

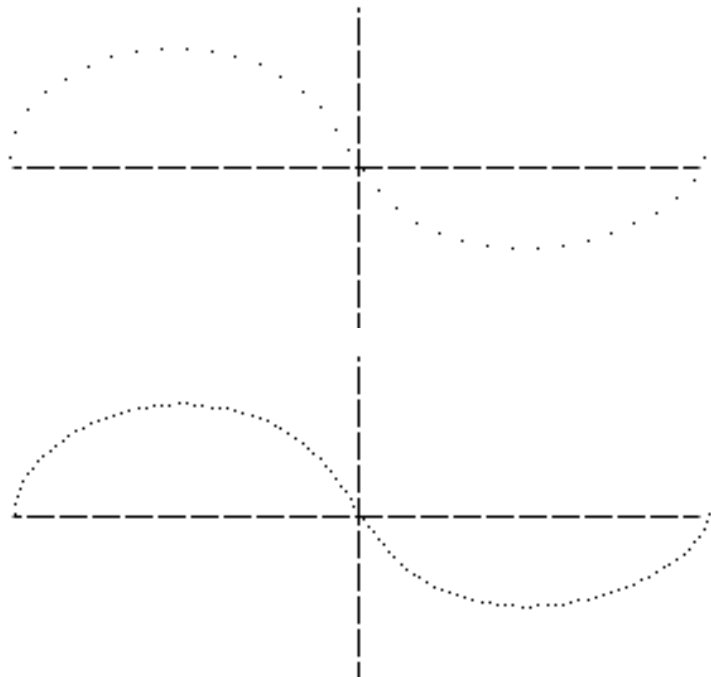
RealTime Sound FX Mixing

Intro

I don't care how cool the graphics are or how radical the music score is, without realtime sound mixing it means CRAP! I rely heavily on realtime sound FX mixing. I first started using the services of a driver that came with my Sound Blaster 16 board. At first it seemed pretty cool. Difficult to master at first, but pretty neat none the less. I was fascinated with the ease of use after I had gotten the interface to the driver done. Everything worked flawlessly, but there was still one thing bothering me. Even if I could invite all my friends (ok friend) over to see my awesome new discovery, I was still limited by the functionality of the driver. I had access to only the things that it gave me. No realtime sound mixing, no directional dampening, no echo abilities, no special effects ,no,no,no. The list kept on getting bigger. It was then that I decided to either learn how to program the SB directly or kill myself trying. Well luckily the first happened before the last one. I now behold to you in all its glory realtime sound mixing and directional dampening. All these ideologies have come only from myself. I haven't bothered to buy any books on these subject simply because these methods suit my needs perfectly. I anticipate that later on I will get an itch to attain programming spiritual creaminess and you just might catch me sneaking a peak at a book or two on this subject.

Sample Representation

Sound quality is based on the sampling rate and the how accurately the sound was digitally converted. Using an 8 bit sample will produce more distorted results than using a 16 bit sample. Lets say we could take a wave just like the sin wave and have it represent a sound. It is going to fluctuate from positive and negative one. Lets say at any given x,y will be some number. If we are using a calculator we could solve this problem with ANY x and it will give us the correct y down to the precision of the calculator, which now a days seems infinitely precise.

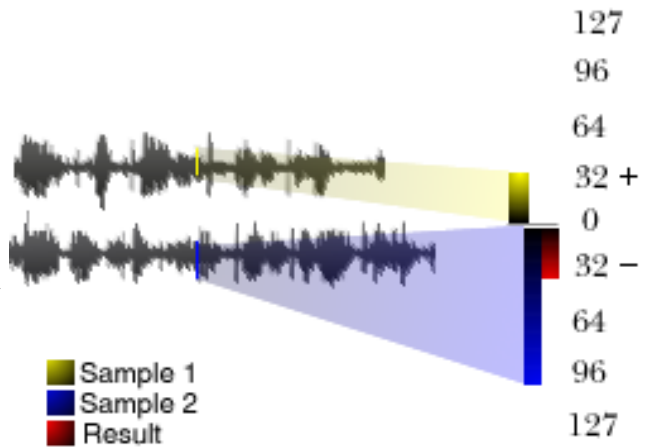


Now what if we had to split up ALL possible y 's into only 255 steps. Whenever we inputted a number we most certainly would get a very gross approximation. Now what if you divided all possible y 's into 65,000 steps. This is a lot more precise than 255 steps and this will give us some very desirable results. For tests samples I like to use 8-bit 11025k hz sound files. For REALLY good sound I would switch to 16 bit 22k or 44k if you can afford the processor time.

RealTime Sound Mixing Method 1

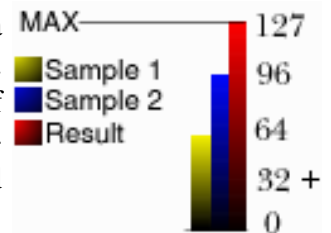
Now this whole problem boils down to one simple problem. How does one mix 1 sound with another. Well the answer is: take one sample from each sound, add it together and divide by 2. Wow that amazing! It is just that simple. If we are going to mix an unknown number of sounds then we have to add them all together and divide by however many we sounds we decided to collect. Five sounds divide by five. One kajillion sounds divided by one kajillion. Although this method is easy, it comes with some downfalls.

As you can see in the example we have just two samples we want to mix together. At the byte we are at, sample one has a value of 32 and sample 2 has a value of -96. The final result that will be placed into our outgoing buffer is -32 => $(-96+32)/2$. The problem is that we induce a silencing effect by always filling our outgoing buffer with samples that have been divided. The more samples we mix, not only does the math load increase, but the result will be consistently quieter than what they would sound like in real life. We could create some routines to correct this problem, like a lookup table for increasing the volume depending on how many sounds are mixed at a time. This would help some, but it really wouldn't be addressing the real problem at hand, the division, and there just aren't any good methods to correct this problem. We need something better! Luckily we have an alternative!



RealTime Sound Mixing Method 2

An alternative to the previous example would be to simply add the two samples together and put a cap on the signal to make sure that it is within our data type, unsigned char or char for 8 bit and short or unsigned short for 16 bit. Instead of adding all the sounds together and dividing by the total number of sounds we collected, just add them all together and clip! Just like in our example, sample 1 has a value of 64 and sample 2 has a value of 96. Added together we get 160 but this exceeds our char data type so let's clip it to fit! This works very well and is very fast compared to the previous method. I previously had problems combining this method with my 3d sound routines, this problem has been fixed and works excellently!



Many thanks go to Cyberfrog who brought this method to my attention. If he hadn't leaked this idea to me, I wouldn't be able to offer it! Thanks again Cyberfrog!!

I only know of these two methods for sound mixing for now. This tutorial was meant to get you to start thinking about how samples are mixed together before being sent off to the speaker. If you can understand that, then you can start thinking about different variations of these routines, creating some cool effects! These will be covered in the other tutorials. If you have any comments, questions, rude remarks, need to pass gas whatever...give me some feedback!!

Contact Information

I just wanted to mention that everything here is copyrighted, feel free to distribute this document to anyone you want, just don't modify it! You can get a hold of me through my website or direct email. Please feel free to email me about anything. I can't guarantee that I'll be of ANY help, but I'll sure give it a try :-)

Email : deltener@mindtremors.com

Webpage : <http://www.inversereality.org>

Cyberfrog's Webpage:

<http://ourworld.compuserve.com/homepages/RoccoLoscalzo/>

Created by
Justin Deltener