What is the format of the data in the AudioBuffer memory buffer, and how do I convert it to something else?

devblogs.microsoft.com/oldnewthing/20201104-00

November 4, 2020

Raymond Chen

The Windows Runtime AudioBuffer class represents a buffer of audio data. What is the format of this data?

The memory buffer you obtain from the AudioBuffer object takes the form of an array of audio samples. Each audio sample is a collection of IEEE single-precision floating point numbers which represent a linear range of waveform amplitude from −1.0 to +1.0.

In C#, this floating point format is known as `System.Single`. In C++, it is typically represented by `float`.

Each sample contains one value for each channel, and the channels come in a specified order, described in the documentation for the `WAVEFORMATEXTENSIBLE` structure.

For example, suppose you have a dual-channel audio buffer, say stereo left/right. The table in the `WAVEFORMATEXTENSIBLE` documentation says that the channels come in the order *left* then *right*. Therefore the values come in this order:

Sample index	Channel	Value index
0	Left	0
	Right	1
1	Left	2
	Right	3
2	Left	4
	Right	5
⋮	⋮	⋮

Some people call this interleaved format, but whether it's interleaved depends on what color glasses you're wearing.

It's interleaved if you look at it from the point of view of a channel, since the data from one channel is interleaved with data from the other channels.

But it's perfectly linear format if you look at it from the point of view of the samples, since all the data for one sample is packed together.

Okay, next question: How do you convert this to other formats?

Well, that depends on what other format you're converting it to. If you're converting to a linear format,[1] then you can perform a simple linear conversion. We know that the result is going to be $f(x) = ax + b$ for some values of a and b. We just need to figure out what those values are.

Substitute $x = -1.0$:	$v_{min} = a \times (-1) + b$
Substitute $x = +1.0$:	$v_{max} = a \times (+1) + b$

Solving the system of simultaneous equations gives

$$a = (v_{max} - v_{min})/2$$

$$b = (v_{max} + v_{min})/2$$

There is an extra wrinkle to this formula: If the destination is an integer range, then the negative values will have one extra value of range compared to the positive values. For example, a 16-bit signed value will range from −32768 to + 32767. If we plug zero into the function, we get just b, which is the average between the high and low values, and which will be numerically −½ rather than zero.

I don't know how audio people usually solve this problem. One option I've seen is to throw out the most negative value, so the effective range for a 16-bit signed value is −32767 to +32767. In that case, the formula simplifies to merely multiplying by v_{max}.

Maybe some audio people can tell me what they do here.

[1] There are nonlinear formats for audio data. For example, μ-law is a companding algorithm which uses a nonlinear formula in order to express a wider dynamic range in a small space, at a cost of resolution at the extremes.

Raymond Chen

Follow