# Transport Path Precomputation for Real-time Room Reverb

Gregor Mückl
Karlsruhe Institute of Technology
Karlsruhe, Germany

Carsten Dachsbacher
Karlsruhe Institute of Technology
Karlsruhe, Germany

## ABSTRACT

Believable immersive virtual environments require accurate sound in addition to plausible graphics to be convincing to the user. We present a practical and fast method to compute sound reflections for moving sources and listeners in virtual scenes in realtime, even on low end hardware. In a preprocessing step we compute transport paths starting from possible source positions and store virtual diffuse sources along the path. At runtime we connect these subpaths to the actual source position and the virtual diffuse sources to the microphone to compute the impulse response. Our method can be implemented as a single pass GPU rendering process and requires only a few milliseconds for each update.

## CCS CONCEPTS

• **Computing methodologies** → **Real-time simulation**; **Interactive simulation**; *Virtual reality*;

## KEYWORDS

room acoustics, interactive room reverb computation, real time room acoustics

**ACM Reference Format:**
Gregor Mückl and Carsten Dachsbacher. 2018. Transport Path Precomputation for Real-time Room Reverb. In *I3D '18: I3D '18: Symposium on Interactive 3D Graphics and Games, May 4–6, 2018, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3190834.3190840

## 1 INTRODUCTION

Especially the recent revival of consumer virtual reality applications has highlighted the need of convincing acoustics simulations for immersive experiences. However, to avoid discomforting side effects to the user, consistent high framerates are also required. This is challenging even for modern high performance graphics hardware. Therefore, few computational resources tend to be available for other tasks, such as the simulation of sound propagation.

Fast simulation methods based on geometric acoustics, most notably path tracing, have been developed during the recent years. The room acoustic rendering equation [Siltanen et al. 2007], which is conceptually similar to the rendering equation published by Kajiya [1986], provides the theoretical basis for ray-based simulations of sound reflections. Methods based on geometric acoustics are especially suited to larger scale scenes. In contrast, wave propagation-based methods (e.g. Boundary Element Method, Finite-Difference Time-Domain with Adaptive Rectangular Decomposition [Mehra et al. 2012], or Pseudo-Spectral Time Domain Methods) quickly become infeasible for large and complex scenes due to their tendency to scale polynomially, even though those methods model diffraction effects directly.

In this paper, we present a practical approach to computing believable high order impulse responses for arbitrary scenes on a GPU in real time using ray acoustics. In a preprocessing step it computes path sections along which sound can be reflected. At runtime, the normal hardware rendering pipeline is used to connect these sections to the source and microphone and to compute the room impulse response. Our method allows sources and listeners to move dynamically through a scene while the scene itself is assumed to be static. It is computationally inexpensive and easy to implement. We show that it is able to achieve real time performance in complex environments even on low end hardware.

## 2 RELATED WORK

Just as ray optics is a valid model of light behaviour when wavelengths can be considered small compared to the scene geometry, the same holds for ray acoustics. Siltanen et al. [2007] use this fact to derive a room acoustic rendering equation using only ray acoustics and the requirement that reflections can be modelled using acoustic BRDFs. The result has the structure of the familiar rendering equation with the addition of retardation due to the finite speed of sound.

We make use of these similarities. Our method can be considered a variation of manylight methods, which have been used to compute indirect illumination based on few virtual light sources distributed in the scene by following paths from a light source. Originally developed by Keller [1997], these methods have evolved significantly [Dachsbacher et al. 2014]. As these methods trace paths starting from the light sources in the scene they are inherently related to Monte Carlo Light Tracing [Dutré et al. 1993]. We exploit this concept to reuse acoustic transport paths.

Many methods have been proposed for interactively finding purely specular paths through a scene. Both beam tracing and path tracing based methods have been proposed. Funkhouser et al. [2004] proposed an interactive beam tracing method based on precomputing all possible propagation paths in a static scene. AD-Frustum [Chandak et al. 2008] performs beam tracing with simplified beam-geometry intersection to find potential paths to the listener which are then validated by ray tracing. GSound [Schissler and Manocha 2011] shoots stochastic paths from the source into the scene. Lists of intersected primitives are stored for valid paths for fast path re-validation after the source or listener moved. This caching scheme was later extended to include diffuse paths and


*I3D '18, May 4–6, 2018, Montreal, QC, Canada*

ACM ISBN 978-1-4503-5705-0/18/05…$15.00
https://doi.org/10.1145/3190834.3190840

diffraction [Schissler et al. 2014]. Recently, a combination of diffuse path caching and clustering of primary sources has been proposed to compute room reverb for large numbers of sources interactively [Schissler and Manocha 2016]. Bidirectional path tracing combined with path caching can be used to obtain accurate near-realtime diffuse impulse responses in complex scenes [Cao et al. 2016]. An error metric based iterative distribution of sample budgets to reflection orders improves convergence of this method.

Antani et al. [2011] proposed precomputing the diffuse transfer between sample points placed on scene surfaces. The full time-dependent transfer between each pair of points is computed. To store the resulting set of echograms in memory, they created a lossy compression scheme. At runtime, these transfer echograms are accumulated to form the late impulse response.

Raghuvanshi et al. [2014; 2010] extracted a parameterised shape of the room reverb from wave based precomputations of the room acoustics in a virtual scene with source positions throughout the scene and reconstructed the reverb based on a fast parameter lookup during runtime. Cheng [2014] presented an interactive acoustic raytracer and auralization that is executed entirely on the GPU.

## 3 ROOM ACOUSTICS USING REUSABLE TRANSPORT PATHS

We briefly recapitulate the room acoustic rendering equation [Siltanen et al. 2007]:

$$L_o(x_{i-1} \leftarrow x_i) = L_e(x_{i-1} \leftarrow x_i) + \int_{\mathcal{M}} f(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G'(x_i \leftrightarrow x_{i+1}) L_i(x_i \leftarrow x_{i+1}) dA_{x_{i+1}} . \tag{1}$$

The points $x_0 \ldots x_n$ are path vertices. $x_0$ is the microphone location and $x_n$ is the source location $x_S$. The other $x_i$ are points on surfaces. This equation describes the radiance emitted from a point $x_i$ on a surface towards a point $x_{i-1}$ as the sum of the emission $L_e$ and the reflection of all incoming radiance according the BRDF $f$. $G'$ is an extended geometry term containing a retardation operator $S_r$:

$$G'(x_i \leftrightarrow x_{i+1}) = S_{|x_i - x_{i+1}|} \, G(x_i \leftrightarrow x_{i+1}) . \tag{2}$$

The retardation operator represents the travel time of sound with velocity $c$ over a distance $r$:

$$S_r I(t) = I\left(t - \frac{r}{c}\right) . \tag{3}$$

Multiple applications of this operator result in cumulative time shifts:

$$S_{r_1} S_{r_2} I(t) = S_{r_1 + r_2} I(t) = I\left(t - \frac{r_1 + r_2}{c}\right) . \tag{4}$$

We want to compute the reflected sound intensity $I_M$ from all scene surfaces arriving at a microphone $M$. A microphone registers incoming sound according to its directional characteristics. This results in the integral

$$I_M(x_0) = \int_{\mathcal{M}} W_M(x_0 \leftarrow x_1) L_0(x_0 \leftarrow x_1) \, dA_{x_0} , \tag{5}$$

where the reflected sound is the result of multiple reflections of the radiance $L_S$ emitted by a sound source:

$$L_0 = \sum_{n=1}^{\infty} \int_{\mathcal{M}^{n-1}} \prod_{i=1}^{n-1} f(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G'(x_i \leftrightarrow x_{i+1}) \cdot L_S(x_{n-1} \leftarrow x_n) \, dA_{x_1} \ldots dA_{x_{n-1}} . \tag{6}$$

This integral equation describes the sound propagating of all purely reflective paths within the scene. The points $x_0 \ldots x_n$ are path vertices. $x_0$ is the microphone location and $x_n$ is the source location $x_S$. The other $x_i$ are points on surfaces.

Denoting the sequence of points $x_0 \ldots x_n$ as $\overline{x}_n$ and using the definitions

$$T_i = f(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G'(x_i \leftrightarrow x_{i+1}) \tag{7}$$

and

$$T(\overline{x}_n) = \prod_{i=1}^{n-1} f(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G'(x_i \leftrightarrow x_{i+1}) \tag{8}$$

$$= \prod_{i=1}^{n-1} T_i \tag{9}$$

we can rewrite equations 5 and 6 in the form

$$P(\overline{x}_n) = \underbrace{\int_{\mathcal{M}} \cdots \int_{\mathcal{M}}}_{n-1 \text{ times}} L_e(x_{n-1} \leftarrow x_n) T(\overline{x}_n) dA_{x_2} \ldots dA_{x_{n-1}} \tag{10}$$

$$I_M(\overline{x}_n) = \int_{\mathcal{M}} W_M(x_0 \leftarrow x_1) P(\overline{x}_n) dA_{x_1} \tag{11}$$

with $I_M(x_0)$ being the sum over all $I_M(\overline{x}_n)$. For a path with two or more intermediate vertices, we split the integrand into three factors: the first reflection of sound at a surface, the last reflection before reaching the microphone and the third part for all the remaining reflections. Formally, we rewrite equation 9 thus as:

$$T(\overline{x}_n) = T_1 \cdot \left( \prod_{i=2}^{n-2} T_i \right) \cdot T_{n-1} \tag{12}$$

Here, $T_1$ corresponds to the last reflection before the microphone (hereafter *microphone segment*, yellow segments in Fig. 1c), $T_{n-1}$ corresponds to the first reflection from the source (hereafter *source segment*, green segment in Fig. 1c) and the product inbetween is the *transport path* (red subpath in Fig. 1c). The transport path part is naturally independent of the source and microphone locations and can therefore be reused if either moves as long as visibility along the path remains unchanged. This allows us to create transport paths in a precomputation step and to only update the source and microphone segments during runtime. This is central to our algorithm.

In the case of a path of length one, there is only a single reflection. Then, $P(\overline{x}_1)$ equals 1 and the integral in this case becomes:

$$I_M(x_0) = \int_{\mathcal{M}} W_M(x_0 \leftarrow x_1) f(x_0 \leftarrow x_1 \leftarrow x_2) \cdot G'(x_1 \leftrightarrow x_2) L_e(x_1 \leftarrow x_2) dA_{x_2} dA_{x_1} . \tag{13}$$

Obviously, these paths cannot be segmented, but only require a single BRDF evaluation. This is a trivial special case in our algorithm.

# 4 REAL-TIME IMPULSE RESPONSE COMPUTATION

We now present a simple and practical real-time algorithm that exploits the fact that the transport paths described above can be reused. In a preprocessing step, it computes a set of valid transport paths throughout the scene. During runtime it iteratively connects them to the current source and microphone position to compute updated room impulse responses.

We pick a set of evenly distributed points on the scene surfaces as starting points for the transport paths. Typically between 10000 and 100000 starting points are required, depending on the scene. For each starting point, we then trace a random path through the scene. The initial direction must be sampled uniformly regardless of the BRDF as the incoming direction at this point is not known, making importance sampling impossible. All further reflections along the path use importance sampling, though. For each new vertex that is appended to the path, we treat the resulting path as a separate transport path that can be used during runtime and store it. Specfically, we store:

- the path length,
- the starting point (location, normal, outgoing direction, material),
- the intersection point (location, normal, incoming direction, material),
- the value of the product $\prod_{i=2}^{n-2} T_i$.

The latter product does not exist for paths with only a single reflection. The contribution of these short paths can be determined solely by the location and material properties of the single path vertex. Thus, we mark these paths with a flag. These paths are connected to the source and microphone at runtime.

We terminate the paths after they reach a predefined (geometric) length that corresponds to a user-specified impulse response duration to avoid any bias in the result. Shorter paths would result in missing contributions at the end of the impulse response if the connections to the source and microphone are both so short that the total possible path length is shorter than the desired impulse response length. Conversely, precomputed paths need not be longer because the connections to the source and microphone will always add additional signal runtime and therefore move the transport path contribution beyond the desired impulse response length.

After this path generation, we can iteratively generate an impulse response from the stored paths for arbitrary source and microphone positions within the scene. We recompute full paths by reconnecting the transport paths to the current source and microphone positions. This requires testing the visibility of the starting point from the source location and the end point from the microphone location. If both tests succeed, the completed path contributes to the room impulse response and the contributions of the source segment and microphone segment need to be computed. Otherwise, the path does not contribute and is discarded. The final contribution of the path is accumulated in a histogram of average contributions over time (described in section 5). Both visibility checks for each path are from the same fixed source or microphone position and can be performed using shadow mapping. Due to this and the precomputed path contribution, updating the impulse response is very fast.

Note that the explicit connections required in the source and microphone segments contain the geometry term with an inverse square distance to the source or microphone, respectively. It is well known that these terms diverge for small distances, resulting in paths with overly high contributions which distort the final histogram and need to be suppressed [Dachsbacher et al. 2014]. We can achieve this easily by clamping the product of the squared distances of source and microphone. We found that clamping each geometry term individually is much less effective because it does not cover the case where the source and microphone distances are still in an acceptable range individually, but their combined product is small enough to show divergent behaviour.

# 5 ALGORITHM IMPLEMENTATION

Our algorithm has an initialization phase that creates the transport paths and uploads them to the GPU. At runtime, it uses the GPU's rendering pipeline to iteratively update a spectral room impulse response histogram and downloads it to the host where it is processed and convolved with the actual sound signal.

We compute room impulse responses for multiple frequency bands simultaneously. We allocate a two-dimensional single channel floating point texture in which the final result is accumulated. There is one row in this texture for each frequency band and one column for each histogram bin (time axis). The choice of time resolution of the histogram is a tradeoff between resolution and convergence speed. A higher number of narrower histogram bins captures more details in the impulse response but also requires more contributions to achieve convergence. We find that suitable histogram bin sizes are between 1 and 5 milliseconds. We chose to use 12 bands (every other band of the Bark scale) in our implementation. This scale has perceptually equally important frequency bands [Zwicker 1961]. The layout of the histogram texture allows us to accumulate path contributions by rendering vertical lines across the histogram texture with additive blending enabled. This design choice allows us to make use of the rendering pipeline with only simple vertex and fragment shaders.

## 5.1 Initialization Phase

We use a simple path tracer to sample the transport paths as described in section 4. We generate many transport paths with different end points from the same starting points and store the data for each separately. Furthermore, as we need to render a line for each transport path during runtime, both vertices must have access to this information. To avoid needless data duplication in the vertex buffer, we store most information in a set of textures that is referenced by the vertex buffer by index instead.

As illustrated in Fig. 3, we store the position, normal, and outgoing direction of the starting point in a three channel floating point texture, and the position, normal, and incoming direction of the end point in another. The throughput and time delay are stored in separate single channel floating point textures. The vertex buffer then only needs to contain indices referencing the starting points and materials of each path for each vertex. The end points, however, are arranged in the same order as the vertices in the vertex buffer. The vertex shader can thus determine the endpoint index based on the current vertex ID.

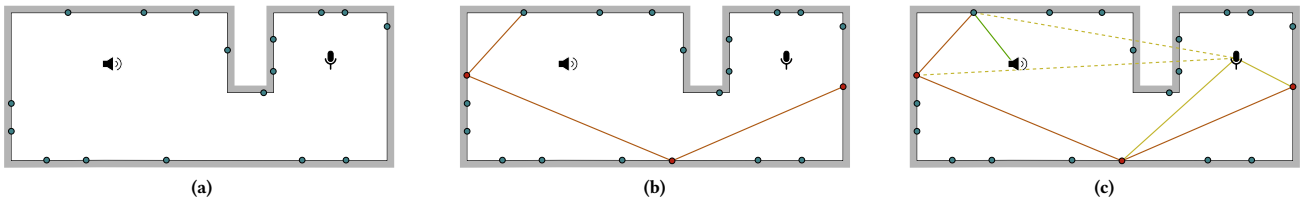



**Figure 1: Our algorithm selects uniformly distributed random starting points (a) from which it traces random transport paths through the scene in a precomputation step (b) and iteratively connects them to the source and microphone at their current locations during runtime to compute the corresponding impulse response contributions (c).**

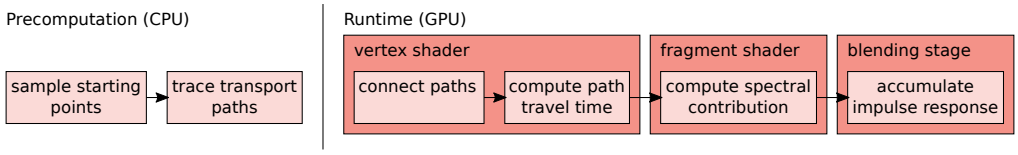



**Figure 2: Schematic overview over the processing steps of our algorithm: a precomputation step traces transport paths through the scene. At runtime, the rendering pipeline is used to iteratively reconnect these paths to the source and microphone to form full paths. Their contribution is added to an impulse response histogram.**

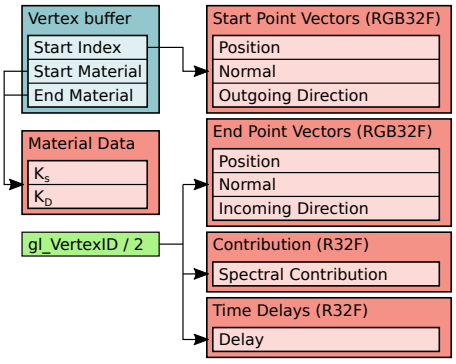



**Figure 3: Data duplication on the GPU is reduced by storing most data in textures and using the vertex buffer only for indexing into these textures. The index for transport path end points is derived from the current vertex ID, reducing vertex buffer size further.**

The material properties are stored in a separate texture and are not duplicated. Our method is independent of the choice of BRDFs which can have arbitrary parameters. However, our implementation only contains a Phong BRDF with separate diffuse and specular coefficients for each frequency band, but a common exponent. While simple, it is sufficient to explore the effects of BRDFs with varying glossyness and is not a limitation of our method.

We assume the scene surfaces to be closed objects. Thus, only the front faces are considered as starting points. Backfaces often form the insides of walls or other relatively tight and enclosed spaces. The tightness of these spaces result in an often excessive number of reflections before the path reaches its target length and is terminated. Due to the enclosed nature of the spaces these paths are confined to, they can never contribute to the result, but would consume a huge number of computational resources. While other mitigations like manually tagging scene surfaces are conceiveable, simply limiting transport path starting points to front faces sufficiently reduces this problem in our example scenes.

## 5.2 Interactive Impulse Response Update

At runtime, the algorithm must determine for each transport path whether a full path can be formed from it, and if so, compute its contribution and add it to the histogram. The visibility tests towards the source and microphone positions can be easily performed using shadow mapping. Thus, as the first step in each iteration, our algorithm renders two omnidirectional shadow maps, one centered around the source and one centered around the microphone.

To compute the actual impulse response, we chose to render a line primitive for each transport path using vertex and fragment shaders that perform the remaining steps of computing the impulse response histogram. The vertex shader first tests the starting point and the end point for visibilty against the source and microphone using the rendered shadow maps. If either test is unsuccessful, the path is discarded by moving the output vertices outside the coordinate space of the histogram texture. If both are successful, the total path length and travel time is computed, and from that the coordinates of the output vertices so that they match the corresponding histogram bin. Finally, the vertex shader computes the required geometry terms and other terms common to all frequency bands and passes the result on to the fragment shader.

Since the histogram texture has one row of texels for each spectral band, the rasterizer invokes the fragment shader once for each band within the histogram bin to which the path contributes. The shader determines the current band from the fragment position within the texture and computes the contribution to the specific spectral band. For this, it evaluates the BRDFs associated with the transport path endpoints based on the BRDF parameters for the

specific frequency band and reuses the geometry term computed in the vertex shader.

The resulting lines are rendered with additive blending enabled to accumulate the path contributions in the histogram texture. Exploiting the OpenGL blending stage in this way allows the histogram generation pass to be simple and have very high performance.

Finally, the spectral histogram texture is downloaded from the GPU to create an impulse response for audio processing. While our implementation performs this transfer synchronously, it could easily be made asynchronous at the cost of some added latency until the updated room impulse response becomes audible.

## 5.3 Audio Processing

Our algorithm generates one histogram per frequency band, resulting in a mix of information in the time and spectral domains. We need to convert this to a pure time domain impulse response upsampled to the sample rate of the source signal. To achieve this, we first upsample each band of the histogram individually to the target sample rate. These upsampled histograms are implicitly band limited by the time resolution of the input histogram. They would act as a low pass filter on the audio signal if they were applied as they are, which is undesirable. However, this can be mitigated by multiplying the histograms with white noise [Lentz et al. 2007]. In other words, the histogram acts an envelope for a band-unlimited equal energy signal. Then, we perform a DFT on each resulting partial impulse response and merge them by combining the relevant frequency band of each response spectrum into a single spectrum. An inverse Fourier transformation of the resulting spectrum produces the final impulse response. Finally, the actual audio signal is convolved with the impulse response within the audio engine in real time, switching to an updated impulse response whenever it becomes available.

Due to the reuse of precomputed paths, the impulse responses are stable and vary slowly with source and listener movement. For this reason, our method does not require averaging or filtering the impulse responses over multiple iterations of the algorithm.

To guarantee fast execution times, all Fourier transformations involved should have lengths of powers of two exclusively. Thus, we choose the number of samples of the resulting impulse response length to be an exact power of two and adjust the number of histogram bins accordingly.

# 6 RESULTS

We implemented the algorithm using OpenGL 3.0. Our implementation uses cube shadow maps for visibility testing on the GPU. We tested it on one older and one new newer hardware platform listed in Tab. 1 to assess performance across a wider range of target platforms. We refer to this table for each result.

## 6.1 Accuracy and Performance

We compare our method to a multithreaded path tracer based on Embree. We use the same number of paths for our method and the path tracer. Our reference path tracer uses all available CPU cores on each system. We have used the scenes in Tab. 2 for our evaluation. The measured execution times are listed in Tab. 4.

**Table 1: Hardware used for performance evaluation.**

| System | CPU | GPU |
|---|---|---|
| System 1 | Intel Core i7-6800K, 3.4GHz | nVidia GeForce GTX 1080 |
| System 2 | Intel Core i7-3770, 3.4GHz | nVidia GeForce GTX 670 |

*Performance.* Our method consistently performs much faster than our reference path tracer. Shadow map rendering performance naturally depends on the geometric complexity of the scene. The runtime of the histogram generation step, however, is completely decoupled from scene complexity and depends only on the number of transport paths. Even on older hardware, our method clearly provides interactive update rates for most scenes. The only exception is the Bistro scene which contains extremely detailed geometry, slowing down shadow map rendering. However, since the actual layout of this large scene is rather simple, the number of transport paths required to achieve acceptable convergence remains reasonably small and thus the histogram rendering step still remains very fast.

*Accuracy.* Tab. 2 shows that the path traced impulse response is sometimes more accurate. The reason for this is that the path tracer starts its paths always from the microphone while our method tries to connect the microphone to fixed points on surfaces, which may or may not be occluded. This may reduce the number of usable transport paths, depending on the scene and the current location of the source and microphone. Also, transport paths starting and terminating on glossy surfaces can challenge the accuracy of our method if the density of endpoints is not high enough. For a faithful sampling of a glossy reflection, at least one end point must exist on the surface where the reflection occurs with a suitable incoming or outgoing ray direction. Otherwise, that reflection is missed entirely. This is analogous to the well known drawback of many lights methods in rendering [Dachsbacher et al. 2014].

*Memory usage.* Our method consumes a moderate amount of GPU memory as shown in Tab. 3. While the memory required to store position, normal and incoming directions for end points is comparatively high, sharing the equivalent data for starting points between paths keeps the related memory usage small. This shows that memory usage is strongly related to the number of endpoints. Also, the number of endpoints scales inversely with mean free path length in the scene: the smaller the mean distance between scene surfaces, the more reflections occur and the more endpoints are generated. Our choice of 12 distinct frequency bands results in a high memory usage for path contributions. As the number of frequency bands is arbitrary, memory requirements for contributions is thus adjustable. We use full 32 bit precision for all floating point values, although some values, for example, direction vectors could probably be stored with less precision without noticable impact on the final result. However, we did not experiment with more compact encodings, as our test systems were able to handle the generated amount of data without any problems.

**Table 2: Scenes used for evaluation. "Paths" is the number of path starting points and "Endpoints" denotes the number of valid path end points generated from them in the preprocessing step. GPU memory is the sum of vertex buffer and texture memory used.**

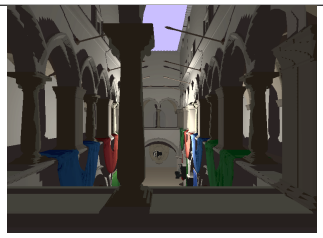
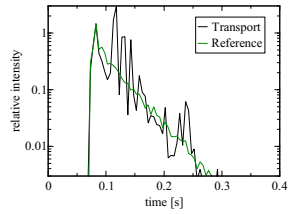
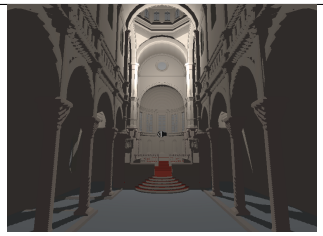
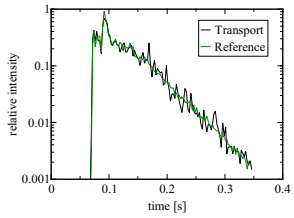
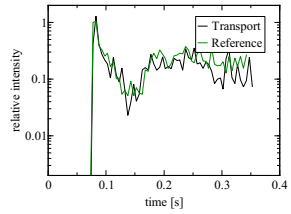
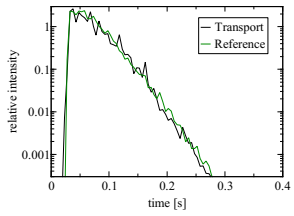
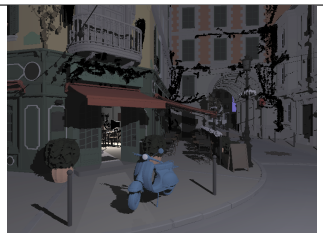
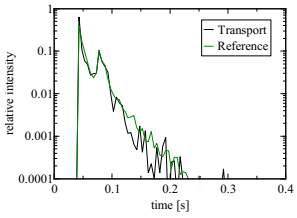
| Scene | Screenshot | Impulse Response | Paths | Endpoints |
|---|---|---|---|---|
| Sponza |  |   | 80000 | 915758 |
| Sibenik |  |   | 20000 | 213392 |
| Tradeshow |  |   | 100000 | 787026 |
| Door |  |   | 10000 | 433891 |
| Bistro |  |   | 250000 | 1522656 |

**Table 3: GPU memory usage breakdown for different scenes in MB.**

| Scene | VBO | starting points | end points | contributions | time delays | total |
|---|---|---|---|---|---|---|
| Sponza | 10.5 | 2.7 | 31.4 | 41.9 | 3.5 | 90.1 |
| Sibenik | 2.4 | 0.7 | 7.3 | 9.8 | 0.8 | 21.0 |
| Tradeshow | 9.0 | 3.4 | 27.0 | 36.0 | 3.0 | 78.5 |
| Door | 5.0 | 0.3 | 14.9 | 19.9 | 1.7 | 41.7 |
| Bistro | 17.4 | 8.6 | 52.3 | 69.7 | 5.8 | 153.8 |

## 6.2 Extensions

*Directional Microphone, HRTF.* So far, we have only demonstrated a perfectly omnidirectional single channel microphone. Alternatively, we can easily simulate an array of several narrowly directional microphones at the listener position and simulate the impression of the room reverb reaching the listener from different directions by applying the appropriate directional HRTF to each individual response [Embrechts 2007]. To achieve this, we extend the histogram texture to multiple histograms side by side, one for each directional microphone. The vertex shader then determines

**Table 4: Measured execution times. Times for shadow maps and histograms were taken using OpenGL timer queries. CPU total denotes total time spent on GPU from the start of shadow map rendering to the end of histogram download.**

| Scene | Hardware | Cube Shadow Maps [ms] | Impulse Response [ms] | CPU [ms] | PT [ms] |
|---|---|---|---|---|---|
| Sponza | System 1 | 1.10 | 0.62 | 2.00 | 366 |
| | System 2 | 4.09 | 2.21 | 6.75 | 612 |
| Sibenik | System 1 | 0.37 | 0.17 | 0.82 | 49.2 |
| | System 2 | 1.45 | 0.64 | 2.52 | 85.2 |
| Tradeshow | System 1 | 0.76 | 0.57 | 1.61 | 77.1 |
| | System 2 | 2.68 | 1.91 | 5.02 | 137 |
| Door | System 1 | 0.15 | 0.31 | 0.73 | 53.0 |
| | System 2 | 0.57 | 1.38 | 2.37 | 91.7 |
| Bistro | System 1 | 13.3 | 0.95 | 14.8 | 414 |
| | System 2 | 56.0 | 3.56 | 60.3 | 674 |

the directional microphone that registers the path contribution based on the incoming ray direction and emits the corresponding vertex positions. The individual histograms are then processed separately as outlined above, convolved with appropriate HRTFs for the microphone's direction, and accumulated separtely for each channel in a binaural setup.

Our implementation uses 24 directional channels (3 vertical divisions, 8 horizontal divisons). We find that this has a negligible impact on the GPU part of our method. However, the creation of the final impulse response is much more expensive, as the upsampling and spectral merging have to be done for each directional separately. It takes 65ms in a single thread on system 1 compared to 3ms in the omnidirectional case.

*Large Scenes.* To scale our method to very large scenes, the limit on the computed room impulse response length could be exploited to implement a dynamic streaming method to only load the currently relevant subset of the precomputed transport paths onto the GPU. The impulse response length naturally limits the distance that sound can travel in that time. Consider a transport path consisting of only a single vertex. This vertex must lie within a spheroid with the source and microphone in the focal points and the surface defined by the maximum possible travelling distance of the sound. Only then, sound reflected from that vertex can reach the microphone within the time span covered by the impulse response. If the vertex is outside the spheroid, this is never possible. Longer paths will naturally have tighter limits because of the additional travel time along the path. Thus, we can safely discard any path start or end points outside of this bounding volume without introducing any errors. This would allow us to spatially sort all paths by their endpoints and dynamically upload only those paths to the GPU that have endpoints within the current spheroid. We leave this implementation to future work.

## 7 CONCLUSION

We demonstrated a novel, and easy to implement method to compute room reflections for interactive reverb computation on the GPU. We showed that it is very fast and uses a moderate amount of resources on the GPU in scenes of realistic size and complexity. We achieved this by adapting the concept of instant radiosity. The resource usage is decoupled from the geometric complexity of the scene. If required, more sophisticated data structures reduce resource usage further, at the cost of an increase in the complexity of the implementation.

The main limitation of our method is that it does not simulate the effects of edge diffraction. In future work, we would like to investigate the inclusion of the Biot-Tolstoy-Medwin model in our approach [Siltanen and Lokki 2008].

Since the shadow map rendering is the most expensive part of our method, we would like to explore other methods for determining visibility, for example imperfect shadow maps [Ritschel et al. 2008] or distance field shadows [Wright 2015]. This would allow our method to scale to a higher number of sound sources.

In addition to the streaming implementation outlined above, we would like ot extend our method to update the transport paths at runtime. This would allow fully dynamic scenes as well as the dynamic generation of transport paths for streaming instead of reading them from a larger database of precomputed paths. Ideally, we would also find ways to automatically tune the algorithm's parameters to a scene without the need for user input.

We have shown that the concept of virtual point lights can be applied to room acoustics. This opens up new possibilities. For example, scalability of our method could be improved by a suitable clustering scheme for path end points.

## ACKNOWLEDGEMENTS

The Tradeshow scene was kindly provided by the GAMMA resarch group of the University of North Carolina at Chapel Hill.

## REFERENCES

L Antani, A Chandak, L Savioja, and D Manocha. 2011. Interactive Sound Propagation using Compact Acoustic Transfer Operators. (September 2011). Accepted for publication.

Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou. 2016. Interactive Sound Propagation with Bidirectional Path Tracing. *ACM Transactions on Graphics (SIGGRAPH ASIA 2016)* (2016).

A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha. 2008. AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation. *Visualization and Computer Graphics, IEEE Transactions on* 14, 6 (2008), 1707 –1722. https://doi.org/10.1109/TVCG.2008.111

Zuofu Cheng. 2014. *Design of a real-time GPU accelerated acoustic simulation engine for interactive applications.* Ph.D. Dissertation. University of Illinois at Urbana-Champaign. http://hdl.handle.net/2142/50364

Carsten Dachsbacher, Jaroslav Krivanek, Milos Hasan, Adam Arbree, Bruce Walter, and Jan Novak. 2014. Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum* 33, 1 (2014), 88–104. https://doi.org/10.1111/cgf.12256

Philip Dutré, Eric P. Lafortune, and Yves D. Willems. 1993. Monte Carlo light tracing with direct computation of pixel intensities. In *3rd International Conference on Computational Graphics and Visualisation Techniques.* Alvor, Portugal, 128–137.

J. J. Embrechts. 2007. Computation and applications of directional echograms in room and concert hall acoustics. *Proceedings of the 19th International Congress on Acoustics.*

http://www.montefiore.ulg.ac.be/services/acous/STSI/file/ICA_Madrid_2007.pdf

Thomas Funkhouser, Nicolas Tsingos, Ingrid Carlbom, Gary Elko, Mohan Sondhi, James E. West, Gopal Pingali, Patrick Min, and Addy Ngan. 2004. A Beam Tracing Method for Interactive Architectural Acoustics. *Journal of the Acoustical Society of America* 115, 2 (Feb. 2004), 739–756.

James T. Kajiya. 1986. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH '86)* 20, 4 (1986), 143–150. Issue 4. https://doi.org/10.1145/15886.15902

Alexander Keller. 1997. Instant Radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 49–56. https://doi.org/10.1145/258734.258769

Tobias Lentz, Dirk Schröder, Michael Vorländer, and Ingo Assenmacher. 2007. Virtual Reality System with Integrated Sound Field Simulation and Reproduction. *EURASIP Journal on Advances in Signal Processing* 2007 (2007).

Ravish Mehra, Nikunj Raghuvanshi, Lauri Savioja, Ming C. Lin, and Dinesh Manocha. 2012. An efficient GPU-based time domain solver for the acoustic wave equation. *Applied Acoustics* 73, 2 (2012), 83 – 94. https://doi.org/10.1016/j.apacoust.2011.05.012

Nikunj Raghuvanshi and John Snyder. 2014. Parametric Wave Field Coding for Precomputed Sound Propagation. *ACM Trans. Graph.* 33, 4, Article 38 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601184

Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin, and Naga Govindaraju. 2010. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Transactions on Graphics* 29, 4 (2010). https://doi.org/10.1145/1778765.1778805

Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008)* 27, 5 (2008).

Carl Schissler and Dinesh Manocha. 2011. GSound: Interactive Sound Propagation for Games. In *Audio Engineering Society Conference: 41st International Conference: Audio for Games*. http://www.aes.org/e-lib/browse.cfm?elib=15770

Carl Schissler and Dinesh Manocha. 2016. Interactive Sound Propagation and Rendering for Large Multi-Source Scenes. *ACM Trans. Graph.* 36, 1, Article 2 (Sept. 2016), 12 pages. https://doi.org/10.1145/2943779

Carl Schissler, Ravish Mehra, and Dinesh Manocha. 2014. High-Order Diffraction and Diffuse Reflections for Interactive Sound Propagation in Large Environments. *ACM Trans. Graph.* 33, 4, Article 39 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601216

Samuel Siltanen and Tapio Lokki. 2008. Diffraction modeling in acoustic radiance transfer method. *The Journal of the Acoustical Society of America* 123, 5 (2008), 3759–3759. https://doi.org/10.1121/1.2935340

Samuel Siltanen, Tapio Lokki, Sami Kiminski, and Lauri Savioja. 2007. The room acoustic rendering equation. *The Journal of the Acoustical Society of America* 122, 3 (2007).

Daniel Wright. 2015. Dynamic Occlusion with Signed Distance Fields. In *SIGGRAPH 2015 Courses*.

E. Zwicker. 1961. Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen). *The Journal of the Acoustical Society of America* 33, 2 (1961), 248–248. https://doi.org/10.1121/1.1908630