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 on 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] and to a different simulation algorithm [2]. Also, the rather special merits of such an approach, such as 4D reverberation and simple rounding of geometry, are shortly discussed and presented. 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 github.com/hrtlacek/rayMarchReverb.

1. INTRODUCTION

Sphere tracing as defined by [3] is used extensively in the so called "demo scene" to render 3D video demos via shaders in real time for decades. Sphere tracing is a version of the ray casting algorithm [4]. It relies on the geometry being defined as so called signed distance functions (SDFs), and does not directly support the import of standard 3D Polygonal geometry or meshes. One of the advantages lies in the algorithm's potentially improved speed in comparison to fixed-step ray casting. SDFs are used to describe implicit surfaces, typically via a function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$(although in principle, arbitrary dimensions are possible, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, see Section 6). In order to define the desired geometry, this f should be designed to return a negative value if the locus of the point is inside the geometry, a positive value if outside and 0 if on the surface. Additionally it is considered advantageous to design this function in a way so it is lipschitz continuous. If f is defined carefully, the distance to the nearest surface is returned by this function and therefore always known during stepping along a view-ray. From this follows that, the step size of a ray casting algorithm can be dynamically adjusted, see Figure 1, resulting in fewer iterations along a ray, and therefore in significant speedups. This dynamic adjustment of the step size to unbound spheres around the current step the core idea of sphere tracing and the reason for its name[3].

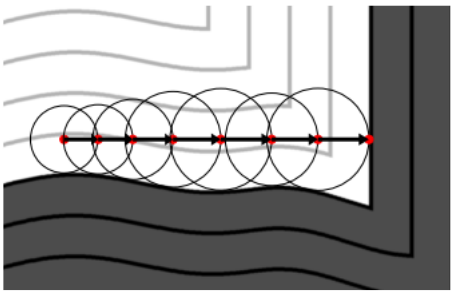


Figure 1: *Visualization of the sphere tracing algorithm in 2D. A ray is sent from the source (left) to the right until it hits a surface, always moving the maximum distance as the SDF informs the tracing algorithm about the distance to nearest surface.*

1.1. Previous Work

A lot of previous work exists both in the field of Ray/sphere tracing and RIR estimation. As shown in [5] and [1] there are numerous approaches for estimating RIRs. Besides Ray-based methods such as the "Image Source Method", Ray-tracing and hybrid approaches, wave-based methods such as Finite Differences are becoming increasingly interesting due to advancements in computation power and research. Still, wave-based methods seem to be too slow for real-time applications. With the Pascal Architecture, NVIDIA introduced real-time ray tracing done on the GPU with NVIDIA VRWorks™ Audio [6].

Sphere Tracing

Sphere tracing itself was first described by [3]. Many authors since then described improvements in speed e.g. [7] or the addition of features such as [8], [9] or the activity of the shadertoy.com-community. 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 [10] but also some commercial software has implemented raymarching and SDFs per default such as `Houdini` or `Notch`.

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 rel-

atively 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 is one common method of approaching the problem of RIR estimation. Sphere tracing has a number of advantages over ray tracing meshes or polygonal surfaces: It is "procedural" by default, since all geometry is defined by implicit surface equations. Sphere tracing approximates cone tracing for reducing aliasing artifacts in the pixel domain[3]. In the audio domain, beam tracing is considered to have advantages but is very time consuming in a non-SDF setup[5]. 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.

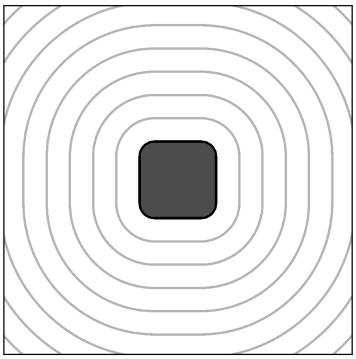
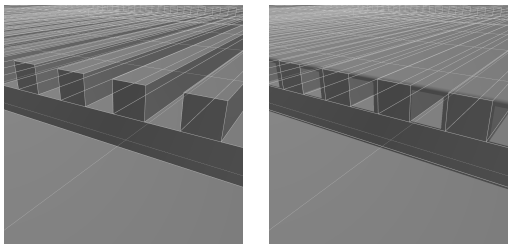


Figure 2: *rounding the box given in Equation 1 by subtraction of* 0.7. *Visible iso-lines (gray) are generated by the distance function. The subtraction shifts the surface to a different, more round, iso-line.*

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. For example [11] gives an overview of GPU development environments suitable for RIR estimation. The choice of a shader has the advantage of being more operating system independent and hardware independent as for example the use of CUDA ties to NVIDIA GPUs. 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 use of the Graphics Library Shading Language (GLSL) is 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.



(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).*

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_xxR_yxR_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*

[3] 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) [12]. Unidirectional ray tracing has been used, also for simplicity reasons, although [13] 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 [14], [5]. 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 [15], [5], and [16] the receiver was allowed to grow in volume. While [15] and [16] use time as a factor to let the receiver grow, in this attempt the reflection count, k is used. Initially when a ray is sent, $k = 0$ and when it hits a surface, this counter is increased by one so the source grows by this factor for this particular ray. So instead of using time, the model provided in [5] is used and augmented with the k term:

$$V_r = (k+1)\omega d_{SR}\sqrt{\frac{4}{N}} \tag{3}$$

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 largely follows the original formulation in [3] and is implemented as:

```
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`. In this implementation a resolution of 1024 x 1024 is used, resulting in 1024^2 initial rays and a maximum RIR length of 1024^2. 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.*

In Listing 3, `castRay()` refers to a function that looks similar to the sphere tracing function in Listing 2 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 in one file that is referenced via an `include` statement into a compute shader to calculate the RIR and into a fragment shader for visualization. This is particularly handy during construction of geometry but also makes it possible to calculate the RIR of existing sphere traced scenes easy for reverberation in V/A performance or other 3D video content.

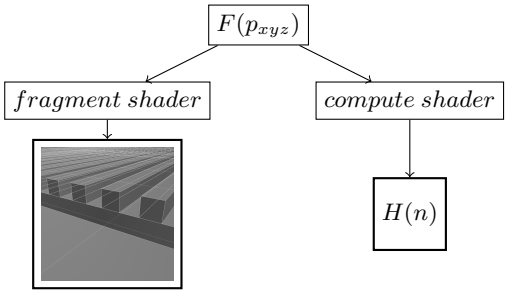


Figure 4: *A lot of the code, especially the map function, $F(p_{xyz})$, which contains the SDF can be shared between a fragment shader used for visualization and a compute shader responsible for RIR calculation.*

4. IMPULSE RESPONSE GENERATION

The room is assumed to be a linear time invariant (LTI) system. 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 τ_s in the impulse response, $H(n)$,that corresponds to the distance:

$$H(n) = \sum_{m=0}^{M} \delta(n - \tau_s(m)) \cdot \alpha(m) \cdot -1^{K(m)} \tag{5}$$

The total attenuation per ray $\alpha(m)$ can be computed by using a material dependent coefficient of reflection α_{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:

$$\alpha(m) = \prod_{k=0}^{K(m)} \alpha_{mat}(k, m) \tag{6}$$

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

$$\alpha(m) = \alpha_G^{K(m)} \tag{7}$$

Additionally, a proof-of-concept for frequency dependent loss for each reflection is introduced into the model. As a computationally efficient heuristic, a binomial filter is used [17], [18]. A simple one-zero filter $G(z)$ is applied at each reflection:

$$G(z, K) = (1 + z^{-1})^K \cdot \frac{1}{2}^K \tag{8}$$

The advantage of using such a simple filter is that a K-stage cascade's impulse response can be computed easily without applying the filter repeatedly. By doing the discrete time inverse fourier transform of the transfer function $G(z)$ the N-length impulse response is obtained:

$$G(n, K) = \mathfrak{F}^{-1}\{G(z)\} = (\frac{1}{2})^K \frac{1}{N} \sum_{\omega=0}^{\pi} (1 + e^{-j\omega})^K e^{j\omega n} \tag{9}$$

A binomial filter's impulse response converges to a gaussian bell curve which can be approximated as:

$$G(n, K) \approx \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2}(\frac{n-\mu}{\sigma})^2} \tag{10}$$

with

$$\sigma = \sqrt{K 0.231 + 0.562} \tag{11}$$

and

$$\mu = \frac{K}{2} + \frac{1}{2} \tag{12}$$

The right hand side of Equation 10 is simply the normal distribution, which is computationally efficient to calculate for any K.

So instead of adding $\delta(n - \tau(m))$ into the RIR, simply $G(n - \tau(m), K)$ is used. Similar to many other implementations, a high-pass filter is applied to the resulting RIR to compensate for low-frequency components resulting mainly from the spacial extendedness of the sound receiver shape.

After computation of the RIR, it needs to be written to the shader's output texture. So, an intermediate rectangular representation of N by M pixels is produced. The RIR, $H(n)$ is written to the Texture $T(x, y)$:

$$T(x, y) = H(x \bmod N + \frac{y}{M}) \tag{13}$$

Depending on the use-case (saving the RIR to a file, possible auralisation or convolution on the GPU) this data then needs to be fetched from the GPU and the process of needs to be reverted to obtain a one-dimensional signal again.

5. RESULTS

All of the following results have been computed with 10 reflection passes, and 1024^2 rays. The computations were made on a strong consumer-grade machine (Intel i7 CPU, NVIDIA Geforce 1080ti). The computation time for simple scenes such as scene 1, rigid in [1] was at $< \frac{1}{60}$ seconds including the computation of a simple visualization in real time. The most complex scenes(geometrically and in regards to the amount of reflections) that where tested, the shoebox scene and scene 1 with added diffusor, where computed in less than $< \frac{1}{50}$ seconds. As expected, the method therefore outperforms CPU based methods in computation time.

The results are compared to a different simulation using the image source method [2] and [1] who generated a dataset featuring recordings in in an anechoic chamber with simple shapes. Note that for the comparison with [1] in figure 7 ignores the frequency response of the speaker and microphone that was used in the original recording. As can be seen in the plot, the original recording features visible amount of difference to an ideal impulse even with the direct signal.

Since this implementation lacks of features such as material-specific reflection coefficients and passes for multiple octave bands, it is not possible to accurately compare previous work with the proposed method by simply using the same materials. It can be shown that by adjusting reflection coefficients, the method can be matched up with existing work.

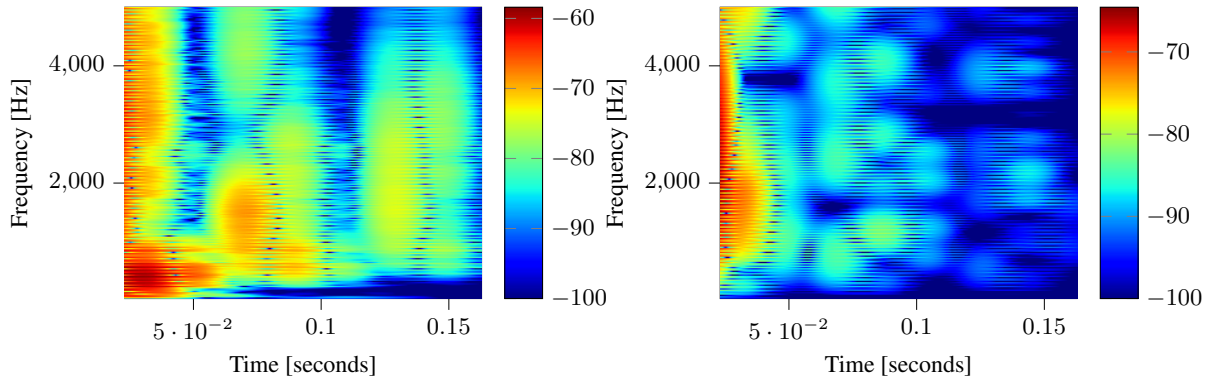
The shoebox scene features a small room with dimensions 3 x 4 x 2.5 meters. Note that [2] used a sampling rate of 16kHz and has been upsampled to 44.1 kHz for comparison reasons. This probably led to the pre-ringing artifacts present in Figure 9.

6. CONCLUSIONS AND FUTURE WORK

This work could show that RIR estimation via SDFs in a compute shader in real-time compatible manner is possible. It remains to be tested how far this method can reach. While it still is computationally costly to do these calculations in general, the use of SDFs can offer some significant advantages such as their procedural nature, their efficiency and simplicity. A number of improvements both in speed and accuracy are possible in the proposed technique, such as material dependent attenuation and scattering for example. Also, a number of speed improvements and optimizations could be implemented such as the removal of if statements in the SDF function. Furthermore a more detailed comparison and more mature and rigorous mathematical analysis of the process seems promising as well as an analysis of the sphere-tracing-specific artifacts and their effects in the audio domain. As mentioned in Section 1.2, sphere-tracing seems to offer a surprising simplicity and elegance for example when it comes to smoothing geometry. Simply the fact that it is a rather different approach than classical ray casting of polygons seems to promise opportunities for further research.

An SDF's closeness to classical mathematical structures (in contrast to sets of triangles, vertices, edges etc.) might lead to easier analysis or simpler comparison of different algorithms. Their popularity in the graphics community leads to constant development and the shier activity in the field seems to promise greater ease of use in the future.

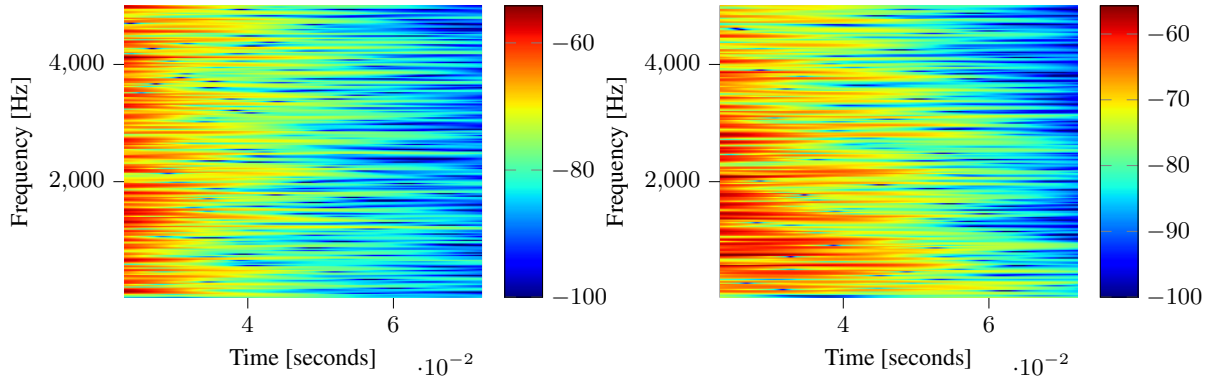
Certain rather experimental ideas also are easy to realize in this technique. Due to the aforementioned connection between geometry formulation in sphere tracing and mathematical formulas, this algorithm is often used to render fractals. In this context this would



(a) *Original recording, scene 1, multiple reflections from [1].*

(b) *Simulated using the proposed method.*

Figure 5: *Comparison of spectra of a real impulse response (a) and the proposed method (b).*



(a) *Spec by matlab.*

(b) *Simulated.*

Figure 6: *Impulse response of shoebox room. Computed using the proposed method and [2].*

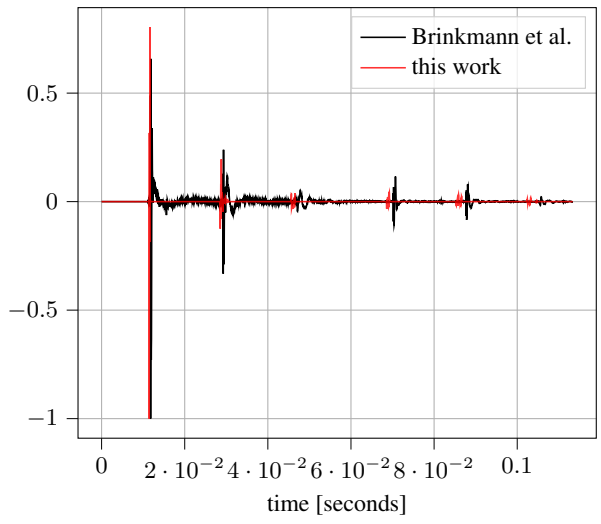


Figure 7: *RIR computed using the proposed method and measured by [1], scene 3, multiple reflection at opposed MDF plates.*

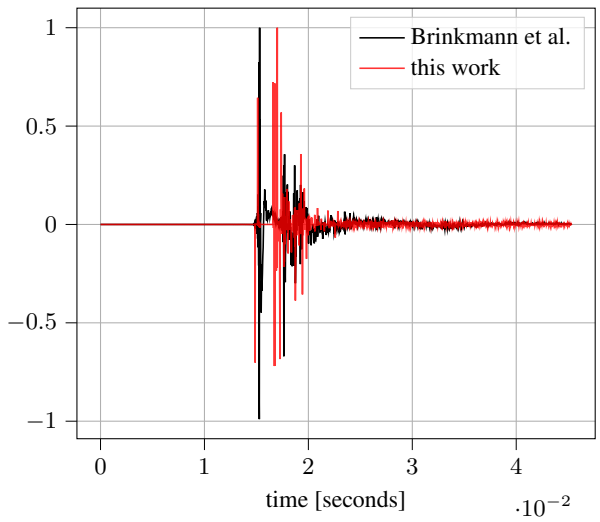


Figure 8: *RIR computed using the proposed method and measured by [1], scene 1, a single reflection with a diffuser.*

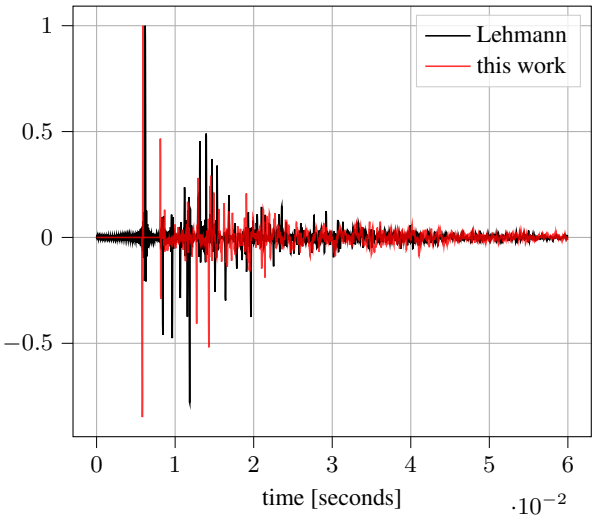
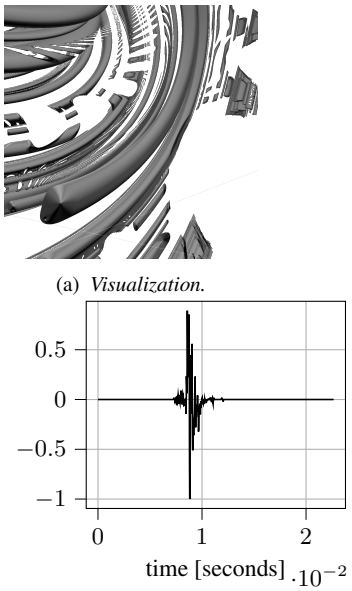


Figure 9: *Impulse response of shoebox room. Computed using the proposed method and [2].*



(a) *Visualization.*

(b) *RIR simulated using the proposed method.*

Figure 10: *Visualization (a) and RIR (b) of a 3-dimensional projection of a 4-dimensional function, the Julia Set [19] .*

mean that it is rather easy to calculate the RIR of a menger sponge or a Julia Set(see Figure 10). Moreover, many geometries/SDFs can be formulated in a dimension independent way, resulting in a straight forward way to render 4-dimensional geometries, leading to the idea of calculating the RIR of a 4-dimensional room. While these thoughts don't offer any apparent practical application and might "only" lead to aesthetically interesting results, it is possible to find applications in the sonification of high dimensional data.

7. ACKNOWLEDGMENTS

This work was supported in part by the university of Applied sciences St.Poelten.

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] Eric Lehmann, "Fast simulation of acoustic room impulse responses (image-source method)," 2020.

[3] 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.

[4] Scott D Roth, "Ray casting for modeling solids," *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109–144, Feb. 1982.

[5] 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.

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

[7] Csaba Bálint and Gábor Valasek, "Accelerating Sphere Tracing," 2018.

[8] Inigo Quilez, "Inigo Quilez :: fractals, computer graphics, mathematics, shaders, demoscene and more," .

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

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

[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] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Matthew Aubury, "Binomial Filters," *Journal of VLSI Signal Processing*, vol. 12, pp. 35–50, 1996.

[18] Konstantinos G Derpanis, "Overview of Binomial Filters," p. 3.

[19] Inigo Quilez, "Inigo Quilez :: fractals, computer graphics, mathematics, shaders, demoscene and more," .

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

[21] 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.

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

[23] Lukasz Jaroslaw Tomczak, "GPU Ray Marching of Distance Fields," p. 79.

[24] Inigo Quilez and Pol Jeremias, "Shadertoy BETA," .