

## INTERACTIVE ACOUSTIC VIRTUAL ENVIRONMENTS USING DISTRIBUTED ROOM ACOUSTIC SIMULATIONS

Frank Wefers, Jonas Stienen, Sönke Pelzer, Michael Vorländer

Institute of Technical Acoustics,  
RWTH Aachen University,  
Aachen, Germany

{fwe, jst, spe, mvo}@akustik.rwth-aachen.de

### ABSTRACT

This publication presents how the computational resources of PC clusters can be used to realize low-latency room acoustic simulations for comprehensive virtual scenes. A benefit is not only to realize more complex scenes (including a multitude of sound sources, acoustically coupled-rooms with sound transmission), but also to minimize the system response times for prototyping applications (e.g. interactive change of materials or geometry) in simpler applications.

PC clusters prove to be a suitable platform for room acoustic simulations, as the incorporated algorithms, the image source method and stochastic ray-tracing, are largely free of data interdependencies. For the computation in massive parallel systems the simulation of a room impulse response is separated into individual parts for the direct sound (DS), early reflections (ER) and diffuse late reverberation (LR). Additional decomposition concepts (e.g. individual image sources, frequency bands, sub volumes) are discussed. During user interaction (e.g. movement of the sources/listeners) the system is continuously issued new simulation tasks. A real-time scheduler decides on significant updates and assigns simulation tasks to available cluster nodes. Thereby the three simulation types are processed with different priorities. The multitude of (asynchronously) finished simulation tasks is transformed into room impulse responses. Convolution with the audio signals is realized by non-uniformly partitioned convolution in the frequency domain. The filter partitioning is adapted to the update rates of the individual impulse response parts (DS, ER,LR). Parallelization strategies, network protocols and performance figures are presented.

### 1. INTRODUCTION

Nowadays fast algorithms together with powerful computers allow to simulate the acoustics in rooms in almost *real-time* compatible rates [1, 2, 3, 4]. This makes comprehensive simulation of the acoustics for interactive virtual environments possible. Such systems are valuable tools for architectural prototyping, room acoustic engineering and noise assessment. The computational demands for the required simulations are very high and their comprehensiveness is usually limited by the engaged hardware. Established geometrical acoustics (GA) methods, such as image source or ray tracing methods, are efficient enough to bring down computation times within the range of a few hundred milliseconds. Advanced aspects however, in particular diffraction modeling, are exceedingly complex and make the computation several magnitudes slower. Wave-based simulation techniques can nowadays be realized in

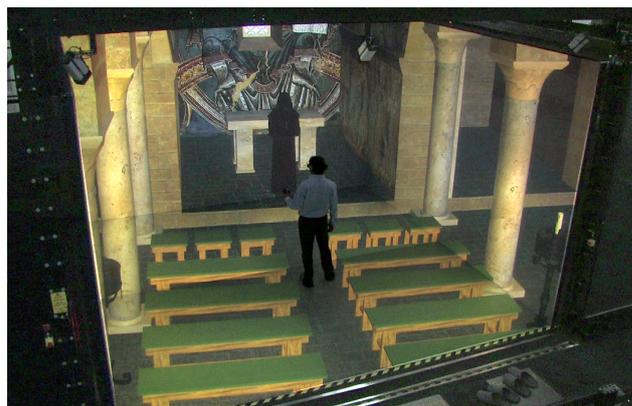


Figure 1: Interactive room acoustics demo of a medieval church in the aixCAVE virtual environment at RWTH Aachen University.

real-time for a limited low-frequency range (e.g. finite-difference time-domain (FDTD) and derivatives using graphic processors [5, 6]). Due to their time and memory complexity they do not provide a suitable solution for the full audible frequency range (20-20.000 Hz).

Parallelism became a key concept in achieving real-time capability of the simulations. Wave-based solvers inhere distinct data inter-dependencies. Domain decomposition approaches for numerical methods require the exchange of sound field variables (e.g. pressure) on the boundaries of spatial sub domains. These 'updates' have to be performed for every simulation step, creating excessive amounts of communication. A shared memory access with high throughput and low latency (e.g. video RAM) is preferable here. Typical GA computations, such as image sources and ray tracing (RT) are largely independent, thus permitting isolated computation of fractions of a simulation result. Audibility checks of image source (IS) and the tracing of rays can be easily distributed, i.e. on several cores of a single computer unit. Only the assembly of the resulting filter (e.g. based on a list of audible image sources or energy histograms) depends on all fractions of the simulation. The independence of data and partial steps in the computation make GA algorithms particularly suitable for distributed simulation on computer clusters. Given that a single shared-memory machine cannot provide the necessary performance, the use of multiple units is a promising approach to increase the required computation power. Strategies to use PC clusters for efficient *real-time* room acoustics simulations are examined in this paper.

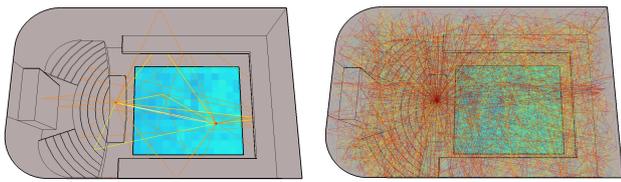


Figure 2: Visualization of computed sound propagation paths in a concert hall model, showing image source traces (left) and simplified ray tracing (right, with very low number of rays).

## 2. ROOM ACOUSTICS SIMULATION

Today's widely used acoustics simulation programs mostly base on the principles of classical GA. Already in the 1960s, Schroeder and Krokstad applied the ray tracing technique to predict acoustics properties of specific room shapes [7, 8]. Since then, GA algorithms have continuously advanced, so that modern software can handle complex wave effects, including scattering [9] and diffraction [10, 11]. Benchmark tests certified GA simulation methods as a valuable tool for the design and prediction of room acoustics [12, 13, 14, 15]. Major uncertainties in current simulations are rather reasoned by the input data than to the pure algorithms [15].

### Simulation methods

State-of-the-art implementations use hybrid combinations of both ray tracing and image sources. Vorländer [16] and Vian et al. [17] presented the basis for the cone-, beam- and pyramid-tracing dialects, e.g. [2, 18], by showing that forward tracing is a very efficient method for the modeling of early reflections in a room impulse response (RIR). During the 1990s it was shown that GA cannot solely be based on specular reflections [12] which lead to the integration of scattering with activities on the prediction, measurement and standardization of scattering and diffusion coefficients of corrugated surfaces [19]. Advances in the binaural technology enabled the incorporation of spatial attributes to room impulse responses. The key equation of the contribution of one room reflection,  $\underline{H}_j$ , is given in spectral domain with

$$\underline{H}_j = \frac{e^{-j\omega t_j}}{ct_j} \cdot \underline{H}_{air}(ct_j) \cdot \underline{H}_{Src}(\theta, \phi) \cdot \underline{H}_{Rec}(\vartheta, \varphi) \cdot \prod_{i=1}^{n_j} R_i \quad (1)$$

where  $t_j$  is the reflection's delay,  $\omega t_j$  the phase,  $1/ct_j$  the distance law of spherical waves,  $\underline{H}_{Src}$  the source directivity in source coordinates,  $\underline{H}_{air}$  the loss pass of air attenuation,  $R_i$  the reflection factors of the walls involved, and  $\underline{H}_{Rec}$  the head-related transfer function of the sound incidence at a specified head orientation. The complete binaural RIR is composed of the direct sound and the sum of all reflections. Audible results are obtained by convolution of the binaural RIR filter with an anechoic source signal.

The computation of the filters is implemented using the simulation software Room Acoustics for Virtual Environments (RAVEN) [20]. The software features a hybrid image source and ray tracing engine that can operate under real-time conditions. This was achieved by increasing the simulation speed using concepts of spatial subdivision [21].

### Spatial data structures and room models

The performance of GA methods is strongly governed by intersection tests, which consume the major part of the runtime. Spatial search concepts (binary space partitioning (BSP), bounding volume hierarchies (BVH) or spatial hashing (SH)) are commonly used techniques to accelerate the intersection testing. In case of interactive geometry modifications, the spatial data structures have to be updated before a new simulation. Performance analysis showed that the most efficient way of reacting to a geometry update is the usage of a hybrid approach including two different spatial data structures. In direct comparison, BSP trees allow faster intersection tests than SH hashing [22]. However, the update of a BSP tree needs significantly more time compared to a spatial hash table. As the early reflections involve much fewer intersection tests compared to the late diffuse reflections, they can be effectively accelerated by using SH [23] for the intersection tests during IS audibility tests, in particular if the geometry is subject to frequent modifications. For the late reflections in the RIR, our RT algorithm uses BSP [21] to reduce the number of intersection tests. This reduces the total computation time of the RT process significantly, although there is a substantial delay due to the necessary reconstruction of the BSP tree. For details of the implementation the reader is referred to [22].

In general for any of the spatial data structures there is a high dependency of the performance on the number of polygons in the scene. It is known that the 3D model for acoustics calculations can (and must) be modeled with much less details than the optical representation, as shown in Figure 3. However, it is important to define proper characteristics for the surface parameters (frequency dependent absorption and scattering coefficients). In prior investigations it was analyzed how much structural detail is needed to be included in the acoustic model. The findings indicated that for typical rooms details smaller than 0.5 m can be neglected [24].

### Sound transmission and diffraction

A building acoustics module in the RAVEN framework allows the simulation and auralization of sound transmission through structures [25]. Any pair of polygons in the scene can act as an bidirectional surface source and surface receiver pair. Using appropriate transfer functions the structure born sound propagation can be accounted for, e.g. through walls, floors and doors [26].

In case of coupled volumes with small apertures, but also for large objects in rooms or generally in outdoor scenes, the sound diffraction around edges is an important wave effect and must be taken into account. The RAVEN library implements edge diffraction for the IS algorithm as well as for the RT technique. Details on the implementation are published in [10]. Especially for these very complex and computationally expensive operations, the possibilities of distributed computing play an important role, so that these effects can be rendered in real-time, too.

### Simulation processing

The RAVEN software has a modular architecture. It can be used by a graphical user interface, but also as a pure network service. For the distributed real-time sound field rendering, several *simulation nodes* were configured and controlled via network. Each RAVEN node is controlled using a remote interface. A simulation is remotely executed by handing over a simulation task to a simulation node. The received simulation task is then locally translated

into an execution of a simulation algorithm. An initial simulation will consist of a) the generation of an image source cloud for the current source position, b) an audibility test to filter audible image sources for the current receiver position, and c) ray tracing for the current source and receiver positions. During the interactive simulation, only steps b) and c) are continuously executed. Each simulation runs asynchronously, blocking the node in the mean time. When the computation is finished, it returns either a set of (potential/audible) image sources or histograms containing the spectral energy envelopes of the late decay. This intermediate result is then passed to a *filter controller*, which constructs time-domain room impulse responses. The obtained impulse responses for early (IS) and late (RT) parts are superposed in time domain. Each simulation task contains a snapshot of the virtual scene. Changes to the prior state, such as translated sound sources or listeners, are efficiently implemented by translation of the image source cloud using pre-calculated translation matrices. Therefore, no reconstruction of the image source tree is necessary. For ray tracing it must be mentioned that re-calculating the spectral late decay is only necessary if the position of either the source or the receiver is subject to significant translation. In typical rooms the late part of the impulse response is diffuse and homogeneous and will only be updated if the source or receiver moved more than 1 m away from the last simulated positions. In contrast the direct sound must already be updated if a source or the listener moves a few centimeters or rotates by a few degrees. This part of the impulse response is crucial for the localization of sources and is in full attention of the listener. However, this operation is very fast as only a test for a line of sight must be performed on the scene geometry and a short filter of source directivity, receiver directivity (e.g. HRTF) and air attenuation is assembled.

### Shared-memory parallelization

Operations with high computation time consumption such as the generation of the image source trees, the audibility tests of image sources and the tracing of rays are parallelized at node-level using *OpenMP* [27]. For IS the parallelization is very effective in the construction of the IS cloud. Here, higher IS orders dependent on lower order image sources, but the IS tree branches of all first order sources can be constructed in parallel without any interdependencies. Especially the ray tracing algorithm scales nearly perfectly linear to the number of locally available cores and has no data interdependencies. In typical rooms, 10k-100k rays are to be shot, each with similar computation time and completely independent of each other. Therefore, all available processor cores are set up

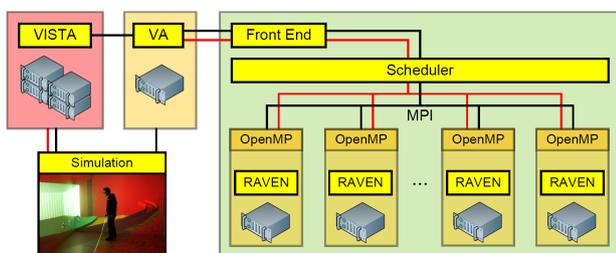


Figure 4: Hybrid inter-node and intra-node level parallelism inside the RAVEN simulation framework.

Simulation	Computation time (single node)
Direct sound (DS)	3.4 ms
Early reflections (ER)	23 ms
Late reverberation (LR)	2.04 s

Table 1: Average computation times for different simulation types computed on a single cluster node (shared-memory parallelization). Each runtime corresponds to the generation of a binaural filter (DS, ER or LR) for one source↔listener pair.

with one single thread on each core that traces one single ray at a time. By sorting the angles of the launched rays to spatially coherent directions the utilization of the CPU’s low-level data caching is improved.

In RT different frequency bands are usually simulated separately (mostly in octave resolution) and are executed in sequential order. A parallelization across frequency bands has the disadvantage that higher frequency bands have much shorter computation times than low frequencies due to typical material and air absorption properties. In this case the workload is not equally balanced across the cores/threads leading to unnecessary overhead. Given that only few simulations are computed in parallel, the capabilities of a compute cluster, featuring a multitude of nodes cannot be fully utilized. In order to achieve the best performance, the multitude of rays must be distributed across the nodes. The energy histograms of each partial result can simply be superposed and are identical to a simulation with the summed number of rays on a single machine. Locally on each node, the room acoustics simulation should employ shared memory parallelization due to their excellent scaling capabilities, instead of running multiple concurrent simulators on a multi-processor node. The hybrid inter-node and intra-node level parallelization concept considered in this paper is illustrated in figure 4 .

### Test scene

As a representative test case, a virtual scene of the medieval spanish church *San Juan de Baños* has been selected to benchmark the performance of distributed simulations of room acoustics. The virtual and acoustic model are depicted in Figure 3. Image sources are calculated up to the second order. Ray tracing has been performed using a target filter length of 2.8 seconds with 10.000 particles for each of the 10 frequency bands in octave resolution from 31 Hz to 16 kHz, a time resolution of 6 ms and a detection sphere radius of 0.5 m. For accelerating simulation time, the BSP method has been used. Furthermore, the computations were parallelized using 24 threads (compare section 5 for further details). The resulting computation times on a *single computer* node of the cluster are listed in table 1. The technical details of the hardware are outlined in section 5.

## 3. AUDIO RENDERING

Audio rendering is considered the process of transforming the monaural source signals into appropriate listener signals, which in our case are binaural. The foundation is the description of the virtual scene, referred to as the *scene state*. It describes all stationary attributes of acoustically relevant entities in the virtual scene. These include positions, orientations, velocities of objects and specific properties, like directivity of sources, head-related impulse

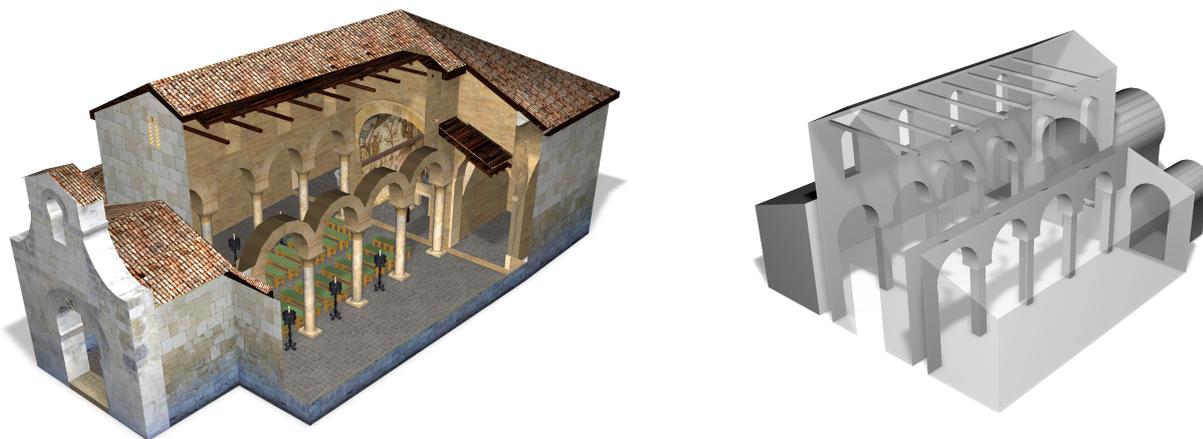


Figure 3: Visual and acoustics model of San Juan de Baños chapel. The acoustic model (right) uses 1.300 polygons with an estimated volume of  $756\text{ m}^3$  and surface of  $776\text{ m}^2$ . The predicted reverberation time (Sabine) is 3 seconds.

response (HRIR) datasets of listeners. Moreover, the properties of the propagation medium (static pressure, temperature, humidity), the scene geometry (polygons), acoustic parameters of surfaces (absorption, scattering), etc. Any modification, for instance pose changes acquired from motion tracking that control the movement of the listener, result in a state change. User interaction may additionally alter positions, orientations or even change internal states of objects (switching material of a wall, interacting with a machine that results in a change of emitted sound). Also, autonomous objects (e.g. avatars or moving vehicles) can be controlled independently by a Virtual Reality (VR) software. Given a scene change (update), the auralization software needs to rapidly update the acoustic stimuli for the user, with respect to the new situation.

Most auralization methods describe the virtual scene only in an instant of time disregarding history data—all computation is based on the current state from the viewpoint of the listener. This simplification is acceptable as long as sound propagation times are negligible small, i.e. in small spaces. For the simulation of room acoustics and outdoor scenarios it is helpful to preserve access to past states of the scene along a time history. This allows to provide for acoustic effects that occur with respect to finite medium propagation (i.e. delayed arrival of acoustic events, Doppler shifts at sound source) and, in addition, enables algorithms that can reuse results, i.e. a ray tracing simulation for a specific position. This becomes even more relevant, when several hundred simulation jobs are executed in parallel, yet the delivery of results is not fast enough to cover a specified amount of spatial resolution. In this case, the contribution of an obsolete simulation result may lead to a better auralization of the current scene state. In the following we present a scene data structure that fulfills this requirements and suits a real-time computation.

### Scene description

A single *scene state* of a virtual environment describes a static snapshot of all relevant entities and their properties. It represents the top-level (global scene) and consists of multiple sub-states (e.g. states of sources and listeners). These on their own are assembled by further lower level states (e.g. position or orientation). States form a hierarchy and are implemented as a tree. Each time the virtual scene is altered, a new scene state is derived from the cur-

rent state (copy). Then only the affected parts (sub trees or leafs) actually modified in the newly derived state. It holds the differential information to the prior state. Unchanged elements are simply referenced, keeping the memory footprint compact. A state is kept in memory, until the last reference is removed. This scheme makes it very easy to compare different states and to track back parameters over time. Moreover, the data structure allows to handle asynchronous resource allocations and deallocations. In detail, the creation of states is taking place in the control-flow of the caller (external user), while destruction of states is decided in the context of the real-time audio rendering (in case a state becomes obsolete). Decoupling of these actions is realized using object pools and reference counters. Thereby lock-free data structures are employed to avoid the problem of blocking in the time-critical audio processing.

Update rates in VR applications are often determined by connected input devices (e.g. motion tracking with 120 Hz). For the signal processing some parameters, like positions of objects, are required with even higher rates, i.e. if the audio streaming process updates its elements with 340 frames per second (128 samples at 44.100 Hz). Parameter interpolation [28] can be easily added on top of the scene description. In particular, positions of fast moving objects are real-time predicted and interpolated using a motion model, producing a quasi-continuous trajectory. Attributes can be queried, even at audio sampling rates, if necessary.

### Signal processing

Signal processing is based on the abstraction of propagation of sound between a single source and a single listener. The superposition of all source $\leftrightarrow$ listener pairs represent the entire virtual situation, and their relationship is illustrated in figure 5. The core element within the processing chain is represented by the convolution module that convolves the source signal with the simulated RIR. For the sake of high-quality auralization that minimizes acoustic artifacts, some aspects like propagation delay and high resolution distance-dependent attenuation caused by spreading loss have to be realized independently. In the following the processing is explained in more detail.

Common simulation methods encode the traveling time  $T = r/c$  of the sound waves into the impulse response, as it is the case

for a measured impulse response. The direct sound impulse is located at the offset  $T \cdot f_S$  (samples). Prior filter coefficients are zero, realizing a delay that corresponds to the traveling time. When it comes to real-time auralization, this concept has several disadvantages: firstly, leading zeros increase the computational effort of the filtering procedure, and secondly, changes in distance can only be updated with each filter exchange. Also they manifest in time-shifts of parts of the impulse response, which is inconvenient for the fast convolution. Simple implementations based on simple switching of filter coefficients or applying additional cross fading of the output stream may have negative effects: they usually produce audible artifacts (clicks, comb-filters). A superior approach is to separate propagation delays from the RIRs by the use of a variable delay-line (VDL). Based on the continuous-time motion model mentioned earlier the VDL parameters can be continuously adapted (per sample) and hence comb filter artifacts can be avoided. VDL can be realized with a low computational complexity (i.e. with fractional delays or polynomial interpolators).

For the use with delay-lines, the initial traveling time is removed from the simulated room impulse responses (see figure 6). The direct sound impulse (red) is always located at the beginning (accounting for a fixed pre-ring offset in the case of non-minimum-phase HRIRs). The other parts, ER (green) and LR (blue), reside in more or less fixed ranges of filter coefficients. Moreover, the impulse responses are normalized in magnitude, so that the amplitude of the direct sound part is 1 (amplification of the complete impulse response by  $r$ ). This approach simplifies the time-varying finite impulse response (FIR) filtering and allows the effective use of output crossfading techniques [28] to prevent audible artifacts for the filter changes.

The filtering with the room impulse response is implemented using non-uniformly partitioned convolution in the frequency-domain [29] [30] [31]. Our convolution engine performs Overlap-Save (OLS) using the Fast Fourier Transform (FFT) and is specialized for binaural processing (two output channels). The non-uniform filter partition is optimized [32] [33] with respect to the filter subdivision and desired update rates for these parts of the room impulse response (DS, ER, LR). A smooth exchange of filters is accomplished by computing the convolution result streams for both filters (old and new) during exchange and cross-fading the output samples in time-domain.

The second to last step is reapplying the previously removed gains for spreading losses to the output signals. These are now applied *per sample*, allowing to smoothly adapt the gain over the samples of each frame. Updating them *per filter* results in steps in the gain envelope, which cause clearly audible clicks due to inconsistency of the signal wave form. Again, based on the motion model, the source $\leftrightarrow$ listener distances are available for the beginning and end time of each audio frame and then a gain envelope is interpolated. The process is illustrated at an example trajectory in figure 7. For frame rates  $>300$  Hz this piece-wise linear gain curve turns out to be sufficient. Finally, the left/right ear signals of each sound source are added up (superposition at the listeners ears), resulting in a single binaural stimulus for each listener.

In sense of a more physically-correct auralization, the time-varying propagation delays should be considered individually for each image source (specular reflection), too. The relative shifts in time-delays for the direct sound and early reflections are more or less independent. Each image source up to a given order, could be implemented using an independent VDL and convolution of HRIR and reflection filter. But this would increase the computational ef-

fort of the auralization by several magnitudes, limiting the possible number of sound sources and listeners. Although it remains an open scientific question, if this expense is necessary with respect to audibility.

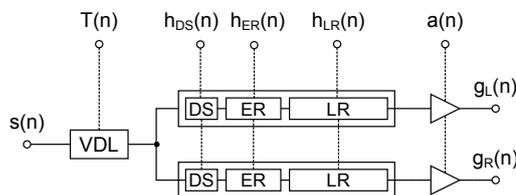


Figure 5: Audio rendering of a source $\leftrightarrow$ listener sound path. The monaural source signal  $s(n)$  is transformed into a binaural room acoustic stimulus  $g_{LR}(n)$ . Individual parts of the impulse response are updated asynchronously. Propagation delays and distance gains are updated per sample and realized apart from the RIR.

### Time-varying scenes

Due to the block-based signal processing, the maximum filter update rate is determined by the frame rate of the audio stream. Therefore, it is reasonable to check for filter updates and trigger them within the context (thread) of the audio processing. Embedded into this processing call, the auralization module performs parameter changes that require time-critical update rates (like distance gain and medium propagation delay). The actual filter calculation is realized in separate threads. Validation for updates and the simulation are decoupled using update requests, called *tasks*, for certain parts of the signal processing chain (i.e. the reconstruction of ER and LR). Tasks are lightweight, allowing to issue (create) them within the audio context, without diminishing the computation time. Each task includes all the necessary information for performing a room acoustic simulation (DS, ER or LR). Together with a reference tag and a unique identifier, each task is then handed over to a *scheduler*, which runs asynchronously, in a dedicated thread. It is responsible for the further handling of the request, its simulation and the final application of the computed updates. Details of the scheduler are explained in the succeeding section 4. Eventually, simulation results are received in yet another decoupled thread that assembles the entire RIR and updates relevant filter partitions in the convolution process. For the direct-sound part, this is a HRIR signal, the audibility status (audible, inaudible) and optionally a signal with diffracted components of the direct sound. The result of an image source computation is a list of audible image sources, which are combined into an image source filter. This filter is inserted into the given RIR of the according source $\leftrightarrow$ listener pair and the covered range of filter coefficients is updated in the convolution. Ray tracing results (late reverberation filters) are handled in a similar way.

On many-core shared-memory systems several simultaneous simulator threads are created and assigned to specific types of tasks. The distribution of the computation (signal processing, room acoustic simulation, etc.) to the available computation units (processors or cores) is a distinct procedure that relies on the available hardware. The decision of calculating and updating sections of the transfer path depends on the number of source $\leftrightarrow$ listener pairs, available cores, computational complexity, etc. Direct sound tasks are updated instantaneously within the context of an audio frame.

All other tasks are handed over to the scheduler, which decides if the task is worth an update and then assigns its computation to an available core. Predefined numbers of cores are assigned to the different computations (real-time convolution, image source and ray tracing simulations), with respect to the underlying hardware and scene complexity. Thereby the real-time convolution has the highest priority. Dropouts must be strictly avoided. The remaining cores are assigned to room acoustics simulations. Usually, a reasonable compromise between an acceptable image source computation time and ray tracing update rate has to be chosen.

#### 4. DISTRIBUTED SIMULATION

The presented concept of distributed room acoustic simulation is realized in fashion of a client-service structure (see fig. 4). The audio renderer, as a client, generates an arbitrary amount of tasks and transfers those to scheduler. The scheduler acts as a server, planning and organizing the received simulation requests and passing on tasks to the worker nodes in the cluster. In shared-memory systems, data is exchanged between threads by the use of shared variables or pointers. Convenient tools for compiler-generated loop-level parallelization are available, for instance *OpenMP* [27]. In contrast, distributed memory systems lack a common memory which all nodes can collectively access. These systems are typically programming using the paradigm of *message passing*, e.g. using *MPI* [34]. All data has to be transmitted in-between nodes using *messages* that are delivered by communication channels. *MPI* introduces an abstraction layer, that hides underlying hardware structures like the machine type of each node and network communication channels, making deployment, execution and expansion easily manageable. Most modern clusters are assembled from multi-/many-core computers (distributed shared-memory systems). Therefore, both parallelization concepts (e.g. *MPI* + *OpenMP*) are often combined, known as *hybrid parallelization*. This is as well applied in the presented approach.

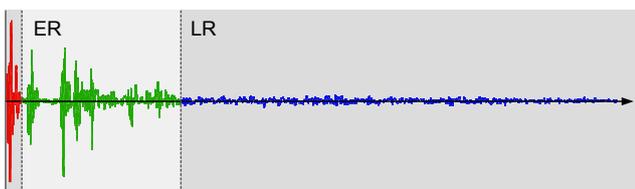


Figure 6: Parts of simulated room impulse responses (DS, ER, LR) are mapped to individual filter segments in the non-uniformly partitioned convolution.

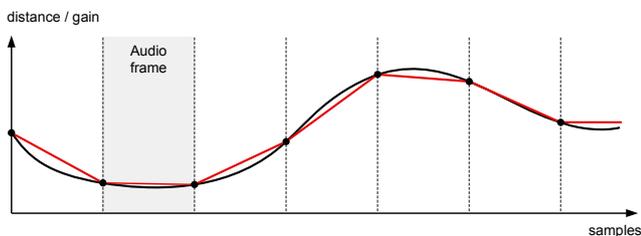


Figure 7: Interpolation of  $1/r$  gains for spreading losses. (Black: Continuous distance function obtained from motion model, Red: Linear-interpolated per sample, used as VDL input)

#### Method

The problem of distributed room acoustic simulations particularly suit the hybrid parallelization on computer clusters because it requires minimal network communication. Given that a simulation task is scheduled for computation on a vacant worker node, only the task data (a few kilobytes) need to be transmitted. The simulation node can adapt to any given virtual scene and requested simulation type. During simulation, no further communication with any other node is necessary. Each simulation node executes simulations sequentially. Once a task has been assigned to it, the scheduler considers the worker node busy and does not send any further tasks. On completion of a task, the worker node communicates a result message to the master node containing the simulation results, i.e. the binaural filter parts. In order to decrease the package size, leading and trailing zeros of the impulse response are stripped prior to transmission. After successful transmission, the worker node receives new tasks.

#### Scheduling

During a running *real-time* auralization, a vast amount of simulation tasks is generated in the context of the audio rendering, exceeding the available computing power of the cluster power by far (i.e. on every scene state change for each source↔listener pair). But this design is intended. It is the responsibility of the scheduling instance to cope with the flood of simulation tasks without compromising the rendering process. The large number of updates allows the scheduler to decide which update is meaningful and which parts are given the highest priority. In the implemented approach, a two-phase scheduling procedure is applied to prevent high network load and to maintain a lightweight handover of tasks between rendering and scheduling. In order to achieve this, both the client-side and the server-side, carry out a replacement strategy with the same procedure, thus preventing transmission of tasks that most likely would have been discarded in the first place.

This paper considers a simple scheduling strategy by means of priority and replacement. An existing task is considered outdated, if an incoming task with same reference exhibits a novel time stamp, and will therefore be discarded. Tasks are treated equally, i.e. first task is served first and the list of pending tasks is polled cyclically.

#### 5. BENCHMARKS

In order to evaluate the performance of the proposed distributed room acoustic simulation, benchmarks were conducted on a compute cluster. The throughput of simulations was measured under real-time constraints with different numbers of cluster nodes. All three different simulation types (DS, ER and LR) were benchmarked individually, with a variation of the total number of cluster nodes between 1 and 24. For the tests the scheduler was flooded with tasks from the client side, simulating real-time circumstances, where vast amounts of tasks are issued. This assured that the system was always stressed to the maximum. The scheduler then organized and planned the execution of tasks and distributed them to the available compute nodes, where they were simulated. On client side, the finished tasks collected, analyzed and the achieved overall update rate  $f_{max}$  [filter updates per second] was counted. This measure is first of all unrelated to a number of sound source and listeners. The compute resources can be shared accordingly,

when a virtual scene contains  $M$  sources and  $N$  listeners. Given the assumption that the simulation times are mostly independent of the individual source $\leftrightarrow$ listener pairs (equal sharing), the achievable update rate per sound path is approximately  $f_{\max}/(M \cdot N)$ .

### Hardware

The benchmarks were conducted on the computing infrastructure around the *aixCAVE* CAVE-like virtual reality system (fig. 1) at RWTH Aachen University. This display system combines five-sided high-resolution back-projection screen setup with active shutter technique for stereoscopic vision. An optical tracking system captures motion of the user and of input devices thus providing the possibility to interact with the virtual scene. A dedicated acoustics sub system (auralization server) receives control information and scene data over a network interface from the VR application. This system is linked to a high performance computer cluster. Aurulation server and compute cluster communicate via a custom designed TCP/IP network protocol over Gigabit-Ethernet. Cluster nodes communicate with each other using the Message Passing Interface (MPI) in form of the OpenMPI library version 1.6.4 (*multi-threaded*). Each cluster machine consists of an Intel Xeon X5650 (*Westmere*) processor with 12 cores running at 2.7 GHz and 24 GB DDR3 memory available. For the benchmark testing a division of 24 similar nodes and an additional master node of same specifications has been used exclusively. The nodes are running Scientific Linux version 6.4 as operating system with kernel version 2.6.32 (*x86\_64*). The software and test suite has been implemented in the C++ programming language and was compiled and linked with g++ version 4.4.7.

### Results

Figure 8 shows the achieved update rates as a function of the number of used cluster nodes for the chapel scene. The achievable maximum update rate  $f_{\max}$  is affected by two aspects: (a) the runtime of the simulation on each node and (b) the data transmission times for inter-node communications. The computation times for the ray tracing on each node are in the range of 2 seconds (table 1). An almost perfect linear scaling (double the number of compute nodes  $\leftrightarrow$  double the filter update rate) can be observed. This indicates that data transmission times are comparably small. The image source computation times are magnitudes shorter. On a single node they are in the range of 20 ms. Here, the scaling is nearly linear as well, until a number of twelve nodes. But beyond that, the inter-node communication becomes a bottleneck and starts to diminish the performance benefits by additional nodes. For 20 nodes and above the limitation becomes clear. The smaller the computation time of a task is, the more significant becomes the necessary data communication for it. Direct sound audibility checks compute in less than 4 ms (table 1). For them, the scaling is far from optimal. When twelve or more nodes are employed, the communication overhead becomes a serious bottleneck and no significant improvements can be achieved by using further nodes.

## 6. CONCLUSIONS

The results show a clear benefit from a distributed computation. Filter update rates can be significantly increased by several magnitudes over a simulation which runs on a single (multi-core) machine only. It can be concluded, that mostly the ER and LR tasks

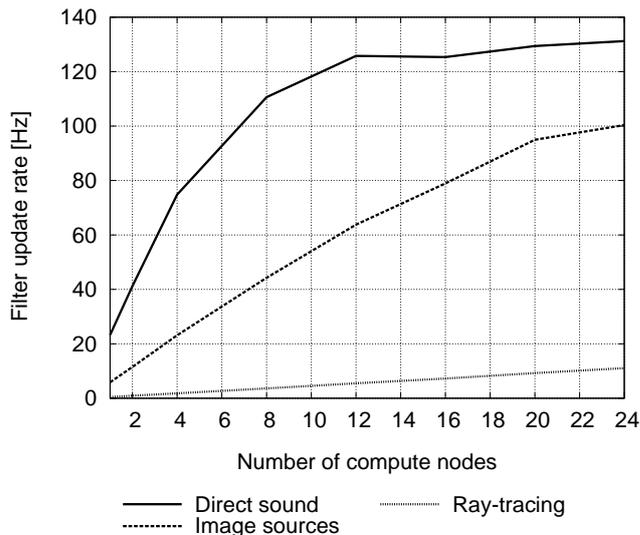


Figure 8: Maximum filter update rates  $f_{\max}(N)$  for individual parts (DS, ER, LR) over the number of cluster nodes  $N$ .

gain profits from the cluster implementation for distributed simulation. Short-timed computations in the range of milliseconds, like simple DS audibility checks, are too much affected by the required data transmission. It is suggested to avoid the distribution of these tasks and compute them on the target system directly. By using more compute nodes, the update rates, at least of the ray tracing, are likely to be even higher.

## 7. OUTLOOK

Although the achieved filter update rates are high, the method does not minimize the latency of the computation process itself. The system could achieve  $> 10$  ray tracing simulations per second, but still the time from task creation to finish was similar to that on a single machine (table 1). That is due to the fact that each task was only parallelized on each node and could not exploit the whole cluster. In order to speed up the single tasks and thus reduce the latencies, single IS or RT simulations must be further decomposed and distributed to multiple nodes (e.g. partitioning of image sources, different ray-packages).

For conceptually simple scenes the cluster parallelization might outreach the necessary update rates with respect to the human perception. Which rates are reasonable for the individual parts of the simulation remains to be researched. We see target applications of the proposed technique in the real-time auralization comprehensive urban noise scenarios (large numbers of sound sources and listeners) and room acoustics planning and prototyping (interactive change of materials, A-B comparisons).

## 8. ACKNOWLEDGMENTS

The authors thank Alexander Diaz Chyla for providing the 3D models and his work for creating the demo scene. Special thanks go to Dominik Rausch of the RWTH Virtual Reality Group for supporting this project and helping with the implementation.

## 9. REFERENCES

- [1] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Creating Interactive Virtual Acoustic Environments," *Journal of the Audio Engineering Society*, vol. 47, no. 9, 1999.
- [2] T. A. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, M. Sondhi, J. E. West, G. Pingal, P. Min, and A. Ngan, "A beam tracing method for interactive architectural acoustics," *Journal of the Acoustical Society of America*, vol. 115, 2004.
- [3] T. Lentz, D. Schröder, M. Vorländer, and I. Assenmacher, "Virtual Reality System with Integrated Sound Field Simulation and Reproduction," *EURASIP journal on advances in signal processing*, vol. 2007, 2007.
- [4] D. Schröder, F. Wefers, S. Pelzer, D. S. Rausch, M. Vorländer, and T. Kuhlen, "Virtual Reality System at RWTH Aachen University," in *20th International Congress on Acoustics (ICA 2010)*, Sydney, Australia, 2010.
- [5] L. Savioja, "Real-time 3D finite-difference time-domain simulation of low-and mid-frequency room acoustics," in *Conference on Digital Audio Effects (DaFX-10)*, Graz, Austria, 2010.
- [6] N. Raghuvanshi, R. Narain, and M.C. Lin, "Efficient and Accurate Sound Propagation Using Adaptive Rectangular Decomposition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 5, 2009.
- [7] M. Schroeder and B. Atal, "Computer simulation of sound transmission in rooms," *Proceedings of the IEEE*, vol. 51, pp. 536 – 537, 1963.
- [8] S. S. A. Krokstad and S. Sorsdal, "Calculating the acoustical room response by the use of a ray tracing technique," *Journal of Sound and Vibration*, vol. 8, pp. 118–125, 1968.
- [9] D. Schröder and A. Pohl, "Modeling (Non-)uniform scattering distributions in geometrical acoustics," *Proceedings of Meetings on Acoustics*, vol. 19, no. 1, 2013.
- [10] D. Schröder and A. Pohl, "Real-time hybrid simulation method including edge diffraction," *EAA Auralization Symposium, Espoo, Finland*, 2009.
- [11] D. Schröder, P. Svensson, U. Stephenson, A. Pohl, and M. Vorländer, "On the accuracy of edge diffraction simulation methods in geometrical acoustics," in *Proc. of Inter-Noise 2012, New York*, 2012.
- [12] M. Vorländer, "International Round Robin on room acoustical computer simulations," in *15th International Congress on Acoustics*, Trondheim, Norway, 1995.
- [13] I. Bork, "A comparison of room simulation software - the 2nd round robin on room acoustical computer simulation," *Acustica - Acta Acustica*, vol. 86, 2000.
- [14] I. Bork, "Report on the 3rd round robin on room acoustical computer simulation - part ii: Calculations," *Acta Acustica united with Acustica*, vol. 91, 2005.
- [15] S. Pelzer, M. Aretz, and M. Vorländer, "Quality assessment of room acoustic simulation tools by comparing binaural measurements and simulations in an optimized test scenario," *Acta acustica united with Acustica*, vol. 97, no. S1, 2011.
- [16] M. Vorländer, "Simulation of the transient and steady state sound propagation in rooms using a new combined sound particle - image source algorithm," *Journal of the Acoustical Society of America*, vol. 86, 1989.
- [17] J. Vian and D. van Maercke, "Calculation of the room impulse response using a ray-tracing method," *Proceedings of the ICA Symposium on Acoustics and Theatre Planning for the Performing Arts, Vancouver, Canada*, 1986.
- [18] U. Stephenson, "Quantized pyramidal beam tracing - a new algorithm for room acoustics and noise immission prognosis," *ACTA ACUSTICA united with ACUSTICA*, vol. 82, 1996.
- [19] P. D. Antonio J. J. Embrechts J. Y. Jeon E. Mommertz T. J. Cox, B.-I. L. Dalenbäck and M. Vorländer, "A tutorial on scattering and diffusion coefficients for room acoustic surfaces," *Acta Acustica united with ACUSTICA*, vol. 92, 2006.
- [20] D. Schröder, *Physically Based Real-time Auralization of Interactive Virtual Environments*, Ph.D. thesis, RWTH Aachen University, 2011.
- [21] D. Schröder and T. Lentz, "Real-time processing of image sources using binary space partitioning," *Journal of the Audio Engineering Society*, vol. 54, no. 7/8, 2006.
- [22] S. Pelzer, L. Aspöck, M. Vorländer, and D. Schröder, "Interactive real-time simulation and auralization for modifiable rooms," *Proceeding of the International Symposium on Room Acoustics, Toronto, Canada*, 2013.
- [23] A. Ryba D. Schröder and M. Vorländer, "Spatial data structures for dynamic acoustic virtual reality," *Proceedings of the 20th International Congress on Acoustics (ICA)*, Sydney, Australia, 2010.
- [24] S. Pelzer and M. Vorländer, "Frequency- and time-dependent geometry for real-time auralizations," *Proceedings of 20th International Congress on Acoustics*, 2010.
- [25] F. Wefers and D. Schröder, "Real-time auralization of coupled rooms," Espoo, Finland, 2009.
- [26] R. Thaden, *Auralisation in Building Acoustics*, Ph.D. thesis, RWTH Aachen University, Aachen, Germany, 2005.
- [27] OpenMP, "Application program interface specifications version 4.0," API, July 2013.
- [28] J.-M. Jot, V. Larcher, and O. Warusfel, "Digital Signal Processing Issues in the Context of Binaural and Transaural Stereophony," in *Audio Engineering Society Convention 98*, Feb 1995.
- [29] W. G. Gardner, "Efficient convolution without input-output delay," *Journal of the Audio Engineering Society*, vol. 43, 1995.
- [30] G. P. M. Egelmeers and P. C. W. Sommen, "A New Method for Efficient Convolution in Frequency Domain by Nonuniform Partitioning for Adaptive Filtering," *IEEE Transactions on Signal Processing*, vol. vol 44, 1996.
- [31] E. Battenberg and R. Avizienis, "Implementing Real-Time Partitioned Convolution Algorithms on Conventional Operating Systems," in *Digital Audio Effects Conference (DAFx)*, 2011.
- [32] G. Guillemo, "Optimal filter partition for efficient convolution with short input/output delay," in *Audio Engineering Society, Convention Paper 5660*, 2002.
- [33] F. Wefers and M. Vorländer, "Optimal filter partitions for non-uniformly partitioned convolution," in *45th AES Conference on Applications of Time-Frequency Processing in Audio, Helsinki, Finland*, 2012.
- [34] MPI, "A message passing interface standard version 2," API.