Audio Engineering Society

Convention Paper 7438

Presented at the 124th Convention
2008 May 17–20 Amsterdam, The Netherlands

Audio software development – an audio quality perspective

Jonas Ekeroot[1] and Jan Berg[1]

[1] *Luleå University of Technology, Department of Music and Media, Piteå, Sweden*

Correspondence should be addressed to Jonas Ekeroot (jonas.ekeroot@ltu.se)

ABSTRACT

When developing audio applications, different choices on software implementation aspects influence the total audio software signal path and can be of importance from an audio quality perspective. The field is not well documented in the literature. A study was carried out aiming at identifying relevant questions that must be considered. The general development perspective was on audio software written in C++ to be run on general purpose CPUs. A research review, comprising literature from different fields such as audio engineering, computer science and software engineering, was conducted to summarize and integrate an overview of the field. The result can be viewed as a map of questions for future research activities, consisting of further literature studies and experiments with software prototypes.

1. INTRODUCTION

The components of an audio system all contribute to the system's audio quality. Great efforts have been made to develop the quality of these components, especially those aimed for professional use. A computer handling audio information is now an important component of numerous audio systems and the quality of the audio handling within the computer is therefore of vital importance. The different components and processes of the computer can be thought of as parts of the audio signal path. Consequently the software can be considered as the audio software signal path.

The audio software signal path through a computer audio system can be quite complex and hard to unambiguously elucidate. Kelly [1] noted:

> "The process of getting audio in and out of the computer is not always clear to the uninitiated, even to an otherwise experienced programmer."

He also identified a common need for custom software tools in audio research. When developing such audio applications, e.g. listening test software, an optimal technical audio quality is strived for in the sense that no unknown or uncontrolled processing or degradation of the audio signal has occurred as a result of the computer system's data handling itself, and consequently that no audio quality degradation can be detected. A contrasting meaning of audio quality, primarily based on ease of use, could be claimed in a home user environment. In this paper audio quality should be understood in the former sense above.

Many models for software development are built on the classic waterfall model, described by Royce [2]. In summary the main phases, based on Wohlin [3], are requirements specification, design, coding (alternatively referred to as implementation) and testing. As found by Boehm *et al* [4] ambiguity in the software requirements specifications is one of the biggest sources of software problems. Ekeroot [5] treated audio software development extensively, and pointed out the need for a considerable element of professional audio production knowledge in all the software development phases. Anecdotal experiences from broadcasting (Swedish Radio, SR) have indicated problems with non bit-transparent handling of multi-channel encoded audio streams, inter-channel delays, quantization distortion due to lack of dithering, CPU[1]-heavy floating point applications and the paradoxical situation where full system load (by a custom dummy application) leads to uninterrupted playback by audio applications while low system load leads to interruptions.

Software metrics can be defined as the field of measurement in software engineering, as discussed by Wohlin *et al* [6]. Fenton and Pfleeger [7] treated the modeling of software quality and measuring aspects of quality and stressed the importance of striving for fact, not opinion, and to question claims to see if they are based on empirical evaluation and data rather than intuition and advocacy. Kan [8] further remarked on the ambiguity and multidimensionality of software quality, where the dimensions of quality include entities of interest and quality attributes of those entities. The type of systematic approach that exist in this field needs to be applied also in an audio software quality context, to enable control and improvement and as a contribution to the prevention of problems such as those briefly exemplified above.

The perspective of development in this paper is on software written in C++, using an audio application programming interface (API) provided by an operating system platform such as Windows, Mac OS X or Linux, to be run on a general purpose CPU, i.e. not software written in assembler to be run on a specialized DSP chip.

Although knowledge within the area treated in this paper might presumably already exist within commercial software development companies as corporate protected knowledge, there is a need for a free text that treats such issues.

The objective of this paper is to show the need for a more stringent approach to audio software quality, to address issues that have to be given an audio quality emphasis including an initial attempt to define a uniform terminology, and to compile and present questions that have to be addressed in audio software development.

2. TERMINOLOGY FOR AUDIO SOFTWARE DEVELOPMENT

The terminology within the field of audio software development has shown to be inconsistent in the sense that various meanings can be given to a specific word. In order to present some of the inconsistencies of the terminology as well as to define some key concepts, selected terms are reviewed below, followed by short examples of differencies in meanings for definitions that show ambiguity. Different meanings can also exist due to differences between target platforms, e.g. Windows, Mac OS X and Linux. Yet, to promote and enable unambiguous communication with clear distinctions within the field, a uniform multi-platform terminology free from uncritical acceptance and vague terms, would be of great value. This paper contributes a proposal of an initial set of definitions, given in a conceptual order below, to be used as a more uniform terminology. Figures 1 and 2, presented by Ekeroot [5], illustrate the relationships of some of the terms.

computer audio system A personal computer with some form of installed *audio hardware* for audio input/output (I/O), e.g. a sound card or

[1] central processing unit

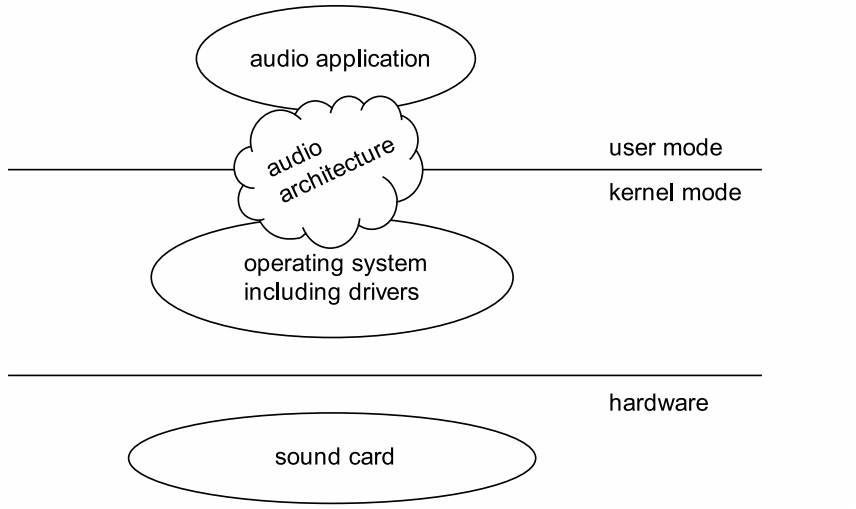

Fig. 1: The vague term audio architecture illustrated as a cloud somewhere in and/or between an audio application and an operating system. After Ekeroot [5].

an external audio interface, and *audio software* for recording, editing, mixing, etc. Normally no ambiguity in the term, but included for completeness.

user mode Machine instructions pertaining to audio applications (see below) run on the CPU in this mode [9], where simultaneously running applications are protected from interfering with each other's resources, e.g. memory areas, to avoid application and operating system crashes. Normally no ambiguity in the term from an audio perspective, but needed for following definitions. See figure 1.

kernel mode Machine instructions pertaining to the operating system run on the CPU in this mode [9]. Such code has more unrestricted access to resources in the computer audio system. Typcially better performance than for user mode software. Normally no ambiguity in the term from an audio perspective, but needed for following definitions. See figure 1.

audio application User mode software, normally with a graphical user interface (GUI) for human user interaction like recording, editing, mixing, etc. See figures 1 and 2.

native audio API An audio application programming interface (API) that defines and provides the high-level, e.g. C++, functions that an audio application can call to get audio functionality, like communicating with installed audio hardware. Implemented in user mode. Normally proprietary. A "native" audio API is one that is provided by the operating system on the respective platform. One or several native audio APIs can exist for the same operating system. See figure 2 and section 4.1.1.

multi-platform audio API An audio API that is layered on top of a number of native audio APIs on multiple platforms. Implemented in user mode. Normally open source. Is dependent on and calls audio functions provided by the underlying native audio API. See figure 2. Examples include PortAudio[2], OpenAL[3] and JACK[4].

driver Kernel mode software that communicates with the underlying audio hardware on a low machine level. Provides a level of abstraction that conceals hardware level details for a specific user mode native audio API that is layered

[2] http://www.portaudio.com/
[3] http://www.openal.org/
[4] http://jackaudio.org/

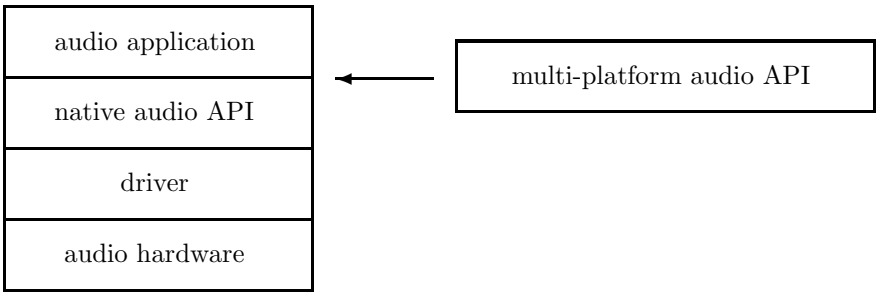

Fig. 2: A formal view of the general layered audio software structure in a computer audio system, including the possible use of a multi-platform audio API. After Ekeroot [5].

on top of the driver. See figure 2.

audio architecture Computer architecture treats hardware-related issues like memory organization, input/output devices and bus structure [9]. However, architecture can also refer to software-related high-level design issues in the software devlopment process [3]. Typical use of the term audio architecture refers to user and kernel mode software that, in addition to an audio application, native audio API and driver, are part of the audio handling in an operating system. See figure 1. This vague term may cause uncertainty about what is referred to, and its use needs clarification.

audio subsystem The hardware and software in a computer audio system, including e.g. native audio API and driver, that provides audio functionality.

audio engine A term with different known uses in various layers in figures 1 and 2. This vague term may cause uncertainty about what is referred to, and its use needs clarification.

audio software signal path The *total* audio signal path through *all* the user and kernel mode *software* in an audio subsystem in a computer audio system.

The above list of definitions is an initial attempt to compile such terminology, without any claim as to being complete and exhaustive. Changes, additions and refinements will possibly be made by future work.

Examples of confusing use of terminology which led to the formulations above include the use of the notion *audio API* without a distinction between *native* and *multi-platform*. The dependence of a multi-platform audio API on an underlying native audio API is then not explicitly treated, as in Kelly [1] when discussing OpenAL. A confusion between drivers and native audio APIs was reported in a Windows professional audio white paper [10] under *Observation 3: The term "driver" is misunderstood*:

> "A true 'driver' runs in the kernel ... Technologies like MME, ASIO ... are merely user-mode APIs, not drivers."

MME and ASIO are discussed in section 4.1.1. Further examples of confusing use of terminology were treated by Ekeroot [5].

Summing up this section, a condensed view of a general layered audio software structure, including an optional multi-platform audio API, can be formalized as in figure 2. The inconsistencies in terminology is also a part of the problem of achieving an accurate and unambiguous understanding of the software components essential for audio quality.

3. METHOD

The study reported in this paper was focused on identifying questions that have to be asked and carefully considered in audio software development on

issues that can have an influence on audio quality. The model for the research process for the study was a *research review*, i.e. a summarizing and integrating overview of the state of knowledge through a literature review. The motivation for this approach – to make a research review that results in questions for future studies – was that the topic of the study can be viewed as a new subfield within the subject of audio technology, with little written about it. A professional audio perspective guided the summarizing review, which also observed what is still missing in literature of relevance to audio software development from an audio quality perspective.

3.1. Finding relevant literature

Audio software development is not traditionally treated in the audio engineering literature. Other disciplines such as computer science and software engineering had to be searched for literature to provide useful information. Finding relevant literature for the study was done by searching the electronic databases Audio Engineering Society, Compendex and Inspec. Also on-line sources were searched and studied. See section 4.1.2 and associated footnotes.

Audio related open source code projects such as CLAM[5], SndObj[6], Audacity[7] and Ardour[8] could also provide detailed insights on coding practices, and be used for further experimentation by modification and prototype development. The effort required to gain a detailed understanding of such projects highly depends on how well structured and commented the source code is and is normally quite time consuming. This was therefore left outside the scope of this study, and remain for future research activities.

3.2. Strengths and weaknesses

The choice of restricting this study to a literature review has some consequencies. A weakness is that experimental verifications of observations have not been carried out yet. Though, the scarce amount of previous work in audio software development from an audio quality perspective makes an inventorying and summarizing review approach a strength that can point to necessary studies to perform, theoretical as well as experimental.

[5] http://clam.iua.upf.edu/
[6] http://sndobj.sourceforge.net/
[7] http://audacity.sourceforge.net/
[8] http://ardour.org/

4. RESULTS OF THE LITERATURE REVIEW

This section presents observations resulting from the literature review, and is organized into four parts treating audio subsystem, floating point, audio quality and questions that are essential to address in audio software development.

4.1. Audio subsystem

On a high abstraction level a computer can traditionally be described as by Patterson and Hennessy [11] as organized into the five fundamental components *input*, *output*, *memory*, *datapath* and *control*, where the last two collectively are referred to as the CPU. This paper mainly treats input and output (I/O) in the context of audio samples, focusing on software aspects of the audio subsystem involved in the communication between an audio application and some audio hardware. Referring to the terminology treatment in section 2, the term audio subsystem is consistently used throughout this paper, and the vague term audio architecture is avoided.

The electrical characteristics and objective quality measurements of hardware for audio I/O, including analogue/digital (A/D) and digital/analogue (D/A) converters, were left outside the software focused treatment in this study. Such hardware issues were discussed by Jones *et al* [12] and in the *AES Information document for digital audio – Personal computer audio quality measurements* AES-6id-2006 [13].

4.1.1. Native audio APIs

Current native audio APIs for Windows, Mac OS X and Linux were summarized and presented by Ekeroot [14] and later treated further in [5]. They are listed in table 1. The majority of the native audio APIs are proprietary with the exception of ALSA, where open source code is available[9]. The latest native audio API by Microsoft is WASAPI and it is only available in Windows Vista. A specific native audio API on a target platform defines a collection of user and kernel mode components that in a hierarchical fashion communicate with each other to make up parts of the audio software signal path.

4.1.2. Audio software signal path

When developing audio applications, different choices between native audio APIs result in different

[9] http://www.alsa-project.org/

platform	native audio API	company
Windows	MME (Multimedia Extensions)	Microsoft
	DS (DirectSound)	Microsoft
	WASAPI (Windows Audio Session API)	Microsoft
	ASIO (Audio Stream Input/Output)	Steinberg
Mac OS X	Core Audio	Apple
Linux	ALSA (Advanced Linux Sound Architecture)	open source

Table 1: Current native audio APIs for Windows, Mac OS X and Linux. After Ekeroot [5].

audio software signal paths. An intial attempt to illustrate the resulting audio software signal path in a summarized way for different native audio APIs on Windows, Mac OS X and Linux, including information about the actual user and kernel mode software components, was presented by Ekeroot [5] [14]. It was based on information extracted and compiled from *Microsoft Developer Network*[10], *Steinberg 3rd Party Developers*[11], *Apple Developer Connection*[12] and *Advanced Linux Sound Architecture – ALSA*[13]. The main characteristics of the audio software signal path for each native audio API in table 1 are summarized in the following paragraphs.

An MME audio application under the current Windows Driver Model (WDM), e.g. on Windows XP, calls audio functions provided by the file `winmm.dll`. Several other user and kernel mode files are part of the stack of software components that make up the audio software signal path. This path includes a kernel mode audio mixer, `kmixer.sys`, which can perform audio sample format and sample rate conversion. The only control of this mixer is via a control panel, `mmsys.cpl`, with a GUI slider to adjust the quality of the sample rate conversion in three positions – Good, Better, Best – without further indication of conversion details.

For a DirectSound (DS) audio application under WDM, calling `dsound.dll`, the kernel mode audio mixer can either be in or out of the audio software signal path. This depends on specifications of the audio hardware, but is out of control from the source code in the audio application layer.

An ASIO audio application, e.g. on Windows XP, bypasses all the Microsoft software layers compared to MME/WDM and DS/WDM including the kernel mode audio mixer.

In Windows Vista the Windows Audio Session API (WASAPI) was introduced. MME and DS can still be used but are layered on top of WASAPI. Most of the kernel mode software components in MME/WDM and DS/WDM, except for the driver part, have been removed and replaced with a new user mode Global Audio Engine (GAE), with mixing functionality reminiscent of the kernel mode audio mixer in the MME/WDM and DS/WDM cases. There is a possibility to bypass the Global Audio Engine altogether by using the WASAPI native audio API directly, which gives a situation similar to using ASIO.

CoreAudio and ALSA were for time priority reasons so far only briefly examined, and detailed studies remain. In the ALSA case software internals can be

[10] http://msdn.microsoft.com/
[11] http://www.steinberg.net/324+M52087573ab0.html
[12] http://developer.apple.com/
[13] http://www.alsa-project.org/

analyzed in detail through the access to open source code.

Other attempts to illustrate audio signal paths in a computer audio system reported in the literature, were made by Jones *et al* [12] and in AES-6id-2006 [13]. These documents contain signal path diagrams solely for a general Windows case, without specific indication of the intended native audio API. Nor is information about the actual user and kernel mode software components given.

4.1.3. Level diagram

Level diagrams for audio subsystems which indicates nominal level, maximum allowed level before clipping and noise floor are so far missing, or as Jones *et al* [12] expressed it:

> "This is an area of difficulty in PC audio devices; there is seldom an adequate indication of signal level at any part of the signal path."

From an audio quality perspective, such level diagrams would be a helpful complement to signal path diagrams in studies and discussions.

4.2. Floating point

Goldberg [15] commented on the ubiquity of floating point arithemtic in computer systems and the common esoteric view of it. Knuth [16] further noted the non-triviality of the subject contrary to general belief and stressed the importance of knowledge about the topic for programmers. As discussed by Ekeroot [5], audio applications commonly use floating point audio sample representation, providing large headroom and large dynamic range in intermediate signal processing.

Current computer audio systems based on general purpose CPUs from e.g. Intel and running Windows, Mac OS X or Linux, handle non-integer numbers according to the *IEEE Standard for binary floating-point arithmetic* ANSI/IEEE Std 754-1985 [17].

Schwarz *et al* [18] remarked on the difficulty of hardware implementation of denormalized numbers, i.e. very small numbers close to zero, in floating point units leading to some designs handling them in software, resulting in problems with increased execution times. De Soras [19] reported on the "denormal bug" where the CPU load increase for denormal number processing eventually can affect application stability. He proposed resetting the denormalized values to the normalized value zero, thereby eliminating them.

4.2.1. C++ data types

Figures 3 and 4 summarize the C++ data types available for audio sample representation, including typical maximum and minimum (i.e. 0 dB full scale) audio sample values. IEEE 754 formats and corresponding C++ data types (in parenthesis) are:

- 32-bit single format (`float`)
- 64-bit double format (`double`)
- 80-bit double extended format (`long double`)

The fixed point integer audio sample values, asymetrically distributed around zero due to two's complement representation, typically span their whole respective number range leading to an absolute clip limit at the extreme values (dark areas on vertical axis in figure 3). 24-bit fixed point integer audio sample values can be handled using 32-bit `int` since C++ does not have a 24-bit fixed point integer data type.

Floating point values on the other hand are symmetrically distributed around zero and have +0.0 and -0.0, both normally treated as zero. Typcial audio sample values only span the range [-1.0,+1.0] giving plenty of headroom in intermediate calculations (dark areas on vertical axis in figure 4), though the signal must eventually be in the [-1.0,+1.0] range for file storage or output.

De Castro Lopo [20] discussed conversions between `short` (e.g. from a WAV file) and `float` (e.g. in an audio application). Converting `short` spanning [-32768,+32767] by dividing by 32768 gives `float` spanning [-1.0,+0.999...]. Conversion from `float` spanning the full range [-1.0,+1.0] by multiplying by 32767 gives `short` spanning [-32767,+32767]. Such conversion process is not bit transparent, i.e. it changes some audio sample values. He argued for minimizing the number of conversions to maintain audio quality, and use fixed point integer only for final file storage and use floating point for both processing and intermediate file storage.

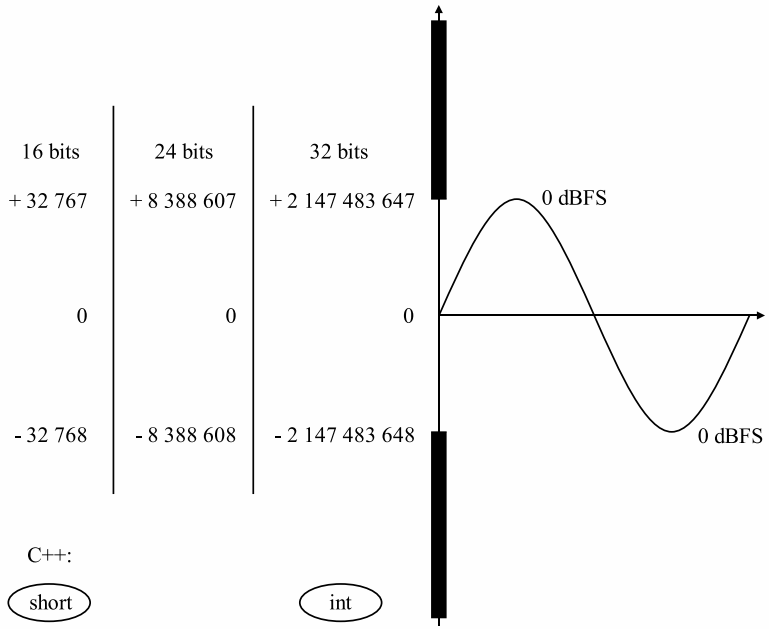

Fig. 3: C++ data types – fixed point integer. After Ekeroot [5].

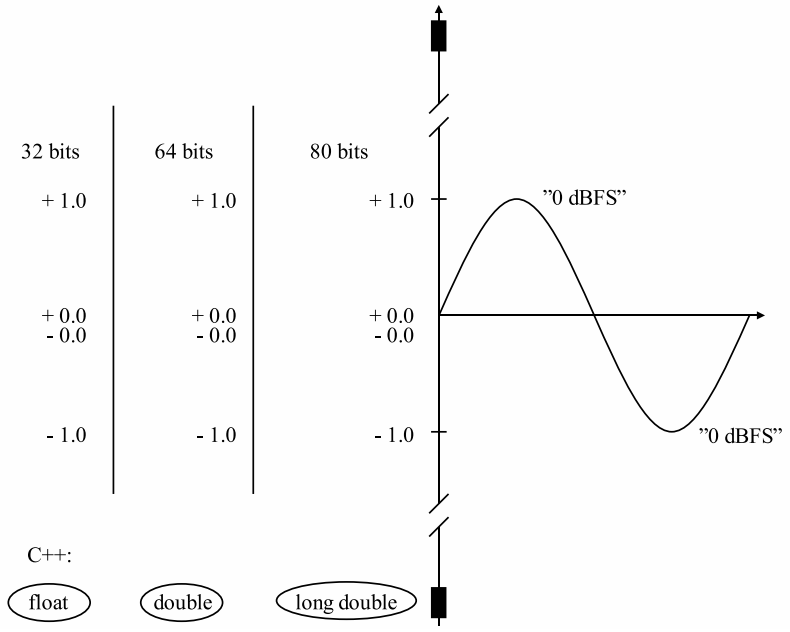

Fig. 4: C++ data types – floating point. After Ekeroot [5].

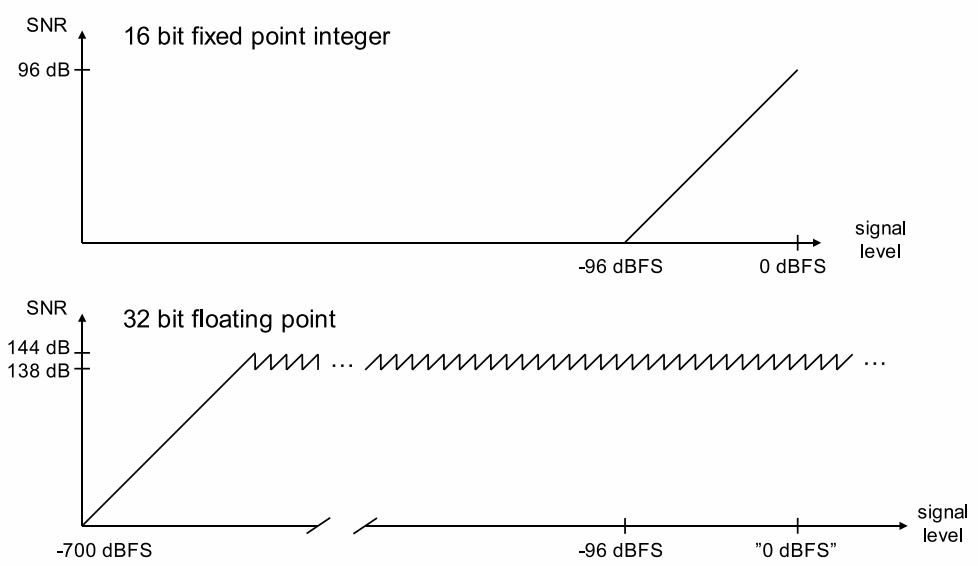

Fig. 5: SNR – fixed point integer versus floating point. After Ekeroot [5].

4.2.2. Signal-to-noise ratio

Zölzer [21] gave mathematical derivations for the resulting signal-to-noise ratio (SNR) for both fixed point integer and floating point numbers, and this was also treated by Muheim [22] in the context of the design and implementation of a computer audio system. Ekeroot [5] [14] presented a summarizing illustration comparing SNR for `short` and `float`, which is shown in figure 5.

As floating point numbers approach zero, the exponent scaling the fraction takes on smaller values. Thereby the gap between consecutive numbers gets smaller and hence the precision increases. So, when the floating point signal value decreases, so does the noise floor for every decreasing step of the exponent value. This gives the sawtooth shape in the `float` case, down to the denormalized numbers several hundred dB down from "0 dB full scale" level. In summary, this gives a large dynamic range for floating point numbers.

4.2.3. Dither

The general dither literature scarcely treats dither in a floating point context, but explicitly or implicitly assume digital dither for requantization to be fixed point. Wannamaker [23] gave a thorough treatment of the mathematical theory of dither, but did not specifically treat floating point.

Dunay *et al* [24] remarked that although usually very small, the quantization error in floating point number representation is sometimes not negligible, and proposed a uniform or triangular-shaped dither for the mantissa with the same exponent as the current number. The amplitude of the dither then is directly related to the amplitude of the audio signal, and this argued Aldrich [25] against, by saying that such correlation leads to distortion and not noise. On the other hand, not matching the exponent values between dither and signal risk that the dither level is either too high or too low to adequately dither the signal. Aldrich concluded the impossibility to adequately dither floating point such as can be accomplished for fixed point, and the possibility that programmers trying to errantly apply fixed point dither principles to floating point can make the result worse than not using dither at all. A distinction also has to be made between dithering and truncating e.g. `double` to `float`, versus e.g. `float` to `short` which typically also involves a scaling (compare section 4.2.1).

Due to the outlined problems of purely theoretical approaches to floating point dither, attempts at solutions must be accompanied by audio quality

evaluations.

4.3. Audio quality

Being able to evaluate the audio quality of audio applications and audio subsystems is important in audio software development, to test and verify that no hidden processing, conversion or degradation of the audio signal occurs. From the review in this paper, such degradation could for example be caused by audio sample losses due to high CPU load when processing denormalized numbers, fixed point integer to floating point conversions and vice versa, dither issues for floating point numbers, and possible sample rate conversion by a kernel mode audio mixer software component like in section 4.1.2. These types of audio quality could be evaluated from some different points of view.

4.3.1. Bit transparency

Testing for bit transparency could be done by inserting a known bit stream into a well defined point in an audio software signal path, and then at an equally well defined point extract the bit stream. A bitwise comparison not showing bit identity would then call for further objective measurements and perceptual evaluations. In the literature review in this study, reports on this type of verification was not found.

4.3.2. Objective measurements

In a computer audio system context, issues concerning objective measurements like frequency response and total harmonic distortion were addressed in AES-6id-2006 [13]. However, being restricted to Windows, an extension of this document to also include Mac OS X and Linux would be useful, and the document does not either specifically consider various audio software signal paths resulting from the use of different native audio APIs.

4.3.3. Perceptual evaluation

Objectively measured audio quality degradations might or might not be perceived by a human listener depending on e.g. the order of magnitude of the degradation [26]. Berg [27] expressed the importance of perceptual evaluation of various tools in audio work. One study was found in the literature review, by Muheim [22], in which the ITU-R BS.1387 PEAQ (Perceptual Evaluation of Audio Quality) method [28], which at that time was the current version of the standard, was used to measure the perceptual effect of a sample loss concealment algorithm in the implementation of a computer audio system. An interesting verification of the PEAQ results would have been to compare them with the comparable measures from a subjective listening test using the ITU-R BS.1116-1 [29] method in which a group of listeners assess the perceived audio quality and the mean value of their given grades is calculated as a resulting quality measure. This was however not done in Muheim's study, and no other reports were found in the literature on perceptual evaluations of computer audio systems.

4.4. Questions for audio software development

The research review conducted in this study was done in order to identify questions that need careful attention and that are still unanswered in the literature. A map of questions are presented below for future research activities, consisting of further literature studies and experiments with software prototypes. These questions may in particular form a foundation for experimental evaluations of audio quality in audio software.

What is the exact audio software signal path, including all user and kernel mode software components, through an audio subsystem using a specific native audio API?
Initial and in some cases simplified overviews are presented in the literature, but details remain to make exhaustive descriptions.

How can such an audio software signal path be unambiguously elucidated and visualized?
An efficient way to achieve this is of utmost importance as a basis for further studies and experimentation.

What is the internal audio sample format of each user and kernel mode software component in such an audio software signal path?
This is important information to know in order to build an understanding of the possible format conversions involved in the audio software signal path.

How can a level diagram for an audio subsystem, in particular the audio software signal path, be established?

The literature mentions the difficulty of this area and the lack of indication of signal level in the audio software signal path, but no solution is proposed.

How should the conversion of audio sample values, in various combinations of bit sizes, from fixed point integer numbers to floating point numbers and back again be performed?
Even if such conversions should be kept to a minimum in order to maintain audio quality, they can hardly be avoided altogether. Therefore a clear understanding of the consequences of different alternatives is required.

Is the performance of an audio application improved by eliminating denormalized floating point audio sample values by replacing them with the value zero?
Not eliminating them is reported as a cause of possible high CPU load. These values are several hundred dB below floating point "0 dB full scale" and their elimination should therefore be without an audible effect. As an extreme precaution though, this inaudibility could be experimentally verified, but no reports on this were found.

Should dither be applied or not in the context of floating point audio sample representation, and if applied, how should it be implemented?
There are theoretical treatments on the problems with floating point dither, but no simple solutions. Perceptual evaluations are presumably needed to resolve the issue, which would involve extremely demanding listening tests.

Does the use of floating point dither with exponent values matching signal exponent values result in audible distortion?
This type of dither is proposed in the literature, but no objective measurements or perceptual evaluations of its use were found.

How should the audio quality of an audio subsystem, in particular the audio software signal path, be evaluated in a rational way?
There are treatments on objective measurements on entire computer audio systems in the literature, but no reports that include explicit considerations of the audio software signal path resulting from the use of a specific native audio API.

What is the audio quality of each user mode and kernel mode software component in an audio software signal path?
No reports on this were found in the literature whatsoever. It is still of great importance however in order to establish a clear understanding of the resulting overall audio quality of the audio subsystem as a whole.

5. DISCUSSION

A systematic approach to audio software development from an audio quality perspective is important. The review presented in this paper shows that such an approach as best is at its very beginning and that there is a need for deeper and more thorough work. Jones *et al* [12] noted that much computer audio development has been by designers trained in areas other than audio. A personal computer can further be said to constitute a consumer device, as remarked by Ternström [30]. In this perspective lies the challenge of using personal computers for professional audio production and research work, since a "consumer device PC" is mainly designed for general usability in a wide range of areas rather than according to requirements for use in scientific or professional high-quality audio work. Several important questions are waiting to be addressed by the communities of researchers and software developers.

5.1. Limitations

At this point no experimental evaluations of audio quality were made, which is a limitation in this study. Priority was given to contributing to the understanding of software issues in computer audio systems based on a literature review. Another limitation from an academic generalizability perspective is the *moving target* character of audio subsystem software of current operating system versions. However, the progress of change is not faster than that a solid understanding can be established as a basis for studies of future releases, and without which audio quality issues in computer audio systems remain largly intangible to audio researchers.

5.2. Validity and reliability

Although the problem partly was identified by anec-

dotal experiences, it may be considered as valid as it is indicated by some findings in the literature. The research review was based on literature found in recognized databases and can therefore be said to possess a sufficient level of reliability. The observation that the questions resulting from this study still remain unanswered in the literature provides a claim for their validity.

5.3. Conclusions

The audio software signal path through an audio subsystem in a computer audio system can be quite complex and hard to unambiguously elucidate, and can involve obscure stages of conversions and processing, e.g. fixed point-floating point, dithering and mixing software components. Audio software development terminology is imprecise, and there is a need for a systematic approach with an audio quality perspective.

The results reported in this paper point to the need of addressing the following type of questions: How should requirements specifications be made for audio software so that significant audio quality aspects are handled? How should audio software be verified with regard to given requirements? How can relevant measurements and metrics for audio software quality be defined? Can an audio software quality model be devised, based on existing general software quality models?

5.4. Further work

A natural direction of further work following this literature based study is to perform experimental evaluations of audio quality with software prototypes, based on the resulting questions presented in this paper. The development of some form of multiplatform software tool to automate the elucidation of audio software signal paths would be a helpful aid in efficiently dealing with some of the questions.

As a brief remark on another direction, open source studies of ALSA could provide a deeper understanding of native audio API internals.

Finally, an interesting step in further systematic studies of audio software development from an audio quality perspective would be to incorporate theories, methods and models from research in software engineering into the field – specifically software metrics and software quality – in an attempt to adapt and apply these in an audio technology context with a focus on audio software.

6. ACKNOWLEDGEMENTS

The authors wish to thank Norrbottens forskningsråd, Sweden, for financial support during these studies.

7. REFERENCES

[1] Kelly, M. (2006) Using game-audio tools to build audio research applications. *J. Audio Eng. Soc.* Vol.**54**, No.11, pp. 1102-1108.

[2] Royce, W. (1970) Managing the development of large software systems. In *Proceedings of the IEEE Western electronic show and convention (WESCON)*, 25-28 Aug 1970, Los Angeles, USA, pp. 1-9.

[3] Wohlin, C. (2005) *Introduktion till programvaruutveckling.* Studentlitteratur, Lund. (In Swedish)

[4] Boehm, B., Brown, J. and Lipow, M. (1976) Quantitative evaluation of software quality. In *Proceedings of the 2nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 13-15 Oct 1976, San Francisco, USA, pp. 592-605.

[5] Ekeroot, J. (2007) *Audio software development – an audio quality perspective.* MSc thesis, Luleå University of Technology, Luleå.

[6] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B. and Wesslén, A. (2000) *Experimentation in software engineering – an introduction.* Kluwer Academic Publishers, Norwell, Massachusetts.

[7] Fenton, N. and Pfleeger, S. (1996) *Software metrics – a rigorous and practical approach.* International Thomson Computer Press, London.

[8] Kan, S. (2003) *Metrics and models in software quality engineering*, 2nd ed. Addison-Wesley, Boston, Massachusetts.

[9] Tanenbaum, A. and Woodhull, A. (1997) *Operating systems: Design and implementation*, 2nd ed. Prentice-Hall, Upper Saddle River, New Jersey.

[10] Twelve Tone Systems, Inc. (2007) *Future of professional audio on Windows*. White paper.
URL: http://www.cakewalk.com/DevXchange/audio_i.asp

[11] Patterson, D. and Hennessy, J. (1998) *Computer organization and design – the hardware/software interface*, 2nd ed. Morgan Kaufmann Publishers, San Francisco.

[12] Jones, W., Wolfe, M., Tanner Jr., T. and Dinu, D. (2003) Testing challenges in personal computer audio devices. Presented at *AES 114th Convention, Amsterdam*. Preprint 5814.

[13] AES-6id-2006 (2006) *AES information document for digital audio – Personal computer audio quality measurements*. Audio Engineering Society, New York.

[14] Ekeroot, J. (2005) Audio in computers – a professional audio software perspective. Presented at *22nd Nordic Sound Symposium, Bolkesjø, Norway*.

[15] Goldberg, D. (1991) What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, Vol.**23**, No.1, pp. 5-48.

[16] Knuth, D. (1981) *The art of computer programming, Vol.2: Seminumerical algorithms*, 2nd ed. Addison-Wesley, Reading, Massachusetts.

[17] ANSI/IEEE Std 754-1985 (1985) *IEEE Standard for binary floating-point arithmetic*. IEEE Press, New York.

[18] Schwarz, E., Schmookler, M. and Trong, S. (2003) Hardware implementations of denormalized numbers. In *Proceedings of the 16th IEEE Symposium on computer arithmetic (ARITH-16'03)*, 15-18 Jun 2003, Washington, USA.

[19] De Soras, L. (2005) *Denormal numbers in floating point signal processing applications*.
URL: http://ldesoras.free.fr/doc/articles/denormal-en.pdf

[20] De Castro Lopo, E. (2006) *libsndfile – a C library for reading and writing sound files*.
URL: http://www.mega-nerd.com/libsndfile/FAQ.html#Q010

[21] Zölzer, U. (1997) *Digital audio signal processing*. John Wiley & Sons, Chichester, West Sussex.

[22] Muheim, M. (2003) *Design and implementation of a commodity audio system*. PhD thesis, Swiss Federal Institute of Technology, Zürich.

[23] Wannamaker, R. (1997) *The theory of dithered quantization*. PhD thesis, University of Waterloo, Waterloo.

[24] Dunay, R., Kollár, I. and Widrow, B. (1998) Dithering for floating-point number representation. In *Proceedings of the 1st International on-line workshop on dithering in measurement*, Mar 1998, Prague, The Czech Republic. Czech Technical University.

[25] Aldrich, N. (2005) *Exploring dither in floating-point systems*.
URL: http://www.cadenzarecording.com/images/floatingdither.pdf

[26] Staff Technical Writer (2007) Can you really hear it? Psychoacoustics in action. *J. Audio Eng. Soc.* Vol.**55**, No.1/2, pp. 75-79.

[27] Berg, J. (2002) *Systematic evaluation of perceived spatial quality in surround sound systems*. PhD thesis, Luleå University of Technology, Luleå.

[28] ITU-R (1998) *Recommendation BS.1387, Method for objective measurements of perceived audio quality*. International Telecommunication Union Radiocommunication Assembly.

[29] ITU-R (1997) *Recommendation BS.1116-1, Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems*. International Telecommunication Union Radiocommunication Assembly.

[30] Ternström, S. (2007) *Using personal computers for acoustic analysis in the voice laboratory*. White paper, Royal Institute of Technology, Stockholm.
URL: http://www.speech.kth.se/voice/white/WhitePaperUsingPCs.pdf