Script

INTRO

Hello! Today we’re going to explore the Fourier Transform through the context of sound. Now, having no interest in boring you to death, we won’t tackling equations, nor will we be getting caught up in any details. Resultantly, this video wont require any prerequisites. However, that also means I will be leaving huge fields of some really powerful and beautiful math untouched. But even without the details, a meaningful qualitative and applicational understanding is still obtainable. However, if you happen to come across something you found interesting and would like to dig a little deeper, I will be providing links to some excellent resources that unearth much more sophisticated and powerful details. Now without further ordure, let’s get started.

ROADMAP

First, a quick roadmap of what’s to come.

We’ll First tackle the basics of exactly what sound is.

From there, we can begin to mix sound together and observe the subsequent problem of Convolution.

Than ultimately, we’ll use convolution to introduce and explore The Fourier Transform

Lastly, I made neat little program to hit this all home.

WHAT IS SOUND?

Given that this entire video will be explored in the context of sound. It’s worth taking some time to understand exactly what sound is. Condensed into a single sentence, sound is the propagation of pressure oscillations across a medium. Alright, So perhaps the most concise explanation isn’t always the most helpful. It’s much more helpful to simply to see it. The somewhat mesmerizing areas of high pressure moving across the medium (air for example) are picked up by our ears and interpreted by our brains into the noises we all know and love. This makes for an accurate model of a sound wave, but It is also relatively cumbersome and difficult to quantify. In the interest of simplifying our representation of a wave, imagine placing a microphone within our medium. A microphone is very much like a barometer in that it measures air pressure…just 1,000s of time a second. The data from microphone tells us that the pressure around it is oscillating back and forth. Plot the data across time, and you form a wave, hence the name “sound wave”. But notice that by changing the manner in which we represent the wave, information about that wave becomes more easily accessible. For instance, watch as I first play with the magnitude of the sound…or volume, then frequency, or pitch. The changes are arguably more apparent in the wave representation rather than the particle representation. However, possibly the best perk of this new representation is our new-found ability to add (or mix) sound wave. And luckily, doing so is very straight forward. First, we’ll need two waves (or rather two sounds) in which to add. We’ll represent both waves using our hypothetical microphones. At any given instant, the pressure formed by our new wave…our sum, can be found simply by adding the pressures of each of the parts. Consequentially, if you’re given drawn-out waves to work with, finding the sum is as simple as sampling each wave a regular interval, than returning the sum at each interval. Do this with enough samples, then you get a good idea to what your sum of waves should look like. Before we move on from here, let’s take a quick step back to appreciate what happened. When contemplating the addition of sound waves, representing our sounds literally as particle systems…well, it’s not helpful. Rather, our solution comes from finding a new way to represent our waves. That’s worth recognizing as this theme will reappear within this this video.

ADD TWO WAVES (\*\*EXPLAIN CONVOLUTION IN ADDITION OF WAVES\*\*)

But not everything is fine and dandy, Notice that when adding waves together, the result is convoluted slightly. Looking at the result, it may not be obvious what frequencies had to have made it up.

ADD N WAVES (\*\*EXAGERATE CONVOLUTION\*\*)

When only adding two waves, the convolution may not seem problematic. However, when multiple frequencies are mixed…well, the result may no longer bear ANY APEARENCE to any of the waves that built it up. This convolution will be our primary antagonist.

BUILDING SOUNDS

But I’m getting ahead of myself. First, why should we worry about mixing sounds in the first place? Well…One obvious context…would be in the construction of musical chords. If we chose three frequencies to add together at random, its likely that our result would be unpleasant and dissonant. However, if our frequencies happen to relate to one other with a simple ratio, than the resultant sound is harmonious. It’s actually a rather neat property of sound.

MUSICAL INTERVALS

The particular ratio in which two notes are related to each other is called an interval. Mixing and matching varius intervals produces various Chords. Importantly though, each interval represents a simple ratio to the root note, as shown in the table. It’s the simplicity of those ratios that gives the subsequent chords their harmonicity.

INTRODUCE ALGORITHM

Alright, So then If I SHOW you two sounds by their graphs only. Could you then tell me weather or not each of them should sound harmonious or dissident? I’ll give you a second to guess, or better yet, devise a strategy. (\*\*Play SOUNDS\*\*) Did you get it? If so, what were you looking for? To ensure we’re tackling this problem analytically, let's look for a set of steps…an algorithm that would solve this problem. It seems like a rather daunting task, but how can that be? If we understand exactly what makes sounds harmonize, well…than what’s wrong, what’s missing?

ADD THREE GRAPHS

Well…using frequency information to determine harmonicity is only useful…well, if you have frequency information. However, that luxury is rarely offered to us in real life applications. If you were to record some collection of sounds, the microphone simply shows you the summed-up waveform, NOT the individual frequencies that made it up. Although we understand how to add, or mix sounds together, now we need to somehow work backwards. That is, we must figure out of to dissect a waveform into its component frequencies. This is made difficult becasue waves become convoluted under addition.

CHECK RATIOS

If we can figure it out, then devising our hypothetical algorithm becomes simple. If our sound is harmonious, than upon breaking it up into its individual frequencies, we would than find that the frequencies relate to each other with some simple ratio. And if our sound is dissonant, meaning it isn’t harmonious, than we would find that they do NOT. Alright, so than it boils down to how exactly can we break down a waveform into its individual frequencies?

PRE-RECS

The solution lies within the Fourier transform. Although the math behind it is a necessity for computing the transformation, it really isn’t important for understanding what it does, or even vaguely how it works. So leaving the equations aside, let’s explore the Fourier Transform

FOURIER VISUAL

Imagine taking a wave, 1 Hz for simplicity. But rather than graphing it along a line, imagine sending it around in circles about the origin\*\*PASUE\*\*. Importantly, in doing so, we have a degree of freedom here, the frequency at which we send it about origin can be whatever we’d like\*\*PASUE\*\*. By playing with that frequency, we can learn valuable information about our wave. For purposes of clarity, ill extend the length of the wave. Then ill show you the “Average Value” of our wounded graph. Perhaps visualize it as a center of mass of sorts. Then pay attention to this average value as I change the frequency at which I wind the wave\*\*PASUE\*\*. You’ll find that the graph balances itself out such at the Average value is always near the origin\*\*PASUE\*\*that is with one notable exception. You see, when the frequency at which the wave is wound about the origin matches the frequency at which the wave itself oscillates, all of the peaks of our wave appear at the same point, and Instead of balancing itself about the origin, the average value reaches a maximum value from the origin\*\*PASUE\*\*. Now, in itself, that might not be all that impressive. However, watch what happens when we give it a slightly more complicated input. A sum of a 1 and 2 Hz wave for example. However, nothing else about the process changes. Same as before, send the wave in circles and watch the Average value as we change the frequency at which we send it around the graph\*\*PASUE\*\*. Amazingly, although the input signal is convoluted, the resultant graph remains unconvoluted and still tells us information as to which frequencies are present within the original sound. Peaks In our graph occur at every frequency present within our sound. And here, we’ve found our proverbial golden ticket.

SIMPLE EXPLANATION

Shifting to a more applicational perspective; the Fourier Transform converts from a time domain to a frequency domain. It represents our sound in a new way. Just as moving from a particle representation to a wave representation enabled us to mix sounds together. Now the Fourier transform allows us to represent our sounds in a way that isn’t vulnerable to our antagonistic convolution. Our original sound is shown in a time domain, meaning we see TIME plotted along the x axes. The Fourier Transform simply transforms it into a frequency domain, meaning we see frequency plotted along x axes. There are 3 names you’ll most often come across regarding the transformation. First, there’s simply the Fourier Transform. It has one short coming, that being it only operates on a perfect continuous set of data, like a mathematical sine function for example. That problem is solved by the second name you might come across, the Discrete Fourier Transform; Also called the DFT, its just an implantation for the Fourier Transform that operates on discrete sets on data, like the pressure data samples from a real-life microphone. Then lastly, there’s the Fast Fourier Transform. Most commonly referred to as FFT, It a particular implementation of the Discrete Fourier transform that is SIGNIFCANTLY faster than the older naive algorithm, especially when operating larger data sets. But most importantly for the purposes of this video, you need to know that the Fourier Transform converts from a time domain to a frequency domain.

CHECK RATIOS 2

Finally, we are ready to solve our initial problem. Our hypothetical algorithm will first apply the Discrete Fourier transform to pick out the frequencies within our sound, then check If these frequencies relate in some simple ratio…or in reality, at least come close enough to fool our ears. Then that’s it! The difference between a daunting problem and a trivial problem was how we chose to represent our data. By representing it in the frequency domain with the help of the Fourier Transform, it becomes trivial.

Noise Filter

Our hypothetical algorithm was one of MANY places where the Fourier Transform shines. Another one, relevant to the making of this video is it’s capacity for noise reduction. Among the many additional features of the Transformation that we haven’t discussed yet, are two that make it a great candidate for noise reduction. First, it isn’t easily thrown off by imperfections such as noise. Let’s listen to a C Major, notice that the noise is very distracting. Additionally, its very difficult to see how the graph is supposed to look, so it isn’t obvious how we could repair the sound…unless you guess using the Fourier Transform. The Frequencies are still picked our by the transformation, and the static makes it’s into the transformation via small bumps across otherwise flat portions of the graph. By having a computer flatten those areas, you’ve eliminated the noise. From here, the other one of the two properties that can save us, the Inverse Fourier Transform. The Fourier Transform can be undone just as easily as it was applied. So after the noise is filtered out, you can undo your transformation and be left with your original sound\*\*PASUE\*\*…except without the noise. To further exemplify, \*\*\*\*This is the raw audio output from my microphone, it contains static that can range from annoying to downright distracting. \*\*\*\*However, thanks to the Fast Fourier Transform my software can clean it up for me. In a very similar you could remove an annoying BG high pitch from your sound too.

More Applications

Believe me when I say that we haven’t even scratched the surface when it comes to exploring all the applications and math involved with the Fourier Transform. The applications alone stretch FAR beyond anything that appears even related to frequencies and sound. As far as I’m concerned here, this video is nothing more than an introduction. Partially because I’m not sure how much I can add in the way of details that hasn’t already been wonderfully laid out in plethora of free online resources, my favorite of which you can find in the video description.

Interactable FFT demonstration

Alright, before I let you go, there’s one last thing I have to offer. In the video description below, you’ll find a link to a windows forum program I developed for this project. Because of course, learning is about more than just listening to someone else ramble. The program is relatively straight forward to download and use. However, I will have an additional upload showing the installation and use, linke in the description. And with that, I invite you to play around, get comfortable with the Fourier Transform. Demonstrate its usefulness yourself. Maybe even follow along this video again with the tool by your side. But most importantly…THANKS FOR WATCHING.