CPE 458 APC: 3D Room Acoustics

Bryan Ching, Matt Crussell

I. Introduction

THIS project is an extension of the Real-time Ray Tracing lab. Our goal was to produce virtual 3d sound based on the dimensions of a room created by ray tracing. To produce acoustics, additional ray tracing was performed. Sound waves were represented by rays were set to bounce off objects and degrade over time.

II. Representing Sound

Sound is generally represented as either waves or rays, each introducing their own set of pros and cons in their usage. Sound shares many of the same properties of plane waves, including frequency, wavelength, direction, amplitude, etc. Representing waves as sounds allows one to display most if not all of these properties. To accomplish this task, one would have to release waves from the sound source and release additional waves on collisions. The other most common way to represent sound is through rays. Rays are not able to emulate some important properties of sound, such as interference and diffraction. Ray tracing can also be more computationally heavy when the degree of reflection increases. An extension to ray tracing is a method called beam tracing. Beam tracing traces a beam from the source to the room. Collisions spawn more beams recursively. The similarity to ray tracing can clearly be seen.

We have decided to use ray tracing because of our familiarity with our previously built ray tracer. As mentioned before, ray tracing is not able to emulate some properties of sound, such as diffraction and interference. Moreover, ray tracing by nature provides less resolution and is dependent on the number of rays used. On the other hand, acoutsic ray tracing is a lot simpler to implement after an initial functional ray tracer has been made. Because we do have that initial framework of a ray tracer, we decided to use rays to represent the sound. We understood the consequences in terms of accuracy and computation, but we believed that with enough released rays, we could still be able to produce an accurate representation of sound in a room.

III. Implementation

A. Audio Output

Initially we had looked into using the SDL library for stereo audio output. SDL Mixer provided a simple distance, angle, sound argument input and allowed multiple consecutive audio playing. Upon implementing 3d audio and testing the SDL library, we observed artificating and generally an unclear sound. We had originally selected the SDL audio library for its cross platform compatibility.

The next platform we looked at was the OpenAL library. This library has been prominent in several games including Bioshock, Battlefield 2, and Doom 3. OpenAL also provided wide cross-platform compatibility, though due to a lack of documentation we decided against implementing it.

We finally decided on implementing audio using the FMOD ray. FMOD provides automatic logarithmic scaling of sound with respect to distances. FMOD is capable of mapping a 3d space and providing audio alteration for both a stable speaker, or a speaker with a changing velocity. On testing the audio in a 3d environment, we did not hear any artifacting or defunct sounds. kd

B. Video Output

We had originally planned on moving to the NVIDIA Optix 3.0 Ray Tracing engine for performing efficient tracing. Upon testing the engine with lab machines, we noticed driver issues. CSL lab machines as well as room 255 machines were unable to run Optix. Due to time restrictions, we made the decision continue using our simple OpenGL ray tracer. For a varied ray-tracing device, future Applied Parallel Computing students may consider installing the Optix library.

Video output is performed using the glm library. In the CPE458/CP570 tandem ray tracing project we create a real-time parallel tracer. Code was provided by the instructor for simple OpenGL output. Pixel data is written to a specific memory slot in device memory in order to display results.

C. Audio Kernel

On the highest level, this kernel generates two vectors: 1) A vector of distances to speaker from the listener, and 2) A matching vector of directions for each distance. This kernel will shoot of ranges of rays for a single ear. Thus the kernel must be run twice.

The audio kernel computation can be broken into several steps. First we must determine the audio ray direction. This is done by computing and offset value in which sound must be traced in two axis. When computing sound, camera orientation is a large component for direction. Secondly we determine if our sound rays have hit a speaker. To emulate reflections of sounds, we must compute the reflection of audio against the normal of the object hit. With each reflection, sound attenuation is accounted for by increasing distance from speaker. After N reflections the function will return a distance upon hitting speakers, or float max value if not.

The resultant directional vectors and distances are then written to global memory.

D. Reduction

In order to reduce our vectors into a smaller subset for sound output, we must perform a reduction. For simplicity we have decided for a minimum reduction. With this technique, we can successfully reduce sound into single vector/point so we can determine where the object would be located. We chose to perform a min reduction because of its ease of complexity, although a more accurate acoustic modelling may take into account compounding sound to emulate echoing.

Our min reduction must be run twice, once to reduce all memory in each block, and another to reduce the final block. Audio was output on two channels, one for each left and right headphone speaker. Audio is now output in only one channel, and the audio library can determine what should be played in each speaker.

IV. Results

Initially our method of determining acoustics focused on shooting rays directly from each ear. Performing multiple-reflective tracing can be very costly and we observed a frame rate of 28 frames/second when shooting off on the order of hundred of thousands of Rays. Reguardless of such a large quantity of sound rays being fired, we still noticed acoustic blind-spots in the design.

By altering the method of determining directional vectors, we were able to increase the frame rate to 81 frames/second on CSL lab machines. This was done using unit circle properties, and instead of focusing simply on areas for a left and right ear, rays were computed in every direction. The large bump in frame rate was due to dramatically decreasing the number of rays fired. Our initial algorithm clustered rays towards the perpedicular to each ear, while unproportionally considering rays forward and backwards. With our new method, we were able to mostly eliminate the blind-spots that we observed earlier. We were also able to decrease the number of rays shot and cover a greater range. This just goes to show how bad our initial implementation was.

One other tweak we made since our first implementation was to reduce our vectors into one sound source instead of one for each ear. This gave us a more realistic acoustic experience. When behind an object, we could hear a muffled sound. When to the side of a sound source, we could hear the sound coming from that side. Everything worked very intuitively and sounded as you expected it should.

V. Conclusions

Acoustic Ray Tracing was a suitable application as an extension to Video Ray tracing, but it's limited nature would prevent commercial use of the technique. For more accurate representations of acoustic processes, beam tracing or wave tracing must be considered.

Although a high ray count might improve accuracy, It is critical to ensure algorithms that determine direction provide an accurate spread of rays. Without this, a person may encounter 'conal' or 'spotty' audio.

VI. Future Work

The most obvious path for future work to go is more sources. Having one audio source is not very realistic and adding additional sources of audio is the next path for us to take if this project were to continue.

CUDA is still a growing platform for parallel computing on the GPU. With the release of CUDA 5.0 comes a dynamic parallelism, a new way to toy with parallelism and the thread model. Dynamic parallelism, simply, allows a CUDA kernel to call another CUDA kernel. This allows for many more opportunities that were unavailable before CUDA 5.0 or would have been overly complex in both code and computation. These opportunities include recursion inside a kernel or GPUs generating more work for a GPU to do. For future work, we could exploit these new features to our code base, especially in the way we handle reflections. We could also use the new recursion to explore wave based sound representations. A simple way to imagine wave based sound is with growing spheres that change upon collisions. Collisions with walls will spawn more growing spheres and collisions with other spheres could have different effects such as interference, both constructive and destructive.

There are also certain sound properties supported by ray tracing which we have yet to implement. If given more time, we could implement sound refraction through a medium, sound attenuation, and sound energy loss on collisions.

Another field of computing that we have used but have not touched is audio processing. Audio processing is just computation on audio data and can also be parallelized. We did not touch the audio side of geometric acoustics as much as we did the geometric side; however, if we introduced more properties of sound to our simulation, we would have to manipulate the sound to match the effects. It is at this time when the parellelism of sound processing would be needed.

If given more time, a better reduction could be implemented for our acoustic ray tracer. Firstly, we did use a vector reduction that was optimized for GPU, but we did not particularly use one that was optimized for acoustic ray tracing. We could explore the effects of using other reductions, such as bucketed min reduction or just min reduction but to more vectors. With this, we could look at how more sound sources behave in our environment.

VII. Related Work

Room acoustics is definitely not a new subject to field of computing. Sound is generally modeled as either waves or the more approximated rays. Modeling acoustics by rays is referred to as geometric acoustics or geometrical acoustics. Research has been conducted on the differences as well as the limitation and benedifts of these different methods [1]. Geometric acoustics, as opposed to wave-based acoustics, are generally less complex but ignore certain properties of sounds, such as diffraction. Geometric acoustics can also become extremely computationally expensive when the number of reflections increase. Siltanen et al. has proposed a method of combining different methods to utilize the benefits and reduce the consequences that each provide.

Geometric acoustics have also already been applied to the field of parallel computing. A very similar project to ours, [2], uses ray acoustics to model sound using the acoustic energy propogation model. It is not clear yet, but our project will likely be a subset of the work done here. Lastly, the use of parallelization in the audio processing has also been proposed, as seen in [3]. They suggest parallelizing FIR filtering instead of the HRTF filters seen in gpu computing today.

References

[1] S. Siltanen, T. Lokki, and L. Savioja, “Rays or waves? understanding the strengths and weaknesses of computational room acoustics modeling techniques,” in *Proc. Int. Symposium on Room Acoustics*.

[2] N. Röber, U. Kaminski, and M. Masuch, “Ray acoustics using computer graphics technology.” Citeseer.

[3] L. Savioja, D. Manocha, and M. Lin, “Use of gpus in room acoustic modeling and auralization.”

[4] C. J. Webb and S. Bilbao, “Virtual room acoustics: A comparison of techniques for computing 3d-fdtd schemes using cuda,” *Watermark*, vol. 1, 2012.

[5] A. Krokstad, S. Strom, and S. Sørsdal, “Calculating the acoustical room response by the use of a ray tracing technique,” *Journal of Sound and Vibration*, vol. 8, no. 1, pp. 118–125, 1968.

[6] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West, “A beam tracing approach to acoustic modeling for interactive virtual environments,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 21–32.