COS 526 Final Project Report: Acoustic Modeling by a Modified Image Source Method

Chris Tralie
Kenneth Jenkins

January 19, 2011

1 Introduction

1.1 Purpose

The purpose of this project is to simulate the acoustics of a virtual environment by borrowing techniques from computer graphics and digital signal processing. This has many applications. Constructing virtual environments and hearing what they sound like can be used in acoustic prototyping of expensive environments, such as concert halls. If a successful virtual auralization technique is implemented, the designer can tweak parameters on the fly instead of having to spend time constructing environments in real life. It can also be used to find the optimal parameters of damping materials needed in conference rooms to increase sound quality, along with many other environments. In addition, an accurate acoustic simulation can be used to hance the virtual reality experience in general, which has applications in everything from emergency response training to video games. In sum, being able to simulate the acoustics of arbitrary environments is widely applicable.

1.2 Basics of Sound

Although it may seem tempting to directly adapt light propagation techniques from computer graphics to model sound propagation, there are some important differences between sound and light that need to be considered. First of all, sound travels at about 343 meters/second at room temperature. This means that the time of propagation now matters, as opposed to graphics techniques which ignore the negligible propagation time of light that travels at 3×10^8 meters/second. Therefore, most techniques that model acoustics will need to worry about the *impulse response* of the environment. This is primarily what we're concerned with computing in this entire project.

Another difference between sound and light is that sound is much lower frequency, and consequently, it has a much larger wavelength. This means that we now need to worry about wave phenomena that we could ignore with light, especially diffraction. Diffraction can occur around corners to carry sound to shadow regions. Diffraction can also occur around obstacles in a scene for low frequency waves. We will implement diffraction based on an approximation described in [5].

One very subtle difference between light and sound is that while the intensity of light falls off as $\frac{1}{r^2}$, the magnitude of sound falls off as $\frac{1}{r}$. This isn't immediately apparent because they are both spherical wave phenomena and, as such, are both expected to project their energy over the surface area of a sphere, which is proportional to r^2. Indeed, the intensity of sound falls off as $\frac{1}{r^2}$. However, our ears are sensitive to changes in *pressure*, which is the square root of intensity. Therefore, our ears percieve a drop off of $\frac{1}{r}$ in sound, and this is what is stored in sound files. This means that we will model an attenuation of $\frac{1}{r}$ in our impulse responses.

In the physical world, sound reflection and transmission is highly frequency dependent. For instance, many solid materials act as lowpass filters for transmitting sounds, since the crystal lattice has too much intertia to transmit high frequencies. In our application, we store 20-dimensional tranmission and reflection coefficients along with every material. Each coefficient represents the material's affect on the sound wavefront in a band 1000hz wide. For instance, the third band would represent the reflection/transmission of 3000-4000hz signals.

In spite of the differences between light and sound, many of the same techniques can still be used. Particularly in the case of frequency-dependent transmission and reflection, 20-dimensional ray packet can be traced similarly to how a 3-Dimensional RGB packet was traced for visual ray tracing.

1.3 Past Work

One approach to this problem that's been tried in the past is to actually solve the Helmholz wave equation directly by splitting up space into a set of volumes that can be solved analytically, and modeling the interface between the boundaries of the sub-volumes [6]. However, this is computationally expensive and difficult to implement in practice.

More practical approximation methods usually make use of scene geometry. For instance, ray tracing can be extended to the sound domain to approximate paths taken by sound [2] [6]. This class of methods tends to be the simplest to implement, but also the most approximate. Many paths are missed, and there is also a high potential to count the same path more than once, thereby biasing the result.

Beam tracing is a technique that tries to overcome some of the problems with ray-tracing [4]. Instead of reflecting infintesimally small rays at boundary surfaces, the techniques reflects volumetric frustra. This is able to reprsent infinitely many more paths, and diffraction can be implemented naturally with

this technique. However, this techniques is difficult to implement in practice so we did not consider it.

Image Sources is another class of techniques that can capture all specular reflections up to a certain order between a source and a listener. The motivation behind this is that sound reflections are mostly specular, so if we had to pick one type of path to model it should be specular. We ultimately implemented a modified version of image sources, which will be described shortly. One problem is this technique does have an exponential blowup as the order of reflection paths increases, but we decided to cap the order of reflections at 3, which seemed to be adequate for getting a general feel for the acoustics of a scene.

2 Our Combined Image Source/Ray Tracing Method

Our main is to find paths of specular reflection between source and listener for a particular scene geometry. For each path found, we use the total distance of the path, as well as the product of all the k_s values for the materials along the way, to compute that path's contribution to the impulse response of the scene. We find these specular paths by ray-tracing, but with several key differences from ray-tracing as typcially used for computer graphics.

For an ordinary (visual) ray-tracer, we might typically cast one or more rays per pixel for the image we wish to construct, and compute a radiance value along the ray by examining the positions of the lights in the scene. We might use some illumination model, which yields diffuse and specular lighting.

Instead, in this application we are less concerned with diffuse reflections, and so we need to find the exact paths of specular reflection. If we simply cast rays about blindly, the chance that a ray from the receiver will directly hit the source (or vice versa) is very slim.

For this reason we employ an image source method, as in iSound [5]. The motivation is this: consider two points A and B which are to one side of a planar reflecting surface P. If we want to find the path of exact specular reflection from A to B, we make use of the fact that for specular reflections, the angle of incidence equals the angle of reflection. Hence we must solve for the point on the plane such that these angles are equal. The insight leading to the image source method is that this problem becomes simple if we just reflect point A about P, to yield a point A', and then intersect the ray $\overrightarrow{A'B}$ with P. See Figure 1.

This method can easily be extended to handle higher-order reflections, if the order of the faces in the reflecting path is known. We can recursively reflect the source image, and trace rays through all of the faces along the path to solve for the exact point of reflection on each face. The problem then becomes, how do we find the order of faces to consider?

The approach we take is to recursively find "visibility sets," for each possible face belonging to a specular path. First, we compute a zeroth-order visibility set from the sound source, by casting rays uniformly distributed about a sphere centered on the source. For every face intersected by one of these rays, we add it to the set, which represents every face visible from the source.

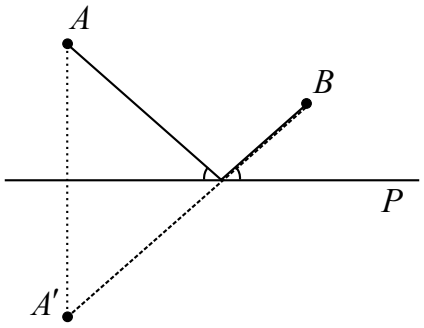


Figure 1: Using image source A' to compute the path of exact specular reflection. To validate this path, rays are traced from B to the point of reflection and then from that point to A, to ensure that no other object is blocking the actual path.

For each one of these faces, then, we reflect the source about the plane of the face (to get the corresponding image source) and trace rays *through the reflecting face* to get a first-order visibility set for that face. This is a simple form of pruning that does not appear to be mentioned in the literature that we have read on the image sources method, which selects paths which are at least plausible. At every depth in the recursion, we also add as a candidate path the straight line-of-sight path to the target receiver. Then when we come back up the call tree, at every level we check that the current portion of the path found so far is unobstructed.

2.1 Extensions

The image sources method as described above only computes paths of direct specular reflection. If we adopt a notation commonly used for expressing types of paths in graphics ray tracing, we could say that the paths are of type S* (S for specular reflection.) With a few simple tweaks to the above algorithm, we can support paths of type S*(T*|D*), where T represents transmission of sound through a surface and D represents diffraction of sound about an edge in the scene.

We achieve this by replacing the direct-path visibility check in the recursive path tracing routine with two other routines. First, for transmission, we simply keep casting along the direct line of sight ray, and add nodes to the path at every obstacle we reach.

For diffraction, we do not attempt rigorous physical realism. Instead, we simply determine if the face we have intersected contains an edge which is considered capable of diffraction. Such edges have been marked previously, when we load the scene file. An edge is considered significant for diffraction according to the same criteria as used in iSound [5]: all boundary edges are marked for diffraction, as well as edges where the dihedral angle is at most $7/8\pi$. However some of our meshes contain faces where both sides of the face must be consid-

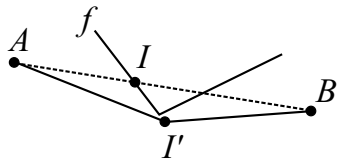


Figure 2: Simplistic diffraction simulation.

ered, so we end up marking all edges where the dihedral angle is greater than $9/8\pi$ as well.

When we find a face obstructed the line-of-sight path to the source that contains a diffracting edge, we project the point of intersection to the edge of the face, plus a small distance ϵ further, and again check the line-of-sight path to the source. If we now have an unobstructed path from the receiver to this new point and an unobstructed path from the new point to the source, then we add this path to the list. For example, see Figure 2 In this example, a ray traced from point A to point B encounters an obstructing face f. The point of intersection I is projected towards the sharp edges of f, one of which yields the new point I', which has clear line-of-sight from A to B.

The goal of this approach is to yield the kind of smooth fall-off that we naturally expect when moving into the shadow region of some object between us and a sound source. A sharp boundary is physically unrealistic. Instead, we smoothly attenuate the contribution from this path according to the angle of the bend required to clear the obstacle.

We can also handle multiple orders of diffraction at this point in the path. If the first projection of an intersection point still does not yield a clear line-of-sight, we can check the new obstruction for diffracting edges, project again, and check for visibility from this new point. This procedure can be repeated to a certain depth, but in the worst case will involve exploration of 3^d faces if every edge of every face encountered is a diffracting edge. For our simulation we found that a diffraction order of 2 was sufficient for the `roomdoor` scene, where the thickness of the door opening required two bends to clear, and did not greatly increase computation time.

2.2 Limitations

We are limited to scenes containing only piecewise-planar surfaces, i.e. meshes. The image sources method only makes sense when computing reflections over planar faces. In practice this is not a huge limitation, as many of the scenes which we would like to model can be represented well with a collection of meshes. In most video games, for instance, all of the geometry is composed of meshes.

This method may fail to find some paths of specular reflection. This can happen if the sampling to determine visibility sets manages to miss some faces which were actually visible. The trade-off here is in how many samples we take: too few, and we may miss some faces; too many, and we may waste time finding

the same faces over and over again. However, this also gives us a trade-off between computation time and accuracy. If quick results are desired, we could sample only a few faces, and most likely, we will end up with a set containing the larger visible faces (as there is more chance that a uniformly-sampled ray will end up intersecting a face which subtends a larger solid angle.) If these faces are those that would yield the majority of the impulse response, then this may be a reasonable trade-off to make.

There are also some important limitations inherent in the ray-tracing model we are using here. This model, in general, cannot perfectly capture the acoustics of a particular scene. Sound behaves quite differently than light, and ray tracing may not be the most appropriate tool to model it. In particular, our model is unable to simulate such phenomena as standing waves, sympathetic vibrations, or other resonance effects. In addition, we model all sound sources as omnidirectional point sources. Handling sound source such as the string of an acoustic guitar, which is naturally amplified by the geometry of the guitar body, is far beyond the scope of this project.

3 Audio Processing

The end result of the ray-tracing is to produce a list of paths for the sound waves to take; each path contains a list of nodes, represented by a C++ `struct` which stores the point in space of the node, the type of path node to which this point corresponds (specular, transmissive, or diffractive), and the acoustic material of the surface at this point (for specular and transmissive nodes.)

3.1 Fast Convolution

Once we have an impulse response describing a particular scene's acoustic properties, we can convolve that impulse response with an arbitrary sound signal to make it sound as if that signal were emitted from the sound source in the scene and the listener was standing at the chosen position within the scene. We borrowed and modified code that Perry Cook wrote to do fast convolution using the Fast Fourier Transform (FFT) [3]. His C library also has routines for reading and writing WAVE files. In our application, all sounds are sampled at 44100hz and are represented using 16-bits per sample (a C++ short).

3.2 Bandpass Filterbank

As mentioned earlier, the reflection and transmission coefficients have a dimension for each of 20 frequency bands equally distributed from 0hz to 20khz. In our model, we send along a 20-dimensional ray packet that gets multiplied by the 20-dimensional reflection and transmission coefficients at each bounce along a path. Another way to say this is that we trace 20 impulse responses simultaneously, so that by the end each band has its own impulse response.

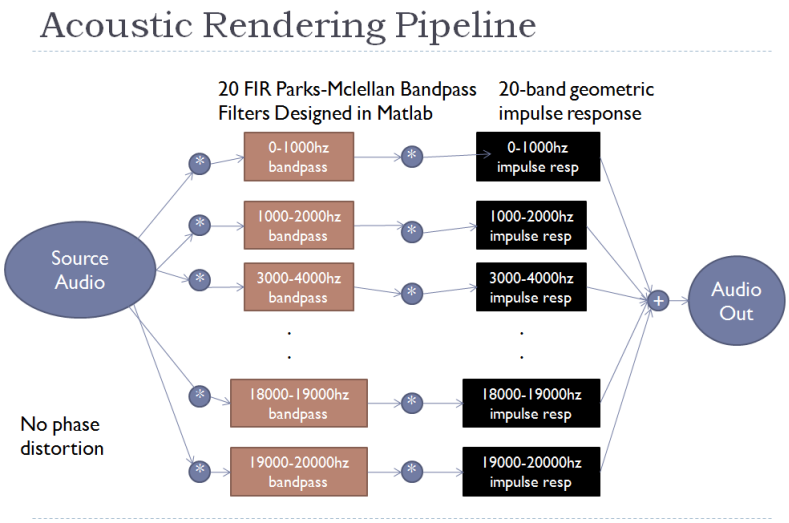


Figure 3: Diagram of the audio processing.

At the end of this path tracing, we need to separate the sound signal into 20 frequency band and convolve each frequency band with its corresponding impulse response. This can be done with a 20-dimensional filterbank as shown in Figure 3.

A set of 20 feed forward, FIR Parks-McClellan filters are used with a transition frequency between bands of 50 Hz and an allowed ripple of 0.1. These filters were designed in Matlab anud saved to a text file, which could then be imported into our program and convolved with the impulse responses using the fast convolution code just described. The order of each filter is 1895 samples, which is negligible at a sample rate of 44100, so these bandpass filters do not add much computational overhead.

Once each impulse response has been convolved with its corresponding bandpass filter, the result can be convolved with the input sound signal and summed with the other bandlimited impulse responses to produce the final output. The final output is saved as a stereo WAVE file using Perry Cook's wave library, and the result is played back in the user interface using the SDL Library Mixer [1]. Here's an equation that summarizes the process of convolving each impulse response I with its corresponding bandpass filter B to create the final impulse response, which is then convolved with the input sound X:

$$Y = X * \left(\sum_{i=1}^{20} H_i * B_i \right) \tag{1}$$

3.3 Binaural Approximation

Humans have evolved to percieve the 3D location of a sound source by comparing the differences in sound signals at both ears. Normally to mode this phenomenon in a simulation, it would be necessary to implement a Head-Related Transfer Function (HRTF) to model the interaction of the signal at both ears as it passes through the head and other parts of the body [6]. We did not have time to implement an HRTF, but we did the next best thing we could think of; we calculated the impulse response at two virtual ears. The head was assumed to have a radius of about 8.8 cm, and two impulse responses were calculated: one 8.8 cm to the left of the viewpoint and 10 8.8cm to the right of the viewpoint. Each impulse response was convolved with the input sound, and the result was saved as stereo audio in a WAV file, with the first channel being sound heard in the left ear and the second channel being soudn heard in the right ear. This worked surprisingly well, and will be revisited in the results section.

3.4 Materials

An AcousticMaterial class was set up to represent the transmission and reflection coefficients that surfaces in a scene can have. We adapted three different models for each of these. The first was a coefficient that decreased exponentially as the band frequency increased. The second decreased as 1/freq as the band frequency increased. The third was a pure lowpass filter with only the 0-1000hz band nonzero. These all have different applications testing results, and will be explained further in the results section.

4 User Interface

We have developed an interface for visualization of scene geometry and of the path tracing results. This framework specifies a hierarchical description of a scene that was adapted from the COS 426 ray tracing assignment and ported over to the new GAPS library. It has code that allows nested transformations and bounding box checks during ray tracing that can greatly speed up ray casting for scenes with multiple meshes. Note that all meshes are transformed back into world coordinates before ray tracing for simplicity.

5 Results

Screenshots from the UI and samples of the resulting audio are available online at `http://www.princeton.edu/~ctralie/Projects/ImSAcoustics/Results/`.

Here are some of the more interesting results. For one, we can show the difference made by our approximation to diffraction compared with no diffracting paths at all. The scene `roomdoor` consists of two rooms joined by an open door between them. The listener is positioned so that there is no line-of-sight to the sound source, so the only rays we find are higher-order specular reflections,

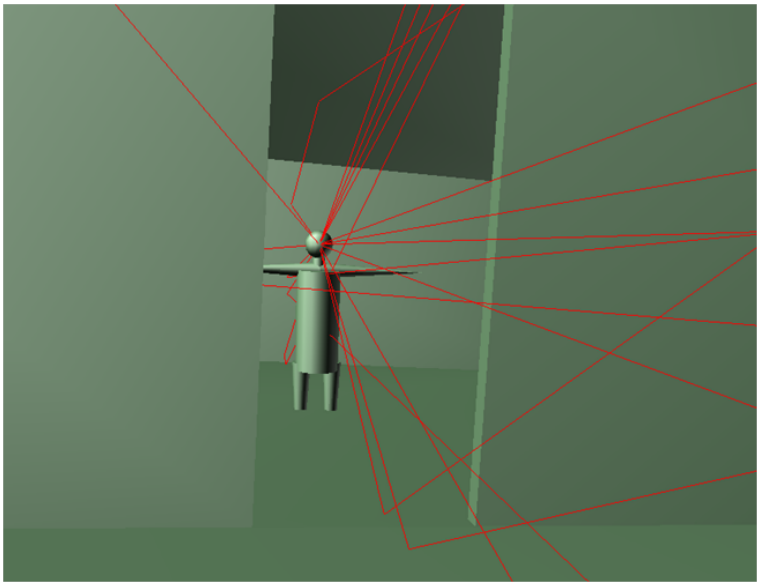
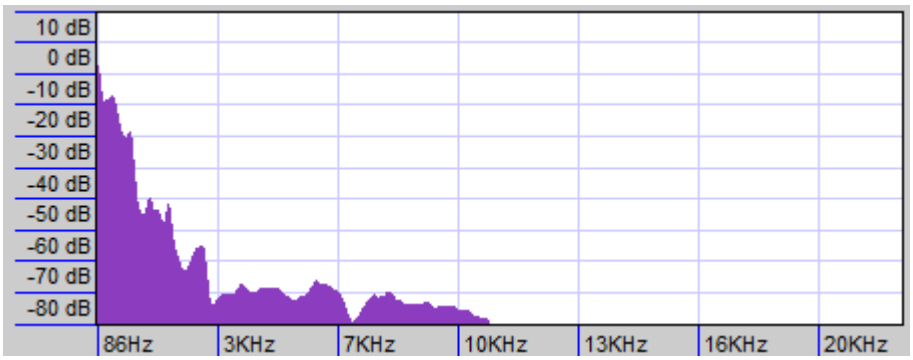


Figure 4: Example screenshot from the UI, showing the paths found to the listener.

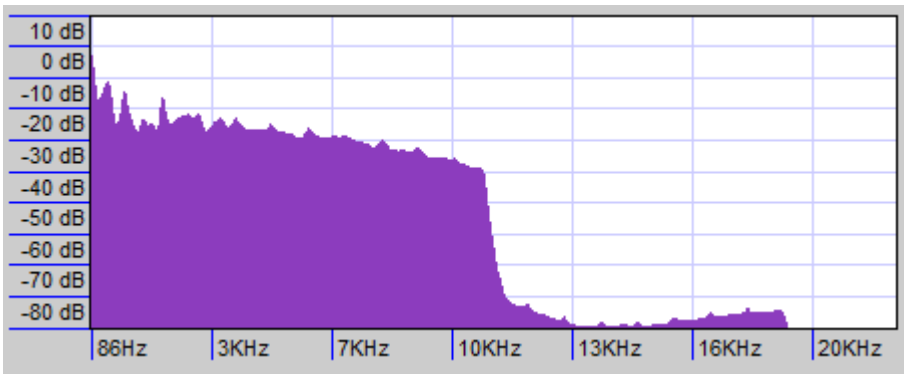
which somehow manage to find their way through the door. When we enable diffraction, we suddenly add an almost direct path from source to listener, one which is undamped by the many reflections that the specular rays have undergone. Figure 5 shows the difference in the resulting filter response, in the frequency domain.

We can also illustrate the effect of sound transmission through a surface. In the scene `carstoplight`, we have a car stopped at a stoplight, blasting loud music. Within the car, the frequency repsonse we compute is fairly flat, while outside the car, all we get are the low bass frequencies. Figure 6 shows the comparison between filter responses computed inside and outside the car.

Lastly, we also tested inside an ellipsoidal "whisper chamber." Of course, this scene does not contain a perfect ellipsoid, but a discretized mesh. We tested with two different locations for the source and listener: in the first, each is located at a focus of the ellipsoid, and in the second, both are moved slightly away from the foci. For the former case, in a perfect ellipsoid all the paths from one focus to the other would be of the same length, so we would expect the impulse response to contain a single component. Essentially, the whisper chamber naturally amplifies the source sound by focusing all of the waves to one location. The impulse responses for these two placements are shown in Figure 7.

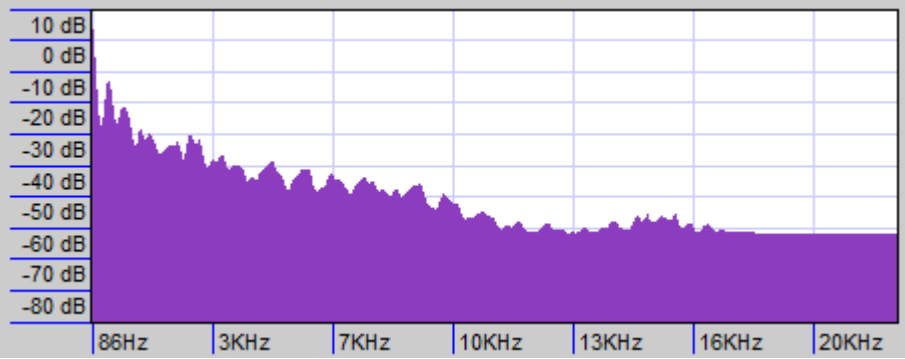


(a) Filter response without diffraction.

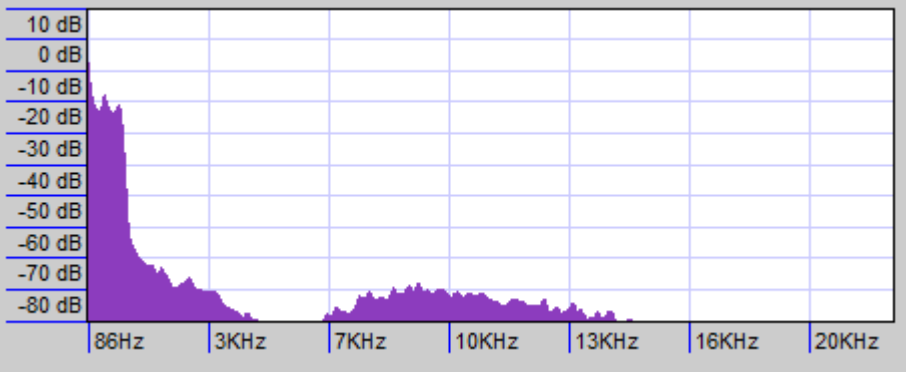


(b) Filter response with diffraction.

Figure 5: Change in filter response when diffraction is enabled for the `roomdoor` scene.

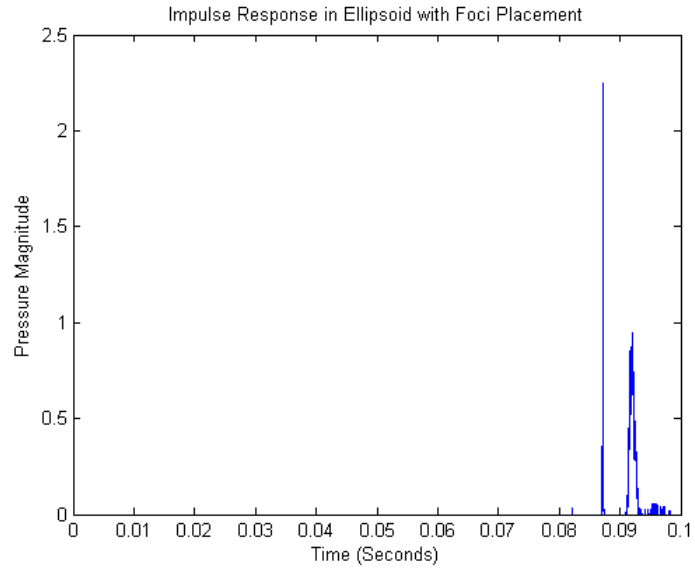


(a) Filter response for a listener inside the car.

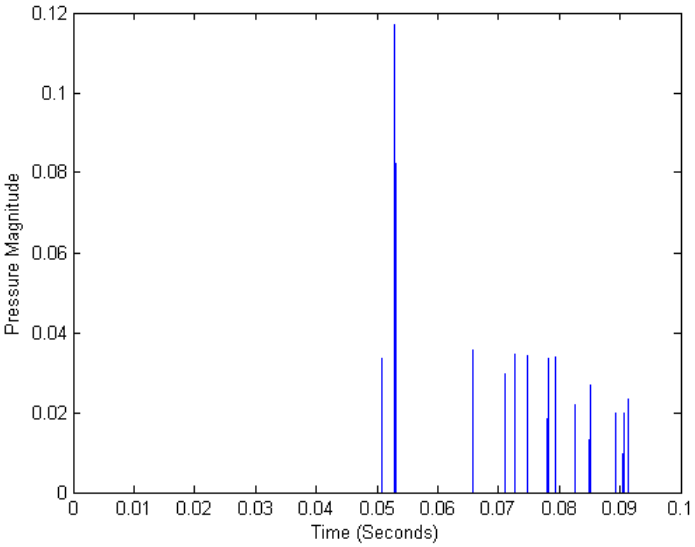


(b) Filter response for a listener outside the car.

Figure 6: Changes between a listener inside the car mesh and outside the car mesh for the `carstoplight` scene.



(a) Impulse response when both source and listener are placed on the foci.



(b) Impulse response when source and listener are offset from the foci.

Figure 7: Differences in the placement of the source and listener for the `ellipsoid` scene.

5.1 Conclusions

Overall, our method worked surprisingly well at modeling many different acoustical phenomena. The speed of sound was clearly evident in the city scene with large distances between buildings. Diffraction was demonstrated in the two rooms separated by a door, and frequency-dependent transmission was demonstrated by blasting hip hop music through a car sitting at a stop light. We were even able to get a primitive binaural response implemented that sounds relatively convincing. Also, although most of our testing scenarios involved qualitative observations, there was one qualitative experiment we were able to carry out with the ellipsoidal chamber to ensure that specular reflection was working properly.

6 Further Work

We believe that our code is designed in a fairly modular way, which serves to facilitate ease of experimentation with new features.

6.1 Hardware acceleration

A team at UNC took advantage of the highly parallel architecture of an NVidia video card and implemented a real-time audio auralization technique by casting many rays in parallel on the NVidia GPU [5]. We used many of the concepts in that paper to help design our system but we never got to doing it in real-time. We were originally planning to design a real-time system but we never got past the software phase. But because our code is extremely modular, the ray tracing is completely separate from the rest of our code, so we believe that it may be possible (and quite easy) to create a new ray tracing model that relies on real-time ray tracing in hardware. This is similar to what was done in the iSound paper, except using the recently released NVidia OptiX real-time ray tracing engine built on top of CUDA.

Even if real-time ray tracing were implemented there is also the challenge of interpolation for changing viewpoints as the user moves through the scene in real time. Each impulse response has been calculated for a fixed source and fixed listener, so somehow new impulse responses will have to be merged together as the source and/or listener move. Techniques described in iSound address this but more extensions would probably have to be made.

References

[1] Sdl mixer.

[2] S. Strom A. Krokstad and S. Sorsdal. Calculating the acoustical room response by the use of a ray tracing technique. 1968.

[3] Perry Cook. Cos 325 FFT and fast convolution source code.

[4] Thomas Funkhouser, Nicolas Tsingos, Ingrid Carlbom, Gary Elko, Mohan Sondhi, and Jim West. Modeling sound reflection and diffraction in architectural environments with beam tracing. 2002.

[5] M. Taylor, A. Chandak, Q. Mo, C. Lauterbach, C. Schissler, and D. Manocha. iSound: Interactive GPU-based sound auralization in dynamic scenes. 2010.

[6] Nicolas Tsingos Thomas Funkhouser, Jean-Marc Jot. Sounds good to me! computational sound for graphics, virtual reality, and interactive systems.