3D-SOUND PROGRAMMING WITH MICROSOFT DIRECTX

Surround sound in practice

HÄMEEN AMMATTIKORKEAKOULU

Thesis of the University of Applied Sciences

Degree programme in Business Information Technology

Visamäki Fall 2013

Ville Niskanen

Mr. Ville Niskanen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

ABSTRACT

Unit	Visamäki
Name of degree programme	Business Information Technology
Option	System development

Author Ville Niskanen **Year** 2013-11-28

Subject of Bachelor's thesis 3D-sound programming with Microsoft DirectX - Surround sound in practice.

ABSTRACT

The thesis work done consists of computer application that plays various 3D-sound effects. The application has been realized with the main application and the actual sound effects have been implemented in the form of extraneous plug-in files. Rough implementation of the produced program structure has been described in this thesis document essay part. As the subject of the thesis has been an individual research project there have been no work life relations.

The objectives of the thesis have been quite simple targeting, as can be presumed based on foregoing, in programming application that plays various 3D-sound effects, and furthermore creating mathematical algorithms to implement 3D-sound effects.

The knowledge basis, based on the fact that Microsoft DirectX-programming environment, as well as likely all Microsoft-programming components, is well-documented, and on the actual Microsoft DirectX-programming documentation found from the www, and furthermore on the material found from the web covering the subject of digital signal processing. The actual programming skills basis has been attained by studying several programming courses including UOAS-course DirectX-programming and by individual studying of programming.

Actual programming work of the Thesis done have been done in Windows-platform using Microsoft Visual Studio-programming environment, DirectX SDK programming components and programming with Visual Basic.NET-programming language.

As a conclusion, completed research and programming project has shown that Microsoft DirectX and DirectX DirectSound programming suits excellently fine for 3D-sound and 3D-sound effects programming.

Keywords 3D-Sound, Sound Effects, Microsoft DirectX DirectSound, Digital Sound Processing, Mathematical algorithms

Pages 28 p.

TIIVISTELMÄ

HAMK
UNIVERSITY OF APPLIED SCIENCES

TOIMIPISTE	Visamäki
Koulutusohjelman nimi	Tietojenkäsittely
Suuntautumisvaihtoehto	Systeemityö

Tekijä Ville Niskanen **Vuosi** 2013

Työn nimi 3D-sound programming with Microsoft DirectX - Surround sound in practice

TIIVISTELMÄ

Tehty opinnäytetyö käsittää tietokoneohjelman, joka soittaa 3D-ääniefektejä. Ohjelma on toteutettu pääohjelman ja erillisten 3D-ääniefekti plug-in -tiedostojen muodossa. Kuvaus ohjelmasta löytyy tästä opinnäytetyön tutkielmaosuudesta. Opinnäytetyö on itsenäinen tutkimus- ja ohjelmointiprojekti, eikä sillä ole yhteyttä työelämään.

Kuten edellä olevasta voi käydä ilmi, opinnäytetyön tavoitteena on ollut ohjelmoida 3D-ääniefektejä soittava ohjelma sekä luoda 3D-ääniefektejä tuottavia matemaattisia alkoritmejä.

Tietoperustan opinnäytetyötyölle on luonut tosiasia, että Microsoft DirectX–ohjelmointiympäristö, kuten yleensäkin kaikki Microsoftin ohjelmointikomponentit, on hyvin dokumentoitu sekä www-ympäristöstä löytyvä digitaalista signaalin käsittelyä käsittelevä materiaali. Varsinainen ohjelmointiosaaminen perustuu ammattikouluopintoihin kuuluviin ohjelmointikursseihin, mukaan lukien DirectX-ohjelmointi kurssi, sekä itsenäiseen ohjelmointiopiskeluun.

Varsinainen opinnäytetyöhön lukeutuva ohjelmointityö on hoidettu Windows-alustalla käyttäen Microsoft Visual Studio -ohjelmointiympäristöä, DirectX SDK -ohjelmointikomponentteja ja Visual Basic.NET -ohjelmointikieltä.

Tehdyn tutkimus- ja ohjelmointityön perusteella voi todeta, että Microsoft DirectX sekä DiredX DirectSound -ohjelmointi sopii erinomaisesti 3D-äänen ja 3D-ääniefektien ohjelmointiin.

Avainsanat 3D-ääni, ääniefektit, Microsoft DirectX DirectSound, DSP, matemaattiset algoritmit

Sivut 28 s.

ACRONYMS AND VITAL CONCEPTS

3D	Tree Dimensional, surrounding
A/D	Analog to Digital (conversion)
API	Application Programming Interface
D/A	Digital to Analog (conversion)
DSP	Digital Signal Processing, refers to process where electrical signal is first converted into digital signal (A/D conversion), then digital (signal) data is modified programmatically (by computer) and then digital (signal) data is converted back into analog signal (D/A conversion)
LFO	Low Frequency Oscillator
SDK	Software Development Kit
WPF	Windows Presentation Framework, set of APIs and presentation methodologies for user interfaces
.NET	large set of Microsoft Application Programming Interfaces with automatic / managed memory management for users of programming languages like Visual Basic and C#
.NET-programming	computer programming using .NET-APIs
3D-sound	three-dimensional sound, surround sound
Analog sound effect	sound effect created by electrical sound signal modification
Digital sound effect	sound effect created by digital sound data modification
DirectSound	programming API for high-performance audio
Managed code	program execution environment where (RAM-) memory management is automated
Microsoft DirectX	programming APIs for high-performance graphics and audio
Plug-in programming	application structure where application functionality can be extended with additional file-components
Sampling rate	how many digital values representing amplitude of sound at certain moment are taken of the sound per second
Sound card	computer interface for electrical in / out sound signal where analog signal is converted into digital (A / D) and vice versa (D / A)
Surround sound	usually vertical 3D-sound (or two-dimensional sound)
Visual Basic	programming language

CONTENTS

1 INTRODUCTION

While the actual thesis work consists of programming (3D-) sound and (3D-) sound effects, this Thesis report tries to answer the questions:
What are DirectX-sound programming objects and how to use those? What are digital sound processing-mathematical algorithms and how to implement those in computer programming? What kind of digital sound processing characteristics are there in DirectSound-programming components? How to implement 3D-sound-mathematical algorithms and what kind of algorithms are those? 3D-sound, could that be art?

These questions are considered in the chapters starting with basic representation of DirectX, DirectSound and DirectSound-programming, continuing in relations of sound effects, digital signal processing and mathematics, which sets a good basis for understanding how to implement digital (sound) data processing in DirectSound-programming at the digital sound data sample-level. Furthermore this is a good premise for implementing sound processing and sound effects in 3D-environment that is gone through in chapter 4.3 - Programming 3D-sound effects with DirectSound.

Chapter 5 - Results - is rough representation of the produced plug-in program structure and overview of produced 3D-sound effects and few thoughts about them.

Finally, there is speculation concerning the value of 3D-sound as an art, entertainment, etc.

2 MICROSOFT DIRECTX, DIRECTSOUND AND 3D-SOUND

2.1 Microsoft DirectX

Microsoft DirectX was introduced in 1995 (Wikipedia 2013a). It is a collection of application programming interfaces (API) targeting for high-performance graphics and sound programming in windows platforms. High-performance is attained by as close as possible programming code communication with hardware. Main components of DirectX consist of components for handling graphics, sound, input devices and network traffic. Graphics and sound components contain APIs for 3D features. (Microsoft 2013a)

DirectX can be programmed with Microsoft C++- and .NET-Framework so called managed-programming languages like Visual Basic.NET and C#. I have no such a specific knowledge that I could say differences in programming features between C++ and managed languages DirectX-programming; at least DirectX for managed languages is fully featured DirectX programming environment.

Microsoft .NET-Framework managed languages (managed referencing automated memory management) were introduced in 2002 (Wikipedia 2013b). At the same time Microsoft released DirectX-programming components for .NET-managed languages (Wikipedia 2013a). Around of the zeroes to tennies decade turn there were some kind of pause in DirectX programming environment development, but as now in Windows 8 platform DirectX has made nice comeback (Microsoft 2013b). Unfortunately, there has been no possibility to find out if .dll-files used with managed DirectX-programming on Windows 7 platform run on Windows 8 platform, or if managed DirectX-programming can be done on Windows 8 platform. Anyhow, if you want to develop Windows Store Apps, that most likely run both on windows 8 and windows RT-platform, using DirectX, you have to use C++-programming language (Microsoft 2013c; Microsoft 2013d).

2.2 Microsoft DirectX DirectSound and 3D-sound

Microsoft DirectX DirectSound has been the most efficient and usable environment for sound programming in Windows platform since it was released. It consists of components for sound device, sound data and wave-formats, 3D-positioning of sound with multichannel sound cards, some common pre-build sound modification effects (even pre-build 3D-reverb effect) with adjustable parameters, and sound capturing. DirectSound sound-buffers are implemented that way that it is very easy to modify sound data at byte-, or let´s say, sample-level.

3 DIGITAL SIGNAL PROSESSING, SOUND EFFECTS AND MATHEMATICS

3.1 Digital Signal Processing (DSP)

Signal processing is about modifying a signal (data) into another form of original signal. Analog signal is electrical signal, and analog signal processing (ASP) consist of modifying this signal some kind mechanical-static way e.g. with electronic component like resistors and transistors. Whereas digital signal processing (DSP) consist of modifying (signal) data programmatically by (computer) CPU (central processing unit). Digital signal data in DSP can originate from analog to digital (A / D) signal conversion process or it can be created programmatically.

3.2 Sound effects

Usually quoting term sound effect refers to sound that is modified with somewhat of signal processing. In that way the sound effect can be created by analog or digital signal processing. Anyhow, sound effect can be also created by programmatically using some kind of mathematical algorithms.

An example of sound effect might be flanger-effect which originates to effect created with two tape machines playing same sound at same time with little varying speeds, where the varying playing speed is carried out by lightly touching and releasing the flanger of the other reel. So the sound output of both reels is in out of synch by varying time spans. (Smith 2010a.)

3.3 Sound data modification with mathematics

A simple digital signal processing mathematical algorithm representing so called feedforward comb filter is shown in picture 1.

$$\mathcal{Y}(n) = \mathcal{X}(n) + \mathcal{X}(n - M)$$

Picture 1. Feedforward Comb Filter mathematical representation. *(Smith 2010b.)*

This algorithm combines unmodified input signal (*x(n)*) and delayed input (*x(n - M)*) signal into the output signal (*y(n)*). We can think that variable *x* represents sound data and algorithm modifies that sound data.

DSP feedforward comb filter is base in creating common electronic music flanger-effect with few additional parameters. At foregoing algorithm symbol *n* represents time that is, as this algorithm applies to digital sound data, index (or position) of the data sample in the sound data array. Symbol *M* represents time the delayed signal is delayed, and in digital sound data, the length of delay in samples. If we write the algorithm with few more parameters, we get an algorithm

$$\mathcal{Y}(n) = \mathcal{X}(n) + A\mathcal{X}[n - (M + S)]$$

Picture 2. Mathematic algorithm to create flanger-effect.

where *S* represents time-varying (sample) shift of the delay and *A* represents amplitude (or volume) of the delayed signal (sample). To get time-variance factor *S* we have to use another DSP algorithm (which, by the way, are often called filters), the Low-Frequency Oscillator (LFO) algorithm that can generate time-variant (or sample-variant) output with known bounds e.g. -50 to 50. (Smith 2010a). So in this way we can modify digital sampled sound data with mathematics.

Furthermore, common digital signal processing and its algorithms are not the only way to create sound effects with mathematics. All mathematical methods and functions that can produce and modify arrays of numbers (real number sets in mathematics) can be used to create digital sound and sound effects. Quite a simple sample of this kind of digital sound producing is an algorithm that creates simple sine wave like in picture 3.

$$\mathcal{X}(n) = \sin(2\pi f_o n)$$

Picture 3. Mathematic algorithm to create simple sine wave. (Lyons 2010).

3.4 Mathematical time-variant algorithms equality in computer programming code

A scientific mathematic notation can be quite exotic for a computer programmer who has no knowing of this subject. Time-variant algorithms are expressed in computer languages with so called loops. For a long it has been common notation for looping-code in programming to use *for-next* loop. If we get algorithm in picture 3 and write equivalent code in Visual Basic.NET-programming language we get code like in picture 4,

```
SamplesPerWave = SamplesPerSecond \ Frequency
For n = 1 To SamplesPerWave
x(n) = System.Math.Sin(2 * System.Math.PI * (n / SamplesPer-
Wave))
Next
```

Picture 4. Visual Basic.NET-program code for creating simple sine wave sound data.

where *SamplesPerSecond* is same as sound sampling rate, while *SamplesPerWave* is samples (*x(n)*) needed to get full wave of sine curve and is calculated by dividing sampling rate with desired frequency.

4 DIRECTSOUND BASIC AND 3D-SOUND EFFECTS PROGRAMMING

4.1 Programming environment used in this work

DirectSound testing and programming environment used in this work consisted of

Microsoft Windows 7 Enterprise and Professional 64bit operating systems
Microsoft Visual Studio 2010 Ultimate programming environment
DirectX SDK (June 2010)
WPF- and Visual Basic.NET –programming tools
Computers with:
4 to 8 GB of RAM-memory
Dual core 2.8 GHz processors
5.1-channel DirectSound compatible sound card set on 4-channel mode
2-channel sound cards
Stereo headphones and 4.0 speaker configuration

4.2 Programming basic sound effects with DirectSound

Creating simple sound in DirectSound programming consist of creating objects for

```
Microsoft.DirectX.DirectSound.Device
DirectSound.WaveFormat
DirectSound.BufferDescription
DirectSound.SecondaryBuffer
SoundDataBytes
```

where *Device* refers to actual computer sound card. *Waveformat* consist characteristics for sound data playback like *BitsPerSample* and *SamplesPerSecond*. *BufferDescription* defines characteristics of sound data buffer like can buffer´s volume be controlled, is buffer played while program window is minimized and can buffer´s position be adjusted in 3D-space. *SecondaryBuffer* represents actual sound data that can be played via sound card and speakers. (Microsoft 2013e) Still for better performance it is reasonable to use *SoundDataBytes* data-array for sound data modifications and then write that data at once into *SecondaryBuffer*.

4.2.1 Programming base flanger effect with DirectSound pre-build flanger-effect

Programming DirectSound flanger-effect consist of

1. Creating object

```
Microsoft.DirectX.DirectSound.EffectDescription
```

2. Assigning effect-guid for *EffectDescription*

```
EffectDescription.GuidEffectClass =
Microsoft.DirectX.DirectSound.DSoundHelper.StandardFlangerGuid
```

3. Setting *EffectDescription* for *SecondaryBuffer*

```
SecondaryBuffer.SetEffects({EffectDescription})
```

4. Getting *FlangerEffect* object

```
DirectSound_FlangerEffect = SecondaryBuffer.GetEffects(0)
```

5. Getting *EffectsFlanger* object

```
DirectSound_EffectsFlanger =
DirectSound_FlangerEffect.AllParameters
```

6. Setting parameters / properties for *EffectsFlanger* object

```
With MDD_EffectsFlanger
     .Waveform      = [value]
     .Frequency     = [value]
     .Phase         = [value]
     .Delay         = [value]
     .Depth         = [value]
     .WetDryMix     = [value]
     .FeedBack      = [value]
End With
```

7. Setting *EffectsFlanger* objects parameters to *FlangerEffect* object

```
DirectSound_FlangerEffect.AllParameters =
DirectSound_EffectsFlanger
```

at this point, as we play *SecondaryBuffer* that to *EffectDescription* was set above at stage 3, we hear original *SecondaryBuffer* sound data modified with flanger-effect.

4.2.2 Programming base flanger effect with DirectSound and DSP flanger-effect algorithm

In chapter 3.3 is mathematical algorithm for flanger-effect creation

$$\mathcal{Y}(n) = \mathcal{X}(n) + A\mathcal{X}[n - (M + S)]$$

where, *x* is input value at time *n*, *y* is output value at time *n*, *M* is delay time, *S* is low frequency oscillator (LFO) that here represents varying time value, *A* is amplitude of flanger effect.

For creating programming code that represents this mathematical algorithm we use

1. For-next loop that creates simple sine wave like in chapter 3.4., but this time looping the whole sound buffer rather than just one wave of sine curve.

```
Pi = System.Math.PI
' We can also use Zulu-constant in place of Pi-constant.
' Zulu-constant represents Pi-value with the accuracy of
' six decimals of modern calculated Pi (Wikipedia 2013c).
Zulu = 355 / 113
SamplesPerWave = SamplesPerSecond / SoundFrequency
For n = 1 To SoundDataBufferSampleCount
x(n) = System.Math.Sin(2 * Zulu * (n / SamplesPerWave))
```

2. Yes, there is no *Next* in the above code because we put the code equivalent to the flanger-effect mathematical algorithm inside this loop. First we check if sample (*x(n)*) is far enough, measured as sound data samples, from the start of the sound data buffer so that we can get former sample behind at the delay length in samples. We also have to add width or depth of the low frequency oscillator varying time value to the delay length needed to get former sample in data buffer (that here is same as array of numeric values).

```
If n > DelayInSamples + DepthInSamples Then
```

3. If this is true we calculate the varying delay offset value (that was represented as *S* in the mathematical representation of the flanger-effect algorithm) at the relation to current sample index (*n*) using low frequency oscillator which frequency is defined with *FlangerSpeed* parameter, that is, how many cycles our flanger effect takes in a second.

```
SamplesPerFlangerWave = SamplesPerSecond / FlangerSpeed
LFO_ValueInSamples = DepthInSamples *
System.Math.Sin(2 * Pi * (n / SamplesPerFlangerWave))
```

4. After that we can get the flangered sound data sample by adding the LFO delay offset value (*LFO_ValueInSamples*) to the static; the static here in for-next loop, flanger delay depth value (*DelayInSamples*) that gives us the sample index of the former sample in the sound data buffer in relation to the current sample index (*n*) and then multiply this former sample value with the desired amplitude and add the result to the current sample at index *n*.

```
x(n) = x(n) + Amplitude *
       x(n - (DelayInSamples + LFO_ValueInSamples))
```

This is nearly the same as the mathematical representation of the flanger-effect in picture 2.

5. Then we can close our loop and write the flangered sound data into *SecondaryBuffer*-object and then play it.

```
End If
Next
SecondaryBuffer.Write(0, x, 0)
```

4.3 Programming 3D-sound effects with DirectSound

By using four-speaker configuration we can, in fact, adjust the sound only in two-dimensional space without the up and down scope. However e.g. four-speaker configuration can be considered as an environment for 3D-sound with left to right and front to back factors just without up and down dimension.

By using eight-speaker configuration, where there would be one high-speaker above each of the four-speakers, that now would become the low-speakers, we would get a speaker configuration that would offer property for adjusting the sound position in up-down dimension in addition to left-right and front-back dimensions. In this eight speaker configuration the speakers / channels would become FLL (front-left-low), FRL (front-right-low), RLL (rear-left-low), RRL (rear-right-low), FLH (front-left-high), FRH (front-right-high), RLH (rear-left-high) and RRH (rear-right-high).

In DirectSound programming, however, if the speaker configuration used is four speakers, the sound position adjusting is still done by *Verctor3*, that would be used also for other multi-speaker configurations, like if there would be eight speakers in use.

In continuation for chapter 4.2., managing 3D-sound with DirectSound is quite straightforward in addition to creating base DirectSound sound. We create objects for

```
Microsoft.DirectX.DirectSound.Buffer3D
Microsoft.DirectX.Vector3
```

then we wrap existing *SecondaryBuffer* into *Buffer3D*, and adjust *Buffer3D*´s position with *Vector3* and its *XYZ*-properties. (Microsoft 2013f.)

4.3.1 Programming 3D-flanger effect with DirectSound and DSP flanger-effect algorithms

The flanger-effect as a 3D-implementation can be programmed in DirectSound by

1. Creating few *SecondaryBuffer*-objects
2. Writing flangered sound data to these buffers like in chapter 4.2.2. so that the depth, width and speed of flanger-effect are different, in other words, so that different flanger-effects are non-synchronized, but the base frequency of sound is the same so that ears comprehend the sounds as one certain sound.
3. Applying *Buffer3D*-objects to *SecondaryBuffers*
4. Adjusting position of 3D-buffers by *Vector3*
5. And then playing each *SecondaryBuffer* at the same time

4.3.2 Programming various 3D-sound effects with DirectSound and mathematical algorithms

Random 3D notes.

This effect can be produced as follows

1. Creating four *SecondaryBuffer*s
2. Creating *System.Random*-object (Microsoft 2011)
3. Creating loop and inside of loop
4. Getting random numbers for speaker and note pitch in the range of the speaker count and desired note pitch
5. Creating sine wave equivalent to desired pitch and duration of the note
6. Playing created sine wave with the speaker returned by the random number in the stage 4
7. Looping stages three to six desired times

Moving beat figures.

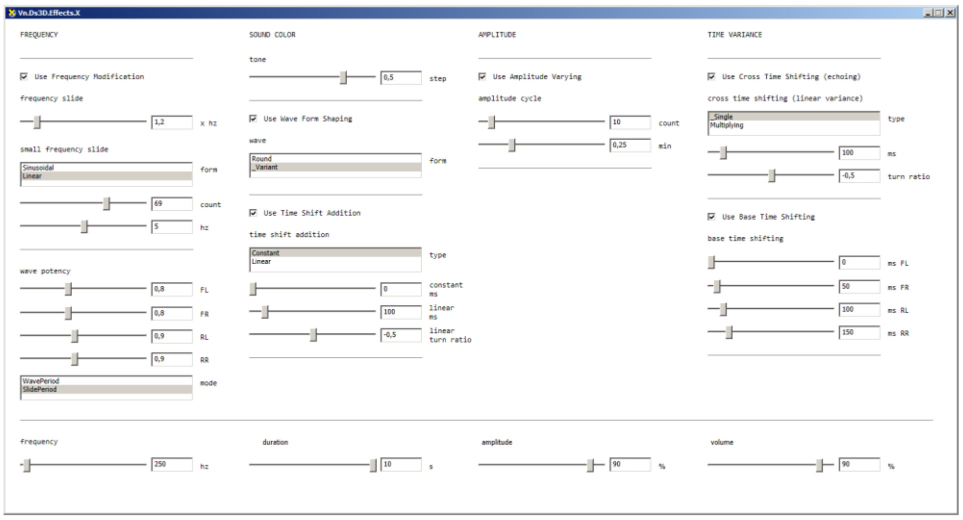
We can produce this effect like

1. Constructing four *SecondaryBuffer*s with simple single one channel (mono) drum beat sample wav-file (Microsoft 2013g)
2. Creating loop and inside of loop

3. Setting the position of each of four *SecondaryBuffer*s in a kind of geometry / symmetry in relation to each other and the former positions (using *Buffer3D*-object and *Vector3*-structure)
4. Playing each *SecondaryBuffer* nearly at the same time in sequence with a time span like 5 to 10 milliseconds between the playing start times (so that the different sounds are to be distinguished)
5. Pausing Thread execution for a while e.g. for 100 milliseconds with *System.Threading.Thread.Sleep*-method (Microsoft 2009a)
7. Looping stages three to five so many times that the moving beat figure has been achieved

4.4 Domains of the sound characteristics and programming 3D-sound

We can find different characteristics domains on the sound. The most likely found domains would be frequency, sound color, amplitude, time and space domains.
Here I made a base testing application for testing these sound characteristics with different variables. The user interface of this application can be seen on picture 5.


Vn.Ds3D.Effects.X

FREQUENCY
[✓] Use Frequency Modification
frequency slide: 1,2 x hz
small frequency slide: form Sinusoidal [✓ Linear]
69 count
5 hz
wave potency: 0,8 FL; 0,8 FR; 0,9 RL; 0,9 RR
mode WavePeriod [✓ SlidePeriod]

SOUND COLOR
tone: 0,5 step
[✓] Use Wave Form Shaping
wave: form Round [✓ _Variant]
[✓] Use Time Shift Addition
time shift addition: type [✓ Constant] Linear
0 constant ms
100 linear ms
-0,5 linear turn ratio

AMPLITUDE
[✓] Use Amplitude Varying
amplitude cycle: 10 count; 0,25 min

TIME VARIANCE
[✓] Use Cross Time Shifting (echoing)
cross time shifting (linear variance): type [✓ _Single] Multiplying
100 ms
-0,5 turn ratio
[✓] Use Base Time Shifting
base time shifting: 0 ms FL; 50 ms FR; 100 ms RL; 150 ms RR

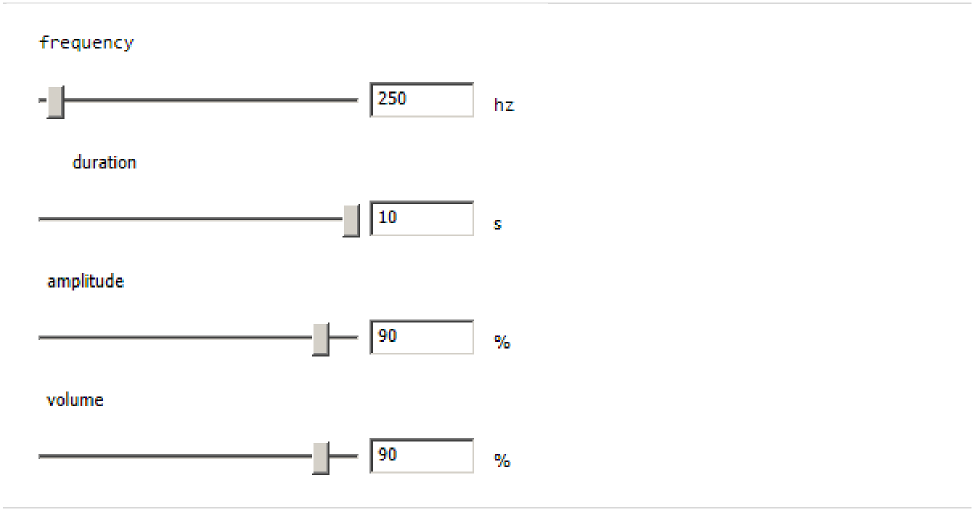
frequency: 250 hz
duration: 10 s
amplitude: 90 %
volume: 90 %

Picture 5. User Interface of Sound characteristics domains testing application.

Different sound domain properties are adjusted on a chain. Most likely order for these sound domain modifications would be in order of appearance:

Frequency
Sound color
Amplitude
Time
Space

In this base testing application (that is likely to appear in a form of plugin part for the main application of this thesis work) the base sound for testing sound characteristics domains is defined with base sound frequency and amplitude. The base settings for sound characteristics domain testing are seen on a picture 6.



frequency
250 hz

duration
10 s

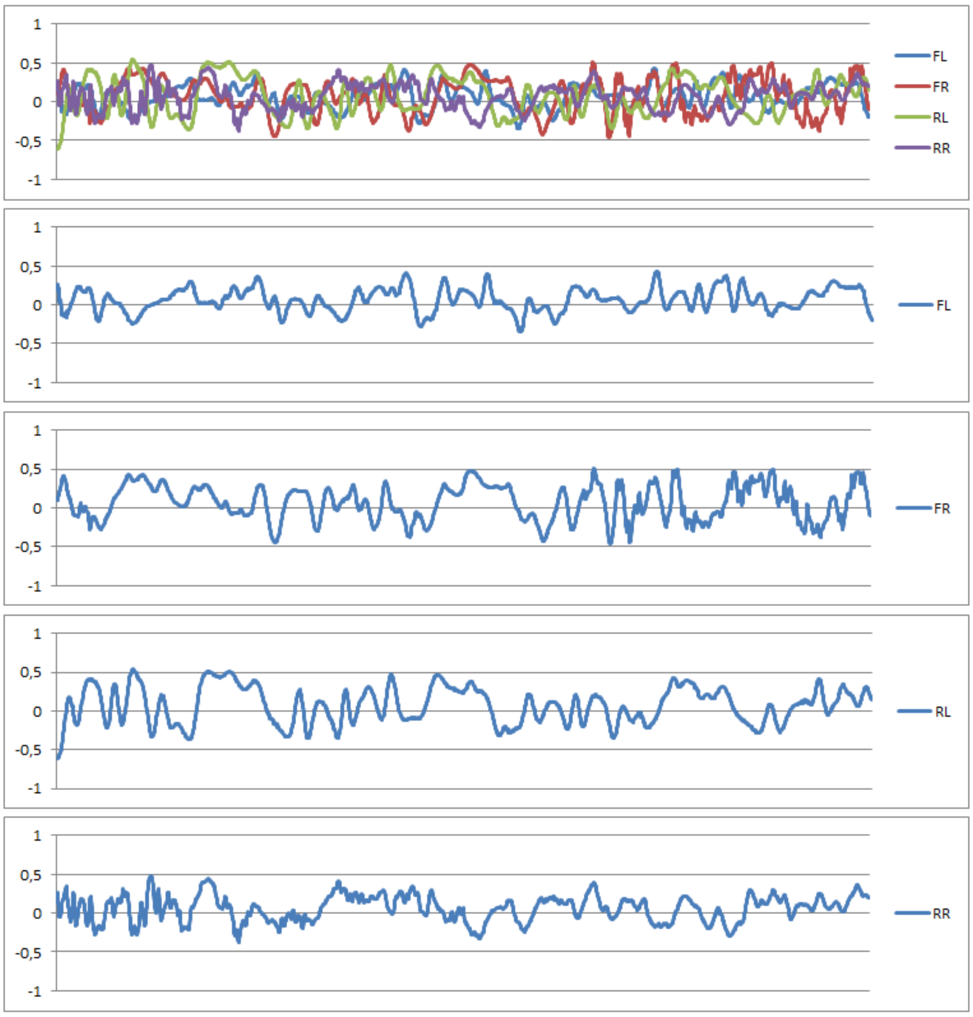
amplitude
90 %

volume
90 %

Picture 6. Base settings for the sound characteristics testing application.

The application is made for four speaker configuration, that are marked here as FL (Front Left), FR (Front right), RL (Real Left) and RR (Rear Right). The sound is varied generally so that each sound characteristic domain have also the positional dimension (so the sound generated by the application is 3D / surround sound itself).

The sound waves of each speaker of the ten seconds sound sample at the half way of the sound with the length of 40 milliseconds, after the sound is modified through the domains chain, sample by sample in each domain, is seen on the picture 7.



Picture 7. 40 milliseconds part of the sound waves modified through a chain of sound characteristics domain modifications.

With ten seconds sound sample, put through five modification domain, sample by sample, sample here meaning one sound data Int16 (Microsoft 2013h) value, makes the number of single sound data modification to appear with some ten million in count (44.100 samples per second x 10 seconds x 4 speakers x 5 chain modifications). When a single data sample modification contains multiple calculations, the number on operations made by computer comes up with some ten times more calculations in count. This is not any performance threshold for a computer with frequency of gigahertzes and even multiple processors performing this amount of sound data modifications in tenths of seconds.

4.4.1 Frequency domain

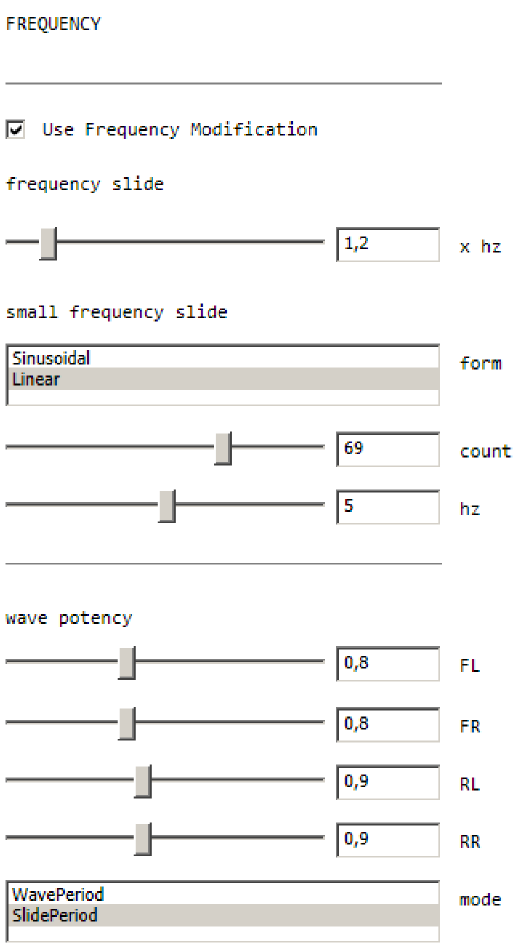
Frequency is the most effective sound characteristic domain. Varying and modifying frequency affects the most on the sound.

In my testing application I used base sinusoidal wave with base frequency for the base sound with adjustable length and amplitude. This is the base step before the frequency domain modification.

Here in frequency domain modification I changed the frequency by sliding it up and down. There were two adjustable separate slides, base slide sliding frequency up and down in linear form and the smaller slide with sinusoidal or linear form that is added to the base frequency slide. The result is sliding sound with smaller shivering.

The other modification for frequency is exponential wave frequency modification in a period of single wave or the whole sound. The Exponent can be below or above value of one. In a whole sound period value below one makes frequency dive on a low frequencies, and value above one makes frequency fly from low to high. On a wave period, exponentiation does not have influence for the frequency.

The user interface part of the settings for the frequency domain modification of the testing application is seen in the picture 8.


FREQUENCY

☑ Use Frequency Modification

frequency slide

1,2 x hz

small frequency slide

Sinusoidal
Linear
form

69 count

5 hz

wave potency

0,8 FL

0,8 FR

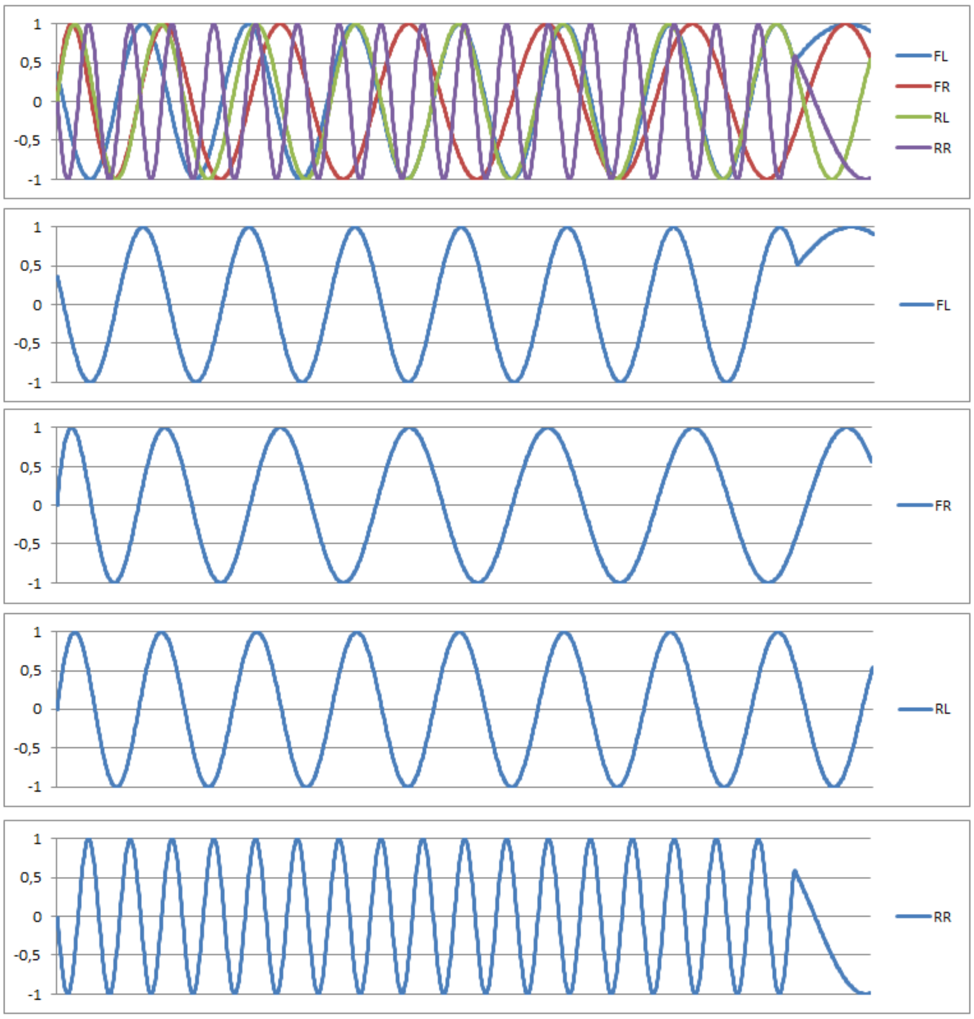
0,9 RL

0,9 RR

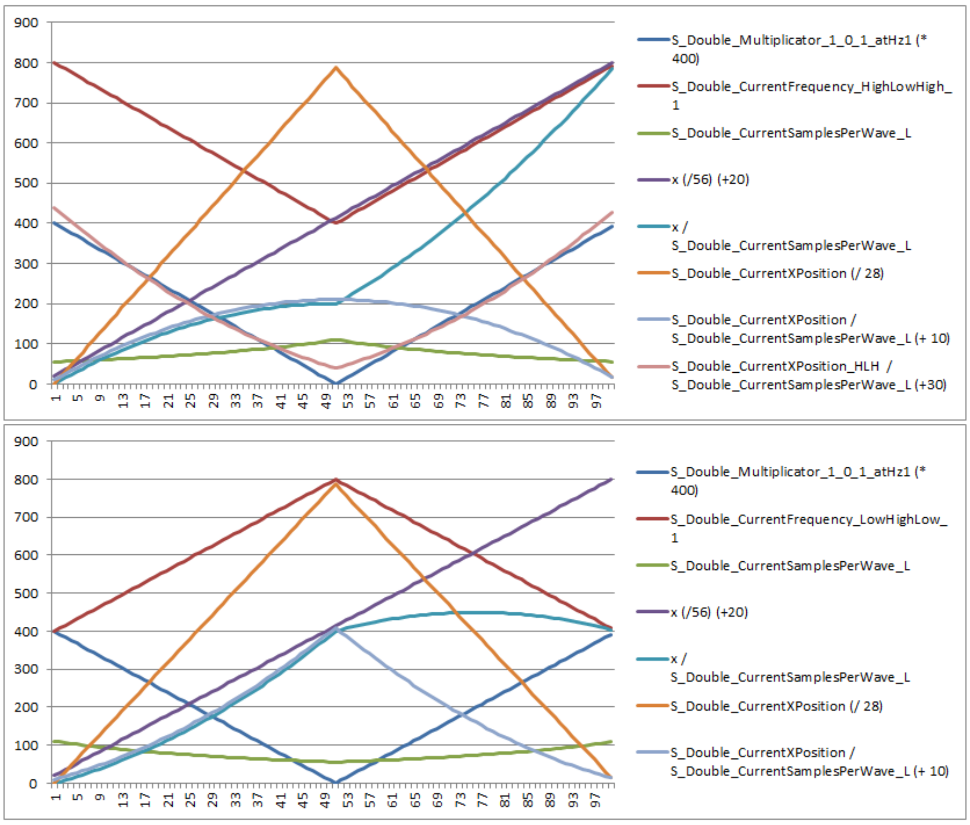
WavePeriod
SlidePeriod
mode

Picture 8. Frequency modification domain settings for the testing application.

The frequency modified sound data waves for each of the four speakers in a length of 40 milliseconds is see on the picture 9, where the base frequency is modified with the settings as on the picture 8.



Picture 9. Frequency domain modified sound data in length of 40 milliseconds.



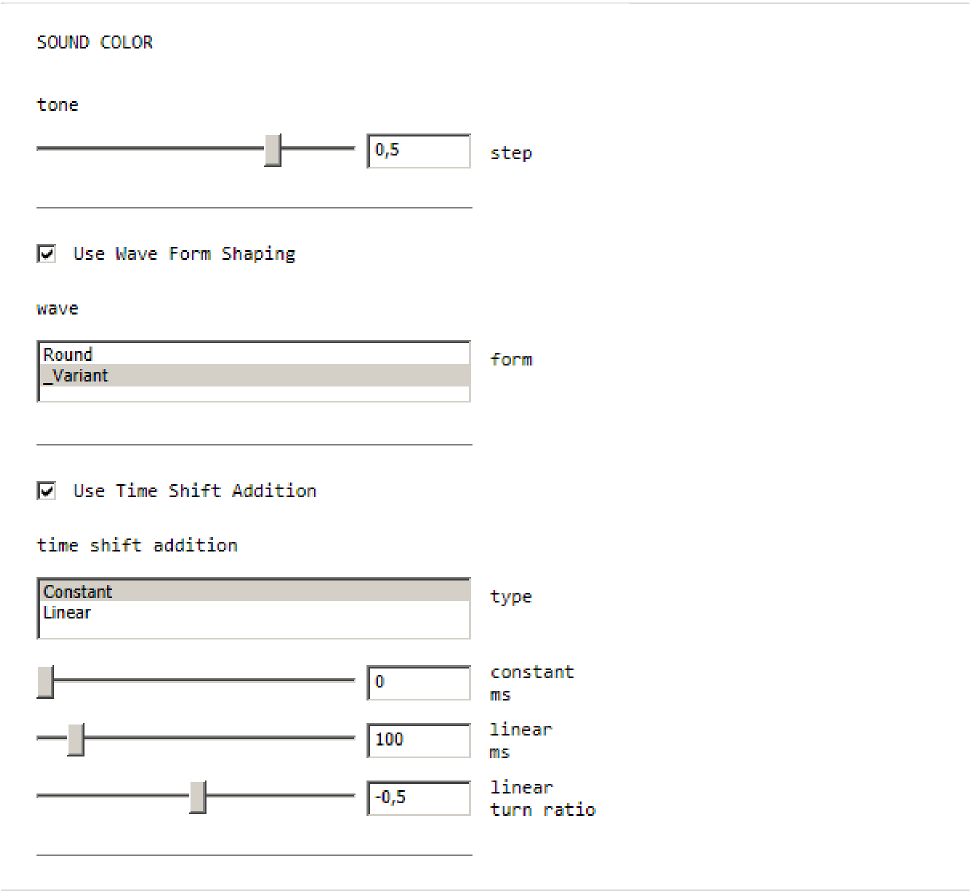
Picture 10. Researching frequency domain slide behavior.

4.4.2 Sound color domain

Sound color is the domain of the sound that makes sound to sound personal. Different musical instruments, like violin or accordion, have their own personal sound colors. Musical chords also make up sound colors.

In my testing I modified sound color with simple tone added for base frequency. I also used some wave form shaping with trigonometric functions. And also modified sound by adding time shifted values of the base sound into sound itself. Varying time shifting with the sound itself is the same sound modification method like found in a flanger effect in the former chapters. In my testing application the varying time shifting is done in a linear manner in comparison to sinusoidal varying.

Sound color domain modification settings of the testing application are seen on a picture 11. The turn ratio of the time shift addition is the ratio of receding and reverting period of the time shift swinging. Smaller value makes receding period faster and vice versa. With value zero there would be no swinging, and with positive values time shift swinging would be in the sound data samples ahead, as now the samples with negative values reside before the base sound samples. Base sound modified with sound color domain modifications by settings as on picture 11 are seen on a picture 12.


SOUND COLOR

tone

0,5 step

☑ Use Wave Form Shaping

wave

Round
_Variant

form

☑ Use Time Shift Addition

time shift addition

Constant
Linear

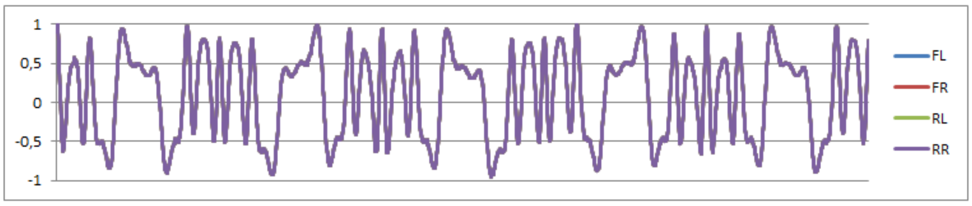
type

0 constant ms

100 linear ms

-0,5 linear turn ratio

Picture 11. Settings for sound color domain modification of the testing application.



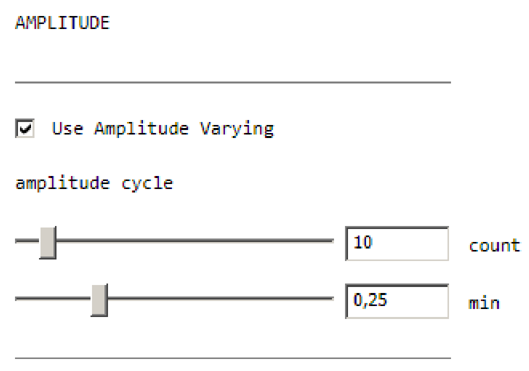
Picture 12. Sound data in length of 40 milliseconds modified with sound color domain modification.

4.4.3 Amplitude domain

Amplitude domain modification is quite straightforward, and is quite a same like adjusting the volume of the sound in a music player.

In my testing application I adjusted amplitude of each speaker separately. Amplitude varying of each speaker is done with floor amplitude and linear amplitude varying between this floor amplitude and full amplitude with specified number of the amplitude cycles in the period of the base sound. There is also time variance between each speaker that makes amplitude vary interlocked making the space property of the sound also to be present.

In the picture 13 there is a user interface part of the amplitude domain modification of the testing application. In the picture 14 is seen the sound wave of the base sound data modified by amplitude modification in a length of 0.4 seconds.



AMPLITUDE

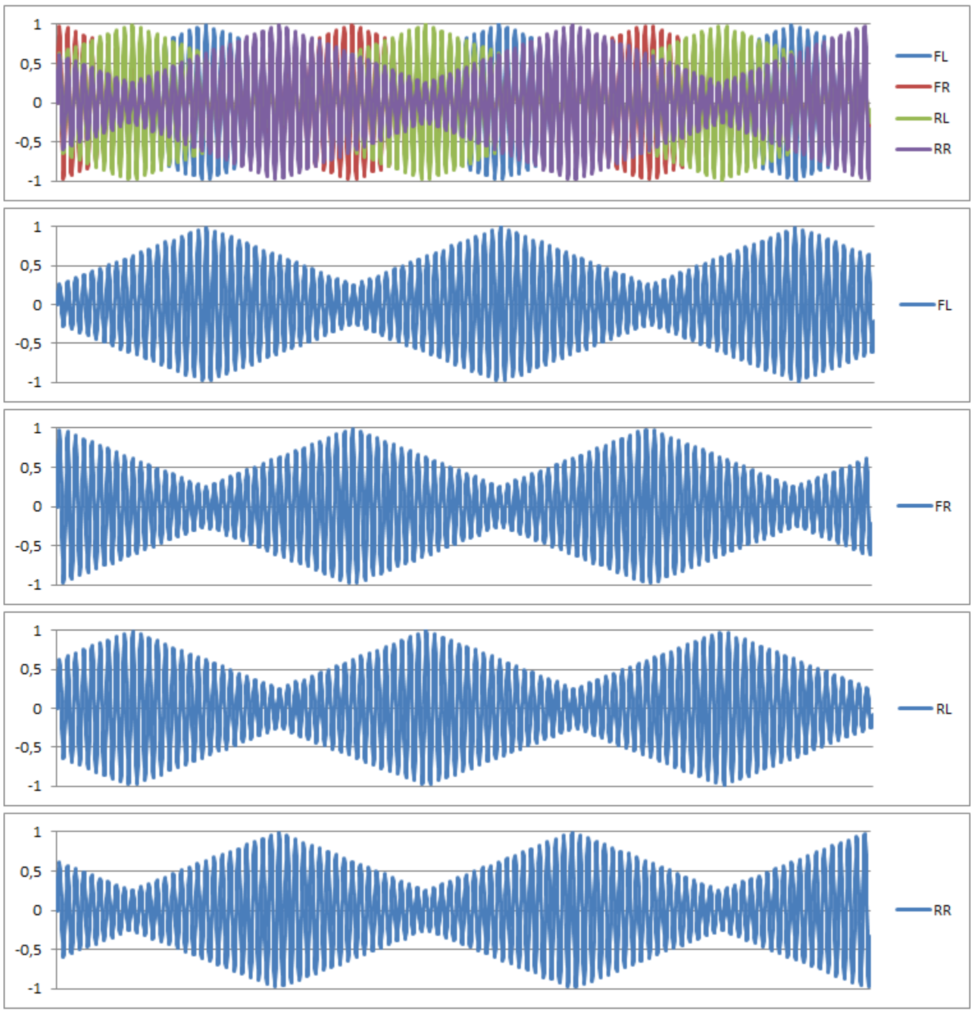
☑ Use Amplitude Varying

amplitude cycle

10 count

0,25 min

Picture 13. Amplitude modification domain settings for the testing application.



Picture 14. 0.4 seconds of the sound data modified by amplitude modification.

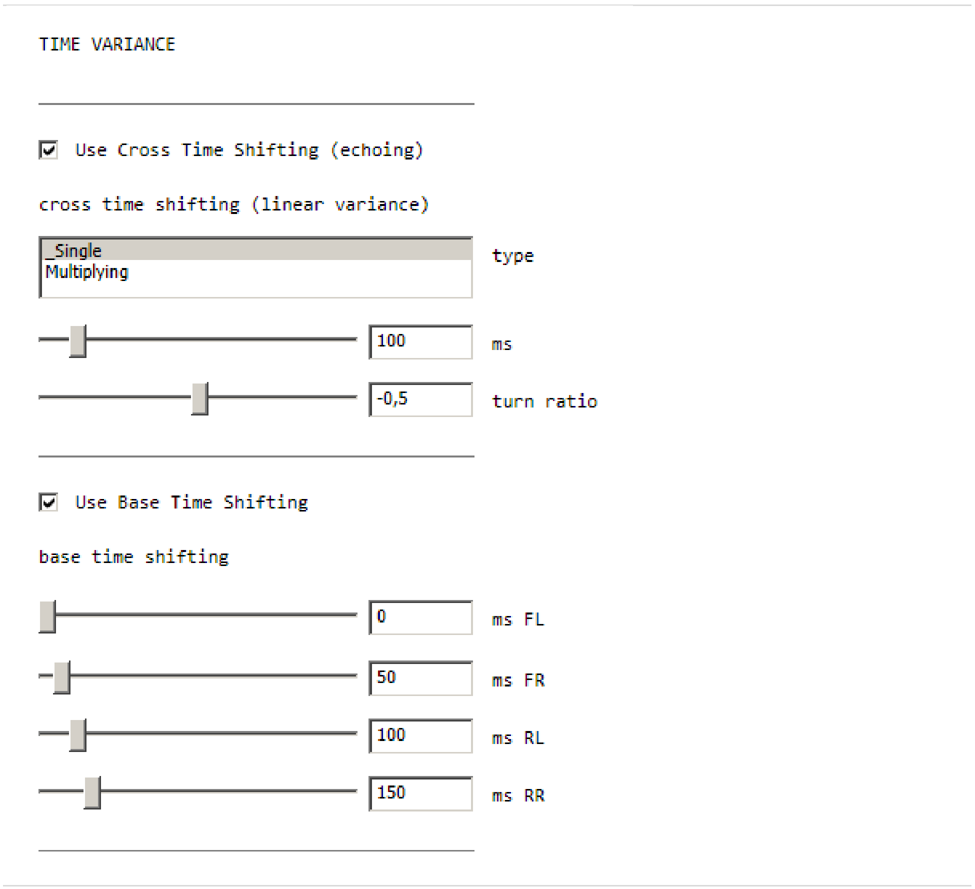
4.4.4 Time domain

Time domain of sound characteristic domains is where sound features like echo and distance appear. In my testing application I tested echoing, and base time shifting that makes up distance feel.

In the test application user interface, Echoing is explained with the term Cross Time Shifting, and this is how echoing is made up on separate speakers that is, playing the sound of the one speaker and replaying it on other speaker after specific time period. The Multiplying echoing makes echo to be added back on the source speaker and then again on the echoing speaker and so on gradually fading. This multiplying echo method does not seem to be quite valid, as it causes some amplifying effect on the beginning of the echoing. But as my thesis is in general about learning the digital sound processing, I am not a professional in the field of Digital Signal Processing.

The base time shifting of the testing application simply shifts (delays) the whole single speakers sound data specified amount of time making up some feel of space on a four speaker configuration. Term Delay is not used here because sound data could also be advanced by shifting the ahead sound data to be played at the current playing time.

In the picture 15 there is a user interface part of the time variance domain modification of the testing application. Picture 16 shows the time variance modified sound data with the length of 40 milliseconds (0.04 seconds). As the base sound for each speaker is the same, the cross time shifting (echoing) with the specific time period causes the modified sound data to be same at the period of this specified time period on the each speaker in relation to each other. That can be seen on a picture 16 where FL and RL, and FR and RR speakers have the same sound wave data at the same position, as the settings for the cross time shifting (echoing) have been 100 milliseconds and base time shifting differences for these speakers have also been 100 milliseconds.



TIME VARIANCE

☑ Use Cross Time Shifting (echoing)

cross time shifting (linear variance)

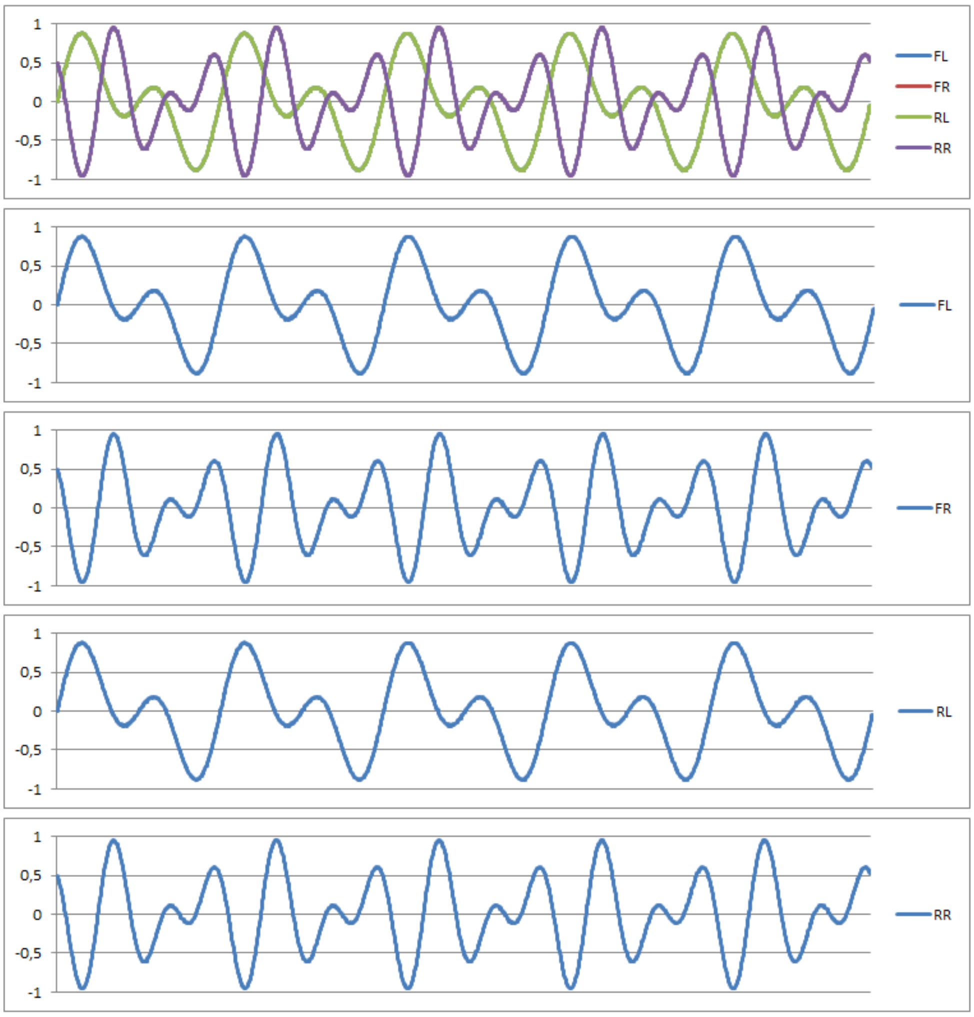
_Single / Multiplying	type
100	ms
-0,5	turn ratio

☑ Use Base Time Shifting

base time shifting

0	ms FL
50	ms FR
100	ms RL
150	ms RR

Picture 15. Time variance domain settings for the testing application.



Picture 16. 0.04 seconds of the sound data modified by time variance modification.

4.4.5 Space domain

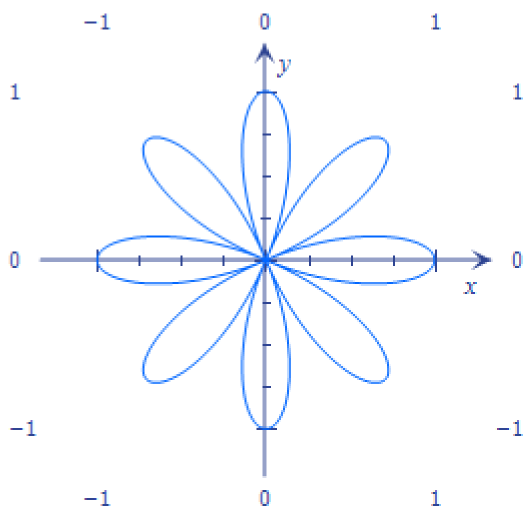
Space domain of the sound characteristics domains represents the position of the sound. In surround sound it is the position of the sound in front-back (y-axel on Cartesian coordinates) and left-right (x-axel on Cartesian coordinates) dimensions. Single mono sound can have exact position in Cartesian xy-space (Wikipedia 2013d). The testing application explained above produces one surround sound in four speakers. This four speaker sound cannot exactly be moved on xy-space. To move it exactly on xy-space the speakers would have to be moved physically. This four speaker sound can still be moved with stereo-sound-like adjusting of the sound in xy-dimensions in comparison to stereo sound x-dimension.

In xy-dimensional space the sound can practically be moved with algorithms when using polar coordinates (Wikipedia 2013e). For example, to move sound in flower-like path, we can use polar rose function like seen on picture 17.

$$r = \cos(4\theta)$$

Picture 17. Function for creating eight petals rose path (Wikipedia 2013f).

This rose function produces path like on picture 18.



Picture 18. Rose path on polar coordinates.

Unfortunately, as I worked for the subject of this chapter 4.4. for an extra 10 credits, the implementation of the last part of this chapter anyhow left on its half way.

5 RESULTS

5.1 Produced plug-in application structure

The program, produced in this thesis-work consists of main-application written with Visual Basic.NET on Visual Studio 2010 Ultimate *WPF-application* template with .NET-framework version 3.5 (you cannot use newer version of .NET-framework because those are not supported by managed DirectX) and Plug-in effect .dll-files that are first programmed with same prerequisites as main program in the form of *WPF-application*, that is then adapted into *Class Library*-project type, where the *WPF-User Control* item is added to project representing the UI and Visual Basic.NET code of the .dll-effect plug-in file (Microsoft 2009b; Microsoft 2013i).

.dll-plug-in files are loaded into the main application with *System. Reflection.Assembly*-class and the *System.Appdomain.CreateInstanceFrom AndUnwrap*-method and then exposed in *WPF-window* item with *System. Windows.Controls.ContentControl.Content*-property (Microsoft 2013j; Microsoft 2013k; Microsoft 2013l).

In general, it is a good practice to set a separate *Thread* for the sound data processing, for example to use *System.ComponentModel.Background Worker*-class (Microsoft 2010), so that User Interface-thread does not get blocked while sound data is processed.

5.2 Produced 3D-sound effects

Unfortunately, 15 credits, the volume earmarked for the UOAS Thesis, have not been anywhere near enough to fulfill the qualifications set for the Thesis work. Therefore, the results following the actual objective of the Thesis work, to produce 3D-sound effects, have been quite low. Anyhow, a few 3D-effects have been done and good basis for to continue the work has been made.

Fortunately, I had an opportunity for working some extra 10 credits for my Thesis. First 15 credits I worked on spring 2013 and the extra 10 points on autumn 2013. The extra credits work is reported on chapter 4.4. "Domains of the sound characteristics and programming 3D-sound". The extra 10 credits work contained studying and programming of different sound characteristics - that were here frequency, sound color, amplitude, time, and space - and making up User Interface for controlling different parameters of those.

Random 3D notes

This effect plays quite short notes (approx. 40 milliseconds) with randomly set pitch sequentially on randomly set speaker (with a four-speaker configuration). Effect sounds quite a techy.

Moving beat figures

This effect rests on the question that single sound has direction and distance and as a human hears that sound, it is mapped on a (kind of) visual space (Wikipedia 2013g; Smith et al. 2010). Therefore, when there are multiple sounds at the same time they show up some kind of visual figures at the human consciousness of the surroundings. In practice this effect plays moving drum samples in somewhat of geometry and symmetry. This sound effect-method can produce very techy beat patterns.

5.3 Fulfillments of the Thesis work objectives.

Objectives of the Thesis

The objectives set for the Thesis work - main application and sound effect plug-in files, and studying and creating mathematical methods dealing with 3D -sound and -effects - got reached in a fundamental level.
Programmed application is of good quality in a structure and useful for the programming work further of the subject and in a general way a good basic structure for any programming work.
Sound data producing and modifying methods and algorithms programmed are useful in a fundamental way, and in a present form already produce some good quality sound and surround sound.

The questions of the work got answered on actual Thesis work and on this Thesis report.

Own studying motives

In general, this Thesis was about studying programming in general and studying programming of sound.
As in general, programming is learnt by reading the documentation of the specific programming language and by actually doing programming. And as so, some learning of programming with WPF / XAML / .NET, quite a much of programming DirectX DirectSound, and fundamentals of programming sound, got reached.
Working around a sound data at sample level gives a good learning point for sound processing and producing, so has also been in the case of this Thesis work.

5.4 Possible future directions of the research and development of the subject

Based on the sound characteristics domains research done here and developed further, different kind of base and 3D-sound effects could be developed.

Using eight speaker configuration, containing four lower and four upper speakers, the y-dimension of the space could be used for sound effects.

If the sound data buffer based positioning of the 3D-sound is in use, in comparison to Vector3 based positioning, the sound data can be saved on a disk in a form of sound file like e.g. .wav-file.

6 INFERENCES

6.1 Methods of 3D-sound effect positioning

As in DirectSound programming, the position of sound can be controlled with *SecondaryBuffer´*s *Buffer3D*-object and its *Vector3*-properties, or the position can be controlled directly by programmatically adjusting the sound data of each sound data buffer representing each separate speaker of the surround sound speaker configuration. Controlling each sound data buffer’s sound data programmatically in conjunction with each other gives unlimited possibilities in producing 3D-sound and 3D-sound effects. This is where mathematical algorithms stand in the front line. While controlling 3D-sound with simple three-dimensional vector is more a practical and straightforward way of controlling a single sound position in 3D-space.

6.2 Inferences of the 3D-sound in the consideration of art

Sound as an art is most likely to be present as music. Although there are differences in the definition of the music, we could say that music comprises of the rhythm, sound color (or timbre), pitch and harmony (Wikipedia 2013h; Wikipedia 2013i; Wikipedia 2013j; Wikipedia 2013k; Wikipedia 2013l). If we think of 3D-sound, this is where we work in new element for music, that could be like a new dimension on the art of sound and music.

7 LIST OF SOURCES

Lyons, R. 2010. 3rd Edition. Understanding Digital Signal Processing. Discrete sequences and systems. Referenced 24.1.2013. http://www.amazon.com/ Understanding-Digital-Signal-Processing-Edition/dp/0137027419

Microsoft. 2009a. Thread.Sleep Method (Int32). Referenced 7.2.2013. http://msdn.microsoft.com/en-us/library/d00bd51t(VS.90).aspx

Microsoft. 2009b. UserControl Class. Referenced 4.2.2013. http://msdn.microsoft.com/en-us/library/ system.windows.controls.usercontrol(VS.90).aspx

Microsoft. 2010. BackgroundWorker Class. Referenced 7.2.2013. http://msdn.microsoft.com/en-us/library/ system.componentmodel.backgroundworker(VS.90).aspx

Microsoft. 2011. Random Class. Referenced 7.2.2013. http://msdn.microsoft.com/en-us/library/system.random(VS.90).aspx

Microsoft. 2013a. DirectX 9.0 for Managed Code. Referenced 24.1.2013. http://msdn.microsoft.com/en-us/library/bb318658(VS.85).aspx

Microsoft. 2013b. MSDN Blogs. Where is the DirectX SDK? Referenced 3.2.2013. http://blogs.msdn.com/b/chuckw/archive/2012/03/22/where-is-the-directx-sdk.aspx

Microsoft. 2013c. Technologies for developing Windows Store games for Windows (Windows Store apps). Referenced 3.2.2013. http://msdn.microsoft.com/en-US/library/windows/apps/hh452780.asp

Microsoft. 2013d. Windows RT: Frequently asked questions. Referenced 3.2.2013. http://windows.microsoft.com/en-US/windows/windows-rt-faq

Microsoft. 2013e. DirectX. DirectSound. Playing sounds (Managed Code). Referenced 24.1.2013. http://msdn.microsoft.com/en-us/library/ms804971.aspx

Microsoft. 2013f. DirectX. DirectSound. 3-D Sound (Managed Code). Referenced 24.1.2013. http://msdn.microsoft.com/en-us/library/ms804990.aspx

Microsoft. 2013g. SecondaryBuffer.SecondaryBuffer(Stream,Device) Constructor (Microsoft.DirectX.DirectSound). Referenced 7.2.2013. http://msdn.microsoft.com/en-us/library/ms812402.aspx

Microsoft. 2013h. Int16 Structure. Referenced 13.11.2013.
http://msdn.microsoft.com/en-us/library/system.int16(v=vs.100).aspx

Microsoft. 2013i. Class Library Template. Referenced 4.2.2013.
http://msdn.microsoft.com/en-us/library/3e6w3tyx(VS.100).aspx

Microsoft. 2013j. Assembly Class. Referenced 4.2.2013.
http://msdn.microsoft.com/en-us/library/
system.reflection.assembly(VS.90).aspx

Microsoft. 2013k. AppDomain.CreateInstanceFromAndUnwrap Method. Referenced 4.2.2013.
http://msdn.microsoft.com/en-us/library/
system.appdomain.createinstancefromandunwrap(VS.90).aspx

Microsoft. 2013l. ContentControl.Content Property. Referenced 4.2.2013.
http://msdn.microsoft.com/en-us/library/
system.windows.controls.contentcontrol.content(VS.90).aspx

Smith, D. Davis, B. Niu, K. Healy, E. Bonilha, L. Fridriksson, J. Morgan, P. Rorden, C. 2010. Spatial Attention Evokes Similar Activation Patterns for Visual and Auditory Stimuli. U.S. National Institutes of Health's. National Library of Medicine. National Center for Biotechnology Information. Pub-Med Central. Referenced 9.2.2013.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2846529/

Smith, J. 2010a. Physical Audio Signal Processing. Flanging. Referenced 24.1.2013.
https://ccrma.stanford.edu/~jos/pasp/Flanging.html

Smith, J. 2010b. Physical Audio Signal Processing. Feedforward Comb Filters. Referenced 24.1.2013.
https://ccrma.stanford.edu/~jos/pasp/Feedforward_Comb_Filters.html

Wikipedia. 2013a. DirectX. Referenced 24.1.2013.
http://en.wikipedia.org/wiki/DirectX

Wikipedia. 2013b. .NET Framework. Referenced 3.2.2013.
http://en.wikipedia.org/wiki/.NET_Framework

Wikipedia. 2013c. Milü. Referenced 25.1.2013.
http://en.wikipedia.org/wiki/Milü

Wikipedia. 2013d. Cartesian coordinate system. Referenced 13.11.2013.
http://en.wikipedia.org/wiki/Cartesian_coordinate_system

Wikipedia. 2013e. Polar coordinate system. Referenced 13.11.2013.
http://en.wikipedia.org/wiki/Polar_coordinate_system

Wikipedia. 2013f. Rose (mathematics). Referenced 13.11.2013.
http://en.wikipedia.org/wiki/Rose_(mathematics)

Wikipedia. 2013g. Auditory spatial attention. Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Auditory_spatial_attention

Wikipedia. 2013h. Music. Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Music

Wikipedia. 2013i. Rhythm. Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Rhythm

Wikipedia. 2013j. Timbre. Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Timbre

Wikipedia. 2013k. Pitch (music). Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Pitch_(music)

Wikipedia. 2013l. Harmony. Referenced 9.2.2013.
http://en.wikipedia.org/wiki/Harmony