# Automated Inbetweening from Keyframes

Russell Chou
russchou@stanford.edu
December 10, 2010

## Motivation

Hand drawn 2-D animation has produced many wonderful works of art in recent decades. However, the general method for 2-D animation has not changed much, even as technology has advanced. Also, the prevailing method involves a great deal of non-artistic inbetweening, where junior artists fill in the frames between the keyframes drawn by a more senior artist. Inbetweening is often pretty routine and has a specific procedure. I hope that with a machine learning algorithm I can automate parts of this tedious process to speed up and reduce the costs of the animation process, as well as freeing up the artists to do more useful tasks.

## Previous research

There has been much work for a related problem, the correspondence problem. However, there are two distinct differences with this current task. Firstly, the source for most common applications of the correspondence problem are real world images, where each pixel can contain unique information to distinguish it from its neighbors. Also, physical models can be applied for reasoning about real world geometric phenomena.

There are two common approaches to the correspondence problem, correlation-based and feature-based. Correlation attempts to match specific image intensities of different corresponding regions of the image. Feature attempts to find corresponding features in both images. I will attempt both methods and see which one performs better.

For the second half of the inbetweening problem, producing new frames of animation, research suggests interpolation of the found features during the required time intervals. There are some different ways to interpolate, from simple linear to complex curves. I will start off with simple linear interpolation between correlated pixels and features.

## Procedure

I started off with samples of hand-drawn grayscale line art of a figure at various time steps. The algorithm will take these images and find significant points in each image. Then, using a regression model, it can extrapolate the locations and changes of those significant points into the images for the inbetween frames of given time steps.

## Corner Detection

In order to find useful features for the algorithm to use, I used a Harris & Stephens or Plessey corner detector algorithm. This consists of a few steps:

1. Calculate the local horizontal and vertical derivative for every pixel in the image using Gaussian convolutions (Sobel operator).
2. Use them to get the convolution matrix:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

$$where: A = \left(\frac{\partial I}{\partial x}\right)^2, B = \left(\frac{\partial I}{\partial y}\right)^2, C = \left(\frac{\partial I}{\partial x}\frac{\partial I}{\partial y}\right)$$

3. Determine the "cornerness" of each pixel using:
   $C(x,y) = \det(M) - k*\mathrm{trace}(M)^2$
   $\det(M) = \lambda_1 \lambda_2 = A*B - C*C$
   $\mathrm{trace}(M) = \lambda_1 + \lambda_2 = A + B$
   $k$ = constant = .04
4. Threshold the cornerness values, used 1e10.
5. Find to local maxima to get the x,y coordinate for each coordinate.

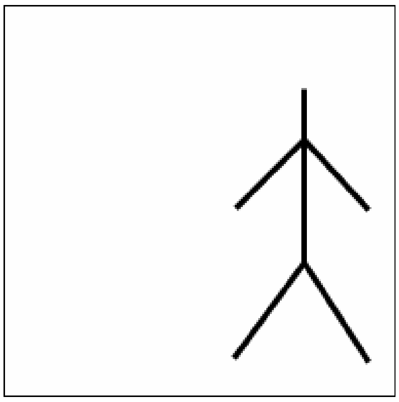
This gives me significant points in the form of x and y coordinates of corners. Once I have the corners for each image, I correlate them using proximity, so that I get a set of points that change location through each keyframe.

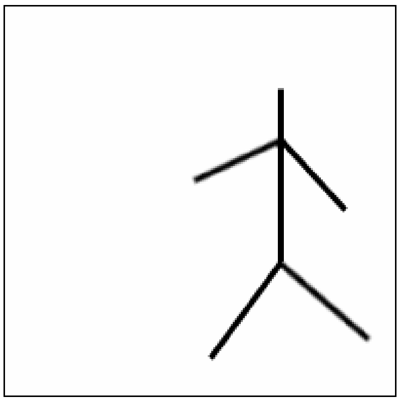
**Weighted Linear Regression**

With keyframe data on all the corners in the images, I proceed to use weighted linear regression on each corner. The only independent variable is time, and each corner has two dependent variables, x and y, which are assumed to be independent with each other and with other points. Therefore, I perform independent weighted linear regression on each x and y coordinate of each point, and input the inbetween time steps into each model to get corner coordinates for each inbetween frame.

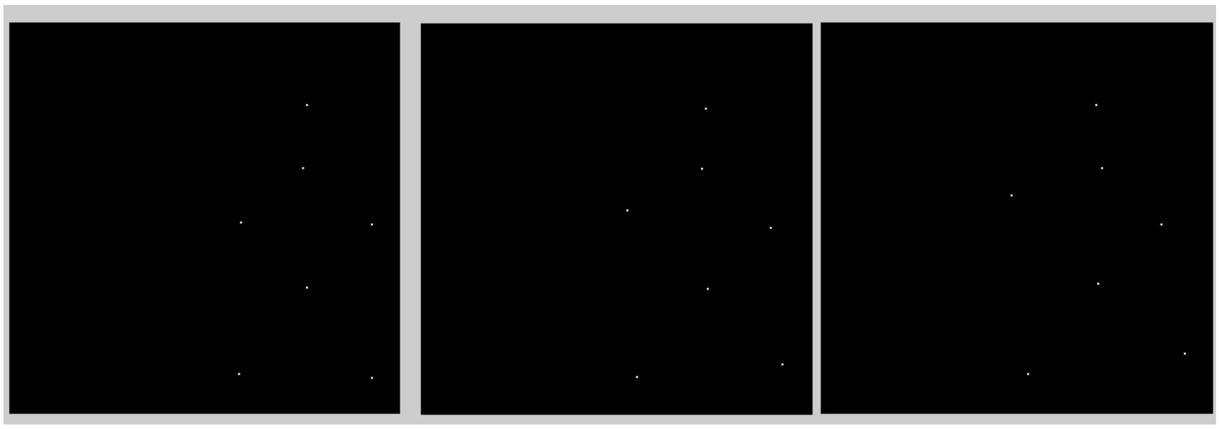
**Tests**

Input


time = 0


time = 1

Output


time = 0 time = .5 time = 1

(white dots are detected corners)

From this test, we see that the algorithm can find the corners (significant points) and then move them for the appropriate frames. This can be done with any timestep, and even past the ends of the keyframes, but that would probably get undesired behavior.

## Potential Problems

Edge detection for this category of images worked pretty well, since they were high contrast with low noise. However, this only works well with straight lines. The edge detector does not work well with curves and either gets too many corners or too few. This is a liability since either the curve will be missing or the curve will behave very chaotically.

Also, the algorithm at the moment cannot handle significant points disappearing or appearing. Future research could be done to handle appearing and disappearing features.

## Future Research

At the moment, I do not have a way to turn my significant points into edges in the generated frames. Future research could be spent on following edges or calculating which corners have edges inbetween them.

Another option for future research is to try with different algorithms for correlation such as using ordering instead of proximity. That way, keyframes can be more widely spaced apart, and the significant points should still be able to maintain correlation. Also, different equations instead of regression can be used to find inbetween coordinates, such as splines or other interpolation.

## Conclusion

For simple greyscale lineart, corner-detection plus weighted linear regression works well for finding and then extrapolating the significant features in inbetween frames. However, more work is to be done to extend this algorithm to handle curves and edges.

## References

Hutton, Jamie and Dowling, Ben. "Corner Detection." Computer Vision Demonstration Website. http://users.ecs.soton.ac.uk/msn/book/new_demo/corners/

Parks, Donovan and Gravel, Jean-Philippe. "Harry/Plessy Operator." Corner Detectors.
http://www.cim.mcgill.ca/~dparks/CornerDetector/harris.htm