REAL-TIME AURALISATION SYSTEM FOR VIRTUAL MICROPHONE POSITIONING

Tom Barker
DETI*, IEETA†
Universidade de Aveiro
Aveiro, Portugal
tommysb@gmail.com

Guilherme Campos
DETI*, IEETA†
Universidade de Aveiro
Aveiro, Portugal
guilherme.campos@ua.pt

Paulo Dias
DETI*, IEETA†
Universidade de Aveiro
Aveiro, Portugal
paulo.dias@ua.pt

José Vieira
DETI*, IEETA†
Universidade de Aveiro
Aveiro, Portugal
jnvieira@ua.pt

Catarina Mendonça
CCG‡ , Algoritmi Centre
Universidade do Minho
Guimarães, Portugal
catarina.mendonca@ccg.pt

Jorge Santos
EPsi,§ , CCG‡ , Algoritmi Centre
Universidade do Minho
Guimarães, Portugal
jorge.a.santos@psi.uminho.ptr

ABSTRACT

A computer application was developed to simulate the process of microphone positioning in sound recording applications. A dense, regular grid of impulse responses pre-recorded on the region of the room under study allowed the sound captured by a virtual microphone to be auralised through real-time convolution with an anechoic stream representing the sound source.

Convolution was performed using a block-based variation on the overlap-add method where the summation of many small sub-convolutions produced each block of output data samples. As the applied RIR filter varied on successive audio output blocks, a short cross fade was applied to avoid glitches in the audio.

The maximum possible length of impulse response applied was governed by the size of audio processing block (hence latency) employed by the program. Larger blocks allowed a lower processing time per sample. At 23.2ms latency (1024 samples at 44.1kHz), it was possible to apply 9 second impulse responses on a standard laptop computer.

1. INTRODUCTION

A recording engineer can choose where to place the microphone relative to a sound source to achieve different recording effects. Each possible position will impart a different characteristic on the recorded audio; for example, the ratio of direct to reverberant sound (clarity) and the frequency response (equalisation), two major factors in indoor sound recording quality, can be largely controlled by skillful microphone positioning. The FORTIUS project [1] investigates the subjective, empirical microphone positioning criteria used by recording engineers and how they correlate to the objective characteristics of the sound field, envisaging the development of an intelligent robotic microphone positioning system. Given the location of the sound source, the room acoustic effects for each recording position are completely described by the respective room impulse response (RIR); any objective acoustic parameter can be extracted from it. Convolving this RIR with an anechoic recording of the sound source, it is possible to auralise the audio stream that would be captured by a microphone at that position. Therefore, objective characterisation of the sound field amounts to obtaining a dense grid of RIR in the region of interest. Since room impulse responses can be accurately and efficiently measured *in situ*, with one popular method being the logarithmic sine-sweep technique proposed by [2], a 2-axis position control platform (figure 1) was adapted and programmed to perform this task automatically.

Developing a virtual microphone positioning tool based on the pre-recorded RIR, i.e. a computer application with a graphical user interface (GUI) which allows the user to move the virtual microphone within the representation of the RIR-grid region and auralise the output in real time, is invaluable for the FORTIUS investigation, as it can immensely facilitate subjective positioning data collection and systematic evaluation tests. The main challenge, addressed in this paper, is convolving the source anechoic audio stream with the corresponding impulse response to obtain the microphone output in real time, with seamless transitions between RIR as the microphone position is changed.

* Department of Electronics, Telecomunications and Informatics
† Institute of Electronics and Telematics Engineering of Aveiro
‡ Centre for Computer Graphics
§ School of Psychology

2. AURALISATION PROCESSING

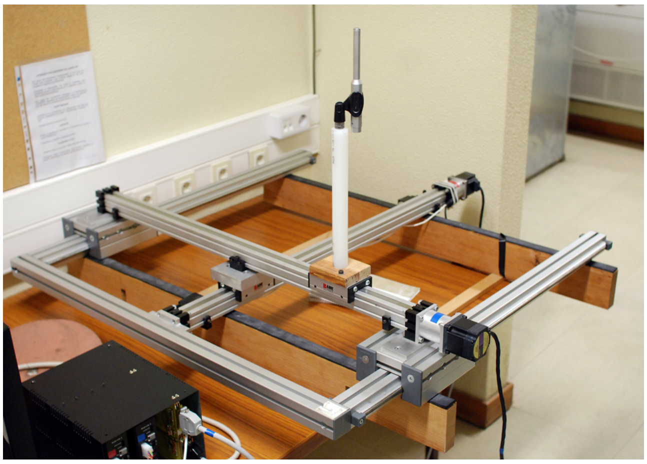

Figure 1: *Mechanical platform used for recording a 2-dimensional grid of room impulse responses.*

The grid of recordings was used as the primary source material for the virtualisation software described. Convolution between an audio stream and any of the measured RIRs allows the user to simulate the positioning of a microphone at points within the grid.

The IRs used in the auralisation were measured sequentially using logarithmic chirps, [2] over an 8x10 grid spaced at 10cm. These recordings were later synchronised in time as described in [3]. The anechoic audio source material used was from a library of recordings held at the Universidade de Aveiro.

The convolution method employed was based on the overlap-add method, [4] where blocks of data for output are produced efficiently by retaining a portion of the previous output calculation. An adaptation of the traditional overlap-add method takes account of the variation of the filter applied on each block of output data. Long RIRs (>3s at 44.1kHz) are efficiently processed so that for each block of output data, redundant sample calculations are reduced. The filter may change on successive output block calculations, so retaining samples for the overlap-add method is only useful in calculating samples to cross-fade into new data blocks.

Real-time binaural audio spatialisation using head-related transfer functions (HRTFs) presents a similar problem and requires similar convolution and cross-fading procedures. There is, however a significant difference in terms of the length of the IR kernel; RIRs, used in this case, are a few seconds long, while HRTFs, used in binaural audio spatialisation, are in the ms range (normally 128 samples at 44.1 kHz), allowing short DFTs of incoming data to be used directly in convolution. The RIR convolution procedure addressed in this paper is therefore much more complex and real-time operation becomes a comparably non-trivial challenge.

2.1. Convolution for Auralisation

For fixed microphone and source positions, auralisation is produced by convolution between an incoming audio stream, $x[n]$ and the RIR filter $h[n]$:

$$y[n] = x[n] * h[n] \tag{1}$$

For signals longer than a few samples, it is almost always more efficient to perform convolution in the frequency domain. The Discrete Fourier Transform (DFT) turns the otherwise computationally expensive time-domain convolution operations into simple dot products between discrete frequency spectra:

$$y[n] = iDFT(X(f) \cdot H(f)) \tag{2}$$

Moreover, since the RIR data are pre-recorded, the corresponding DFTs can also be pre-calculated, saving time in the production of convolution output. The use of long RIRs requires the DFT of a large number of previous samples. These are obtained in small portions using shorter length DFTs, but retention of the frequency-domain data allows efficient reuse in calculating successive output data. Figure (2) highlights how calculated data can be reused to improve throughput for the real-time application described.

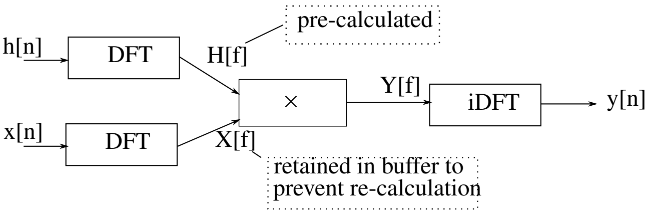

Figure 2: *Block diagram indicating where efficiencies useful in increasing real-time throughput can be achieved.*

The overlap-add (OA) method[4] of convolution allows for output $y[n]$ to be produced in segments, so that immediate output can be produced without requiring the complete set of data (as is the case with streaming audio input). Using OA, only the samples required for immediate audio output are calculated at each stage.

2.2. Auralisation with changing microphone position

In this application, the RIR changes according to the position of the microphone, tracked in real time. This means that the output signal for auralisation, y, must be divided into blocks for processing. The overlap-add method accommodates this requirement very well. The blocks must be relatively short, since their length determines the response time of the system. It must be emphasized, however, that regardless of block length, input signal samples as far back as the RIR length must be correctly integrated into the output sum. The nearest neighbour recorded impulse response is chosen in each instance. The key to real-time auralisation under these conditions is to perform the minimal number of calculations necessary to obtain the current output block data. A method of convolution was employed to ensure just that. The use of long DFTs is avoided by transforming the incoming data in small blocks of 128 to 4096 samples. Summation of multiple (frequency domain) convolutions between pairs of data blocks allows a single block of output data to be produced efficiently. Due to its linear nature, a single iDFT can be used to transform the summation of many frequency-domain block convolution pairs; the result is the same as would be achieved through summation in the time domain, which would require an iDFT for each block pair. Division of the convolution into blocks also ensures that all DFTs are performed on block sizes which are close to powers of 2; transforms of this size are generally carried out more efficiently.

In the general case, a different RIR is applied to each successive block of input data, since the user may have moved to a new virtual position. This would make storing data for use with the next block calculation redundant. However, these retained data are used to generate the samples for cross-fading, increasing efficiency by re-use of prior calculation. The crossfade is a requirement to avoid artefacts produced by sudden transitions between RIR filters.

Each output block is generated using the OA method, through the summation of the latter and former halves of the output of convolutions. Since the filter being applied potentially changes for each block of output data, both portions of the overlapping region must be calculated for each output block.

2.3. Convolution Method

Incoming blocks of audio data, length N, are padded with zeros to the length 2N-1. They then undergo the DFT and are stored in a circular buffer. The length of the RIRs being used influences the size of the buffer. An RIR length L requires a storage buffer of (L/N)+1 blocks of data. RIR data is transformed into the frequency domain offline. For each block of output (length N) produced, only the DFT of incoming data and iDFT of each half of the overlap add convolution need be calculated in addition to the dot-product multiplications. Since only N output samples are required for each output block, these data are calculated through the summation of many short sub-convolutions as shown in figure (3).

2.3.1. Convolution using real data

Any sample in the output stream is computed by the convolution between the RIR filter h, (length L) and the input stream, x. The value of each output sample is described by the standard convolution sum. Since h is real valued and causal, each output sample value is calculated from a summation involving the previous L input samples and all non-zero samples of the RIR. RIR sample values before time $t = 0$ are zero-valued and do not contribute to the output sum. For this application, the convolution sum can therefore be written as:

$$y[n] = \sum_{k=0}^{L-1} x[k]h[n-k] \tag{3}$$

2.3.2. Block Based Convolution Notation

To explain how blocks of samples are combined within the convolution scheme, a notation for separate regions of samples is defined. Sample indices are such that the first sample in time is $x[0]$ and $h[0]$ in the incoming audio stream and any RIR respectively.

Dividing both input $x[n]$ and RIR streams $h[n]$ into blocks of length N, the m^{th} blocks, x_{B_m} and h_{B_m} (where $m >= 0$), are defined as:

$$x_{B_m} = x[mN; (m+1)N - 1] \tag{4}$$

$$h_{B_m} = h[mN; (m+1)N - 1] \tag{5}$$

For example, with $N = 512$, the third block of the input stream ($m = 2$) is $x[1024; 1535]$.

A block of output for the static RIR case (no change in microphone position) is calculated as the overlap-add of two intermediate outputs. Each intermediate output is produced through summation of convolutions between blocks as described:

$$y_{B'_n} = \sum_{l=0}^{NB-1} x_{B_{[l]}} * h_{B_{[n-l]}} \tag{6}$$

where

$$NB = L/N \tag{7}$$

It is assumed that L is an integer multiple of N, and this can be achieved through zero padding where necessary.

2.3.3. Output generation from block convolution

Convolution is performed in the frequency domain (figure 2), and summation of block convolutions occurs before transformation back to the time domain. An output of length $2N-1$ is produced. Summation (equation 9) of the first $N-1$ samples with the last $N-1$ samples of the previous block calculation ($y_{B'_{(n-1)}}$ shown in figure (3)) produces $N-1$ valid output samples (sample N requires no summation). Due to the use of frequency domain convolution one DFT and two iDFTs of length $(2N-1)$ are required per output block.

From (3) and (6) each sample in $y_{B'_n}$ is :

$$y_{B'_n[k]} = \sum_{l=0}^{NB-1} \sum_{k=0}^{L-1} x_{B_{[l]}[k]} h_{B_{[n-l]}[m-k]} \tag{8}$$

In the non-static RIR case (change in microphone position), the previous $N-1$ samples are calculated using the previous stored input data and the current RIR data leading to the expression:

$$y_{B_n}[n] = \begin{cases} y_{B'_{(n-1)}}[n+N] + y_{B'_n}[n] & 0 <= n < N-1 \\ y_{B'_n}[n] & n = N-1 \end{cases} \tag{9}$$

Still, samples

$$y_{B'_n}[N; 2N-1] \tag{10}$$

calculated with the previous RIR, should be retained for the crossfade procedure described in the following section.

In the case of the static RIR, with the same filter applied to all output blocks, equation (8) is all that is required to generate correct output.

2.4. Calculation of the Crossfade

2.4.1. Creating Crossfade Data

In order to reduce audible glitches as different filters are applied to successive blocks of audio data, a crossfade must be applied. Convolution of the current block of input data and the previous RIR is faded with the convolution with the current block and current RIR (figure 4). The overhead of creating one block of crossfade data is roughly an additional 50% of the previous block, since half of the calculation was already performed in the previous time period, as per equation (10). Calculating an entire block of data for crossfading allowed experimentation with crossfade length. Note that:

- The shorter the processing block length, N, the less distance the user's cursor can move between block outputs, since the position is polled more regularly. Less extreme variations of RIR will occur, since RIRs close to the current one will be selected on subsequent output.

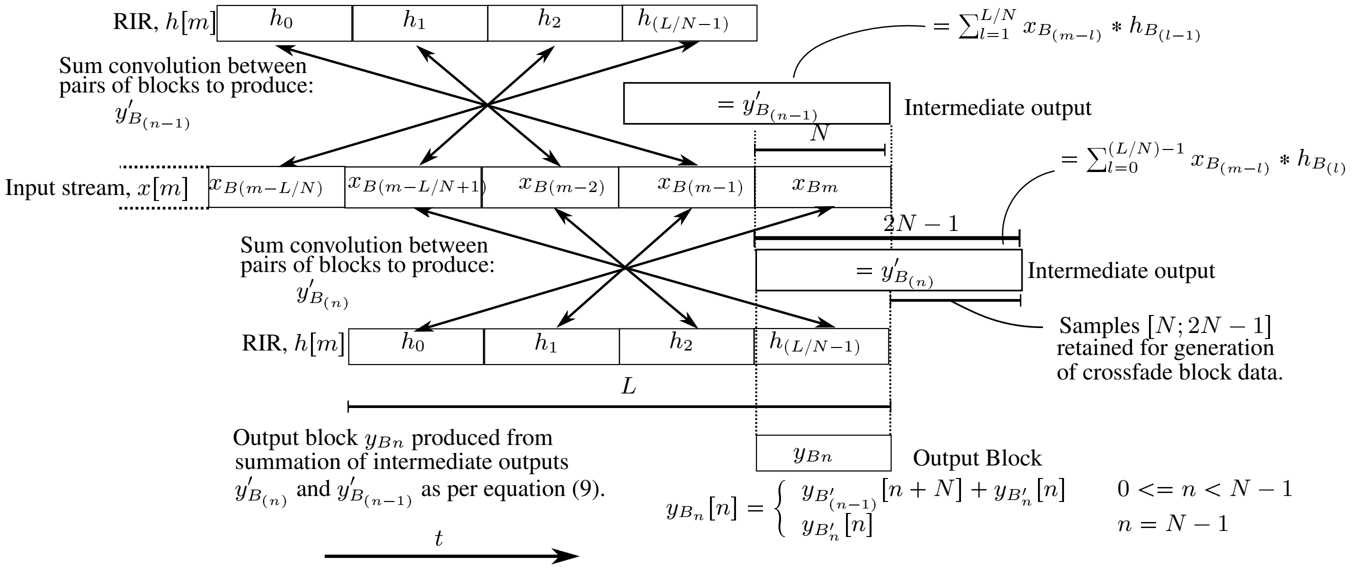
RIR, $h[m]$
h_0
h_1
h_2
$h_{(L/N-1)}$
$= \sum_{l=1}^{L/N} x_{B_{(m-l)}} * h_{B_{(l-1)}}$
Sum convolution between pairs of blocks to produce: $y'_{B_{(n-1)}}$
$= y'_{B_{(n-1)}}$
Intermediate output
N
$= \sum_{l=0}^{(L/N)-1} x_{B_{(m-l)}} * h_{B_{(l)}}$
Input stream, $x[m]$
$x_{B(m-L/N)}$
$x_{B(m-L/N+1)}$
$x_{B(m-2)}$
$x_{B(m-1)}$
x_{Bm}
$2N-1$
Sum convolution between pairs of blocks to produce: $y'_{B_{(n)}}$
$= y'_{B_{(n)}}$
Intermediate output
Samples $[N; 2N-1]$ retained for generation of crossfade block data.
RIR, $h[m]$
h_0
h_1
h_2
$h_{(L/N-1)}$
L
Output block y_{Bn} produced from summation of intermediate outputs $y'_{B_{(n)}}$ and $y'_{B_{(n-1)}}$ as per equation (9).
y_{Bn}
Output Block
$y_{B_n}[n] = \begin{cases} y_{B'_{(n-1)}}[n+N] + y_{B'_n}[n] & 0 <= n < N-1 \\ y_{B'_n}[n] & n = N-1 \end{cases}$
t

Figure 3: *Diagrammatic representation of how blocks of the input stream x and an RIR, h, are combined to produce a block of output data. In this example, $length(h) = 4N$. All convolutions are performed in the frequency domain. Block pairs indicated with arrows are convolved, and added to a cumulative output sum, y'_{B_n} or $y'_{B(n-1)}$. Blocks y'_{B_n} and $y'_{B(n-1)}$ are overlap-add combined to produce y_{B_n} as per equation (9).*

- The longer the crossfade, the less sudden the transition between output blocks.
- During the crossfade, output is only unrealistic when the RIRs applied on successive blocks are different.

With these points in mind, it was found that keeping the crossfade proportional to block length was the best option. A linear fade of 10% of the block length was applied.

At each output block instance, two blocks of data are available for output. They consist of the current input block, x_{Bn} convolved with both the current and previous RIRs y_1 and y_2, to produce two blocks of output $y_{1B_{(n)}}$ and $y_{2B_{(n)}}$ respectively, (figure 4).

2.5. Algorithm Pseudocode

Pseudocode describing the generation of the program output is divided into two sections: Set Up and Iterative Processing. In the set up, preprocessing is carried out to initialise all data for efficient frequency domain convolution. During the iterative processing, the convolution method of producing valid output data for each block of input data is described.

2.5.1. Set Up

At the initialisation of the program, the environment is set up to reduce processing times when incoming audio data is processed. Buffers are initialised and RIR audio data is partitioned into blocks and transformed into the frequency domain. The set up procedure is as follows:

- Determine length of RIRs to be convolved (L), and block size (N). Define number of blocks as $NB = ceil(L/N)$.
- Allocate circular buffer for $NB + 1$ blocks of incoming data, length $2N - 1$
- Read data from RIR files block by block. Transform into frequency domain, stored in NB buffers $2N - 1$ long.

2.5.2. Iterative Processing

The following code is continually looped:

- Get cursor position and determine corresponding RIR.
- Read block of incoming audio data
- Perform DFT on incoming data and store latest block in circular buffer
- Reset all output arrays to zero
- **To generate crossfade block data:**
- Generate the data for the crossfade block $y_{1B_{(n)}}$ using previous RIR position data $y_{1B'_{(n-1)}}$
- For (K = NB; K++)
    - Read the K'th most recent block of complex data from the circular buffer
    - Perform sample-by-sample multiplication with complex RIR data block (K) and sum result into output array $Y_{1B'_n}$.
- iDFT $Y_{1B'_n}$ and sum with $y_{2B'_{(n)}}[N; 2N-1]$ from previous block calculation to produce crossfade block.
- **To generate current output block data:**
- For (K = NB; K++)
    - **To generate** $Y_{2B'_n}$
    - Read the K'th most recent block of complex data from the circular buffer

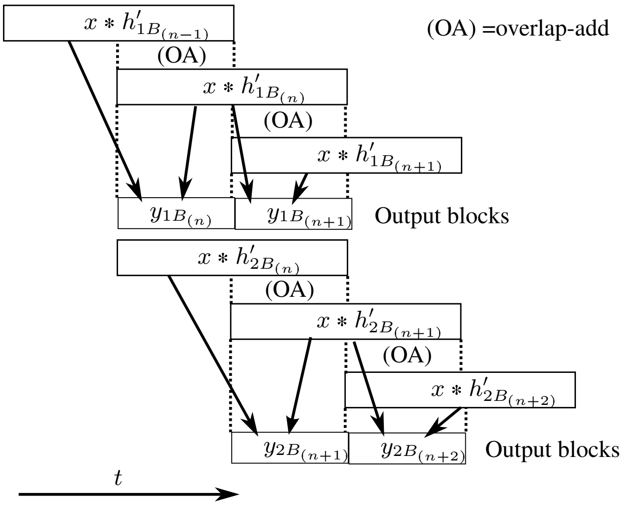

Figure 4: *Crossfade Calculation Scheme. Three sub-blocks calculated with equation (6) are summed to create pairs of output blocks. Output blocks $y_{1B_{(n+1)}}$ and $y_{2B_{(n+1)}}$ are crossfaded, so that change in the applied convolution filter does not produce a sudden, perceptible transition in the audio output.*

  - Perform sample-by-sample multiplication with complex RIR data block (K) and sum result into output array $Y_{2B'_n}$.
  - **To generate** $Y_{2B'_{(n-1)}}$
  - Read the (K+1)'th most recent block of data
  - Perform sample-by-sample multiplication with complex RIR data block (K) and sum result into output array $Y_{2B'_{(n-1)}}$.

- Perform iDFT on output arrays $Y_{2B'_{(n-1)}}$ and $Y_{2B'_n}$.
- Sum $y_{2B'_{(n-1)}}$ and $y_{2B'_{(n)}}$ as in (9).
- Retain $y_{2B'_{(n)}}[N; 2N-1]$ for crossfade calculation in next iteration.
- Store RIR index for calculation of crossfade

3. PROGRAM IMPLEMENTATION

3.1. Implementation Details

The software was written in C++ for a Microsoft Windows environment. The audio source to be convolved was a .wav file, read using the LibSndFile library [5]. Audio output was performed using PortAudio[6] at a sample rate of 44.1kHz. Convolution was performed in the frequency domain, and the fftw3 [7] library was used to perform the DFT and iDFT on incoming blocks of audio data and output blocks, respectively. A Fast Fourier Transform (FFT) algorthm is selected by fftw3 at runtime; the fastest method algorithm is sought within the many available to fftw3. The depth of the search is user specifiable; the PATIENT flag was used which results in a fairly thorough but not exhaustive search of available methods of performing the DFTs.

3.2. User Interface

The user is presented with a small window which represents the 2-dimensional space in which the RIRs were recorded. The position of the mouse cursor within the window is monitored. To correspond with the 8x10 grid of RIR recording positions, the window is divided accordingly into 8x10 regions. The RIR to apply to the incoming audio stream is chosen based on the position of the mouse cursor. The nearest neighbour RIR is selected (figure 5).

3.3. Validation Tests

3.3.1. Convolution Validity

The result of convolutions performed using the method described were compared with frequency domain fast convolutions calculated in Matlab. Output from the block-based convolution was calculated with a static RIR and stored as a .wav file. Normalised algorithm output was equal to that produced by direct convolution in Matlab.

3.3.2. Changing RIR Validity

The calculation of varying RIR output was compared with similar output computed in Matlab. Within Matlab the same source file s was convolved with two different RIRs, a and b. These outputs were divided and concatenated so that the first portion of $a * s[1 : L/2]$ preceeded $b * s[L/2 + 1 : L]$. Within the block based convolution, the RIR used for convolution was a prior to block $L/2 + 1$ and b afterwards. During this test, no crossfade was applied in either environment. Output with the proposed algorithm was consistent with that produced directly in Matlab.

3.3.3. Crossfade Valididity

Since separate RIRs can be applied to successive blocks, a small crossfade was applied. The crossfade was linear, and lasted for 10% of the block duration. The same data a was used for each RIR, and the crossfade was applied as described. In all cases, the output was the same as direct convolution $a * s$.

4. SYSTEM PERFORMANCE

The method exhibits a more or less linear computational increase with increasing filter length, allowing for greater ease of calculation of system performance limits. Latencies of powers of 2 between 128 and 2048 samples were tested, and maximum RIR lengths of between 0.5 and 10 seconds were successfully applied.

The processing time to produce one block of output was analysed for different RIR lengths. The results were measured as a portion of the available processing time. At 44.1kHz, a 512 sample processing block allows 11.6ms to process the output, before the next block is output. Increasing the block size increases the available processing time, yet increases system latency, and also the number of additions and multiplications which must be performed.

4.1. Measurement method

The processing time was measured using functions inside the *time.h* C header file. Time taken to perform processing of each block of output data was measured for each block and averaged over 30s of

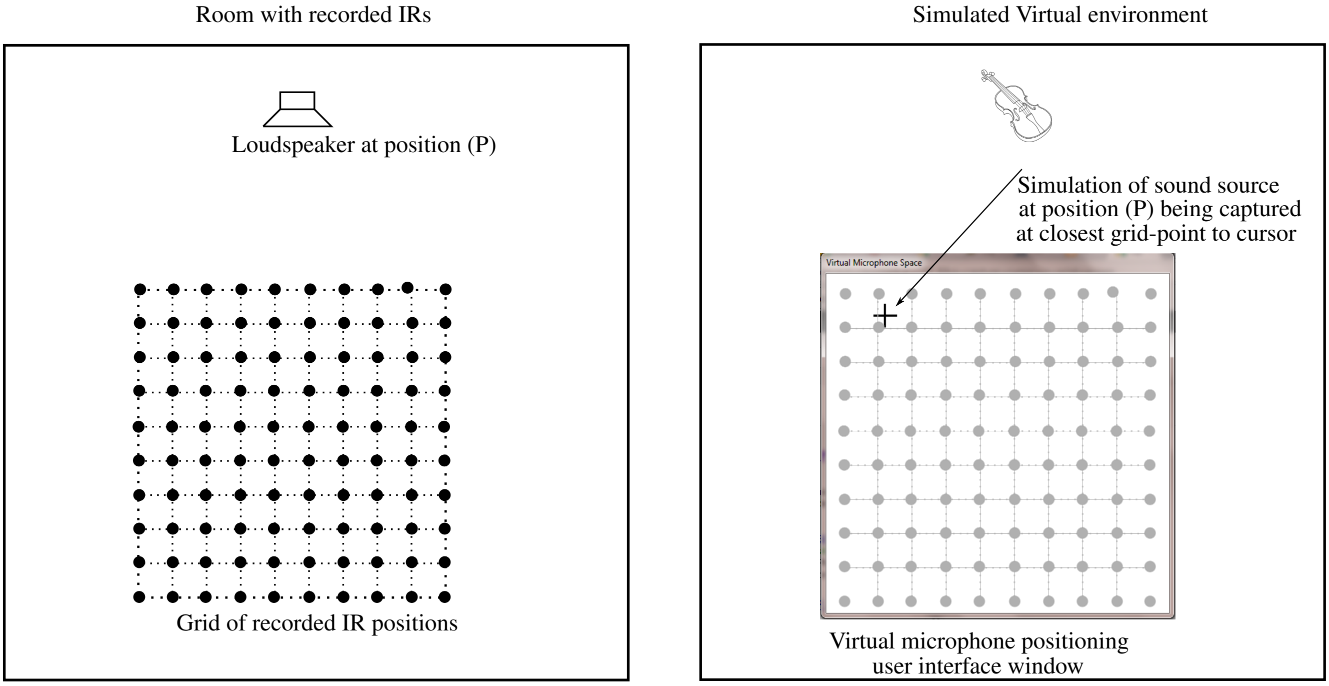
Room with recorded IRs
Loudspeaker at position (P)
Grid of recorded IR positions
Simulated Virtual environment
Simulation of sound source at position (P) being captured at closest grid-point to cursor
Virtual Microphone Space
Virtual microphone positioning user interface window

Figure 5: *How the user interface window allows the user to simulate the position of a microphone relative to a fixed sound source.*

output. The computing platform was a laptop running Microsoft Windows 7 on an Intel Core i3 M370 CPU at 2.4GHz with 3GB of RAM. Longer RIRs of up to 10 seconds were created by appending exponentially decaying noise onto the end of shorter RIR recordings. Shorter RIRs were produced by trimming the captured RIRs.

4.2. Results

As the length of the RIR to be convolved increased, so did the processing time, for all block sizes. The absolute processing time for each block was lower for smaller block sizes, since less samples need to be calculated. The calculation time per output sample increased with increasing RIR length, and decreased with block size.

For longer block lengths, the processing time per block increased, but as a percentage of the available computation time for that block, decreased. Longer RIRs produced longer computation times in all cases.

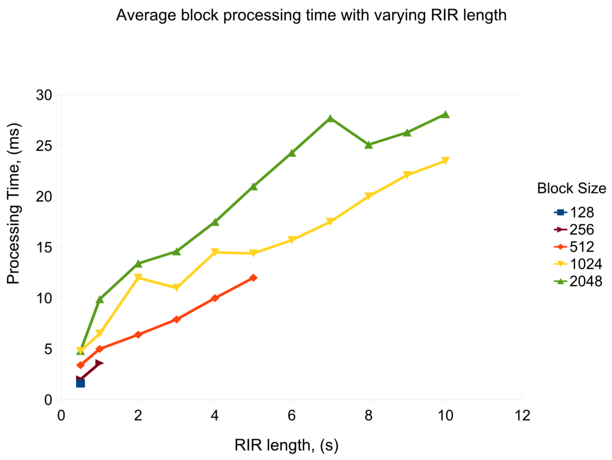

Figure 6: *The average time taken to process each block of output data increased with both block size and RIR length*

4.3. Analysis

Larger block sizes allow more efficient calculation of output samples. By shifting priority away from response time, larger chunks of data can be processed in each step, reducing the overheads produced when transforming between time and frequency domains. For large RIRs, the majority of the processing time is spent in multiplying frequency-domain samples rather than transforming data. This produces a relatively linear increase in processing time with increasing RIR length, as the number of complex multiplications required is directly proportional to the length of the RIR applied. The increase in per-sample computation time does not exhibit a monotonic decrease with increasing RIR length though. As larger RIRs are used, larger contiguous chunks of memory are used to store the data being processed, which affects the resources available to the program. The fftw library does not, by default, produce deterministic behaviour for the same size transform between different executions of the program. Resources available at runtime influence the DFT algorithm selected by fftw, and this can vary for different RIR lengths. Slower DFT algorithms will increase the per-sample calculation time; hence the curve shapes in figure (6) and (7). Data movement and allocation requirements also vary

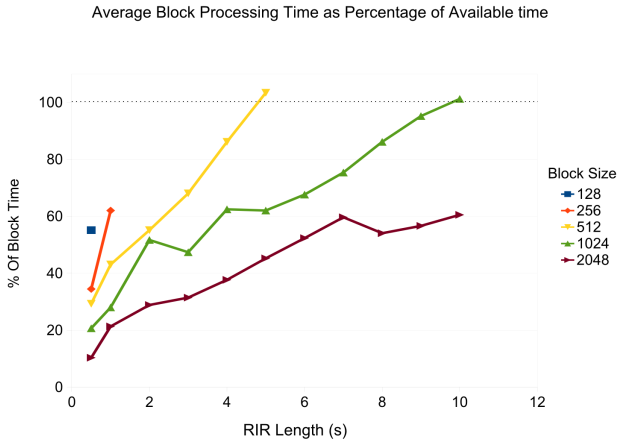

Figure 7: *The percentage of block time to calculate each sample of output decreased as processing block size increased. Increasing RIR size produced longer computation times in all cases.*

with RIR length; different compilers and computer architectures may exhibit slightly different behaviours as a result.

5. CONCLUSIONS

The software allowed for long, varying impulse responses to be convolved with streams of incoming audio data in real time. The maximum length of RIR which can be convolved can be profiled and measured for a given latency. Where the convolution will not meet the processing deadline, response latency can be increased.

5.1. Future Work

5.1.1. Virtual Reality Auralisation

Potential use of this software is by no means restricted to the FORTIUS project. The main goal is now to adapt it for virtual reality auralisation systems with user position tracking (AcousticAVE project). Large numbers of simulated RIRs can be generated offline using a physical modelling approach; incorporation of head-related transfer functions as an extension of the model could provide a convincing feeling of immersion. With constraint of the user's position to particular paths, it would be feasible to allow a large degree of freedom in a physically modelled virtual environment without the processing requirements of performing physical modelling in real time.

5.1.2. Increasing Calculation Speed through Parallelisation

Convolution calculation speeds could be increased by use of modern computer architectures. The algorithm lends itself to parallelisation. For example, setting the block dot products to separate threads is likely to produce a large increase in speed on multi-core systems, now commonly used. Processing times could realistically be almost halved on a simple two-core system. Use of general-purpose graphics processing units (GPGPU) is also an interesting possibility.

5.1.3. 3D extension and GUI Refinement

A third (z) positioning axis was already fitted on the platform used for RIR data collection, to allow automatic recording of 3D RIR grids. Adapting the program and its GUI accordingly is straightforward and will improve its applicability as a training tool for sound recording technicians. The convolution algorithm at the core of the processing can remain the same. It is hoped that subjective feedback about the software can be obtained, especially from experienced sound recording technicians.

6. ACKNOWLEDGMENTS

This work was funded by the FCT as part of the project 'Auralisation Models and Applications in Virtual Reality Environments' (AcousticAVE) reference PTDC/EEA-ELC/112137/2009.

7. REFERENCES

[1] C. Rodrigues, J. Vieira, and G. Campos, "Fortius: Robot para captação de som," in *Proc. AES Portugal*, Lisbon, Portugal, Dec. 12-13 2008.

[2] A. Farina, "Simultaneous measurement of impulse response and distortion with a swept-sine technique," *108th AES Convention*, vol. 108, pp. 1–24, 2000.

[3] A. Vieira, J. Vieira, and G. Campos, "Caracterização objectiva e subjectiva de campos sonoros," in *Proc. AES Portugal*, Lisbon, Portugal, Dec. 12-13 2008.

[4] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, vol. 3, California Technical Publishing, 1997.

[5] E. de Castro Lopo, "Libsndfile," *http://www.mega-nerd.com/libsndfile/.*

[6] R. Bencina and P. Burk, "Portaudio - an open source cross platform audio api," in *International Computer Music Conference Proceedings*, 2001, http://www.portaudio.com/.

[7] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.