August 2022

MEASURING AUDIO SMOOTHNESS

Richard Zhang

Colin Downs-Razouk

MEASURING AUDIO SMOOTHNESS

ABSTRACT

A computing device (e.g., a smartphone, a laptop computer, a tablet computer, a smartwatch, etc.) may determine audio smoothness (e.g., an aspect of audio quality) of an application session based on an audio smoothness metric that measures the number of bytes of an audio sample that arrive at audio output device (e.g., a headset, an earpiece, a speaker, etc.) on time. For example, the computing device may determine the audio smoothness metric based on the number of buffer overruns (e.g., an anomaly that occurs when a program writes data to a buffer at a faster speed than the data is being read from the buffer) and buffer underruns (e.g., an anomaly that occurs when a program writes data to a buffer at a lower speed than the data is being read from the buffer). In some examples, a computing system may parse the data collected by the computing device for debugging purposes (e.g., to generate one or more dashboards providing insight into which applications are experiencing regressions in audio quality and the causes of the audio problems).

DESCRIPTION

FIG. 1 below is a conceptual diagram illustrating a computing device 100 and a computing system 102. As shown in FIG. 1, computing device 100 may include one or more processors 104, one or more audio output devices 106, and one or more storage devices 108. Storage devices 108 may store an operating system 110 (“OS 110”), one or more applications 112, audio hardware abstraction layer 114 (“audio HAL 114”), audio smoothness module 116, and audio quality data repository 118.

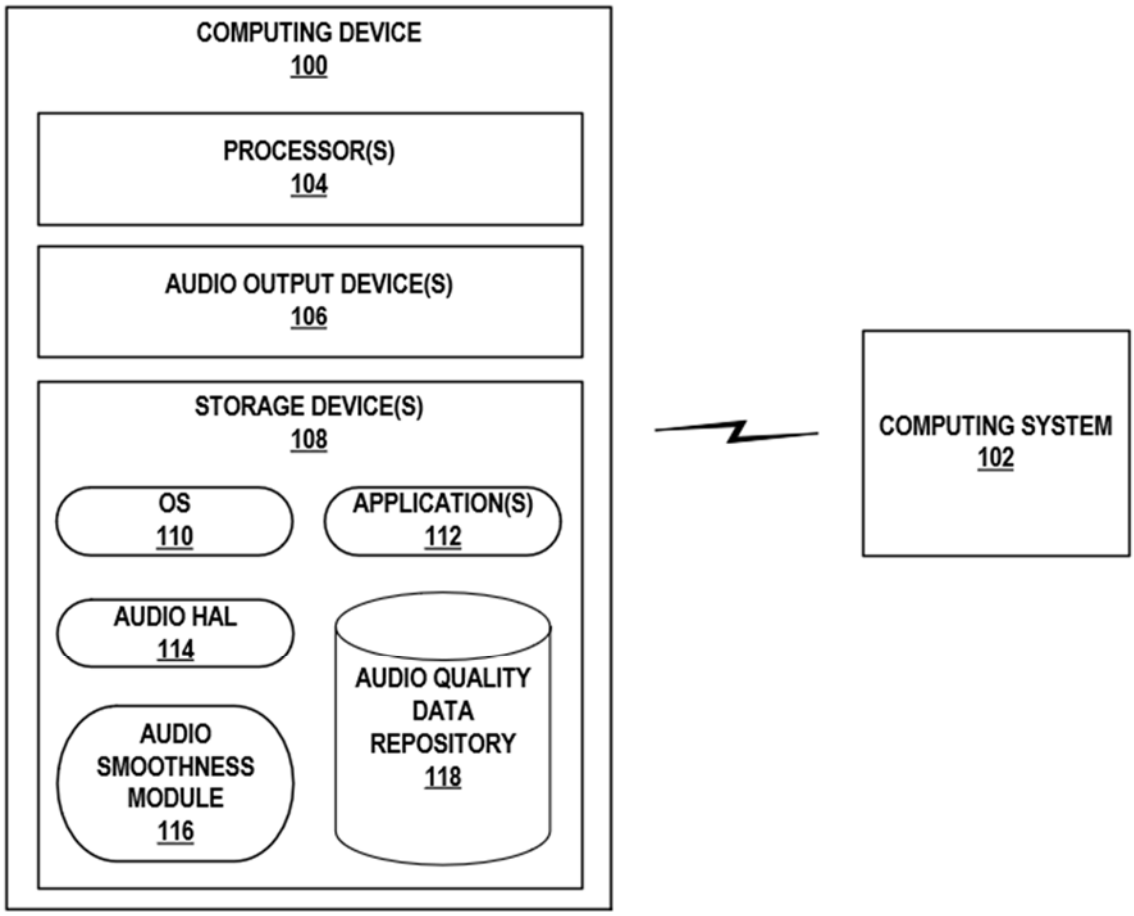

FIG. 1

In the example of FIG. 1, computing device 100 may be a wearable computing device, a mobile computing device, or any other computing device capable of outputting audio via audio output device 106, such as speakers which may or may not be integrated within computing device 100. Computing device 100 may be a mobile phone, such as a smartphone. However, computing device 100 may also be any other type of computing device such as a camera device, a tablet computer, a personal digital assistant (PDA), a smart speaker, a laptop computer, a desktop computer, a gaming system, a media player, an e-book reader, a television platform, an automobile navigation system, or a wearable computing device (e.g., a computerized watch, computerized eyewear, computerized ring, computerized clothing, etc.).

Processors 104 may implement functionality and/or execute instructions associated with computing device 100. Examples of processors 104 include one or more of an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), an application processor, a display controller, an auxiliary processor, a central processing unit (CPU), a graphics processing unit (GPU), one or more sensor hubs, and any other hardware configure to function as a processor, a processing unit, or a processing device. In some examples, processors 104 may represent a system on a chip (SoC) that includes an integrated circuit for implementing one or more of the above referenced examples of processors 104, along with supporting memory and/or storage, and possibly various interfaces, modems, etc. as a single package.

Storage devices 108 may include one or more computer-readable storage media. For example, storage devices 108 may be configured for long-term, as well as short-term storage of information, such as instructions, data, or other information used by computing device 100. In some examples, storage devices 108 may include non-volatile storage elements. Examples of such non-volatile storage elements include magnetic hard disks, optical discs, solid state discs, and/or the like. In other examples, in place of, or in addition to the non-volatile storage elements, storage devices 108 may include one or more so-called “temporary” memory devices, meaning that a primary purpose of these devices may not be long-term data storage. For example, the devices may comprise volatile memory devices, meaning that the devices may not maintain stored contents when the devices are not receiving power. Examples of volatile memory devices include random-access memories (RAM), dynamic random-access memories (DRAM), static random-access memories (SRAM), etc.

Computing device 100 may, as shown in the example of FIG. 1, include OS 110 that provides an execution environment for one or more applications, such as applications 112. OS

110 may represent a multi-threaded operating system or a single-threaded operating system with which applications 112 may interface to access hardware (e.g., audio output devices 106) of computing device 100. OS 110 may include a kernel that facilitates access to the underlying hardware of computing device 100, where kernel may present a number of different interfaces (e.g., application programmer interfaces (APIs)), such as audio HAL 114, that applications 112 may invoke to access the underlying hardware of computing device 100.

Computing device 100 may output audio via audio output devices 106. Examples of audio output devices 106 include speakers, headsets, earpieces, etc. Computing device 100 may include an audio architecture (e.g., audio frameworks) that processes an audio signal to enable audio output devices 106 to output audio. The audio architecture may include, but is not limited to, OS 110 and audio HAL 114.

The audio architecture may perform a number of operations to support a variety of hardware (e.g., High-Definition Multimedia Interfaces (HDMIs), earpieces, headsets, speakers, microphones, Bluetooth® devices, etc.) and software (e.g., media players/recorders, Voice over Internet Protocol (VoIP) applications, session initiation protocol (SIP) applications, etc.). For example, audio HAL 114 may connect higher-level, audio-specific framework APIs to the underlying audio driver and hardware of computing device 100.

The audio architecture of computing device 100 may include one or more shared memory audio buffers ("buffers") for communicating between devices or processes. The buffers may be areas of memory set aside to hold data (e.g., bytes of an audio signal) while moving data from section of a program to another or between programs. In some examples, the buffers may experience buffer overrun ("overrun") and/or buffer underrun ("underrun").

Overrun may refer to when an audio API (e.g., of applications 112), while writing data to a buffer, flows over the buffer's boundary and overwrites adjacent memory locations. That is, overrun may occur when a program writes data to a buffer at a faster speed than the data is being read from the buffer. This may result in erratic program behavior, such as audio crackling. Conversely, an underrun may refer to when an audio API writes data to a buffer at a lower speed than the data is being read from the buffer. An underrun may require the audio architecture to pause processing of an audio signal while the buffer refills, resulting in audio skips.

Because various implementations of audio HALs exist, different audio HALs may experience audio quality issues due to overruns and underruns to varying degrees. Further, it may be difficult to measure the effect of audio HAL implementation on audio quality. For example, measuring the smoothness of audio may require a human to manually listen for audio glitches (e.g., crackles, skips, etc.) which may be impractical or inaccurate.

In accordance with techniques of this disclosure, audio smoothness module 116 may determine audio smoothness (e.g., an aspect of audio quality) based on an audio smoothness metric that measures the number of bytes of an audio sample that arrive at audio output device 106 on time. In some examples, audio smoothness module 116 may determine the audio smoothness metric based on the percentage of audio HAL buffer writes that do not overrun or underrun. Audio smoothness module 116 may store the audio smoothness results and other data in audio quality data repository 118. Computing system 102 may parse the data in audio quality data repository 118 for debugging purposes (e.g., to generate one or more dashboards providing insight into which of applications 112 are experiencing overruns and/or underruns, the frequency of overruns and/or underruns, etc.).

As noted above, audio HAL 114 may write to a buffer during audio signal processing. Audio smoothness module 116 may detect overruns and underruns while audio HAL is writing to the buffer. For example, audio smoothness module 116 may monitor writes and reads to the buffer to determine if bytes are being dropped (which may be indicative of overrun) or whether the audio signal includes a gap filled with zero values (which may be indicative of underrun). Audio smoothness module 116 may then determine an audio smoothness metric based on the total number of overruns and underruns. For example, audio smoothness module 116 may determine the audio smoothness metric using the following equation:

$$S=1-o+uw$$

where *S* represents the audio smoothness metric, *o* represents the total number of overruns, *u* represents the total number of underruns, and *w* represents the total number of writes. A relatively large audio smoothness metric, such as a value close to 1, may indicate an audio signal with few overruns and underruns (and thus few audio crackles and skips) and thus good audio quality. On the other hand, a relatively low audio smoothness metric may indicate an audio signal with many overruns and underruns (and thus many audio crackles and skips) and thus bad audio quality. In some examples, audio smoothness module 116 may measure the number of overruns and underruns that happen in a time range block (e.g., a 1-minute window) in order to determine if specific periods of an application session have poor audio quality.

The data for overruns, underruns, and writes may pertain to a specific application session. Audio smoothness module 116 may collect the overrun, underrun, and write data and write the data in audio quality data repository 118 so that they can be parsed by any component that has access. For instance, computing system 102 may parse the data in audio quality data repository 118 when collecting metrics during benchmark testing and runtime.

For benchmark testing, audio smoothness module 116 may query for overrun, underrun, and write data at the end of a test run. Computing device 100 may then upload the data (as well as data derived therefrom, such as the audio smoothness metric) to computing system 102. For runtime metrics collection, audio smoothness module 116 may periodically (e.g., every 5 minutes) query for overrun, underrun, and write data. Audio smoothness module 116 may store the data in audio quality data repository 118. When an application session is finished, audio smoothness module 116 may query one more time and combine the queried data with what is stored in audio quality data repository 118.

Computing device 100 may locally analyze the data stored in audio quality data repository 118. Additionally or alternatively, computing system 102 may analyze the data stored in audio quality data repository 118. Computing device 100 may upload all the data for an application session to computing system 102 for analysis. In some examples, computing device 100 may periodically upload collected data to computing system 102 (in addition to uploading the data to computing system 102 after the application session is finished). In some examples, computing device 100 may upload collected data to computing system 102 after the end of an application session.

Computing device 100 and/or computing system 102 may generate one or more graphical user interfaces (GUIs) that indicate audio quality based on the data collected by audio smoothness module 116. In some examples, the dashboards may indicate audio smoothness metrics for specific applications, for the server, at specific times, etc. The dashboards may help with detection of bugs and regressions. For example, a significant negative change in any of the metrics included in the dashboards may indicate a concerning decrease in audio quality. If a specific application has poor audio smoothness (e.g., based on a low audio smoothness metric),

but other dashboards do not show regression in the server overall, then computing system 102 may notify the developer of the application of a potential bug with the application. If the regression is happening to the server as a whole, then computing system 102 may notify personnel responsible for maintaining the server of the audio quality issue.

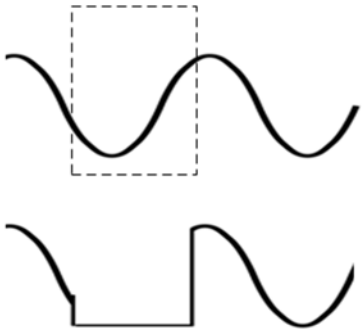

FIG. 2B

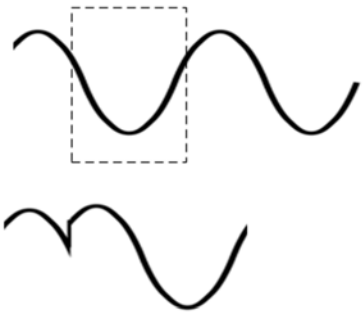

FIG. 2A

FIG. 2A is a conceptual diagram of an overrun. As shown in FIG. 2A, a portion of bytes of an audio signal (i.e., the portion of the audio signal within the dashed box) has been dropped due to overrun. Consequently, the portion of the audio signal is missing, which may result in audio crackling. FIG. 2B is a conceptual diagram of an underrun. As shown in FIG. 2B, a portion of bytes of an audio signal (i.e., the portion of the audio signal within the dashed box) has been filled with zero values as a result of underrun. Consequently, an audio skip may result.

As discussed above, audio smoothness module 116 may collect data relating to overruns and underruns. Audio smoothness module 116 may store the data in audio quality data repository 118. Computing device 100 and/or computing system 102 may parse the data in audio quality data repository 118 to generate dashboards that may provide insight into bugs and regressions affecting audio quality.

It is noted that the techniques of this disclosure may be combined with any other suitable technique or combination of techniques. As one example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2009/0113303A1. In another example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2021/0064333A1. In yet another example, the techniques of this disclosure may be combined with the techniques described in U.S. Patent Application Publication No. 2017/0169834A1. In yet another example, the techniques of this disclosure may be combined with the techniques described in "Overview of the audio system used for playing in-game sounds," Epic Games, April 26, 2022. In yet another example, the techniques of this disclosure may be combined with the techniques described in "Audio Terminology," Android, April 26, 2022.