

# Cinemagraph: Automated Generation (CAG)

Walter Li  
Stanford University  
walterli@stanford.edu

Bo Xian See  
Stanford University  
bsee@stanford.edu

## Abstract

*We implement a method of automating the creation of cinemagraphs for an input video using stabilization, object tracking, image blending and video texturing. Currently, artists must carefully setup and edit the video to create a cinemagraph. This process is both tedious and time-consuming. Our method streamlines the pipeline to reduce the amount of work the user have to perform to create a cinemagraph. The only user input our method requires is a selection of four points for stabilization and tracking.*

*Using the fact that the frames are highly similar, we adopted an affine model for stabilization for computation efficiency. Next, we use temporal gradients to track the object's motion-subject region between frames. Then, an efficient two-band blending is used to blend across frames. Lastly, we adopted video texturing to find a good seamless loop for the GIF images. Our method will enable users to create cinemagraphs easily using videos captured from any devices, including their smart phones.*

## Future Distribution Permission

The author(s) of this report give permission for this document to be distributed to Stanford-affiliated students taking future courses.

## 1. Introduction

Cinemagraphs are short, repeating animated images of a mainly static scene with slight motions, usually in GIF format. It was first introduced by a professional photographer, on his blog in Tumblr [1]. As mentioned earlier, it is extremely difficult for a common user to create cinemagraph as it takes time, skills and equipment to create one. An artist must carefully set up a scene to create a cinemagraph, as any motion shake increases the complexity of creating cinemagraphs. Moreover, artists must manually select masks for the area in motion, and blend frames onto one another.

Our novel system enables users to automate the creation process. The user can feed in a video recorded casually, removing the need to carefully set up the scene. We can thus assume that the frames will be highly similar, since the

video is assumed to be taken from the same scene with roughly the same field of view.

### 1.1. Method

An overview of our system is as follows. Firstly, based on the review of stabilization techniques by Rawat and Singhi[2], we found out that the software stabilization technique proposed by Farid and Woodward[3] was best suited for our application. The algorithm is computationally efficient and its affine model captures sufficiently rich range of motion such as translation, rotation and scaling. The user's input will serve as the region of interest for video stabilization. Since as mentioned before, the video is assumed to be taken from the same scene, this method is well-suited for our needs.

Originally, we used a SIFT descriptor to find the bounding box of the tracked object through the scene. In this step, we require the user to select the bounding box for tracking. The user can choose to skip this step if the change in the object's position is lower than a threshold. However, this method was very inaccurate and we had to switch to motion tracking using Temporal Gradient. This filter tracks moving objects and returns a mask of the areas with large fluctuations in intensity.

After tracking the bounding box, we blend the bounding boxes of subsequent frames onto the first frame using the two-band blending algorithm proposed by Lowe[4]. This method fast and provides a good result for highly similar frames.

Lastly, we implemented techniques for transforming normal videos into video textures [5] to create a seamless endless loop for our video. These techniques include methods of calculating transition cost and probability, optimizing loops for minimum cost, and sequencing a list of transitions into a loop for GIF format. Due to a lack of a video texture format, we settled on producing an animated GIF format image file. Although this loses the randomness of true video textures, this is coherent with how cinemagraphs are generated.

### 1.2. Data Set

We intend to record video using any digital camcorder. Our sample size should be at least five videos of different scenes. Sample data will include approximately

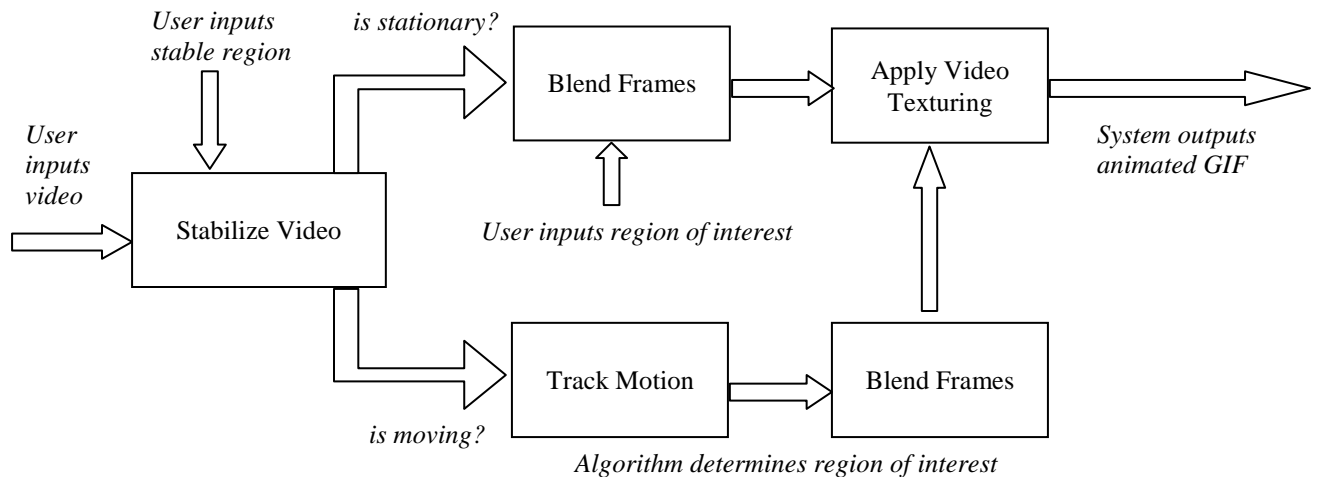


Figure 1: Pipeline of our system.

A video is first fed into our system. The user then selects a region that is not moving in the video, and chooses if the object of interest is moving. If it is not, we simply blend the region of interest selected by the user. If it is, we apply motion tracking to track the region of interest before blending the frames. The last step of our pipeline is to create an infinite look using video texturing,

five-second clips of mostly static scenery. For each video, there must be a segment with static scenery for image stabilization. The tracked object can be either stationary or moving through the scene.

### 1.3. Evaluation

Evaluation of a piece of art is usually taken qualitatively. We will visually evaluate the final animated image and grade it based on smoothness of frame transitions, and the smoothness of blending with background image. However, some quantitative measurements can be employed to evaluate the results.

Intermediate results are produced by each subsystem, and evaluated individually. First, the video stabilization system can be evaluated by considering the complete video excluding the region of interest of the subject. The average standard deviation in pixel intensities can be used to compare the difference between the original video and the stabilized video.

Second, the object tracking and pyramid blending systems can be hard to quantify. But we can overlay a highlighting color over the tracked region of interest, and generate an animation over all the frames. A human user can evaluate all frames and count the number of frames with a correct and complete highlighting overlay.

Third, the two-band image blending step is also difficult to quantify. On the one hand, sharp edges due to image boundaries are unwanted. On the other hand, we do not want to blend away important subject features. Image overlay boundaries can be detected using edge detection. The presence of edges as an outline around the object signifies a non-optimal blending scheme. Faded features

sometimes may be necessary to reduce sharp jumps in loop transitions, and thus unavoidable.

Finally, video textures can be evaluated based on total loop length, total cost of transitions in final animation (in comparison with other possible transitions not used), and the *loop performance*, which is an aggregate metric defined as length divided by cost.

## 2. Related Work

Previous related works to automate the process use video texturing to continuously loop through a video at selected regions [6]. However, their method does not track objects in motion, but instead consider regions containing dynamic objects. This loses the context of the scene as the regions are considered independent of each other. Also, this means that dynamic scenes with many moving objects will not be handled well.

In this paper, we aim to overcome this by using object tracking. Instead of creating the blending area based just on using changes between frames, we use tracking to determine the blending area across frames. This accounts for translational movement, and thus is more robust than the pipeline proposed by James et al.

We also referred closely to the works and tutorials by artists [7-9], to understand how cinemagraphs are traditionally created.

### 3. Technical Approach

#### 3.1. Video Stabilization

To stabilize the video, we modelled the camera movement between two frames using a 2x2 affine matrix, and a 2x1 translational vector, as described by Farid and Woodward [3] via the equation:

$$f(x, y, t) = f(m_1x + m_2y + m_5, m_3x + m_4y + m_6, t - 1)$$

The variables  $m_1, m_2, m_3, m_4$  form the  $2 \times 2$  affine matrix  $A$  and  $m_5$  and  $m_6$  the translation vector  $T$ .

We solve for these 6 unknown variables by minimizing the quadratic error between the two frames.

$$E(\vec{m}) = \sum_{x,y \in ROI} [f(x, y, t) - f(m_1x + m_2y + m_5, m_3x + m_4y + m_6, t - 1)]^2$$

The ROI is the region of interest. Using the Taylor expansion, and discarding high order terms, we can solve the minimization function efficiently.

$$\begin{aligned} \vec{m} &= \left[ \sum_{ROI} \vec{c}\vec{c}^T \right]^{-1} \left[ \sum_{ROI} \vec{c}k \right] \\ \vec{m} &= [m_1 \quad \dots \quad m_6]^T \\ \vec{c}^T &= (xf_x \ yf_x \ xf_y \ yf_y \ f_x \ f_y) \\ k &= f_t + xf_x + yf_y \end{aligned}$$

The terms  $f_x, f_y,$  and  $f_t$  are partial differentials of  $f(x, y, t)$ .

As long as the user selects a reasonably big region of interest, this minimization function is solvable.

A coarse-to-fine scheme is adopted in order to compute coarse movements efficiently. We use a 3-level Gaussian Pyramid to give us better estimation for larger movements. We chose to use 3-levels because we know that the frames will be from the same scene, and a 3-level Gaussian Pyramid should be sufficient to capture the camera shakes, without causing a significant slowdown.

#### 3.2. Object Tracking

To track the object through the scene, we used both Scale Invariant Feature Tracking (SIFT)[11] and Temporal Gradient[10].

Our first attempt at tracking object was done using a SIFT descriptor. We first compute keypoints of every frame in the video. Then, the user inputs the original bounding box of the object, and the bounding box is tracked in the scene using SIFT. When no bounding boxes are found, or if the bounding box is determined to be incorrect, the user reselects the bounding box of a scene. To detect an incorrect bounding box, we check for discontinuity in velocity, position, scale and rotation of the object. This is because we know that the object must follow a certain path

in the scene. A scene of about 50 frames typically requires the user to reselect the bounding box 3 times. However, we realized that this method gives us a significant number of false positives, claiming that a match has been found, and giving incorrect positions. This is especially true during frames where the object is blurred due to motion. Because of the blur, we lose important keypoints and edges of the object, either causing a false negative or a false positive.

Also, it was extremely difficult to find a frame with good keypoints because in most frames, the object is in motion. A possible way to detect good frames is to use Sum-Modified Laplacian[12], that gives an estimate on how sharp the image is. However, we felt that overall, SIFT will not work well in this situation.

To overcome this, we switched to intensity gradients with respect to time to identify areas with motion. Here, we are under the assumption that a change in pixel intensity is equivalent to some form of motion. To reduce the possibility of missing important regions of interest, we do not apply a Gaussian blur first. We find that a Gaussian blur simply reduces our chances of finding the areas of motion. However, this introduces the possibility of higher noise in our result. Since we do not plan on using excessively noisy video in the system, this is not an issue. Any noise at the edges of motion is suppressed by a dilation of the edges created motion-tracking mask, and background-blending later on.

The video is first converted to a 3D array with axes in the *height*, *width*, and *time*. This allows us to easily apply a Sobel filter in the *t*-axis using MATLAB's 'imfilter' command. The magnitude of the resulting intensity rate of change at each spatial-temporal point is normalized to the maximum and then a threshold value is applied to consider only regions with a large amount of motion. Empirically, a value of around 0.15 to 0.25 works sufficiently well. Finally, the points above the threshold is dilated using 'imdilate' with a disk of radius 15px (usually ~10% relative to the minimum of width and height of the video).

Any noise (now typically circles with radius of 15px) in the mask is blended away in the last step.

#### 3.3. Two-Band Blending

We implemented a modified version of the two-band blending algorithm proposed by Lowe et al. [4]. In the paper, blending was done only for overlapping regions between two panorama images. Our modification enables the user to blend bounding boxes onto a common frame.

To blend the two frames, we first divide the frames into its high frequency component and its low frequency component. This is achieved using a simple two level Gaussian Pyramid on both the outer frame and the inner frame, using the equations:

$$\begin{aligned} LowFreq(x, y) &= Image(x, y) * Gaussian(\sigma, size) \\ HighFreq(x, y) &= Image(x, y) - LowFreq(x, y) \end{aligned}$$

The variables  $x, y$  represents the position of the pixel in the blending region, while  $\sigma$  and  $size$  are parameters that the user specifies. For our code, we used  $\sigma = 0.5$  and  $size = [3 \ 3]$ .

We combined the low frequency components using linear interpolation over radius that is specified by the user.

$$\begin{aligned} LowFreq_{blended} &= LowFreq_{outerFrame} * \alpha \\ &+ LowFreq_{innerFrame} * (1 - \alpha) \\ \alpha &= \frac{d}{r} \end{aligned}$$

In this equation,  $d$  is the distance to the inner frame and  $r$  is the radius specified by the user. The low frequency component of the blended image is thus the two dimensional linear interpolation of the overlapping region. As described in the paper, the high frequency component is not interpolated because we want to keep the details of the objects. We simply used the high frequency of bounding box because it contains the details we want to preserve.

The overall blended frame will then be the combination of the low frequency component and high frequency component.

$$\begin{aligned} BlendedImage(x, y) \\ = HighFreq(x, y) + LowFreq(x, y) \end{aligned}$$

To accommodate for uneven shapes, we used a Region of Interest (ROI) mask to indicate the blending regions. This is an extension to the existing algorithm, as the existing algorithm only accounts for vertical seams. We dilated the original ROI with a radius specified by the user to get the blending region. In other words, we extended the ROI by radius amount while keeping the original shape. Blending is then done in this region outside the ROI.

Similar to the blending with bounding boxes, the high frequency detail of the inner frame is kept, while the low frequency detail is linearly interpolated. Note that the weights of the low frequency components are now calculated based on the distance away from the nearest pixel within the ROI, and the above equations still hold.

### 3.4. Video Texturing

Given a series of frames, the objective now is to generate an infinite loop of frames with minimal discontinuities between frames. The video textures work by Arno Schodl, et al. can be applied to find the optimal transitions between non-continuous frames [5]. Since we are only concerned with a subsample of each frame, video texturing is given only a subsample of the whole video.

First, we compute the cost of a transition as L2-norm distances between the  $i$ th and  $j$ th frame for all the frames. This is saved as an  $n$ -by- $n$  square matrix ( $D_{ij}$ ). Dynamics preservation and future costs are also incorporated by,

respectively, iteratively computing and solving the following set of equations.

$$\begin{aligned} D_{i,j} &= \|I_i - I_j\|_2 \\ w_k &= \begin{bmatrix} \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{bmatrix} \\ D'_{ij} &= \sum_k w_k D_{i+k, j+k} \\ D''_{i,j} &= (D'_{i,j})^p + \alpha \min_k D''_{j,k} \\ P''_{i,j} &= K \cdot e^{-\frac{D'_{i,j}}{\sigma}} \end{aligned}$$

The matrices  $I_i$  and  $I_j$  are the images at frame  $i$  and  $j$ , which can be used to calculate the cost matrix ( $D$ ). The vector  $w$  is a set of weights for computing the cost matrix after considering preserving dynamics ( $D'$ ). Finally, the cost matrix that also considers future costs is calculated as  $D''$ . The probability of each transition is modelled exponentially, and normalized such that each row sums to one.

This results in the 3D plots for probability of each transition shown in figure. This matrix is then filtered for local maxima, which will become our list of possible transitions. To make things interesting, we would like to weight the probabilities by distance of each transition (in indices from source to destination). This allows us to prefer multiple large jumps to many small transitions which could produce jittery motions in the animation. This is an issue because we must make deterministic rather than probabilistic decisions on transitions.

Following analysis of transitions, we consider only the top 20 (or however many) transitions. These are then used to find the optimal loop for up to a certain number of frames, with the lowest cost. This is optimized by dynamic programming. With the optimal set of transitions, we then sequence them using the steps outlined in section 4.3 of Schodl's paper [5] to generate the final sequence of frames, which is written to an animated GIF file.

## 4. Evaluation and Results

Because our results are in animated GIF format, please go to the following links to view the results. The original input GIF, taken by our smart phone and imported into Matlab, can be found here: <http://www.stanford.edu/~bsee/cgi-bin/cs231a/fan/original.gif>

### 4.1. Video Stabilization

The stabilized video can be found here: <http://www.stanford.edu/~bsee/cgi-bin/cs231a/fan/stable.gif>.

There are still some movements in the non-subject areas of the image. However, this is a limitation of affine transformations. A shaky camera still experiences slight

perspective changes, which cannot be fixed with an affine transform.

Fig. 2 shows the results of our image stabilization. In this image, the black region is the region selected by the user as moving parts. This is a parameter that the user can select, to denote if the area selected is moving or stable. The intensity of the pixel represents the variance of the pixel across all the frames. Thus, a pixel with high intensity has high variance, representing instability, while a pixel with low intensity represents low variance, and hence, stability.

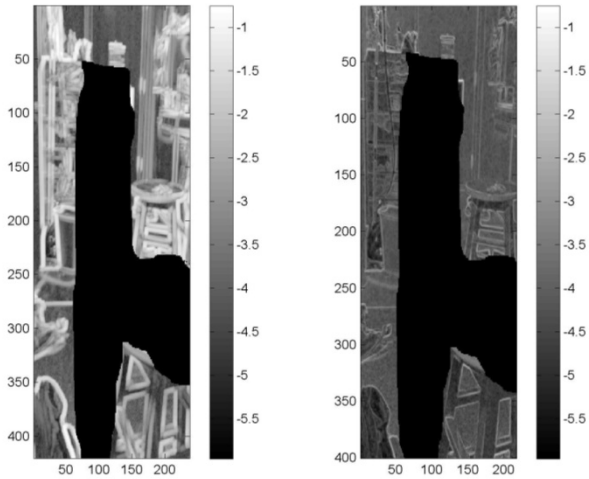


Figure 2: Evaluation Results of Image Stabilization

The figure on the left is the output of the original video, while the figure on the right is the output of the stabilized video. From the figures, it is clear that the video is a lot more stable after stabilization, because the average intensity of the pixels is much lower.

However, the result of the stabilization technique is heavily dependent on the user input. A bad selection for the stable region might yield a worse output compared to the original.

## 4.2. Two – Band Blending

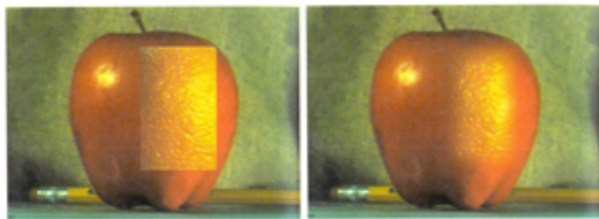


Figure 3: Unblended image (left) and blended image (right) with bounding box blending

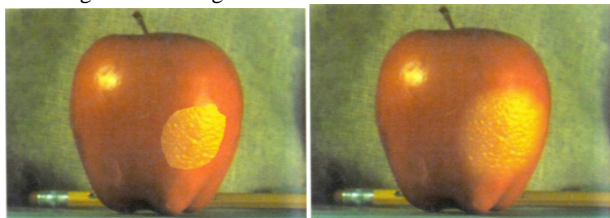


Figure 4: Unblended image (left) and blended image (right) with region of interest selection.

The images shown in figure 2 and figure 3 are the results of image blending an orange with an apple. The original image samples are taken from Lowe’s paper.

To evaluate the blending method, we used a Sobel filter to perform edge detection on the blended images. This is a good measure of image blending because a well-blended image will not have discontinuity at the blending regions. The blend is considered good when no edge is detected at the edge of the region of interest, and when all other edges previous present are preserved.

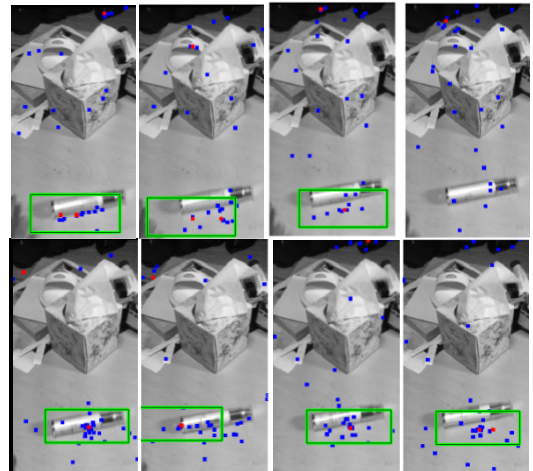
We blended 50 random images, creating 25 blended images, similar to the ones above. In all our images, no discontinuity was detected at the region of interest, and all original edges were preserved.

We also evaluated the efficiency of our algorithm. The bounding box blending took an average of 0.5 seconds, while the ROI blending took an average of 30 seconds for a 738x502 pixel image. The huge disparity between the bounding box blending and the ROI blending is due to the fact that in the ROI blending, the blending region is no longer uniform. Thus, we are unable to perform column-wise and row-wise operation, causing a significant slowdown in our pipeline. Instead, each pixel was calculated individually. We realize that there is a trade off between efficiency and robustness, and we chose to be robust in our pipeline, since most natural objects do not occur in uniform shapes.

## 4.3. Object Tracking

### 4.3.1 Scale Invariant Feature Tracking

We evaluated our SIFT-based tracking by counting the number of true positives, false negatives and false positives. We know that there are no true negatives in our data set as the object is present through the entire scene. Unfortunately, the SIFT-based tracking performed worse than expected, detecting very few true positives.



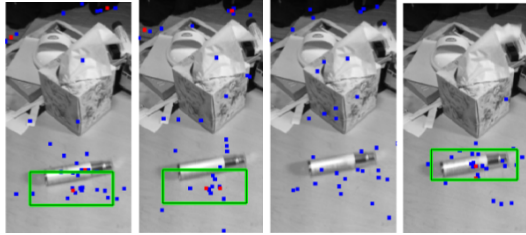


Figure 5: 12 Frames from the SIFT-based tracking output.

By displaying the tracked bounding box and the tracked center, we are able to identify true positives, false positives and false negatives. We ran this algorithm on five separate files with 50 frames each, and realized that on average, the descriptor only yields 80% accuracy across all frames. This is even after discarding bounding boxes, which are determined to be inaccurate.

#### 4.3.2 Temporal Gradient

A gradient with respect to time can produce accurate masks that capture the area of motion. Some noise is produced due to unstable background images. With a less noisy video, such as one taken on a tripod, with only the subject in motion, such noise in the mask does not appear. Dilating the mask helps to keep desired motion-rich areas below the threshold by taking advantage of their proximity to areas above the threshold.

Some types of motion do not lend well to the intensity gradient method. For example, shadows are low frequency content, in both spatial and temporal dimensions. Thus, they can be lost during thresholding. If we limit our subjects to higher frequency content, such as moving objects, this method will be sufficient.

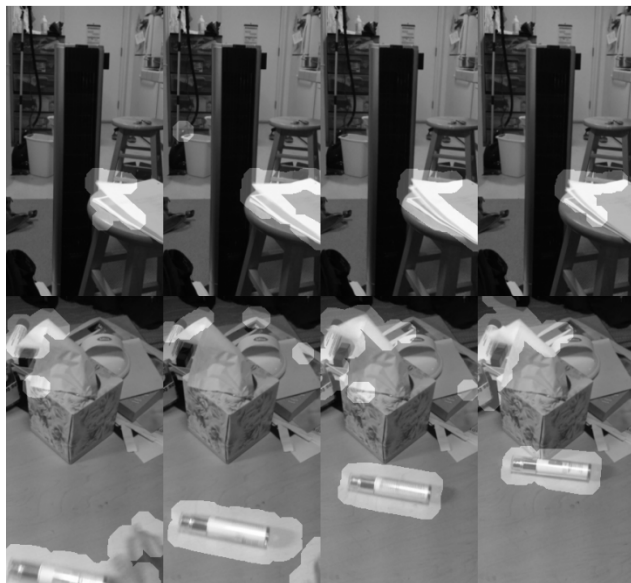


Figure 6: Tracking mask overlay produced by temporal gradient method for two separate videos at four different points in time.

## 4.4. Video Texturing

The following charts depict some of the intermediate variables and data used by video textures to determine the optimal transitions. Each local maximum is considered good transitions to take. This is necessary because many local maxima are far lower in probability than the peaks in the center, but we still want to take distant transitions, and not a lot of small steps.

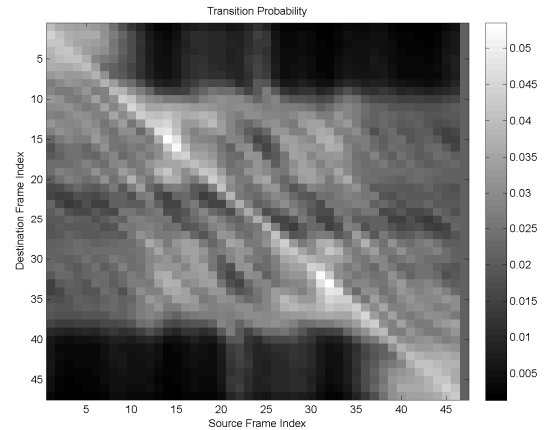


Figure 7: Probability of transition, modeled exponentially on the cost matrix, after considering dynamics preservation and future costs.

We define a new metric in order to measure the “performance” of a transition. A high performing transition would take a longer jump, but still have very low cost. The transition performance (TP) is defined as:

$$TP = \frac{\text{Distance (Length) of Transition}}{\text{Cost of Transition}}$$

This effectively weights the farther transition as “better” than very tiny transitions that only move back by a few frames. Such small jumps are unwanted because they appear to have high frequency jittery motion that is unappealing to the viewer. On the other hand, large jumps usually contain some meaningful action appearing between two transitions. In fig. 8,

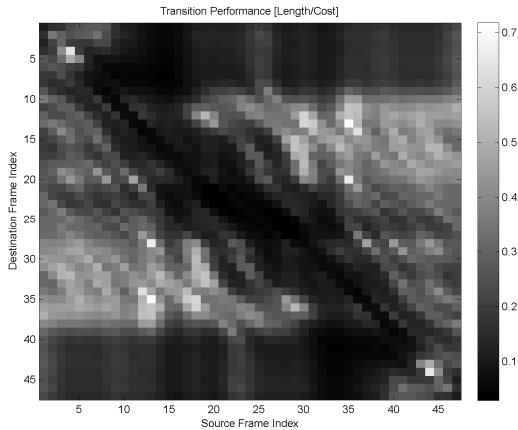


Figure 8: Transition performance is defined as the length of the loop created by the transition divided by the cost of that transition.

#### 4.5. Final Cinemagraph

The result after creating a seamlessly looping animated GIF image using video textures can be found here:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/fan/final.gif>

Qualitatively, the result looks like a seamless infinitely looping animation. Depending on the subject, the user may wish to insert motionless frames at the end to provide a pause between the action. This is purely an artistic decision, and so it is not implemented here.

#### 5. Conclusion

In conclusion, we have created a complete pipeline that enables novice users to create cinemagraphs without laborious photo editing and careful setup. Each of our results were actually taken by hand using our smart phones, and we were still able to create cinemagraphs.

We adopted various existing solution to help create the pipeline. First, we used the affine-approximation stabilization technique proposed by Rawat and Singhai[2] to stabilize the video. Next, if the target object is in motion, we use motion tracking algorithm to track the object. Although our attempt at SIFT[11] failed, we achieved a good result using Temporal Gradient[10]. We then blend the region of interest using the Two-Band Blending proposed by Lowe et al[4]. Even though this technique was originally used for 1-dimensional stitching, we extended it to enable blending across a general shape. There is, however, a trade-off between robustness and efficiency, as we see that our runtime significantly increase with the freehand blending. Lastly, we used video texturing, a technique by Schodl et al.[5]. to create a seamless loop between frames. This is necessary as cinemagraphs are continuous and loops infinitely.

A typical cinemagraph of 400x300 pixels takes approximately 5-7 minutes to render from start to finish on a 8GB 2.8Ghz Core i7 processor. It also requires two user

inputs: one for stabilization and another for blending without motion.

#### 6. Future Work

We realize that our output videos are in grey-scale. One obvious step forward is to extend this technique to all 3 color channels.

Also, we know that the bottle neck of our pipeline is blending the image and creating the video texture. There is definitely room for improvement for the above techniques.

Lastly, in the future, we would like to include color filtering on the frames of the video. The duo-tone color filter is what gives cinemagraphs a distinct look and feel for it. Because our video lacks colors, there was no need for duo-tone coloring. However, this is a feature that is definitely nice to have.

#### 7. Other Results

We have also created several other cinemagraphs to test our system. You can view the other cinemagraphs via the following links:



Figure 9: Ceiling Fan

Input:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/ceiling/original.gif>

Output:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/ceiling/loop.gif>



Figure 10: Curtains I

Input:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/curtain/original.gif>

Output:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/curtain/final.gif>

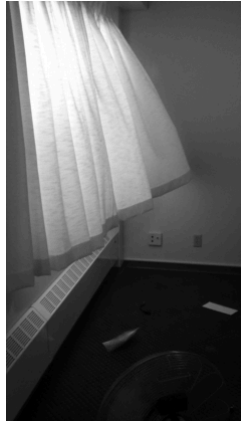


Figure 11: Curtains II

Input:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/curtain/original.gif>

Output:

<http://www.stanford.edu/~bsee/cgi-bin/cs231a/curtain/final2.gif>

## 8. References

- [1] J. Beck, K. Burg. (2011, Feb.). *From Me To You – les tendrils*. [Online]. Available: <http://fromme-toyou.tumblr.com/post/3263597796/les-tendrils-kaelen>
- [2] P. Rawat and J. Singhai, "Review of Motion Estimation and Video Stabilization techniques For hand held mobile video," *Signal & Image Processing: Int. J.*, vol. 2, no. 2, pp. 159–168, Jun. 2011.
- [3] H. Farid, and J. B. Woodward. "Video Stabilization and Enhancement." *Science* (1997)
- [4] M. Brown, and D. G. Lowe, "Recognizing Panoramas." *International Conference on Computer Vision*. 2003. 1218-1225.
- [5] A. Schödl, R. Szeliski, D. H. Salesin, I. Essa, "Video Textures." *Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH 00* (2000) : 489-498.
- [6] J. Tompkin, F. Pece, K. Subr, J. Kautz, "Towards Moment Images: Automatic Cinemagraphs." *Proceedings of the 8th European Conference on Visual Media Production (CVMP 2011)*. Nov. 2011.
- [7] F. J. Baez. (2011, Apr.). *Cinemagraph Tutorial*. [Online]. Available: <http://fernandojbaez.com/cinemagraph-tutorial/>
- [8] L. Banks. (2011, May). *How to Make a Cinemagraph with Photoshop and After Effects*. [Online]. Available: <http://lesterbanks.com/2011/05/how-to-make-a-cinemagraph-with-photoshop-and-after-effects/>

<http://www.adorama.com/alca/article/How-To-Make-A-Cinemagraph>

- [9] P. Edenberg, et al. (2011, Jun.). *How to Make a Cinemagraph*. [Online]. Available: <http://www.adorama.com/alca/article/How-To-Make-A-Cinemagraph>
- [10] Dollar, P.; Rabaud, V.; Cottrell, G.; Belongie, S.; , "Behavior recognition via sparse spatio-temporal features," *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on* , vol., no., pp. 65- 72, 15-16 Oct. 2005 doi: 10.1109/VSPETS.2005.1570899
- [11] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.
- [12] Nayar, S.K.; Nakagawa, Y.; , "Shape from focus: an effective approach for rough surfaces," *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on* , vol., no., pp.218-225 vol.2, 13-18 May 1990 doi: 10.1109/ROBOT.1990.125976