CORRECT AND FAST IMPULSE RESPONSE CALCULATION AS A MUST FOR INTELLIGIBILITY PREDICTION AND AURALISATION

W Ahnert AFMG Technologies GmbH, Berlin, Germany
S Feistel AFMG Technologies GmbH, Berlin, Germany
A Miron AFMG Technologies GmbH, Berlin, Germany

1 INTRODUCTION

To calculate correct acoustic measures we need correct simulation results with any prediction tools, comparable to measured correct binaural impulse responses.
In the beginning 90ies the computer power has been weak, so only pure mirror image or straight ray tracing model have been used to calculate a sequence of reflections mainly for the first 200...300ms[1]. The so-called reverberation tail was afterwards added by statistical assumptions and based on partly incorrect Sabine or Eyring decay data.
The method to derive the reflection sequence for the first part of an impulse response has been changed and different hybrid procedures are now in use, but quite often the combination with a statistical reverb tail may be observed until now.
In the beginning of this century the computer power allowed to calculate full impulse responses without to take into consideration any statistical completion. This way we introduced in EASE4.0 a new algorithm, called AURA to calculate even binaural impulse responses in full length[2, 3, 4]. However, there are still general problems associated with this approach. On one hand, a full-length room acoustic simulation for a mid-size space requires some calculation time, usually several hours on modern PCs. On the other hand, historically coming from a strong numerical and scientific background, room acoustic simulations provide many parameters to adjust, which in turn means many degrees of freedom for the user.

In this work we present some new improvements to accelerate the calculation in time to obtain realistic impulse responses as a base for intelligibility estimations and to derive real auralised files for sound reproduction.

2 CALCULATION OF IMPULSE RESPONSES AS A BASE FOR REALISTIC POSTPROCESSING

2.1 New Intersection Algorithms

For room acoustic models brute-force ray tracing (testing all triangles for intersection) is often impractical, since computation time scales linearly with the number of triangles. Improved performance is obtained by structuring triangle data such that each ray is tested for intersection only with a subset of triangles. Current methods are based on two main strategies: hierarchical bounding volumes (HBV)[5] and space partitioning [6, 7]. In the former case, a hierarchy of simple bounding volumes (such as spheres) is constructed, where a particular volume may include either a number of smaller child-volumes or actual triangles. A ray is tested for intersection starting at the top of the hierarchy, such that a particular child-volume is only tested if the parent was hit. The cost of ray-bounding volume intersection is small, and the resulting computation scaling with the number of triangles is approximately logarithmic. In space partitioning schemes, the physical space where the triangles reside is partitioned into smaller cells or so-called voxels. Rays are followed through

adjacent voxels and tested only against triangles pertaining to those voxels. The partitioning may be uniform or more complex, e.g. hierarchical, adaptive, etc.

Previous studies[8] indicate that no particular ray-tracing acceleration structure is obviously the most efficient, since the total computation cost depends both on algorithm and hardware implementation. Whereas highly refined hierarchical acceleration schemes may require less intersection tests, the associated data structures are non-uniform (i.e. hard to parallelize), involve traversal of non-local data structures and as such are less suitable for cache and vector processing optimisations as available on modern processors and graphics cards. On the other hand, space partitioning methods, in particular those involving simple data structures like uniform grids, are more suitable to efficient implementation on vector processing elements.

We chose to implement a uniform grid ray-tracing algorithm similar to Amanatides and Woo[7]. A 3D uniform grid is assigned to the simulation box and each triangle is associated with every cell having a common interior point with it. The grid spacing in every direction is determined automatically via an empirical formula, which in our tests achieved a performance close to hand-optimisation: the number of cells on each axis is proportional to the square root of the total number of triangles and to the box length along the axis divided by the average box dimension (since in general the triangles form approximately a 2D shell, such a formula matches the average cell dimension to the average triangle dimension). Up to 64 cells per axis are allowed, in order to limit memory requirements. Given a ray specified by an origin and direction vector, a fast grid traversal algorithm computes the next grid cell intersected by the ray. Each triangle associated with this grid cell is then tested for intersection with the ray. No particular optimisation is done to avoid duplicate ray-triangle tests when one triangle spans multiple voxels. Thus a ray-triangle intersection is only considered if it occurs within the boundaries of the current cell. The grid traversal continues until a hit point is found or the ray exits the simulation box.
The software implementation was carefully designed to facilitate vector optimisation on SIMD-capable processors, i.e. to minimize branching and optimise instruction scheduling, in particular it is easily transferable to programmable graphics processing units. We tested the algorithm against the previous EASE ray tracing library which uses a HBV method. Using the Intel C++ optimising compiler on a Pentium4 processor, the grid algorithm applied to realistic test cases with more than 10000 triangles can be up to 5 times faster than the previous method (Fig. 1).

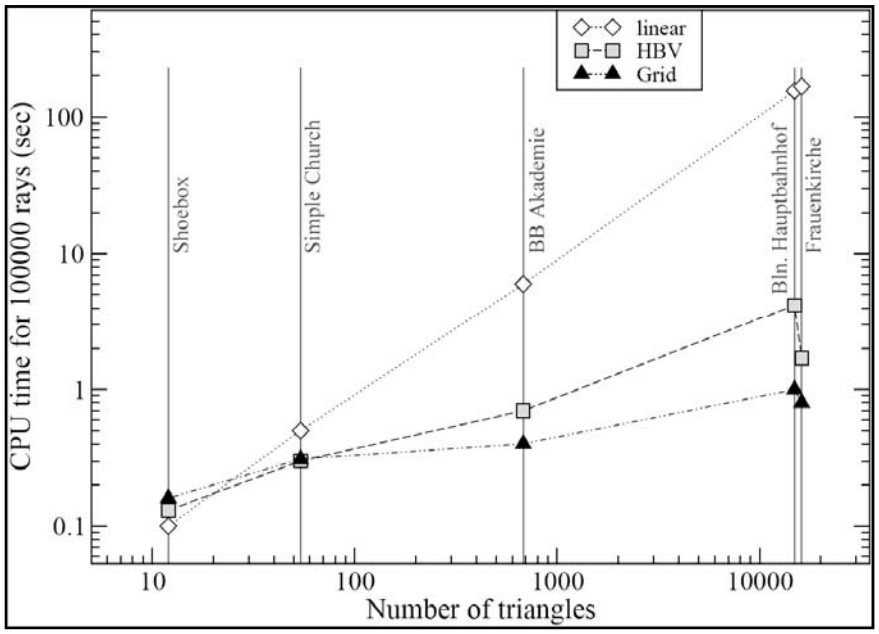

Figure 1: Comparison of algorithm performance vs. number of triangles for typical room acoustic geometries

2.2 Multi-thread mode

The ray-tracing part of the auralisation algorithm is well-suited to parallelization. Each speaker component generates a large number of rays that uniformly sample the surrounding physical space. The only interaction between these rays occurs when multiple rays hit the same receiver, at which point the sum of the contributions from each ray must be properly constructed. In the multi-threaded parallelization model, the ray-tracing for each speaker component is split among multiple threads (that can run for example on the multiple cores of a multi-core processor), whereby each thread is in charge of a random subset of the total rays (Fig. 2). The only thread synchronization is needed when generating a new ray (so the proper global distribution is preserved) and when updating a receiver (so the contributions from different threads/rays are properly summed). Since these synchronization points represent a small fraction of the total computation time, most of which is spent in the actual ray-tracing of each ray, the parallel efficiency is very high (Fig. 3): for typical projects, the use e.g. of a quad-core processor provides a speed increase of 3.5 relative to a single-threaded computation.

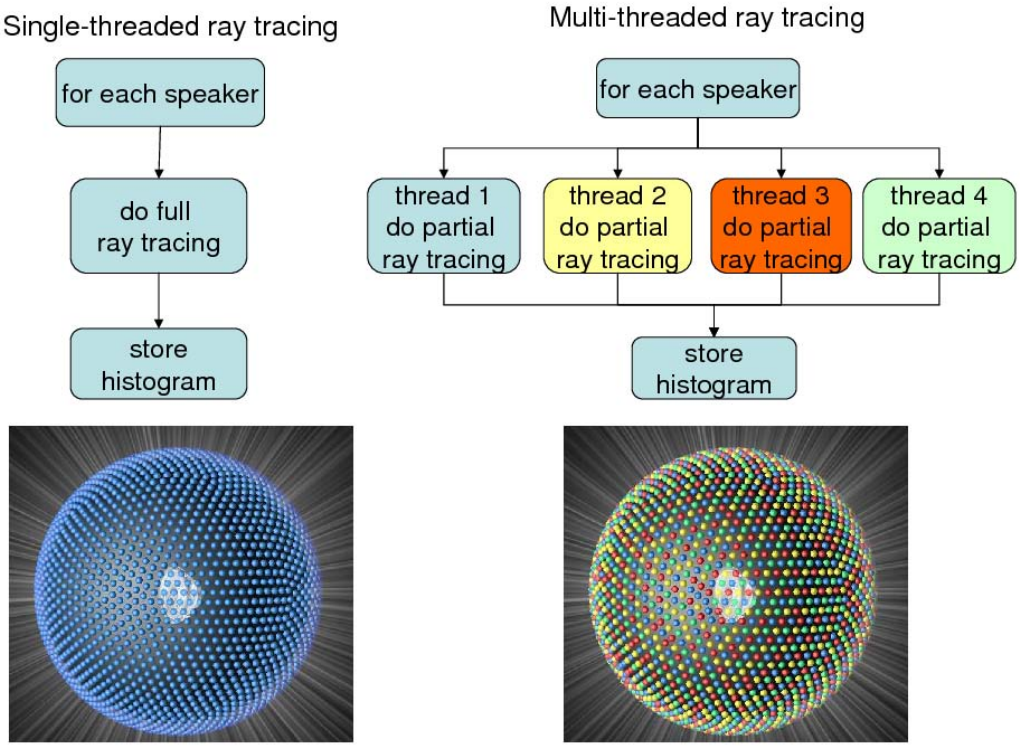

Figure 2: The points on the sphere represent ray directions used for sampling, originating at a speaker. At the left, all rays are processed sequentially in a single thread; while at the right rays with different colours are processed in parallel in separate threads.

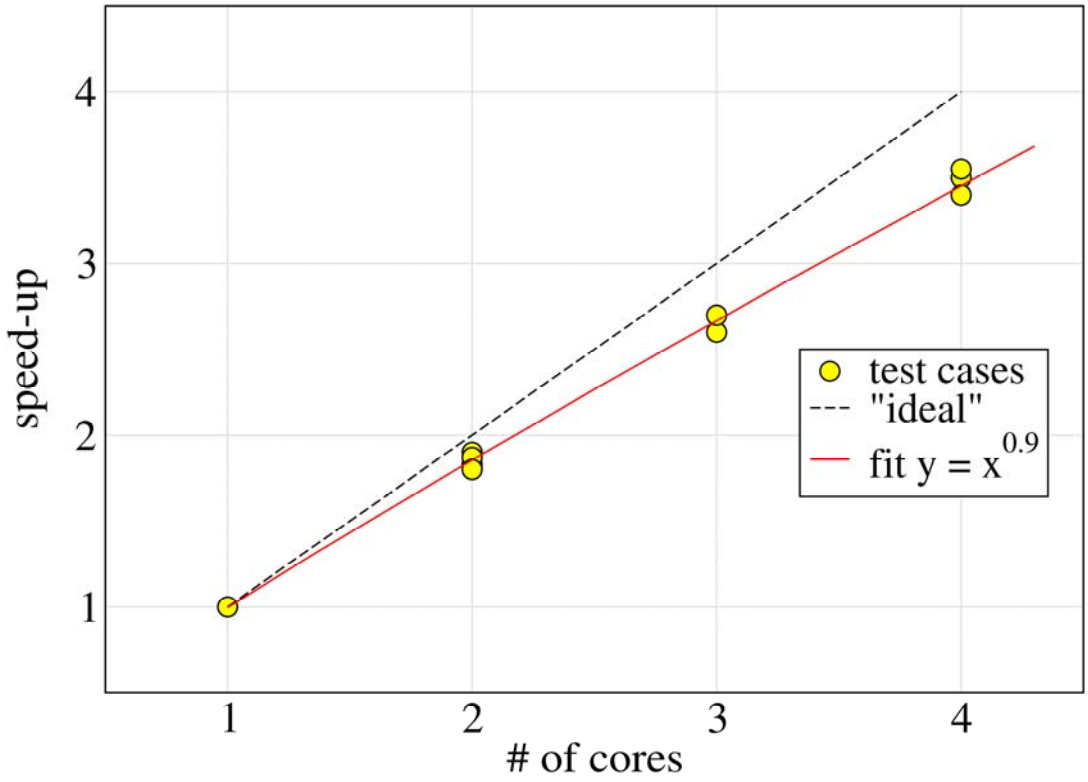

Figure 3: Performance of the multi-threaded AURA implementation on multi-core processors. Speed-up is computed as the ratio of the running time for the single-threaded computation to the running time for the multi-threaded/multi-core computation.

2.3 Network calculation

The multi-threaded parallelization is based on a shared-memory model. The calculation can be further parallelized according to a distributed-memory model, such that the computation can be split e.g. among independent computers connected by a network. In the simplest model, each computer runs the simulation for a different speaker, since contributions from each speaker are separately computed for each receiver place and do not interfere except at the final combination step. However, in this case achieving good load balancing (= avoiding idle time) is only possible when the number of speakers is much larger than the number of processors. If such is not the case, then better efficiency is obtained by extending the multi-threaded parallel ray-tracing to the distributed-memory architecture, such that a “thread” becomes now an independent process and synchronization occurs through explicit inter-process communication across the network. While synchronization is costlier than in the shared-memory implementation, the computation scales better since, on one hand, memory limitations of a single processor are overcome and, on the other hand, the number of processors that can be employed in parallel is virtually unlimited.
The next figure 4 shows the block diagram of a system to allow project processing using separate computer nodes[9, 10]. A user or via a web server the project is uploaded to a master PC (master node). Via a web server even some projects to become processed may be uploaded. This way a job queue is produced. The master node controls the processing nodes Node 1 to Node i via job specification. Between the processing nodes a communication network allows the needed synchronization and information over the processing progress. Finally the results are via network sent back to the master node. He gathers all results and produces the final results to send it back to the server or directly to a user.

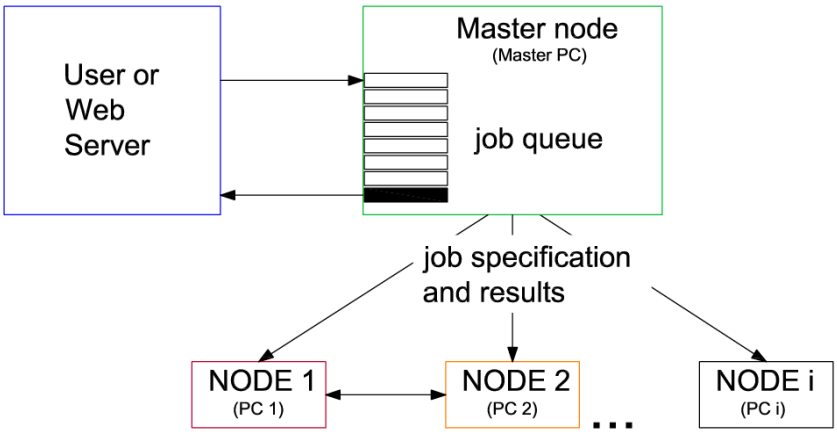

Figure 4: Network processing

The next question is now how the job splitting may be organized. There are two possibilities:

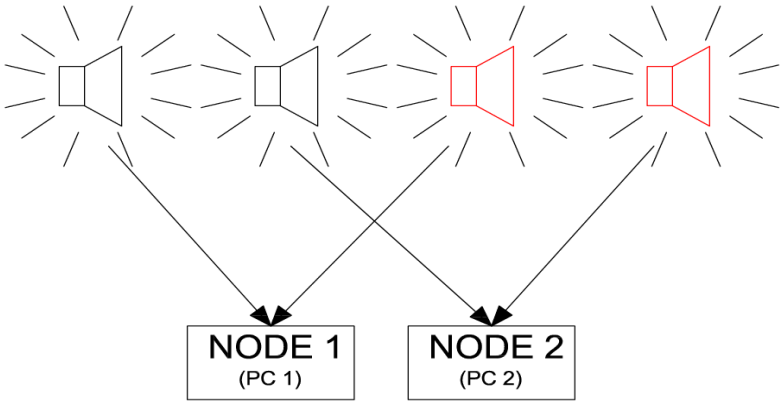

Figure 5: Source splitting

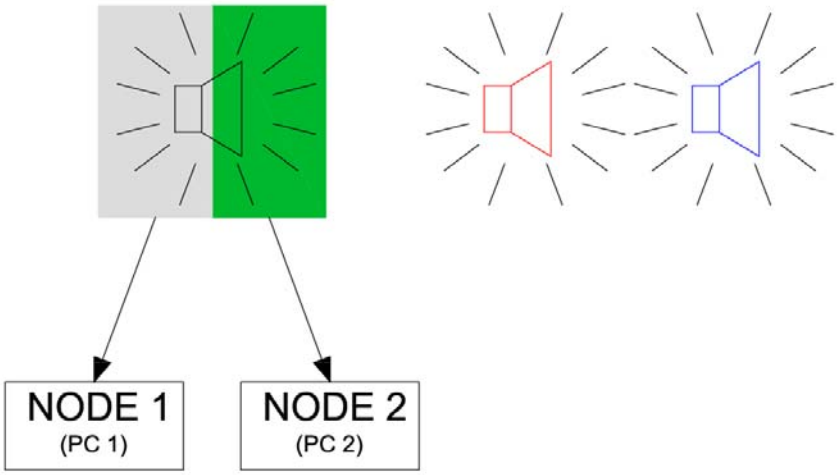

Figure 6: Particle or Ray splitting

The question is now how this splitting should happen, based on figure 5 or 6? We believe to do it this way: If the number of sources (loudspeaker) is significant larger as the number of available nodes than the source splitting should be preferred. In contrast to that, i.e. in case of only some sources by default the rays of a single source should be distributed to all nodes. Here more care must be taken for the synchronisation of the single nodes.
The implementation of this network processing is in the test phase, experiences with it must be reported later.

3 PRACTICAL TESTS

In a simple room and the corresponding model (see Fig. 7) the multi-thread mode has been checked. There we did impulse response measurements in the room at 12 selected seats and calculated the impulse response by using different approaches:

A Raytracing within the first 200ms and adding a statistical tail
B Raytracing within the first 200ms and adding a predicted tail
C AURA Raytracing for full length by using a one thread computer
D AURA Raytracing for full length by using a dual core computer
E AURA Raytracing for full length by using a quad core computer

It must be known that the tests A and B don't use scattering effects.
The results of the pure calculation time of the impulse response are shown in table 1:

	A	B	C	D	E
Calculation Time	20 min	20 min	60 min	29 min	16 min

Table 1: Calculation time comparison

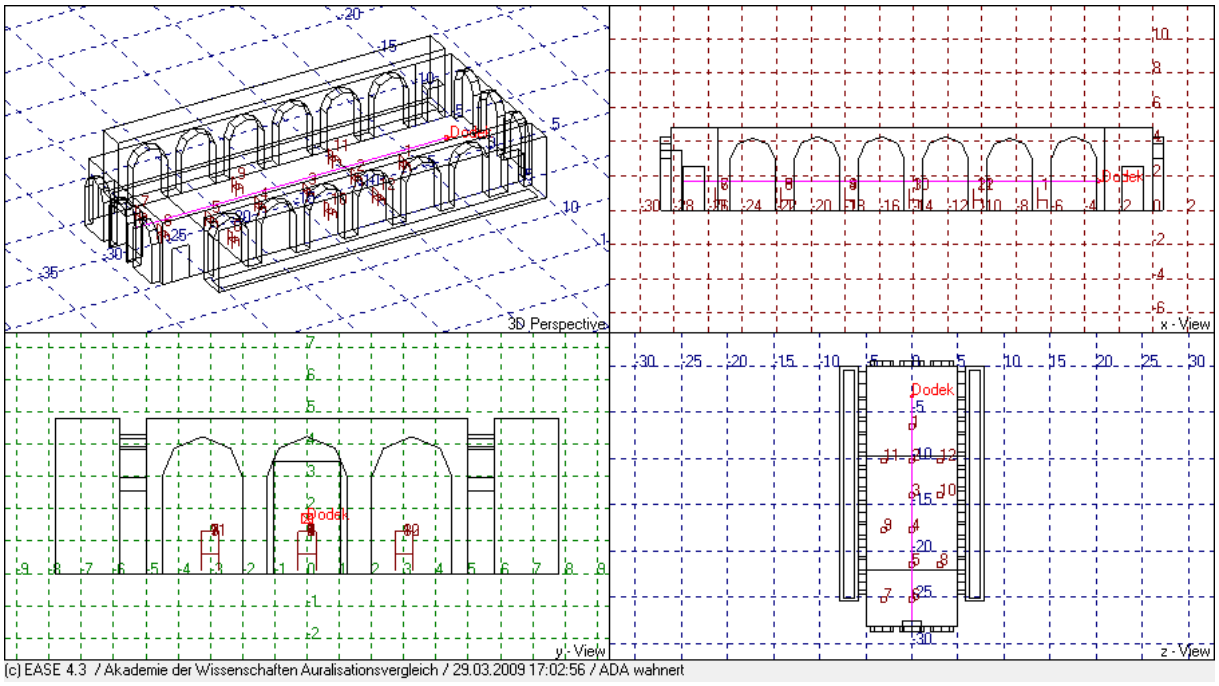

Figure 7: Wireframe model of the test room with an omni-source and 12 listener seats

4 CONCLUSIONS FOR INTELLIGIBILITY TESTS AND AURALISATION

As a results of any calculation, slowly done or fast we derive the impulse response. This calculated one will be later compared with a measured one and should come to similar results. The expected intelligibility and the later by measurements derived one should be in the same range. Otherwise any prediction is becoming worthless.

The next figure 8 shows a comparison between two simulated impulse responses, one with our EASE-AURA calculated (in margenta and red) and one with another software (blue and green). We recognize a certain hump in the last example, because only the first part of this ETC is calculated by Tracing-Routines, the late part is added statistically and creates the hump to get a smooth overall energy distribution. We understand that such calculations lead to wrong results to calculate STI or to use it for auralisation.

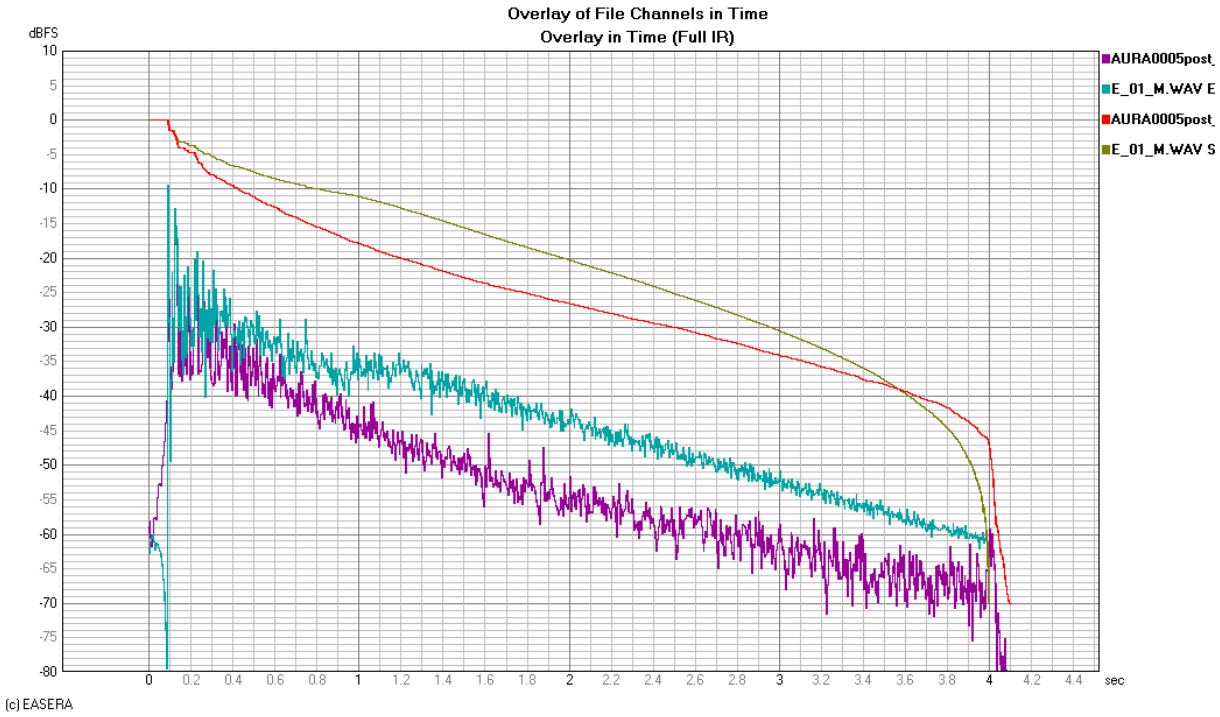

Figure 8: ETC comparison for fully and partially calculated impulse responses

STI calculations must be based on realistic derived full impulse responses including any **noise** and **masking** influences. Therefore deriving STI values from a direct sound distribution with added statistical tails should not be done anymore. The tools are available now to calculate full impulse responses in the same way we are doing later the corresponding measurements. A time frame is not set to measure in real rooms only the first 200ms and then to extrapolate the rest. Any time has his tools but also his restrictions. But today the era in doing predictions has started which needs realistic results and because of calculation time all these routines shouldn't used anymore which lead to incorrect results.

Regarding **auralisation** the time has come to convert this "toy" of the past in a tool useful not only for the acoustician, but also any user who expects to hear in advance realistic presentations.

5 SUMMARY

By using multi-thread methods and computer networks it becomes possible to derive complex impulse responses which still have taken 8 hours two years ago, now in one hour or less (2 quad-core computer, soon comes up octo core computer!). This way to derive realistic impulse responses will become a common method nowadays to check the sound coverage over an audience area.
Based on these realistic impulse responses precise predictions of intelligibility values are becoming possible.
The same way the auralisation procedure is becoming now a useful tool after years considered as a toy. The reason for this change is the way to derive the impulse responses used for auralisation in a right way and in very short time. With head tracker installations even with head phones the auralisation becomes a very realistic tool[11] and will be used more and more for pre-decisions of acoustic design issues.

6 REFERENCES

1. Ahnert, W., Feistel, R. Binaural Auralization from a Sound System Simulation Program. Presented at the 91st AES Convention, 1991, October 4-8, New York, Preprint No. 3127
2. Schmitz, O., Feistel, S., Ahnert, W., Vorländer, M. Merging Software for Sound Reinforcement Systems and for Room Acoustics, 110th AES – April 2001, Amsterdam.
3. O. Schmitz, Entwicklung und Programmierung eines Hybrid-Verfahrens zur Simulation der Schallübertragung in Räumen, Diploma Thesis, Institute of Technical Acoustics, RWTH Aachen University, 1997
4. Schmitz, O., Feistel, S., Ahnert, W., Vorländer, M., Grundlagen raumakustischer Rechenverfahren und ihre Validierung. Fortschritte der Akustik - DAGA 2001, Hamburg-Harburg, S. 24-25.
5. Rubin, S. M., Whitted, T., Computer Graphics 14(3), 1980
6. Fujimoto, A., Tanaka, T., Iwata, K. IEEE Computer Graphics and Applications 6(4), 1986
7. Amanatides, J., Woo, A. Eurographics'87 Conference Proceedings, 1987
8. Havran, V., Prikryl, J., Purgathofer, W. Tech. Rep. TR-186-2-00-14, Institute of Computer Graphics, Vienna University of Technology, 2000
9. OpenMP http://www.openmp.org
10. MPI http://www-unix.mcs.anl.gov/mpi/
11. Moldrzyk, C., Ahnert, W., Feistel, S. Acoustical Simulation based on Head-tracked Auralization and measured high-resolution HRTF and IR, Presented at 148th Meeting of the ASA, 15--19 November 2004, Session 1pAA, San Diego, California