Application of Non-photorealistic Speed-lines to summarize object motion in Video sequences

Kartik Sayani : 201451030
Meghna Ayyar : 201451027
Mentor : Prof. Pratik Shah

***Abstract*—**

I. Introduction

II. Literature Survey

In this section we discuss the several papers and articles published in recent times that fall in the same area of interest as ours. Finding and tracking an object has been extensively studied and researched. Several novel techniques have been proposed for rendering of motion lines and speed-lines.

In [1], a lot of ways of depicting motion in still images are introduced in depth, followed by three different algorithms that when applied directly on a video can make it possible to render three types of motion depicting speed-lines. The algorithms deal with separating the object of interest from the background, creating a mask for the object and using it for identifying and tracking the object's motion in following images. The algorithms work effectively well when the objects of interest are the only moving objects, the background is uniform over the entire sequence and there is very less occlusion occlusion. [2] Also proposes yet another algorithm to summarize a short video into a single image by condensing the motion of moving objects in the final frame of the video so that the motion is depicted using comic-like speed-lines. It is novel in the sense that it does not require tracking any objects and hence can handle more complicated rototranslational motions. [3] on the other hand presents a non-photo-realistic rendering technique that can project a moving 3D object's motion on a still 2D image by using speed-lines. While [1], [2] work on the edges of objects to realize and render motion depiction, [3] considers surfaces of 3D objects and generates motion line polygons for rendering motion lines. A technique for depicting collisions is also proposed in [3]. In his thesis in [4], W. Song has formulated a tool to create speed-lines in 3D computer animations. He has exhaustively defined algorithms for rendering dynamic normal speed-lines, repetition of contours as well as perspective speed-lines for 3D animations which can also be ported for 2D animations. In our project, we have based our work heavily on [5]. The paper starts with introducing and defining various motion depiction techniques like Parallel speed-lines, tracking curves, radiative rays, contour replication and jagged contours, which is followed by techniques of rendering these on images. Parallel speed-lines and tracking curves being the closest to our project, we have derived our procedures similar to those described in [5]. The paper then works on the assumption that all the motion is considered to be as a feature of the object itself. Which is to say that even if a part of an object is in motion, the motion depiction is applied to the entire object. Which proved to be very effective in registering the track of an object over the frame and plotting speed-lines accordingly. In [6], a specialized way of applying speed-lines to human postures and gestures in order to enhance the scene artistically are discussed. A point to be noticed about this method is that the motion is measured, i.e., more drastic the motion, denser and longer are the speed-lines. Video summarisaztion techniques are discussed in [7]. The method is effective in scenarios where the background remains unchanged over time and the objects follow certain rules of motion, i.e., this method is application dynamic. It can be applied to traffic and security surveillance where the moving objects mostly move in recognized patterns. [8] proposes an algorithm to introduce speed-lines to inanimate objects. It shows how speed-lines and motion lines can be applied to still frames which can very well enhance its visual nature. The proposed method involves a good degree of user interaction to draw the speed-lines and motion lines.

In this work, we propose an experimental algorithm which is based on algorithms and rendering techniques from all the above mentioned articles, thesis and papers. We incorporate these techniques together to render parallel speed-lines and tracking curves for motion depiction. This algorithm is effective with synthetic, cartoon as well as real life image sequences. Our implementation involves user interaction as well to enable better rendering of speed-lines.

III. Proposed Algorithm

We present the details of the proposed speed-lines rendering algorithm in this section. We assume all along that there is no high degree of deformation of the object involved. We also assume that the motion of the object does not form loops over its path and does not stop or pause for a long time, i.e., there some movement of the object in every frame. The reason behind this assumptions is so that processing can be minimized and the process is smoother to work with. We further explain the reasons where these assumptions fall relevant.

The algorithm takes a video sequence(sequence of N image frames) as the input. We broadly divide the algorithm in three steps:

- *Object Determination and Tracking:* The very first step involves the user to point out the object on which we need to need to track and render speed-lines. This region

of interest(ROI) is marked and a mask is constructed for the ROI. We use the Mean Shift algorithm to track and register the track of the marked object.

- *Track registration and Back-edge Determination:* In every frame, once the object is tracked and found using mean shift algorithm, we draw a rectangle surrounding the object and mark it as the ROI. The parameters of the rectangle, such as the position of left corner, its width and height are stored. This is done over the entire video sequence. At the end, we have a sequence of rectangles, which is essentially the path of the object. The boundaries of the rectangles are used as the boundaries of the object itself. We use the path to construct motion vectors for ever pair of consecutive frames, which, along with the rectangle edges, are used to determine the back-edges.
- *Rendering curves and parallel speed-lines:* The back-edges registered in step 2 are history of the object over time. We use the sequence of points on these back-edges to interpolate cardinal splines for drawing tracking curves. For parallel straight lines, the entire path of objects is not required, just the back-edges of first and last frames. We use Bresenham's Line algorithm to interpolate all the points in between and draw the straight line. We use drawing technique described in [5] for rendering parallel straight lines as well as curved lines.

A. Object Determination and Tracking

Drawing speed-lines and motion curves has to do with entire object, i.e, even if there is motion in only a part of the object, speed-lines are drawn to represent the motion of the entire object as whole. For example, in a video sequence of a man running, motion is limited to his legs only. However, it is more legitimate to render speed-lines for the man himself and not only the legs. With this approach in mind, once the video is loaded, the algorithm presents the user with the first frame of the video. The user determines the object to be tracked and speed-lines drawn over. We call this the Region of Interest(roi). We consider this roi as the object itself. We firstly convert the format of the roi patch into HSV format, which is easier to work with image intensities. We then further create a mask of the roi, which is nothing but creating a binary image of it only this time, the mean of all the intensities of roi is used as the threshold and some extra parameters are used to remove any noise present. We create an intensity histogram of the hsv roi. This histogram is used for tacking in the Mean Shift algorithm.

Mean Shift is an iterative algorithm that is used for a lot of purposes such as finding nodes, clustering, etc. A brief idea about what Mean Shift does is this: It registers a window around all the points, computes the mean within the window and shifts the window to the mean and repeats the procedure again until convergence. Mean Shift is also applied to track objects. Here, the window constructed is the boundary of roi and the data points are the intensity values at each pixel in the roi. The mean of intensities is computed and the window is shifted to that pixel which is closest to the center of the window and has the smallest difference to the mean intensity. We repeat the iteration with termination criteria of 10 iterations or a movement of at least 3 pixels. Since we are using masks as the data points for Mean Shift algorithm, this is where the assumption on not having high degree of object deformation is relevant. If the object gets deformed, the mean shift fails in the long run since Mean Shift does not deal with object deformation. We use dynamic masks to suppress this to a certain level by replacing the roi from the previous image with the region of interest found in the next image as the new roi. This compensates for small deformations or changes in the structure of the object.

The Mean Shift algorithm efficiently tracks and finds the roi in subsequent frames. For every pair of consecutive frames, once the roi is detected, we save the parameters of the roi, i.e., we save the pixel location (lx, ly) of the top left corner of the roi and the width and the height of the roi (w, h). This data (lx, ly, w, h) is what represents the history of the object. This history is used for making the track and determining the back-edges in the further steps.

Figures 1 to 4 show the results of the object Wall-E tracked over 4 frames.

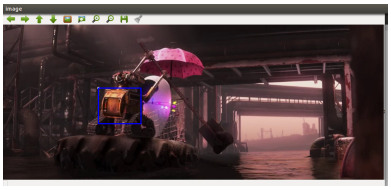

Fig. 1. Frame 1

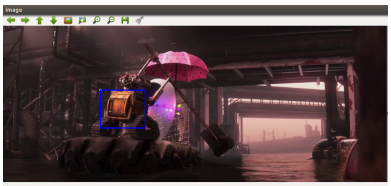

Fig. 2. Frame 7

Fig. 3. Frame 23

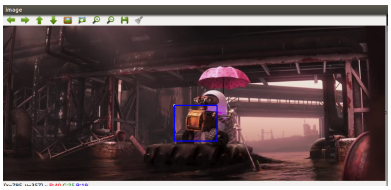

Fig. 4. Frame 32

B. Track registration and Back-edge Determination

We have the parameters of rectangles registered in the previous step We use these parameters to determine the boundaries, i.e., the pixels that will lie on the boundaries of the rectangles. As mentioned previously, we use these pixels on the rectangle boundaries as the edges of the object itself. This works very well because in order to render speed-lines, the exact boundaries of the objects are not required at all, as is considered in the many references we have studied. Since the speed-lines are to be applied to look like they have been drawn haphazardly behind the object, the use of boundaries of rectangles as the edges does the job very well. This assumption of pseudo edges also eliminates the need of subtracting the background or segmenting the image to determine the objects in motion and then determining the contours followed by isolation of back-edges form the many contour lines.

While we calculate the pixels on the boundary of the rectangles, we do them in such a way that they are sorted in anti-clockwise order with respect to the center of the rectangle. This makes it possible to easily separate back-edge points from frontier-edge points in a continuous manner without needing them to be resorted later. We visit these rectangular boundaries one by one and the points they are constituted of are checked for their eligibility for the back-edge.

The Back-edges of an object, in this context, means that part of the edge of the object that falls on the opposite side of the side that is facing the motion direction. Hence, we need to determine local motion vectors V in order to determine the back-edges. We do so by once again using the parameters of the rectangles. We use the (lx, ly, w, h) to determine the centers of the rectangles.

$$C_i = (lx_i + \frac{w}{2}, ly_i + \frac{h}{2}) \tag{1}$$

A vector from center C_i in the i^{th} frame to the center C_{i+1} in the next frame is what we use the local motion vector $V_{i,i+1}$. The side that falls on the opposite direction of this vector is the back side of the motion and the back-edge is the part of this side.

If $V_{i,i+1}$ is the vector of motion of the object from C_i to C_{i+1} where C_i and C_{i+1} are the centers of the object's bounding rectangles in two consecutive frames and $P_{i,r}$ is the vector directed towards the r^{th} point on the edge of the bounding rectangle in i^{th} frame from $C_{i+1,k}$ then if the following condition is met:

$$P_{i,k,r}.V_{i,i+1,k} > \cos(\theta) \tag{2}$$

the point can be considered as a part of the back-edge. Here, θ can be adjusted to have a larger or a smaller back-edge as per required. We have used this Equation 2 from [5]. Figures 5 and 6 show the back-edges of the objects.

This is how we determine the back-edges over the contours. Since we inculcate the motion vector, the back-edge points are collected on the side opposite to direction of the object's motion vector.

We have here applied the back-edge determination over a pair of consecutive frames. We apply steps 1 and 2 of Motion Segmentation and Back-Edge Determination over all the pairs of consecutive frames. Thus we have the back-edges of all the moving objects in frames 2 to penultimate frame. These back-edge points are all stored corresponding to the back-edges in the next frame are the tracks of the object in the total sequence. This is how the track of the objects is registered by consecutively determining the back-edges using centers of the objects and their motion vectors.

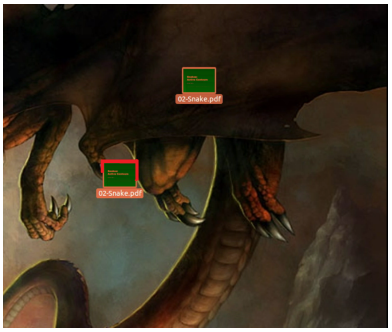

Fig. 5. Back-edge points in a frame of IconMove sequence #2

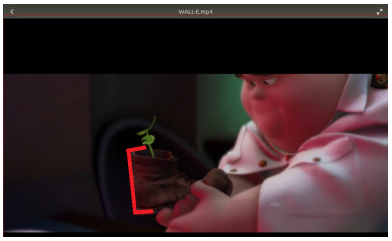

Fig. 6. Back-edge points in a frame of PlantShoe sequence #3

C. Rendering curves and parallel speed-lines

This is a fairly simple step. We have points on tracks of the object in form their consecutive back edges. The edge points of the object's bounding rectangle in the last frame act as the starting points of the speed-lines and the edge points of the bounding rectangle in the first frame as the end point of the lines. We simply use Bresenham's Line algorithm to select all the pixels that lie between first point and the last point of the two collection of back-edge points. We use the parallel speed-lines rendering technique described in [5].We render these speed lines on the very last frame. The length of these lines can be anywhere between a default minimum and the maximum distance the object's center has moved from first frame to the last frame.

For rendering curved speed-lines, that cover the exact track of the objects, we use Catmull Rom Splines, with the tracked points on the back-edge forming the control points of the Splines. This gives use a spline equation for every consecutive pair of track points. We can get the pixels lying on this spline with the help of this equation and thus we have a continuous track of the object from certain points on the back-edge collection.

The density of the speed-lines is dynamic and varies with the object's size and the amount of motion. The local vectors determined in the previous steps are summed up and the

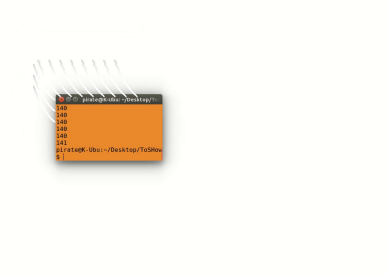

Fig. 7. Curved speed-lines 1

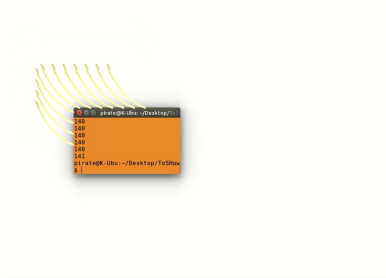

Fig. 8. Curved speed-lines 2

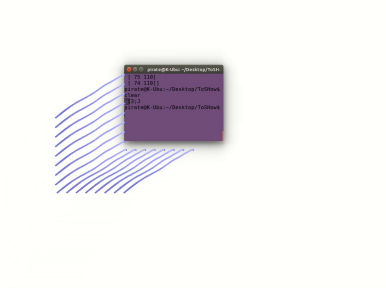

Fig. 9. Curved Speed-lines 3

resulting motion vector, which is the amount of overall motion of the object, determines the density of the speed-lines. We define a parameter α for every local vector that is set from 0 to 1 depending on the value of the local vector. The average of these α values α^0 is used to determine the density as well as the length of the speed-lines.

In the following section we include the results of our implementation.

IV. Experiments and Results

Figures 7 to 12 show Curved speed-lines rendered for various objects with varying motion complexities in video sequences. The color of the speed-lines can be manipulated as well as the length. The density of the speed-lines rendered varies with the amount of motion but can also be manually controlled by adjusting a few parameter α^0.

Figures 13 to 18 show Parallel Speed-lines renders for single as well as multiple objects. The lengths of the speed-lines has been varied and the application is demonstrated for synthetic, cartoon and real life sequences.

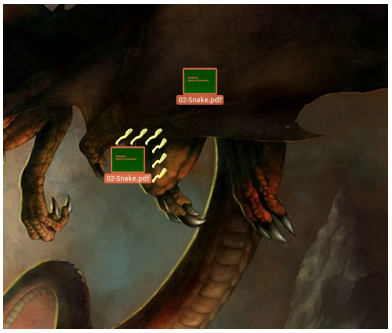

Fig. 10. Curved Speed-lines 4

Fig. 11. Curved Speed-lines 5

Fig. 12. Curved Speed-lines 6

Fig. 13. Parallel Speed-lines 1

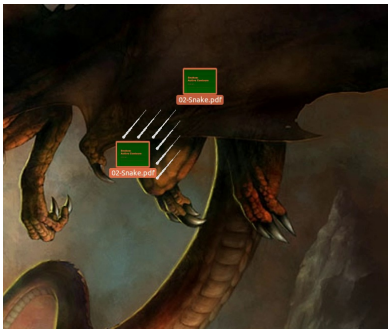

Fig. 14. Parallel Speed-lines 2

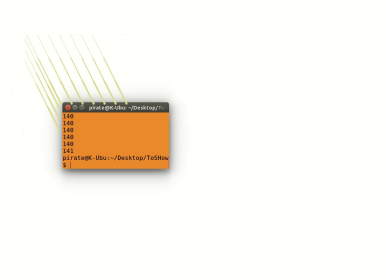

Fig. 15. Parallel Speed-lines 3

Fig. 16. Parallel Speed-lines 4

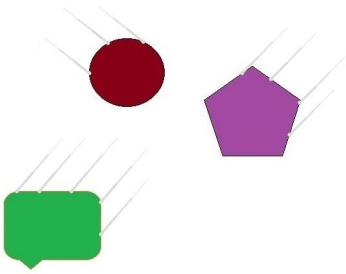

Fig. 17. Parallel Speed-lines 5

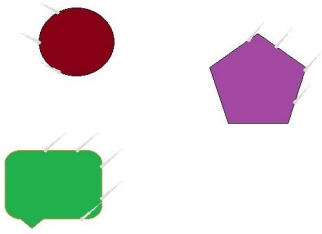

Fig. 18. Parallel Speed-lines 6

V. Conclusion

References

[1] Prof. V. Caglioti and Ing. A. Giusti, *Non-photo-realistic Rendering Of Speed-lines*

[2] V. Caglioti, A. Giusti, A. Riva and M. Uberti, *Drawing Motion Without Understanding It*

[3] T. Moriya and T. Takahashi, *An Extended Non-Photorealistic Rendering Technique for Depicting Motions of Multiple 3D Objects*

[4] Won Chan Song, *Speed-Lines for 3D Animations : A Thesis*

[5] Ying-Hua Lai and Dr. Zen-Chung Shih, *The Synthesis of Non-photo-realistic Motion Effects for Cartoon*

[6] T. Kawabe and K. Miura, *Representation of Dynamic Events Triggered By Motion Lines and Static Human Postures*

[7] Y. Gong and X. Liu, *Generating Optimal Video Summaries*

[8] M. Masuch, S. Schlechtweg, and R. Schulz, *Speedlines: Depicting Motion in Motionless Pictures*