Music, it turns out, is digitally encoded as just a long list of numbers. In an uncompressed .wav file, there are a lot of these numbers - 44100 per second per channel.

A channel is a separate sequence of samples that a speaker can play. Think of having two ear buds - this is a “stereo”, or two channel, setup. A single channel is called “mono”. Today, modern surround sound systems can support many more channels. But unless the sound is recorded or mixed with the same number of channels, the extra speakers are redundant and some speakers will just play the same stream of samples as other speakers.

**Sampling**

The mysterious choice of 44100 samples per second seems quite arbitrary, but it relates to the [Nyquist-Shannon Sampling Theorum](http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem).

we need to sample twice as frequently as the frequency we want to see to make sure we’re detecting it.

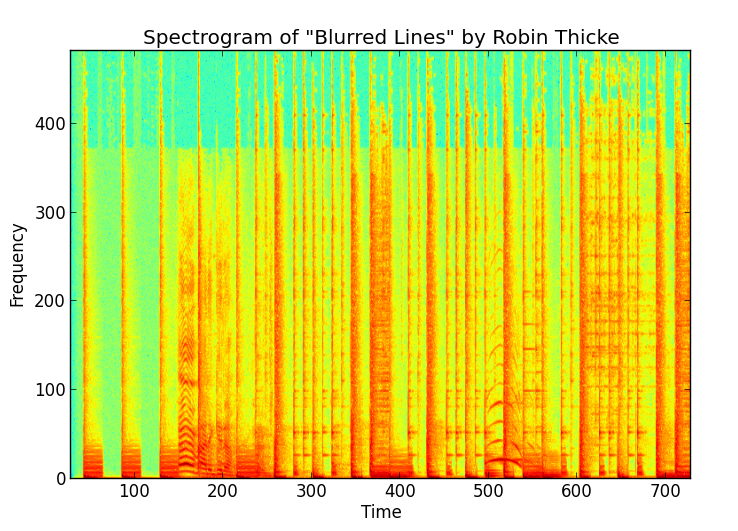
In the case of recording audio, the accepted rule is that we’re OK missing out on frequencies above 22050 Hz since humans can’t even hear frequencies above 20,000 Hz. Thus by Nyquist, we have to sample twice that:

Samples per sec needed = Highest-Frequency \* 2 = 22050 \* 2 = 44100

The MP3 format compresses this in order to 1) save space on your hard drive, and 2) irritate audiophiles, but a pure .wav formatted file on your computer is just a list of 16 bit integers (with a small header).

## Spectrograms

Since these samples are a signal of sorts, we can repeatedly use an FFT over small windows of time in the song’s samples to create a [spectrogram](http://en.wikipedia.org/wiki/Spectrogram) of the song.Ex.



It’s important to note that the frequency and time values are discretized, each representing a “bin”, while the amplitudes are real valued. The color shows the real value (red -> higher, green -> lower) of the amplitude at the discretized (time, frequency) coordinate.

Now that we’ve got a specrogram of our audio signal, we can start by finding “peaks” in amplitude. We define a peak as a (time, frequency) pair corresponding to an amplitude value which is the greatest in a local “neighborhood” around it. Other (time, frequency) pairs around it are lower in amplitude, and thus less likely to survive noise.

hash(frequencies of peaks, time difference between peaks) = fingerprint hash value

First, notice we have not only a hash and a song ID, but an offset. This corresponds to the time window from the spectrogram where the hash originated from. This will come into play later when we need to filter through our matching hashes.

Now when we have to match two fingerprints:

We make a dictionary having the offset values