# ROOM IMPULSE RESPONSE ESTIMATION USING SIGNED DISTANCE FUNCTONS

*Patrik Lechner* *

Institute 1
University of Applied Sciences St.Poelten
St.Poelten, Austria
ptrk.lechner@gmail.com

## ABSTRACT

Several algorithms and approaches for Room Impulse Response (RIR) estimation exist. To the best of the authors knowledge, there is no documentation of accuracy, speed or even the feasibility of using signed distance functions (SDFs) in combination with sphere tracing for this task. Here a proof of concept with a focus or real time performance is presented, that lacks many features such as frequency dependent absorption and scattering coefficients, arbitrary source and receiver directives etc. The results are shown and compared to real room impulse responses recorded by [1]. The implementation happens mostly inside a compute shader, an example application is provided in the framework `TouchDesigner`. The application as well as all generated data and `Jupyter Notebooks` can be found on this project's github repository at https://github.com/hrtlacek/rayMarchReverb.

## 1. INTRODUCTION

Sphere tracing [2] is extensively in the so called "demo scene" to render impressive 3D video demos via shaders in real time for decades. As a version of ray tracing that relies on the geometry being defined as so called signed distance functions (SDFs), it does not directly support the import of standard 3D Polygonal geometry. One of the advantages lies in the algorithm's potential improved speed in comparison to fixed-step ray tracing. SDFs describe implicit surfaces, via a function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. A function returns a negative value if the locus of the point is inside the geometry, a positive value if outside and 0 if on the surface. If defined carefully, the distance to the nearest surface is always known as the full geometry of the scene describes an ideally lipschitz continuous distance field in $\mathbb{R}^3$.

Since the distance to the nearest surface is always known, the step size of a ray tracing algorithm can be dynamically adjusted, resulting in fewer iterations along a ray, see Figure 1..

## 1.1. Previous Work

A lot of previous work exists both in the field of Ray/sphere tracing and RIR estimation. As shown in [3] and [1] there are numerous approaches for estimating RIRs. NVIDIA is working in the field of real time ray traced audio simulation NVIDIA VRWorks™ Audio (introduced with the Pascal GPU architecture)

[4] bidirectional ray tracing.

* Thanks to the predecessors for the templates

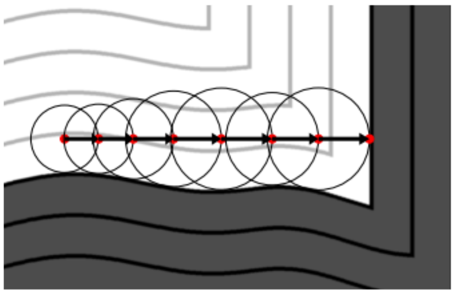


Figure 1: *Visualization of the sphere tracing algorithm in 2D. A ray is sent from the source (left) to the right until it hits a surface, always moving the maximum distance as the SDF informs the tracing algorithm about the distance to nearest surface.*

### RIR Estimation

[3] gives an overview of methods in use for RIR estimation.
image source method, wave based, ray tracing.

### Sphere Tracing

Defining SDFs is an active field of research and there are several projects that aim at easier construction of SDFs and integration in 3D frameworks such as https://github.com/Flafla2/Generic-Raymarch-Unity and [5].

## 1.2. Motivation

The reasons why sphere tracing in a compute shader for RIR estimation has not been documented until now probably lie in the relatively new introduction of compute shaders as well as in the difficulty of creating SDFs(in comparison to using existing 3D /CAD models and import them to polygon based ray tracers).

### *1.2.1. Sphere tracing*

As described above, ray tracing in general is in use. Sphere tracing has a number of advantages over ray tracing polygonal surfaces. It is "procedural" by default, since all geometry is defined by implicit surface equations. More over sphere tracing approximates cone tracing for reducing aliasing artifacts in the pixel domain[2], which in the audio domain, is considered to have advantages but is very time confusing in a non-SDF setup[3]. The deformation and rounding of geometry is possible in a very efficient way, which might offer an opportunity to approximate low-frequency response due to diffraction artifacts. Since geometry is not defined via vertizes and edges, there is no such thing as increasing the complexity of a shape in this way. Rounding a geometrical shape is a mere subtraction since it just shifts the rendering to another iso-surface, which is getting increasingly smooth as shown in 2. Depending on the construction of the geometry, this way, holes and cavities (such as in a diffusor) can also be made disappear for a low-frequency pass as shown in figure 3.

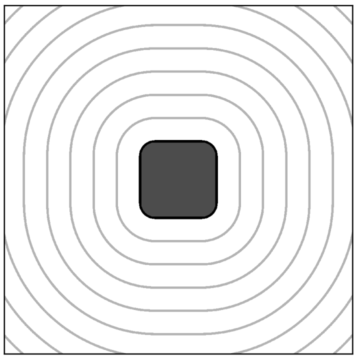
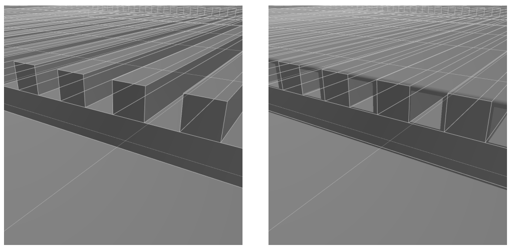


Figure 2: *rounding the box given in Equation* **??** *by subtraction of* 0.7.



(a) *Diffusor shape* (b) *After subtraction: rounded and closed.*

Figure 3: *The diffusor from scene 1 in [1] was reconstructed exactly(a). A subtraction of* 0.01 *from the distance function causes the holes to close(b).*

procedural by default deforming geometry

*1.2.2. Implementation*

It is possible to implement the chosen algorithm on the CPU and the GPU. A number of frameworks could be chosen for GPU accelerated computation such as OpenCL or NVIDIA CUDA. The choice of a shader has the advantage of being more operating system independent and hardware independent. Compute shaders (in contrast to fragment shaders) make it possible to write to arbitrary output locations which is necessary for generating the actual impulse response from the measurement of timings. Since they are available since OpenGL 4.3 (August 2012) / OpenGL ES 3.1 they are both aged enough to have received broad support in other frameworks and relatively new in respect to first publications about sphere tracing. Another reason for the choice of compute shaders is their simplicity. In comparison to CUDA and OpenCL, shaders are easier to write and the GLSL(Graphics Library Shading Language) is more widespread. Achieving the whole computation, from the definition of the geometry, to the ray tracing up until the actual impulse response computation in a single shader makes this attempt highly portable and expandable.

## 2. GENERATION OF SDFS

Only rather simplistic shapes where needed for this proof-of-concept. Mostly boxes are used and combined in various ways to achieve reflection areas, shoe-box scenes and the little more complex diffusor shape of scene 1 in [1]. A simple 3D box SDF with a size of $R_x$x$R_y$x$R_z$ can be described by:

$$f(p_x, p_y, p_z) = \sqrt{c_0(p_x - R_x)^2 + c_0(p_y - R_y)^2 + c_0(p_z - R_z)^2} \tag{1}$$

where $c_0$ is just clipping at 0:

$$c_0(x) = max(x, 0) \tag{2}$$

which translates to GLSL conveniently:

```
float box(vec3 pos, vec3 R){
    return length(max(abs(pos) - R,0));
}
```

Listing 1: *GLSL code for creating a box SDF*

[2] gives a list mathematical definitions of many shapes and for example http://mercury.sexy/hg_sdf/ provides a rich and advanced library of shapes and operators that are ready to use for creation of more complex scenery.

## 3. SPHERE TRACING

For simplicity, deterministic equal-angle Ray Tracing is used in contrast to Monte Carlo or Equal Area Ray tracing (EART) [6]. Unidirectional ray tracing has been used, also for simplicity reasons, although [4] has shown that bidirectional ray tracing offers advantages. Since the classical sphere tracing algorithm was adapted, it was found to be simplest to consider the "camera" to be the receiver/microphone as it would receive light. It sends out rays that might hit the sound source, which acts as a receiver of rays. The sound source is chosen to be a sphere. Choosing a correct volume for the receiver is critical and using a constant size can introduce systematic errors [7], [3]. A number of models are available to compute the receiver Volume, $V_r$. Typically factors such as room volume, number of rays and the distance from source are used for this computation. As in [8], [3], and [9] the receiver was allowed to grow in volume. While [8] and [9] use time to as a factor to let the receiver grow, in this attempt the reflection count, $k$ is used. Initially when a ray is sent, $k = 1$ and when it hits a surface, this counter is increased by one so the source grows by this factor. Instead of using time, the model provided in [3] is used and augmented with the $k$ term:

$$V_r = k\omega d_{SR}\sqrt{\frac{4}{N}} \tag{3}$$

with

$$\omega = log_{10} V_{room} \tag{4}$$

where $d_{SR}$ is the source-receiver distance, $N$ is the number of initial rays and $V_{room}$ is the volume of the room. The actual sphere tracing follows the original formulation in [2]:

$$t_{i+1} = t_i + F(t_i) \tag{5}$$

Here, this is implemented in the following way:

```
for (i=0;i++;i<imax){
  vec3 pos = ro + t*rd;
  Sdf res = map(pos);
  t += res.x;
  if(res.x<epsilon){
    break
  }
}
```

Listing 2: *GLSL pseudo code for sphere tracing*

In the above simplified code listing, `ro` is a `vec3` that defines the ray origin, `rd` the ray direction and `epsilon` can be set to adjust the algorithms precision. The `map()` function in this case not only returns the distance computed via the SDF(`res.x`) but a `struct` that can contain material properties (reflection coefficients etc.). Here it was used to distinguish between a sound source and a regular reflective body. The space is sampled spherically and `rd` is generated from the compute shader's `gl_GlobalInvocationID`. From there, the space is sampled using the following very simplified pseudo code:

```
Sdf res = castRay(ro,rd)
for (int k=0;k<numReflections;k++;){
  if(ComingFromReflectiveBdy){
    rd = relect(rd,normal);
  }
  res = castRay(ro,rd);
  t = res.x;
  pos = ro+rd*t;
  if(res.body == soundSource){
    travelDistance+=t;
    readWrite(travelDistance,k);
  }
}
```

Listing 3: *GLSL pseudo code for sampling the space and writing to the RIR.*

Here, `castRay()` refers to a function that looks similar to the sphere tracing function above and `readWrite()` refers to a function that takes the total distance a ray has travelled from source to receiver including all reflections and the number of reflections. From the distance it computes the location where to write to in the impulse response and the attenuation. It then reads the current value at this location and adds the corresponding value, making use of a compute shader's capability to read and write it's output and write to arbitrary output locations. The necessity to write to arbitrary output locations is the main reason for choosing a compute shader over a fragment shader. But it is possible to define most of the functionality to in one file that is included to a compute shader to calculate the RIR and to a fragment shader for visualization. This is particularly handy during construction of geometry.

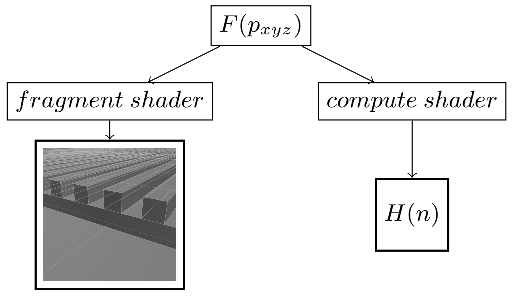


Figure 4: *A lot of the code, especially the map function, $F(p_{xyz})$, which contains the SDF can be shared between a fragment shader used for visualization and a compute shader responsible for RIR calculation.*

1. reflection
2. low frequency pass

## 4. GENERATION OF IMPULSE RESPONSE

The room is assumed to a linear time invariant system(LTI). Due to the proof-of-concept nature of this proposal, a highly simplified model for RIR computation is used. Sphere tracing delivers distances and number of reflections for $M$ rays in this implementation. Each ray follows a number of reflections $K(m)$. Each reflection pass adds up to a total travel distance of the ray, $d(m)$. The sound source is assumed to send out a discrete impulse, the kronecker delta function, $\delta(n)$. As non-integer delays are ignored in this implementation, the shader will just write to the rounded integer delay location $\tau_s$ in the impulse response, $H(n)$,that corresponds to the distance:

$$H(n) = \sum_{m=0}^{M} \delta(n - \tau_s(m)) \cdot \alpha(m) \cdot -1^{K(m)} \tag{6}$$

The total attenuation per ray $\alpha(m)$ can be computed by using a material dependent coefficient of reflection $\alpha_{mat}$ that is stored in the `Sdf struct`. At each reflection pass $k$ this results in a possibly different reflection pass dependent coefficient $\alpha_{mat}(k, m)$ and keeping a running product in the loop of Listing 3:

$$\alpha(m) = \prod_{k=0}^{K(m)} \alpha_{mat}(k, m) \tag{7}$$

For simplicity's sake, here only one global coefficient $\alpha_G$ is implemented resulting in:

$$\alpha(m) = \alpha_G^{K(m)} \tag{8}$$

Additionally, a proof-of-concept for frequency dependent loss for each reflection is introduced into the model. As a computationally efficient heuristic, a simple one-zero filter $G(z)$ is applied at each reflection:

$$G(z, K) = (1 + z^{-1})^K \cdot \frac{1}{2}^K \tag{9}$$

The advantage of using such a simple filter is that a $K$-stage cascade's impulse response can be computed easily without applying

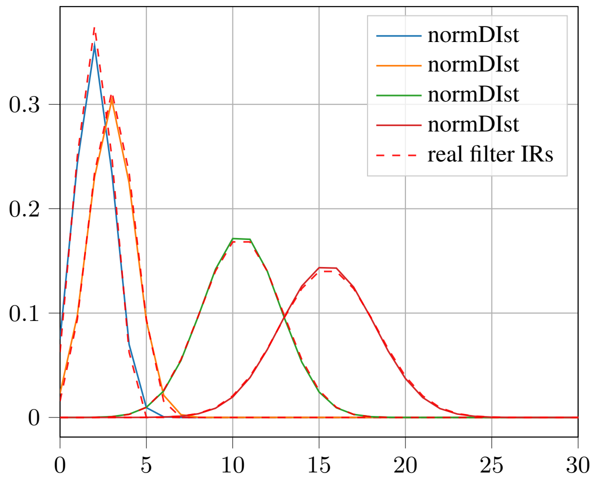


Figure 5: A PGF histogram from `matplotlib`.

the filter repeatedly. By doing the discrete time inverse fourier transform of the transfer function $Gz$ the $N$-length impulse response is obtained:

$$G(n,K) = \mathfrak{F}^{-1}\{G(z)\} = (\frac{1}{2})^K \frac{1}{N}\sum_{\omega=0}^{\pi}(1+e^{-j\omega})^K e^{j\omega n} \quad (10)$$

Numerically, it can easily be shown that

$$G(n,K) \approx \frac{1}{\sigma\cdot\sqrt{2\pi}} \cdot e^{-\frac{1}{2}(\frac{n-\mu}{\sigma})^2} \quad (11)$$

with

$$\sigma \approx \sqrt{K0.231 + 0.562} \quad (12)$$

$$\mu \approx \frac{K}{2} + \frac{1}{2} \quad (13)$$

The right hand side of Equation 11 is simply the normal distribution, which is computationally efficient to calculate for any $K$.

So instead of adding $\delta(n-\tau(m))$ into the IR, simply $G(n-\tau(m),K)$ is used.

normDist

## 5. RESULTS

1. compare to [1]
2. compare to [10]

## 6. CONCLUSIONS

This template can be found on the conference website. For changing the number of author affiliations (1 to 4), uncomment the corresponding regions in the template `tex` file. Please, submit full-length papers (max. 8 pages both oral and poster presentations). Submission is fully electronic and automated through the Conference Web Submission System. DO NOT send us papers directly by e-mail.

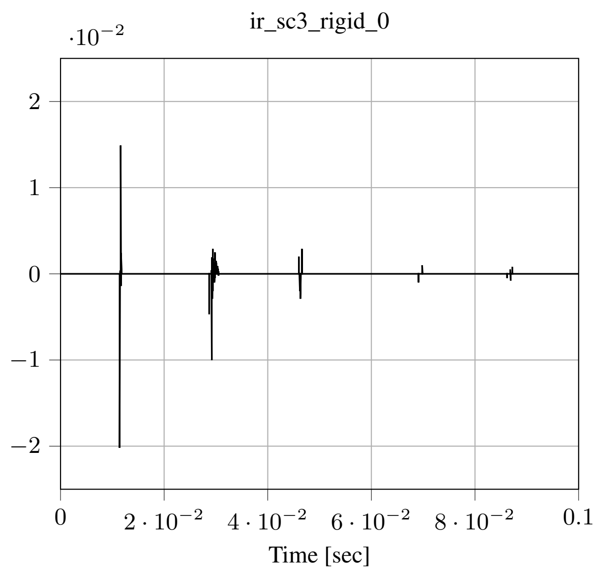


Figure 6: A PGF histogram from `matplotlib`.

## 7. ACKNOWLEDGMENTS

Many thanks to the great number of anonymous reviewers!

## 8. REFERENCES

[1] Fabian Brinkmann, Lukas Aspöck, David Ackermann, Steffen Lepa, Michael Vorländer, and Stefan Weinzierl, "A round robin on room acoustical simulation and auralization," *The Journal of the Acoustical Society of America*, vol. 145, no. 4, pp. 2746–2760, Apr. 2019.

[2] John C. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, Dec. 1996.

[3] Adil Alpkocak and Malik Sis, "Computing Impulse Response of Room Acoustics Using the Ray-Tracing Method in Time Domain," *Archives of Acoustics*, vol. 35, no. 4, pp. 505–519, Dec. 2010.

[4] Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou, "Interactive sound propagation with bidirectional path tracing," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 1–11, Nov. 2016.

[5] Patrik Lechner, "hrtlacek/TDraymarchToolkit v1.1," Feb. 2020.

[6] C. Gu, M. Zhu, H. Lu, and B. Beckers, "Room impulse response simulation based on equal-area ray tracing," in *2014 International Conference on Audio, Language and Image Processing*, July 2014, pp. 832–836.

[7] Zeng Xiangyang, Chen Ke'an, and Sun Jincai, "On the accuracy of the ray-tracing algorithms based on various sound receiver models," *Applied Acoustics*, vol. 64, no. 4, pp. 433–441, Apr. 2003.

[8] Eric Brandão, Rodrigo DAL Fiume, Gonçalo Morgado, William D'A Fonseca, and Paulo Mareze, "A Ray Tracing

Algorithm Developed at Acoustical Engineering Department of the Federal University of Santa Maria (UFSM) in Brazil," p. 8.

[9] Bengt-Inge L. Dalenbäck, "Room acoustic prediction based on a unified treatment of diffuse and specular reflection," *The Journal of the Acoustical Society of America*, vol. 100, no. 2, pp. 899–909, Aug. 1996.

[10] Douglas R Campbell, Kalle J Palomäki, and Guy J Brown, "Roomsim, a MATLAB Simulation of "Shoebox" Room Acoustics for use in Teaching and Research," p. 4.

[11] Larisa Stoltzfus, Alan Gray, Christophe Dubach, and Stefan Bilbao, "Performance portability for room acoustics simulations," in *Proceedings of the 20th International Conference on Digital Audio Effects*. 2017, pp. 367–374, University of Edinburgh.

[12] Benjamin Keinert, "enhanced sphere tracing," 2014.

[13] Dirk Schröder, "Physically Based Real-Time Auralization of Interactive Virtual Environments," p. 231.

[14] "VRWorks Audio SDK Overview.pdf," .

[15] Andrzej Kulowski, "Algorithmic representation of the ray tracing technique," *Applied Acoustics*, vol. 18, no. 6, pp. 449–469, 1985.