UNIVERSITY OF MIAMI

HYBRID REVERBERATION ALGORITHM USING TRUNCATED IMPULSE RESPONSE CONVOLUTION AND RECURSIVE FILTERING

By

Sean Browne

A RESEARCH PROJECT

Submitted to the Faculty of the University of Miami
in partial fulfillment of the requirements for the degree of
Master of Science in Music Engineering Technology

Coral Gables, Florida
June, 2001

BROWNE, SEAN (M.S., Music Engineering Technology)
(June 2001)

Hybrid Reverberation Algorithm Using Truncated Impulse Response Convolution and Recursive Filtering

Abstract of a Master's Research Project at the University of Miami.

Research project supervised by Professor William Pirkle.
No. of pages in text: 79

Artificial reverberation can be developed by convolving an input signal with the specific impulse response of an acoustic space. This computationally exhaustive method achieves superb reverberation, but is often abandoned for simpler filtering methods. Since it is only the length of the impulse response that determines the computational cost and resolution, a new hybrid algorithm is developed that borrows from the original impulse response easing the computational burden. A truncated impulse response can still convey the rich energy of the space while a digital filter bank can model the late echoes and reverberation tail. This more efficient algorithm will use a reduced length block convolution and recursive filter network to achieve similar high quality reverberation.

Table of Contents

Table of Figures

1. Introduction

Digital audio effects have afforded new levels of creativity for recorded and performed sound. Audio engineers have experimented with both analog and digital audio effects to enhance music while maintaining high fidelity. Artificial reverberation introduces a spatial dimension to a piece of recorded sound. Implementation in the digital domain gives us the flexibility of portable, repeatable and dependable algorithms on which to base reverberation effects.

The frequency coloration and delay properties of a particular reverberation effect can simulate general listening environments. For example, high frequencies give the impression of hard walls while their absence gives the impression of a soft-carpeted room.[9] Long reverberation times provide the feeling of a large hall, while short reverberation times give the impression of smaller rooms.

Artificial reverberation can also be used to model a specific acoustic environment in which to affect a dry unaltered signal. In the digital domain, processor speed is the limiting factor, thus the balance of fidelity, processing time and application becomes the crux of appropriate reverberation. If the application is a real-time system, fidelity has to be sacrificed so that the processing time can be maintained according to the application demands. However, if the application is strictly post-processing in nature, fidelity can be extremely high with processor-intensive calculations.

This thesis will try to destroy this dichotomy and investigate a new method by which a real-time system can achieve high fidelity with modest processor demands. This will be achieved through the merging of the two commonly used techniques: DSP filtering and acoustic impulse response convolution. While any acoustic environment can be modeled in the manner, the testing of this thesis was limited to concert halls, auditoriums, churches, etc. These acoustic spaces have predictable reverberation characteristics and high quality measured impulse responses are easily accessible. In this paper, the general term "room" will be used to represent these acoustic spaces.

Chapter 2 will introduce the major design tools, building blocks, cornerstone implementations and quality measurement systems that are common in current reverberation research. In order to realize a perceptually optimized artificial reverberation architecture, we must first be fluent in the ideology of current reverberation techniques. The utilities of convolution, comb filters and all-pass filters have been a staple ingredient in the pursuit of high quality reverberation. A hybrid architecture that marries the two typically independent reverberation techniques is described in Chapter 3. Methods of impulse response manipulation, frequency equalization and diffuse reverberation are also introduced. The specific design criterion is explored in Chapter 4. Practicality, necessity and fidelity are weighed against each other for finalizing the new algorithm. The new hybrid architecture is implemented and tested in Chapter 5 with a variety of listening material and using a PC based ABX software package. Finally, Chapter 6 presents applications, shortcomings and future development of this system.

2. Reverberation Introduction

Digital artificial reverberation effects can be found in software programs, instrument amplifiers, public address systems, rack-mounted effects consoles, etc. to add a spatial component to audio. This spatial component stamps a recording with a character unique to the given environment. Whether this environment is artificial or natural, it can be just as crucial as the recorded information itself. Reverberation effects can be achieved by using any combination of techniques.

2.1 Artificial Reverberation: Impulse Response Convolution

The impulse response of an acoustic environment represents how any single digital sample will behave in the acoustic space. Figure 2.1 shows a theoretical impulse response with its three main components (direct signal, early reflections and reverberation tail) as well as a measured medium room impulse response used in the Winamp "RealReverb" plugin.

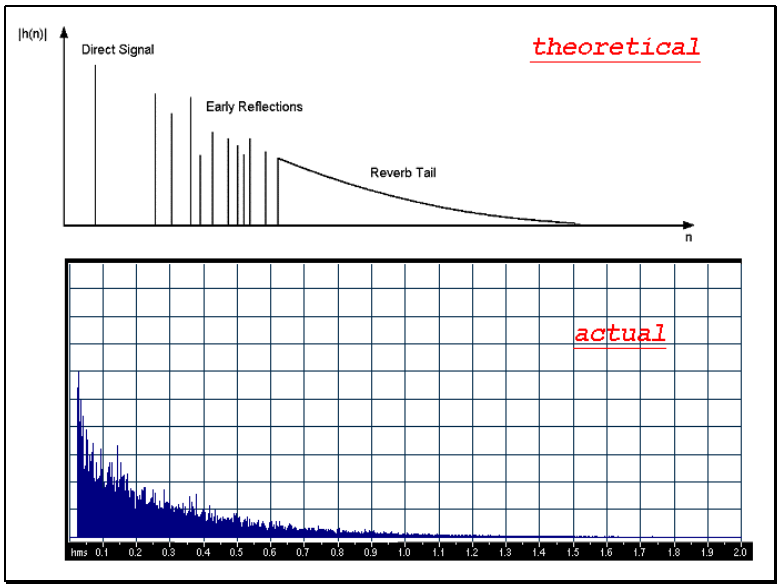

Figure 2.1: Ideal impulse response and measured (RealReverb Plugin)

For digital systems, the most natural sounding artificial reverberation is achieved by convolving an anechoic ("dry") input sequence with an impulse response. When this is done, the output of the system will sound like the input was excited and recorded in the same manner and location as the impulse response. For example, an impulse response is measured by playing a stimulus through a speaker and recorded from the rear of an auditorium. Convolving any input with this impulse response will result in an output that sounds exactly as if the input was played through the same speaker and recorded in the exact same location in the rear of the auditorium. The impulse response is the response of a system to a unit impulse [23] and a digital audio signal is a sequence of attenuated and delayed impulses. Thus, the convolution operation defines how each of these samples behaves within the system defined by the impulse response. Several convolution techniques are discussed in section *2.1.2*.

2.1.1 Measuring Impulse Response

Determining the impulse response of an acoustic space has become a rich area of research with many different methods currently in widespread use. Some consider only the geometry and the materials of a space, while others use specific sound sources and recording techniques in order to capture the acoustic signature of a room.

All acoustic spaces are designed in an attempt to make every listening location possess the same ideal sound perception. However, due to the frequency dependent directional nature of sound waves, not all locations will experience the same sound perception, and none are guaranteed to be ideal. When a single monophonic impulse response is to be

used for generating artificial reverberation, great care is taken in deciding where in the acoustic space the impulse response is going to be measured. It is this location that will become the virtualized listening location for the newly reverberant signal.[2] While producing an acceptable impulse response is not within the scope of this thesis, the various techniques are examined so that educated decisions regarding impulse response selection can be made.

2.1.1.1 Geometrical Acoustics

The Image-Source model is a geometrical/materials method of determining a room's impulse response without using an acoustic stimulus and recording device. This method examines the effects of an acoustic source in a room with corresponding sources located in image rooms with reflecting boundaries. Each of the infinite image sources will produce attenuated, filtered and delayed versions of the original acoustic input. The total model effects are summed to produce the transfer function and impulse response. It is the materials of the boundaries, contents of room model, and the size of the room model that will determine the level of attenuation of the image sources, determining the reverberation characteristics.[26]

Figure 2.2 shows an example of how the Image-Source model is used. The original source produces virtual sources (indicated by **x**) that are attenuated due to the materials of the boundaries and the effects of each are accumulated at the listener location (indicated by **o**).

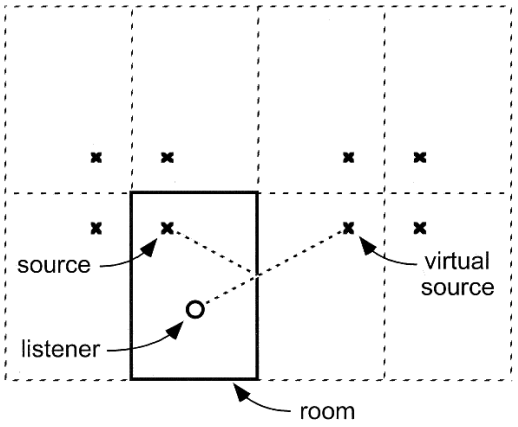

Figure 2.2: Image-Source model

A common reverberation measurement is the "reverberation time." This figure was defined in 1922 by Wallace Sabine to be the time taken for a steady-state sound to be switched off and drop in intensity by 60dB. Sabine introduced a method of predicting this time for the Image-Source model that was based on wall surface area and absorption. Since his studies were done typically using the middle C note on a pipe organ, most reverberation times *(RT_{60})* were initially given for 512 Hz, but are now usually given for octave to ⅓ octave intervals across the human hearing spectrum. RT_{60} values can be as high as 20 seconds for large empty concert halls and as short as 200 milliseconds in the case of an automobile. [6]

A family of geometry-based models exists that center around a spherical point source in a model room emitting ray, cone or pyramid shaped sound "traces." These shaped traces are drawn throughout the simulated room and are reflected and attenuated at boundaries according to the materials of the walls. The effects of the traces are summed at a particular location of interest in order to realize the time/frequency characteristics the room introduces.[26] Figure 2.3 depicts a stereo ray tracing example where **L** and **R** are the left and right ear components. The geometrical shape of the traces emitted from the point source has been the most developed component of this type of measuring method. Farina has developed the pyramid-tracing simulation technique for Ramsete, an indoor and outdoor acoustic simulation program.[7]

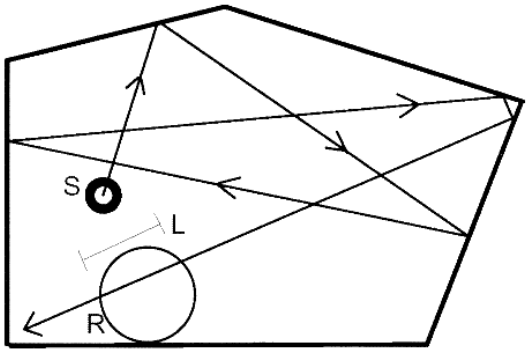

Figure 2.3: Ray-tracing model

These Geometrical Acoustics methods for determining the impulse response of a room do not provide the most useful impulse response on which to perform musical operations. Architects generally use these techniques in the planning stages of constructing the acoustic properties of a specific space, not for modeling an existing room. A far easier and more accurate method is used for capturing actual room responses.

2.1.1.2 Acoustic Measurement

In 1997 The International Standards Organization published the revised standard #3382: "Acoustics - Measurement of the reverberation time of rooms with reference to other acoustical parameters." This standard outlines specific measurement parameters and methods for testing rooms. The specifications include various kinds of reverberation time and clarity measurements:

- EDT (Early Decay Time) – time for impulse response to decay from 0dB to –10dB
- T20 – time for impulse response to decay from –5dB to –20dB
- T30 – time for impulse response to decay from –5dB to –25dB
- C50 – the ratio of energy in the first 50ms to the remainder of the impulse response
- C80 – the ratio of energy in the first 80ms to the remainder of the impulse response

Because of the frequency dependence, each measurement is evaluated over the standard 10 octave bands from 31.5 Hz to 16 kHz. For each of these frequency bands a signal-to-noise ratio of 20dB – 45dB is needed for accuracy. Background noise in rooms is completely unpredictable and varies with occupancy, frequency, materials, airflow, measurement location, and acoustic stimuli. [13]

When acoustic stimuli are used to measure an impulse response, a host of outside influences are put into play that can easily corrupt the data that is gathered. When acoustic signals are used, such as swept sinusoids, pseudo-random sequences, or noise, an amplifier and speaker are used to inject the stimulus into the room and a microphone and computer are used to capture the results. Each of these additional components will affect the validity of the measurements.

Short impulsive high-energy sources are often used to excite an acoustic space for measurement purposes. Firearms such as cap guns and blank pistols as well as handclaps and balloon bursts have been used to satisfy this need. Ideally an impulse would need to provide:

- flat frequency spectrum
- sufficient S/N ratio for the given acoustic space
- adequate energy for measurement equipment
- omnidirectionality
- repeatability

Unfortunately, none of these impulsive acoustic sources provide ideal measurement properties, but do provide quick and easy computations.[13]

An impulse response measurement can be found without using an impulse-like stimuli, but rather using a deterministic input that can be used to derive the impulse response. In 1967 Richard Heyser developed a method known as Time Delay Spectrometry (TDS) which uses a swept sine wave as the acoustic stimulus. When the product of the input and the recorded response are integrated, the result is the impulse response. Unfortunately, while little computation is required, non linearities in the system can reduce the system signal-to-noise ratio.[14]

D. Giuliani describes the use of a “chirp-like” (swept sinusoid) stimulus for measuring the impulse response of a busy office. The particular application was for hands-free speech recognition for office use using a Hidden Markov Model (HMM) speech recognizer. The “chirp-like” signal is output from a loudspeaker and recorded using a

microphone array placed in the office. Since the input signal is designed to have autocorrelation close to a perfect Dirac delta function, the recorded signal can be easily deconvolved by cross-correlating the recorded signal with the original input. This will give the exact impulse response of the room (plus any contributions from the speaker/microphone arrangement).[12] The use of swept sine waves for acoustic measurements do encounter a drawback in enclosed rooms. Rooms can have signature "modes" in the form of peaks and/or notches in the frequency response. This is created when standing waves set up between walls. While listening spaces are designed to minimize these effects, a swept sine wave will bring attention to the particular frequencies that are affected.

The use of a maximal length sequence (MLS) is often used because of its flat spectrum and mathematical convenience. The MLS is an apparent random sequence of 1s and 0s which, when output from a transducer will have a white overall frequency spectrum. In hardware systems, such as MLSSA, a feedback shift register of length L_{SR} generates the MLS signal with a clocking frequency much greater than those of acoustical interest. The pattern is periodic by:

$$T_{MLS} = \frac{f_{clock}}{2^{L_{SR}}}$$

An MLS signal can also be created within software programs such as Aurora written by Dr. Angelo Farina.[17] Once the signal is generated, it is output from a loudspeaker and recorded and saved in some digital format. To get the impulse response of the room from the recorded signal, deconvolution must be performed to remove the input dependence.

Deconvolution is a method of removing the effects of a filtering process. In the frequency domain, this can be accomplished by multiplying the output spectrum by the inverse of the transfer function. In the time domain, it can be accomplished by convolving the output with the impulse response of the inverse transfer function:

$$y(n) = x(n) * h(n)$$

$$Y(z) = X(z) \cdot H(z)$$

$$\Downarrow$$

$$X(z) = \frac{Y(z)}{H(z)}$$

$$X(z) = H_{inverse} \cdot Y(z)$$

$$x(n) = h_{inverse} * y(n)$$

Since the input was made up of only 1s and 0s the Fast Hadamard Transform (see section *2.1.3*) can be used. This transform, like the Fast Fourier Transform (FFT), makes use of symmetries to reduce required calculations. Since the FHT kernel is based on +1 and –1, only unity multiplications are performed allowing the FHT to consist only of addition operations.

Although the MLS technique does provide a quick and easy computation method, it relies on the assumption that the acoustic system measured has linear behavior and is time stationary. In addition this technique requires that the input signal used is at least as long as the room response in order to capture the full decay of the reverberation tail. However, measuring the tail accurately is problematic due to the fact that this very soft

portion of the decay is being measured in the presence of a high-energy wide band noise source.[14]

The industry standard loudspeaker of choice for output of any of the above measuring signals is the dodecahedron loudspeaker. This is a 12 sided, 12 equal driver arrangement that ensures maximum omnidirectionality of the output signal.

2.1.2 Convolution Techniques

Convolution is a mathematical process that is used to define the interaction of an input signal with a linear, time-invariant system. For digital artificial reverberation, the output of a reverberator is defined as the input convolved with a particular acoustic impulse response. This method produces the most accurate effect because the impulse response defines how each sample behaves in the room and the convolution applies this to every sample. This operation is plagued with computational complexity and has evolved into two principle implementations: direct linear convolution and block convolution.

2.1.2.1 Direct Linear FIR

Convolution is a filtering process that can be described by the following equation:

$$y(n) = x(n) * h(n) = \sum_{m=0}^{N-1} x(m)h(n-m)$$

For the application to artificial reverberation, *x(n)* is the input, *h(n)* is the acoustic impulse response and *y(n)* is the reverberated signal. This process can be realized in the

time domain via an FIR (Finite Impulse Response) architecture as shown in Figure 2.4.

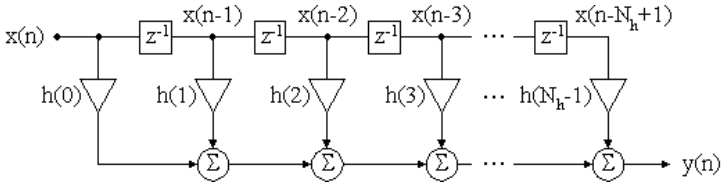

Figure 2.4: FIR implementation of convolution

This is a stable real-time system because each input sample will produce an output sample and the impulse response is of finite length. If the input is N_x samples long and the impulse response is N_h samples long, the output will be $N_x + N_h - 1$ samples long. This brute force process requires $N_x \cdot N_h$ multiplication and addition operations. The output of a 1 second input with a typical room impulse response (about 2 seconds) sampled at 44.1kHz would require ~3.89 billion multiply/adds.

It is apparent that linear convolution is an exhaustive process and if the impulse response is any more that a couple of hundred points, real-time processing quickly becomes impossible with common practical processors. In fact, if the impulse response is any more than 30 points, a much more efficient algorithm can be used to compute the convolution.[21]

2.1.2.2 Direct Frequency

Convolution can also by done in the frequency domain, by taking advantage of the property that convolution in the time domain is equivalent to complex multiplication in

the frequency domain:

$$y(n) = x(n) * h(n) \Rightarrow X(z) \cdot H(z)$$

This method cannot function as a real-time system because the entire input signal must be defined prior to any processing. The entire output sequence is determined by taking the Inverse Discrete Fourier Transform of the product of the DFTs of the input and impulse response:

$$y(n) = IDFT\big(DFT(x) \cdot DFT(h)\big)$$

The input and impulse response lengths must be the same size to ensure that the correct frequency bins of the DFT are being multiplied. Thus, the input length, N_x, and impulse response length, N_h, must be made equal. To preserve the same output length, N_y, that the direct linear method produced, the input length must be padded with $N_h - 1$ zeros, while the impulse response length must be padded with $N_x - 1$ zeros. This will ensure that the output of the direct linear and direct frequency convolution processes is identical.

This method requires performing two $N_x + N_h - 1$ DFTs, $N_x + N_h - 1$ complex multiplications, and 1 $N_x + N_h - 1$ IDFT. If the input and impulse response are padded with enough zeros to make $N_x + N_h - 1$ a power of 2 the Fast Fourier Transform (FFT) algorithm can be utilized. This will reduce the required computations to less than 1% of those needed for the direct linear operation.[27] Direct Frequency convolution depends on the complete definition of the input signal before processing, thus, it cannot be used in a real-time, or quasi real-time application.

2.1.2.3 Overlap and Add

This technique processes small blocks of the input data with the full impulse response in the frequency domain, and uses the FFT to drastically reduce computational requirements. The selected input blocks are non-overlapping, sequential audio samples. For direct linear convolution, the output is expected to be the length of the input (N_x) plus the length of the impulse response (N_h) minus *1*. For example, consider the input signal and impulse response in Figure 2.5.

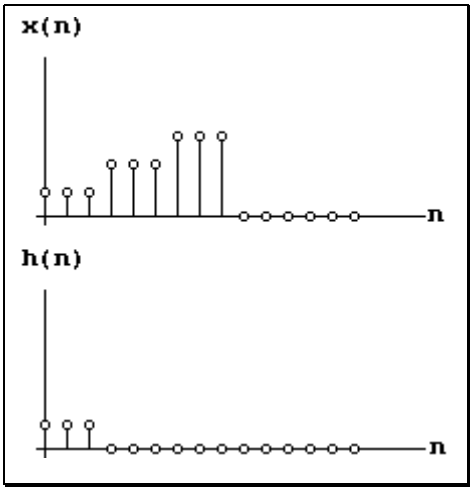

Figure 2.5: Overlap and add example (input and impulse response)

The input length is 9 and impulse response length is 3, therefore the length of the convolution should be 9 + 3 –1 = 11. By choosing the input block size of 3, several short convolution operations can be executed instead of one lengthy computation. The output blocks will overlap by 3 – 1 = 2 samples and the final output can be found by summing

the overlapping samples (Figure 2.6 and Figure 2.7)

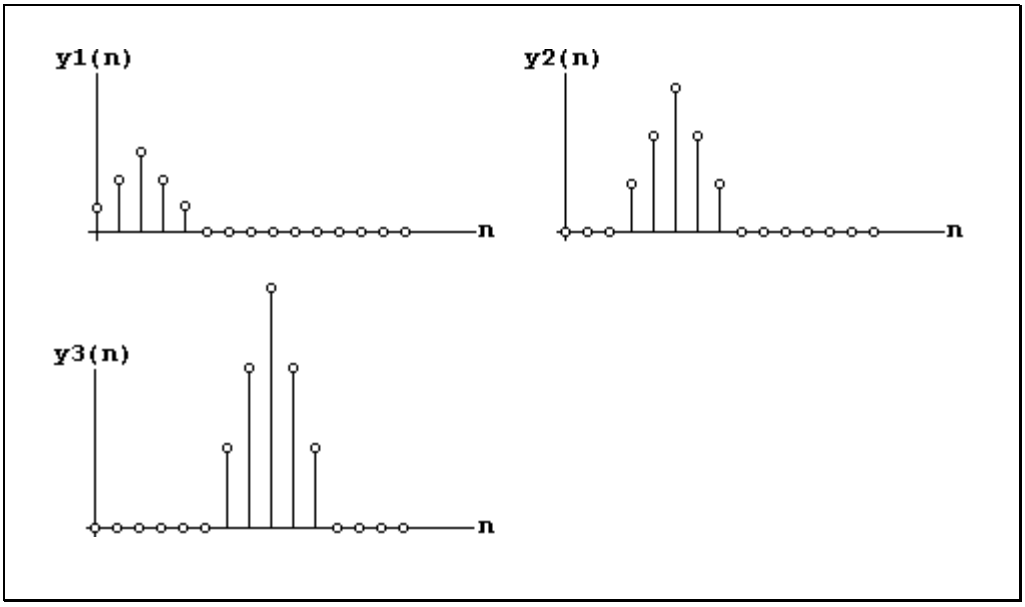

Figure 2.6: Overlap and add example (output blocks)

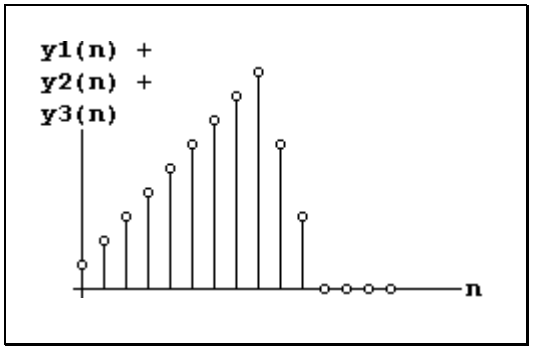

Figure 2.7: Overlap and add example (sum overlapping blocks)

When selecting the length of the input block, we must zero pad the input and impulse response to maintain this output length when we multiply in the frequency domain and perform the IFFT. If using radix-2 FFTs, the FFT length (N_{FFT}) is chosen to be the next highest power of two that is greater than $N_h.N_x$ is then determined according to:

$$N_x = N_{FFT} - N_h + 1$$

The input block is then zero padded up to N_{FFT}. This provides enough room for the convolution expansion.[25] Each output block overlaps the next by a length of $N_h - 1$ and must be summed to maintain exact congruence with linear convolution.

Even though this form of block convolution requires 2 FFTs, 1 IFFT, and N_{FFT} multiplications, the efficiency of the FFT algorithm allows this method to outperform linear convolution. However, no output samples are calculated until the first N_x input samples are collected and processed. If a strict real-time application is needed, this method will not satisfy; however, if the application can tolerate modest I/O delay, this is a much more efficient alternative.

2.1.2.4 Overlap and Save

The Overlap and Save block convolution method is similar to the Overlap and Add method except the input blocks are selected to overlap and the output blocks are not. When the input blocks are processed using circular convolution, the first $N_h - 1$ samples are discarded, and each block is concatenated to form the total output.

Consider the same example (Figure 2.8).

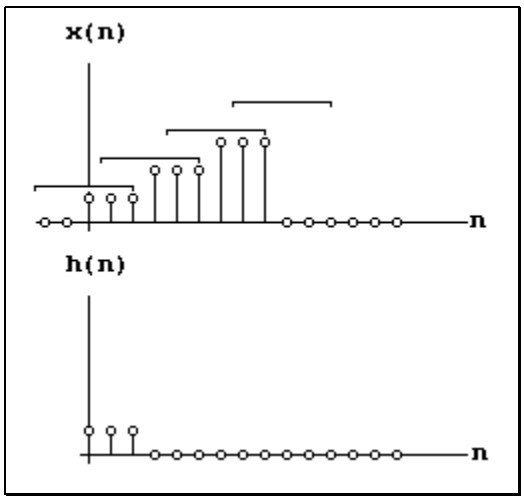

Figure 2.8: Overlap and save example (overlapping input blocks and impulse response)

By selecting overlapping input blocks, the need for summing the output blocks is eliminated. The circular convolution artifacts, which show up as the first two samples, can be eliminated in each of the output blocks (Figures 2.9).

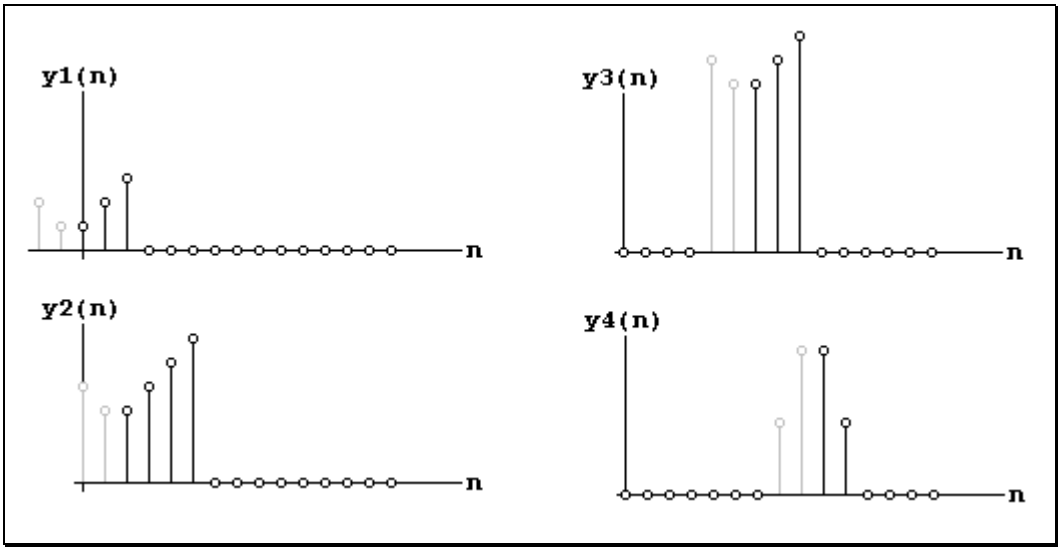

Figure 2.9: Overlap and save example (output blocks)

The output blocks can then be concatenated to form the full output (Figure 2.10).

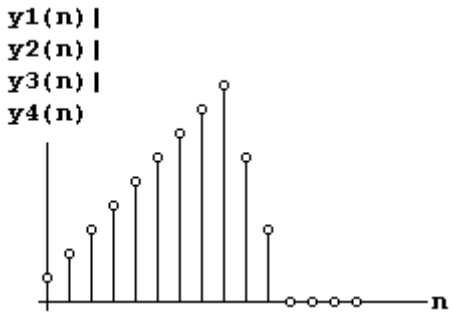

Figure 2.10: Overlap and save example (concatenated output)

This method still imposes an I/O delay, but does not have the accumulation step for the overlap sections like the Overlap and Add method.

2.1.2.5 Minimum Delay Block Convolution

William Gardner developed a block convolution algorithm that combines the real-time architecture of linear convolution with the computational efficiency of block convolution. This hybrid architecture uses a modest FIR direct convolution stage to ensure output samples will not be delayed while input samples are being collected for the block convolution operation.[10]

2.1.3 Pertinent Transforms

Although real-time audio is a time-based phenomenon analysis based on time characteristics alone often do not suffice. The most common alternative analysis is frequency-based due to the inherent pitch qualities of audio signals. This analysis is achieved using the Fourier Transform, a mathematical operation relating an analog waveform to its analog sinusoidal components:

$$X(\Omega) = \int_{-\infty}^{\infty} x(t) e^{-j\Omega t} dt$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\Omega) e^{j\Omega t} d\Omega$$

For digital signals, the Discrete Fourier Transform (DFT) is used:

$$X(f) = \sum_{0}^{N-1} x(n) e^{-j 2\pi f n / N}$$

$$x(n) = \frac{1}{N} \sum_{0}^{N-1} X(f) e^{j 2\pi f n / N}$$

Because the transform's kernel is a sinusoid ($e^{-j2\pi fn/N}$) the output of this transform reveals the level of each sinusoidal component within the original signal.

The Fast Fourier Transform (FFT) is a method of calculating the DFT of a signal in a more efficient way. If N is length of a digital signal, direct application of the DFT will require N^2 operations. However, by exploiting the symmetries of the DFT, the FFT requires $N\log_2 N$ operations.[23] A 30 second audio signal sampled at 44.1kHz will require 1.75 trillion operations to calculate the DFT and only 26.9 million operations to calculate the FFT (0.001% of the required DFT operations). The FFT is used in calculating the early reflection portion of the hybrid reverberation.

While the Fourier domain produces a familiar audio characteristic, it is not the only useful analysis transform. A transform with a different kernel will reveal how the signal is composed of this new kernel. The Hadamard Transform uses a kernel made of only +1 and –1. Its matrix implementation restricts it to only digital signals. [17]

Figure 2.11 shows an $N = 8$ Hadamard kernel.

$$T = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

Figure 2.11: Hadamard kernel *(N = 8)*

2.2 Artificial Reverberation: filtering

Only a few simple reverberating building blocks make up the basic library of digital reverberator circuits. The inverse comb filter (FIR), comb filter (IIR) and all-pass filter (IIR) are the basic structures that have been combined in different ways in an attempt to imitate the effects of various rooms. Figure 2.12 shows the measured waveform and the frequency response of the dense diffuse reverberation of the Bergamo Cathedral in Italy. It is this characteristic that is the goal of the filtering techniques explores in this chapter.

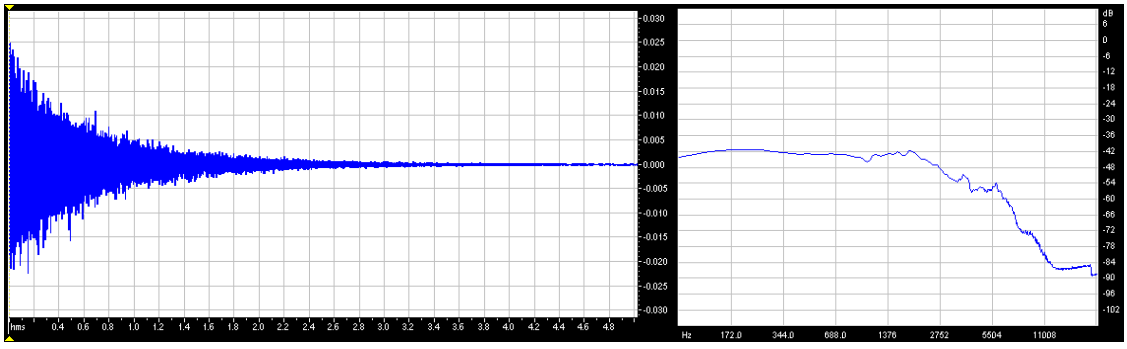

Figure 2.12: Diffuse reverberation waveform and frequency response

2.2.1 Inverse Comb Filter

Filtering techniques are used to simulate convolution with an acoustic impulse response. The simplest building block of an artificial reverberator sums every incoming sample with a delayed and attenuated version. This is accomplished with a simple delay element in a feedforward architecture. This FIR reverberator is shown in Figure 2.13.

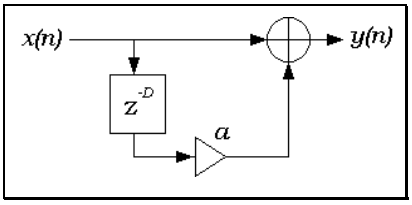

Figure 2.13: FIR reverberator

If the feedforward coefficient is less than 1, this circuit will output the input plus a delayed and attenuated version of the input. Note, however, that regardless of the value of this coefficient, the system remains stable. This topology has difference equation:

$$y(n) = x(n) + ax(n - D)$$

and transfer function:

$$H(z) = 1 + az^{-D}$$

and impulse response:

$$h(n) = \delta(n) + a\delta(n - D)$$

The frequency and impulse response of a simple FIR reverberator is shown in Figure 2.14. This filter topology is known as an "inverse comb filter" because the notches in the frequency response resemble the teeth of an upside down comb.

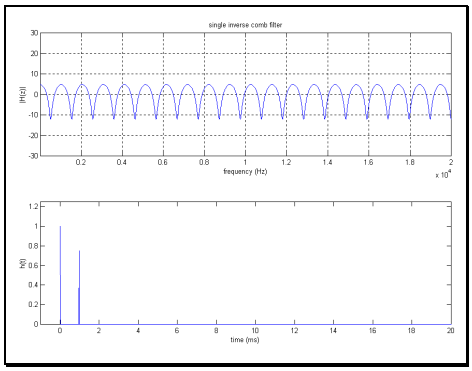

Figure 2.14: Inverse comb reverberator (frequency and impulse response)
1ms delay, 75% feed-forward gain

In the z-plane (shown in Figure 2.15), this filter topology has equally spaced zeros around the unit circle with a radius specified by the attenuation constant *"a."* The delay time (in samples) will determine the amount of zeros spaced from 0 to 2π and these zeros will create nulls in the magnitude response of the filter causing coloration of the signal.

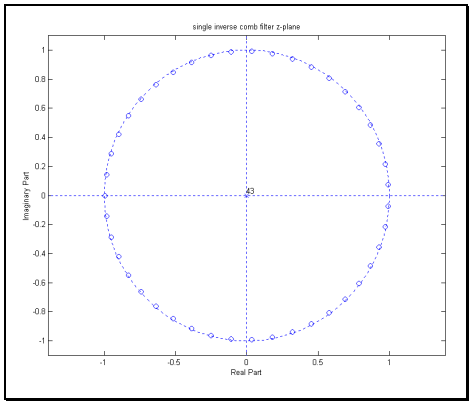

Figure 2.15: Inverse comb reverberator (z-plane)

This filter, as well as parallel and cascade combinations, has a finite length impulse response and can be guaranteed to be stable regardless of the magnitude of *"a."*

However, room impulse responses consist of very dense series of echoes that cannot be practically realized using this architecture.

2.2.2 Comb Filter

In room acoustics, sound is reflected off of all of the walls in an enclosed space. A sound wave reflecting off of a ceiling at a given angle, will also reflect off another wall, and still another, etc. This acoustic phenomenon is not accurately represented by the FIR delay structure in Figure 2.13. It can be modeled by feeding the output back through a delay element and summed with the input. This structure is realized in the block diagram in Figure 2.16.

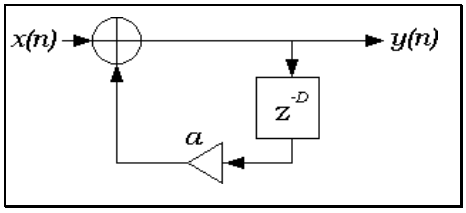

Figure 2.16: IIR reverberator

This topology has difference equation:

$$y(n) = x(n) + ay(n - D)$$

and transfer function:

$$H(z) = \frac{1}{1 - az^{-D}}$$

and impulse response:

$$h(n) = \delta(n) + a\delta(n - D) + a^2\delta(n - 2D) + a^3\delta(n - 3D) + \ldots$$
$$= \sum_{m=0}^{\infty} a^m \delta(n - mD)$$

This filter topology is known as a “comb filter” because the peaks in the frequency response resemble the teeth of an upright comb (Figure 2.17).

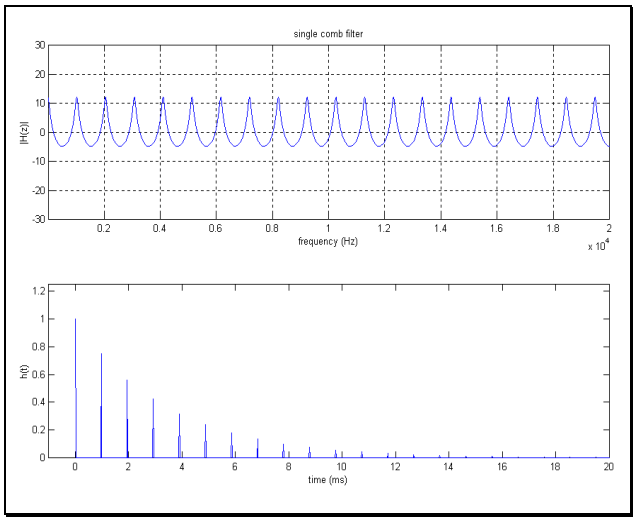

Figure 2.17: Comb reverberator (frequency and impulse response)
1ms delay, 75% feedback

Each output sample of this filter, *y(n)*, consists of the sum of the input sample, *x(n)*, and a delayed and attenuated version of the output sample *D* samples prior. Of course, the output *D* samples prior is equal to the input sample at that time plus the output sample *D* samples prior.

This topology has equally spaced poles around the unit circle with a radius specified by the attenuation constant *“a.”* These poles will create peaks in the magnitude response of

the filter causing coloration of the signal. The z-plane plot is shown in Figure 2.18.

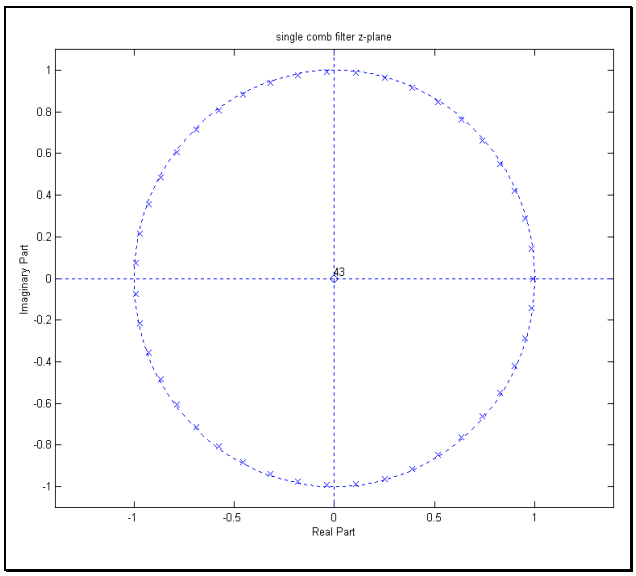

Figure 2.18: Comb reverberator (z-plane)

Stability is the chief concern with this filter type. If the coefficient "*a*" is greater than 1, the poles will be outside of the unit circle and the filter will overload. Audibly this will cause severe clipping, often shutting down the application using the algorithm. If the coefficient "*a*" is equal to 1 (poles on the unit circle), the filter will oscillate.

The heavy peaking in the frequency response of the comb filter does not provide a realistic sounding reverberation. Comb filters are also susceptible to flutter echoes. These are distinct echoes that arrive over 25ms in time without significant attenuation. The audible effect of flutter echoes is a sharp metallic sound that is heard in a non-absorbent room with parallel walls.[2]

2.2.3 All-Pass Filters

Reverberation is generally used for its time-based effect, not to change the frequency

response of the input signal. Both the "inverse comb" and "comb" filters have poles and zeros in their transfer functions which alter their frequency response. To maintain the overall frequency content of the input and still take advantage of the time characteristics of the reverberation algorithm, an "all-pass" alternative structure can be used. In the z-plane (Figure 2.19), an all pass filter has poles and zeros at the same frequency (angle) and at inverse radii such that the effects of each cancel:

$$r_z = \frac{1}{r_p}$$

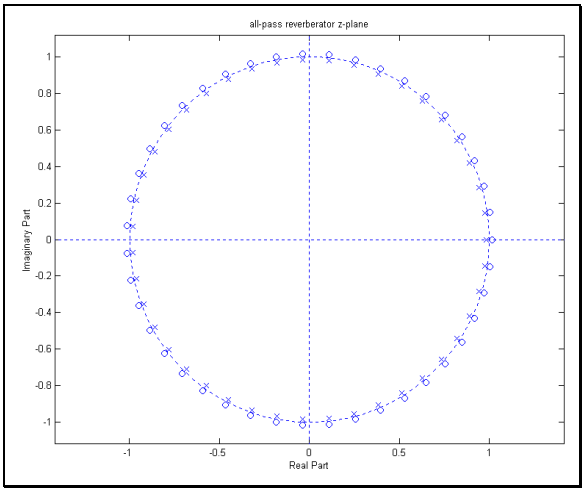

Figure 2.19: All-pass reverberator (z-plane)

The all-pass filter accomplishes its unique pole/zero distribution through a feedback and feedforward path as shown in Figure 2.20.

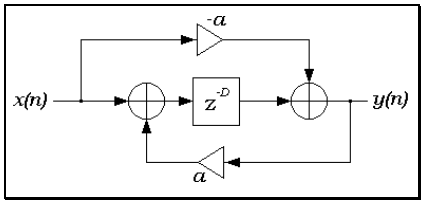

Figure 2.20: All-pass reverberator

It is this combination that secures an overall flat frequency response (Figure 2.21).

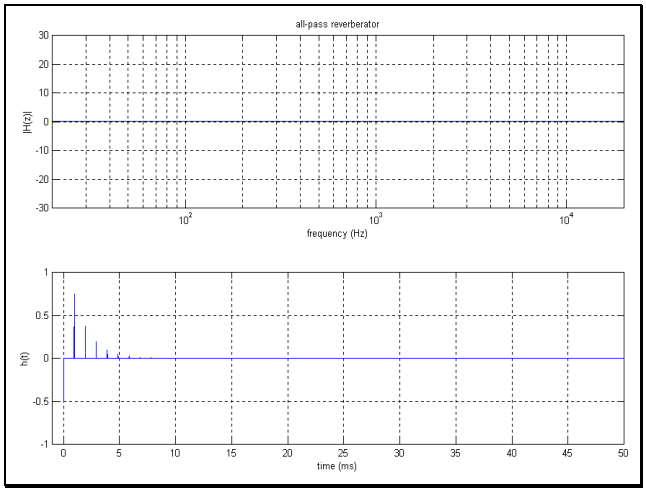

Figure 2.21: All-pass reverberator (frequency and impulse response)
1ms delay, 50% feedback

The all-pass reverberator has difference equation:

$$y(n) = -ax(n) + x(n-D) + ay(n-D)$$

and transfer function:

$$H(z) = \frac{-a + z^{-D}}{1 - az^{-D}}$$

and impulse response:

$$h(n) = -a\delta(n) + (1-a^2)\delta(n-D) + a(1-a^2)\delta(n-2D) + a^2(1-a^2)\delta(n-3D) + \ldots$$
$$= -a\delta(n) + (1-a^2)\sum_{m=1}^{\infty} a^{m-1}\delta(n-mD)$$

While the magnitude response of the all-pass filter is overall flat, the phase response is not. This filter will pass all frequencies equally over the course of the entire input, however this does not apply to short time intervals or transient-type sounds. Just like the comb and inverse comb, the all-pass also has a distinct and recognizable timbre.[20]

2.2.4 Implementation

Neither the all-pass, comb, nor inverse comb filter described above has the echo or frequency density to properly simulate room reverberation on its own. However, they can be used as building blocks to combine and create suitable artificial reverberation.

Figures 2.22 and 2.23 show Schroeder's combinations of comb and all-pass filters for the purposes of modeling room reverberation.[20]

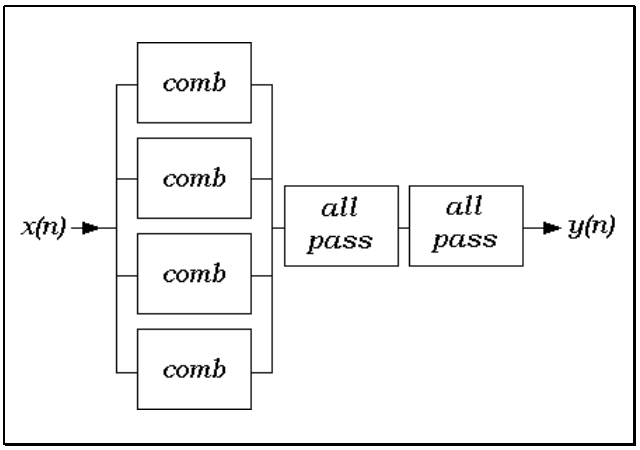

Figure 2.22: Schroeder reverberator (combination 1)

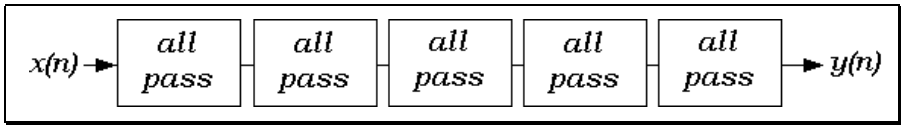

Figure 2.23: Schroeder reverberator (combination 2)

The series all-pass configuration (Figure 2.23) provides a buildup of dense reflections, and maintains a flat overall frequency response. However, this combination still suffers from the same transient coloration that the single all-pass component does. The parallel combination of comb filters with two all-pass series components (Figure 2.22) allows for a more dense buildup of echoes and when each of the comb filter poles are given the same magnitude, the frequency coloration of the individual filters is masked.[18] The

addition of the series all-pass filters allow for further density buildup without adding general timbre changes.

Moorer suggested a slight improvement to the Schroeder scheme by having 6 parallel comb filters, each with a single pole low-pass filter in the feedback loop (Figure 2.24).

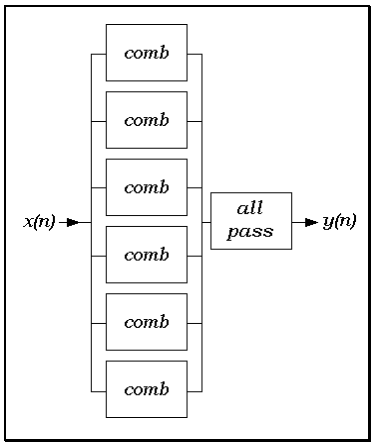

Figure 2.24: Moorer reverberator

The purpose of this filter is to simulate the high frequency attenuation produced by air and materials absorption. Extensive experimentation found that nothing was gained by using more complicated filters in the feedback loop; the 1st order filters function well in smearing the echoes of more impulsive input sounds. The addition of a filter in the feedback loop of the all-pass filter did nothing to improve the performance of the overall structure.[20] The filter in the feedback loop will increase high frequency rolloff each iteration, which is consistent with room acoustics. By setting the cutoff frequency at 12kHz initially the effect will be subtle and will dynamically filter the high

frequencies.[1] Figure 2.25 shows the impulse and frequency response of a single comb filter with 1ms delay and feedback coefficient value of 0.85 with a single pole low-pass filter (3kHz cutoff frequency) in the feedback path:

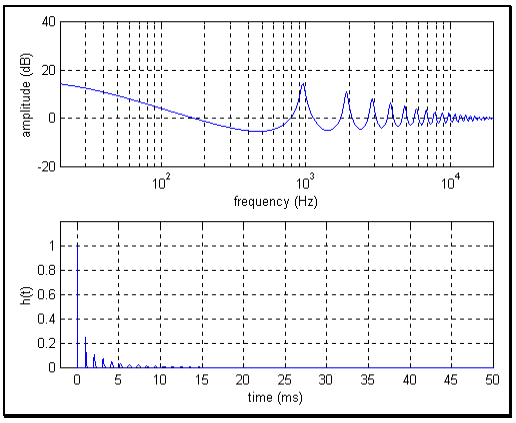

Figure 2.25: Single comb filter with LPF in feedback loop

The addition of the low-pass filter in the feedback loop will migrate the pole/zero distribution in the z-plane, increasingly collapsing down towards the origin (Figure 2.26).

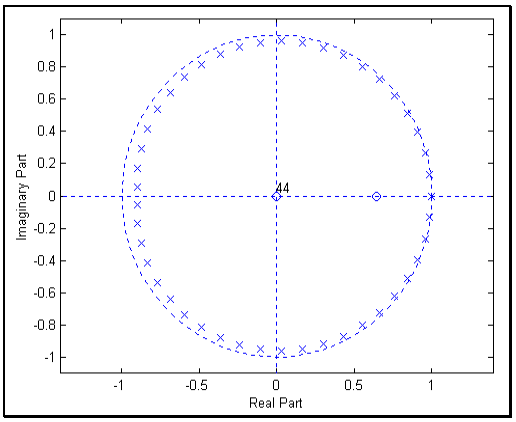

Figure 2.26: Single comb filter with LPF in feedback loop (z-plane)

The parallel comb structures that Schroeder and Moorer proposed can provide dense late echo characteristics only when delay times are selected to be as close to a prime ratio as

possible. Violating this will cause discrete echoes to overlap creating noticeable peaks and/or nulls in the frequency response.

For memory considerations, the large banks of parallel comb filters (Figures 2.22 and 2.24) may quickly saturate the host platform resources because each comb filter will require a separate delay line. An implementation similar to this parallel structure is the multi-tap comb filter (Figure 2.27).

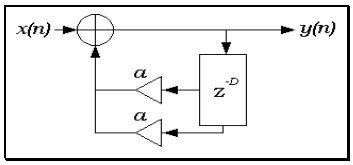

Figure 2.27: Two tap comb filter

This filter can have several outputs from a single delay line. The impulse will include the sum of the impulse responses of each tap, as well as "cross-tap delays." These delays are those created by each tap due to the outputs of the other taps. For example, if $tap_1 = 3$ samples and $tap_2 = 5$ samples, the impulse response will include echoes spaced 3 samples apart due to each of the tap_2 outputs and echoes spaced 5 samples apart due to each of the tap_1 outputs (Figure 2.28).

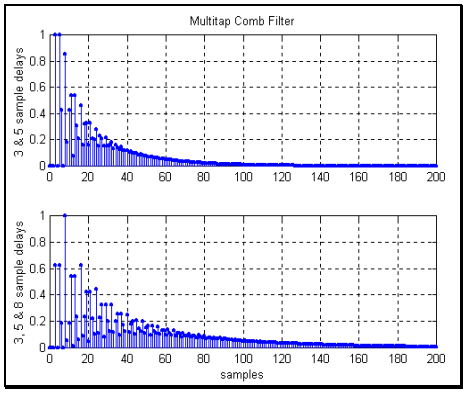

Figure 2.28: Multi-tap comb filter

This architecture allows one multi-tap delay structure to generate a denser reverberation tail than its multiple parallel counterpart. To maintain output stability the feedback values must be selected so that the sum of the absolute values of each tap's feedback gain is less than 1.[23] Delay times are selected to be relatively prime to minimize the amount of overlapping contributions from delay taps, but despite the memory advantages of the multi-tap structure, the limited range of stable feedback values prevents this structure from producing natural sounding reverberation.

The inevitable overlapping will create unnatural discrete echoes in the later reverberation tail [15] and high frequency anomalies as shown in the frequency response plots of Figure 2.29.

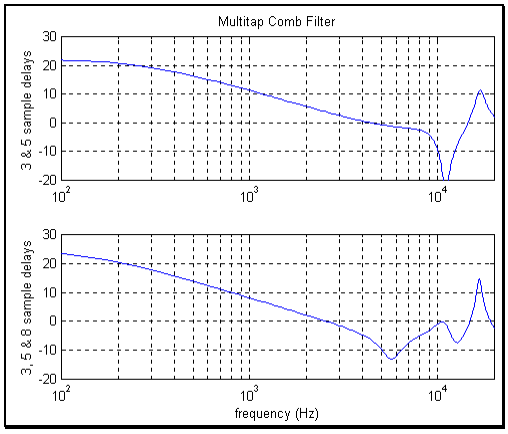

Figure 2.29: Multi-tap comb filter (frequency response)

While the multi-tap topology utilizes many outputs and few delay lines, the feedback matrix topology uses many delay lines and single outputs.

This topology is shown in Figure 2.30.

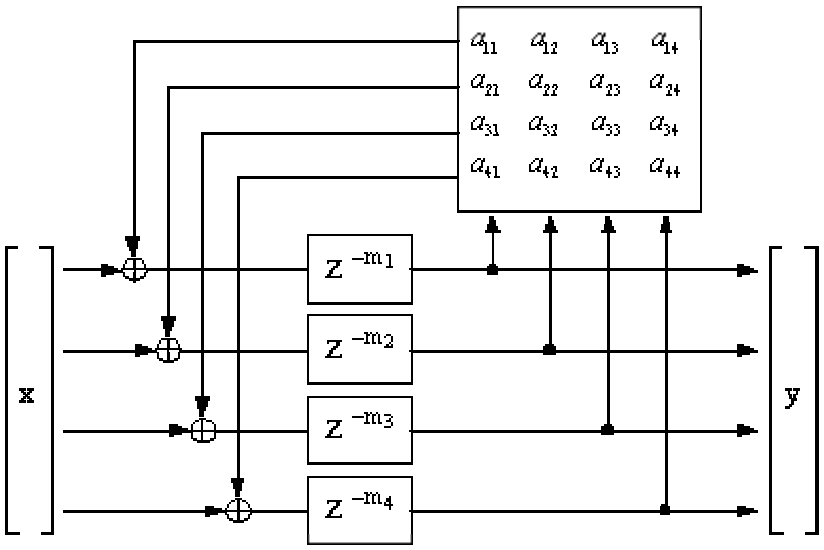

Figure 2.30: General feedback matrix topology

Extremely dense echoes are built up when the outputs of each delay line are fed back and mixed with the inputs of all of the delay lines. The outputs of the delay lines are mixed according to a coefficient matrix. [18]

2.3 Measuring Reverberation Quality

Methods have been developed that assess the quality of artificial reverberation. Acoustic impulse responses are confined to the ISO3382 measurement parameters (section *2.1.1.2*). Recursive filter networks can also utilize this measurement system by implementing the structure, exciting it with an impulse and evaluating the audio output. However, these mathematically defined networks can have easily derived formulas assessing reverberation properties. The Time and Frequency Densities have been adopted as the staple quantitative measurement of these reverberation systems.

Unfortunately, only the single comb filter and the network of parallel comb filters have had these quantities defined mathematically.

2.3.1 Frequency Density

The Frequency Density, or Modal Density, is defined as the number of eigenfrequencies (frequency peaks) per Hertz. For predictable topologies mathematical relationships have been developed. For example, the frequency density of a network of parallel comb filters is:

$$D_f = \sum_{p=0}^{P-1} \tau_p$$

Frequency Densities greater that 0.15 peaks per Hertz are accepted for adequate reverberation.[18] For the Moorer topology of Figure 16, the parallel comb filter bank has delay times of 50ms, 56ms, 61ms, 68ms, 72ms, and 78ms. The Frequency Density is 0.385, greater than the needed amount.

Insufficient Frequency Density can be heard as a ringing of particular frequencies when excited by an impulse-like input, or a predominance of particular frequencies with a steady-state input.

2.3.2 Time Density

The Time Density, or Echo Density, is defined as the number of reflections per second. Similar to Frequency Density, a mathematical relationship has been developed for predictable known topologies. The Time Density of a network of parallel comb filters is:

$$D_t = \sum_{p=0}^{P} \frac{1}{\tau_p}$$

Time Densities greater than 10,000 are necessary for adequate reverberation.[18] For the Moorer topology, the Time Density is 95.67, far less than the 10,000 echoes/second needed. However, this does not account for the low-pass filter in each of the comb filter feedback loops or the series all-pass filter. Both of which are known to increase time and frequency densities.

2.3.3 Energy Decay Relief

Schroeder developed a measurement technique called the Energy Decay Curve. This provided a visualization that would display how the frequency response changed over time (or how the waveform changed over different frequencies):

$$EDC(t) = \int_{t}^{\infty} h^2(\tau) d\tau$$

Jot [19] altered the EDC to form the Energy Decay Relief (EDR) which was a 3 dimensional visualization of the time and frequency characteristics of a signal. Figure 2.31 shows the Energy Decay Relief of a single comb filter.

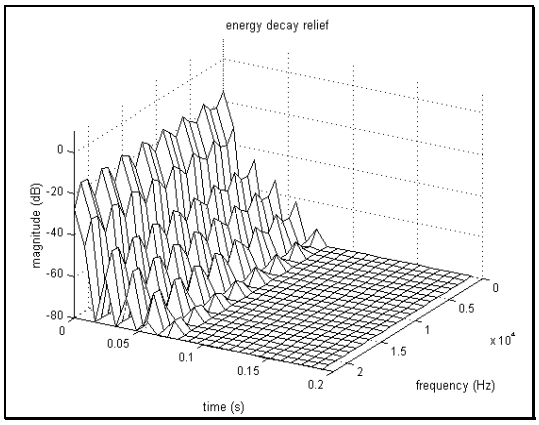

Figure 2.31: EDR of single comb filter

The discrete line of echoes or evident along the time axis while the notches are evident along the frequency axis.

Figure 2.32 shows the Energy Decay Relief of an all-pass filter. The same line of discrete echoes are evident along the time axis, however the flat overall frequency response can be seen as well.

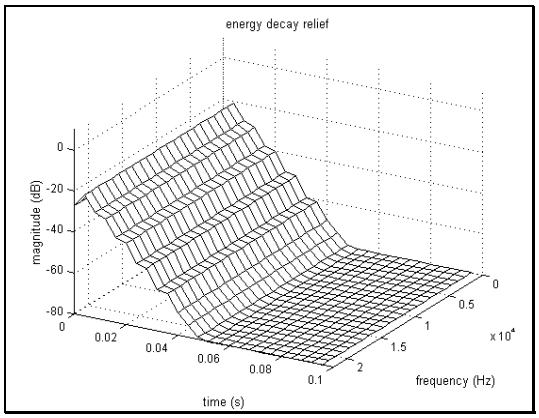

Figure 2.32: EDR of single all-pass filter

3. Hybrid Model Introduction

The hybrid model implementation will combine the two general techniques explored in Chapter 2. By utilizing an acoustic impulse response we can extract the very important early echoes and frequency response envelope. The output of this portion can be determined using a convolution operation while a general comb, inverse comb, and/or all-pass generic network can be used to simulate the reverberation tail portion. A block diagram describing this process is shown in Figure 3.1.

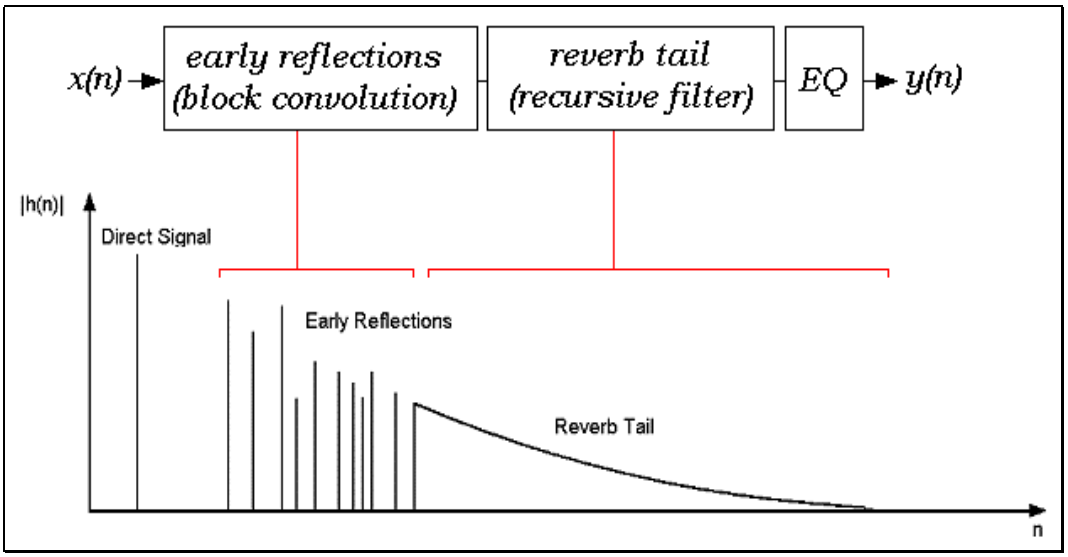

Figure 3.1: Hybrid model

3.1 Implementation

Both Moorer [20] and Moore [21] have suggested using an FIR filter to model the early echoes and a recursive topology for the late reverberation section. These designs rely on a careful selection of FIR coefficients that best simulate generic early echoes. This design does not use a measured impulse response and lacks the ability to simulate a specific room. In addition, a practical FIR length will not capture the early echo nuances and high order FIR filters can quickly saturate the resources of a practical processor.

3.1.1 Impulse Response Truncation

The notion that the impulse response can be truncated is based on "temporal fusion." This phenomenon occurs when the human biochemical auditory system can not discriminate between discrete incoming sound events. A sound event occurring less than 30ms - 50ms after another event will not be distinctly heard. Griesinger describes sound occurring in this time as being assigned to the previous sound event by the brain. [16] For this reason the portion of an impulse response containing discrete echoes greater than 50ms apart must be kept; this is the early echo response that can be discretely heard by the listeners. The portion where the discrete echoes have been diffused so they occur less than 30ms apart is the reverberation tail. It is when the impulse response decays to this point that the convolution stage can be abandoned in favor of a recursive model.

Two different justifications have been explored in determining where a given impulse response can be truncated. Where in time this is to occur will make a significant impact on the reverberation accuracy and the computational requirement.

3.1.1.1 Time Based Truncation

Jot describes the first reflections (approx. first 80ms) as the most crucial element of an impulse response providing the room's spatial impression. The late reverberation (reverberation tail) is independent of listener position and can be modeled in a more generic fashion.[18] The research of both Farina [8] and Borish [1] has found the most distinct portion of a typical room impulse response to be between 50ms – 150ms. The influence of temporal fusion will prevent listeners from resolving the short reflections

occurring after this time. Farina suggests that an inverse relationship exists between the crucial portion of a room impulse response and the room's size; larger room impulse responses could be truncated earlier than smaller rooms.[6] For rooms with long reverberation times, it is the lengthy "statistical" portion of the impulse response that covers the details of the early reflections.

3.1.1.2 Windowing

Truncating the impulse response, like any signal, will introduce time and frequency distortions that need to be restored to achieve transparency. The recursive late reverberation component of the algorithm will provide the time correction, and an equalization component will voice the output to compensate for the frequency coloration caused by truncation.

The truncation process can be executed in many different ways. When we simply decide to keep a certain amount of samples and set the rest to zero, we are actually applying a rectangular window to the original signal. If the point of truncation falls on a non-zero sample value, the sharp change to zero will introduce a high frequency component, this is known as "frequency leakage." This is illustrated in the magnitude spectrum of a rectangular window in Figure 3.2.

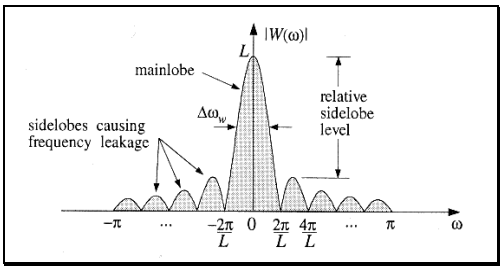

Figure 3.2: Magnitude spectrum of rectangular window

The “frequency leakage” occurs in the sidelobe portion of the response. The effects can be minimized by using other windows that do not have sharp transitions to zero, and minimize the sidelobe presence.[23] We want to be careful to keep as much of the impulse response intact while not introducing extra high frequency information during the window (truncation) process. Figure 3.3 shows three common window functions (rectangular, Hamming and Blackman) in both time and frequency.

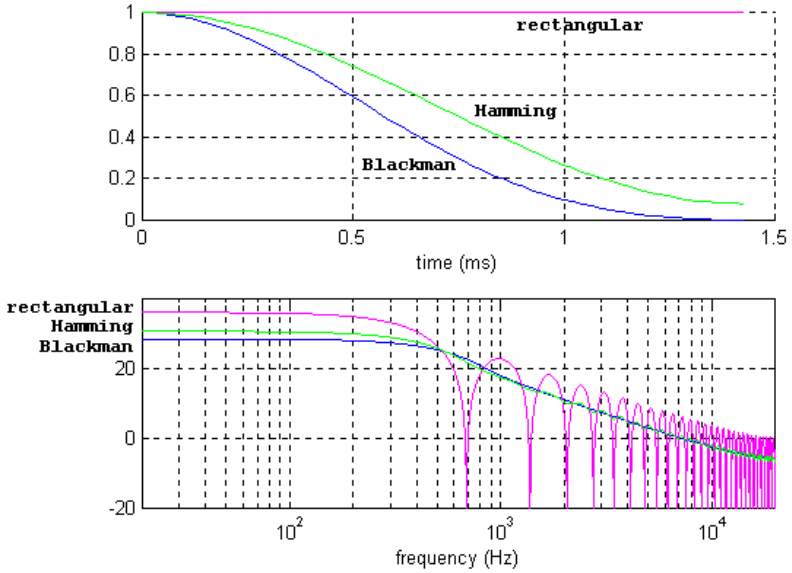

Figure 3.3: Three 64-sample long window function

3.1.1.3 Equalization

The high frequencies of an acoustic impulse response spectrum decay much quicker than the low frequency components due to absorption by wall material, room contents, and air. This effect is modeled in the reverberation tail by the 1st order feedback filters. However, by truncating the impulse response a significant portion of the signal (containing low frequencies) has been discarded.

With the majority of the signal discarded a new tonal balance is created (Figure 3.4).

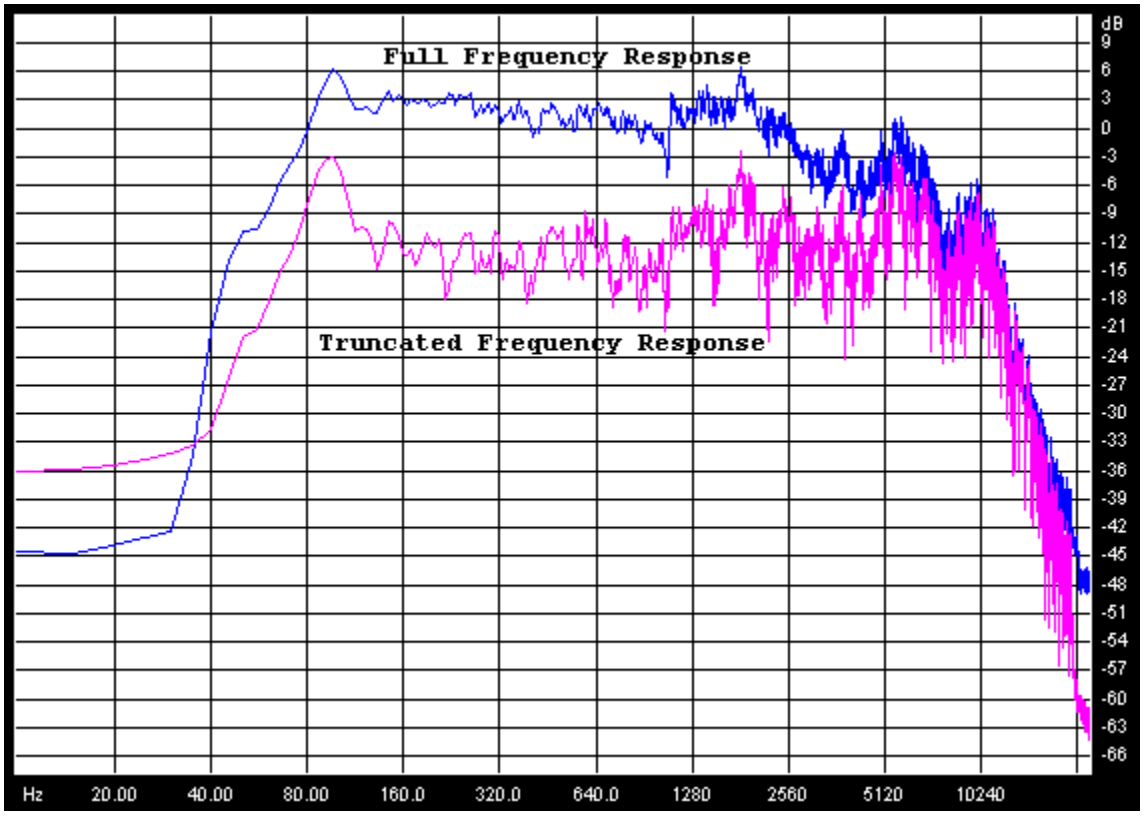

Figure 3.4: Full and truncated frequency response or medium hall impulse response

Since the first portion of the response is kept, thereby retaining the high frequency signature of the room, equalization must be performed to ensure adequate frequency balance. Because this is a linear system, the equalization can occur within one of the processing stages (convolution or the recursive reverberation tail) or a separate pre- or post-equalization (EQ) component.

4. Hybrid Model Design

The purpose of this algorithm is to produce realistic artificial reverberation without the burden of impractical processing. However, impractical design of the building blocks can become the weakness. The success of this design relies on the practical implementation of each of the components:

- truncation method (time or energy based)
- convolution
- windowing
- reverberation tail structure
- frequency equalization

4.1 Truncation

Both methods of impulse response truncation were implemented (*section 3.2.1*). No significant output difference was noticed, however processor performance was eased greatly when using the time-based method. This allowed for the static truncation time of 150ms (6615 samples) and the use of 8192 point FFTs for the block convolution stage. Because of this modest FFT demand, the processor (500 MHz Intel Celeron) was able to execute the convolution portion in less than one sample period and no additional latency was experienced.

Figure 4.1 shows the waveforms of the two full and truncated impulse responses that are used in the testing section (Chapter 5). These signals are in the WAV format and are designated as *IR1.wav* and *IR2.wav*.

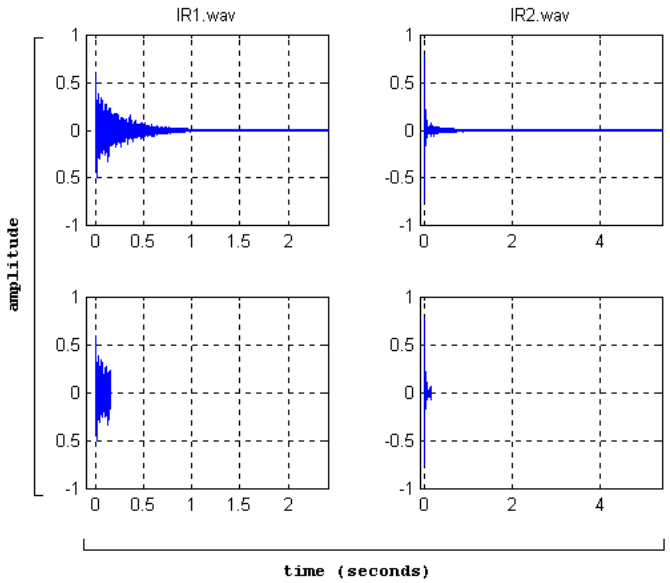

Figure 4.1: Original and truncated impulse responses

4.2 Convolution

The direct FIR implementation is not a practical realization for this reverberation process. Typical room impulse responses are too lengthy and sampled too fast for current processors to execute in a linear convolution. When this hurdle will not permit real-time functionality, quicker, low-latency alternatives are readily available. The truncated impulse response will generally be greater than 100 ms (4410 samples at f_s of 44.1 kHz). The Overlap-Add method will introduce an initial input latency when gathering samples, but a quasi real-time process can be realized if the block convolution can be performed in less than one sampling period.

The convolution stage is used to implement the early reflection portion of the reverberation effect. By using the Overlap-Add method for the convolution stage, the early reflection portion of the hybrid algorithm will be exactly the same as that of the linear convolution. In Figures 4.2 and 4.3 a simple percussion signal is convolved with a full and truncated impulse response and the "residue" is the difference between these two outputs. With the early reflection portions of the residue plots equal to zero, it is evident that the truncated and full convolution processes are identical in this region.

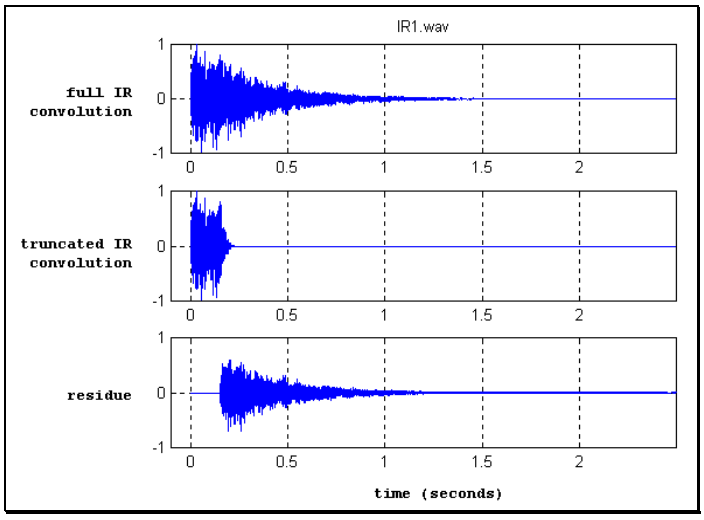

Figure 4.2: Full and truncated impulse response convolution (*using IR1.wav*)

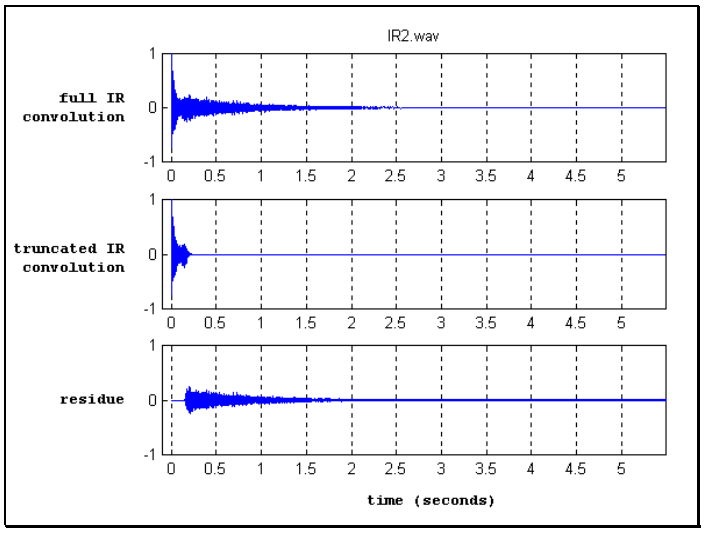

Figure 4.3: Full and truncated impulse response convolution (*using IR2.wav*)

4.3 Windowing

The generally exponential decay of the impulse response describes its envelope only. The true decay is extremely jagged and random. Thus, ensuring a smooth truncation with a windowing function has little effect on these types of signals. In fact, the effect of any windowing only served to add to the computational expense with negligible effect on the high frequency content of the truncated response. Of much greater importance is the low frequency attenuation caused by this premature truncation.

4.4 Reverberation Tail

The Moorer diffuse reverberator exhibits the time and frequency density requirements of a proper room reverberation and can be used to simulate different sized rooms simply by adjusting the feedback control and relative gain of the filter network. In addition, the Moorer topology includes feedback low pass filters to account for the air and material absorption of the room. While prior research has found that a first order low-pass filter with f_c set to 12kHz is suitable for modeling a general room [1], it is not sufficient for modeling a specific room. Impulse responses can be measured in rooms with drastically different absorptive materials and humidity conditions. Therefore, using a fixed cutoff frequency does not apply. The cutoff frequency of the low-pass filters was estimated through listening and visual inspection of the frequency rolloff.

4.5 Equalization

As stated in section *3.2.1*, an EQ component is needed to compensate for the low frequency attenuation of the truncation process. Figure 4.4 depicts the frequency response of a full impulse response and its 150 ms truncated counterpart.

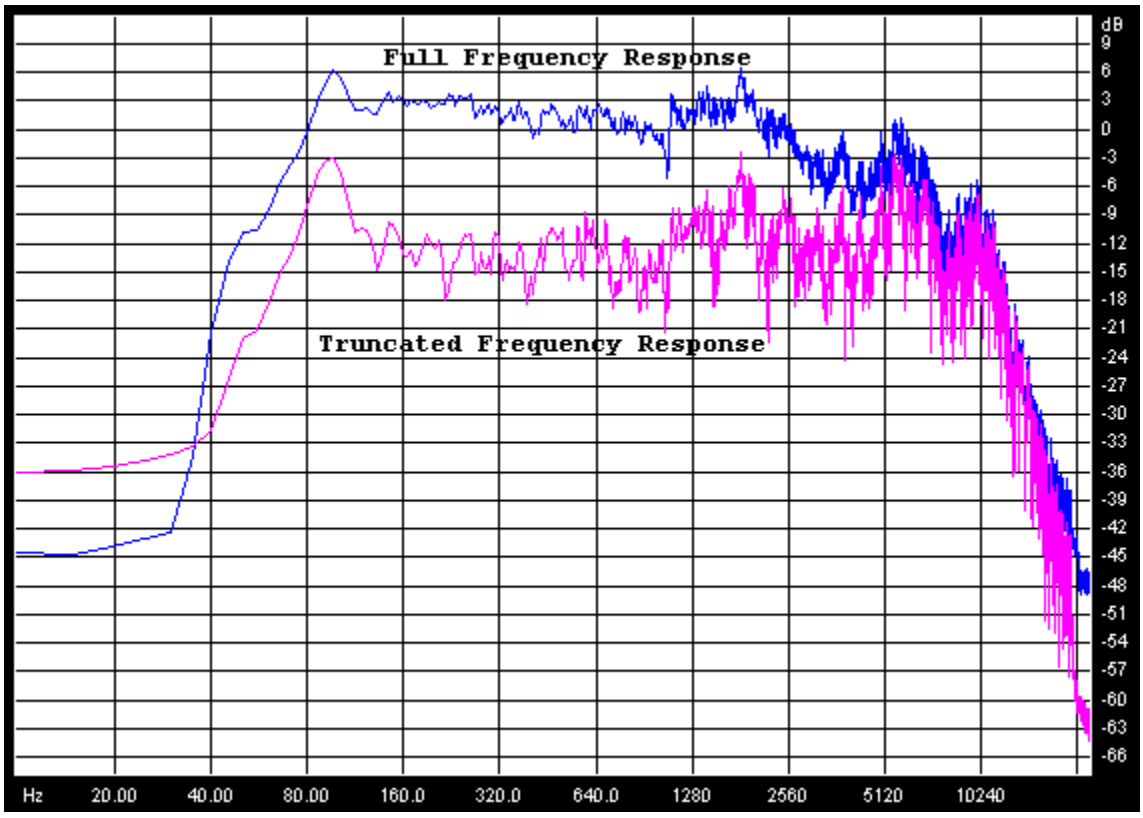

Figure 4.4: Full and truncated frequency response of Bergamo Cathedral, Italy

The low frequency attenuation is as much as 15 dB at some frequencies, and must be corrected to achieve transparency. This compensation can come experimentally by employing common low frequency boosting filters, or by an adaptive means derived from the spectrum of the entire impulse response.

The EQ stage should be equal to the ratio of the full impulse response spectrum to the truncated impulse response spectrum so that the effects of the truncation cancel.

$$Y(z) = X(z) \cdot H(z) \cdot EQ(z)$$

$$EQ(z) = \frac{H_{full}(z)}{H_{truncated}(z)}$$

In practice, the lengths of each spectrum must be equal to maintain accurate frequency equalization. The spectrums *X(z)*, *Y(z)*, and *H(z)* all have length 8192 (2^{13}) samples. For memory preservation the full impulse response is limited to 5.9 seconds (2^{18} samples). To maintain frequency domain accuracy, the original impulse response FFT will also be 2^{18} samples long. A simple decimation algorithm is employed to reduce this spectrum to the 8192 length by keeping every 32nd frequency bin sample and discarding the remaining. While this will reduce the frequency resolution of the signal from 0.17 Hz/bin to 5.38 Hz/bin, the envelope of the response will remain intact and can be used to calculate the EQ stage.

This decimation eliminates the need for an additional filter stage:

$$Y(z) = X(z) \cdot H_{truncated}(z) \cdot \frac{H_{decimated}(z)}{H_{truncated}(z)}$$

$$Y(z) = X(z) \cdot H_{decimated}(z)$$

Figure 4.5 shows the truncated frequency response of the Bergamo Cathedral and the FFT decimated frequency response (both of length 8192):

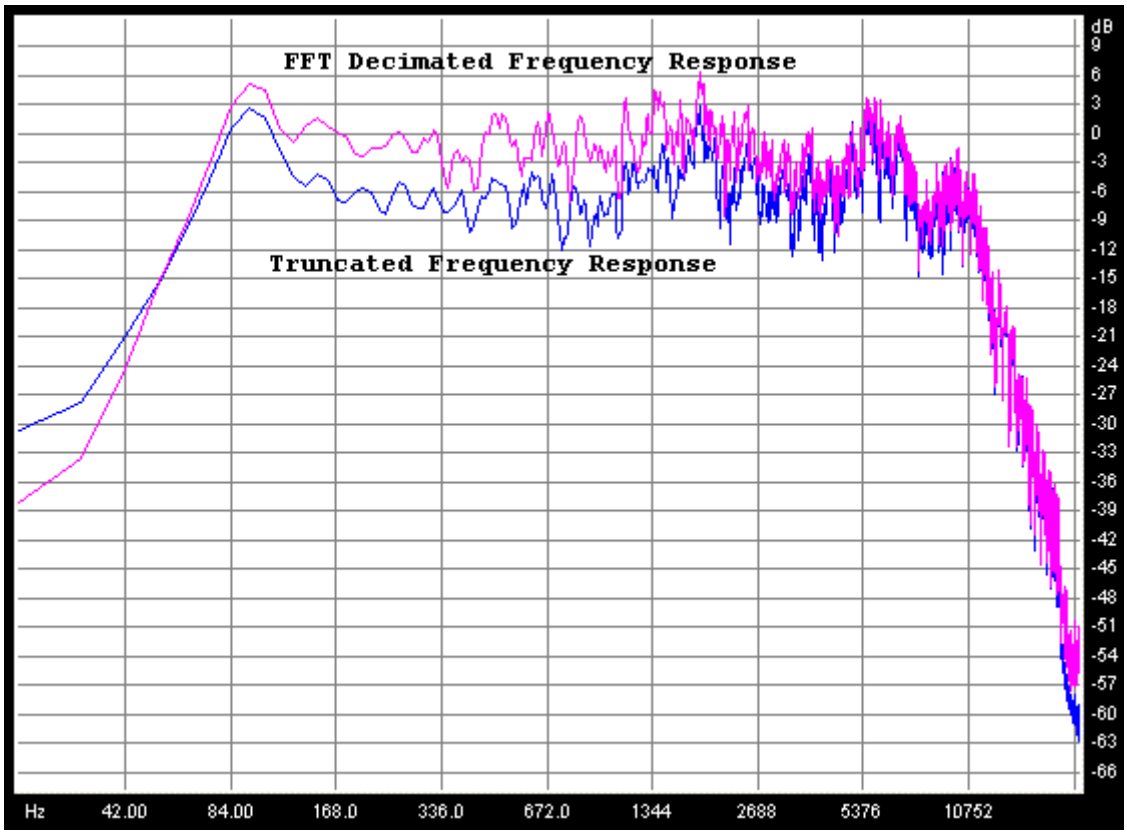

Figure 4.5: Truncated and FFT decimated frequency response of Bergamo Cathedral, Italy

Decimating the FFT, in effect, smoothes the frequency response of the room. The sharp transitions between frequency bins of the full FFT are not maintained in the decimated version. However, the same amplitude envelope will remain. It can be seen from Figure 4.5 that the decimated FFT frequency response has significant low frequency boost while maintaining the high frequency.

Decimating in this way does not preserve an impulse response that is 150 ms (6615 samples) long. The decimated frequency response is 8192 points and to preserve time

domain accuracy, the impulse response is also 8192 samples long. This prevents appropriate expansion in the overlap-add block convolution process discussed in section *2.1.2.3*. Therefore, the implementation used was a more simple 1^{st} order shelving filter that provides a low frequency boost while maintaining the high frequencies. The specific parameters of the filter must be selected manually for each impulse response, recognizing that some may not require one at all. Using the decimated approach would require a significant initial process (including a 2^{18} point FFT, and storage of full and decimated buffers) that the shelving filter approach does not.

5. Testing

The ideal artificial reverberation is created when an anechoic input signal is convolved with an acoustic impulse response. This ideal reverberation will be compared with the hybrid algorithm to evaluate its transparency. The hybrid algorithm was implemented as a Steinberg VST plugin equipped to perform on any computer running one of the Steinberg sequencing software packages (Cubase, Cubasis, etc.). As stated in Chapter 1, the goal of this thesis is to eliminate the inverse relationship between high fidelity artificial reverberation and computation cost. Therefore, in testing this new architecture we will be focusing on the similarities of its output with that of the ideal (linear convolution) output as well as examining the processor demands imposed by the implementation.

Testing the performance of this algorithm was done with a double-blind comparator test. The test determines if a person can discriminate between the output of the hybrid algorithm (convolution of truncated impulse response and recursive filtering) and the direct linear convolution of the full impulse response.

5.1 Source Material

All sound files are 16 bit, 44.1 kHz, mono .WAV files. Each sound was processed and exported using Steinberg Cubasis 3.7

Impulse Responses Used (44.1kHz, 16bit, mono)	
IR1.wav	**medium hall (***taken from Real Reverberation Winamp Plugin***)**
IR2.wav	**Bergamo Cathedral, Italy (***recorded by A. Farina using swept sine wave, w/ aurora software and dodecahedron loudspeaker***)**

Input Signals Used (44.1kHz, 16bit, mono)	
speech.wav	**male voice speaking "echo" (***recorded directly to PC using Cool Edit Pro***)**
orchestra.wav	**clip from Symphony #9 in C minor (***from Denon – Anechoic Orchestral Music Recording***)**
drums.wav	**drum set loop**
guitar.wav	**dry electric guitar (***recorded directly to PC using Cool Edit Pro***)**

Two impulse responses were tested with four music samples described in the table above. These music samples encompass an adequate cross-section of typical sound sources that would use artificial reverberation effects.

5.2 Time-Frequency Comparison

Comparing the Energy Decay Relief plots of the original and hybrid impulse responses will determine the degree of transparency that the hybrid algorithm will produce. Figure 5.1 shows the Energy Decay Relief plot of the full impulse response (*IR1.wav*) and the impulse response of the corresponding hybrid algorithm.

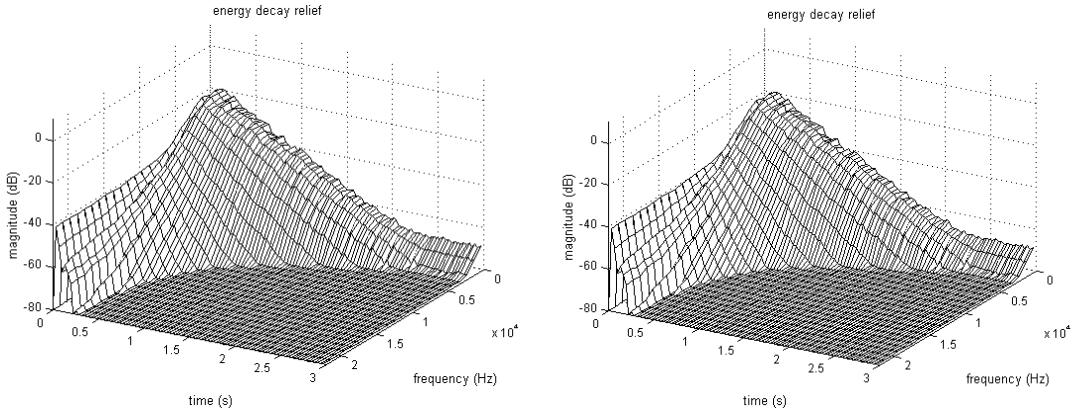

Figure 5.1: EDR of original and hybrid impulse response *(IR1.wav)*

Figure 5.2 shows the Energy Decay Relief plot of the full impulse response (*IR2.wav*) and the impulse response of the corresponding hybrid algorithm.

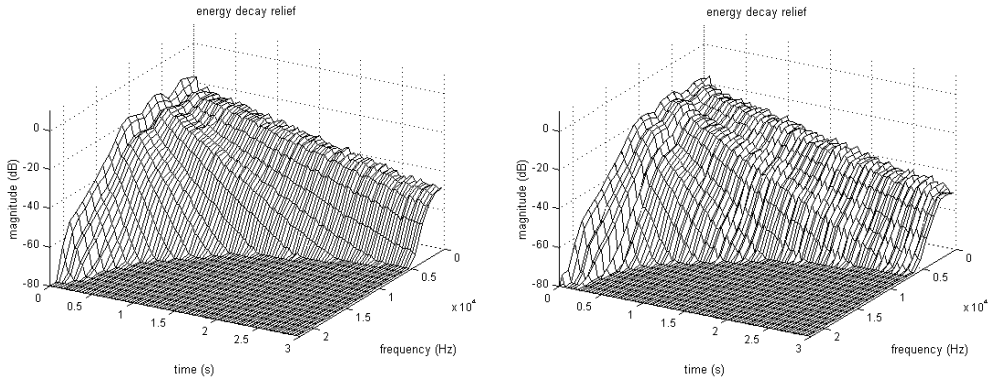

Figure 5.2: EDR of original and hybrid impulse response *(IR2.wav)*

The drum set loop input signal has a wide frequency range with heavy transient attacks from the percussion instruments. This sample proved to be the best overall test for proper comparison because it provided more obvious solutions to voicing needs.

Figure 5.3 shows the waveform of the drum set loop using the medium hall impulse response *(IR1.wav)*. The top waveform is the output due to direct linear convolution while the bottom is the output of the hybrid algorithm.

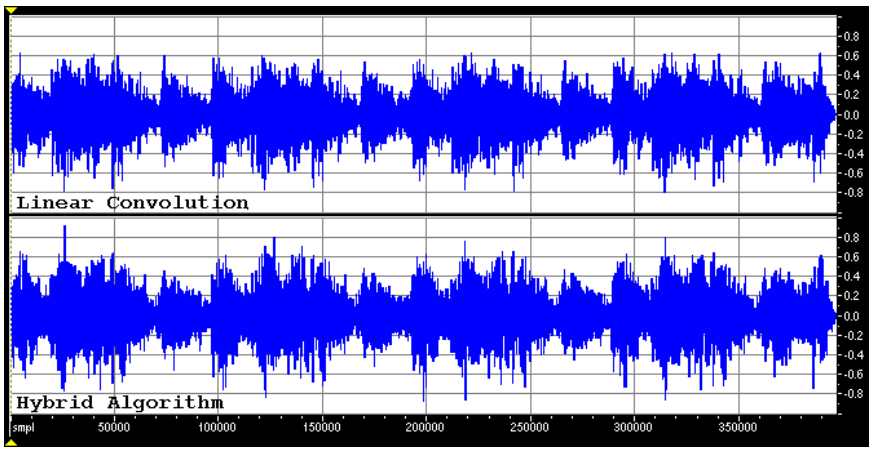

Figure 5.3: Waveform of drum set sample

Figure 5.4 shows the frequency response of each of the signals in Figure 5.3. This is the overall frequency content of the entire drum set loop providing only a general view.

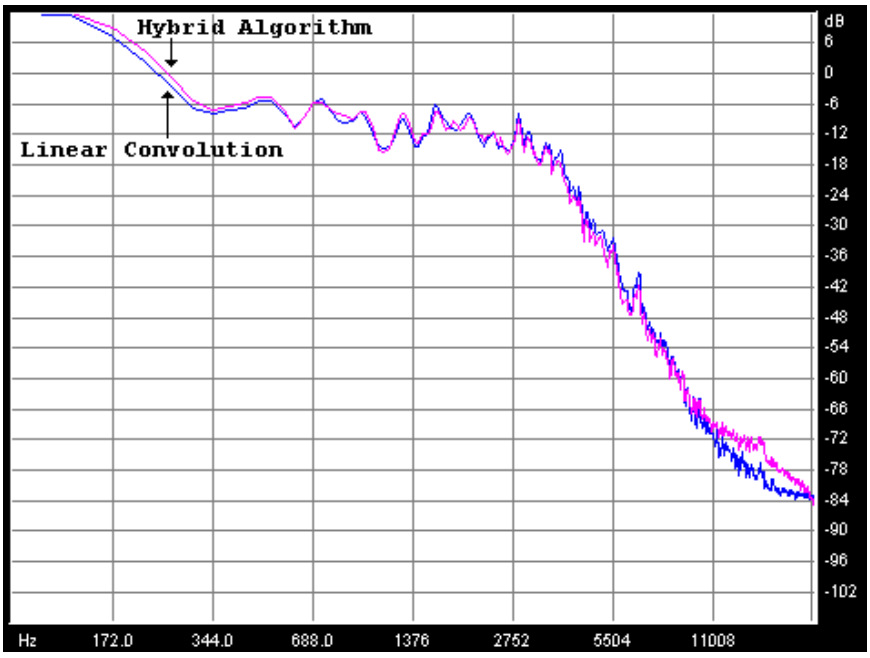

Figure 5.4: Frequency response of drum set sample

With the frequency content suitably identical, the reverberation decay becomes the final concern. This portion is controlled with the comb and all-pass filter feedback parameters and gain relative to the convolution stage.

Figures 5.5 and 5.6 show the time and frequency contents of the decay portion.

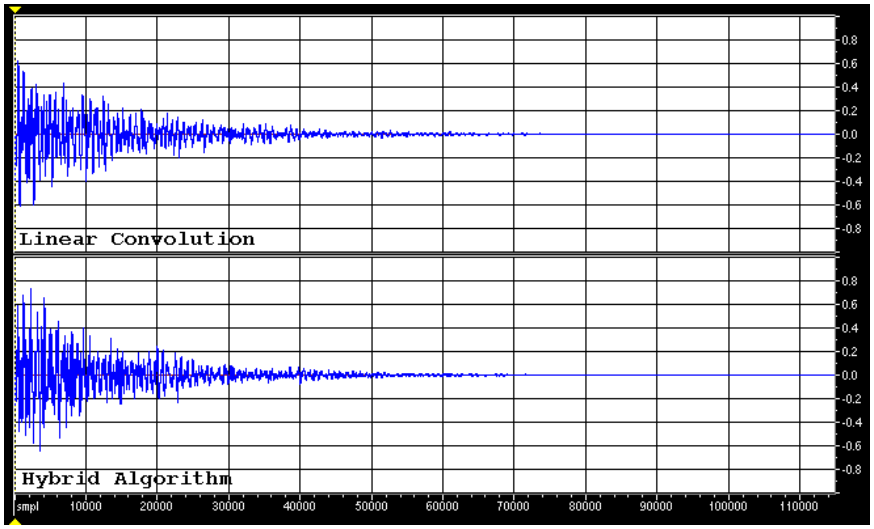

Figure 5.5: Waveform of drum set decay

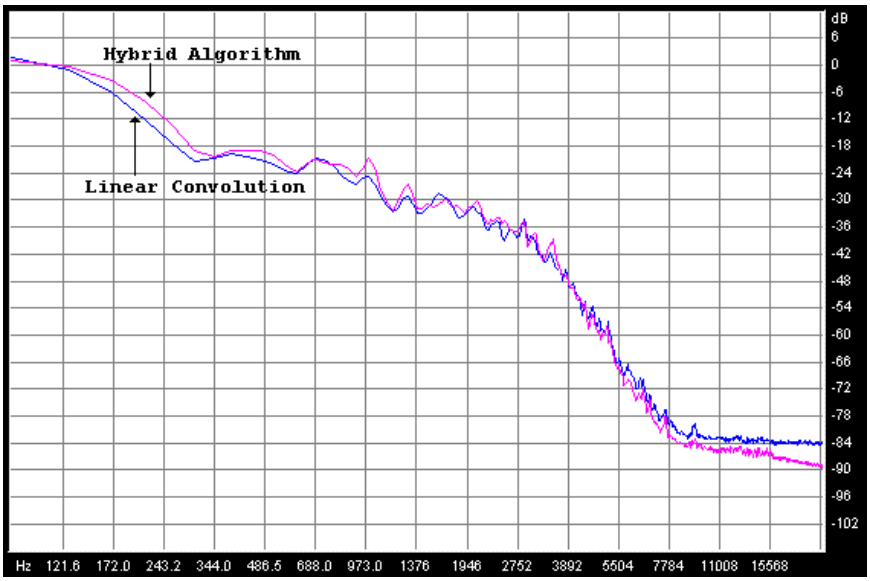

Figure 5.6: Frequency response of drum set decay

When using the medium room impulse response *(IR1.wav)* each of the 4 input samples showed similar transparency, but with some clear distinctions with particular instrumentation. For example, the pick attack of the guitar sample and high brass notes

of the orchestra sample were more distinct in the hybrid output. However, these anomalies were not apparent in the frequency response plots.

Figures 5.7 and 5.8 illustrate the time and frequency response of the reverberated guitar sample using the medium hall impulse response.

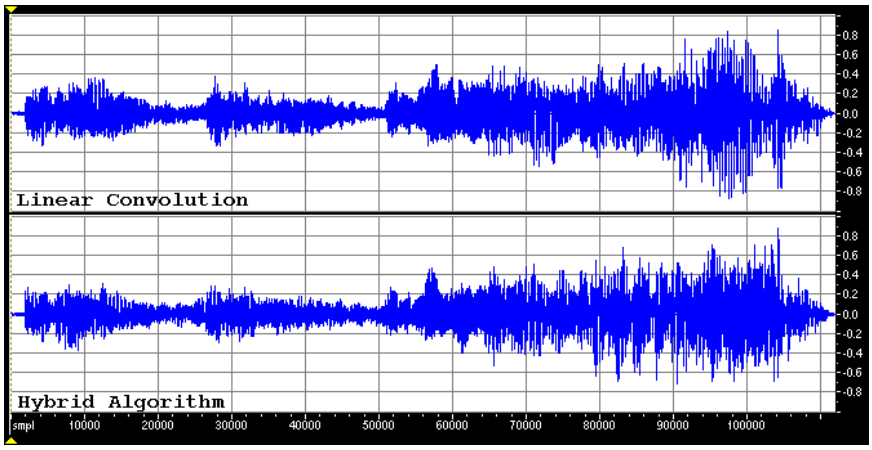

Figure 5.7: Waveform of guitar sample *(IR1.wav)*

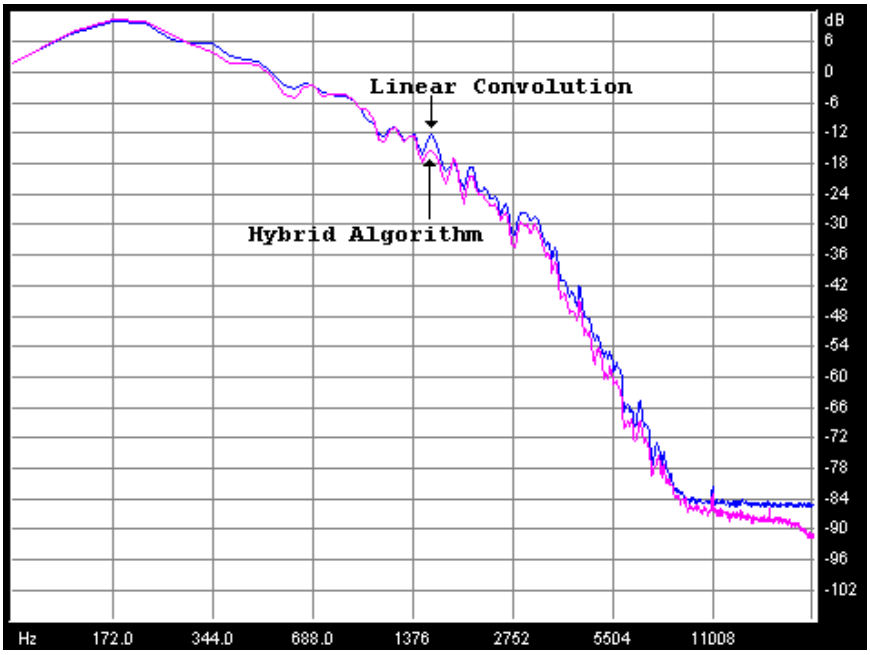

Figure 5.8: Frequency response of guitar sample *(IR1.wav)*

Figures 5.9 and 5.10 illustrate the time and frequency response of the reverberated orchestra sample using the medium hall impulse response.

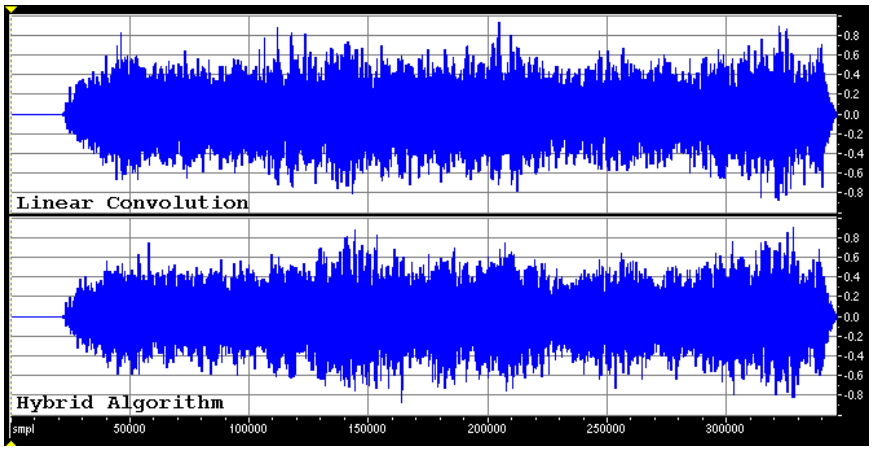

Figure 5.9: Waveform of orchestra sample *(IR1.wav)*

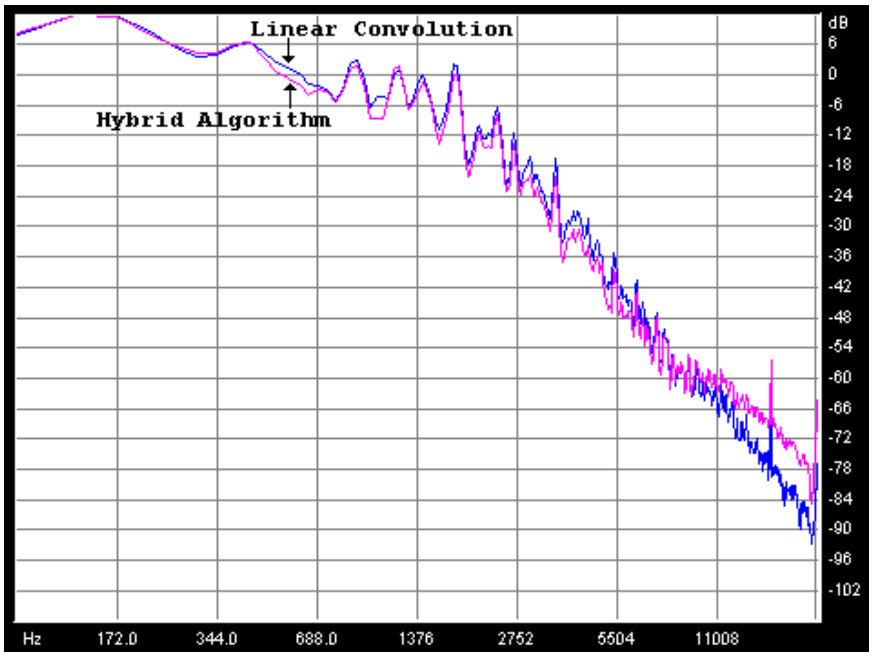

Figure 5.10: Frequency response of orchestra sample *(IR1.wav)*

Figures 5.11 and 5.12 illustrate the time and frequency response of the reverberated speech sample using the medium hall impulse response.

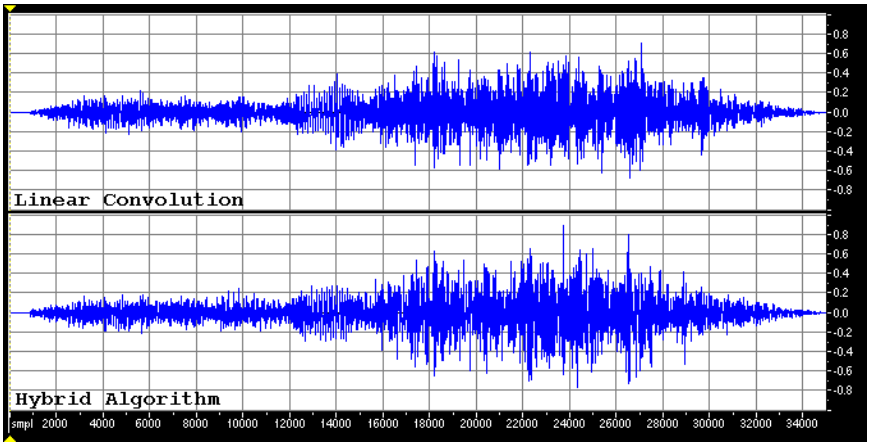

Figure 5.11: Waveform of speech sample *(IR1.wav)*

Figure 5.12: Frequency response of speech sample *(IR1.wav)*

Figures 5.13 and 5.14 illustrate the time and frequency response of the reverberated drum set sample using the Bergamo Cathedral impulse response.

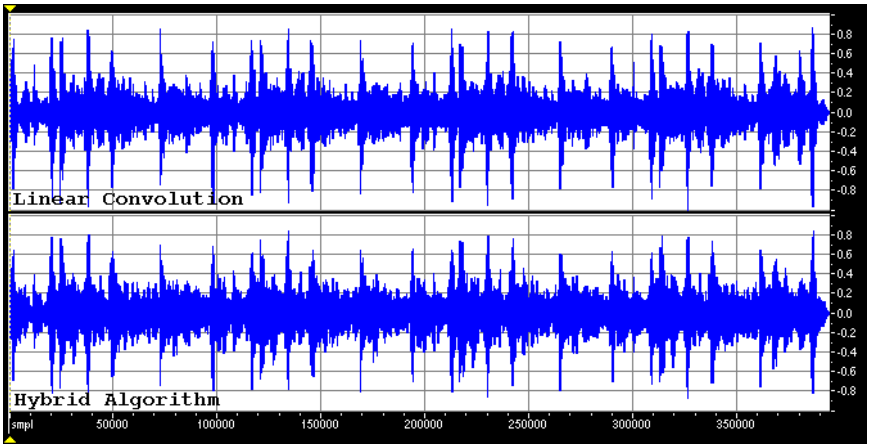

Figure 5.13: Waveform of drum set sample *(IR2.wav)*

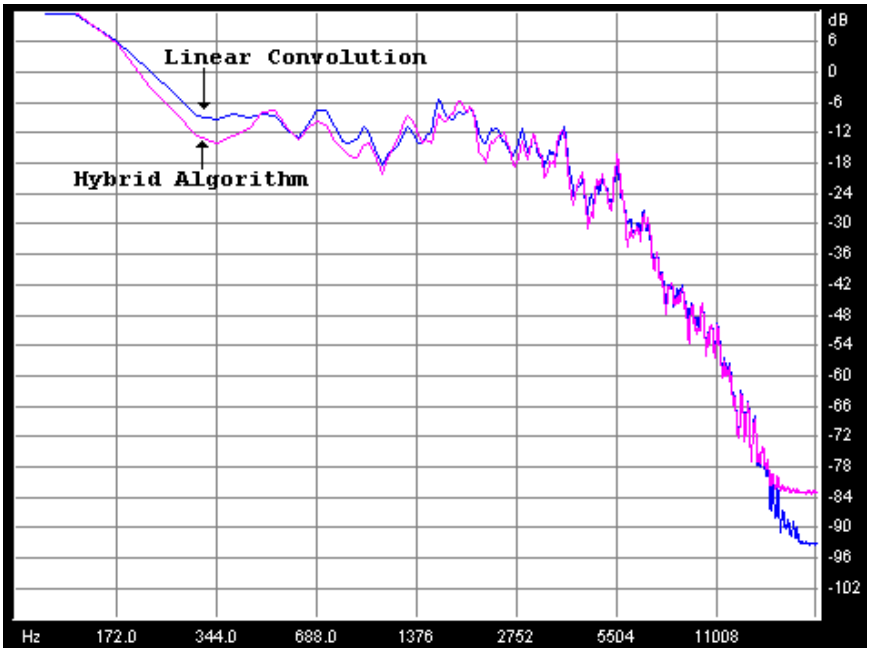

Figure 5.14: Frequency response of drum set sample *(IR2.wav)*

Figures 5.15 and 5.16 illustrate the time and frequency response of the reverberated guitar sample using the Bergamo Cathedral impulse response.

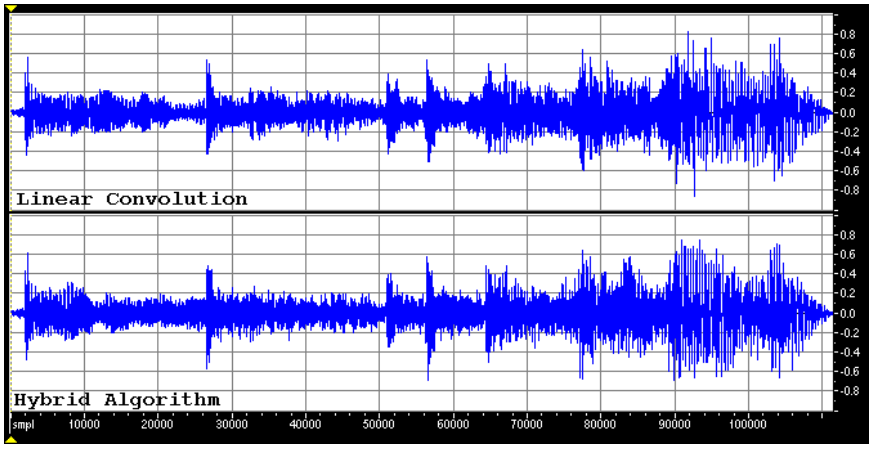

Figure 5.15: Waveform of guitar sample *(IR2.wav)*

Figure 5.16: Frequency response of guitar sample *(IR2.wav)*

Figures 5.17 and 5.18 illustrate the time and frequency response of the reverberated orchestra sample using the Bergamo Cathedral impulse response.

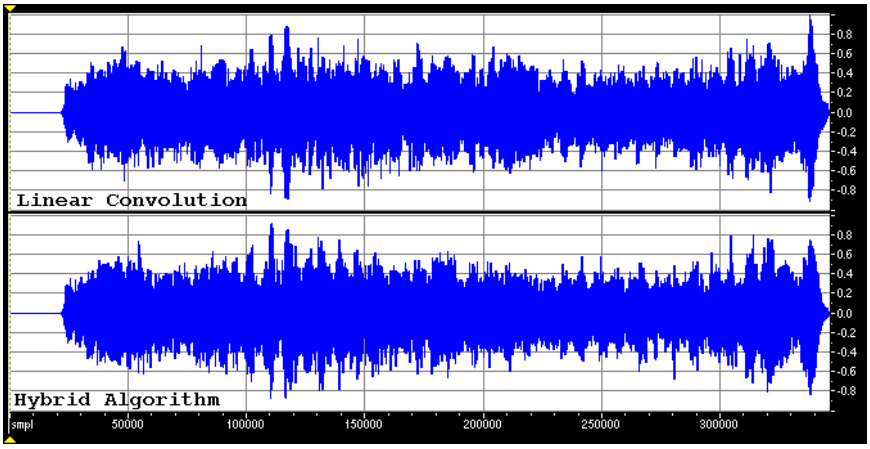

Figure 5.17: Waveform of orchestra sample *(IR2.wav)*

Figure 5.18: Frequency response of orchestra sample *(IR2.wav)*

Figures 5.19 and 5.20 illustrate the time and frequency response of the reverberated speech sample using the Bergamo Cathedral impulse response.

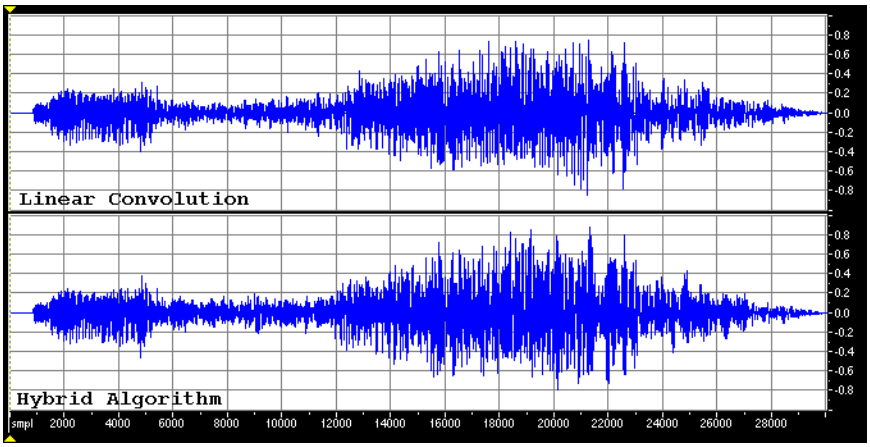

Figure 5.19: Waveform of speech sample *(IR2.wav)*

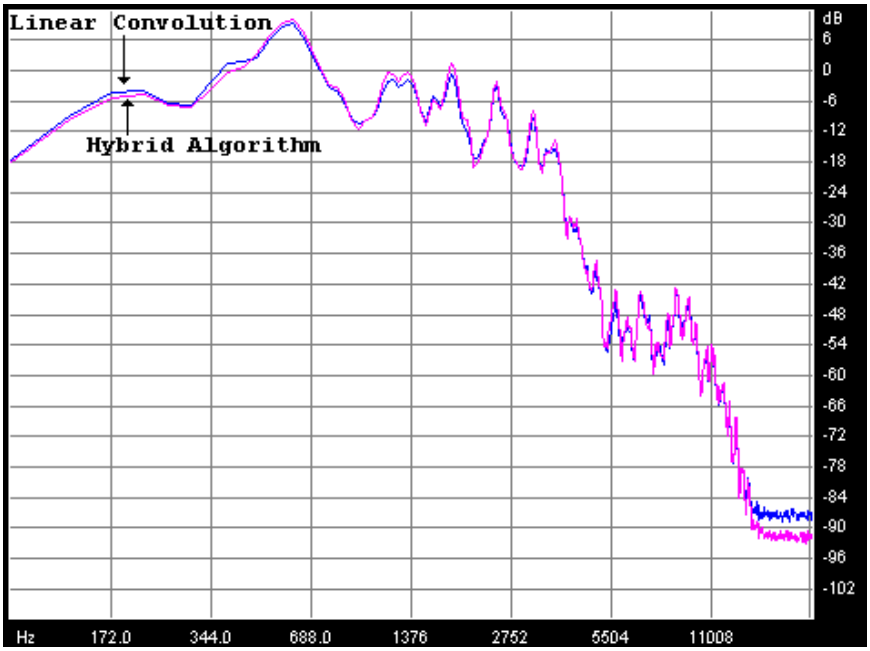

Figure 5.20: Frequency response of speech sample *(IR2.wav)*

5.3 ABX testing

The testing mechanism used was the PCABX testing software. The user interface is shown in Figure 5.21.

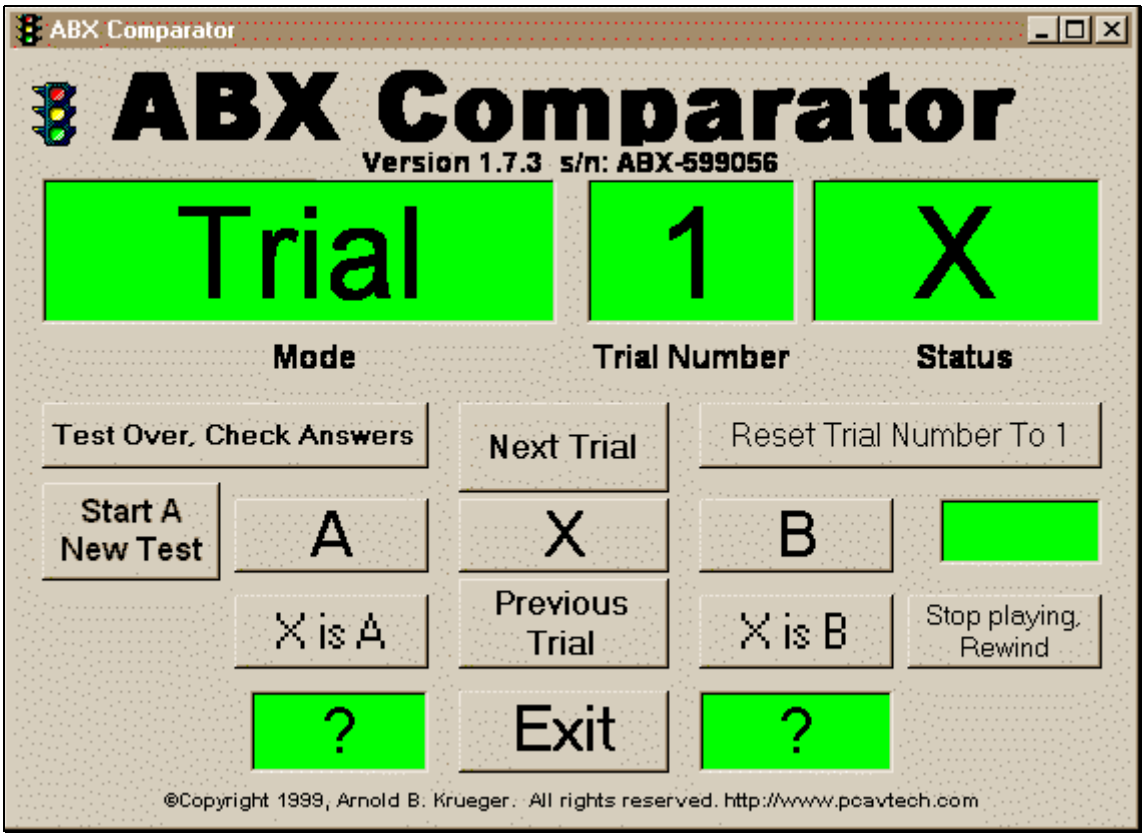

Figure 5.21: PCABX user interface

The linear convolution was performed for all four inputs for both impulse responses in MATLAB and output to a .WAV file (input "**A**"). The hybrid algorithm was implemented as a Steinberg VST real-time plugin and exported as .WAV (input "**B**"). PCABX randomly chooses which of the two shall be input "**X**" and the user must determine which of the two is "**X**".

Each of the 20 test subjects tested each of the 8 output samples 20 times each. If the subject was able to correctly identify "**X**" greater than 75% of the time, it was assumed that they could discriminate between the hybrid algorithm output and the linear convolution. If greater than 75% of the test subjects were able to discriminate, it was assumed that the hybrid algorithm output was not transparent to the linear convolution.[3]

averages	IR1.wav	IR2.wav
drums.wav	66.43%	85.71%
guitar.wav	71.07%	85.36%
orchestra.wav	73.93%	76.07%
speech.wav	78.21%	86.79%

No clear affirmation can be made about the quality of this hybrid algorithm based on the listening test results. The averages seem to show that the test subjects were able to discriminate between the linear convolution and the hybrid output for a majority of the input samples. However, investigating the standard deviations across the 20 test subjects reveals huge deviations from the average.

standard deviations	IR1.wav	IR2.wav
drums.wav	19.16%	23.60%
guitar.wav	18.21%	21.52%
orchestra.wav	20.77%	21.32%
speech.wav	13.10%	19.77%

For example, the average listening test score for the guitar sample under the Bergamo Cathedral impulse response, was 85.36% but the scores were on average 21.52% away from this value. An average score of 85.36% would suggest that the subjects were able to discriminate between the two signals overall. However, a 21.52% standard deviation indicates that some of the subjects could discriminate very well, while others could not. With these high standard deviation values, we can not conclude that the hybrid algorithm output can not be distinguished from the linear convolution

6. Conclusion

The research suggests that the hybrid algorithm was suitably transparent for some of the test subjects, but not for all, and not across the average of the 20 subjects tested. However, using an algorithm that uses a convolution scheme to create distinct early echoes and a recursive filter network to create a diffuse reverb tail is a viable artificial reverberation design strategy.

There is room for further research into many of the topics considered in this thesis. An adaptation of the decimation technique for the equalization component can be explored instead of the subjective shelving architecture. The use of non-radix-2 DFTs could provide the sample lengths needed for smooth and tonally balance block convolution. The addition of this type of quasi-adaptive voicing will greatly enhance the robustness of the algorithm, eliminating the need for subjective equalization design. The algorithm would be capable of evaluating its own voicing needs based on any discrepancies between the full and truncated impulse responses.

A limiting utility of reverberation network architecture is the lack of adequate time and frequency density measurement tools. Currently, only a mathematical approach for a single and parallel comb network have been utilized. Correlation of the ISO3382 time and clarity measurements could be adapted into accurate Time and Frequency Density measurement values. These measurements could be compared to the Schroeder and Jot specified values for accurate reverberation.

With processor performance as a major limiting factor, Moorer suggests varying the bit depth of the impulse response to ease the burden of computation. Instead of encoding every sample with 16 bits or greater, the rapid decay can be exploited by encoding the first 30% at over 12 bits, the next 30% between 8 and 11 bits, the next 30% between 4 and 8 bits and the final 10% with less than 4 bits. A typical monaural impulse response of 2 seconds, encoded at CD quality (16 bit, 44.1 kHz) will contain 1.4 Mb (176 kB), however, by employing this bit ratio, the impulse response could contain between 961 kb (120 kB) and 661 kb (83 kB).

This hybrid model is able to capture the general reverberation and frequency character of the acoustic spaces examined. When using an impulse response of an auditorium, room, hall, etc. the implementation of this algorithm produces a quick, quasi real-time reverberant output that can easily be used as an alternative to a convolution system.

References

[1] J. Borish, ”An Auditorium Simulator for Domestic Use,” J. Audio Eng. Soc., vol. 33, pp. 330 – 341, May 1985.

[2] J. Borwick, *Loudspeaker and Headphone Handbook*, Butterworth and Company, England, 1988.

[3] H. Burstein, “By the Numbers”, Audio, vol. 74, pp. 43 – 48, Feb. 1990.

[4] P. Cook, *Music, Cognition, and Computerized Sound*, MIT Press, Cambridge, MA, 1999.

[5] A. Czyewski, “A Method of Artificial Reverberation Quality Testing,” J. Audio Eng. Soc., vol. 38, pp. 129 – 141, March 1990.

[6] A. Farina, Personal Email Communication, 2001.

[7] A. Farina, “Pyramid Tracing vs. Ray Tracing for the simulation of sound propagation in large rooms”, *Computational Acoustics and its Environmental Applications*, pp. 109-116, 1995

[8] P. Fausti, A. Farina, “Acoustic Measurements in Opera Houses: Comparison Between Different Techniques and Equipment”, J. Sound and Vibration, v. 232, pp. 213 – 229, 2000

[9] W. G. Gardner, “3D Audio and Acoustic Environment Modeling,” Wave Arts, Inc., 1999.

[10] W. G. Gardner, “Efficient Convolution without Input-Output Delay,” J. Audio Eng. Soc., vol. 43, pp. 127 – 136, March 1995.

[11] W. G. Gardner, The virtual acoustic room. Master’s thesis, MIT Media Lab, 1992.

[12] D. Giuliani, M. Matassoni, M. Omologo, P. Svaizer, “Training of HMM With Filtered Speech Material For Hands-Free Recognition,” Proc. of ICASSP, vol. 1, pp. 449-452, March 1999.

[13] D. Griesinger, "Beyond MLS - Occupied Hall Measurement With FFT Techniques," in Proc. Audio Eng. Soc. 101st Conv., 1996, preprint 4403.

[14] D. Griesinger, "Impulse Response Measurements Using All-Pass Deconvolution,” 1992.

[15] D. Griesinger, “Practical Processors and Programs for Digital Reverberation,” in Proc. Audio Eng. Soc. 7th Int. Conf., pp. 187-195, 1989.

[16] D. Griesinger, “The Theory and Practice of Perceptual Modeling - How to use Electronic Reverberation to Add Depth and Envelopment Without Reducing Clarity,” preprint from Tonmeister conference in Hannover, 2000

[17] C. Gumas, “A century old, the fast Hadamard transform proves useful in digital communications,” Personal Instrumentation and Engineering News, pp. 57 – 63, Nov. 1997

[18] J.-M. Jot and A. Chaigne, “Digital Delay Networks for Designing Artificial Reverberators,” in Proc. 90th Conv. Audio Eng. Soc., Feb. 1991, preprint 3030.

[19] J.-M. Jot, “Efficient Models for Reverberation and Distance Rendering in Computer Music and Virtual Reality,” Int. Computer Music Conf. (Thessaloniki, Greece), Sept. 1997.

[20] J. A. Moorer, “About This Reverberation Business,” Computer Music Journal, vol. 3, no. 2, pp. 13 – 28, June 1979.

[21] F. R. Moore, *Elements of Computer Music*, New Jersey, 1990

[22] S. E. Olive and F. E. Toole, “The Detection of Reflections in Typical Rooms,” J. Audio Eng. Soc., vol. 37, pp. 539 – 552, July 1989.

[23] S. Orfanidis, *Introduction to Signal Processing*, New Jersey, 1996.

[24] J. W. Panzer and R. H. Campbell, “Multiple Driver Modeling with a Modern Lumped Element Simulation Program,” 102nd Conv. Audio Eng. Soc., March 1997, preprint 4441.

[25] S. W. Smith, *The Scientist’s and Engineer’s Guide to Digital Signal Processing*, California Technical Publishing, 1997.

[26] U. Zölzer, *Digital Audio Signal Processing*, John Wiley & Sons, New York, NY, 1997

[27] J. VandeKieft, “Computational Improvements to Linear Convolution With Multi-rate Filtering Methods,” Master’s Thesis, University of Miami, 1998.

[28] R. A. Wannamaker, “Psychoacoustically Optimal Noise Shaping,” J. Audio Eng. Soc., vol. 40, no. 7/8, pp.611 – 620, 1992.

Appendix

VST C++ Source Code

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <windows.h>
#include <io.h>
#include "sb_FFTConv.hpp"
#include "AEffEditor.hpp"

#define PI 3.14159265358979323846264338327795
#define TWOPI 6.28318530717958647692528676655 8
#define SWAP(a,b) tempr=(a); (a)=(b); (b)=tempr
#define NPOINTS 8192          // Size of FFT, next power of 2 geater than TRUNCATE POINTS
#define TRUNCATEPOINTS 6615   // The amount of samples to keep (0 -> TRUNCATEPOINTS -1)
#define INPUTPOINTS 1578      // NPOINTS - TRUNCATEPOINTS + 1

//-----------------------------------------------------------------------------

sb_FFTConvProgram::sb_FFTConvProgram ()
{
        // for IR1:
        fGain1 = 0.36307805477f;      // -26.4 was -26.6dB
        fGain2 = 0.50504878811f;      // -17.8 was -22.4dB
        fGain3 = 0.25f;
        fFeedBack = 0.82f;

        // for IR2:
        //fGain1 = 0.926118728129f;   // -2dB
        //fGain2 = 0.926118728129f;   // -2dB
        //fGain3 = 0.0f;
        //fFeedBack = 0.93f;

        inPosx = 0; outPosy = 0;

        // Moorer reverberation tail structure
        fDelay1 = 0.050f;
        fDelay2 = 0.056f;
        fDelay3 = 0.061f;
        fDelay4 = 0.068f;
        fDelay5 = 0.072f;
        fDelay6 = 0.078f;

        fDelayAP = 0.006f;  // 6msec

        //1st order LPF at 10kHz
        LPFa0 = 0.4634255f;
        LPFb1 = -0.0731489f;
        //1st order LPF at fc = 4kHz and fs = 44.1kHz
        //LPFa0 = 0.2265588f;
        //LPFb1 = -0.5468823f;
        //1st order LPF at fc = 8kHz and fs = 44.1kHz
        //LPFa0 = 0.3905532f;
        //LPFb1 = -0.2188936f;
        //1st order LPF at fc = 9kHz and fs = 44.1kHz
        //LPFa0 = 0.4273671f;
        //LPFb1 = -0.1452659f;

        LPF1output = 0.0f;
        LPF1x1 = 0.0f;
        LPF1y1 = 0.0f;

        LPF2output = 0.0f;
        LPF2x1 = 0.0f;
        LPF2y1 = 0.0f;

        LPF3output = 0.0f;
```

```
	LPF3x1 = 0.0f;
	LPF3y1 = 0.0f;

	LPF4output = 0.0f;
	LPF4x1 = 0.0f;
	LPF4y1 = 0.0f;

	LPF5output = 0.0f;
	LPF5x1 = 0.0f;
	LPF5y1 = 0.0f;

	LPF6output = 0.0f;
	LPF6x1 = 0.0f;
	LPF6y1 = 0.0f;

	// 3800Hz, boost 6dB
	SHFa0 = 0.5817138f;
	SHFa1 = -0.2540805f;
	SHFb1 = -0.3357944f;
	SHFx1 = SHFy1 = 0.0f;

	strcpy (name, "Init");
}

//------------------------------------------------------------------------

sb_FFTConv::sb_FFTConv (audioMasterCallback audioMaster)
	: AudioEffectX (audioMaster, 16, kNumParams)
{
	buffer_h = new float[NPOINTS];
	buffer_x = new float[NPOINTS];
	buffer_x1 = new float[INPUTPOINTS];
	buffer_y = new float[NPOINTS];
	buffer_conv = new float[NPOINTS];
	size = 4410;   // 100 msec delay lines
	buffer1 = new float[size]; // comb 1
	buffer2 = new float[size]; // comb 2
	buffer3 = new float[size]; // comb 3
	buffer4 = new float[size]; // comb 4
	buffer5 = new float[size]; // comb 5
	buffer6 = new float[size]; // comb 6
	buffer7 = new float[size]; // AP

	programs = new sb_FFTConvProgram[numPrograms];

	// for IR1:
	fGain1 = 0.36307805477f;      // -26.4 was -26.6dB
	fGain2 = 0.50504878811f;      // -17.8 was -22.4dB
	fGain3 = 0.25f;
	fFeedBack = 0.82f;

	// for IR2:
	//fGain1 = 0.926118728129f;   // -2dB
	//fGain2 = 0.926118728129f;   // -2dB
	//fGain3 = 0.0f;
	//fFeedBack = 0.93f;

	inPosx = 0; outPosy = 0;

	// Moorer reverberation tail structure
	fDelay1 = 0.050f;
	fDelay2 = 0.056f;
	fDelay3 = 0.061f;
	fDelay4 = 0.068f;
	fDelay5 = 0.072f;
	fDelay6 = 0.078f;

	fDelayAP = 0.006f;  // 6msec

	inPos1 = 0;
	outPos1 = outPos2 = outPos3 = outPos4 = outPos5 = outPos6 = 0;
```

```
	outPosAP = outPosx1 = outPosy = 0;

	//1st order LPF at 10kHz
	LPFa0 = 0.4634255f;
	LPFb1 = -0.0731489f;
	//1st order LPF at fc = 4kHz and fs = 44.1kHz
	//LPFa0 = 0.2265588f;
	//LPFb1 = -0.5468823f;
	//1st order LPF at fc = 8kHz and fs = 44.1kHz
	//LPFa0 = 0.3905532f;
	//LPFb1 = -0.2188936f;
	//1st order LPF at fc = 9kHz and fs = 44.1kHz
	//LPFa0 = 0.4273671f;
	//LPFb1 = -0.1452659f;

	LPF1output = LPF1x1 = LPF1y1 = 0.0f;
	LPF2output = LPF2x1 = LPF2y1 = 0.0f;
	LPF3output = LPF3x1 = LPF3y1 = 0.0f;
	LPF4output = LPF4x1 = LPF4y1 = 0.0f;
	LPF5output = LPF5x1 = LPF5y1 = 0.0f;
	LPF6output = LPF6x1 = LPF6y1 = 0.0f;

	// 3800Hz, boost 6dB
	SHFa0 = 0.5817138f;
	SHFa1 = -0.2540805f;
	SHFb1 = -0.3357944f;
	SHFx1 = SHFy1 = 0.0f;

	// 3800Hz, boost 4dB
	//SHFa0 = 0.5327609f;
	//SHFa1 = -0.3552158f;
	//SHFb1 = -0.3879766f;
	//SHFx1 = SHFy1 = 0.0f;

	if (programs)
		setProgram (0);

	setNumInputs (2);
	setNumOutputs (2);
	canMono();
	canProcessReplacing ();
	setUniqueID ('sbth');
	suspend ();		// flush buffer
	loadIR();
	setDelay(&fDelay1, &outPos1);
	setDelay(&fDelay2, &outPos2);
	setDelay(&fDelay3, &outPos3);
	setDelay(&fDelay4, &outPos4);
	setDelay(&fDelay5, &outPos5);
	setDelay(&fDelay6, &outPos6);
	setDelay(&fDelayAP, &outPosAP);
}

//-----------------------------------------------------------------------
sb_FFTConv::~sb_FFTConv ()
{
	if (!buffer_h)
	{
		delete[] buffer_h;
		delete[] buffer_x;
		delete[] buffer_x1;
		delete[] buffer_y;
		delete[] buffer_conv;
		delete[] buffer1;
		delete[] buffer2;
		delete[] buffer3;
		delete[] buffer4;
		delete[] buffer5;
		delete[] buffer6;
		delete[] buffer7;
	}
```

```
	if (programs)
		delete[] programs;
}

//------------------------------------------------------------------------
void sb_FFTConv::setGain1 (float gain1)
{
	// set the UNCOOKED var
	fVSTGain1 = gain1;
	// set the current program's delay variable
	programs[curProgram].fVSTGain1 = gain1;
	// logish
	fGain1 = fVSTGain1*fVSTGain1*fVSTGain1;
}

void sb_FFTConv::setFeedBack (float feedback)
{
	fVSTFeedBack = feedback;
	programs[curProgram].fVSTFeedBack = feedback;
	fFeedBack = (float)(1*fVSTFeedBack);
}

void sb_FFTConv::setGain2 (float gain2)
{
	// set the UNCOOKED var
	fVSTGain2 = gain2;
	// set the current program's delay variable
	programs[curProgram].fVSTGain2 = gain2;
	// logish
	fGain2 = fVSTGain2*fVSTGain2*fVSTGain2;
}

void sb_FFTConv::setGain3 (float gain3)
{
	// set the UNCOOKED var
	fVSTGain3 = gain3;
	// set the current program's delay variable
	programs[curProgram].fVSTGain3 = gain3;
	// lin
	fGain3 = fVSTGain3;
}

void sb_FFTConv::setDelay (float* pfdelay, long* poutPos)
{
	*poutPos = (size-1) - (long)(*pfdelay*44100);
}
//------------------------------------------------------------------------
void sb_FFTConv::setProgram (long program)
{
	sb_FFTConvProgram * ap = &programs[program];
	curProgram = program;
	setParameter (kGain1, ap->fGain1);
	setParameter (kFeedBack, ap->fFeedBack);
	setParameter (kGain2, ap->fGain2);
	setParameter (kGain3, ap->fGain3);
}

//------------------------------------------------------------------------
void sb_FFTConv::setProgramName (char *name)
{
	strcpy (programs[curProgram].name, name);
}

//------------------------------------------------------------------------
void sb_FFTConv::getProgramName (char *name)
{
	if (!strcmp (programs[curProgram].name, "Init"))
		sprintf (name, "%s %d", programs[curProgram].name, curProgram + 1);
	else
		strcpy (name, programs[curProgram].name);
}
```

```
//-----------------------------------------------------------------------
void sb_FFTConv::suspend ()
{
	memset (buffer_h, 0, NPOINTS * sizeof (float));
	memset (buffer_x, 0, NPOINTS * sizeof (float));
	memset (buffer_x1, 0, INPUTPOINTS * sizeof (float));
	memset (buffer_y, 0, NPOINTS * sizeof (float));
	memset (buffer_conv, 0, NPOINTS * sizeof (float));
	memset (buffer1, 0, size * sizeof (float));
	memset (buffer2, 0, size * sizeof (float));
	memset (buffer3, 0, size * sizeof (float));
	memset (buffer4, 0, size * sizeof (float));
	memset (buffer5, 0, size * sizeof (float));
	memset (buffer6, 0, size * sizeof (float));
	memset (buffer7, 0, size * sizeof (float));
}

//-----------------------------------------------------------------------
void sb_FFTConv::setParameter (long index, float value)
{
	sb_FFTConvProgram * ap = &programs[curProgram];

	switch (index)
	{
		case kGain1		:	setGain1 (value); break;
		case kFeedBack		:	setFeedBack (value); break;
		case kGain2		:	setGain2 (value); break;
		case kGain3		:	setGain3 (value); break;
	}
	if (editor)
		editor->postUpdate ();
}

//-----------------------------------------------------------------------
float sb_FFTConv::getParameter (long index)
{
	float v = 0;

	switch (index)
	{
		case kGain1		:	v = fVSTGain1; break;
		case kFeedBack		:	v = fVSTFeedBack; break;
		case kGain2		:	v = fVSTGain2; break;
		case kGain3		:	v = fVSTGain3; break;
	}
	return v;
}

//-----------------------------------------------------------------------
void sb_FFTConv::getParameterName (long index, char *label)
{
	switch (index)
	{
		case kGain1   :		strcpy (label, "Gain1"); break;
		case kFeedBack:	strcpy (label, "RoomSize"); break;
		case kGain2   :		strcpy (label, "Gain2"); break;
		case kGain3   :		strcpy (label, "Gain3"); break;
	}
}

//-----------------------------------------------------------------------
void sb_FFTConv::getParameterDisplay (long index, char *text)
{
	switch (index)
	{
		case kGain1		: dB2string (fGain1, text);break;
		case kFeedBack		: float2string (fFeedBack, text);    break;
		case kGain2		: dB2string (fGain2, text);break;
		case kGain3		: float2string (fGain3, text);break;
	}
```

```
}

//-----------------------------------------------------------------------
void sb_FFTConv::getParameterLabel (long index, char *label)
{
	switch (index)
	{
		case kGain1		:	strcpy (label, "dB");  break;
		case kFeedBack		:	strcpy (label, "amount");     break;
		case kGain2		:	strcpy (label, "dB");  break;
		case kGain3		:	strcpy (label, " ");   break;
	}
}

//-----------------------------------------------------------------------
//-----------------------------------------------------------------------

void sb_FFTConv::process (float **inputs, float **outputs, long sampleframes)
{
	float* in1 = inputs[0];
	float* out1 = outputs[0];
	float* out2 = outputs[1];
	float  d1, d2, d3, d4, d5, d6, dAP;
	d1 = d2 = d3 = d4 = d5 = d6 = dAP = 0.0f;
	float  xn1, xnAP;
	float SHFout, SHFin;

	while (--sampleframes >= 0)  // for every incoming sample
	{
		if (inPos1 > (size-1))
			inPos1 = 0;
		if (outPos1 > (size-1))
			outPos1 = 0;
		if (outPos2 > (size-1))
			outPos2 = 0;
		if (outPos3 > (size-1))
			outPos3 = 0;
		if (outPos4 > (size-1))
			outPos4 = 0;
		if (outPos5 > (size-1))
			outPos5 = 0;
		if (outPos6 > (size-1))
			outPos6 = 0;
		if (outPosAP > (size-1))
			outPosAP = 0;

		if (outPosy > (NPOINTS-1))
			outPosy = 0;

		xn1 = buffer_y[outPosy];

		d1 = (buffer1[outPos1] * fFeedBack);
		doLPF1(&d1);
		d2 = (buffer2[outPos2] * fFeedBack);
		doLPF2(&d2);
		d3 = (buffer3[outPos3] * fFeedBack);
		doLPF3(&d3);
		d4 = (buffer4[outPos4] * fFeedBack);
		doLPF4(&d4);
		d5 = (buffer5[outPos5] * fFeedBack);
		doLPF5(&d5);
		d6 = (buffer6[outPos6] * fFeedBack);
		doLPF6(&d6);

		buffer1[inPos1] = d1+xn1;
		buffer2[inPos1] = d2+xn1;
		buffer3[inPos1] = d3+xn1;
		buffer4[inPos1] = d4+xn1;
		buffer5[inPos1] = d5+xn1;
		buffer6[inPos1] = d6+xn1;
```

```
                xnAP = (d1+d2+d3+d4+d5+d6)/6.0f;
                dAP = (-0.85f * xnAP) + buffer7[outPosAP];
                buffer7[inPos1] = xnAP + (0.85f * dAP);

                if (inPosx > (INPUTPOINTS-1))
                {
                        doFFTConv();
                        inPosx = outPosx1 = 0;
                }

                buffer_x[inPosx] = *in1;

                SHFin = (fGain3*buffer_x1[outPosx1]) + (fGain1*buffer_y[outPosy]) +
                    (fGain2*dAP);
                SHFout = SHFa0*SHFin + SHFa1*SHFx1 - SHFb1*SHFy1;
                SHFout *= 2.0f;
                SHFx1 = SHFin;
                SHFy1 = SHFout;

                *out1 = SHFout*0.5f; //multiply by .5 to cancel the 6dB boost of the shelf
                *out2 = *out1;

                in1++; out1++; out2++;
                inPosx++; outPosy++;
                inPos1++;
                outPos1++; outPos2++; outPos3++; outPos4++; outPos5++; outPos6++;
                outPosAP++;
                outPosx1++;
        }
}

//-----------------------------------------------------------------------------
void sb_FFTConv::processReplacing (float **inputs, float **outputs, long sampleframes)
{
        sb_FFTConv::process (inputs, outputs, sampleframes);
}

//-----------------------------------------------------------------------------
void sb_FFTConv::doFFTConv()
{
        float TWOoverNPOINTS = 2./NPOINTS;

        // pad input with zeros up to NPOINTS
        for (int n=0; n<INPUTPOINTS; n++)
                buffer_x1[n] = buffer_x[n];

        doFour2(buffer_x-1, NPOINTS, 1);


        // do freq. multiply, store in convolution buffer
        buffer_conv[0] = buffer_x[0]*buffer_h[0];
        buffer_conv[1] = buffer_x[1]*buffer_h[1];
        for (n=2; n<NPOINTS; n += 2)
                buffer_conv[n] = buffer_x[n]*buffer_h[n] - buffer_x[n+1]*buffer_h[n+1];
        for (n=3; n<NPOINTS; n += 2)
                buffer_conv[n] = buffer_x[n-1]*buffer_h[n] + buffer_x[n]*buffer_h[n-1];

        // IFFT to get the time domain portion
        doFour2(buffer_conv-1, NPOINTS, -1);
        for (n=0; n<NPOINTS; n++)
                buffer_conv[n] = buffer_conv[n]*TWOoverNPOINTS;

        // zero out the first 1578 samples
        if (outPosy >= (INPUTPOINTS-1))
        {
                for (n=(outPosy-INPUTPOINTS); n<outPosy; n++)
                        buffer_y[n] = 0.;
        }
        else
        {
                for (n=0; n<outPosy; n++)
```

```
                buffer_y[n] = 0.;
        for (n=(NPOINTS-INPUTPOINTS+outPosy-1); n<NPOINTS; n++)
                buffer_y[n] = 0.;
    }

    // add the overlapped part
    for (n=0; (n<(NPOINTS-outPosy)); n++)
        buffer_y[outPosy+n] += buffer_conv[n];

    for (n=0; n<outPosy; n++)
        buffer_y[n] += buffer_conv[NPOINTS-outPosy+n];

    // reset the input buffer
    memset (buffer_x, 0, NPOINTS * sizeof (float));
}

//------------------------------------------------------------------------
void sb_FFTConv::doFour1(float* data, int nn, int isign)
{
    long n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    float tempr, tempi;

    n=nn << 1;
    j=1;
    for(i=1; i<n; i+=2)
    {
        if(j>i)
        {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while(m>=2 && j>m)
        {
            j -= m;
            m >>= 1;
        }
        j +=m;
    }
    mmax = 2;
    while (n > mmax)
    {
        istep = mmax << 1;
        theta = isign*(TWOPI/mmax);
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m<mmax; m+=2)
        {
            for(i=m; i<=n; i+=istep)
            {
                j = i+mmax;
                tempr = (float)(wr*data[j] - wi*data[j+1]);
                tempi = (float)(wr*data[j+1] + wi*data[j]);
                data[j] = data[i] - tempr;
                data[j+1] = data[i+1]-tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr = (wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax = istep;
    }
}

//------------------------------------------------------------------------
void sb_FFTConv::doFour2(float* data, int n, int isign)
```

```
{
	long i, i1, i2, i3, i4, np3;
	float c1=0.5f, c2, h1r, h1i, h2r, h2i;
	double wr, wi, wpr, wpi, wtemp, theta;
	theta = PI/(double)(n>>1);
	if (isign == 1)
	{
		c2 = -0.5;
		doFour1(data, n>>1, 1);
	}
	else
	{
		c2 = 0.5;
		theta = -theta;
	}
	wtemp = sin(0.5*theta);
	wpr = -2.0*wtemp*wtemp;
	wpi = sin(theta);
	wr = 1.0+wpr;
	wi = wpi;
	np3 = n+3;
	for (i=2; i <= (n>>2); i++)
	{
		i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
		h1r=c1*(data[i1]+data[i3]);
		h1i=c1*(data[i2]-data[i4]);
		h2r = -c2*(data[i2]+data[i4]);
		h2i=c2*(data[i1]-data[i3]);
		data[i1]= (float)(h1r+wr*h2r-wi*h2i);
		data[i2]= (float)(h1i+wr*h2i+wi*h2r);
		data[i3]= (float)(h1r-wr*h2r+wi*h2i);
		data[i4]=(float)(-h1i+wr*h2i+wi*h2r);
		wr=(wtemp=wr)*wpr-wi*wpi+wr;
		wi=wi*wpr+wtemp*wpi+wi;
	}
	if (isign == 1)
	{
		data[1] = (h1r=data[1])+data[2];
		data[2] = h1r-data[2];
	}
	else
	{
		data[1]=c1*((h1r=data[1])+data[2]);
		data[2]=c1*(h1r-data[2]);
		doFour1(data, n>>1, -1);
	}
}

//---------------------------------------------------------------------------
void sb_FFTConv::doLPF1(float* input)
{
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF1output = LPFa0*(*input + LPF1x1) - (LPFb1*LPF1y1);

	LPF1x1 = *input;
	LPF1y1 = LPF1output;
	*input = LPF1output;
}

void sb_FFTConv::doLPF2(float* input)
{
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF2output = LPFa0*(*input + LPF2x1) - (LPFb1*LPF2y1);

	LPF2x1 = *input;
	LPF2y1 = LPF2output;
	*input = LPF2output;
}

void sb_FFTConv::doLPF3(float* input)
{
```

```
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF3output = LPFa0*(*input + LPF3x1) - (LPFb1*LPF3y1);

	LPF3x1 = *input;
	LPF3y1 = LPF3output;
	*input = LPF3output;
}

void sb_FFTConv::doLPF4(float* input)
{
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF4output = LPFa0*(*input + LPF4x1) - (LPFb1*LPF4y1);

	LPF4x1 = *input;
	LPF4y1 = LPF4output;
	*input = LPF4output;
}

void sb_FFTConv::doLPF5(float* input)
{
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF5output = LPFa0*(*input + LPF5x1) - (LPFb1*LPF5y1);

	LPF5x1 = *input;
	LPF5y1 = LPF5output;
	*input = LPF5output;
}

void sb_FFTConv::doLPF6(float* input)
{
	// y(n) = a0*x(n) + a0*x(n-1) - b1*y(n-1)
	LPF6output = LPFa0*(*input + LPF6x1) - (LPFb1*LPF6y1);

	LPF6x1 = *input;
	LPF6y1 = LPF6output;
	*input = LPF6output;
}

//------------------------------------------------------------------------------
void sb_FFTConv::loadIR()
{
	short temp = 0;
	int i;
	FILE* impulse;
	impulse = fopen("C:/WINDOWS/Desktop/thesis test/IR1.pcm", "rb");

	for (i = 0; i<TRUNCATEPOINTS; i++)
	{
		fscanf(impulse, "%2c" ,&temp);
		buffer_h[i] = (float)(temp/32768.0f);
	}
	fclose(impulse);

	doFour2(buffer_h-1, NPOINTS, 1);

}
```