Proceedings of 20th International Congress on Acoustics, ICA 2010

23–27 August 2010, Sydney, Australia

Spatial Data Structures for Dynamic Acoustic Virtual Reality

Dirk Schröder, Alexander Ryba and Michael Vorländer

dsc@akustik.rwth-aachen.de

Institute of Technical Acoustics, RWTH Aachen University, Neustraße 50, 52066 Aachen, Germany

PACS: 43.55.Ka

ABSTRACT

Over the last decades Virtual Reality (VR) technology has emerged to be a powerful tool for a wide variety of applications such as rapid prototyping, evaluation, therapy, or training tasks. For high quality auralizations (in analogy to visualization) of virtual environments, methods of Geometrical Acoustics (GA) are mostly applied to simulate the propagation of sound inside enclosures. By adapting acceleration algorithms such as BSP- and Octrees, current implementations can manage the computational load of moving sound sources around a moving receiver in real-time – even for complex scenarios. However, insertion, modification and extraction of geometrical objects are basic operations in many real-world experiences, but hierarchical spatial data structures do not support them efficiently. For this purpose the concept of Spatial Hashing was introduced, which is usually applied to collision detection tests of deformable objects in Computer Graphics. This contribution describes the design, implementation and integration of a dynamic object controller in the real-time room acoustics simulation software RAVEN. By adapting the concept of Spatial Hashing to the simulation algorithms, RAVEN is able to handle geometry modifications in real-time. The performance of the newly implemented data handling- and simulation routines is briefly discussed and compared to that of Brute Force and BSP-based algorithms.

INTRODUCTION

In recent years, the development of room acoustics prediction tools and auralization techniques has made a major leap forward enabling a physical-based simulation of virtual environments in real-time and, thus, adding a more plausible auditory component – in comparison to simple audio effects– to multi-modal Virtual Reality (VR) systems. A prerequisite for any immersive virtual environment is thereby the enabling of user interaction with the presented scenery to uphold and enforce the believability of the simulation (real world scenarios are usually not static).

In more demanding applications such as an architectural planning stage, it is convenient to manipulate also the room geometry itself and directly experience the impact on the room acoustics, e.g., via the interactive insertion/manipulation/removal of geometrical objects and the change of material data. While the real-time modification of geometry is a simple operation in visualization (see Fig. 1), a change of the scene geometry additionally affects the whole auralization chain such as the resimulation of corresponding Room Impulse Response (RIR)s, which also includes the update of spatial data structures, the generation/update/removal of Image Source (IS)s with subsequent tests on audibility and new ray tracing simulations.

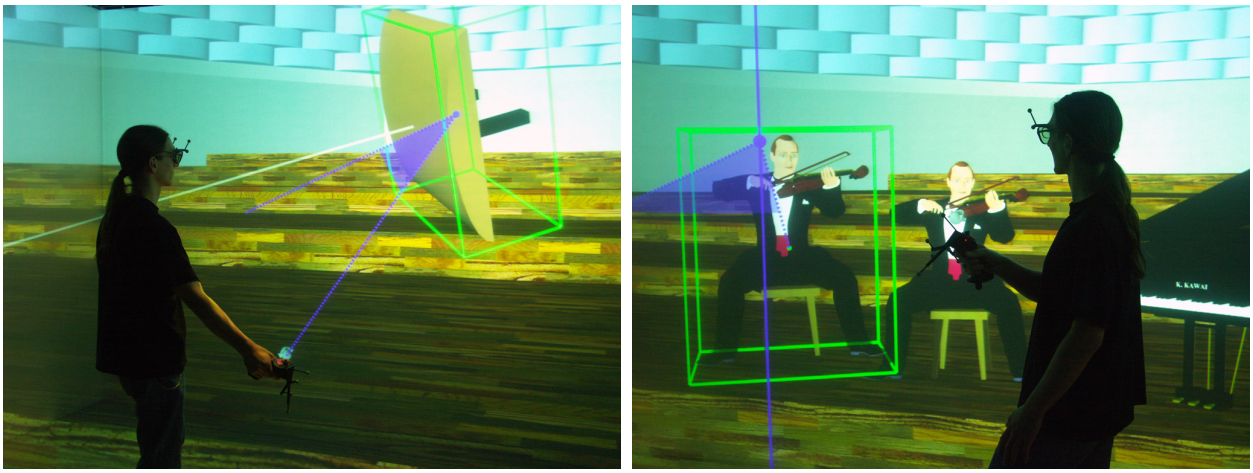
For this purpose the concept of non-hierarchical Spatial Hashing (SH), which also originates from Computer Graphics, is adapted to the requirements of room acoustics simulations and integrated in the hybrid room acoustics simulation framework Room Acoustics for Virtual ENvironments (RAVEN). RAVEN is one integral part of the auralization system of the CAVE-like environment at the Center of Computing and Communication of RWTH Aachen University [1, 2]. In this contribution, the concept of SH is described and the design, implementation and integration of a modular dynamic object controller is presented. The performance of two newly implemented SH-based routines is compared to existing Binary Space Partitioning (BSP)- and Brute Force (BF)- algorithms and SH optimization techniques are analyzed and briefly discussed.

Related Work

For the purpose of auralization of dynamically changing virtual environments three concepts have been presented so far to the authors' knowledge. One simple approach supports only the exchange of surface material data and has been presented in [3]. Lunden (and in an earlier stage Kajastila et al.) introduced a concept based on a beam tracing approach [4, 5] using a BSP acceleration method. The regeneration of BSP-trees, however, still yields a bottleneck in real-time auralization processes. A third concept based on SH has been presented by the first author in 2008 [6], which reduces significantly the complexity of geometry modifications by introducing a special spatial hash table that encodes the geometrical scene (see below). However, this contribution presents new detailed information on implementation concepts and comprehensive insights on algorithmic performance.

HYBRID ROOM ACOUSTICS SIMULATION

Today's computer simulations are accurate enough to compete with the established and reliable scale models [7]. For frequencies above Schroeder frequency, the most common approaches for computing RIRs are based on Geometrical Acoustics (GA), which reduces the sound field description to the dispersion of sound rays with a dedicated frequency and amount of energy, similar to the wave-particle dualism in physics. To overcome the different simulation demands of the early and late part of the response, RAVEN uses a hybrid simulation model that comprises two different simulation methods that are specialized on their concrete task. Here, RAVEN combines the deterministic IS method [8] for the computation of early reflections with a



(a) Insertion and adjustment of a geometrical object (reflector panel).

(b) Insertion and adjustment of a natural sound source (violin player).

Figure 1: Typical user interaction within a virtual environment, here by the example of the CAVE-like environment at RWTH Aachen University. Objects can easily be inserted, modified and removed using an intuitive gesture-driven input device.

stochastic ray tracer [9], which determines the energy envelope of the reverberant sound field. The computational costs of both methods are dominated by the large number of required intersection tests between ray segments and the polygonal scene model due to the sound wave discretization into energy rays. Regarding real-time simulations, these tests have to be significantly accelerated, which can be done by encoding the scene geometry into spatial data structures.

SPATIAL DATA STRUCTURES

Spatial data structures aim at encoding a given geometric scene in an efficient way in order to significantly accelerate common operations such as intersection tests, collision detection and culling algorithms [7]. Most methods are based on a space subdivision that is encoded in tree-like or cell-like data structures following either an object-oriented or space-oriented partitioning strategy.

Bounding Volume Hierarchies (BVH)s, BSP- trees and Octrees (in 2D space Quadtrees) belong to the group of hierarchical tree data structures. Here, BVHs wrap all geometric objects in bounding volumes, such as spheres and (axis-aligned) bounding boxes, that are then enclosed within larger bounding volumes in a recursive way until the whole geometry fits into exactly one bounding volume. Octrees subdivide a 3D-space by means of planes as partitioners that recursively subdivide the space evenly along all three axes resulting in 8^n new subspaces for n iterations. BSP-trees, on the other hand, can be generated in three basic variants, either with an arbitrary, axis-aligned or a polygon-aligned space partitioning. A space partitioning with no constraints on the choice of the dividing planes apparently yields the best possible encoding of the geometry. The crucial part is to find partitioners that produce a balanced tree of minimum height which is the most efficient tree structure in terms of search operations. As there is an infinite number of possible partitioners, heuristic approaches are commonly used to optimize the space partitioning. Computation times vary here in the range from minutes to hours, to weeks. In contrast, axis-aligned BSP-trees and Octrees can be computed much faster as they follow a given subdivision pattern. Axis-aligned BSP-trees are created similar to Octrees, but with only one instead of three partitioning planes. Contrary, polygon-aligned BSP uses planes spanned by the geometry's polygons as it makes the BSP-tree creation much more efficient and transparent. The latter type of BSP-trees is applied in RAVEN for very fast intersection tests of ray segments with the geometric scene [8].

Another strategy of spatial subdivision is called Voxelization which belongs to the group of cell data structures. Similar to the rasterization of a 2D scene, voxelization (non)-uniformly subdivides the 3D space into box-shaped subspaces that are organized in a 3D data grid where each grid cell contains the geometry that is enclosed within the respective subspace. As a matter of principle, search operations on this type of data structures can never compete with the performance of a balanced tree data structure. Instead, voxelization supports another important operation, that is the fast insertion, manipulation and deletion of polygonal scene objects. It should be kept in mind that hierarchical tree data structures always require a recomputation when the geometry has changed while only a few subspaces have to be updated in a voxelized space. The most efficient method for addressing such a cell data structure is called SH, which reduces the complexity of a change in geometry significantly. More details on SH are given in the next subsection, as this concept is used in RAVEN to handle scene modifications in real-time.

Spatial Hashing originates from Computer Graphics and has lately been applied to speed up applications such as the real-time collision detection of huge deformable objects [10]. The concept of SH is based on the idea of subdividing the space by primitive volumes called *voxels* and map the infinite voxelized space to a finite set of one-dimensional hash indices, i.e., a HT, which are en/decoded by a hash function [11]. Using a hash function for spatial subdivision is a very efficient strategy. Each voxel contains the respective encapsulated scene polygons and is addressed by the corresponding hash index. These indices can be computed as follows: Considering an axis-aligned cubical voxel with edge length a as the subdividing volume, the coordinates (x,y,z) of an arbitrary point are quantized to the voxel coordinates (u,v,w), in particular, the coordinates are subdivided by the voxel's edge length a and floored to the next integer with

$$u = \left\lfloor \frac{x}{a} \right\rfloor, v = \left\lfloor \frac{y}{a} \right\rfloor, w = \left\lfloor \frac{z}{a} \right\rfloor. \quad (1)$$

Then the voxel coordinates (u,v,w) are mapped to a hash index

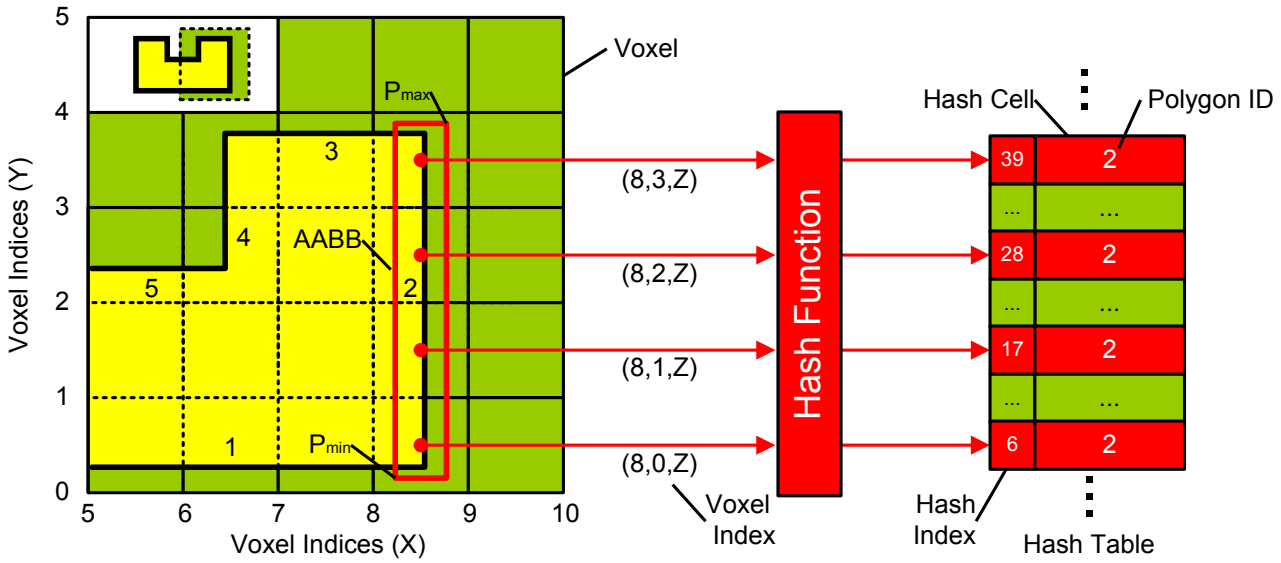


Figure 2: Example of hashing geometrical objects in a voxelized space. Every voxel in space is mapped to an entry (hash index) in the HT. The intersection of an object with these voxels is stored in the HT.

i using a special hash function. In this contribution two different hash functions h_1 and h_2 are considered, which are defined to

$$i = \begin{cases} h_1(u,v,w) & = (u \cdot p_1 \oplus v \cdot p_2 \oplus w \cdot p_3) \ mod \ n \\ h_2(u,v,w) & = (u \cdot p_1 + v \cdot p_2 + w \cdot p_3) \ mod \ n \end{cases} \quad (2)$$

, where i is the hash index, p_1, p_2, p_3 are large prime numbers, $\oplus$ denotes the logical XOR-operator and n stores the size of the HT. As one can see, the modulo operation reduces the infinite 3D space to a finite 1D set, which unfortunately could result in the mapping of two or more voxels to the same hash index, so-called *hash collisions*. Hash collisions do not necessarily result in a simulation error as they can easily be intercepted, though this causes an additional computational effort and should be kept at a low level– at least in real-time applications. To reduce the number of hash collisions, three factors are important: 1) the hash function should distribute the hashes uniformly over the output space to avoid multiple mapping to the same index, 2) the HT size should be kept at a moderate level in terms of memory management and 3) the voxel's edge length in relation to the scene's dimensions should be optimized as it strongly influences the overall performance of search operations. All these factors will be thoroughly discussed in the Performance section.

Fig. 2 illustrates the voxelization of the same simple geometry that was used in the previous subsection. The concept of SH will be demonstrated for polygon 2 using the second hash function h_2 that was described above. In this example, the hash function's prime numbers are indiscriminately set to $p_1 = 7, p_2 = 11, p_3 = 13$, the HT is $n = 50$ in size, and the voxel's edge length is assumed to be $a = 2$. At first the Axis-Aligned Bounding Box (AABB) of polygon 2 is computed, but only the AABB's minimum and maximum coordinate values are actually of interest, denoted as $P_{min} = (17, 0.5, 0)$ and $P_{max} = (17, 6.5, 3)$ in Fig. 2. These two points are then mapped to the corresponding voxel coordinates V_{min} and V_{max}, with

$$V_{min} = (\left\lfloor \frac{P_{min,x}}{a} \right\rfloor, \left\lfloor \frac{P_{min,y}}{a} \right\rfloor, \left\lfloor \frac{P_{min,z}}{a} \right\rfloor) = (8,0,0) \quad (3)$$

$$V_{max} = (\left\lfloor \frac{P_{max,x}}{a} \right\rfloor, \left\lfloor \frac{P_{max,y}}{a} \right\rfloor, \left\lfloor \frac{P_{max,z}}{a} \right\rfloor) = (8,3,1). \quad (4)$$

All possible voxels between the discretized minimum V_{min} and the discretized maximum V_{max} of the AABB have to be traversed, but only the lowest voxel row ($min(z) = 0.5$) will be further regarded, i.e., $V_1 = (8,0,0)$, $V_2 = (8,1,0)$, $V_3 = (8,2,0)$ and $V_4 = (8,3,0)$, as the hashing procedure always follows the same pattern. Here, the corresponding hash indices of the four voxels are:

$$\begin{aligned} i_1 &= (8 \cdot 7 + 0 \cdot 11 + 0 \cdot 13) \ mod \ 50 = 6 \\ i_2 &= (8 \cdot 7 + 1 \cdot 11 + 0 \cdot 13) \ mod \ 50 = 17 \\ i_3 &= (8 \cdot 7 + 2 \cdot 11 + 0 \cdot 13) \ mod \ 50 = 28 \\ i_4 &= (8 \cdot 7 + 3 \cdot 11 + 0 \cdot 13) \ mod \ 50 = 39 \end{aligned} \quad (5)$$

After computing the voxels' hash indices, polygon 2 is sorted into the respective cells of the HT and the next scene polygon is proceeded in the same way. The advantage of SH over other spatial data structures such as BSP-trees is that the insertion/deletion of m vertices into/from the HT takes only $\mathcal{O}(m)$ time. Thus, this method is perfectly qualified to efficiently handle modifications of a polygonal scenery in order to enable a real-time auralization of a dynamically-changing environment.

Intersection Tests

RAVEN's hybrid room acoustics simulation approach is based on the computation of audible ISs and stochastic ray tracing. As a matter of principle, these methods of GA usually demand for millions of intersection searches between ray segments and the scene geometry, which usually builds a bottleneck for real-time applications. At the moment, RAVEN features four strategies of intersection searches using different types of search acceleration: BF, BSP and SH. Here, the BF approach is the most simple strategy, which tests a considered ray segment always against all scene polygons on intersection. In contrast, the BSP-based algorithm yields the best performance by testing only subsets of convex polygon sets, where in case of balanced trees the search complexity drops down from $\mathcal{O}(n)$ up to $\mathcal{O}(log_2(n))$, where n denotes the number of scene polygons. In this contribution, further details on the BSP-based intersection search are omitted

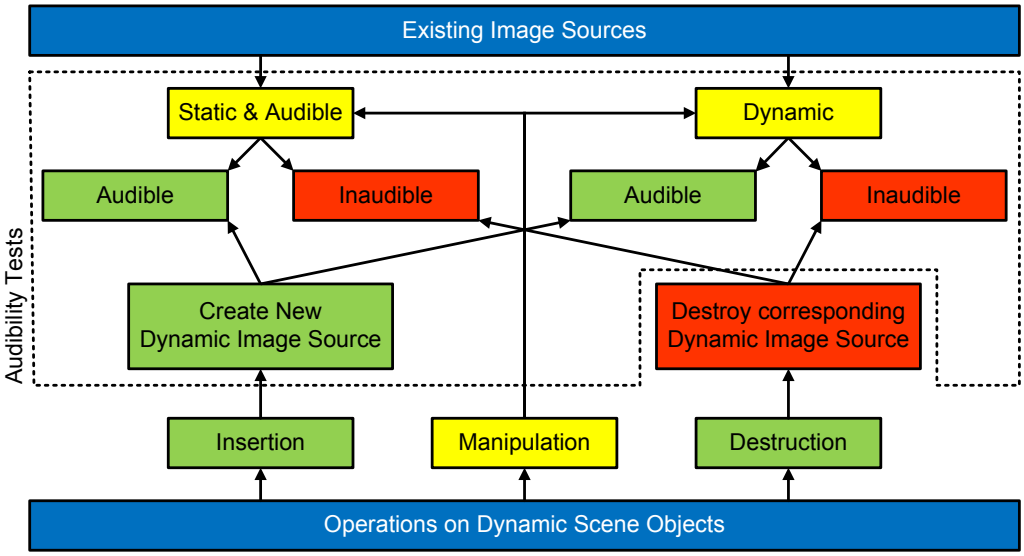


Figure 3: Concept of handling audibility tests in "dynamic mode". Regarding the type of operation on a dynamic object not all image sources have to be tested on audibility. It is assumed, that the "static" scene, as well as the source and the receiver position, are fixed in "dynamic" mode, so that only dynamic objects can be manipulated.

due to space restrictions, though a complete description is given in [8]. Two types of intersection search algorithms are currently implemented in RAVEN that use the concept of SH, called , Voxel Tracing and Voxel Candidates. While Voxel Tracing uses a smart identification of intersection candidates, Voxel Candidates declares all voxels as potential intersection candidates that are enclosed by the bounding box spanned by the starting- and end point of the observed line segment (more details are given in [12]). A brief performance analysis of each method is given in the performance section.

HANDLING OF DYNAMIC SCENE OBJECTS

During a preprocessing step, a *static* room acoustical scene is selected from a database, necessary parameters for ray tracing and image source method are defined and the simulation is initialized. At runtime, impulse responses for all possible sound propagation paths are computed using BSP-accelerated simulation algorithms [8],[13]. In the following, this type of simulation is called *static* mode where no geometry manipulations are allowed. In contrast, in the *dynamic* mode six basic operations on dynamic objects are considered: insertion, destruction, and manipulation, which includes the translation, the rotation, and the scaling of the object as well as the change of surface materials. Here, the object's initial geometry is either imported from a database or sketched on-the-fly. It should be pointed out, that the two modi operandi cause no restriction on user interaction for the VR-simulation– it merely has to be understood as a feature to reduce the computational complexity. Simulation events are always distributed independently on a computing cluster using a master scheduler application [14].

Regarding ray tracing simulations, it is sufficient to update both the geometry and the corresponding spatial data structures, and resimulate either in static or dynamic mode. In contrast, a dynamic handling of IS simulation turns out to be more complicated, as they have to be generated, destroyed and updated (audibility and position) at runtime. For a convenient insertion, manipulation and destruction of ISs, RAVEN uses hierarchical tree data structures to organize ISs, where the tree height corresponds to the reflection order (see Figure 4). Fig. 3 shows the concept of performing IS audibility tests in the *dynamic* mode. If a new dynamic object is created, an audibility test for all new generated *dynamic* image sources must be performed. Assuming that the sound source and the receiver position are fixed, the audibility test of already existing ISs can be reduced to check only currently audible static ISs and audible dynamic ISs of previously inserted dynamic objects. In contrast, if a dynamic object is destroyed, only the existing inaudible ISs, both *static* and *dynamic*, have to be further checked, as already audible ISs stay audible at any case. In case of object manipulation, such as translation, rotation and scaling operations, no ISs can be excluded from the audibility test, which makes these operations the most time consuming events in terms of computational performance. In general, algorithms based on SH cannot compete with the performance of BSP-trees (see section Performance). For this reason, RAVEN switches back to *static* mode as soon as all geometric modifications are carried out, whereby BSP-trees of modified rooms are recomputed and the set of currently audible ISs is updated.

Insertion of a dynamic object

In case of inserting a dynamic object into the geometric scene, new ISs have to be generated that relate to the newly inserted geometry. As mentioned above, RAVEN organizes ISs in a hierarchical tree structure, which therefore has to be expanded accordingly. Fig. 4 illustrates this process where a single polygon, i.e., an additional reflection plane is added to the scene. In each subsequent branch of the primary source, i.e., the image source of zeroth order, a new brother node is inserted into the tree representing another IS (orange nodes). In order to compute all possible ISs up to a given maximum order, the new generated nodes have to be expanded to the full tree (red nodes). The position of each newly generated IS is always computed by mirroring the corresponding father node on the inserted reflection plane.

Destruction of a dynamic object

Removing a dynamic object from the scene is a fast and simple operation. The update of the HT does not affect other existing objects and takes only $\mathcal{O}(n)$ time, where n is the total number of the object's vertices. In order to remove the corresponding ISs from the IS tree, all subsequent trees of the object's father nodes (orange nodes) just have to be destroyed recursively from down to top (see Fig. 4). After all ISs are removed from the tree, a new audibility test is carried out testing only the formerly inaudible ISs as sound propagation paths might be unblocked

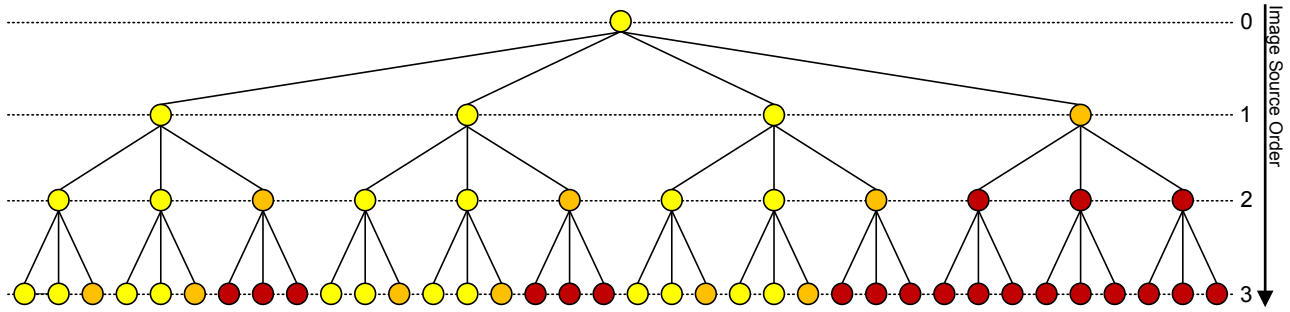


Figure 4: Handling of ISs using a hierarchical tree data structure.

Scene Model	No. of Polygons	No. of Planes	Avg. Edge Length [m]	No. of ISs (1st/2nd/3rd)
Classroom (CR)	391	81	0.79	82/6662/524962
Eurogress (EG)	391	142	3.93	143/20165
Lecture Room (LeR)	147	28	2.89	29/785/21197
Living Room (LiR)	67	31	1.43	32/962/28862
Metro Station (MS)	272	129	4.03	130/16642

Table 1: Information on geometry and number of corresponding ISs of the applied test scenarios.

Scene Model	IS Order (No. of ISs	Hash(VC)[ms]	Hash(VT) [ms]	BSP [ms]	Brute [ms]
Classroom (CR)	2(6562)	33.3 (1.47)	37.6 (1.46)	26.7	193.0
Eurogress (EG)	2(20165)	195.9 (1.80)	126.7 (1.63)	76.1	510.2
Lecture Room (LeR)	3(21197)	177.1 (1.45)	129.2 (1.29)	62.1	293.7
Living Room (LiR)	3(28862)	218.3 (1.23)	174.2 (1.44)	66.7	215.4
Metro Station (MS)	2(16642)	165.0 (1.33)	118.2 (1.50)	59.5	346.3

Table 2: Averaged measurement results of IS audibility tests based on VC and VT, compared to the computation times of both, the BF- and the BSP-algorithm. The corresponding average ratio between voxel size and average edge length of the scene polygons is given in parentheses.

now (see section Handling of Dynamic Scene Objects).

Manipulation of a dynamic object

Each object manipulation goes along with a position update of all corresponding IS, which is basically the reverse search operation in comparison to an object destruction. Therefore, Fig. 4 can also be used to further illustrate this process. Contrary to the object destruction, the IS tree is now traversed from top to down, starting from the object's father nodes (orange) and following all underlying subbranches (red nodes) until the leaves of the tree are reached. Thereby, the position of each traversed IS (orange and red nodes) is updated according to the modified geometry.

PERFORMANCE

In this section the performance of the single algorithms is investigated on real-time capability. Five different geometrical scenes of different complexity and size are considered for this performance analysis. In particular, these are a the models of a classroom, the Eurogress in Aachen, a living room, a lecture room, and a metro station in Warsaw. Information on geometry and the number of image sources for all models is summarized in Tab. 1. Due to a lack of memory on the test system, both models, Eurogress and metro station, are considered only for image sources up to second order. All tests were carried out on an off-the-shelf desktop personal computer with an AMD Athlon 2.7 GHz single core CPU, 4 GB RAM (800 MHz) and Visual Studio 2005 SP1 as development environment. RAVEN is completely written in C++. Although many functions are usually computed in parallel using OpenMP, multi-core CPUs are omitted in this analysis for the sake of comparability – some methods (especially SH) would gain more from the usage of parallel computing than others. Computation times are measured by performing ten iterations of each simulation method and averaging the respective results. The IS audibility tests based on SH were carried out for both methods, VT and VC, in each case with a fixed HT size. The voxel size s was manipulated in relation to the average edge length of all scene polygons a_a resulting in a factor f, with $f = s/a_a$. In Tab. 2 the best results for both approaches are summarized where the chosen values for f are given in parentheses. Additionally, the computation times of the BF- and the BSP-algorithm are given for a better assessment of the results. A voxel size of one and one half of the average edge length of the scene's polygons provided good results for all room models. More elaborated performance tests and optimization techniques are presented in [12].

SUMMARY

In this contribution the concept of SH was adapted to the hybrid room acoustics simulation of virtual environments that include dynamic objects that were insertable, destructible and manipulable by a user at runtime. While this concept was easily embeddable to the applied stochastic ray tracing algorithm where it was sufficient to update just both the geometry and the corresponding spatial data structures, a dynamic handling of ISs simulation turned out to be more complicated, as they have to be generated, destroyed and updated (audibility and position) at runtime. Therefore, a hierarchical tree data structure was introduced that organizes ISs in order to enable a convenient processing of ISs.

One of the main goals considering the handling of dynamic objects was to analyze if the new concept is applicable to real-time applications. Performance analysis were carried out for five different scene models on an off-the-shelf personal computer that

came with a single core AMD CPU with 2.7 GHz, 4 GB Ram (800MHz) and Visual Studio 2005 SP1 as development environment. Here, the analysis of intersection tests of ray segments with scene polygons has shown a strong dependency of the achieved performance on the chosen voxel size where the optimal size turned out to be scene dependent. Good results were obtained for all scene models by setting the voxel size to one and one half of the average edge length of the corresponding scene polygons.

It cannot be denied that BSP-accelerated intersection tests are much more efficient than the two introduced methods based on SH, but each object manipulation requires an update of the BSP-tree which results in additional computation time that – in combination with the IS audibility test – cannot compete with performance of SH. However, in cases of geometry and ISs of high complexity and high order, respectively, a large number of audibility tests has to be performed, where the combination of BSP tree regeneration and fast BSP-based intersection tests can exceed the performance of the SH approach – at least on a single core CPU. Here, the computation time significantly drops with any additional CPU core as the HT data structure is perfectly computable in parallel. Approaches of SH will therefore outperform hierarchical tree data structures on multi-core CPUs.

ACKNOWLEDGMENTS

The authors would like to thank Ingo Assenmacher and Lenka Jeřábková for their support and valuable discussions about the concepts of Spatial Hashing. Furthermore, the authors would like to thank the German Research Foundation (DFG) for funding this project.

REFERENCES

[1] 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*, 2007:19, 2007.

[2] D. Schröder, F. Wefers, S. Pelzer, D. Rausch, M. Vorländer, and T. Kuhlen. Virtual Reality System at RWTH Aachen University. In *Proceedings of the International Symposium on Room Acoustics (ISRA)*, Melbourne, Australia, 2010.

[3] O. Deille, J. Maillard, N. No, K. Bouatouch, and J. Martin. Real time acoustic rendering of complex environments including diffraction and curved surfaces. In *Proceedings of the AES 120th Convention, Paris, France*, 2006.

[4] R. Kajastila, S. Siltanen, P. Lunden, T. Lokki, and L. Savioja. *A distributed real-time virtual acoustic rendering system for dynamic geometries*. Journal of the Audio Engineering Society, 122:7160, 2007.

[5] P. Lunden. Uni-Verse Acoustic Simulation System: interactive real-time room acoustic simulation in dynamic 3D environments. In *2nd ASA-EAA joint conference Acoustics, Paris, France*, 2008.

[6] D. Schröder and I. Assenmacher. Real-time auralization of modifiable rooms. In *2nd ASA-EAA joint conference Acoustics, Paris, France*. 2nd ASA-EAA joint conference Acoustics, Paris, 2008.

[7] M. Vorländer. *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. Springer-Verlag Berlin, 2005.

[8] D. Schröder and T. Lentz. Real-time processing of image sources using binary space partitioning. *Journal of the Audio Engineering Society (JAES)*, 54(7/8):604–619, 2006.

[9] D. Schröder, P. Dross, and M. Vorländer. A fast reverberation estimator for virtual environments. In *Proceedings of the 30th AES International Conference*, Saariselkä, Finland, 2007.

[10] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. *Optimized Spatial Hashing for Collision Detection of Deformable Objects*. VMV '03, 2003.

[11] T. Akenine-Möller and E. Haines. *Real-time Rendering*. A K Peters Verlag, Second Edition, 2002.

[12] D. Schröder, A. Ryba, and M. Vorländer. Real-time auralization of dynamically changing environments. *submitted to Acta Acustica united with Acustica*, May 2010.

[13] D. Schröder and M. Vorländer. Hybrid method for room acoustic simulation in real-time. In *Proceedings of the 20th International Congress on Acoustics (ICA)*, Madrid, Spain, 2007.

[14] M. Schlütter. *Parallelisation of Algorithms for Real-time Room Acoustics Simulation (diploma thesis)*. Virtual Reality Group of RWTH Aachen University, 2009.