School of Physics and Astronomy

Senior Honours Project
Computational Physics 4

Towards a realistic, non-realtime reverberation model

Ewan Hemingway (s0673758)
20th March 2010

Abstract

Various techniques for recreating sound in reverberant spaces are examined, and by building on the image source method, a realistic reverberation simulation is developed. An alternative Monte Carlo method is proposed, and quantitative and qualitative evaluations are made both between the two models, and with reference to an actual recording.

Declaration

I declare that this project and report is my own work.

Signature: Date:

Supervisor: Dr. Jonathan Kemp

Contents

1 Introduction

Analysis of the acoustic properties of a space is an important part of architectural design. Performance spaces in particular have very specific requirements for how much reverberant sound is heard at various positions in the room, as this has a major impact on the clarity of sound. While these properties can be measured on site, this is not always an option. For example, it is clearly not practical to analyse proposed buildings in this way; this is where simulation becomes the only real alternative.

However architectural design is not the only use for reverberation simulations; they also play an important role as an effect for music production. Often musicians might take a dry recording of a voice or instrument in the studio, a room which should ideally be acoustically dead. Then a reverberation effect can be applied to give the appearance of performance in (theoretically) any space. Real-time reverberation effects may also be applied in live performance, but that is beyond the scope of this project.

Two main classes of model are presented in this report:

- Deterministic: These models take input data describing a space, and apply an algorithm resulting in the same output every time. Decisions for the directions of sound waves are made purely on a geometric basis.

- Monte Carlo (or Stochastic): A logical alternative is a model whereby the the properties of a system are discovered by random sampling.

An equal amount of time was spent investigating each category, firstly developing a technique called the *Image Source* method. While this proved to be an accurate model of reverberant sound, it lacked the flexibility to simulate more complex spaces and acoustic phenomena such as diffuse sound scattering.

An alternative method called the *Monte Carlo Ray Tracing* technique was proposed to investigate more complex room behaviour. This proved to be quantitatively consistent both with the Image Source results and with predictions from literature, at least for rectangular rooms. Simulations of the reverberation room at the University of Edinburgh showed some of the observed anomalous behaviour, though numerical predictions didn't exactly match real world recordings. This could have been in part due to inaccuracies in the recording; a failing in the model could be equally likely.

Qualitatively, the Monte Carlo output produced was a good approximation to real world reverberation, and was free of some of the spurious fluttering effects that arose in the Image Source results. All files have been made available for comparison at [1].

2 Background and Theory

Both of the two contrasting methods explored in this project are based on the same underlying physics; they differ only on how they arrive at the result. However first, we should see what exactly the concept of reverberation physically means.

When we hear a sound from a specific source in a real world enclosed space, for example a balloon popping in a cathedral, it is not just direct sound from the source to the listener that is observed. As the sound source generally radiates sound energy in all directions, what is actually heard is the direct sound followed by a series of later reflections off the various surfaces of the space (as seen in Figure 1). As the sound from these later reflections has further to travel, the intensity upon arrival will become increasingly diminished as time progresses, as sound energy is inversely proportional to the distance travelled.

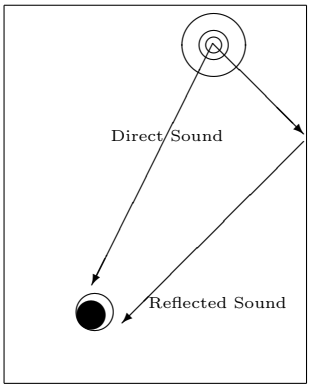


Figure 1: Room Reflections

To simulate this set of reflections digitally, we must introduce the concept of an impulse response. This is a quantitative representation of the reverberant character of a space, and is composed of a discrete set of impulses of varying amplitudes. The operation can be interpreted most intuitively by thinking about the convolution of the impulse response with a signal, which should also be discretised (e.g. a `.wav` file recording).

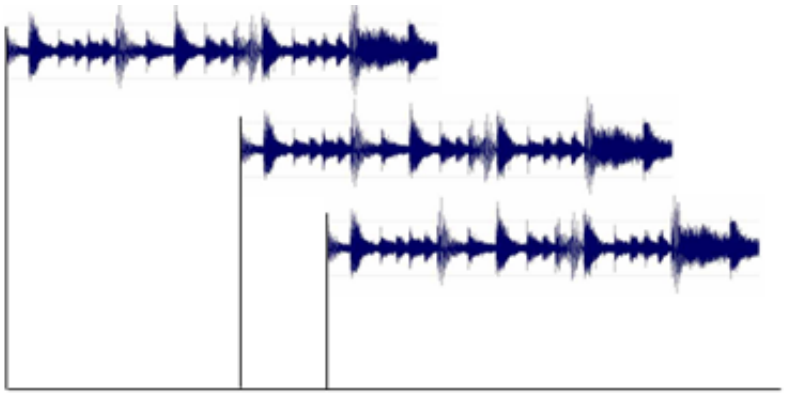


Figure 2: Demonstration of Simple Impulse Response (3 impulses)

Mathematically, these impulses are represented by delta functions. The convolution of a delta function $\delta(t-a)$ with some arbitrary function $f(t)$ is $f(t-a)$, that is the function is time-delayed by a factor a [2]. By extension, an array of impulses at various times, leads to a superposition of several 'copies' of the original function offset by various times. If we set the function to be our input signal, this produces something akin to the example given in Figure 2.

We can see that this will quickly lead to a very muddy (and unphysical) output. This is due to the fact that in the real world, sound decays as $1/r$, meaning that the amplitude

of the later impulses should be scaled appropriately. To do this, we must find how far the sound has travelled for each reflection: this is where the differences in the various approaches appear (see Sections 3.1, 3.2).

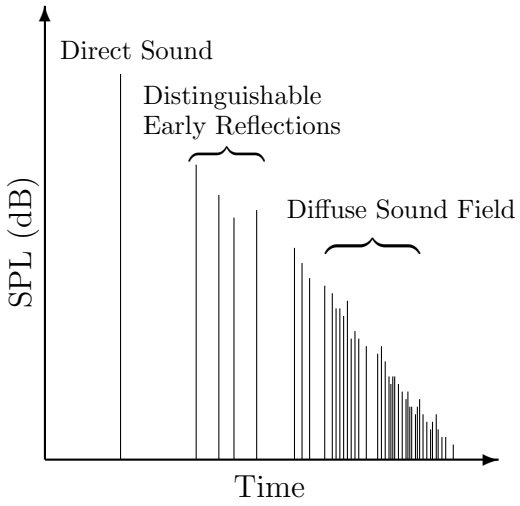


Figure 3: Theoretical Impulse Response

The concept of an impulse response is perhaps better illustrated with a concrete example (Figure 3). As expected, the logarithm of the impulse energy, a quantity that decays as $1/r$ (and therefore $1/t$), obeys a linear relationship. The direct sound travels the least possible distance and so is represented by the largest impulse. Soon after, the early reflections arrive, and having travelled slightly further, exhibit slightly lower amplitudes. Later on, we see the arrival of a large number of late reflections which appear to start to merge, creating a continuous diffuse sound field, with very few of the distinguishable echoes heard earlier [3]. If the model is to provide a qualitatively realistic output, similar behaviour will have to be replicated.

As the project title suggests, the computational time of either method is of little importance, as long as it remains within a sensible range (we don't want to be running the simulation for years!). Once an impulse response has been determined, it can be written to disk and applied to as many recordings as required.

Theoretically, the basic physics of reverberation has had a strong basis for some years. Sabine (1868 - 1919), one of the pioneers of understanding reverberation, is still ubiquitous in the field. Much of his work in *Collected Papers on Acoustics* (1922) [4] still holds up today, in particular the famous Sabine equation.

One of the first known computer based implementations of a ray tracing method was produced by Krokstad et al (1968) [5], though the lack of computing power meant that the results were not particularly accurate nor useful. Ray tracing methods had previously been solved by hand with even less accuracy [6], but this was as more of a proof of concept than anything else. Kuttruff's seminal *Room Acoustics* (1973) [2] provides one of the more detailed descriptions of the behaviour of sound in a space. He has been instrumental in many of the techniques used throughout this report, with particular interest in the modelling of diffuse reflections [7], although this is not covered in detail here.

3 Method

Armed with the theory behind an impulse response, we are ready to start simulating a space. The problem essentially boils down to finding:

(a) The delay before sound from a specific reflection arrives.

(b) The energy of the sound from that reflection.

Each of the two following methods approach this using fundamentally different techniques. Firstly, we shall examine the Image Source method.

3.1 Image Source Method

The Image Source technique works on a purely geometric basis. We can treat the sound as a ray as it travels from source to listener, an assumption whose validity is later justified (see Section 4.3). This means that it obeys specular reflection, that is, the angle of incidence is equal to the angle of reflection. By tessellating the simulated room (shown in bold in Figure 4) with a series of virtual rooms, these can be populated with virtual sound sources in the mirrored positions [8] (see the hollow circles in Figures 4 & 5). The path from listener to virtual source (marked $\times$ and $\bigcirc$ respectively) is then just equivalent to the reflected path (Figure 5).

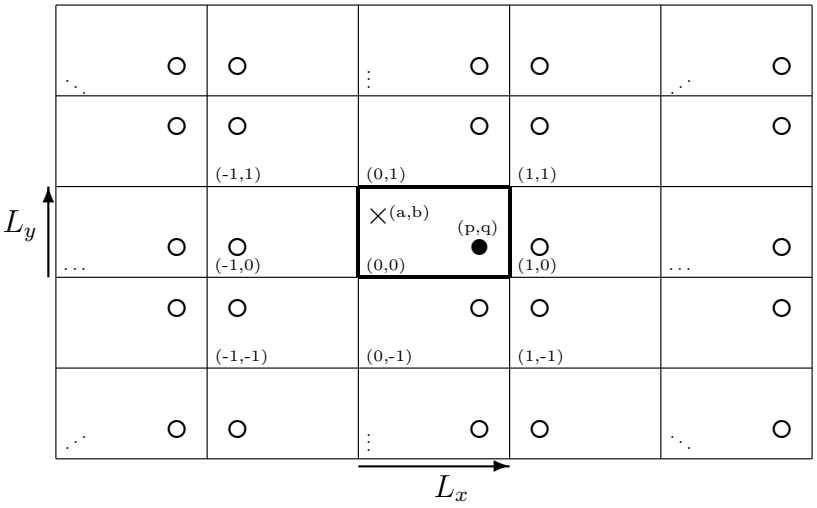


Figure 4: Image source method (adapted from [8])

As the path to the virtual source is now just a straight line, it is simply a case of taking the Euclidean norm of the difference in the two points, and we arrive at the length of the reflection, r. This gives us an an impulse in our array at time $t = r/343$ and intensity proportional to $1/r$.

The process of finding the image sources is well defined algebraically for rectangular rooms. For a virtual room labelled with indices (d, e, f), where the original room is defined as (0, 0, 0), with sound source and listener at points (p, q, r) and (a, b, c) within the room, we get a vector with the following components:

$$A_d = \begin{cases} (d+1)L_x - p - a & \text{for } d \text{ odd} \\ dL_x + p - a & \text{for } d \text{ is even} \end{cases}$$

$$B_e = \begin{cases} (e+1)L_y - q - b & \text{for } e \text{ odd} \\ eL_y + q - b & \text{for } e \text{ is even} \end{cases}$$

$$C_f = \begin{cases} (f+1)L_z - r - c & \text{for } f \text{ odd} \\ fL_z + r - c & \text{for } f \text{ is even} \end{cases}$$

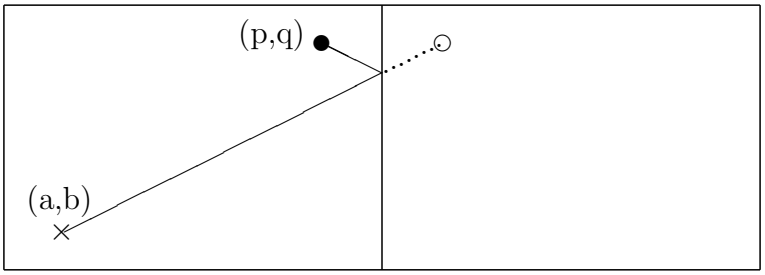


Figure 5: Equivalence of Real and Image Sources

However so far, this model fails to take into account any actual effects of absorption. On each reflection, the sound loses part of its energy proportional to $(1 - \alpha)$ where α is a property of the surface called the absorption coefficient, which ranges from 0 (full reflection) to 1 (total absorption). Therefore for n reflections, this means a factor of $(1 - \alpha)^n$ must be applied. We can easily tell the number of reflections by summing the absolute values of the image room's indices:

$$\mathrm{n} = |d| + |e| + |f|$$

This assumes that the absorption coefficient of the surface has a flat frequency response, which is very rarely the case [9, 10]. This is remedied in a later version of the code, see Section 4.1.1 for details.

3.2 Monte Carlo Ray Tracing Method

The Monte Carlo ray tracing method takes an altogether different approach. Using the idea introduced above of treating sound specularly, the model fires out a ray in a random direction, reflects it appropriately, and repeats this, tracing the ray as it bounces around the room. Once the its energy falls below a certain threshold, the path is terminated. To fully understand how the mechanics of the ray tracing method works, we must look at the geometry of the system.

The walls of the room can be represented as planes, defined either by three points on the plane, or by a point on the plane and its unit normal. If points $\vec{p_1}$, $\vec{p_2}$ and $\vec{p_3}$ lie on the plane, we can find the normal by the following:

$$\hat{n} = (\vec{p_1} - \vec{p_2}) \times (\vec{p_1} - \vec{p_3})$$

The ray itself is similarly defined by a unit direction vector $\hat{d}$ and a source point, $\vec{s}$. Thus the parametric equation for any point $\vec{p}$ on the ray is given by:

$$\vec{p} = \vec{s} + t\vec{d}$$

where t can be thought of as the time. To find at what time the ray intersects the plane, we can use:

$$t = \frac{\hat{n}.(\vec{p_1} - \vec{s})}{\hat{n}.\hat{d}} \quad \text{for } \hat{n}.\hat{d} \neq 0 \text{ [11]}$$

Using this formula we can iterate through each surface, and find the time to reach it. As we are clearly only interested in solutions where we are propagating forward through time; we can straight away ignore any solutions with $t < 0$ (this will be half of the surfaces for a rectangular room). Then, we just choose the solution with the smallest t: this corresponds to the nearest plane. This then becomes the next source point.

Finally we need to reflect the ray. This is essentially just a negation of the ray's component perpendicular to the plane. It is done using the following [12]:

$$\hat{d}' = \hat{d} - 2(\hat{d}.\hat{n})\hat{n}$$

Figure 6 shows the planar view of a sample trace for a single ray in a rectangular room. We can see its behaviour is truly specular, an obvious analogy being with a laser beam reflecting round a room with mirrored walls.

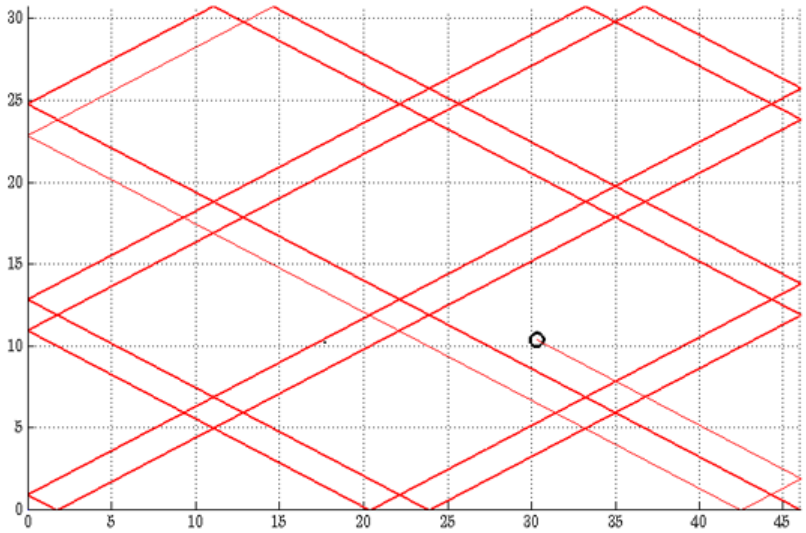


Figure 6: Example trace for a single ray as seen from the X-Z plane. Interestingly, we can see the emergence of room modes; in this case we can see one of the room's tangential modes.

However thus far, the model does not account for the listener in any way. Whereas this requirement is filled implicitly in the Image Source model, the ray-tracing case requires careful consideration and is discussed in depth in Section 4.2.2.

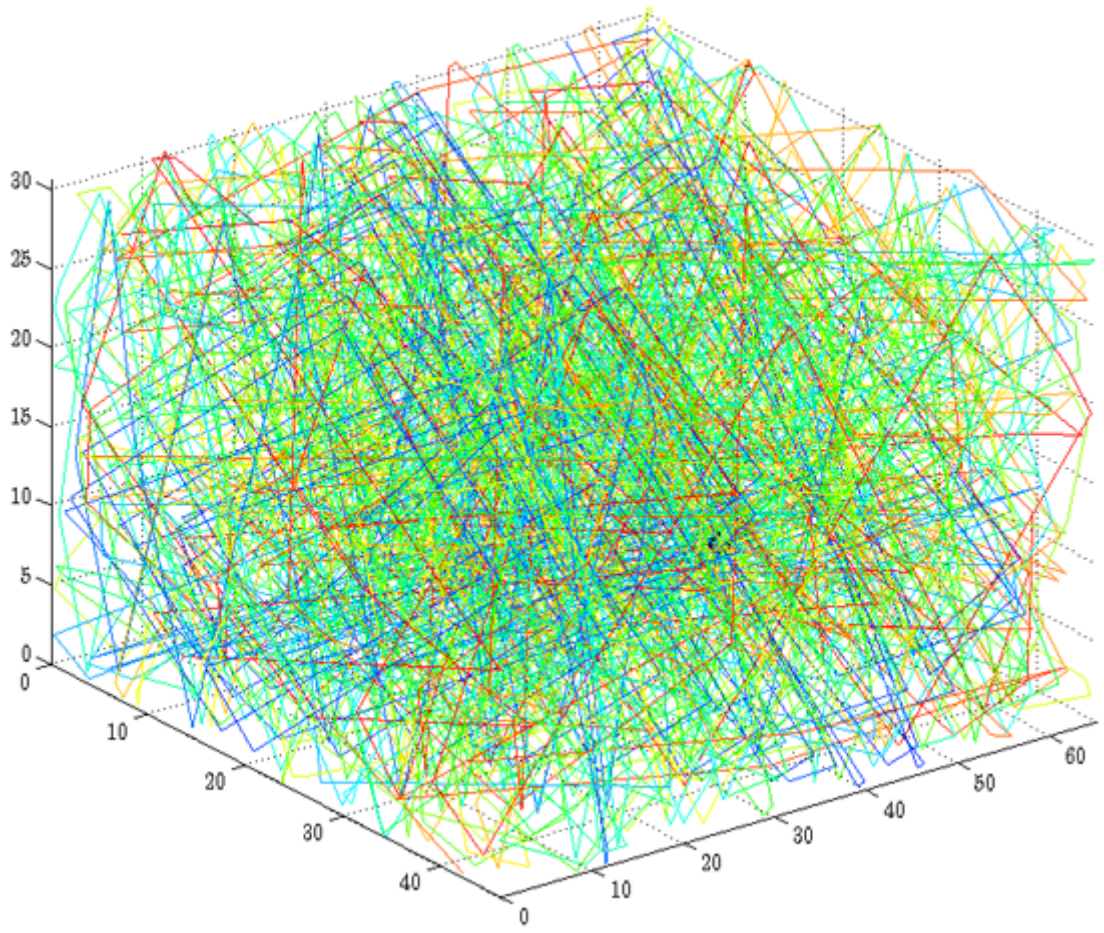


Figure 7: Trace pattern for multiple rays in a rectangular room. Each ray here is represented by a different colour.

With proper detection in place, we can use both the time spent travelling by the ray and its remaining energy to contribute to the impulse response. When this process is repeated many thousands of times, a picture of the room's many reflections is gradually built up (see Figure 7), giving us our final impulse response.

4 Development & Discussion

While both methods in their current state can produce an impulse to a reasonable degree of accuracy, there is still much room for improvement, particularly with regard to qualitative improvements. In the following discussion, we also discuss how best to deal with some of the idiosyncrasies that occur during the transition from pen and paper to computer.

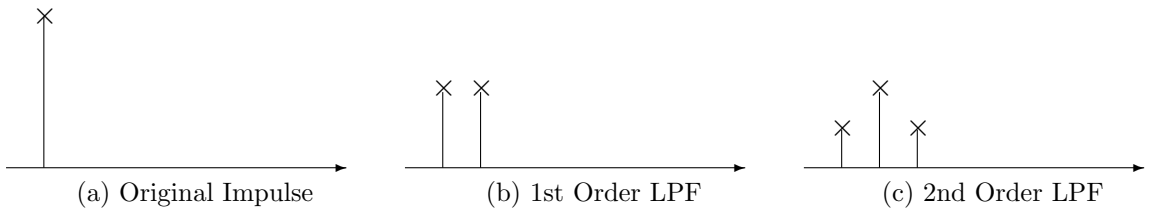
4.1 Image Source Method

4.1.1 Towards A More Realistic Absorption Profile

While the basic Image Source method outlined above produces an quantitatively accurate, on listening to the output we find that it sounds unnaturally 'harsh', that is to say there are a large presence of high frequencies. This is due to the fact that must surfaces have a absorption coefficients with a fairly strong frequency dependence: typically high frequencies are heavily attenuated [2].

While implementing an explicit frequency dependence in the simulation is well beyond the scope of this project, we can approximate the effect by incorporating a basic low pass filter (LPF) which only allows lower frequencies to pass. The filter will need to be applied on each reflection so we'll also need to use a method to find the effect of multiple applications.

One of the most efficient type of filters to use in the time domain (which is where our impulse response is defined), is the *moving average filter* [13], specifically, the two-point moving average filter. It has the basic effect of 'smoothing' the signal it is convolved with, thus reducing high frequencies, which by definition are points in the signal with a high rate of change in value. Lower frequencies, which vary comparatively much less with time are not as affected (for a complete description, see [13, Ch. 16]).



(a) Original Impulse (b) 1st Order LPF (c) 2nd Order LPF

Figure 8: Development of a LPF

Mathematically, this average is implemented using two successive delta functions in our discrete impulse response. This will produce two copies of the original signal offset by a single sample. As the signal will now have twice the amplitude, we must divide the magnitude of these responses by two. Alternatively, we can think of it as the total 'area' of the filter should remain normalised to one. Figure 8b shows the two-point moving average for the initial impulse shown in Figure 8a.

To apply the filter multiple times, we simply convolve it with the signal repeatedly. The simplest way to do this is to use Convolution Theorem. As the convolution operation

is both associative and commutative, the following is equivalent to convolving h n times with itself:

$$h_{new} = \mathcal{F}^{-1}(\mathcal{F}(h)^n) \text{ where n is number of applications/reflections.}$$

It is worth noting that as the number of reflections is a degenerate quantity, it is computationally sensible to perform the above calculation once at the start of the program then recall the values from a lookup table. The result for one reflection ($n = 1$) is shown in Figure 8c. To apply this to the actual impulse response, we simply replace each impulse with its corrected, filtered version. As this technique is not actually specific to a certain method, it can also be as easily applied to the Ray Tracing method.

4.1.2 Phasing Effects in Cuboid Rooms

If we run the Image Source model using a perfectly square room, the output produced exhibits some strange behaviour, producing a sound similar to a flanging effect (see [1] for an example). This can be somewhat be explained by thinking about the virtual rooms on the same 'axis' as the simulated room. As there is none of the randomness seen in the Monte Carlo method, these virtual source can form a definite repeating pattern, and if this is pronounced enough, it can give the perception of pitch. In square rooms. this pattern is mirrored in each dimension causing the effect to magnify, which can produces the observed fluttering effect. Additionally rooms whose walls have integer (and therefore harmonic) ratios of lengths will exhibit the pitch effect, albeit to a lesser extent.

This is just an example of the one of the problems caused by using deterministic methods to simulate physical processes which, in general, behave with at least a small element of stochasticity.

4.1.3 Flexibility

While the Image Source model performs well for rectangular rooms, it struggles to cope with more complex spaces. Generalised algorithms for non-rectangular rooms do exist, although these are generally not completely watertight [14]. There is a danger of placing virtual sources in positions where physically there is an obstruction in the way, and while visibility checks can be performed, this only moves the model back towards the ray-tracing method!

In architectural acoustics, designs for prospective concert halls can exhibit many complex shapes and obstructions, a pillar for example would be nearly impossible and certainly impractical to implement in this way; Monte Carlo methods prove to be much more appropriate.

4.2 Monte Carlo Ray Tracing Method

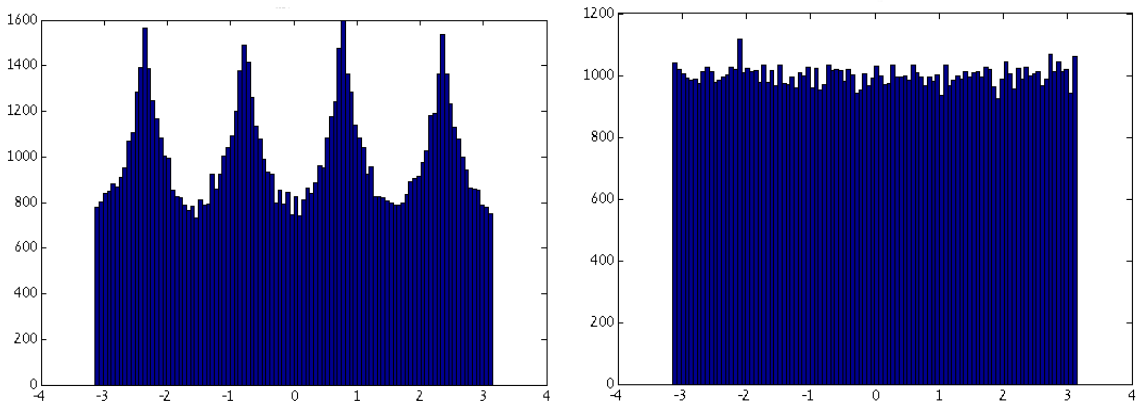
4.2.1 Angular Distribution/Control

One of the benefits of the stochastic ray tracer is the amount of control over the angular distribution of sound propagation. The most basic source to simulate is an isotropic point, that is to say sound is radiated in all directions equally. While in reality sound sources (for example musical instruments) can be fairly directional in their method of

sound production, the point source remains a fairly sensible approximation [15, pg. 104], particularly given its simplicity to implement. Naively, we might implement a random direction vector as follows:

$$\vec{r} = \frac{(x_1, x_2, x_3)}{\sqrt{x_1^2 + x_2^2 + x_3^2}} \tag{1}$$

where x_1, x_2, x_3 are independent random variables taken from a uniform distribution. However if we take a histogram plot of the distribution of angles in a plane through the source we see that the distribution is non-uniform (Fig. 9a). The above method effectively samples points in the volume of a cube: we ideally require something analogous to points on the surface of the unit sphere.



(a) Naive Direction Vector

(b) Gaussian Direction Vector

Figure 9: Comparison of Angular Distributions, each histogram shows number of rays against angle

The next most obvious method to try would be to pick polar and azimuthal angles from the ranges $0 < \theta < 2\pi$ and $0 < \phi < \pi$ respectively. However this again fails as the area element $d\Omega = sin\phi \; d\theta \; d\phi$ is ϕ dependent, resulting in higher densities of points near the poles. This can be corrected by generating [16]:

$$\theta = 2\pi u$$

$$\phi = cos^{-1}(2v - 1) \text{ where } u, v \text{ are random on (0,1)}$$

Unfortunately calls to trigonometric functions are fairly computationally expensive, particularly when repeating millions of times as expected in the ray tracing algorithm. A second method proposed by Muller [17] avoids this cost by using variables taken from a Gaussian random distribution, and applying them to Equation 1. Basic tests in Matlab found this to be roughly 4 times faster than the trigonometric technique, and interestingly, generating random variables from the Gaussian distribution appears to be as quick as the uniform distribution (see Appendix A.3 for details of the test). Figure 9b shows the corrected distribution exhibiting the desired uniform behaviour.

The complexity of simulating even a simple isotropic source reveals interesting phenomena. The problems with incorrectly weighted distributions encountered above could actually be manipulated. A directional sound source such as a loudspeaker could be represented with much greater accuracy using a weighted distribution: it is not very realistic to trace as many rays out from the back of the speaker as from the front!

4.2.2 Ray Detection

Another complication introduced by the ray tracing method is in the method used to detect a ray. In order to get a distance or time for the impulse response, some form of observer is needed, for example, a microphone.

This is a trivial task for the image source method as the ray's path is actually determined by the detector's position, effectively behaving as an infinitesimally small point. This approach fails for the ray tracing method: the probability of an infinitesimally thin ray with a randomly chosen initial direction passing through an infinitesimally small point is very small indeed!

The most natural solution is to use a small sphere centred at the detection point. This has the advantageous property of possessing the same cross sectional profile from any direction. Collision events are also reasonably simple to detect geometrically. We can express a point $\vec{p}$ on the surface of a sphere of radius ρ, centred at position $\vec{f_s}$ by:

$$|\vec{p} - \vec{f_s}|^2 = \rho^2$$

To check for collision, we substitute in the parametric equation for a line with source $\vec{s}$ and unit direction vector $\hat{d}$:

$$|s + t\hat{d} - \vec{f_s}|^2 = \rho^2, \text{ using } \vec{p} = \vec{s} + t\hat{d}$$

Expanding this gives us a quadratic of the form:

$$at^2 + tb + c = 0$$
$$\text{where } a = \hat{d}.\hat{d} = 1,\ b = 2(\vec{s} - \vec{f_s}).\hat{d},\ c = (\vec{s} - \vec{f_s}).(\vec{s} - \vec{f_s}) - \rho^2$$

This is solved in the usual way:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

We can tell from the discriminant alone whether or not a collision has occurred. By using this check before solving the quadratic, we can save ourselves unnecessary computation:

$$b^2 - 4ac = \begin{cases} < 0 & \text{ray misses sphere} \\ = 0 & \text{ray exactly tangential to sphere} \\ > 0 & \text{ray passes through sphere} \end{cases}$$

We are only interested in the latter two solutions, from these we can calculate the time from the ray's source to the sphere, and therefore, also the distance. As the radius ρ tends to zero, the point of intersection can be approximated by the point at the centre of

the sphere, $\vec{c}_s$, meaning the quadratic need not be explicitly calculated. The only caveat is that when the ray is first fired from the source, it is possible for a negative time solution to be erroneously used, that is to say that an intersection would occur should we trace the ray backwards through its origin, but this can easily be eliminated with a quick check in the code.

The difficulty is in choosing an appropriate size for the detector: too large and a disproportionately large number of rays will hit straight away, leading to an unphysical density of early reflections. Conversely, a detector with too small a radius will simply not pick up enough information to produce a realistic impulse response. Studies by Xiangyang et al. [18] suggest a technique for determining the optimal volume (and therefore radius) for the detector, V_r.

$$V_r = \frac{10V}{N} \implies r = \left(\frac{15V}{2\pi N}\right)^{\frac{1}{3}}$$

However this is not claimed to work in general, and application of this to the presented ray-tracing model leads to unrealistically high initial densities of impulses, and not enough late reflections. However in the same paper, Xiangyang et al. suggest a second technique whereby the radius of the detector depends of the length of time the ray spends travelling:

$$\rho = ct_{ray}\sqrt{\frac{2\pi}{N}} \text{ where N is the number of rays, c speed of sound} \tag{2}$$

As the probability of detecting a ray is loosely proportional to ρ^3, Eq. (2) has the effect of changing the emphasis of the sampling process so that the late diffuse sound field is given more precedence. However we must be careful when calculating the radius dynamically that we don't make the sphere extend outside the room. The authors also recognise that it is sensible to account for the volume of the room, and suggest multiplying Eq. (2) by a factor proportional to $\log_{10}(V)$. Ultimately the complexity of the reverberation problem means that each situation should ideally analysed carefully to choose a radius; the above formulae should serve more as guides than definite rules.

4.2.3 Ray Termination

While the Image Source method which has a well defined finite path from source to detector, the rays created in the Monte Carlo Ray Tracing method will propagate indefinitely. Physically, sound the reflects around the room until it possesses only a negligible amount of energy; this is a sensible model to implement in the algorithm. This can be achieved by explicitly calculating the energy at each iteration, but this additional computation can be avoided by setting a termination length d_{term} instead (the distance travelled by the ray is tracked throughout its journey, its energy is only calculated on detection).

As computation time increases linearly with termination length, we must give careful thought to the numerical value for this cutoff. Too long and unnecessary computational time is wasted; too short and we may miss a room's true behaviour. Relative to some initial loudness of the signal, anything more than $\sim 30dB$ below this will be unlikely to have a noticeable effect on the output [15, Chap. 3]. In terms of energies in the impulse response, this corresponds to points with less than 0.1% of the initial impulse. As the energy $g = 1/r$, this gives us a termination length of around 1000m.

However the above assumption does not account for multiple (otherwise insignificant) impulses summing at the same point in time to form a significant contribution, so it is sensible to choose a slightly bigger d_{term}. Like most of the ray tracing parameters we've seen so far, this value is also not completely general. Very large spaces will be undersampled if the ray is terminated too soon. Additionally, spaces with abnormally long reverberation times (see Section 5.2) will require *at least* $d_{term} > cR_t$ metres for proper simulation.

4.2.4 Breakdown in Pathological Cases

While the ray tracing algorithm is geometrically watertight, it is possible that the technique could break down when implemented on computer. Since we can only solve the intersection equation with a finite amount of precision[1], it is possible for the point of reflection to be an infinitesimal distance off the plane (and therefore outside the volume). This means on the next iteration, the ray will reflect off the *outside* edge and will be lost bouncing around space! This is easy enough to rectify: we can simply exclude the reflecting plane from being counted on the next repetition of the algorithm. However, this serves to demonstrate the care that must be taken when translating a continuous theoretical model into a discrete computing environment.

4.3 Suitability of Ray Based Methods

Both the Image Source and Monte Carlo Ray Tracing methods can be classed as ray based methods, although the latter is perhaps a more explicit example. The ray-like approximation of sound is valid for reasonably high frequencies, where wave-like phenomena such as diffraction can be safely ignored [20]. At low frequencies where the wavelength λ is of the same order as the length of the room, the model is less accurate. Additionally, λ must be larger than the roughness of the room's surfaces otherwise pure specular scattering is not observed [2, pg. 60].

Although pseudo standing-wave patterns may be observed in the Ray Tracing model (see Figure 6), in general true wave-like is not reproduced. Notably, any phase based effects such as constructive/destructive interference are missed. This can be particularly important if the source or listener is at a nodal or anti-nodal point of a room mode [15]. While there is growing interest in wave based solutions for solving acoustical problems, particularly at low frequencies [21], ray based methods (or derivatives thereof) remain one of the most popular solutions.

5 Results

5.1 Rectangular Room

The first space to be analysed is an idealised rectangular room with a constant absorption coefficient $\alpha = 0.3$ and dimensions $45.9623 \times 65.23354 \times 30.65432$ [2]. Physically this corresponds to a reasonably large space such as a factory floor, an environment in which

[1] Matlab uses double precision, which can lead to rounding errors typically of the order 10^{-16} [19].

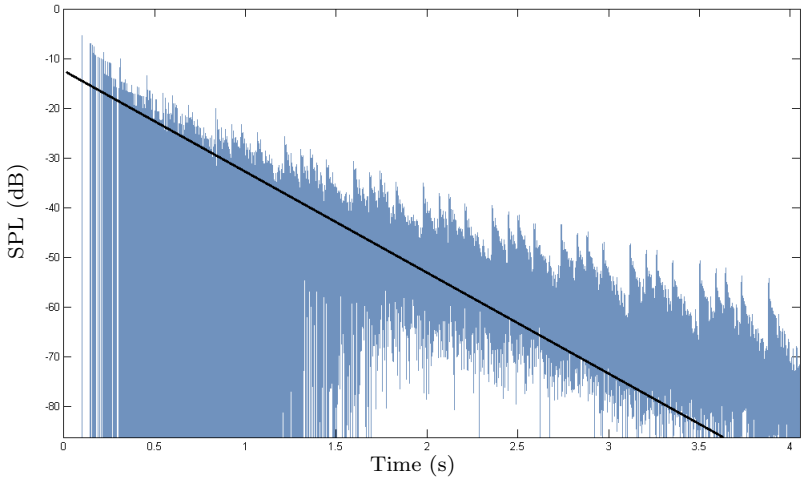
[2] The high precision used for the dimensions is to avoid effects described in Section 4.1.2.

reverberant noise levels can be of great importance. The simulation could just as easily be applied to a small room but the effect would be nowhere near as noticeable.

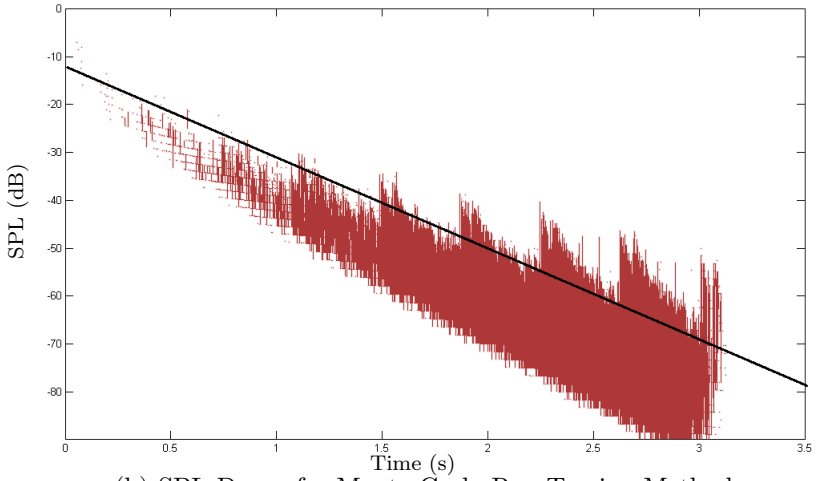
These dimensions gives a surface area of $2\times(L\times W+L\times H+W\times H)=8.97\times10^3$ m^2, and a total volume of 9.19×10^4 m^3. We can use Sabine's formula [15] to predict the amount of time required after an impulsive sound for the SPL to drop by 60dB. This quantity is the most common property to measure of a reverberant space and is known as the *reverberation time*, R_t. Using the above parameters:

$$R_t=\frac{0.161V}{S\alpha}=\frac{0.161\times9.19\times10^3}{8.97\times10^3\times0.3}\approx3.24 \text{ secs}$$

We can obtain the equivalent quantity by converting the simulated impulse into units of SPL, i.e in decibels. This is done by taking $10\log_{10}(\text{iir})$. The results are presented in Figure 10.



(a) SPL Decay for Image Source Method



(b) SPL Decay for Monte Carlo Ray Tracing Method

Figure 10: Reverberation Time For Rectangular Room. Black line shows the Sabine prediction of $R_t=3.24$s. Note the definite cutoff described in Section 4.2.3

While the amount of 'noise' varies greatly due the difference in densities, both show a *reasonably* clear linear relationship. Taking the line of best fit through the peak values, we can obtain values of around $R_t = 3.5$ and $R_t = 3.4$ seconds for the Image Source and Monte Carlo methods respectively. The confidence in these values is fairly low however, with either value only being accurate to around ± 0.15 seconds or so. It is worth mentioning that the Sabine method should not be taken as an exact prediction of R_t. The formula is not without its criticisms [22], but it remains a sensible approximation is this scenario. A good way is of course to physically record and analyse the impulse response of the room, but as in this scenario the space is not based on a specific real world environment, this is not an option.

5.2 Reverberation Room

Fortunately, the second space of interest has a measurable impulse response. The reverberation chamber at the University Edinburgh is a completely asymmetric room. The walls are all of different lengths and heights, and no two surfaces are parallel. When discrete standing wave behaviour occurs, for example in rectangular rooms, there is no net energy propagation. This has the effect of shortening the reverberation time, as the spread of sound energy throughout the room is attenuated. In the reverberation room this effect is negligible, so we expect R_t to be much longer. Additionally, all the surfaces in the room are made from a highly reflective plaster material on solid hard walls, and although the exact absorption coefficient is not known, its value can be expected to be very low, around $\alpha = 0.01$ [15, pg. 531].

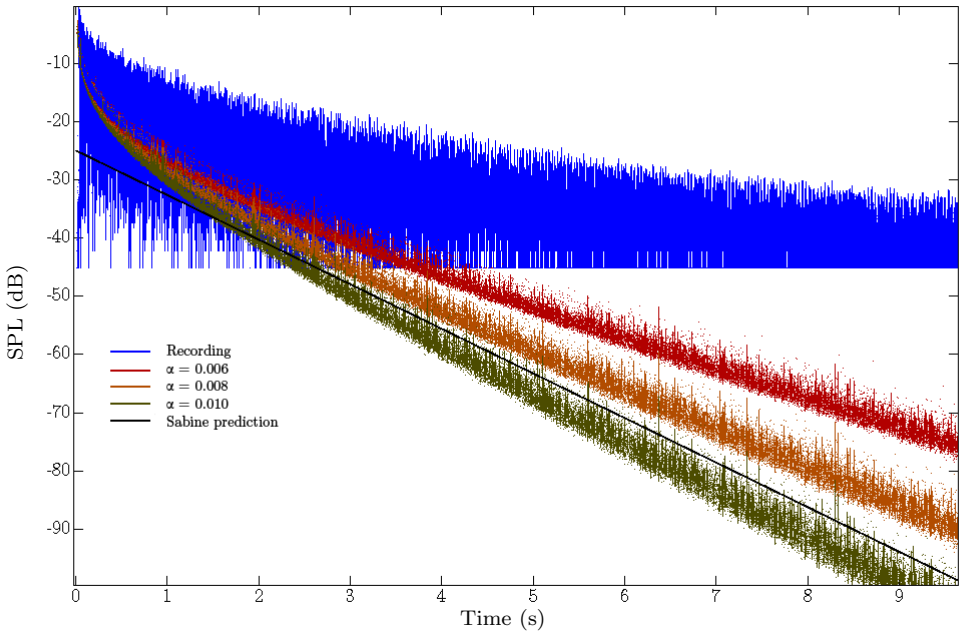


Figure 11: Reverberation Time For Rectangular Room. Black line shows the Sabine prediction of $R_t = 3.24$s.

Figure 11 shows the SPL plots for simulation runs at various absorption coefficients. The dependence of the absorption coefficient α on R_t is explicitly shown here; we can see that it is one of the strongest contributing factors in finding R_t. No method appears to match the experimental recording however. This could be due to inaccuracies in the recording and processing of the impulse response. Experimental determinations of R_t are generally extrapolated from smaller drops in SPL, and in this case, we can only see a change of about 20dB before the background noise level is reached. Ambiguities over when this 'noise floor' becomes noticeable can detract from the confidence in the result. The Sabine plot for $\alpha = 0.01$ is also shown for interest although it should be noted that its effectiveness for non-rectangular room is a point of concern [22]. We can see that the reverberation times for $\alpha = 0.006, 0.008, 0.010$ are approximately $R_t = 11.5, 9.5, 7.5$, although confidence in these values is again low, with a possible error margin of around ± 0.2 seconds. The extrapolation from the experimental data leads very roughly to $R_t \approx$ 16 seconds, though the confidence is even lower for this case at about ± 0.5 seconds.

Although these values don't match the experimental value for R_t exactly, they do exhibit the unnaturally large reverberation times expected in such a room. This is an encouraging sign for the effectiveness of Monte Carlo method, although work is clearly still required before it can be applied to an arbitrary space.

6 Conclusions

We have seen that the accurate simulation of reverberant sound is a complex problem. While still no single model has been found to provide a fully accurate representation, the techniques discussed in this report have come close.

The Image Source model has been demonstrated as a highly efficient way of producing a realistic impulse response, particularly with the addition of 'cosmetic' improvements such as low pass filtering. Although the implementation described in this report limits its application to rectangular rooms, it is well known to have been extended to more irregularly shaped rooms.

Although computationally much more intensive than the Image Source method, the Monte Carlo Ray Tracing method has proved to be much more flexible. By supporting non-omnidirectional sound sources and asymmetric rooms, it has shown itself to be much more applicable in a real world environment. There is also much more scope for improvement with the model. Kuttruff describes how diffuse reflections can also be modelled using ray-tracing methods, albeit at a much greater computational cost [2]. Many cutting edge reverberation simulations are based on the basic ray model, exploring everything from pyramidal and cone based schemes to particle based simulations [6].

Numerically, both models are in good agreement for the rectangular room, predicting $R_t \approx 3.4$ and $R_t \approx 3.5$ seconds for Image Source and Monte Carlo methods respectively, although these values lie slightly above the theoretical value of $R_t \approx 3.24$ seconds determined by Sabine's method. For the reverberation room, while characteristically long reverberation times are observed, they do not numerically match the experimentally measured value for R_t.

Qualitatively, both models also create a fairly realistic sense of space (see [1] for samples). The Image Source method still appears to suffer from a milder version of the flutter effect describe in Section 4.1.2, although it is much less noticeable. The Monte Carlo method produces a much smoother sounding output, and between the two models, it seems to give a more realistic sense of the space. It does suffer however from a slight low frequency 'thwump' sound just after the early reflections. While this could be removed post-convolution using audio editing software, this still suggests that the model is far from perfect, particularly for early reflections.

Ultimately, neither model provides a complete description of a space in its current state. The accuracy of the early reflections worked out by the Image Source model and the smoothness of the later diffuse sound field generated by the Monte Carlo method suggest that perhaps a 'hybrid' model would be the ideal solution to the problem.

7 Acknowledgements

Thanks to Dr. Jonathan Kemp his enthusiasm and time spent supervising the project.

References

[1] Ewan Hemingway. Audio Results and Code from Image Source and Ray Tracing Simulations. `http://www.ewanhemingway.co.uk/reverberationproject`, March

2010.

[2] H. Kuttruff. *Room Acoustics*. Applied Science Publishers, London, 5th edition, 2009.

[3] M. Long. *Architectural Acoustics*. Academic Press, 3rd edition, 2005.

[4] Wallace Clement Sabine. *Collected papers on acoustics*. Harvard University Press, 1922.

[5] A. Krokstad, S. Strom, and S. Srsdal. Calculating the acoustical room response by the use of a ray tracing technique. *Journal of Sound and Vibration*, 8(1):118 – 125, 1968.

[6] David Oliva Elorza. *Room acoustics modeling using the ray-tracing method: implementation and evaluation*. PhD thesis, University of Turku, 2005.

[7] Murray Hodgson. Evidence of diffuse surface reflections in rooms. *The Journal of the Acoustical Society of America*, 89(2):765–771, 1991.

[8] Jonathan Kemp. Image source reverberator. `http://www2.ph.ed.ac.uk/~jkemp/dsp/`, 2010.

[9] Leo L. Beranek and Takayuki Hidaka. Sound absorption in concert halls by seats, occupied and unoccupied, and by the hall's interior surfaces. *The Journal of the Acoustical Society of America*, 104(6):3169–3177, 1998.

[10] E.H. Donald. *Musical Acoustics*. Brooks/Cole Pub. Co., 3rd edition, 2001.

[11] Ray plane intersection. `http://www.siggraph.org/education/materials/HyperGraph/raytrace/rayplane_intersection.htm`, 2010.

[12] E. W. Weisstein. Reflection. MathWorld – A Wolfram Web Resource `http://mathworld.wolfram.com/Reflection.html`, 2010.

[13] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub., 1st edition, 1997.

[14] Funkhouser et al. A beam tracing method for interactive architectural acoustics. *The Journal of the Acoustical Society of America*, 115(2):739–756, 2004.

[15] Murray. Campbell and Clive A. Greated. *The Musician's Guide to Acoustics*, pages 525–548. Dent, London, 1987.

[16] E. W. Weisstein. Sphere point picking. MathWorld – A Wolfram Web Resource `http://mathworld.wolfram.com/SpherePointPicking.html`, 2010.

[17] Mervin E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, 1959.

[18] Xiangyang Z. On the accuracy of the ray-tracing algorithms based on various sound receiver models. *Applied Acoustics*, 64:433–441(9), April 2003.

[19] MATLAB Manual - Accuracy of Floating Point Data. `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/f2-12135.html`, March 2010.

[20] L Savioja. Modeling techniques for virtual acoustics. *J. Audio Eng. Soc*, 47:675–705, 1999.

[21] D. Botteldooren. Finite-difference time-domain simulation of low-frequency room acoustic problems. *The Journal of the Acoustical Society of America*, 98(6):3302–3308, 1995.

[22] Murray Hodgson. Experimental evaluation of the accuracy of the Sabine and Eyring theories in the case of non-low surface absorption. *The Journal of the Acoustical Society of America*, 94(2):835–840, 1993.

A Appendices

A.1 Image Source Code Listing

The following is the MATLAB implementation of the Image Source method.

```
clear;

% read file in, store in vector x
[x,Fs,bits] = wavread('input');

% check for stereo file
% and convert to mono if necessary
xs = size(x);
if xs(2) == 2
    x = x(:, 1) + x(:, 2);

    % normalise
    x = x./max(abs(x));
    x = x.*(1-2^(1-bits));
end

% number of rooms in each direction
N = 20;

% dimensions of room
L = [45.9623 65.23354 30.65432];

% absorbtion coefficient
alpha = 0.7;

% listener position
listenerPosition = [17.645 15.123 10.198748];

% source position
sourcePosition = [30.256 40.7124 10.370239];

% find time of last reflection
```

```
tf = norm(((N+1)*L + sourcePosition - listenerPosition))/343;
% and allocate space for IIR vector
iirImage = zeros(1,ceil(tf*Fs)+400);

% store the initial lowpass impulse
% choose
Nf = 3*N+1;
h = [0.5; 0.5; zeros(Nf-2,1)];

% create lowpass filter which increases in degree
% for each wall passed. vector doesn't wrap
% as long as Nf > numReflections
lpf = zeros(Nf, 3*N + 1);
lpf(1,1) = 1;
for numReflections = 1:3*N
    % the +1 accounts for the fact that we have to
    % store a vector for the case of 0 reflections:
    % Matlab doesn't support zero-based indexing...
    lpf(:,numReflections+1) = ifft(fft(h).^(numReflections+1));
end

% look at each dimension
for d = -N:N    % x indice
    for e = -N:N    % y indice
        for f = -N:N   % z indice

            % store indices in vector
            i = [d e f];

            % find lengths using both odd and & even formulae
            oddR = (i+1).*L - sourcePosition - listenerPosition;
            evenR = i.*L + sourcePosition - listenerPosition;

            % then use masks to chose whether to use the odd or even lengths
            maskOdd = mod(i,2);
            maskEven = mod(i,2) == 0;

            % find response time using distance to image
            R = norm(oddR.*maskOdd + evenR.*maskEven);
            responseTime = R/343;

            % find number of collisions to reach room (d,e,f)
            numReflections = sum(abs(i));

            % calculate amplitudes of response
            g = (1/R).*alpha.^numReflections;

            % find the position within the iir vector
            position = floor(responseTime*Fs) + 1;

            % sum contribution onto IIR vector
            iirImage(position:(position+Nf-1)) = ...
            iirImage(position:(position+Nf-1)) + g.*lpf(:,numReflections+1)';
        end
    end
```

```
end

% setup plotting axis vector
iirsImage = 1/Fs*(1:length(iirImage));
% and normalise iir vector
iirImage = iirImage./max(abs(iirImage));

% if source is at same point as receiver,
% this is effectively an infinitely large impulse
% at t = 0. (This case does nothing interesting,
% this is just exception handling).
if(isnan(iirImage(1)))
    iirImage = 1.0;
end

% avoid clipping, should we choose to export IIR vector
iirImage = iirImage.*(1-2^(1-bits));

% plot iir vector against time
figure(1)
plot(iirsImage,iirImage)
xlabel('Time (s)');
ylabel('Amplitude');
str = 'Image Source: Impulse response for room';
str =  str + sprintf('%0.2f x %0.2f x %0.2f ', L(1), L(2), L(3));
title(str);

% perform the convolution
output = conv(x,iirImage);

% mix in some of the dry signal
x = [x; zeros(length(output)-length(x),1)]';
wet = 0.9;
output = x.*(1-wet) + output.*wet;

% find last zero...
lastZero = 0;
for j = 1:length(output)
    if output(j) ~= 0
        lastZero = j+1;
    end
end
% ...and trim trailing zeros
output = output(1:lastZero);

% normalise and write to file
output = output./max(abs(output));
output = output.*(1-2^(1-bits));

wavwrite(output,Fs,bits, 'outputImageTest')
% ... and we're done.
```

A.2 Monte Carlo Ray Tracing Code Listing

The following is the MATLAB implementation of the Monte Carlo method.

```
clear;

[x,Fs,bits] = wavread('input');

L = 6.5 % length x
W = 3 % width y
H = 3  % height  z

volume = W*H*L

% furthest distance "ray" can travel without intersection
maxDistance = norm([W L H]);

A =          [0 0 0.1; 0 0 0; 0.1 0 0];      % left wall 570cm
A(:,:,2) =   [0 0 0.1; 0 0 0; -0.146 2.80 0];      % back 280cm

A(:,:,3) =   [-0.146 2.80 0.1; 5.98 1.79 0; -0.146 2.80 0];      % right 612.5cm
A(:,:,4) =   [5.7 0 0; 5.7 0 0.1; 5.98 1.79 0];      % front 180cm

A(:,:,5) =   [0.1 0 0; 0 0 0; 0 0.1 0];      % floor
A(:,:,6) =   [5.7 0 2.4 ; 5.98 1.79 2.7; 0 0 2.4];      % roof

% preallocate array for normals for 6 surfaces
normals = zeros(6,3);
% and array of a charateristic point on each
points = zeros(6,3);

% find normals
for a = 1:6
    % find the normal
    normal = cross(A(1,:,a) - A(2,:,a), A(1,:,a) - A(3,:,a));
    % pick point on the plane
    point = A(1,:,a);

    % may as well make it the unit normal while we're here
    normals(a,:) = normal./norm(normal);
    points(a,:) = point;
end

% non-square room
normals

% room coeffecient
alpha = 0.006;

% choose number of "rays" to simulate
numbRays = 10000

% when do we terminate the ray
terminationLength = 5000
```

```
% position and direction vector of sound source
soundSource = [1.1 1.42 1]

% detector position and radius
detectorPos = [4.78 0.77 1.2]

drawRays = false;
% set up plot with sound source marked
if(drawRays)

    % generate a colour map for each "ray"
    colours = colormap(hsv);

    hold on
    figure(1)
    plot3([soundSource(1);soundSource(1)], [soundSource(2);soundSource(2)], ...
    [soundSource(3);soundSource(3)], 'ko', 'LineWidth',2,'MarkerSize',10');
    plot3([detectorPos(1);detectorPos(1)], [detectorPos(2);detectorPos(2)], ...
    [detectorPos(3);detectorPos(3)], 'ko', 'LineWidth',2,'MarkerSize',10');
    view(38,35)
    grid on
    xlabel('x')
    ylabel('y')
    zlabel('z')
end

% find the latest concievable reflection
lastPosReflection = (terminationLength + 2*maxDistance)/343
% and size the iir vector appropriately
iirMonte = zeros(1,ceil(lastPosReflection*Fs)+1000);

sqrtTwoPi = 0.05*sqrt(2*pi/numbRays);

for r = 1:numbRays

    % generate direction vector of sound source and normalise
    direction = randn(1,3);
    direction = direction/norm(direction);

    % set the first point to trace from to be the sound source
    nextPoint = soundSource;
    % track number of reflections
    numReflections = 0;

    % reset counters
    totalDistance = 0;
    nearestSurface = 0;

    % invalid surface
    invalidSurface = 0;

    % loop until the sound ray has negligible energy left
    while(totalDistance < terminationLength)
```

```
% the ray should NEVER travel more than this
% so it is a safe value to start at
shortestT = 2*maxDistance;
source = nextPoint;

rTemp = totalDistance;
if(totalDistance == 0)
    rTemp = norm(source-detectorPos);
end
% dynamically find radius
radius = 0.1*rTemp*sqrtTwoPi;

% a is always 1
b = 2.*dot((source-detectorPos),direction);
c = dot((source-detectorPos),(source-detectorPos)) - radius.*radius;
discriminant = b^2 - 4*c;

% if there is a collision
if(discriminant >= 0)

    % if the radius is small enough
    % the intersection point can be approximated
    % by the sphere centre.
    % similarly if this is the ray's first iteration
    % trace it directly to the source to get a clean
    % impulse, for positive time solutions
    if(radius < 0.01 || (totalDistance == 0 && -1*b > sqrt(discriminant)))
        intersect = detectorPos;
    elseif (totalDistance ~= 0)
        t = (-b + sqrt(discriminant))/2;
        intersect = source + t.*direction;
    else
        intersect = 0;
    end

    if(intersect ~= 0)
        distToMic = totalDistance + norm(intersect-source);
        responseTime = distToMic/343;
        g = (1/distToMic)*(1-alpha)^numReflections;
        position = floor(responseTime*Fs) + 1;
        iirMonte(position) = iirMonte(position) + g;
    end
end


% this is the time component of the ray
% declared here to preserve scope
t = 0;

% for each plane
for a = 1:6

    t = dot(normals(a,:),(points(a,:) - source)) / dot(normals(a,:), direction);

    % we are only interested the smallest solutions
```

```
            % which propagates forward in time
            if(t > 0 && t < shortestT && a ~= invalidSurface)
                shortestT = t;
                nearestSurface = a;
            end
        end

        invalidSurface = nearestSurface;
        p = source + shortestT.*direction;

        dist = norm(p-source);

        nextPoint = p;

        % we can trace the ray in 3D
        if(drawRays)
            plot3([source(1);nextPoint(1)], [source(2);nextPoint(2)], ...
            [source(3);nextPoint(3)], 'Color', colours(5*r, :));
            pause(0.1)
        end

        % track the distance travelled so far
        totalDistance = totalDistance + dist;

        % mirror direction
        direction = direction - 2*dot(direction, ...
        normals(nearestSurface,:)).*normals(nearestSurface,:);

        % increment reflection counter
        numReflections = numReflections + 1;

    end

    % give the user a chance to cancel during long simulations
    pause(0);
    sprintf('Total distance (for ray %i) is %d, with %d reflections', ...
    r, totalDistance,numReflections)
end


% find last zero...
lastZero = 0;
for j = 1:length(iirMonte)
    if iirMonte(j) ~= 0
        lastZero = j+1;
    end
end
% ...and trim trailing zeros
iirMonte = iirMonte(1:lastZero);
% normalize
iirMonte = iirMonte./max(iirMonte);
% set up plotting vector
iirsMonte = 1/Fs*(1:length(iirMonte));

% uncomment below to output file
```

```
% output = conv(x, iirMonte);
% output = (output.*(1-2^(1-bits)))./max(abs(output));
% wavwrite(output,Fs,bits, 'outputMonte')

% plot results
figure(1)
plot(iirsMonte, iirMonte);
str = sprintf('Monte Carlo: Impulse response for room %0.2f x %0.2f x %0.2f ', W, L, H);
title(str);
str = 'done'
```

A.3 Testing Random Number Generator Efficiency

The following code was used to test the efficiency of the various random number generation techniques. The results were:

- 100,000 direction vectors from uniform distribution took 0.34 seconds.
- 100,000 direction vectors from Gaussian distribution took 0.32 seconds.
- 100,000 direction vectors using the Weisstein correction [16] distribution took 1.15 seconds.

```
clear;

numbAngles = 100000;
angles1 = zeros(1,numbAngles);
angles2 = zeros(1,numbAngles);
angles3 = zeros(1,numbAngles);

tic
for i = 1:numbAngles
    % generate 3 uniform random numbers on (-1,1)
    direction = rand(1,3)*2 -1;
    direction = direction/norm(direction);
    angles1(i) = atan2(direction(2), direction(3));
end
toc

tic
for i = 1:numbAngles
     % generate 3 Gaussian random numbers on (-1,1)
    direction = randn(1,3);
    direction = direction/norm(direction);
    angles2(i) = atan2(direction(2), direction(3));
end
toc

tic
for i = 1:numbAngles
    % pick 0 < phi < 2*pi
    phi = rand(1,1)*2*pi;
```

```
    theta = acos(2*rand(1,1)-1);
    direction2 = [sin(theta)*cos(phi) sin(theta)*sin(phi) cos(theta)];
    angles3(i) = atan2(direction2(2), direction2(3));
end
toc

figure(1)
subplot(1, 2, 1), hist(angles1,100);
title('Angular Distribution: Naive Direction Vector Method');
subplot(1, 2, 2), hist(angles2,100);
title('Angular Distribution: Gaussian Method');
```