Creating User-Input Driven Cinemagraphs

Matthew Bollom and Samantha Kuray
University of Wisconsin, Madison
CS 534, Fall 2012
http://www.matthewbollom.com/cinemagraphs/

Abstract
Cinemagraphs are a relatively recent media which turns a video clip into a still photograph with an area of repeated animation. Originally created through manual manipulation with photo editing software, automated tools are now being developed to assist in the creation of cinemagraphs. However, a number of the tools available today are either entirely automatic or does not sufficiently focus on the area of animation. This project aims to create an automated tool to create photorealistic cinemagraphs while still allowing the user to assert creative control over the final product.

1. Introduction, Motivation, and Goals

A photograph has the benefit of recording a particular moment in time while a video captures the context around a moment. However, the additional motion in a video can distract the viewer and dilute the importance of a single moment. A new medium, called cinemagraphs, bridges still photography and video by allowing a part of the image to be in motion while the rest of the image remains still. This has the effect of drawing attention to the motion of interest [3]. Cinemagraphs are often presented as animated GIFs, creating an endless loop of usually repetitive motion much as handshaking, a car driving by, page turning, spinning barber shop poles, and more.

Cinemagraphs start with a short video clip manually modified in a photo editing tool such as Photoshop [6]. Manually creating cinemagraphs requires extensive editing skills, but the results can be very convincing if done correctly, such as those produced by Jamie Beck and Kevin Burg [7]. Some cinemagraph apps for smartphones exist such as Cinemagram, Kinotonic, and iCinegraph, but they require the user to select the entire region of motion across all frames, something that may be undesirable if there is additional motion present within the selection. Alternatively, some algorithms for automating cinemagraphs remove the element of user-driven input entirely [4]. This project aims to allow the user to select an area to animate in the initial frame of a video, and our algorithm will track the motion of this selection throughout the frames and create an endless looping cinemagraph.

For this project, we focused on three areas in implementing cinemagraphs:

1. Automating the creation of cinemagraphs, to allow their creation without requiring photo editing tools.

2. Allowing the user to select the static background and object of motion for the cinemagraph.
3. Improving upon the region of motion compared to commercially-available cinemagraph apps.

2. Related Work

Cinemagraphs were first developed by Jamie Beck and Kevin Burg in 2011 [6, 7]. They envisioned cinemagraphs as “more than a photo, but not quite a video” since only selected parts of the cinemagraph are animated. They created these images by starting with photography frames and manually editing the images to create an animated GIF [8]. Repetitive actions such as a car driving by, or circular rotation of an object are excellent candidates for creation of a cinemagraph since these animated GIFs naturally continuously loop. Creation of cinemagraphs is often done in a digital image editing tool such as Adobe Photoshop. But because this is a manually process, it takes the user significant time and photo-editing experience to create a photorealistic cinemagraph.

Since Beck and Burg released their first cinemagraph in 2011, a variety of methods to animate the process have been proposed in the literature. Tompkin *et al.* present the first known automated method (2011). Their software makes the process of creating a cinemagraph partially automated; it detections motion automatically, but requires the user to provide input such as what objects they wish to have in motion as well as provide some parameters for segmenting motion from the frames of the video. They also developed a method to find the best way to loop the resultant cinemagraph such that the sequence appears to be a continuous loop. Tompkin *et al.* envisioned their system in use on a smartphone, so they aimed for an efficient algorithm. Overall, they conclude their software provides excellent cinemagraphs, but it currently takes too long to create the result.

Bai *et al.* present an alternative method that removes the motion from portions of a video clip (2012). Their algorithm takes user input to determine which parts of the video should have motion removed from and those sections of the video frames are warped to remove the motion. However, doing this can introduce visual artifacts in the background, so they search for other frames within the video to replace warped pixels with other pixels in the video. By warping to remove motion, this paper takes a different track from much of the other literature.

Cliplets is a Microsoft research project that mixes still image and video segments in a way that is spatially and temporally correct. Cliplets differ from cinemagraphs such that the moving elements do not need to loop. In addition, a moving sequence of pixels does not need to be moving throughout the entire sequence of frames. The sequence of frames is created by aligning frames, allowing the user to make some selections and timing choices, and creating the frames through automatic feathering, matting, and compositing with Laplacian blending. This work is considered more general than that of Tompkin *et al.* and Bai *et al.* because of the flexibility it offers the user, but it still requires extensive user input. Cliplets is available for download from Microsoft Research at http://research.microsoft.com/en-us/um/redmond/projects/

cliplets/paper/

Yeh and Li have developed a mobile app that enables the automatic creation of a cinemagraph with no user input (2012). The software stabilizes the input video, analyzes motion via optical flow, and creates a cinemagraph of the largest motion. A user survey was conducted to demonstrate the performance of their software and to analyze user preferences with regards to cinemagraphs. Yeh and Li demonstrated their automated cinemagraphs are visually pleasing, even though their automated approach failed at times.

Finally, there are a variety of mobile apps available such as Cinemagram, Kinotonic, and iCinegraph. Experimentation with these apps showed that they required extensive user input to create the cinemagraphs as well as some thought into the result. For example, the user must select the entire area of motion for the entire video even if the object only takes up a small portion of the frame. If another moving object occurs in that area, the motion will appear in the final cinemagraph, an undesirable result. No technical details on these apps are currently publicly available.

3. Method

We aimed to create software using Matlab that would bridge the work done by Tompkin *et al.* and Yeh & Li. We wanted to allow the user to be able to specify content of the static frame as well as select the object of motion. This differs from Tompkin *et al.* in that the user can choose a different static background, even if the object of motion is not present. This also differs from Yeh & Li in that it allows the user to have some say over what is animated. By providing a selection, our algorithm will track the object throughout the different frames in the input video clip.

We have approached the problem through these steps:

- Allow the user to make a selection for the static frame and object of motion via a graphical user interface
- Segment and track the motion
- Creating the static frame
- Combine the motion and static frame

We have provided implementation details broken into the various steps.

Input Video Clip

Our method assumes that the input video clip is spatially registered as no video registration is performed in our algorithm. Registering the video frames allows for easy detection and tracking of motion and creation of a photorealistic cinemagraph.

Graphical User Interface

In order to allow the user to select the static frame and the object of motion for the cinemagraph, we implemented a basic graphical user interface (GUI) through the use of Matlab's GUIDE tool. The user is shown a single frame of the video and can use a slider to navigate through the frames. The user then indicates which frame should be used as the background frame by clicking the “Select as Background Frame” button (figure 1). The user can then either use the slider to choose a different frame in which to select the object of motion or click the “Select Object” button to begin their selection of the object of motion (figure 2). We wanted to allow the user the opportunity to select different frames the background and to specify the object of motion to account for a scenario where there may be a frame in which the object of motion isn’t present and which should be used, in its entirety, as the background frame. Selection of the object of motion is done through Matlab’s imfreehand function, which allows the user to draw a circle around a single object of motion. Once the user has selected both the background frame and the object of motion, they click on the third button, “Create Cinemagraph” to begin the automated process of creating the specified cinemagraph.

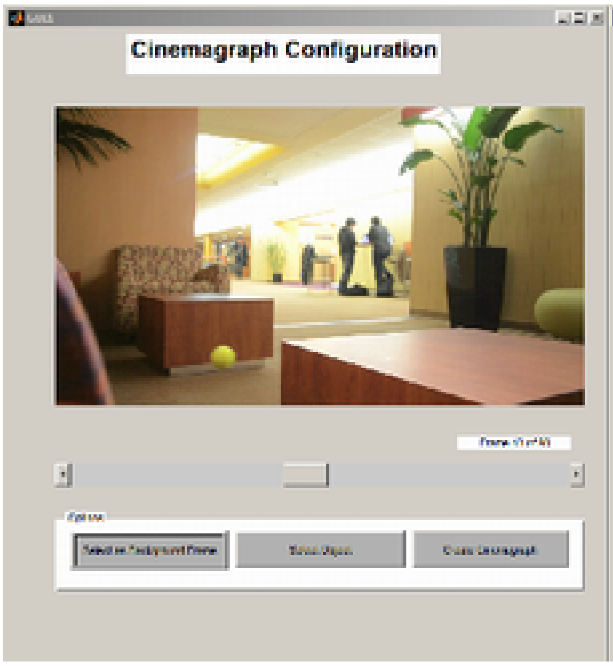

Figure 1: Selecting the background frame

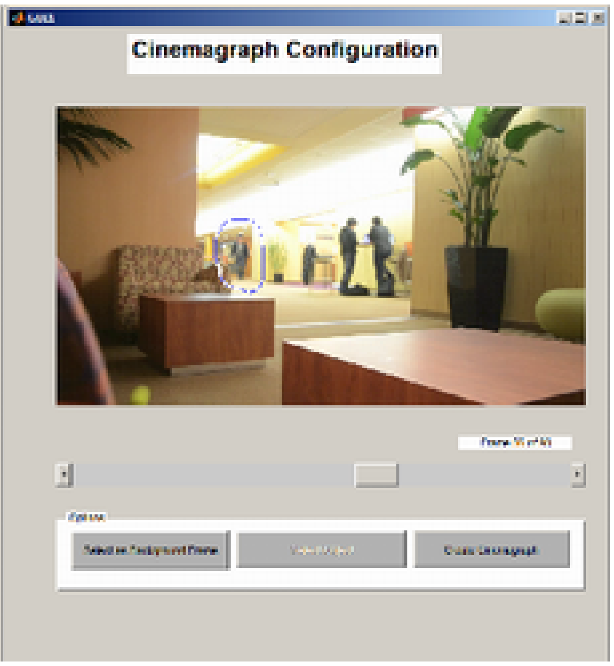

Figure 2: Selecting the object of motion

Segment and Track the Motion

Our motion tracking algorithm is based on work by Tompkin *et al.* (2011) and extended with work by Gonzalez and Woods (2008). We first compute an average image across all frames of the input video (figure 3). This results in an image that is representative of the entire image. Objects that are static throughout the input video remain in their location in the average image while objects that are moving appear as ghosts or are not visible. Objects that appear as ghosts move slowly or are static for several frames while objects that are not visible are most likely in motion during the entire video sequence.

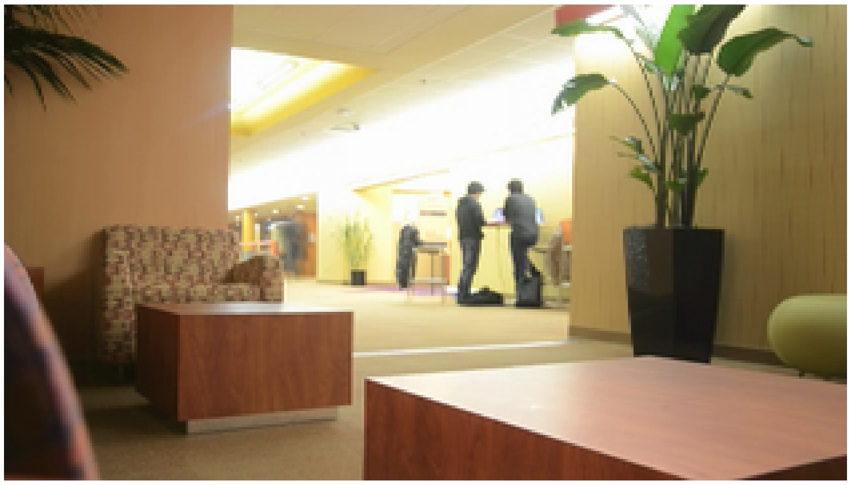
Figure 3: The average image

The average image is then subtracted from each frame in the video, resulting in an image of just pixels in motion. Occasionally, individual noise pixels may remain after subtraction; this is due to quantization noise of the camera sensor and will be removed at a later step. This image is then converted to grayscale for use for the rest of the algorithm (figure 4).

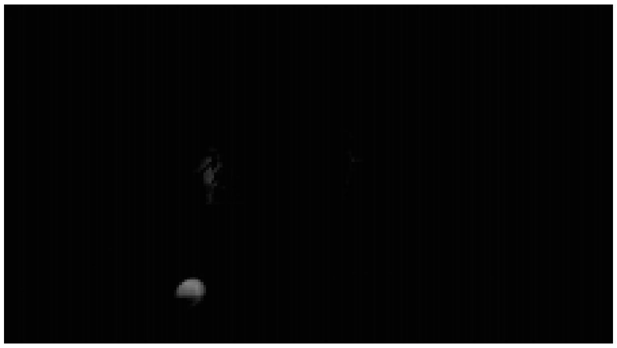
Figure 4: The grayscale difference image

Initially, a frame-by-frame difference was used to detect motion. Since objects in motion would most likely not be in the same position for consecutive frames, non-zero pixel values should indicate pixels that belong to a moving object. This method failed when the object was a constant color and there was not enough texture detail for the individual pixels to be different. The result would be a shadow of the object with the interior pixels indicated as static, an incorrect result. This prevented further steps in the algorithm from segmenting the motion correctly.

Given a grayscale image of candidate motion pixels, the image was then converted to a binary image using a dynamic threshold (figure 5). The threshold was calculated as the mean intensity level plus one standard deviation. One standard deviation was determined through testing to produce a sufficient result without obtaining too few or too many pixels. Since the dynamic threshold is recalculated for each frame, the threshold can quickly adapt to rapid changes in motion. An image with little motion will have a lower threshold than a frame with large motion.

Since the thresholding step allowed the noise pixels to be easily seen, all connected components of less than 0.1% of the image size were removed (figure 6). This threshold was suggested by Tompkin *et al.* and testing showed this threshold worked well. The pixels left correspond to the components of the true motion in the frame.

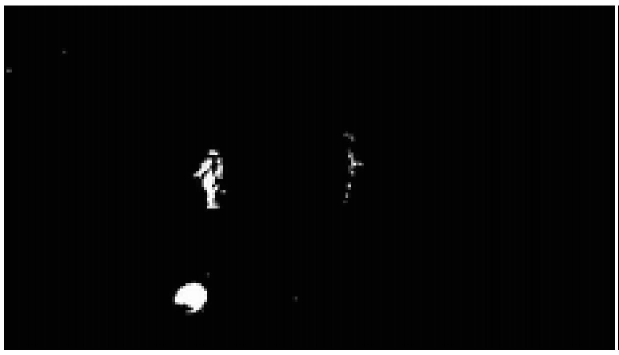
Figure 5: Converted binary image

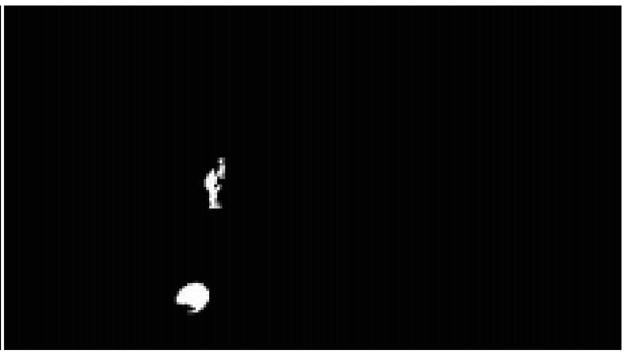
Figure 6: With noise removed

Although the result created thus far provides component of motion in the frame, additional processing is needed to create a smooth result. Small details were often lost when just the above steps were performed, so we needed to extend the method thus far to try to include the rest of the details. Filling holes in the image (figure 7), performing morphological closing (figure 8) and dilating (figure 9), and computing the convex hull (figure 10) created a small border of pixels around the motion objects that allowed for a better cinemagraph.

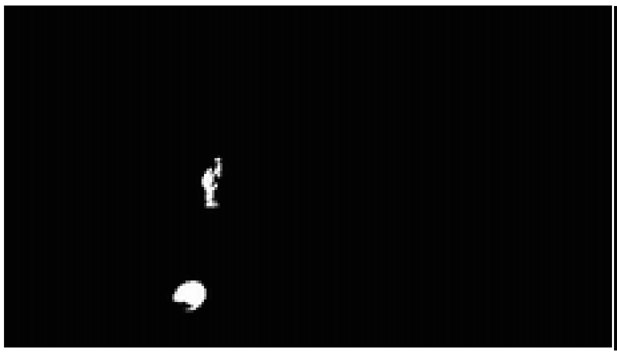
Figure 7: Mask will holes filled in

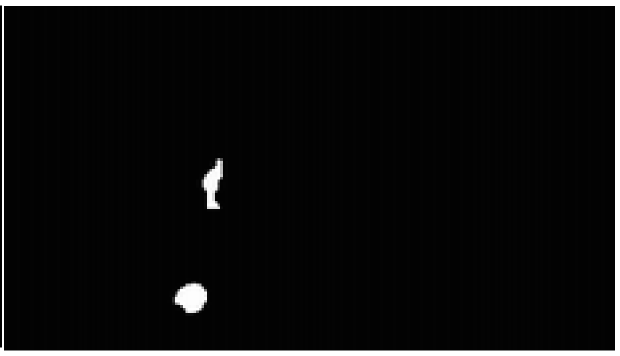
Figure 8: With morphological closing

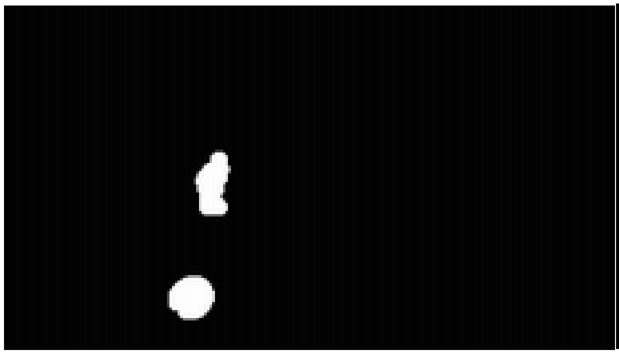
Figure 9: With dilation

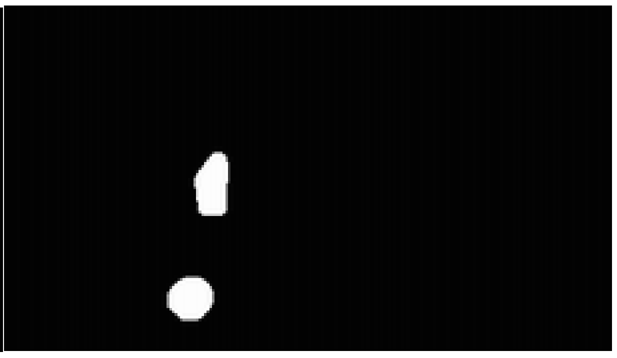
Figure 10: With convex hull

By performing these steps for each frame of the input video, we can create a logical mask depicting the areas of motion in each frame (figure 11).

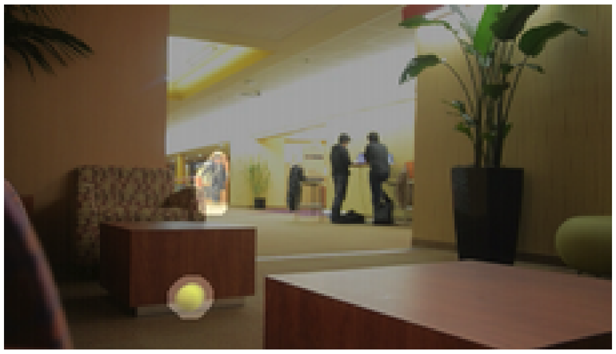

Figure 11: Overlay of logical mask with original frame

With the objects of motion detected, we then track the user's selected object of motion throughout each frame of the input video. We start at the frame the user selected the object of motion with and then search the frames forwards and backwards for the object for overlaps in the motion areas. We assume that the object is moving slow enough such that some part of it will overlap in consecutive frames. If the object moves fast enough so it does not overlap, the object's motion will be lost. The same will occur for objects that may briefly disappear from the frame, such as a ball being through up in the air. If the ball were to go outside the video frame, then come back in, the ball's motion would no longer be able to be tracked once the ball reappeared in the frame.

The results of motion segmentation and tracking yields a logical mask providing the location of the object of motion in each frame of the video (if available).

Creating the Static Frame

In a number of existing cinemagraphs, the static background surrounding the object of motion is of a naturally static scene - a wall or pieces of furniture in a room. However, the static background can also freeze a dynamic moment, such as pedestrians frozen mid-step in a walk through the park [7]. By allowing the user to select the static frame, we give the user greater control over the artistic aesthetic of the final cinemagraph.

Once the user has identified the frame that should be used as the static background, our algorithm determines whether the object of motion is present in that frame and, if it is, removes the object of motion from the static frame. Removing the object of motion from the background frame allows for a cleaner background frame upon which the object of motion can be overlaid. This step is especially necessary in instances where the object of motion moves across a wider range of the frame, such as a ball rolling across the frame.

To identify the location of the object of motion in the background frame, we use the result of the tracking motion portion of the algorithm to create a logical mask the size of a single frame, identifying the location of the object of motion in the static background frame. We then replace the pixels corresponding to the location of the object of motion with an average pixel value for that location, taken across all other frames. In our testing, using the average value to account for the pixels in the location of the object of motion works best when there isn't a lot of motion in the area around the object of motion in the frame. For example, a tennis ball rolling along a table is easily replaced by an average value, as the table remains fairly constant across all frames (figures 12 - 14). However, this method risks losing a static location of other motion in the immediate area that should be frozen, as the average may come across as a blur of motion or a ghost (figures 15 - 17).

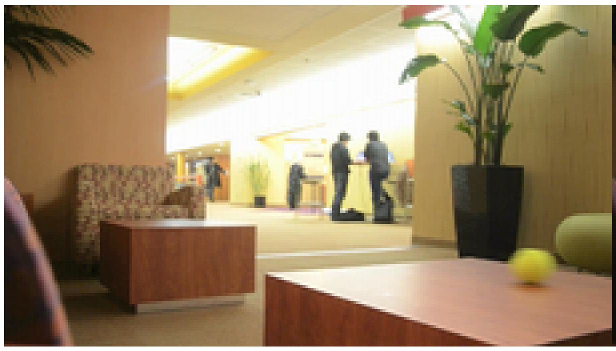
Figure 12: Original frame with tennis ball moving across the table.

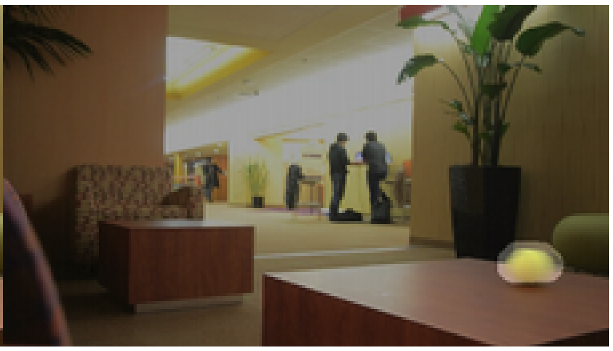
Figure 13: Overlay of the logical mask identifying the user-specified object of motion (tennis ball) onto the original frame

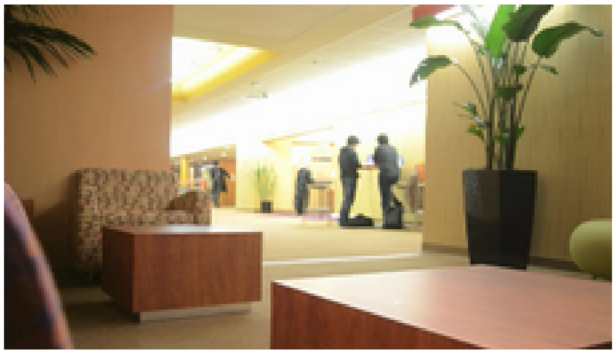
Figure 14: Complete static background frame, with the location of the ball replaced by an average across all other frames, with Laplacian Pyramid blending.

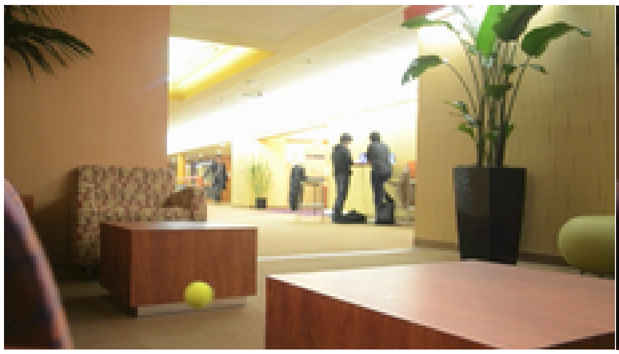
Figure 15: Original frame with person walking in background

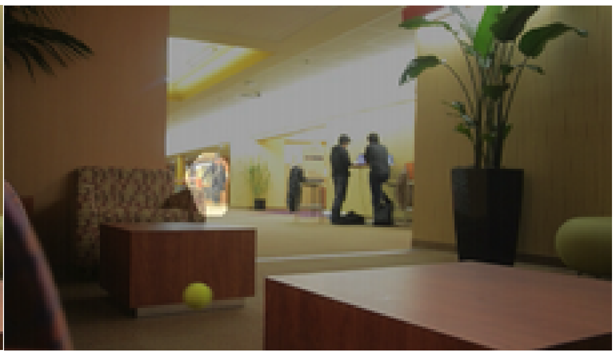
Figure 16: Overlay of the logical mask identifying the user-specified object of motion (person in background) onto the original frame

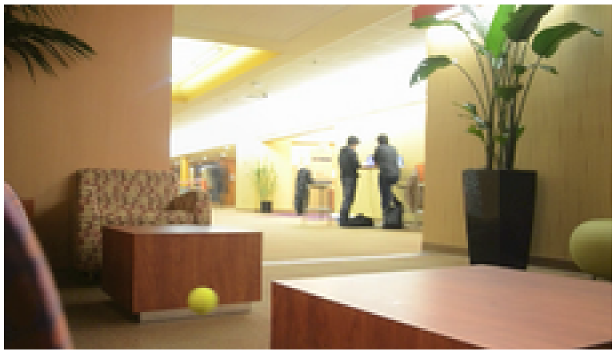
Figure 17: Complete static background frame, with the location of the person in the background replaced by an average across all other frames, with Laplacian Pyramid blending. Not the blurring / ghosting effect caused by the person being in that location through most of the frames.

Initial testing of replacing the object of motion with an average of frames resulted in visible seams between the background frame and the replacement pixels, even for a relatively static scene. We therefore used Laplacian Pyramid blending to smooth the seams. In addition, we found that creating a mask for the blending which is larger than the original motion tracking mask resulted in better blending results.

Combine the Motion and Static Frame

Given the static background frame and the motion mask for each frame in the input video, we can combine the motion pixels with the background frame. Using the motion masks for each frame, pixels of motion were extracted from the video frames (figure 18) and copied on to a copy of the static frame (figure 19) . The sequence of frames were then saved together into a GIF file.

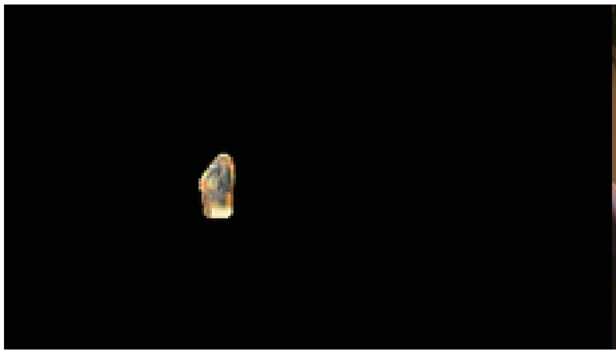
Figure 18: Pixels of motion

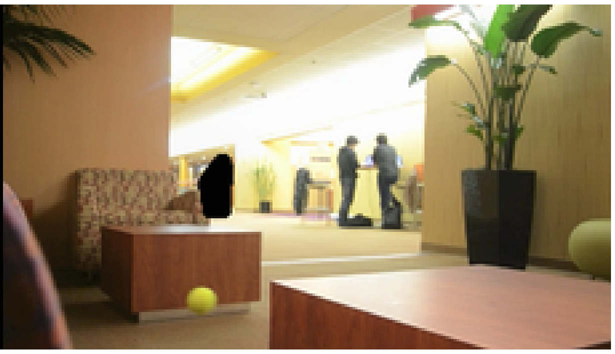
Figure 19: The static background frame

4. Experimental Results

To test our algorithms, we recorded various scenes around the University of Wisconsin campus. We aimed to record a variety of scenes and setups that would allow us to test different variations. To see the original videos and results, please visit our website at http://www.matthewbollom.com/cinemagraphs/results/

Tennis Ball Rolling on Table

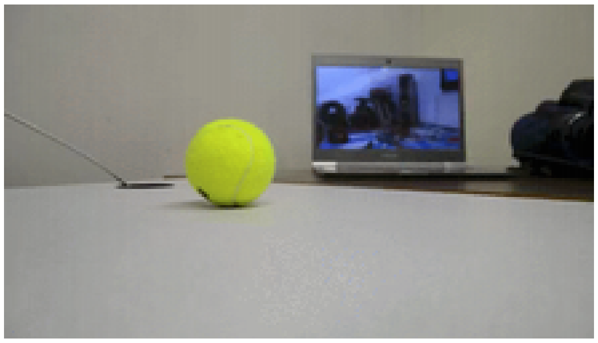

Figure 20: Screenshot from the “Tennis Ball Rolling on Table” video

This input video served as our initial test video. There are two sources of motion: the tennis ball rolling on the table and the cats on the computer monitor. We elected to make the cats on the computer monitor a small size in the input video as our main desire was to fully develop the motion segmentation and static frame generation. Our algorithm can correctly segment most of the tennis ball rolling across the table except for portions of the lower left corner of the ball. This may be due to the lack of color contrast in the tennis ball. In addition, the motion from the cats on the computer is ignored, except for a few frames when the tennis ball is in front of the computer. This addition is not noticeable unless you look closely, but demonstrates the need for an adjustable parameter to specify the border of pixels around the segmented object.

When creating the background frame for this test video, we found that replacing the object of motion with the average pixels could still result in artifacts from the object of motion remaining in the static frame. For example, the tennis ball left a faint yellow reflection on the table, which was not included as part of the region of motion. When we removed the ball from the frame, this yellow reflection remained in the static frame. We therefore modified our algorithm to replace a larger area around the object of motion with the average pixels and to use Laplacian Pyramid blending to combine the two images. After these changes were made, if the background frame that is selected has the ball present, our algorithm completely removes the tennis ball (and its associated artifacts) from the frame and fill the hole with pixels from the average image.

Reading the Newspaper

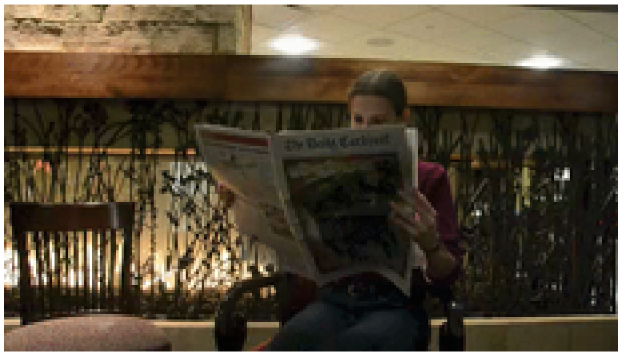

Figure 21: Screenshot from the “Reading the Newspaper” video

In this video, our intention was to have the subject read the newspaper while freezing the motion of the fire. Our algorithm produces a cinemagraph that segments most of the newspaper page turns, but also incorrectly perceives the burning of the fire as part of the newspaper page turn motions. This is because the two separate motions are occurring next to each other, causing the motion detection to believe they are the same motion. Another challenge for the motion segmentation with this video is the relative stillness of portions of the subject reading the newspaper which are not picked up by the algorithm as part of the object of motion - in this case, the subject's head. For most of the video, the subject's face remains in a relatively constant position, however, slight changes of angle in the head towards the end of the video result in a slight distortion of the face in the final output.

This is also a difficult test case for creating a static background frame. Because the paper essentially takes up those pixels for many frames, the average image has a ghost of the newspaper. When we combine the motion pixels with the static background frame, any pixels not taken up by the motion of the newspaper will have ghosting of the newspaper, creating visual artifacts.

Happy Person in the Computer Science Building

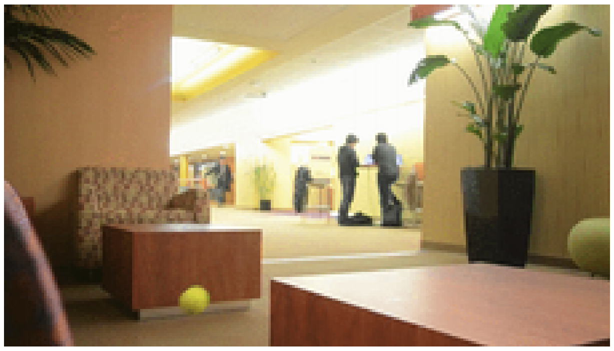

Figure 22: Screenshot from the "Happy Person in the Computer Science Building" video

This input video was used to describe our algorithm earlier in this paper. In this input video, there are two objects of motion occurring at the same time: the tennis ball rolling across the table and the person in the background walking towards the camera. The people standing at the table are also moving in the input video, but their motion is insignificant and not the focus of this video.

The result of freezing the tennis ball in midair and allowing the person to move is our best result to date. The person is segmented correctly, and copying the pixels of the person onto the static frame does not generate any visual artifacts. Part of the success of this cinemagraph may be attributed to the locations of the motion objects in the input video. Since the motions are far enough apart, their motion masks cannot join together and appear as one object.

When you examine the generated static background for this cinemagraph, a faint ghost of the person is visible. This is because many of the pixels the person appears over do not show the door and hallway behind him; therefore part of the average image must be used to fill those pixels. However, this is not a large issue because the motion is always occurring over these pixels and thus the ghost will always be hidden.

Nighttime Cars and Pedestrians

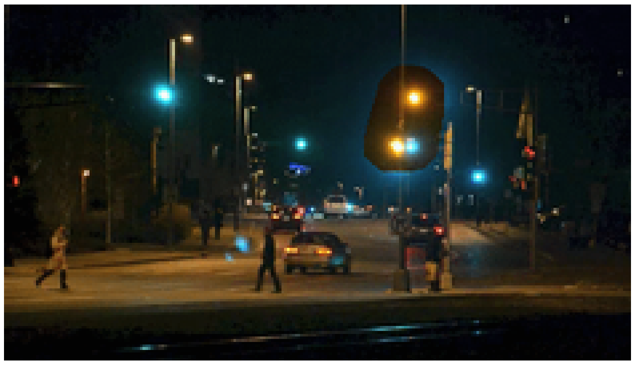

Figure 23: Screenshot from the “Nighttime Cars and Pedestrians” video

We also attempted to create a nighttime cinemagraph of cars and pedestrians near a traffic intersection. Our algorithm failed for this situation since the overall image was dark and thus subtracting the average frame image failed to produce meaningful results. We were able to create one cinemagraph of the traffic lights changing, but the result is unnatural due to the spreading of light. Because the traffic light is green for most of the input video, there is a green haze surrounding the intersection. When the algorithm copies the transition to yellow pixels onto the static frame, part of the green haze remains.

5. Future Work

We has identified several areas where future would could be done to improve upon our algorithm.

Motion Detection and Segmentation

One opportunity for future work is to better align the edges of the motion segmentation with the natural edges of the object of motion - even if the entire object isn’t in motion throughout the entire video. This could involve inserting a form of edge detection into our algorithm as part of detecting the object of motion.

Creation of the Static Background Frame

Our algorithm replaces the location of the object of motion with an average of all other frames, but we saw this cause problems with the “Reading the Newspaper” video where the average pixels pick up a “ghost” of the object in motion. Future work could instead aim to replace the pixels with the chronologically closest pixels of the unobscured background, instead of the average. However, this would need to be done through an efficient algorithm, so as not to drastically impact the overall running time of the cinemagraph creation.

Video Stabilization
This algorithm did not attempt to perform any stabilization on the original input video, instead relying on the video to have been taken from a static position using a tripod. Implementing video stabilization through warping with homographies could allow cinemagraphs to be created with a wider range of video inputs.

Looping
Our algorithm relies on the inherent properties of GIF animations to endlessly loop and does not attempt to ensure that the position of the object of motion in the last frame is compatible with the location in the first frame. Therefore an object may appear to "jump" between locations as the next iteration of the animation begins, as happens with the "Happy Person in the Computer Science Building" video.

Multiple Areas of Motion
Currently, the GUI we created limits the user to be able to animate one object in the video. However, future would could be done to relax this requirement to allow the user to animate multiple unconnected areas in the same cinemagraph.

6. Conclusion

Through the methods described in this paper we were able to automate the creation of cinemagraphs based on user-specified input in a graphical interface. We successfully separated the object of motion from its background using the difference between each frame and the average value, and we were able to combine the motion with a background frame created by blending the original frame and the average. This resulted in the creation of a viable cinemagraph. However, future work would need to be done to strengthen the results and to allow for cinemagraphs to be successfully create with a greater variety of inputs before this automation procedure could be considered commercially viable.

7. References

[1] Bai, J., Agarwala, A., Agrawala, M., & Ramamoorthi, R. (2012). Selectively de-animating video. *ACM Transactions on Graphics (TOG)*, *31*(4), 66.

[2] Joshi, N., Mehta, S., Drucker, S., Hoppe, H., Uyttendaele, M., & Cohen, M. (2012). Cliplets: Juxtaposing Still and Dynamic Imagery.

[3] Tompkin, J., Pece, F., Subr, K., & Kautz, J. (2011, November). Towards moment imagery: Automatic cinemagraphs. In *Visual Media Production (CVMP), 2011 Conference for* (pp. 87-93). IEEE.

[4] Yeh, M. C., & Li, P. Y. (2012, October). A tool for automatic cinemagraphs. In *Proceedings of the 20th ACM international conference on Multimedia* (pp. 1259-1260). ACM.

[5] Yeh, M. C., & Li, P. Y. (2012, October). An approach to automatic creation of cinemagraphs. In *Proceedings of the 20th ACM international conference on Multimedia* (pp. 1153-1156). ACM.

[6] http://en.wikipedia.org/wiki/Cinemagraph

[7] http://cinemagraphs.com/

[8] Flock, E. (2011, July 12). Cinemagraphs: What it looks like when a photo moves. *The Washington Post.* Retrieved from http://www.washingtonpost.com/blogs/blogpost/post/cinemagraphs-what-it-looks-like-when-a-photo-moves/2011/07/08/gIQAONez3H_blog.html.

[9] Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing* (3rd ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

8. Code Details
The code for this project can be downloaded from the project website. The implementation was done in Matlab and includes the following functions:

cinemagraph.m
This is the main driver function for the cinemagraphs program and was entirely created for this project. To run the project, run this code. It includes 50 lines of code.

CinemagraphGUI.m / CinemagraphGUI.fig
These files construct the GUI used in this implementation. It was built using Matlab's GUIDE tool and then manually modified to work with imfreehand and to return values to the driver program. It includes 238 lines of code.

MotionSegment.m
This file analyzes the video frames for motion by subtracting the average image across all frames from each frame. Additional processing occurs to detect all objects of motion within each frame. This file was entirely created for this project using previous works as guidance. It includes 44 lines of code.

TrackMotion.m
This file tracks the user's selection of the object of motion throughout the entire video. It was written for this project and contains 91 lines of code.

getBackgroundLP.m
This code creates the static / background frame used in the final cinemagraph. The code is based upon pyrBlend.m in Edward Wiggin's "image pyramid(Gaussian and Laplacian)" submission to the Matlab Central File Exchange, but was modified to calculate the average frame and create the masks used to specify the regions to blend. This function contains 63 lines of code.

The "image pyramid(Gaussian and Laplacian)" submission is also the source for the implementation of Laplacian Pyramid blending used by this method and in

CreateCinemagraph.m. The submission is available at http://www.mathworks.com/matlabcentral/fileexchange/30790-image-pyramidgaussian-and-laplacian

CreateCinemagraph.m
This files copies the pixels of motion on top of the static frame, blends the two, and saves the resulting frames to a GIF file. This file was created for this project and includes 49 lines of code.

9. Team Member Contributions
The contributions to this project were divided as follows:

Matthew Bollom:

- Wrote initial project abstract
- Developed and implemented motion tracking portions of the code, including *MotionSegment.m* and *TrackMotion.m*
- Developed and implemented combining of background frame with object of motion, including *CreateCinemagraph.m*
- Created project website

Samantha Kuray:

- Developed and implemented GUI in Matlab, including *CinemagraphGUI.m / CinemagraphGUI.fig*
- Developed and implemented creation of background frame, including getBackgroundLP.m
- Coordinated creation of presentation and paper

Joint Efforts:

- Development and implementation of driver program, including *cinemagraph.m*
- Writing project report