for the distance calculation. Otherwise solely the distance between the starting points of the splines is used for the dissimilarity measure.

An extension of the correspondence calculation is provided by the option *Secondlayer*. It is now possible to process an additional correspondence search based on the strokes of the successive keyframe that have no correspondent stroke in the current keyframe yet.

Secondlayer: Until now the calculations between two frames did not consider splines without correspondence. This means that if no correspondences were found for the strokes during the search process in the first layer of the successive keyframe they simply had no correspondence and were morphed to a point. (From an implementation point of view it was not possible to let the stroke vanish, because no transparency value for a stroke can be set.) With the "secondlayer"-option another calculation for the remaining strokes is done in a reversed manner. The algorithm begins at the second frame and tries to find the most similar stroke in the layer of the first frame. Note that some strokes still might have no correspondence. Unfortunately the results of the "secondlayer"-approach were very disappointing.

Allpoints: Compared to the iterations before the distance between two splines is no longer solely based on a single control point. With "allpoints" now all control points of the strokes are considered for the distance calculation.

The calculation of the control point distance:

$$(stroke1cp_1.x - stroke2cp_1.x)^2 + (stroke1cp_1.y - stroke2cp_1.y)^2 + \\ (stroke1cp_2.x - stroke2cp_2.x)^2 + (stroke1cp_2.y - stroke2cp_2.y)^2 + ...,$$

where *strokeIcp$_J$.x* represents the x coordinate of the *J*-th control point of the *I*-th stroke. If the number of control points is not equal between the two strokes, then the distances of the supernomourous control points of stroke with the most control points are calculated to the last control point of the other stroke. In this specific case the distance is calculated as follows:

$$\sum_{i=1}^{n}((stroke1cp_i.x - stroke2cp_i.x)^2 + (stroke1cp_i.y - stroke2cp_i.y)^2 + \\ \sum_{i=n+1}^{m}((stroke1cp_n.x - stroke2cp_i.x)^2 + (stroke1cp_n.y - stroke2cp_i.y)^2,$$

where *n* represents the number of control points of stroke 1 and *m* represents the number of control points of stroke 2, and $n < m$. Finally the distance is normalized by dividing the squared Euclidian control point distance by m times the squared diagonal of the search area window.

Various splinetypes: An additional feature present in the libraries is now supported by the configuration file of the application. In the third iteration the following spline types are available to compute the stroke curves:

- B-SPLINE
- BEZIER-SPLINE
- BETA-SPLINE
- LINEAR
- CATMULLROM-SPLINE

Figure 5(a) shows the original image sequence

Figure 5(a): Original image sequence taken from a filmed sequence

Figure 5(b) is rendered with the morphing approach with the settings “allpoints true” and “secondlayer false”

Figure 5(b): Rendered sequence (Morphing approach)

Figure 5(c) is rendered with the morphing approach with the settings “allpoints true” and “secondlayer true”

Figure 5(c): Rendered sequence (Morphing approach)

In general the results of the new iteration are quite satisfactory. Still the moving arm is doubled which is caused - as already described - from the large differences between two processed frames, but enhancements are also visible. The morphing seems to work better and also the colour changes are now better adapted. Details of the person in the images are distinguishable from one another. The new parameters introduced new possibilities and overall better results could be achieved.

As we tested the effects of the new features we noticed that “secondlayer” set to true has some strange impact. The outcomes were unexpectedly much worse than without this parameter. After the testing-phase we discovered that with the parameters “allpoints” set to true and “secondlayer” set to false show the best results in this iteration.

Iteration 4 – *Final Iteration*

Keywords: new libraries, new OpenCV, new "windowsize" concept, aspect ratio, orientation, frame combiner (additional tool)

For the final iteration new libraries including a new interface for the Painterly Rendering algorithm were used. Besides a new version of the OpenCV library was available.

To allow a better correspondence finding by the morphing algorithm new functionalities were introduced at this iteration stage. Now it is possible to involve the shape and the orientation of the strokes in the correspondence calculation. The shape of the strokes is expressed by the aspect ratio (i.e. the major axis length divided by the minor axis length of the ellipse that is fitted to the stroke), the orientation of the stroke is calculated using the angle between the direction of the major axis length of the fitted ellipse and the horizontal direction. While the colour and the position of the strokes are already given, the new features have to be calculated before the similarity measure can be evaluated between a pair of strokes. For each stroke the orientation and the aspect ratio is stored in an array. A major drawback of the introduced features is the high computational complexity.

Furthermore the concept of "windowsize" has been changed. Now the bounding box of all control points is calculated for a stroke of the first keyframe. Afterwards the bounding box is enlarged by the value of the parameter "windowsize". A stroke in the second keyframe is considered as correspondent, if one of its control points is within the enlarged bounding box.

A major goal of this iteration was to evaluate the significance of the different parameters for the image rendering. Therefore an extensive testing and analysis was set up to discover the strengths and weaknesses of different parameter settings. Some of the conclusions reached during this testing are described below.

The question of the value of the "windowsize" parameter is probably one of the most interesting during this evaluation since it has strong influence on the processing time of the image sequence rendering. It turned out that the "windowsize" of 32 pixels showed the best results from the tested three sizes 16, 32 and 64. While there are some remarkable differences between 16 and 32, there are no significant changes between 32 and 64. For the comparison of the two image sequences an additional tool named "comparison sequence" was written, as introduced in iteration 3.

Some noise and holes around the contours of the image elements (persons, text, etc.) appear only when the image sequence is rendered with different brush sizes. If the morphing is done with only brush size 2 for example, no holes are found in the output images.

The new correspondence criterias aspect ratio and orientation work orthogonal, which means that when all of them are "activated" (i.e. value 1) the distance for example doesn't seem to have big influence on the calculation. A reasonable allocation of the weights for the different dissimilarity parameters has to be tested for each video.

The number of frames to morph depends on the frame rate of the video and on the magnitude of smoothness that should be achieved in the rendered output. For a frame rate of 25 frames per second a setting of 4 morphed frames shows good results.

Although the major memory leaks in the code were solved in this iteration the rendering still takes up some time. Suggestions for improvements in this specific case are described in the "Conclusions and Future Work".

The fourth iteration represents the final version of the program.

Figure 6 shows six comparison frames that were based on a part of a Mission Impossible 3 trailer. The comparison frames show four different versions of the video in the following order:

- In the upper left corner the original video
- In the upper right corner the Painterly Rendered Video with the layers 8, 4 and 2
- In the lower left corner the Painterly Rendered Video with the layers 8 and 2
- In the lower right corner the results of the morphing approach with the layers 8 and 2, where 4 frames were morphed.

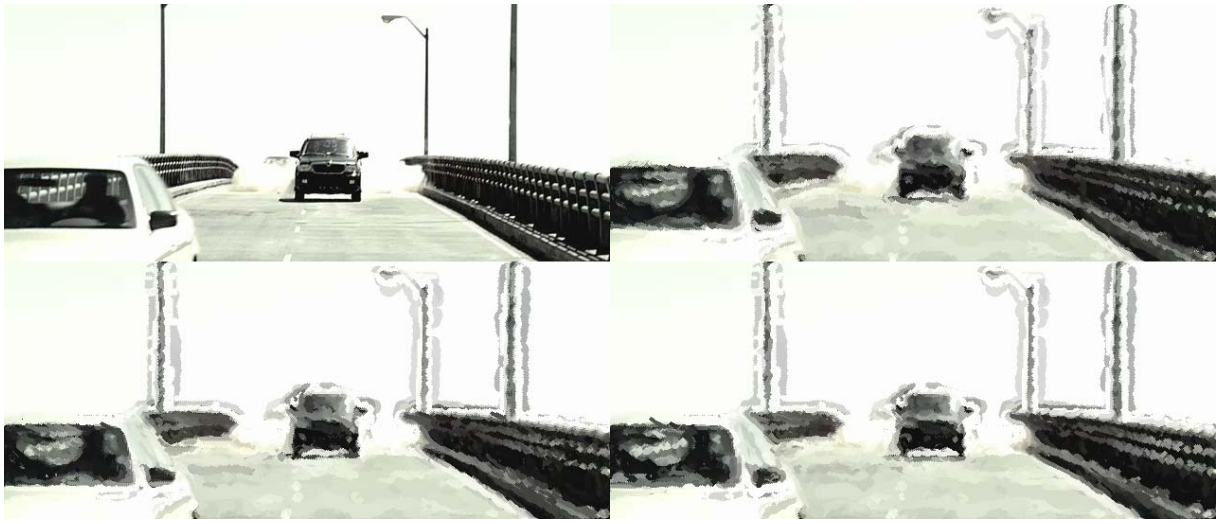

Figure 6(a): Comparison frame 1

Figure 6(b): Comparison frame 2

Figure 6(c): Comparison frame 3

Figure 6(d): Comparison frame 4

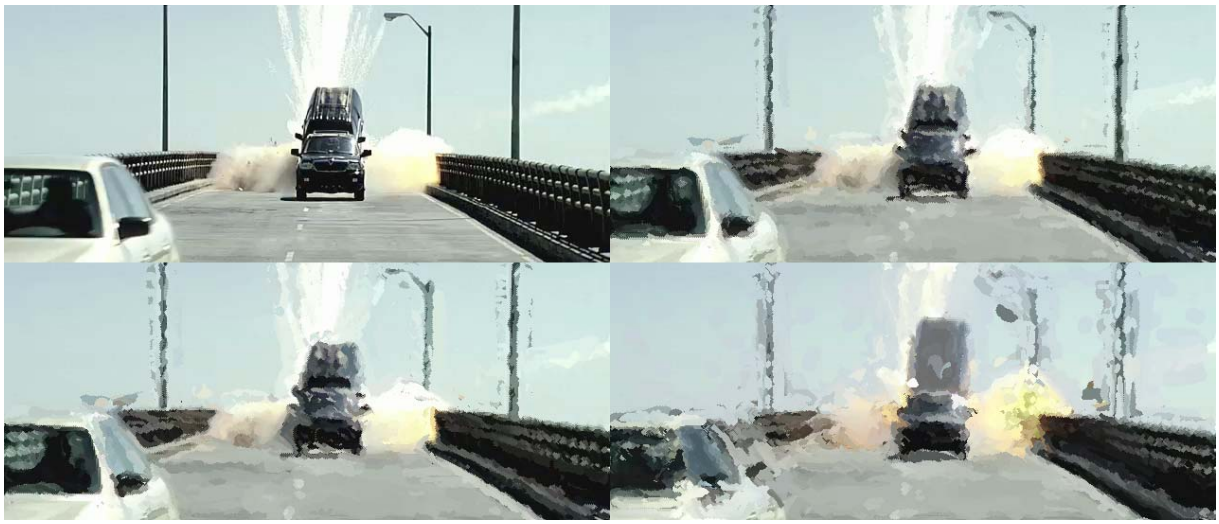

Figure 6(e): Comparison frame 5

Figure 6(f): Comparison frame 6

For the shown frames the Painterly Rendering algorithm paints the strokes of each layer from left to right and from top to bottom. Thus “holes” are introduced because the random stroke order is not used and therefore different errors are corrected by the smaller layers. The reason for this rendering approach is to show the base images on which our morphing approach is working.

If the random order of the strokes in each frames could be accessed by the morphing approach and the random order can be kept over the image sequence, the number of “holes” would be strongly reduced.

Conclusions and Future Work

In this report we presented an approach to overcome the drawbacks of rendering an image sequence with a non-photorealistic rendering algorithm frame by frame. Therefore we took frames from the videos and applied the painterly rendering on them and used the created strokes as basis for the morphing process. Thus intermediate frames are created by drawing morphed versions of the given strokes.

The biggest problem we came across was the loss of the correspondence between the layers which were used to paint the chosen frames (keyframes). This causes "holes" in the morphed frames, because layers with thinner strokes do not cover correctly the rough layer of the frame created by the thicker painted splines. To create good results only one layer was used for painting the keyframes.

The best advantage of using our algorithm is the reduction of flickering on static backgrounds introduced by video noise or small changes in the video content. While the painterly rendering creates small differences in each frame, many significant changes between consecutive rendered frames are generated. The morphing process reduces this effect, because different stroke positions are attenuated by morphing them to each other, that yields a smoother appearance of the video.

Further work could be done to reduce the processing time and find a better correspondence between strokes of two rendered frames. To speed up the implementation of the morphing approach the Painterly Rendering algorithm should be modified too. For example some calculations for the strokes can already be done during their generation (e.g. bounding box calculation, orientation and aspect ratio estimation). The use of arrays instead of Vectors could also accelerate the calculations, but the possibility to add arbitrary many objects would be lost. Also a subdivision of the frame in rectangular areas that hold references to the strokes could make the correspondence calculation faster. The reason therefore is that only strokes in successive keyframes which lie in the same area would be considered for correspondence and not all strokes of both keyframes have to be checked against a search area that is different for each stroke. Furthermore a class representing a morphed stroke could be introduced to speed up the morphing process itself. To find a better correspondence also other features should be examined. Alternatively a different approach of morphing the strokes without finding correspondences could be considered. For this idea the already calculated strokes of the first keyframe could be used in the Painterly Rendering process in a way that it uses the existing strokes efficiently and creates or deletes strokes, if it is necessary.

The morphing approach performs best on videos, where close ups of faces or big structures that move slowly are shown. Also for liquids and fire the morphing produces good results. Fast movements and short cuts create a blending or fading effect.

Acknowledgement

The authors want to thank Stathis Stavrakis for his continuous support and advice during the project. Many thanks also to Margrit Gelautz and Gabriel Wurzer for introducing us to the topic of video processing.

Appendix

Please note that this research document represents only some parts of the ViProII project. Some documents like meeting protocols, analysis evaluations, MATLAB calculations or additional tool descriptions, as well as many image sequence renderings from the different iterations are left out because of clarity reasons. For the sake of completeness a short description of the property file steering the rendering program is stated as follows.

```
// PAINTERLY RENDERING PROPERTIES
brushsize:          8 4 2   →   brushsizes
threshold:          20.0    →   Hertzmann's threshold value, standard setting is 20.0
depththreshold:     0.0     →   depth threshold value, standard setting is 0.0
strokemin:          4       →   minimum length of the strokes, standard setting is 4
strokemax:          16      →   maximum length of the strokes, standard setting is 16
underpaint:         T       →   if enabled an original input image is used as
                                "background", can be (T)rue or (F)alse
savepainting:       F       →   creates directories and results for every intermediate step
                                during the rendering process, can be (T)rue or (F)alse
// linetypes: LINE_LINEAR, LINE_BEZIER, LINE_BSPLINE, LINE_CATMULLROM,
LINE_BETASPLINE
linetype: LINE_BSPLINE      →   defines the type of the splines for the calculation
randomorder:        T       →   not implemented - future work

// BASIC ANIMATION PROPERTIES
filename:           <name>  →   prefix of the input images
startframe:         23      →   first input image to be rendered (has to be > 1)
endframe:           57      →   last input image to be rendered (has to be >= number of
                                the startframe)

// MORPHING APPROACH PROPERTIES
morphing:           F       →   variable to turn on/off the morphing approach, can be
                                (T)rue or (F)alse. (Morphing set to false will use the
                                Painterly Rendering for ALL images)
framestomorph:      4       →   how many frames are morphed between two keyframes

// CORRESPONDENCE PROPERTIES
correspmode:        advanced →  determines the correspondance calculation mode,
                                options: startpoint, advanced
secondlayer:        F       →   enables secondlayer, can be (T)rue or (F)alse
windowsize:         16      →   size (integer) of the window for the correspondence
                                calculation
allpoints:          T       →   enables allpoints, can be (T)rue or (F)alse
```

distweight:	1	→	weight of the distance, has to be between 0 and 1
colweight:	1	→	weight of the colour difference, has to be between 0 and 1
orientweight:	1	→	weight of the orientation, has to be between 0 and 1
asprtioweight:	1	→	weight of the aspect ratio, has to be between 0 and 1

// TEST FUNCTION FOR TRACKING THE MOVEMENT OF STROKES OF A SPECIFIC COLOUR

test:	F	→	enable the test, can be (T)rue or (F)alse
strokecolR:	80	→	value of red component of RGB colour
strokecolG:	52	→	value of green component of RGB colour
strokecolB:	47	→	value of blue component of RGB colour
tolerance:	5	→	tolerance value

References

[1] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In SIGGRAPH, pages 453-460, 1998.

[2] Aaron Hertzmann. A survey of stroke-based rendering. IEEE Computer Graphics and Applications, 23(4):70-81, 2003.

[3] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In NPAR, pages 7-12, 2000.