Final Algorithm for Detection of Moving Bodies and Plotting Speed-lines to Represent Their Motion

Based on the papers

- The Synthesis of Non-Photorealistic Motion Effects for Cartoon
  by Ying-Hua Lai

- Speed-Line For 3D Animation
  by Won Chan Song

- Nonphotorealistic Rendering of Speed-Lines
  by Prof. Vincenzo Caglioti and Ing. Alessandro Giusti

After thoroghly reading and implementing concepts wherever possible, we have identified an algorithm for final implementation of our project, i.e., Rendering Artificial Speed-lines to Represent Motion in Cartoon Sequences.

The algorithm takes place in three steps and operates on a given number of frames of an animation sequence. In the first step, we identify the moving objects in the entire sequence and focus only on them. We also record the motion which is used in step two.
In it's second step, we determine the edges of all the objects and use the the motion records from the previous step to determine the back edges of the motion against the direction of their motion. We do this for all the given frames. This gives us the track of the moving object.
In the third step we use this recorded tracks to render the speed-lines.

The details of these steps follow next. We consider first two frames of the sequence.

Step 1:

In normal animated sequences, majority of the action/ movement takes place on the foreground, while the background remains mostly static. Therefore, we detect the primary target objects by segmenting the image into background and foreground. The background of an image is that part that has a duller colour intensity, as it is in the background. The foreground, on the other hand is the brighter part of the image.

The very first step is to convert the second frame *frm1* to its gray-scale equivalent *frm1_g*. And then convert this gray-scae image to a binary image *frm1_b* using Otsu's clustering based image thresholding algorithm. The white part of the binary image denotes the approximate foreground of the image. This binary image then undergoes morphological Opening operation. The result *frm1_op* is also a binnary image and has all the noisy small white leftovers removed. We put *frm1_op* under morphological Dialation. The resuting output is the actual background plus some neighbouring areas. We call this result *sure_bg1*.

Further, we take the distance transfrom of *frm1_op*. This yields a gray-scale image, which is similar to *frm1_op* except that the gray-level intensities of points inside foreground regions are changed to show the distance to the closest boundary from each point. We again aplly Otsu's thresholding on the distance transform, we get the sure foreground of the image *sure_fg1*. We have now the sure foreground and the sure background of the image. We take the difference of these two images to get the unknown region that lies around the actual boundaries. We use markers to mark the sure background, sure foreground and the unknown regions and feed this marker to Watershed marker based image segmentation algorithm. The watershed algorithm return back the boundaries of the objects in the image. Now we use this informaion to find out the contours, i.e., the edges of the objects of the image.

This is how the trackable objects are detected in the second frame. Along with the contours, we also store the centers of these contours to be used in further steps.

Step 2:

The contours detected in the second frame are now sequentially visited and the points of which they are made up of are sorted in a counter-clockwise direction. This is to assure continous back edge and rear edge points which makes implementation easier.

We loop over these contours, visiting each one after the other. We intend to find where this contour is situated in the previous image, i.e., in *frm0*. We achieve this by constructing a bounding rectangle around each contour. This rectangular template patch is then searched over in *frm0*. Once this patch is detected we take the contour in the detected patch and find its center.

If C_{i+1} is the center of the current contour and C_i is the center of the contour in previous image, then the vector V_i is the direction of the motion of the object represented in the contour.
D_i is the vector pointing towards a point P_i on the contours
All the points that satisfy :
$D_i . V_i > cos(\Theta)$
Are the points on the rear edge of the object. Here Θ is the rear-view angle from the center of the object. The rear edge length can be varied by changing Θ.

We apply steps1 and 2 this for *frm1* as current frame and *frm0* as the previous frame, then, *frm2* as the current frame and *frm1* as the previous frame, and so on. Recording the back edges at every frame pair, for every contour. This is how we track and register the track paths of the moving objects.

Step 3:

This is a fairly simple step. We have the tracked points of the back edges of the objects. We simply use Bresenham's Line algorithm to select all the pixels that lie between two track points w_i and w_{i+1} . *We* then draw circles of decreasing radius starting with the largest radius at w_i and ending with the smalled one at w_{i+1}
This gives a speed-line effect to the objects movement, covening the direction of the movement. We render these speed lines on the very last frame. The edge points of the object contours in the last frame act as the starting points of the speed-lines and the edge points of the contours in the first frame as the end point of the lines. 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. This gives a straight speed-line from start point to end point.

To get curves that cover the entire track of the object's motion, we use spline interpolation to calculate the cubc polynomial expression of curves passing through the triplets of the tracked points.

This is how the speed-lines are rendered to represent the motion.

This was the algorithm that we have figured out. We have already implemented steps 2 and 3. Step 1, i.e., to detect the primary objects for tracking has been implemented on synthetic images and reqires some changes to be done to be applied on normal animated dequences.