Approximating Images with Epicycles

Barry Liu and Edward Yang

June 2018

Contents

1 Introduction

For centuries, the concept of epicycles—paths composed of circles moving along other circles—have been used to describe astronomical motions, ranging from the motions of our own moon to those of Venus or Mars. Interestingly, this same concept has a very convenient mathematical formalism that bears many similarities in form to that of the Fourier transform. As we will show in this paper, this means that we can apply much of the same logic and methodology of the Fourier transform to epicycles. As a result, epicycles, while seemingly reasonable to describe astronomical paths for their time, are actually capable of describing any path, and do not actually provide much insight into the true motions of astronomical objects. While inconvenient for the ideas of classical astronomers, this same result means that we can use epicycles to create approximations of images, the main application of this paper. We take advantage of modern computing capabilities and algorithms to accomplish this, in the form of mathematical computing libraries, graphical libraries, and custom algorithms. Our methodology produces good quality results that provide recognizable approximations of input images. Even still, there remain improvements to be made and further exploration of this idea may be undertaken in the future.

2 The Fourier Transform

The Fourier transform takes a function as an input and outputs another function, but the output function has an independent variable different from the input. In this case, the Fourier transform takes in a function of time such as a waveform or signal and decomposes the function into a function of frequency. The domain of the input function is the time, and the range is the intensity/amplitude, whereas the output function is represented with a frequency domain. Frequency domain refers to the analysis of a signal with respect to frequency, instead of time. This means, rather than showing how a function changes over time, it shows how much each frequency is present in the original function.

Fourier's Theorem states that a periodic function can be rewritten as the sum of sinusoidal functions, also known as a Fourier Series. The Fourier transform applies the ideas of the Fourier Series to non-periodic functions.

2.1 The Equations

The Fourier Transform:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

The Inverse Fourier Transform:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i x \xi} d\xi$$

The Multidimensional Discrete Fourier Transform (DFT):

$$X_k = \sum_{n=0}^{N-1} e^{-2\pi i k \cdot (n/N)} x_n$$

2.2 How Does it Work?

The Fourier transform of a function $f(t)$ is defined by

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i x \xi} dt$$

The result is a function of frequency $\hat{f}(\xi)$. $\hat{f}(\xi)$ tells us how much power $f(t)$ contains at the frequency ξ, and is also known as a power spectrum. The sign of the complex function is arbitrary convention and can be changed accordingly.

In order to simplify the concept, we can represent the signal in a different way by wrapping it around a circle, with the radius at time t, being equivalent to the amplitude at time t of the original function. In this situation, we have two frequencies. The frequency of the signal, measured in beats per second, and the frequency at which we are rotating around the circle, measured in cycles per second.

If we track the average value of the Fourier transform by graphing the average value as the distance from the origin against the frequency, we notice that when the frequency at which we are wrapping matches the frequency of the signal, there is a spike in the distance from the origin at the frequency. Doing this for a signal will reveal every frequency that is present and in what proportions. If we use the inverse Fourier transform on the distance vs. frequency graph, the result will be the original function.

2.3 Why Does it Work?

The Fourier transform can be derived from the Fourier series using trigonometric identities. The Fourier transform is the Fourier series extended to infinity. The Fourier series is used to represent a periodic function by a discrete sum of complex exponentials, while the Fourier transform is then used to represent a general, non-periodic function by an integral of complex exponentials.

In a classical approach, it would not be possible to use the Fourier transform for a non-periodic function. The use of generalized functions, however, frees us of that restriction and makes it possible to look at the Fourier transform of a non-periodic function.

2.4 Applications

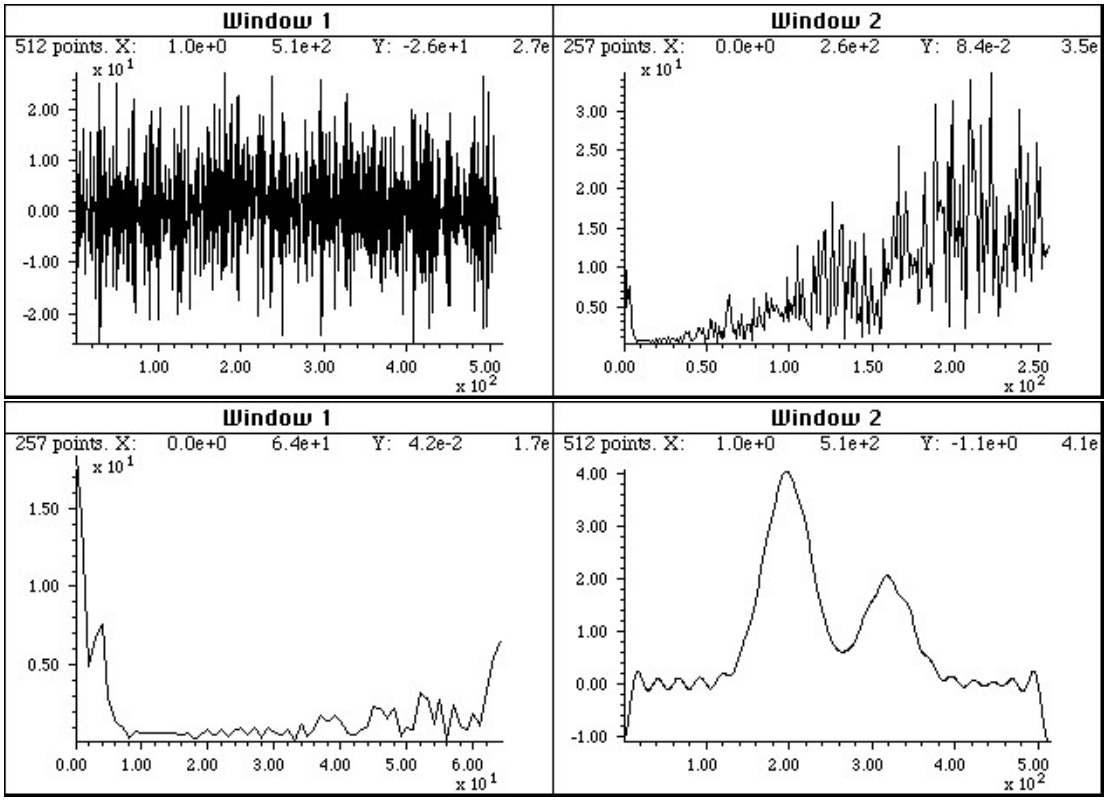
The Fourier transform has a lot of direct applications to fields such as digital signal processing and physics. In signal processing, wave functions like signals can be broken down into their constituent frequencies and modified. In physics,

it is common for quantum physicists to switch between position and momentum functions through the use of a Fourier transform. In both of these situations, the inherent properties of the Fourier transform make the problem much simpler to solve.

2.4.1 Sound Processing

The Fourier transform is a very important tool in sound processing. Much of the sound we hear is generated by a variety of audio machines. The information these machines use to create sound is usually stored and represented digitally which allows it to be easily manipulated by computers.

A common usage of the Fourier transform is blocking out certain unwanted frequencies through the use of a Fourier filter. The Fourier filter works by taking the Fourier transform of a signal, attenuating specific frequencies, and finally using the inverse Fourier transform to produce the desired signal with the unwanted frequencies removed.



In the top-left is a signal of what appears to be random noise. The picture next to it is the power spectrum of the signal. A power spectrum shows

the distribution of the energy of a waveform among its frequencies. In other words, it shows the intensity of each frequency in proportion to the others. The bottom-left picture expands the power spectrum and more clearly shows the low-frequency region. The Fourier filter can be used to remove the higher harmonics and using the inverse Fourier transform, we can reconstruct the signal. The result (bottom-right) shows that the signal contained two bands at x=200 and x=300 that were completely obscured by the noise in the original signal.

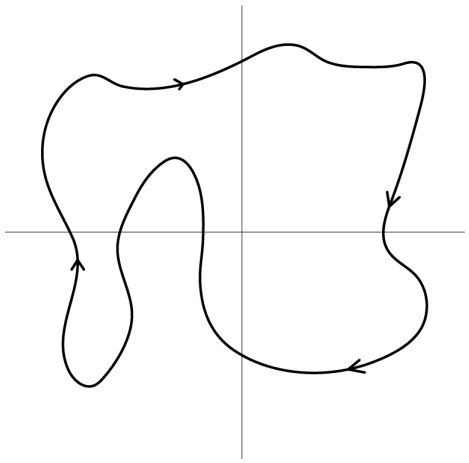
2.4.2 Heisenberg Uncertainty Principle

Another common application of the Fourier transform is Heisenberg's Uncertainty Principle. Heisenberg's Uncertainty Principle states that the position and velocity of a particle cannot simultaneously be known, and that they must obey a specific inequality, which states that the product of their uncertainties must always be greater than or equal to a constant. This means the more precisely defined one is, the more uncertain the other is.

In Physics, it is common for physicists to switch between particles' functions and the Fourier transform of their functions, because particles with a lot of momentum and energy will also have high frequencies. In order to move between position and velocity bases in quantum mechanics, a Fourier transform is required. If you have a wave function in coordinate-space and you want to find the wave function in momentum-space, you Fourier transform the coordinate-space wave function. The inverse Fourier transform can also be used to go from a momentum-space wave function to a coordinate-space wave function.

3 Extracting Fourier Coefficients for Epicycles

Let $F(t)$ be some closed function such that $F : \mathbb{R} \to \mathbb{C}$.



By Fourier's Theorem, we can express any such $F(t)$ as a sum of sines or cosines. Let T be the period of the function. Thus:

$$F(t) = \sum_{n=0}^{\infty} \left[B_n \cos\left(\frac{2\pi nt}{T} + \phi_n\right) + C_n i \sin\left(\frac{2\pi nt}{T} + \phi_n\right) \right]$$

If we allow $A_n \in \mathbb{C}$:

$$F(t) = \sum_{n=0}^{\infty} \left(A_n \left[\cos\left(\frac{2\pi nt}{T}\right) + i \sin\left(\frac{2\pi nt}{T}\right) \right] \right)$$

Applying Euler's Identity:

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

We find:

$$F(t) = \sum_{n=0}^{\infty} A_n e^{\frac{2\pi i t n}{T}}$$

This bears a very clear resemblance to the inverse DFT, denoted as $\mathcal{F}^{-1}[k(x)]$.

$$\mathcal{F}^{-1}[k(x)] = \sum_{x=0}^{N-1} k(x) e^{\frac{2\pi i x}{N}}$$

In this case, $A_n = k(x)$, and we wish to find the coefficients A_n. Thus, because $F(t)$ matches the form of the inverse DFT of these coefficients, we can simply apply the DFT to $F(t)$ to extract these coefficients.

$$A_t = \mathcal{F}[F(t)] = \sum_{t=0}^{N-1} F(t) e^{\frac{-2\pi i t}{N}}$$

Thus, we now have A_t. $A_t = a + bi$, where $|A_t|$ represents the amount of the frequency (aka the radius) present in the function, and b represents the phase offset, in that $\arctan(\frac{b}{a}) = \phi_t$. We can then use these values to draw circles with these specific frequencies of rotation and associated radii and phase offsets. This is essentially performing a graphical version of the inverse DFT, which will trace our original image.

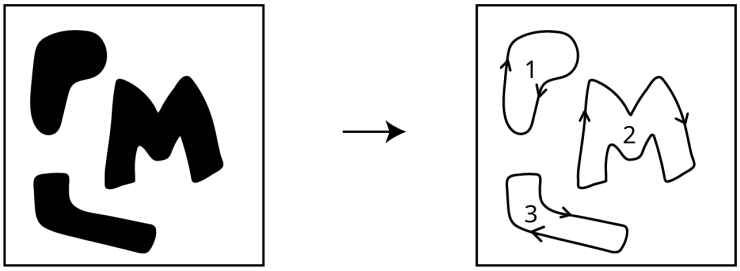
3.1 Approximation-Specific Issues

As we can see from the differences between our original function and the DFT, we are no longer adding an infinite series. This means that we are only getting an imperfect approximation of our original function. Even still, in order to get a good sample of data, we need a large number of original points. However, using the DFT, the number of input datapoints is also the number of output datapoints. This large number of equivalent output circles in our

epicycle presents a challenge to draw quickly, and so we only take a subset of the output circles to draw with. These circles are chosen to be those with the highest intensities (largest radii), as these are relatively higher weighted than the others, and thus would contribute more to any potential error. Because of particularities in the FFT (a way of calculating the DFT quickly) function built-in to the python libraries we used, we use the `fftshift` function to shift our output, which results in an output range of frequencies from $-(T-1)/2$ to $(T-1)/2$. This means that we can extract the circle with a frequency of 0 and use it as our offset, as having a static circle drawn in the center of the animation tends to be distracting to the viewer.

4 Converting Images to Sampleable Discrete Functions

We can use our library functions to read in arbitrary images. After reading in these images, we first convert these images to black and white with the `binarize` function. After we have a binarized image, we can use library built-ins to find the contours of the image. These library functions use the marching squares algorithm to do so. This leaves us with a set of contours, each of which is defined as an ordered set of points.



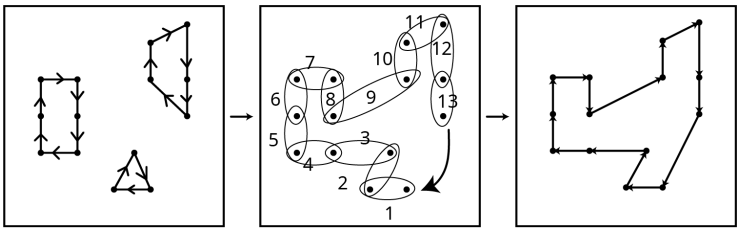
However, this set of contours still does not satisfy our need for a single traceable function. In order to create a single traceable function from these contours, we need to connect each of these contours together in a way that makes sense. When approaching this problem,we can clearly see that it is an instance of the travelling salesman problem for the shortest path through all of the points in the contours. The TSP is known to only have algorithms for optimal solutions that run in $O(n^2 2^n)$ time. Given that the number of points present in each of these contours is on the scale of thousands of points, using one of these exact algorithms for this problem is entirely unreasonable.

4.1 Greedy Algorithm

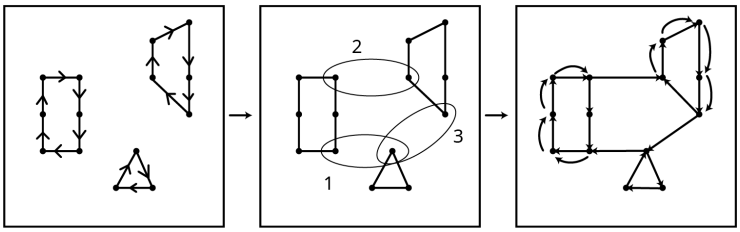
We instead choose to connect the contours with a sub-optimal algorithm. This algorithm is a greedy algorithm, where we pick an arbitrary starting point,

and connect it to the next closest point, and so on and so forth, until we connect all points together, and then connect back to the original. In the interest of keeping the appearance of the original shapes, we can use the ordering of each individual contour, and simply find connecting points between the contours and use them to find how to rearrange the existing contours. We illustrate this variation between a simple greedy algorithm and our method below.

Simple Greedy Algorithm:



Our method:



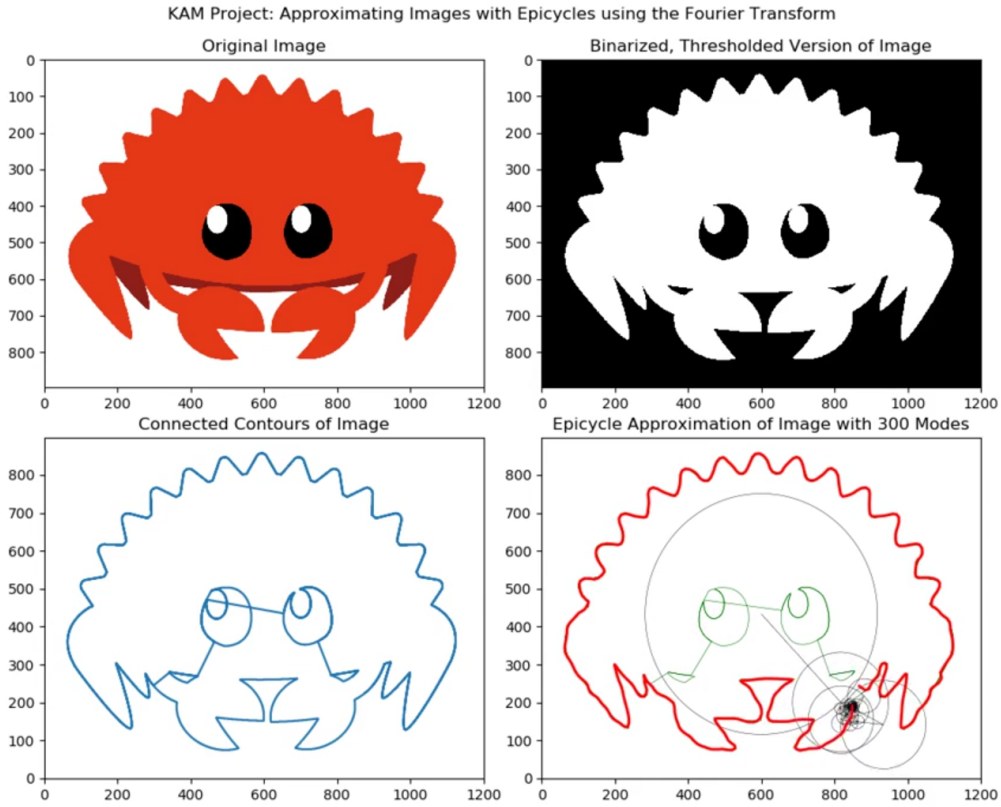
4.2 Unresolved Computational Issues

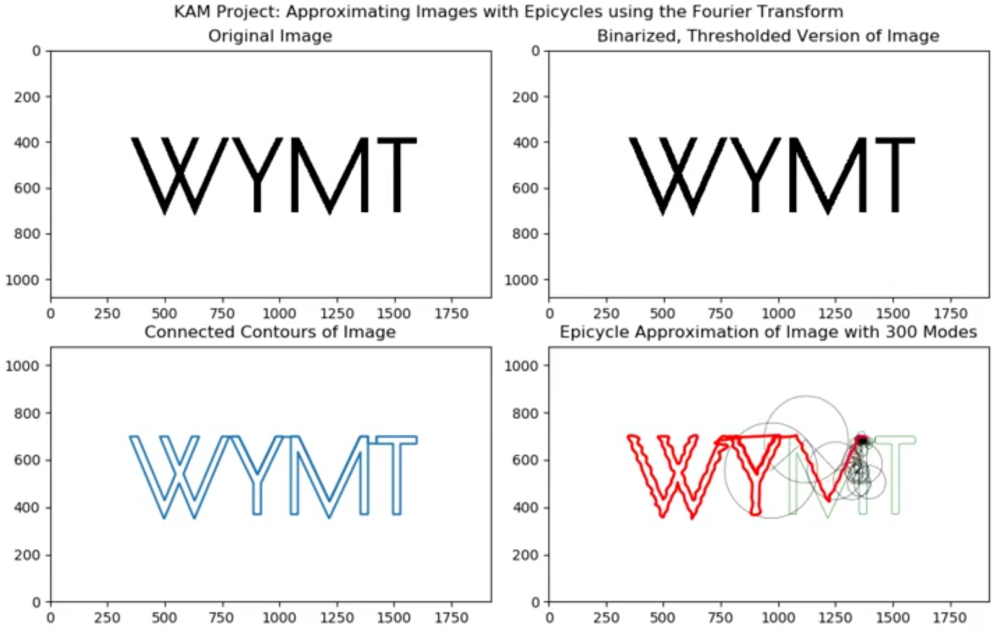
As one can clearly observe from the above, we duplicate some parts of each contour in our method so that the closedness of each individual contour is preserved, and the relative ordering of each point in the contours is preserved. This allows us to sample the function to trace the original more accurately. This algorithm has a runtime of $O(n^2)$, which is a vast improvement over the runtime of the algorithm for the optimal solution. To calculate the distances quickly, we take advantage of vectorized operations and complex numbers to calculate

$$D(p_1, p_2) = |p_1 - p_2| = |(x_1 + iy_1) - (x_2 + iy_2)|$$

extremely quickly in python. However, this method, while fast and easy to implement, has the disadvantage of having a space requirement of $O(n^2)$. This means that images with many contours of many points in their contours may not be able to actually fit in memory, and thus crash the program. This could easily be a point of optimization in the future, but is currently not handled.

5 Example Results





6 Conclusion

As the screenshots of our example results show, our method does a fairly reasonable job of approximating the shape of input images, especially given the computational limitations and compromises that we made in the interest of saving time. The possibility for improvements, computationally, algorithmically, and graphically, cannot be any more overstated. Although it was not touched upon as much, the Fourier transform can be extended to even higher dimensions, and this provides the idea of using the Fourier transform not just to approximate the contours of images as we did here, but to approximate the entire image itself, including color. Whether this is actually a space-effective form of compression is an entirely different question, and is a likely place of exploration in the future, although the usage of Fourier transforms in JPEG compression seems to provide evidence that this is in fact useable. The applications of the Fourier transform extend far beyond the scope of this paper, and it finds especially great use in fields such as physics, signal processing, image processing, and more, and we hope that this is just a stepping stone for what's to come.

Bibliography

[1] 3Blue1Brown. But what is the fourier transform? a visual introduction.

[2] Brett van Zuiden. Drawing by epicycles.

[3] Mathematica Stack Exchange. How to create a new "person curve"?

[4] Mathematica Stack Exchange. Fourier series and epicycles - how to find the radii and angular velocities from a function's fourier series expansion.

[5] Michael Trott. Making formulas... for everything—from pi to the pink panther to sir isaac newton.

[6] sclereid. About epicycles.

[7] Wikipedia. Deferent and epicycle.