PERCEPTUAL AUDIO SOURCE CULLING FOR VIRTUAL ENVIRONMENTS

Ali Can Metan, Hüseyin Hacıhabiboğlu

Graduate School of Informatics,
Middle East Technical University
Ankara, TR-06800, Turkey
canmetan@engineer.com, hhuseyin@metu.edu.tr

ABSTRACT

Existing game engines and virtual reality software, use various techniques to render spatial audio. One such technique, binaural synthesis, is achieved through the use of head-related transfer functions, in conjunction with artificial reverberators. For virtual environments that embody a large number of concurrent sound sources, binaural synthesis will be computationally costly. The work presented in this paper aims to develop a methodology that improves overall performance by culling inaudible and perceptually less prominent sound sources in order to reduce performance implications. The proposed algorithm is benchmarked and compared with distance-based, volumetric culling methodology. A subjective evaluation of the perceptual performance of the proposed algorithm for acoustic scenes having different compositions is also provided.

1. INTRODUCTION

Virtual environments create the perception of being physically present in a non-physical world via the presentation of synthetic audiovisual stimuli. With today's technology, creation of vivid environments require significant computational power. As such, optimisation of any existing process needs to make better use of the limited resources.

Sound scenes encountered in virtual environments may contain many sound sources. Spatialising all of these sources will impose a performance penalty for the underlying hardware. This problem is aggravated for computationally heavy spatialisation methods such as binaural synthesis which is widely used in VR systems.

In order to deal with the performance implications, most game engines and virtual environment software that are in use today, employ volumetric culling [1]. For volumetric culling, volumes (such as spheres or cubes) are defined for each sound source. Only those sources for which the listener is in the active volume, are rendered. While these methodologies are effective in reducing the amount of sound sources, perceived richness of the resulting scene may also be degraded as a result. Here, we define *auditory richness* as the perceived quantity of sound sources in an real or virtual acoustic scene.

There are also methodologies that incorporate various perceptual approaches. Some of these incorporate crossmodal effects such as culling sound sources that reside outside the view frustum of the listener [2]. This method is based on the assumption that audiovisual correspondence will have a major affect on the perception of sound sources. Other methods will cull or spatially cluster sound sources according to their predicted audibility [3][4].

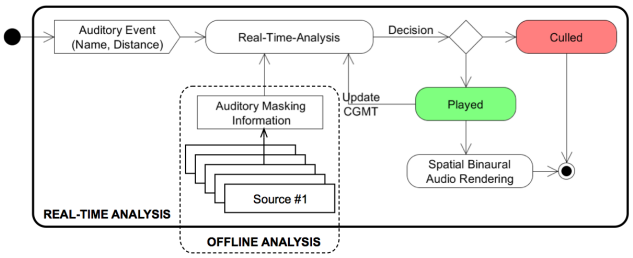

Figure 1: *Overall pipeline for the culling algorithm*

Majority of games and virtual environments include spatially separated, high-energy, broadband sound sources which may also be concurrently active. Explosions, gunshots, engine sounds and other sound sources may render each other inaudible due to auditory masking. If sources that are inaudible can be identified and eliminated from the rendering pipeline, computational savings may be made without degrading the overall quality of the rendered scene. The aim of this paper is to present an algorithm which can select a subset of all sources in the sound scene based on their audibility. The audibility is calculated using an existing model of simultaneous masking which also forms the basis of perceptual audio coding. The sound sources are evaluated and selected according to their perceptual salience due to monaural masking model used in MPEG-1 Layer I. The sound sources selected by using the proposed procedure can then be rendered without significant perceptual degradation of the richness of the auditory scene.

This paper is organised as follows. Sec. 2 presents the proposed algorithm. A performance analysis of the proposed algorithm is given in Sec. 3. The results of a subjective evaluation comparing the proposed culling method with volumetric culling is presented in Sec. 4. Sec. 5 concludes the paper.

2. PERCEPTUAL SOURCE CULLING

The algorithm proposed in this paper, consists of two parts: offline analysis and real time analysis. Extraction of the masking properties of audio sources contained within a scene, is carried out during the offline analysis. During real time analysis, information extracted from the previous stage is used to determine whether a given audio source will be rendered or not. Fig. 1 summarises the algorithm.

2.1. Offline analysis

Offline analysis is performed prior to rendering the actual scene on sound files which may be used in a scene. In this step, masking thresholds of sound source signals are calculated and stored in persistent memory. The model used for calculating the masking thresholds is the MPEG-1 Layer I psychoacoustical model [5]. This application is similar to how the same model is applied in sound synthesis [6]. Since the masking thresholds obtained this way are additive, they need not be calculated at run-time. Details of offline analysis is explained in the remaining parts of this section.

2.1.1. MPEG-1 Audio Layer I Masking Model

MPEG-1 Audio Layer 1 is an audio coding standard incorporating a psychoacoustical model to reduce the bitrate [5]. In the proposed method, this psychoacoustical model is used to determine whether a requested audio event will be audible inside the existing scene or not. The model calculates the masking thresholds of individual sound sources by estimating their tonal and non-tonal components [7]. After relevant maskers have been identified among these components, local masking thresholds are calculated and stored. Details of how the auditory masking thresholds are calculated, is explained briefly below.

As a first step, samples from the audio signal are divided into frames of 512 samples and each individual frame goes under psychoacoustic evaluation. Two separate frequency-domain representations are used. The first representation uses a 32-channel polyphase filter bank for emulating the frequency selectivity of the auditory periphery. The second representation uses a 512-point FFT to determine tonal and noise maskers. Each of the 32 channels generate frames of size 12 resulting in a total of 384 samples. Appropriate time shifting is applied to correct the time delay induced by the filter bank. The frequency resolution of the FFT for a sampling rate of $F_s = 44.1$ kHz is 86.13 Hz. An overlapping Hann window is used in the calculations for obtaining an estimate of the power spectrum, $X(k)$, $k = 0 \dots N/2$ of the frame.

Then, the sound pressure level (SPL) in subband $l = 0 \dots 31$ is computed with respect to a reference of 96 dB. The sound pressure level, $L_{sb}(l)$, is computed for every subband l and stored.

MPEG-1 Audio Layer I psychoacoustical model separates the tonal and noise-like components of the audio signal. The reason for this is the different spreading characteristics of simultaneous masking by these different types of maskers.

The tonal components $X(k)$, are identified based on the local maxima which are determined as the peaks that satisfy the following condition:

$$|X(k)| > |X(k-1)| \qquad \text{and} \qquad |X(k)| >= X(k+1)$$

Subsequently, for each critical band, the remaining noise-like components are summed into a single non-tonal masker component. A local peak is added to a list of tonal maskers if:

$$|X(k)| - |X(k+j)| >= 7 \text{ dB},$$

where j is chosen differently for different frequency bands such that:

$$\begin{array}{cc} j = -2, +2 & \text{for} \quad 2 < k < 63 \\ j = -3, -2, +2, +3 & \text{for} \quad 63 <= k < 127 \\ j = -6, ..., -2, +2, ..., +6 & \text{for} \quad 127 <= k <= 250 \end{array}$$

The remaining spectral lines are identified as non-tonal components.

Less powerful maskers are determined and eliminated in order to obtain a global masked threshold. Two conditions are used for this purpose: 1) tonal or non-tonal component which generates masking thresholds that are below the absolute threshold of hearing are eliminated as these components will not be audible [5], and 2) less powerful tonal components within the distance of less than 0.5 Bark from a powerful tonal masker are eliminated. This way, components with the highest power are kept in the list of tonal components and others are eliminated.

The global masking threshold for each frequency index is derived from the individual masking thresholds of calculated tonal and non-tonal maskers as well as hearing threshold in quiet.

After every frame has been processed, masking thresholds of the 32 subbands are stored in a binary file. For a three second audio signal that has a sampling rate of 44.1 kHz, output will have $\lfloor (44100 * 3)/384 \rfloor = 344$ frames because of overlapping windowing. This results in a file size of 86 KB for the storage of this auxiliary information. This is approximately one fourth of the original sound file. With contemporary hardware specifications, storing this additional information on volatile memory would not cause any issues.

Offline analysis is concluded by storing the masking thresholds for each frame. Global masking threshold of the entire scene will be calculated during the real time analysis.

2.2. Infeasibility of Precalculated Decision Making

One could argue that precalculating and storing global masking threshold of a *dynamic* scene is also a feasible option. For a given scene, masking can only occur after at least two concurrent sound sources are played, which is what is of interest. While at least two sources are necessary for masking to occur, there is no theoretical upper limit on the number of sound sources that can be played at a given time. Even if such a limit is imposed on the number of distinct sound files/sources in order to limit the number of different combinations, since the same sound file can be played infinitely many times, infinitely many possible combinations exist for a given scene. Also, each sound event would have a different onset making precalculation infeasible.

Let there be a scene with 30 sound files that are 4 seconds long each. There are $\lfloor (44100/384) * 4 \rfloor = 459$ frames per sound file. If there are only two unique sound files played at a time, there would be $\binom{30}{2} * (459 + 459 - 1) = 398\,895$ possible combinations just for two overlapping sound sources. In this calculation, onsets of each sound event are constrained to align with the starting pointer of a frame. If there are only three unique sound files played at a time, there would be $\binom{30}{3} * (459 + 459 - 1)^2 = 6\,828\,018\,680$ combinations. This analysis excludes the possibility that same sound source (e.g. a gunshot sound) can be activated more than once in the given duration.

The possible combinations are increasing exponentially and the memory required to store all number of concurrent sound sources becomes infeasible with the contemporary hardware.

2.3. Real-Time Analysis

After the offline analysis is performed on the audio sources contained within the scene, and the masking data stored in memory, real-time analysis can be performed during the program execution. The main purpose of the real-time analysis, is to calculate

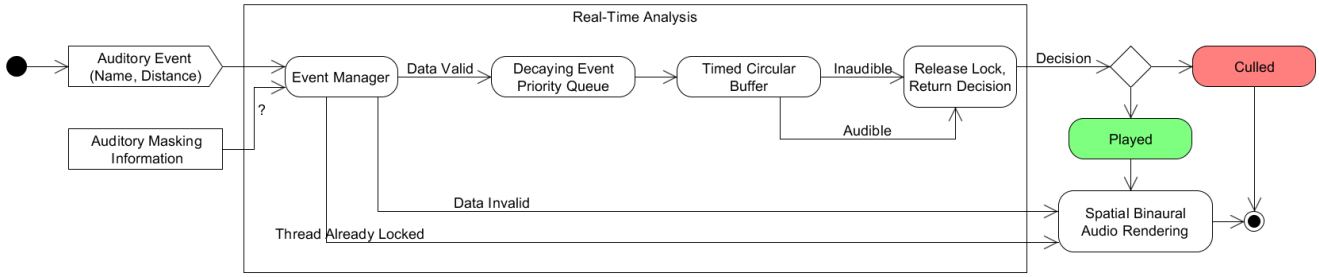

Figure 2: *Overall pipeline for the real time analysis*

the global masking threshold for the dynamic scene on the fly and identify whether a new sound event will be audible or not given the existing events already being rendered. Determining this allows culling inaudible sound sources, which in turn reduces the computational overhead due to rendering spatial audio. At the offline stage simultaneous masking thresholds are calculated for monaural listening conditions. However we assume that the global masking threshold, which is an attribute of monaural hearing is also an indicator of the perceptual salience for binaural hearing. Visual representation of the real time analysis, for handling an auditory event is shown in the Fig. 2.

Real-time analysis involves several stages: Firstly, when an auditory event is triggered, the event manager (EM) does the preliminary operations required to handle this event. EM then places or re-prioritizes this event in the decaying event priority queue (DEPQ) in order to reduce hard drive access delay. After the data is successfully stored in the heap, timed circular buffer (TCB) attenuates masking thresholds according to their distance and adds their masking values to the global masking threshold. Then, the *audibility ratio* of the individual audio source is calculated. If this ratio is larger than a selected threshold value, that audio source is culled.

2.3.1. Concept of Sound Events and Event Manager

For the proposed algorithm, the audio rendering component of the game engine needs to be encapsulated. For every spatial audio render request, sound events are generated, later to be handled by the sound event manager (EM). Each audio event includes:

- An **audio identifier** which denotes both the actual audio file path and the binary file which contains the masking information.
- **Distance** from the listener position

The event manager that was mentioned above has two responsibilities: 1) handling sound event requests, and 2) handling *end-of-audio events* received from the game engine. Event manager runs on its own thread with a single `mutex` that locks each time an event is received. Receiving end-of-audio events require the same `mutex` lock acquisition as receiving an auditory event. Hence they are synchronized, even if multiple threads call them simultaneously. One key difference between them is that, receiving an end-of-audio event is a blocking operation whereas sound event handling is not. Details of each operation are explained below.

Sound Event Handling: Whenever a sound event is triggered, the event manager tries to acquire the thread lock (via the only `mutex` it has). If it fails to acquire the lock, it simply returns without doing any operations and the audio will be rendered. If it acquires the lock, meaning that this is the only running real time analysis, it checks whether the masking information for the given audio signal exists or not. If there is no masking information present, audio will be automatically rendered, without doing any further operations. This could be the case, for example, for background music which should not be subject to culling or for sources which are marked as immutable by the game audio designer.

If the lock is acquired and the audio/masking data are valid, event manager proceeds with the remaining operations. After all the operations and calculations are carried out and the decision is made, the event manager releases the thread lock and returns the result to the calling thread. After that, if the audio is marked as *inaudible*, it will be culled and no further operations will be necessary. If the audio is marked as *audible*, game engine proceeds with spatial binaural audio rendering. A visual representation of the workflow is displayed in Fig. 3.

End-Of-Audio Event Handling: End-of-audio events are signals that indicate that a single audio file has finished playing, and the information of which audio file is finished is not conveyed. These events are required in the execution of the circular masking threshold which will be explained below.

When an *end-of-audio* event is received, lock must be acquired. After the program counter is in the critical section, event manager decrements the active audio count. If there is no active sound in the scene, timed circular buffer is notified and the lock is released. When all sound events have finished playing, TCB is purged. A visual representation of the workflow is displayed in Fig. 4.

2.3.2. Decaying Event Priority Queue

As explained above, auditory masking information for each sound is stored in persistent data storage such as HDDs or SSDs. Compared to the data that was allocated dynamically from heap or stack, accessing data is significantly slower from these resources. In order to counteract this effect, the retrieved auditory masking information is temporarily stored in a decaying event priority queue (DEPQ).

DEPQ is a dynamically allocated priority queue that stores a fixed amount of past auditory event information. Priorities for each event are set according to their arrival order. As the name suggests, priorities for each event are decremented every time a new event comes. When the queue is full and an event that is not stored in queue arrives, the event with the lowest priority (the least active event) is replaced with the newer one.

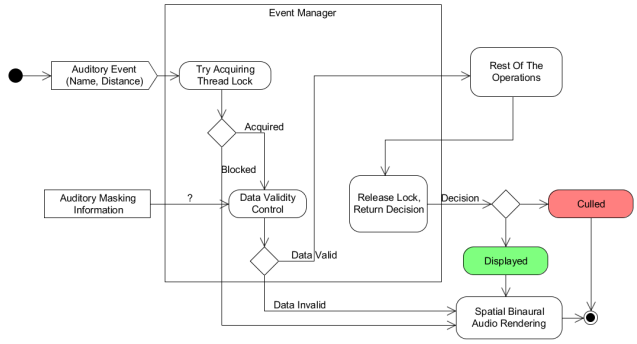

Figure 3: *Event manager workflow for handling auditory events*

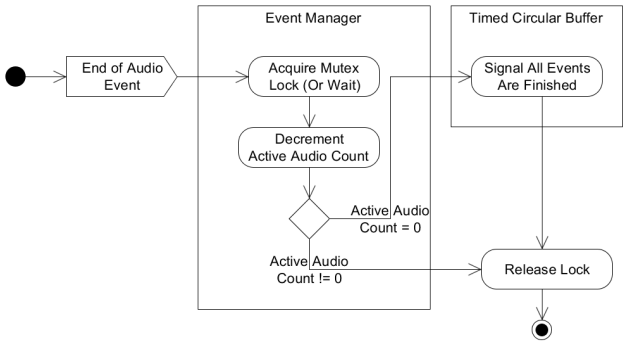

Figure 4: *Event manager workflow for handling end-of-audio events*

2.3.3. Timed Circular Buffer

Timed circular buffer (TCB) contains the global masking threshold and incorporates new masking values into it. All of the masking values come from binary files that are stored during the initial offline analysis stage. TCB also decides whether a given sound will be audible or not by calculating audibility level as a ratio of the number of audible frames to the number of inaudible frames of the audio signal to be played. TCB then returns its assessment to the event manager. The timed circular buffer is a variation of circular buffer with four additional variables besides the size of the buffer and the buffer itself:

1. *Start Index, inclusive*; Points to the first valid data of the buffer. Updated according to time, stored as unsigned integer.
2. *End Index, exclusive*; It points to the first empty index of the buffer. Stored as unsigned integer.
3. *Valid Index, exclusive*; This points to the end of previous sound data and index from which the array needs to be cleared from. One could also say it is this first index after the last valid data. Stored as unsigned integer.
4. *Audibility Percentage*; For a given audio and an existing scene, determines how much of the given auditory event will be audible.

Details of how each index works in conjunction with the algorithm is explained below:

MPEG-1 Layer I uses 12 samples per subband and there are 32 subbands as explained above. Size of the buffer, in the case that the buffer will store 5 s[1] of masking information, can be calculated as follows:

```
bufferSize = Size of buffer for 5 seconds
                        + 1 extra empty frame
           = floor(5000 / Frame Time) * 32 + (32)
           = floor(574.21875) * 32 + 32
           = 18400 Doubles
```

[1]Size of buffer is not constrained to 5 seconds and can be shorter or longer depending on user requirements.

where the duration of a single frame of audio is given as:

```
Frame Time = (Frame Size * 1000 ms) / Sampling Rate
                  = (384 * 1000 ms) / 44100 ≈ 8.7 ms
```

The start index is only modified by the actual time difference. It is updated every time a new auditory event arrives. Calculation of start index, when requested, is implemented as follows:

```
startIndex += ⌊ elapsedTime / frameDuration ⌋ / 32;
            startIndex = startIndex % circBfrSize;
```

Prior to calculating the global masking values for the dynamic scene, the individual masking values are attenuated according to distance using the inverse-square law.

$$\Delta L_p = 10 \log(\frac{r_1}{r_2})^2 \text{ dB}$$

While determining the audibility for a given audio event, a threshold value called *audibility ratio* is defined:

$$\textit{Audibility ratio} = \frac{\textit{number of audible frames}}{\textit{total number of frames}}$$

To determine whether a frame is audible or not, masking values are first compared with the current global masking values. If the given masking value is greater than the existing global masking threshold for any of the 32 subbands, that frame of the audio will be considered audible.

When handling masking values, TCB follows the following steps of execution:

1. Update starting frame according to time,
2. Determine estimated end index and clear deprecated data,
3. Attenuate masking values according to source distance,
4. Calculate the audibility ratio within the existing scene,
5. If audibility ratio is greater than a preset threshold, add these masking values to TCB and return.

3. PERFORMANCE ANALYSIS

The primary purpose of the proposed algorithm is to provide a performance advantage with minimum perceptual degradation. As such, the computational cost of real time analysis should be lower than synthesizing binaural audio. Performance of the proposed algorithm, as well as factors that affect analysis duration is explained below.

3.1. Algorithmic Complexity of Real-Time Analysis

Real time analysis consists of three components: event manager (EM), decaying event priority queue (DEPQ), and timed circular buffer (TCB). All operations of the event manager have a computational complexity of $O(1)$, and the algorithmic complexity depends on the other two components, DEPQ and TCB. In this section, we will assume that the size of the priority queue is m and the size of the masking value sequence is assumed to be n.

Decaying event priority queue (DEPQ) has two main functionalities; handling sound events and handling end-of-audio events. End of audio events are of complexity $O(1)$. Handling audio events on the other hand, not only requires interaction with the queue itself but also the auditory masking values in the file system. Firstly, if the size of the queue is m, searching through the queue has a complexity of $O(m)$. Secondly, fetching the masking values from the file system and placing them into the queue has a complexity of $O(n)$.

For the timed circular buffer (TCB), the sequence of input values is traversed twice. Once for attenuating the masking values according to distance and second time for adding the masking values to the global masking threshold. So the complexity of this operation is $O(n)$.

Three factors determine the computational cost of the analysis stage:

1. Length of the sound signal:
   (a) In the case that the file is not already stored in DEPQ, read time (excluding the seek time) is proportionate to the length of the offline analysis file.
   (b) Each frame coming from the analysis file are compared to the global masking threshold in the TCB to determine whether a given audio is audible or not. Hence number of frames affects the duration of the real-time analysis.
2. Type of persistent data storage: Retrieval from a physical drive takes longer in comparison with a solid-state drive.
3. Storage in DEPQ: If data is not stored in the program stack, time costs of seek time, data transfer rate, and rotational latency (for HDDs) are added to the time cost of finalising the analysis. [8]

A limiting factor in the applicability of the proposed method is the type of persistent data storage from which offline analysis data is retrieved. HDDs incur spin-up delays, seek time delays and slow data transfer rates compared to SSDs. In a system that highly depends on time, such issues may render the system useless due to hardware delays. If the end user were to use HDDs, increasing DEPQ size and storing the whole masking information into the queue would be a feasible option.

3.2. System Performance

An assessment of the performance of the proposed method is presented in this section. Time values listed here are hardware dependent. In order to smooth out peaks due to higher-priority operating system processes, the metrics presented in this section use an average of 10 runs.

The hardware used in the calculation of the reported values includes Intel $i5$-3570K CPU running at 3.40GHz, 8GBs of RAM, 120GBs of SSD, 500GBs of HDD and a GeForce GTX 670 graphics card.

3.2.1. Performance of Real Time Analysis

The time it takes to complete real time analysis is affected by how the pre-calculated data is accessed. Different storage media have different data transfer overheads which in turn affects the duration of real time analysis. This section describes the individual cost of the culling algorithm, including any delays associated with it but excluding the cost of the subsequent (binaural) rendering operations. Table 1 shows the average times to reach a decision, based on the conditions listed.

Different hardware architectures would have different results.

Table 1: *Average time required to complete real time analysis*

Input Size	Storage in DEPQ	Memory Type	Delta Time
1 Frame	Stored	RAM	≈ 0 ms
574 Frames	Stored	RAM	0.51 ms
1 Frame	Absent	HDD	15.626 ms
574 Frames	Absent	HDD	15.918 ms
1 Frame	Absent	SSD	≈ 0 ms
574 Frames	Absent	SSD	0.53 ms

3.2.2. Performance Gained by Culling a Single Audio Source

For a sound signal that was stored in DEPQ, average time it takes to render a 5 second audio is 0.51 milliseconds. During this time, a single CPU core is fully utilized. We can express the total cost of a program execution as:

$$CostOfExecution = Average\,CPU\,Utilization * Time\,In\,Milliseconds$$

Since audio rendering incurs a performance penalty in the beginning of each frame, over the time of execution, we can see whether this methodology is profitable or not. Since we are not considering the memory delay for retrieving the actual audio data from memory, we are not considering the retrieval of the masking values from memory either. For an audio signal that is already inside the DEPQ, and for audio data that was already stored in RAM, the performance metrics are listed in Table 2.

Table 2: *Execution cost comparison*

Input Size	Culling Assessment	Cost Of Execution
1 Frame	No Culling Algorithm	$9\text{ms} * \%0.44 = 3.96$
1 Frame	With Culling Algorithm	$9\text{ms} * \%0.46 = 4.14$
574 Frames	No Culling Algorithm	$5000\text{ms} * \%0.45 = 2250$
574 Frames	With Culling Algorithm	$5000\text{ms} * \%0.49 = 2450$

As shown in the table, the culling algorithm has 8% more performance overhead compared to the existing audio synthesis pipeline. However in the case that the audio will be culled, we will save 92% of the clock cycles. In other words, in order for the proposed algorithm to provide any computational gain, it should cull at least 92% of all the frames. One disadvantage of the culling algorithm, however, is that it will block a CPU core completely until the analysis is done.

At the cost of degrading the overall composition of the scene, a volumetric culling methodology would save %100 percent of the clock cycles instead of our %92 but does not guarantee the retention of perceptually most prominent sources. While the proposed methodology does not provide the best overall performance, it provides a good balance between perceived richness and performance as will be shown in the next section.

4. SUBJECTIVE EVALUATION

Audio produced with sound source culling, regardless of the methodology, is dependent on the user determined variables. Resulting auditory richness produced by the aforementioned methodologies, provide a crude approximation to the original scene. In order to find out which parameters provide sufficient perceived richness, a subjective evaluation was performed.

4.1. Chosen Test Methodology

The ITU-R Recommendation BS.1534, proposes proposes a subjective evaluation method called "MUlti Stimulus test with Hidden Reference and Anchor
(MUSHRA)" which is appropriate for assessing intermediate audio quality [9]. While this paper is not directly related to audio coding or quality, we use MUSHRA to test intermediate levels of auditory richness given a hidden reference and anchor.

4.2. Test Procedure

4.2.1. Presentation of Stimuli

In the MUSHRA test method, a high quality reference signal, a low quality anchor signal and other signals that fall in between them in terms of quality are evaluated [9]. In any given test, there is a single reference and a single anchor that the user is expected to find. Each prerecorded sound can be played as many times as the listener desires. Presentation of both the different experiments and the stimuli contained within the experiments are randomized. So not only each listener listens experiments in a different order but each listener faces a randomized presentation order of the stimuli. For the experiments, a MATLAB interface called MUSHRAM was used [10].

4.2.2. Grading

The whole test procedure is comprised of giving ratings to test signals which are displayed in random sequence. Listeners were instructed to score the presented samples in comparison with the explicitly provided reference signal. The scores that are given, can range between 0 and 100. Listeners were asked to find and rate the reference and the anchor signals as 100 and 0, respectively. For any of the remaining stimuli, listeners were asked to give their scores according to the perceived richness. They were also encouraged to listen to the available reference signal, prior to giving scores on the stimuli.

Table 3: *Contents of Test Scenes*

Experiment	Contents (ss = sound sources)
Impulsive Scene	13 impulsive ss.
Impulsive + Speech + Music	6 impulsive ss., 6 speech ss., 1 Music sound
Impulsive + Sound Effects + Music	6 sound effect ss., 6 impulsive ss., 1 Music sound
Sound Effects + Speech	7 speech ss., 6 sound effect

4.3. Experiment Details

The MUSHRA test applied for this paper is comprised of four different experiments. Each experiment involves a combination of different sound sources of different categories: impulsive sounds, music, static sound effects and speech signals. From these sound sources, we arranged four different psychoacoustical experiments, involving sounds under different categories. In each of these scenes, there were a total of 13 sound sources in the case of reference signals. The applied culling methodology served to reduce the number of sound sources at each rendered scene. Composition of each scene is given in the Table 3.

The test scenes were adjusted so that the maximum length of a stimulus does not exceed 2 s.

4.3.1. Selection of Sound Source Locations

Since both systems need to be tested with the same conditions, a scene that is appropriate to both culling methodologies was required. In order to achieve this, sound sources were distributed along the horizontal plane of the listener. Assuming one source will be directly in front of the listener, the axis needed to be divided into $180/(13-1) = 15$ degree segments. Listener was placed at the coordinate location $(700, 700)$, facing towards the negative x axis.

We used Unreal Engine 4 to test the proposed algorithm. In order to achieve a controllable volumetric culling methodology, each of the sound sources were placed at a fixed radius away from the listener. They were grouped into pairs and were placed at an equal distance and symmetrical angles with respect to the front direction of the listener. Coordinates of sound sources were calculated according to the formula:

$$X = r\cos(\theta) + 700\,(\text{cm})$$

$$Y = r\sin(\theta) + 700\,(\text{cm})$$

Top down view that demonstrates the positions of the sound sources is given in Fig. 5.

For example, a sound source with a culling radius of 200 cm will not be rendered when a listener is positioned 300 cm away from the source. This way of adjusting test scene enables precise control over which sound sources are going to be culled because of the spherical culling radii. Note that these sources were not triggered in the same time frame so that their onsets do not coincide.

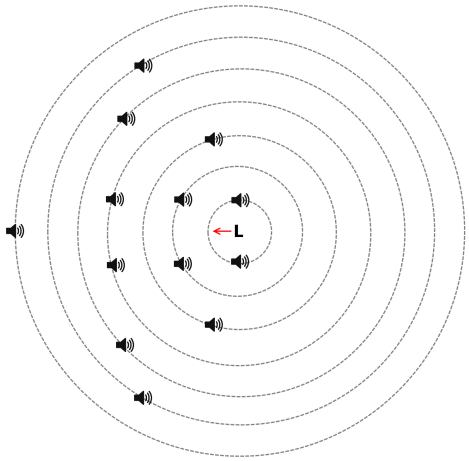

Figure 5: Positioning of the sound sources for the tested scenes

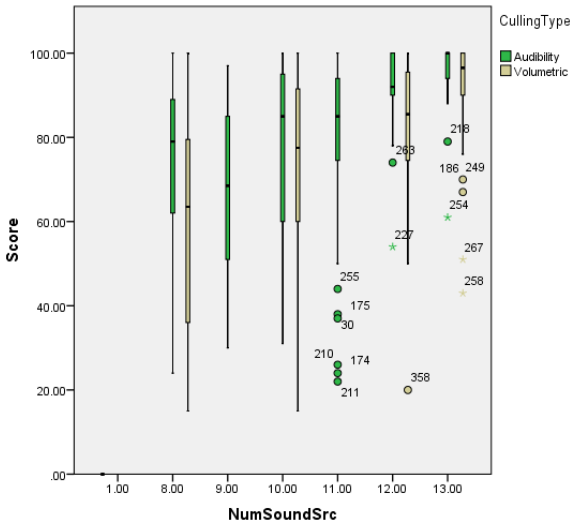

Figure 6: *Box plot of subjective scores from listening tests. Circles denote outliers and stars denote extreme values with their corresponding sample index*

4.3.2. Generation of Test Cases

There are nine stimuli in each experiment. The reference signal includes all 13 sounds. The anchor signal involves only a single sound source to achieve the lowest richness in a scene. The remaining signals are results of scenes with culling methodologies applied. For the perceptual culling methodology, various audibility ratios were tested. For volumetric culling methodology, various radii were tested. In total, there were 4 scenes with 9 culling settings making a total of 36 test cases.

At each run, among the 7 stimuli, 4 of them were produced with perceptual culling and 3 of them were produced with volumetric culling. For perceptual culling, %0, %10, %20 and %30 audibility ratios were used. For distance based culling, 650, 550 and 450 cm culling radii were used. Due to the configuration of the scene, chosen auditory stimuli and culling methodology, test cases that have 9 and 11 sound sources were only available to audibility based culling methodology.

4.4. Statistical Analysis

Eleven participants participated in the test (7 male, 4 female). Fig. 6 shows the results of the scores from the test. As can be observed, scenes that have 8, 10, 12 and 13 sources have results for both of the culling methodologies.

In order to see whether the proposed methodology has any advantage over volumetric culling for a given scene, responses given to scenes with the same number of sources but obtained using different culling strategies can be compared. Fig. 6

Independent-samples t-tests were performed to find out whether mean responses given to stimuli obtained using different culling methods are significantly different. Differences in mean values of subjective responses for 8, 10, and 12 sound source scenes using different culling methodologies is statistically significant at $\alpha = 0.05$ level with auditory culling outperforming volumetric culling for these cases. For the 8 source case, the difference between mean responses was statistically significant with $t(75) = 2.965$, $p = 0.004$ (Variances between groups is significant). For the 10 source case, the difference between mean responses was statistically significant with $t(152) = 2.767$, $p = 0.006$ (Variances between groups is significant). For the 12 source case, the difference between mean responses was statistically significant with $t(59.268) = 2.270$, $p = 0.027$ (Variances between groups is not significant)[2]. For the 13 sources case, the differences between mean responses were not significant. This result was expected as no source was culled through culling methodologies for this case.

5. CONCLUSIONS

A perceptual sound source culling methodology based on models of simultaneous masking was proposed in this paper. The algorithm uses a prioritisation approach based on the audibility of sound sources and those sources which will be less audible than the others are culled.

The algorithm was evaluated with respect to its computational as well as its perceptual performance. Performance evaluations showed that while the culling algorithm itself causes some computational overhead, it may still provide savings when it is realised that culled sources are not processed further for binaural spatialisation and that binaural processing has a higher computational overhead in general. The subjective evaluations involved a comparison of the proposed culling algorithm with volumetric culling where sources are culled based on their distance from the listener. It was observed that for the same number of sources culled using the auditory culling and volumetric culling methods, auditory culling provided higher scores in terms of auditory richness.

While the computational and perceptual performance of the proposed method for culling sources in more complex scenes including a higher number of sources remains to be investigated, the proposed culling algorithm provides a promising approach that

[2] In order to check for the normality assumption, Levene's test was used. The degrees of freedom of the test is modified in order to account for the cases where the assumption that homogeneity of variances does not hold and is called Welch's t-test.

could make it possible to render better spatial audio on low-end devices.

6. REFERENCES

[1] Richard Stevens and Dave Raybould, *Game audio implementation: A Practical Guide Using the Unreal Engine*, Focal Press, 2015.

[2] David Grelaud, Nicolas Bonneel, Michael Wimmer, Manuel Asselot, and George Drettakis, "Efficient and practical audio-visual rendering for games using crossmodal perception," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, 2009, pp. 177–182.

[3] Nicolas Tsingos, Emmanuel Gallo, and George Drettakis, "Perceptual audio rendering of complex virtual environments," in *ACM Transactions on Graphics (TOG)*. ACM, 2004, vol. 23, pp. 249–258.

[4] S Spencer Hooks and Nicolas R Tsingos, "Encoding and rendering of object based audio indicative of game audio content," Aug. 6 2013, US Patent App. 14/414,877.

[5] ISO / IEC, *Coding Of Moving Pictures And Associated Audio For Digital Storage Media At Up To About 1.5 Mbit/S Part 3 Audio*, June 1993.

[6] Mathieu Lagrange and Sylvain Marchand, "Real-time additive synthesis of sound by taking advantage of psychoacoustics," in *Proceedings of the Digital Audio Effects (DAFx01) Conference*, 2001, pp. 249–258.

[7] Davis Pan, "A tutorial on mpeg/audio compression," *IEEE multimedia*, , no. 2, pp. 60–74, 1995.

[8] Hamid D Taghirad and Ehsan Jamei, "Robust performance verification of adaptive robust controller for hard disk drives," *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 1, pp. 448–456, 2008.

[9] ITU-R, *Method for the Subjective Assessment of Intermediate Quality Level of Audio Systems*, bs.1534-3 edition, 10 2015.

[10] E Vincent, "Mushram: A matlab interface for mushra listening tests," *Online] http://www. elec. qmul. ac. uk/people/emmanuelv/mushram*, 2005.