HAL open science

Listen to the Picture! . The StatSon Sound Sonification System, using VST and DSP

Martin Heath, Gordon Hunter

To cite this version:

Martin Heath, Gordon Hunter. Listen to the Picture! . The StatSon Sound Sonification System, using VST and DSP. Acoustics 2012, Apr 2012, Nantes, France. hal-00810585

HAL Id: hal-00810585

https://hal.science/hal-00810585v1

Submitted on 23 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

"Listen to the Picture!". The StatSon Sound Sonification System, using VST and DSP

M. D. Heath[a] and G. Hunter[b]

[a]Kingston University, 229 Boston Road, Hanwell, W7 2AA London, UK
[b]Faculty of Science, Engineering and Computing, Kingston University, Penrhyn Road, KT1 2EE Kingston Upon Thames, UK
martin.dh@o2.co.uk

StatSon is a proof-of-concept C++ plugin for a VST host (for example, the Cubase music production software suite) using Digital Signal Processing (DSP). Its inputs are an audio signal that is converted to the frequency domain using a Phase Vocoder and an image that is analysed statistically. Selected statistics for the image can become pitch manipulation parameters controlling various transformations of the input signal's loudest frequency. Arbitrary input can be rounded to the closest semitone, scalar tone or tone from a selection of three or four note chords, allowing integration with existing music. A wavetable synthesizer converts the transformed frequency to a monophonic sinusoidal output, changing typically every 10ms, producing an outcome which is sometimes reminiscent of granular synthesis. The output is able to track the input and change recognisably in response to altering the image. The modified audio signal can therefore be regarded as a primitive "sonification" of the current image. The aim of the work was to create an audio effect for use live or in the studio, whilst exploring this technology and the field of image sonification.

1 Introduction

Sonification is the aural equivalent of visualisation. Kramer et al. [1] identified three aspects: research into perception, tools and applications. Tools are general purpose software modules used in sonification, such as a music synthesis engine. A sonification application is the combination of tools and possibly further bespoke software into a system involving data and users, producing measurable results. This project is intended to be somewhere between a sonification tool and a sonification application, with the potential to develop into a full application.

VST is an industry standard audio plug-in framework by the audio company Steinberg GmbH. VST originated in the Cubase music production environment and is used to create digital effects and instruments. This project is a VST effect as it manipulates an incoming digital waveform, typically a piece of music, modified using components associated with Digital Signal Processing (DSP). These include wavetable synthesis [2], the Fast Fourier Transform (FFT) [3] and the Phase Vocoder [4] (pp. 557-577).

Various summary statistics are used to provide an "interpretation" of the RGB and brightness components of images; the arithmetic mean, mode, median and harmonic mean. The user is able to use these statistics to modify the incoming music by changing its pitch using semitone, scale and chord rounding transformations.

This paper is based upon a Master of Science dissertation, from which more details can be obtained [5].

2 Sonification

2.1 What is sonification?

A simple and general definition of sonification was given in [1]: "the use of non-speech audio to convey information". An alternative was proposed by Hermann [6]:

"A technique that uses data as input, and generates sound signals (eventually in response to optional additional excitation or triggering) may be called **sonification**, if and only if: **(i)** the sound reflects objective properties or relations in the input data. **(ii)** The transformation is systematic. This means that there is a precise definition provided of how the data (and optional interactions) cause the sound to change. **(iii)** The sonification is reproducible: given the same data and identical interactions (or triggers) the resulting sound has to be structurally identical. **(iv)** The system can intentionally be used with different data, and also be used in repetition with the same data."

If the two definitions are combined, the intent of sonification is to extract "meaning" (information in some sense) from data and to display it as audio, without using speech. Note that the second definition does not preclude the use of speech, but it would have to be triggered by some form of data analysis.

Sonification combines multiple disciplines, primarily psychology (auditory perception), acoustics, music, statistics and computer science. Hermann [6] provides a breakdown of organized sound and shows the location of sonification within this framework. According to this classification, this current project is part of the intersection of sonification and music ((b) in Figure 1).

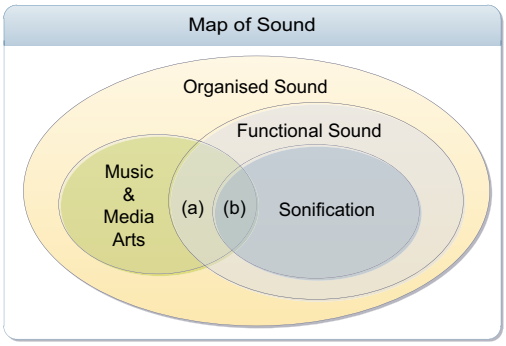

Figure 1: Sonification in relation to other categories of sound (after Hermann [6])

2.2 Sonification Middleware

Sonifications involve analysis of source data using an aural representation, typically by some kind of audio synthesis. There needs to be a mechanism to communicate between the two; this is the job of middleware. This section describes three different types. They perform different functions and so are applicable in different circumstances.

MIDI messages [7] do not contain sound, they are instructions that need to be realised by a synthesizer to create the sound. For example, a "note on" message contains a pitch number to be played, and a "note off" message stops it.

OSC [8] is used more for research, audio programming and composing. It defines some audio message and data types, but does not attempt to specify what a host application should do with the data. It supports much faster communication speeds and higher data rates than MIDI and allows digital samples to be transmitted as well as metadata.

VST is a Software Development Kit (SDK) created by Steinberg Media Technologies GmbH [9] and available free for download. The licensing conditions include

requirements that the source code is not distributed publicly. This means that VST cannot be combined with open source code.

VST provides features to enable a host application, typically music production software (also known as Digital Audio Workstations or DAWs) like Steinberg's Cubase, to either create audio (a VST instrument) or process audio (a VST effect). A GUI interface communicates with a processor via a host application. There are many VST hosts, both commercial and non-commercial. KVR Audio [10] is a useful audio plugin website containing a list of 216 VST compatible host applications.

All VST plug-ins share a common structure (Figure 2). One of the advantages of this framework is that the complexities of the communications between processor and GUI are hidden from the VST programmer. The processor can manipulate the audio based on the user input without any dependencies on the actual input mechanism.

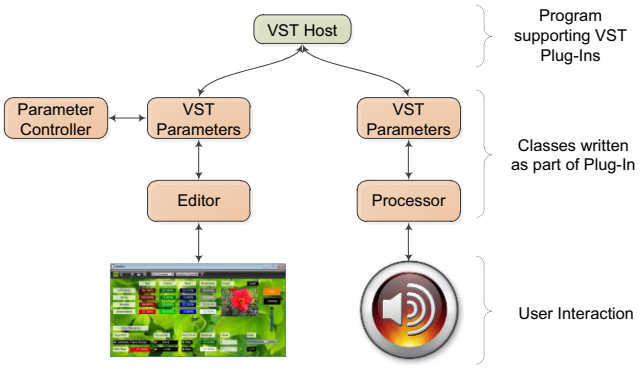

Figure 2: VST Plug-In Structure

2.3 Project Motivation

Kramer et al. [1] created a research agenda identifying a number of desirable features for sonification tools. These provide partial motivation for the design of this project's features, the selection of the underlying technologies and the software employed.

Control: the sound produced must be controllable by parameters. **Mapping**: mapping of data to sound parameters must be easy to understand and use. **Integration**: data should be able to be imported from a variety of formats. **Portability**: the system should be portable to different technical environments. **Experimentation**: mappings should allow experiments in perception. **Interaction**: the interaction with the data should be in real time. **Complexity/Simplicity**: both powerful and easy to use tools are required. **Theory**: psychological mappings of data to sound should be identified. **Distribution**: the source code should be maintainable and widely distributed.

3 Synthesis and Analysis Methods

Linear synthesis methods [11] allow sounds to be reproduced accurately but have an associated computational penalty. A form of wavetable synthesis is used here; it is an efficient implementation of the generation of a single frequency. Eq. (1) gives the computation for an entry of a single period discrete wavetable, where τ is the table, indexed by s, with amplitude A (in this case, set to 1) and of length L samples.

$$\tau_s = A \sin 2\pi n \frac{s}{L} \qquad (1)$$

Interpolation must be used when reading the table for non-integer frequency multiples. The value of L used is 1024, which gives enough entries to make the interpolated curve seem sufficiently smooth. Additional processing is needed to vary the frequency over time, to adjust for the sampling rate and for amplitude scaling; a smoothing function over 20 samples is used to avoid abrupt jumps and consequent artefacts.

In order to manipulate incoming sound and music, StatSon needs to be able to extract the frequency components. Various standard mechanisms have been used and the implementation followed a path from DFT, via FFT and DSTFT to PV. The first two are standard techniques, but the others warrant brief explanations.

Discrete Short Time Fourier Transform (DSTFT) [4] (pp. 540-555): In practice, an infinite time period and unchanging signal are not useful. When the spectrum is dynamic, i.e. the signal changes over time, a mechanism is used to isolate short time portions of the signal. The one that worked well for this project was a "Hanning Window" - a sinusoidal amplitude window producing output that can be processed by an FFT. After each stage, the window is moved on by a "hop" of a quarter of the window size which produces a continuously changing frequency spectrum.

Phase Vocoder (PV) [4] (pp. 557-577): the drawback of the STFT is that it produces output as complex numbers. The Phase Vocoder transforms these into amplitudes and frequencies by converting the complex numbers to polar coordinates, taking the difference between the phases in successive time periods, bringing the phases back into range (-π to π) and scaling appropriately to obtain the frequencies. The amplitudes are the magnitudes from the polar conversions.

4 StatSon Functionality

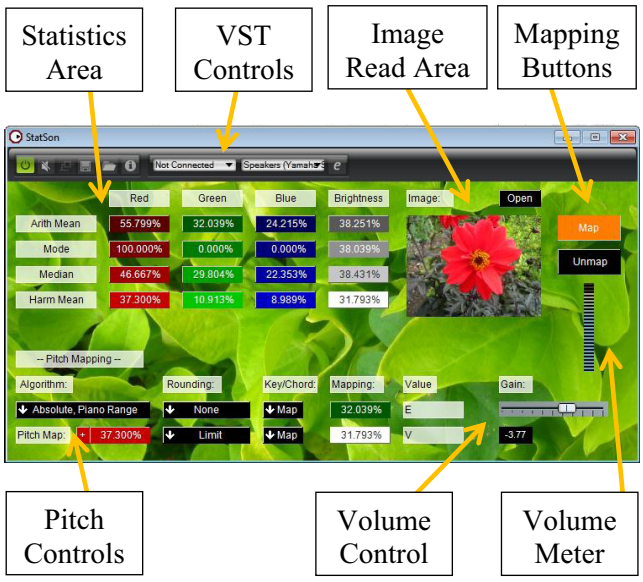

Figure 3 StatSon GUI and Major Control Areas

StatSon is a VST effect. This means it must have an audio input and output within a host program that supports VST. In Cubase, it can be used as an "Insert" on an audio

track. StatSon will receive a mono digital wave signal and its mono output can go to the mixer.

Using the "Open" button selects an image from the file system using a standard Windows dialogue allowing many types of image files to be decoded. Images are analysed to extract the pixel intensities for the three primary colours (RGB). These and their calculated brightness values are fed into the summary statistics calculations to create the statistics display buttons.

The pitch controls consist of two sub-sections, the pitch creation / manipulation algorithm and pitch rounding. The first gives the option to select an algorithm and a value (the Pitch Map parameter) that is used to raise or lower a basis pitch determined by relative or absolute algorithms.

Relative algorithms use the audio input into StatSon. For example, a Cubase audio track or a single frequency sine wave in the test host. This broken up into short time intervals (typically 10ms for CD quality audio) and each of these is analysed to find the loudest frequency present. **Absolute** algorithms ignore the input and start with a reference pitch of middle C. This feature is intended more for testing purposes.

Further options control how the pitch is transformed. If no **Pitch Map** parameter has been mapped from image statistics, the basis frequency will be left unmodified, irrespective of which relative or absolute option is chosen. Otherwise, the basis frequency at any point in time will be modified according to further algorithms given here.

One Octave means that the pitch can be raised or lowered by up to one octave. The Pitch Map parameter determines how far up or down to go. **Two Octaves** is the same but over a wider span. **Piano Range** is over the full range of the piano (A0 = 27.5 Hz to C8 = 4,186 Hz). Listening tests have indicated there is not much musical merit in pitches outside of this.

Signed means that the user selectable sign (+ or -) will be taken into account. For example, "**Relative Signed 1 Octave**" means that a Pitch Map value of +100% will raise the basis pitch by 1 octave, -100% will lower it by an octave and 0 will leave it unchanged.

If **Signed** is not present, it means that the sign will not be used. For example, "**Relative 1 Octave**" means that a Pitch Map value of 100% will raise the basis pitch by 1 octave, 0% will lower by 1 octave and 50% will leave it unchanged.

Manipulations can result in notes of any frequency, not just at traditional semitone values. This will not normally produce usable musical results, so a pitch rounding mechanism is available. The options are: (i) **None** - no rounding, the actual frequencies are used; (ii) **Semitone** - rounds up or down to the nearest semitone; (iii) **Scalar Tone** - rounds up or down to the nearest scalar tone and (iv) **Chord Tone** - a menu allows any basic triad or seventh chord on any scale degree to be selected via a list of I to vii and I7 to vii7. The **Map** option allows the selection of a chord from an image statistic and a label shows the choice generated

A menu contains the names of the major scales from C to B. Sharp keys can be selected via their enharmonic flat keys, for example G♯ is identical to A♭. Natural minor keys can be selected via their relative major, for example A minor has the same key signature as C major. The **Map** option allows selection from an image statistic and a label shows the actual value generated. 0% would select the first key, namely C, and 100% would select the last, B.

These pitch transformations can move pitches to levels outside the range of human hearing (e.g. when moving the lowest note down an octave). Options control how these are processed. If **Limit** is selected, then these notes will be capped at the highest or lowest notes of human hearing, taken to be 20 Hz and 20,000 Hz. **Final Octave** produces a more natural sounding result. Pitch classes are retained (e.g. the note A) but notes are moved to the closest in range octave (high or low, as appropriate).

5 Development

The main hardware used for project development was a Windows 7 PC with Intel i7 2.8 MHz (quad core, 8 processor) CPU, 16 Gb RAM and Steinberg's MR 816x FireWire audio interface. This is the equivalent of a sound card and allows recording, playback and some DSP on 8 channels of high resolution audio (24 bit / 96kHz). The principal software used was Microsoft's Visual Studio 2010, the Visual C++ development language and the VST software development kit (SDK).

Cubase 6 is the latest version of Steinberg's music production software and writing a plug-in for it was one of the key motivators of this project.

5.1 The VST SDK

The VST SDK provides the following components:

VST documentation: there are some manually written overview pages along with some possibly automatically generated detail pages. As a learning tool it could do with considerable improvements.

Base Modules and VST MA (Module Architecture): these are independent of VST (despite the name). The base modules provide a library of C++ classes to handle strings, templates, containers, etc. The MA is a Microsoft COM style architecture for a general host and plug-in environment.

VST 3 API: these are the classes that build on the above to implement an audio plug-in framework.

Helper Classes: these sit above the VST 3 API to provide classes that are useful for the creation of real plug-ins, for example the editor (GUI), parameters, busses etc.

Plug-In Examples: these are working plug-ins such as a volume control (AGain), a delay effect, a multi-effect plug-in and a synthesizer. These were extremely useful as manual code analysis of examples was the main method by which information was obtained. However, this is not ideal for when an example of a feature cannot be found. It is also very time consuming.

Test Applications: these contain the highly useful VST3PluginTestHost standalone application that provides a test harness for plug-ins without having to wait for Cubase to load. It has sample wave form inputs such as white noise and sine wave. The cumbersome USB security dongle that all major Steinberg products use is also not needed.

VSTGUI 3.6: this is the current version of the user interface toolkit and was made open source in May 2003 [12]. It provides classes for controls such as buttons and menus as well as audio specific controls like a knob and a volume meter.

The VST framework uses older Microsoft Windows technologies. There is no reason why more modern controls could not be created, but somebody would have to do the work, most likely as part of VSTGUI. There is little

integration into the Visual Studio environment; there are no drag and drop control features or forms designer. Controls often need to be created or amended manually. For example, AGain contains the definition of a button control which needed to be amended for more flexibility in its colour handling.

5.2 Pitch Algorithms

The pitch class implements pitch rounding and frequency to pitch conversion via an 11 octave table. The frequency of pitches is calculated by reference to the A 440 Hz standard at index 57. The frequency *F* of any note can be calculated from an index *i* by the formula:

$$F = 440 \times 2^{\left(\frac{i-57}{12}\right)} \quad (2)$$

Applying the inverse transformation, it is possible to translate from an arbitrary frequency *F* to a table index *i*:

$$i = 12 \times \left(\frac{\log(F)-\log(440)}{\log(2)}\right) + 57 \quad (3)$$

Semitone rounding is performed for frequencies that do not map exactly to note pitches by converting to floating point indices. These show, on a linear scale, how close the original pitch is to its neighbours. The semitone rounded index is the input index plus 0.5, truncated to an integer. Scale and chord rounding work in a similar way but limit the set of notes that can be selected. The notes of the major scale can be given by index offsets: {Tonic: 0, Supertonic: 2, Mediant: 4, Sub-Dominant: 5, Dominant: 7, Sub-Mediant: 9, Leading Note: 11}

Any note in this sequence can have a three or four note chord built upon it by incorporating a second level of indexing into this index of major scale notes using: {Root: 0, Third: 2, Fifth: 4, Seventh: 6}

It is important to note that this numbering is not the same as that used in music theory, where indices start at one. Some extra refinements are needed in order to ensure that any input pitch is rounded into the 20 to 20,000 Hz audible range.

Once a basis frequency has been identified from the input or by using middle C (261.63 Hz), it can be transformed by the pitch manipulation algorithms. As the human perception of pitch is on a logarithmic scale, the general formula to raise or lower a pitch by a given amount is given by:

$$Output = Input \times base^{power} \quad (4)$$

To implement relative unsigned octave transformations, the base is 2 and the power must be in the range -1 to +1 to go down or up an octave. The pitch change parameters as specified in the GUI are in the range of 0 to 100%, passed to the processor in the range 0 to 1. To find the required input pitch change multiplier (as per equation (4)), use:

$$\text{Relative 1 Octave, multiplier} = 2^{(parameter-0.5)\times 2} \quad (5)$$

$$\text{Relative 2 Octave, multiplier} = 2^{(parameter-0.5)\times 4} \quad (6)$$

For the "Unsigned Relative Piano Range" algorithm, 50% means no change to pitch, 100% goes to the maximum frequency (4,186 Hz) and 0% to the minimum frequency (27.5 Hz). To raise the pitch, rearrange $pitch \times base =$ 4,186 to give $base = \frac{4,186.01}{pitch}$. To lower the pitch, rearrange $pitch \times base^{(-1)} = 27.50$ to give $base = \frac{pitch}{27.50}$. Use this new *base* instead of base 2 in the previous calculations and recalculate for each input pitch. The degree of translation between the limits depends on the input; the effect of a parameter between 0% - 50% on a low pitch is small, but the effect will be large for values between 50% - 100%.

For signed algorithms, the only difference is that the unsigned raising range 50% to 100% is replaced by 0% to +100% and the unsigned lowering range 50% to 0% is replaced by 0% to -100%. Effectively this adds a sign and divides the exponent by two.

6 Discussion

According to Hermann's [6] definition (discussed in section 2.1), this project can be thought of as a data sonification since the output is consistently repeatable from the input. However, it is probably not what would normally be expected of a sonification project as there is no attempt to do any analysis or to reveal any meaning. There is also no means to limit the area for data analysis by scanning or probing within the image beyond the current breakdowns into RGB and brightness. Even so, this does not mean this project has no sonification abilities. It could easily be configured to create an aural distinction between red and green images, for example.

StatSon shows what is possible with a very basic synthesis technique. Listening to some of the musical examples produced, it is perhaps surprising that they could have been created with a single monophonic synthesis engine. Single frequency sinusoidal input can be tracked accurately above F1 (43.65Hz). Simple classical guitar music input creates recognisable output, interspersed with high pitch "bleeps" which could be harmonics of the fundamental or noise during the attack phase of the notes. Music production software like Cubase can use multiple instances, one per part, with no latency issues. Complex musical input can result in sounds reminiscent of granular synthesis [13]. Mappings can be created so that a change of image results in the output music also changing noticeably.

A difference from previous work is that a sonification as a VST plug-in using Cubase is an unusual combination and no evidence has been found of this having been done elsewhere. One possible cause is that the licensing requirements prohibit the incorporation of most public domain software. A further difference from previous projects is that StatSon is about sonification as a musical effect. Other projects generally synthesize sound directly; here existing audio waveforms are modified, although they are re-synthesized using a wave table oscillator.

An overall assessment of the project can be obtained by rating it against the possible requirements for sonification tools, shown in section 2.3 [1]. Five out of nine requirements have been fulfilled well or to a considerable extent.

Control: pitch manipulation parameters can control the modification of the loudest input frequency at rapid, regular intervals. **Mapping**: the mapping of data to sound parameters is easy to understand and use (as it should be in a comparatively small project). **Integration**: data can be imported from all image types supported by the Windows

Imaging Component (WIC). VST plug-ins can work with over 200 types of hosts. **Interaction**: The interaction with the data is in real time. **Complexity / Simplicity**: by necessity, this is an easy to use tool rather than a powerful one.

7 Possible future work

The main future aim is to include additional features; to turn StatSon into a more useful sonification tool. These could include modifications in loudness (amplitude shaping), spatial location (panning), duration, timbre, rhythm and tempo, progressing to melody and further harmonic explorations. The synthesis engine could also be upgraded to use more advanced forms of synthesis, notably by incorporating polyphony.

Other possibilities could be to allow processing of multi-frame bitmaps, leading on to handling video streams. The ability to save and restore mappings and images (serialization) enables the use of Cubase's automation features. These would allow mappings of images and parameters to be configured manually and then be changed automatically in real-time as the track is replayed. A longer term goal is to explore more from the academic image processing discipline; to identify interesting or meaningful aspects of images, videos or general data and to be able to render these as manipulations of existing sound. This would include the ability to select, analyse and navigate through areas within the image, video or data.

8 Conclusion

A significant amount of bespoke software was written for this project; over 4,200 lines of C++ code. However, public domain FFT software [3], Microsoft software components (WIC and GDIplus), helpful reference material [2][4][11], and the VST SDK [9] were all used to speed up the development process.

On the sonification side, although this project complies with Hermann's functional definition, when Kramer et al.'s [1] more meaning-based definition is applied (see section 2.3), it can be seen that the information imparted by sonification in this project is very low. Indeed, it may be only at the level of saying that one image is different to another. Even this is not always possible as two images can quite conceivably have the same or very similar mapped summary statistics, especially in the current system where only three mappings are available. The transformation of the input signal into the frequency domain can also be regarded as a sonification under Hermann's definition, but there is similarly little meaning imparted.

This project provides a framework for further exploration of sonification via image statistics. To enable effective sonifications, these statistics should also be meaningful. However, is it possible to quantify the emotional content of an image? For example, an aggressive man with a knife should score highly on terror, a sunset over a beach framed by palm trees should rate well on desirability or beauty. However, an unexpected demonstration of the power of sonification was its successful application during debugging of the program code. Waveform discontinuities, partially filled in output buffers and incorrect windowing all have audible consequences! The first two created buzzing sounds and the last resulted in poor pitch resolution and excessive noise. Sonification has therefore provided both the motivation for this project and a useful resource in helping to debug DSP code, which could ultimately be a genuine practical application for sonification, worthy of future exploration.

References

[1] Kramer, G., Walker, B., Bonebright, T., Cook, P., Flowers, J., Miner, N.; Neuhoff, J., Bargar, R., Barrass, S., Berger, J., Evreinov, G., Fitch, W., Gršhn, M., Handel, S., Kaper, H., Levkowitz, H., Lodha, S., Shinn-Cunningham, B., Simoni, M., Tipei, S. *The Sonification Report: Status of the Field and Research Agenda*. Report prepared for the National Science Foundation by members of the International Community for Auditory Display. ICAD, Santa Fe, NM, USA (1999).

[2] Loy, G. *Musimathics: The Mathematical Foundations of Music. Volume 1*. The MIT Press, Cambridge, Massachusetts, USA (2006).

[3] Chernenko, S. *Fast Fourier transform — FFT.* [Internet] Available at http://www.librow.com/articles/article-10 (Accessed 3/10/2011).

[4] Boulanger, R. and Lazzarini, V. (eds.) *The Audio Programming Book*. The MIT Press, Cambridge, Massachusetts, USA (2011).

[5] Heath, M. D. *StatSon: Statistical Sonification using VST and DSP*, MSc dissertation, Kingston University, U.K. (2012).

[6] Hermann, T. *Taxonomy and Definitions for Sonification and Auditory Display.* In. Proc. 14th Int. Conference on Auditory Display (ICAD), June 24-27, Paris, France (2008).

[7] White, P. *Crash Course MIDI.* SMT an imprint of Sanctuary Music Publishing Limited, London (2004).

[8] Freed, A. and Schmeder A. *Features and future of Open Sound Control version 1.1 for NIME.* Proceedings of the Conference on New Interfaces for Musical Expression (NIME), Pittsburgh, PA, USA (2009).

[9] Steinberg *3rd Party Developer Area.* [Internet] Available at http://www.steinberg.net/en/company/developer.html (Accessed 2/10/2011).

[10] KVRAudio *Can Host VST Plug-ins | Windows.* [Internet] Available at: http://www.kvraudio.com/get.php?mode=results&s=11&st=q (Accessed 4/10/2011).

[11] Loy, G. Musimathics: The Mathematical Foundations of Music. Volume 2. The MIT Press, Cambridge, Massachusetts, USA (2007).

[12] Sourceforge, *VSTGUI,* [Internet] Available at http://sourceforge.net/projects/vstgui/ (Accessed 6/1/2012).

[13] Roads, C. *Microsound.* MIT Press, Cambridge, MA, USA (2001).