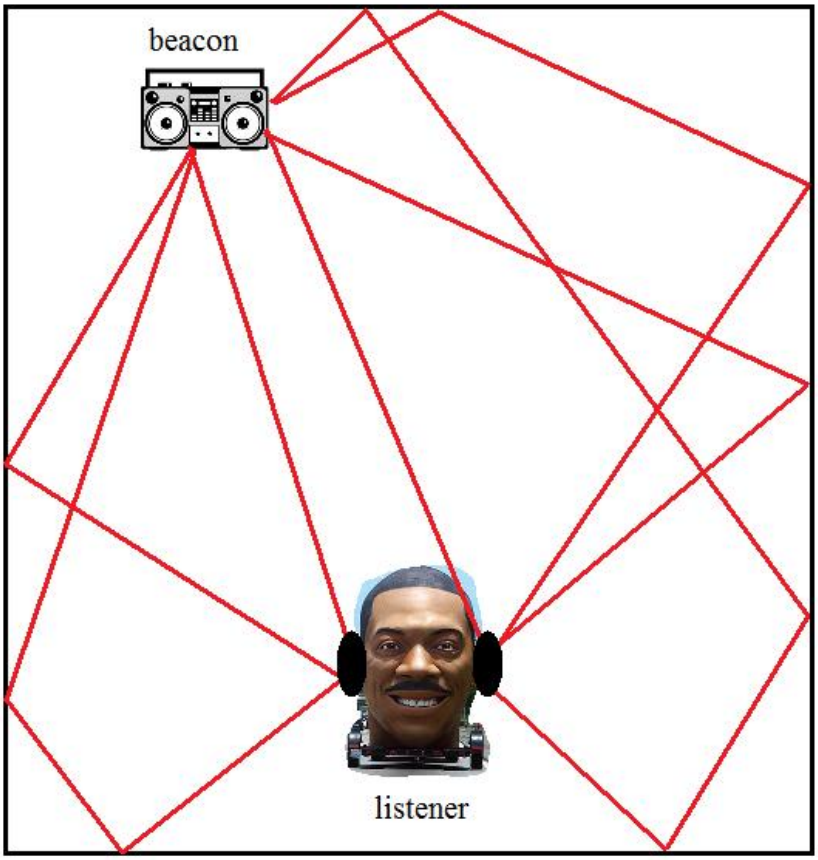
Binaural Acoustic Ray Tracer

COS 325 Spring 2009

Chris Tralie

Development Platform: Windows Vista, VS Studio 2005

Intro (What I did, why I did it):



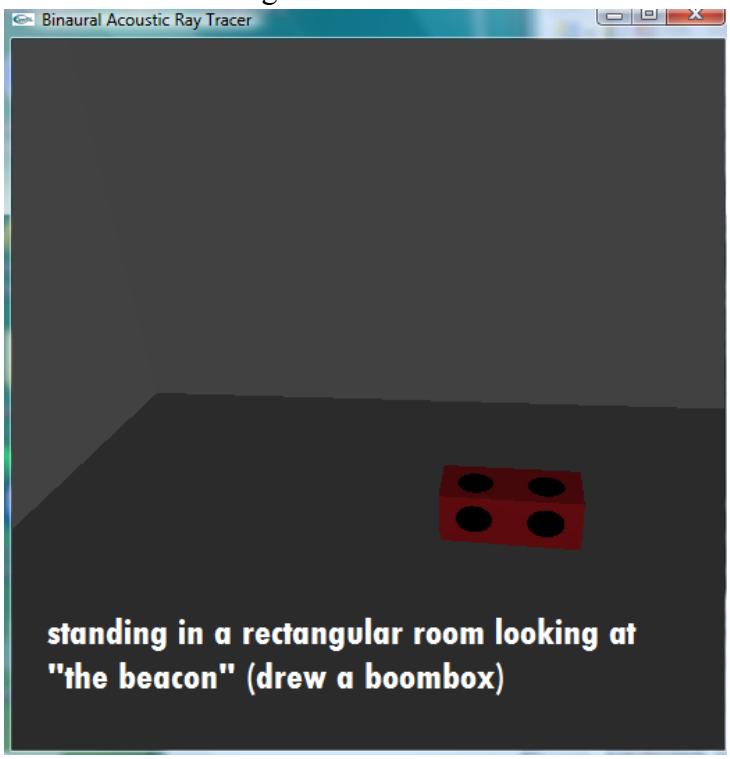
The purpose of this project was to use a well-known technique in computer graphics known as **ray-tracing** to attempt to extract acoustical information from different rooms modeled in the computer. Specifically, I set out to empirically calculate the impulse response of a room made up of different 3D shapes in space by bouncing rays around randomly and seeing what they hit. I placed a "beacon" in one part of the room, which is the source of the sound (I drew it to look like a boombox in the UI), and I placed a listener's head in another part of the room, which is where the paths of the sound waves were measured. I could calculate the impulse response at

each ear this way by measuring the decay and time of each path that I found from the beacon to that ear. In doing both ears (space 9 cms on either side of the head, I got by measuring my own head), I make it more realistic because it sounds like it's in 3D. I used a variant of "reverse ray-tracing" to accomplish this, which means that I started at the ears and randomly traced rays out in different directions and saw which ones ended up at the beacon.

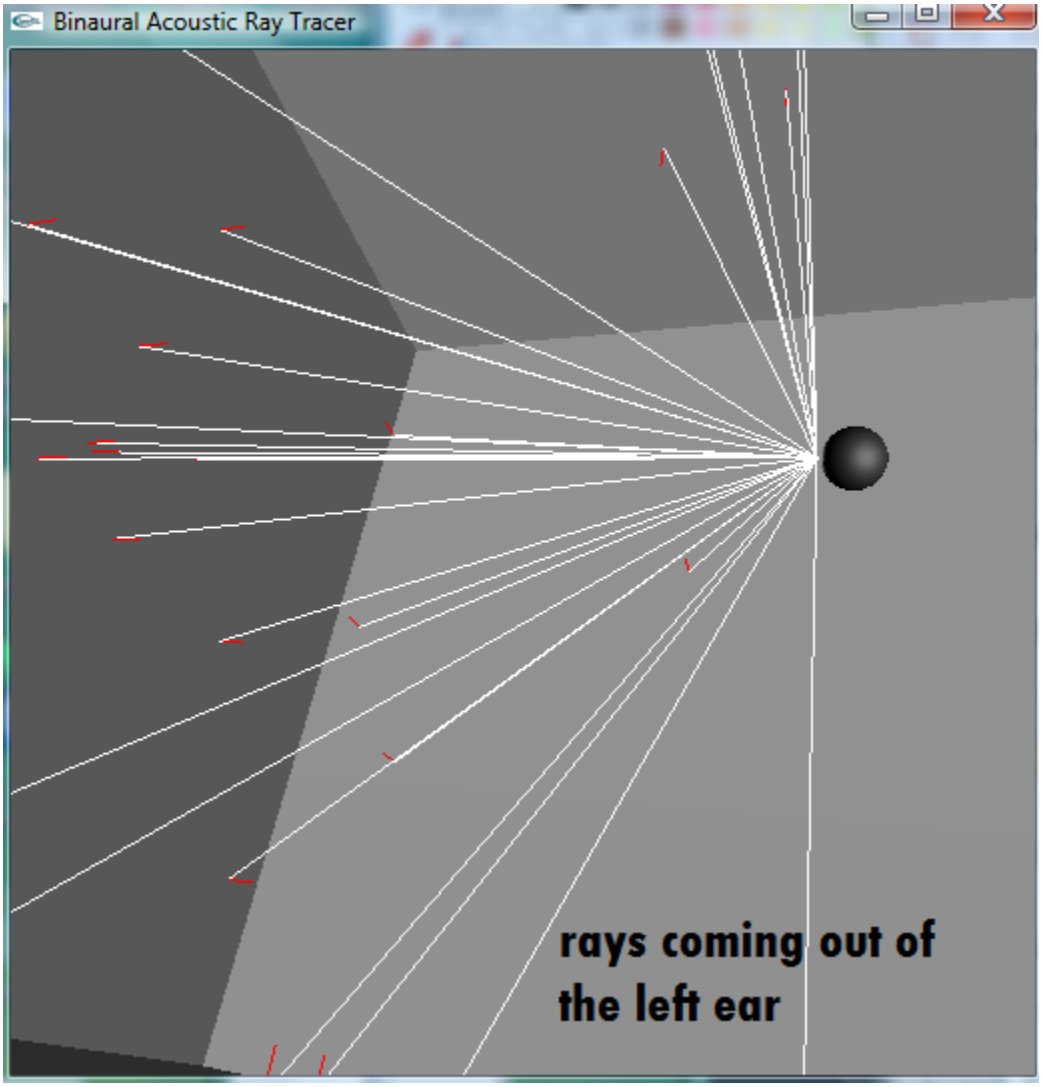
One of the reasons I wanted to do this project is because of my Assignment 3 on convolution, where I went to Blair Arch and measured the impulse response by clapping. The results were so cool to me (and the concept of modeling rooms with just an impulse response seems very powerful), and I wanted to see if I could do something similar "out of thin air" by just having some model in the computer of what a room might look like, and maybe having the capability to tweak the model and hear right away what it might sound like. This could be useful for acoustical architecture to test the feasibility of certain acoustics before actually building them

Methodology/ Implementation

I started off with some code from my computer graphics assignment #3 this semester on ray tracing, which allowed me to specify hierarchical 3D scenes and to render them to the screen. The first thing I did on top of this was to create a way to model the position and orientation of the beacon and head in the scene. To do this, I read in a position and 3 orthonormal basis vectors (right, up, and towards) for the beacon and the head. This way, I could have them in any position or any rotational orientation. The code for doing this is found in **SceneObject.cpp** and **SceneObject.h**. After this, I wanted to create an intuitive way for the user to update the position of the beacon and the head dynamically without quitting the program and editing the scene file. Bascially, I ended up just setting up the standard WSAD control from video games; that is, WSAD are move object forward, back, left, and right, respectively. I mapped up and down for looking up and down and left and right. Here is a screenshot of the user interface:

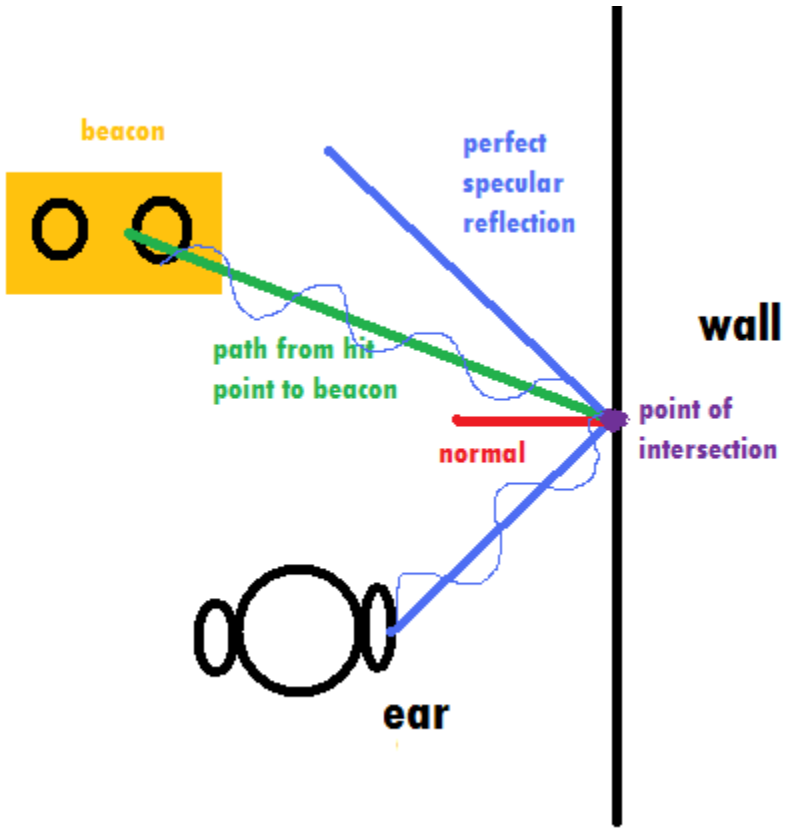


NOTE that all of the code for manipulating the user interface is found in the main file, **RayAcoustic.cpp**. Another feature I added to the user interface for debugging was to allow the user to view the head from the beacon's point of view, and to shoot some test rays out of the head's right ear and see where they hit (to make sure my ray intersection code from the graphics assignment was working properly in this context)



Once I had this working and was confident that I could send rays out, I was ready to start writing the code to do the actual ray tracing. This step was completely different from the ray tracing that's used in computer graphics, because sound is so much slower and decays more rapidly (unlike the directional lighting, which is usually assumed to come from far away and does, therefore, not decay much more over short distances). Like I mentioned before, I decided to do

reverse ray-tracing, starting at the ear and shooting sound rays out and bouncing them around until they hit the beacon. This is so that only the relevant paths of sound propagation from the beacon (the ones that reach the ear) are modeled, minimizing waste. I generated 10,000 rays in random directions in the upper hemisphere next to the right ear. For each ray, I determine the position of the first object of intersection in the scene, along with the normal of the surface at that intersection point.



Through all of these transactions, the sound wave is losing energy. It first loses some of its intensity due to the inverse square law (the sound energy is spreading across a sphere of increasing radius). To model this, I calculated the distance from the ear to the point of intersection and scaled the intensity by $1 / (1 + d^2)$. I put the 1 in the denominator to prevent blowup, and also because of the intuition that if we're right next to something than the sound hasn't decayed at all.

The sound wave also loses some of its energy when it collides with the wall (the wall absorbs some of the energy), so I have a coefficient for each different material I use that determines what this decay is (a decay coefficient of 1 will mean a very reverberant room since the walls do not absorb any energy, while a decay coefficient of 0 will mean a very absorbent room since the walls absorb all of the sound).

Another loss of energy is just part of my model; at each point of intersection, I compute a ray from the point of intersection from the beacon. This isn't always the strongest reflection direction (the strongest reflection direction has equal incoming and outgoing angles with the wall). So I scale it down by the dot product between it and the direction of perfect reflection. After this, scale it down again by the $1 / (1 + d^2)$, where d^2 this time is the squared distance from the hit point to the beacon.

One detail I left out was that I trace one ray directly from the ear to the beacon each time, just to account for the one ray that goes in that exact direction (so that my random generation of rays doesn't miss it). Also note that I can do secondary rays (more than one reflection) by tracing more random rays out of a semicircle around each hit point (I do this, but it doesn't seem to make much of a difference since the decay is so high).

Most of the code for these operations is found in **raytrace.cpp** and **impulse.h** (a file where I have a map of sorted times of ray hits and the associated intensity decay for each ray path)

After the above step is completed, I have a hash map that has a bunch of delay times and the associated intensity that the source would have going through that path. I now have to resample it, send it out to a WAV file, and convolve it with the sound that I wanted to seem like was coming from the "beacon." With the help of Perry Cook's fast convolution file and waveio.h, this task isn't too difficult. The code for doing this can be found in **impulse.cpp**. I do this twice; once for each ear, and store the results in separate files. They can then be mixed together into stereo audio with a program like audacity.

USAGE: From the command prompt, type RayAcoustic box.scn (or any other .scn file)

Hit the space bar when ready. The program will look for a file "test.wav," which is the file that will be "played through the beacon" (convolved with the impulse response of the room). The program will calculate the impulse response at each ear and convolve it with test.wav, and save the result to convL.wav and convR.wav for the left and right ear, respectively

Challenges along the way

There were several challenges along the way that "threw me for a loop," so to speak. First of all, when I initially sent out the sound rays and recorded how much their intensity decayed based on reflection loss, distance, etc., I assumed that this was the exact numerical value that I could copy out to the wav file for that sample of the impulse response. For instance, if the

intensity went down from 1 to 0.15 through some path between the beacon before it reached the ear, I thought I could just copy 0.15*max_amplitude at that time to the output impulse response. I was getting some strange results with this method, though. Most notably, the reverberation died off way to quickly, such that the only pulse that really survived was the pulse directly from the ear to the beacon, if that path even existed. I then realized I should probably be talking about loudness, not intensity. So I then decided to take the log of each sampled intensity over the minimum intensity encountered. But this was giving me some unwanted results as well; all of the reverberations were way too loud, and it didn't seem to make a difference if I damped the reflection coefficients. Finally, I decided just to go back to the exact definition of loudness that's given in every physics textbook; $L = 20 \log(I / I_0)$, where $I_0 = 10^{-12}$ Watts / m^2. I then normalized the loudness values on a scale from 0 to 1 such that 1 was a loudness of 100. This worked A LOT better. Now the effect of reflection coefficients is plainly heard, and the reverberations are reasonably soft.

Another problem I had was sampling the empirically-calculated impulse response. The problem is that there's no guarantee that the spacing between the times that different wave reflections arrive at the ear will be the same. But I eventually needed to convert the impulse response into a uniformly sampled 44100khz sample rate WAV file. I ended up using nearest neighbor interpolation for this, which I know is noisy. This gives a pretty good approximation (it's possible to at least tell generally when the echoes occur), but I'm sure there are better methods for converting a non-uniformly sampled signal into one that's uniformly sampled.

Conclusion / Results

Even though I made a lot of approximations, this program is still able to demonstrate the basic concepts behind reverberation and binaural position sensing. Look in the directory **Results** for some of the testing scenarios I created. The first one is me walking towards a boom box in an extremely reverberant hallway (wall absorption coefficient is 1), then turning my head left, then turning my head right. If you listen with your headphones, it sounds like it's getting louder as you get closer, and then it sounds like the beacon is to the right of you (when you're turned left) and to the left of you (when you're turned right). For fun, I also created a scenario where I turned the reverberation of the hallway way down, and a scene where the user is standing in the upper half of an ellipsoid (like a whisper chamber). See my musical statement for more results.

References

- http://gamma.cs.unc.edu/SOUNDC/Assignments/Assignment_1.html

  (Link to an assignment I found on a graduate course web page an UNC that looks very much like what I was trying to do)

- http://www.cs.princeton.edu/courses/archive/spring09/cos426/assn3/

  (link to the ray-tracing assignment from this spring in COS 426: Computer graphics, where I borrowed a lot of the code to read in scene files and render images to the screen)

- http://www.cs.princeton.edu/courses/archive/spring09/cos325/src/Filters09/convfht.c

  A file that does fast convolution and allows for easy writing to WAV files (borrowed from Perry Cook)