Program description:

The program will let you choose a wave file on your device, open it up and then choose whatever frequencies you want and modify their weights. After that you can save the file or save as.

Features:

* Displays a graph of the fourier transform of the file that gets updated to reflect the changes you made to it
* Displays a graph of the sound waves that gets updated to reflect the changes you made to frequencies as you make them
* Click on the fourier transform graph to select a frequency, then it lets you choose how much you’d like to multiply by or set an absolute value. Also lets you choose a radius around that point that you’d like to affect. Frequencies in that area of the point you chose get multiplied, by the chosen factor at the point itself and interpolated up/down to 1 at the end of the area.

The program supports WAVs with bit-depths of 8, 16, 24, or 32, and sample rates of 8000-48000. The program only supports WAVE\_FORMAT\_PCM files. Internally, samples converted to double-precision floats. When saving, conversion is done back to integers with dithering.

In memory, the PCM will be saved as a sequence of arrays, since the total memory can get too big to store in one contiguous block. Since the max new file length is 3600 seconds and the max frequency is 48kHz, that makes the max samples we might store equal to 3600 \* 48000. This will take 3600\*48000\*8 bytes of memory. Maybe capping each array in the sequence at a fifth of that will be a good way to go.

I should be wary of overflows when doing arithmetic. Using saturated arithmetic may be wise.

Apparently 8-bit WAVs don’t use two’s complement while larger sizes do. I should be wary of this.

Roadmap:

1. Plan GUI

2. Ability to open wave files and save/save as.

3. Temporary GUI for choosing a frequency and a multiplier and applying the modification without the fancy graphs

4. Draw graphs

5. Make frequency selectable from graph

Notes:

* If a chunk’s size is odd, there’s an extra padding byte at the end of it.
* Chunk size does not include the size of the chunk header.
* Some WAVE files may have a byte alignment such that for example you have a 24-bit depth but each sample has a fourth padding byte such that samples come every 4 bytes. I should be wary of this.
* Should nChannels be less than the number of bits set in dwChannelMask, then the extra (most significant) bits in dwChannelMask are ignored (in WAVEFORMATEXTENSIBLE).
* It looks like format chunks used to be different. I should make sure I’m reading the up-to-date format and maybe even supporting the old one too depending on how much it’s still in use (it isn’t).
* When saving a file, I should make sure not to mess up the way the wave data is split into chunks or the cue chunks pointing to these, because there are chunks we ignore such as the associated data chunk which rely on these chunks.
* I think as long as we keep the fccChunk and dwChunkStart fields of cue points in a file updated, there are no concerns with messing up a file by changing the positioning of things. This is only true if we preserve the number of channels.
* Due to confusing ass specifications and the apparent scarcity of these chunks, I will ignore the playlist chunk.
* Converting to mono and then back is problematic because the output isn’t what it should be. However, converting to mono and staying in mono is a pain in the ass because a lot of data in the wave file may rely on it being multichannel, such as playlist chunks and cue chunks and whatnot. Their documentation is also pretty bad so it’s hard to understand exactly how they depend on it. Currently, giving proper support for editing channels separately seems like the easy way out.
* Possible optimization if re-painting the main window becomes an issue: make the new file options menu have the window class style CS\_SAVEBITS.
* The DFT of audio with N samples and a sample rate of F returns a series that is also of length N, and the k’th index has the coefficient of the frequency: (k / N) \* F. NOTE: keep in mind the zero-padding added with regards to N.
* Since fourier is complex, the graph will display the magnitude of complex numbers. Multiplying a complex number by x multiplies its magnitude by x.
* Sounds like DCT output can be a bit off sometimes, and there’s also less resources for it and everyone seems to associate it almost exclusively with compression. We’re doing DFT, not DCT.
* The DFT of a real-valued sequence (what we have) is symmetric with conjugate about the Nyquist frequency. That is if the DFT sequence is X1,…,Xn then Xk = conjugate of X(N-k). This may only start from after the first index, the one that’s just the sum of all samples. This means that any changes I make, I should mirror them for the other end of the graph.
* A fourier coefficient a + bi can be thought of as a magnitude |a + bi|, and a phase arctan(b / a).
* It is absolutely important to modify both the real and complex part of a coefficient. DCT assumes the function is even and thus there is no complex part. If I want to use DFT, I have to also ask for a phase input or assume it’s 0. If I want to use DCT, it’s always 0 so there’s nothing to worry about.
* We’re gonna use DFT. For additive mode, we’ll add a magnitude while preserving the phase at each sample.
* Silence chunks will be ignored by this program.
* For drawing graphs, it’s possible the y axis should be in decibels, meaning logarithmic scaling. Maybe only for the fourier graph.
* We’re gonna apply triangular dithering whenever you save a file, because it’s needed when you convert from float->int after having done some processing. We’re not gonna apply noise shaping, because it’s recommended to only do it once on the last edit of a file, which means I’d want to have an option to turn it off, and I don’t want to bother with the GUI for that.