# Design and Evaluation of Saliency-Driven Progressive Audio Processing,
# Entwurf und Evaluierung prioritätenbasierter progressiver Audioverarbeitung

Diplomarbeit im Fach Informatik

vorgelegt von

**Thomas Moeck**

geb. am 17. Juni 1980 in Nürnberg

angefertigt am

**Institut für Informatik**
**Lehrstuhl für Graphische Datenverarbeitung**
**Friedrich-Alexander-Universität Erlangen-Nürnberg**

Betreuer: Dr. Nicolas Tsingos

Betreuender Hochschullehrer: Prof. Dr. Marc Stamminger

Beginn der Arbeit: 01.08.2006

Abgabe der Arbeit: 01.02.2007

# Contents

# List of Figures

LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Motivation

Spatialized audio rendering is a very important factor for the realism of interactive virtual environments, such as those used in computer games, virtual reality, or driving / flight simulators, etc. The complexity and realism of the scenes used in these applications has increased dramatically over the last few years. The number of objects which produce noise or sounds can become very large, e.g., cars in a street scene or crowds in the stadium of a sports game. In addition, recent sophisticated physics engines can be used to synthesize complex sound effects driven by the physics, for example the individual impact sounds of thousands of pieces of a fractured object.

Realistic 3D audio for such complex sound scenes is beyond the capabilities of even the most recent 3D audio rendering algorithms. The computational bottlenecks are numerous, but can be grouped into two broad types: the cost of *spatialization*, which is related to the audio restitution format used; and the *per sound source* cost, which relates to the different kinds of effects desired. An example of the former bottleneck is Head Related Transfer Function (HRTF) processing for binaural rendering [Møl92], or the rendering of numerous output channels for a Wave Field Synthesis (WFS) system [BdVV93], which can use hundreds of speakers. The latter bottleneck includes typical spatial effects such as delays, the Doppler effect, reverberation calculations, but also any kind of studio effect the sound engineers / designers wish to use in the application at hand.

Recent research has proposed solutions to these computational limitations. Perceptual masking with sound source clustering [TGD04], or other clustering methods [Her99, WS04] do resolve some of the issues. However, the clustering algorithms proposed to date are either restricted to static scenes, or add an unacceptable computation overhead due to a quadratic step in cluster construction when the number of sources is large. In addition, the cost of per source computation, sometimes called premixing, can quickly become a bottleneck, again for complex soundscapes.

For this work two contributions have been developed:

- The first contribution is the speed improvement of the clustering and premixing steps of the [TGD04] approach. First, a new recursive clustering method is introduced, which can operate with a fixed or variable target cluster budget, and thus addresses the issue of spatialization cost mentioned above. Second, a novel "pinnacle-based" scalable premixing algorithm is developed, based on [Tsi05], providing a flexible, perceptually-based framework for the treatment of the per-source computation costs.
- The second contribution is the investigation of perceptual issues related to *clustering* and *premixing*, based on conducted *pilot user studies*. In particular, the influence of visuals on audio clustering for audio-visual scenes was investigated, and the result is a modified metric which takes into account the indication that it is probably better to have more sources in the view frustum. For scalable premixing, the different metrics which can be used were evaluated, including recently developed perceptual models, such as the audio saliency map [KPLL05], and a perceptual quality test was performed for the new algorithm.

In the following there comes a short introduction into sound in general. After that a brief overview of related previous work is attached, including acoustics and perception literature. Then follows a survey of the sound processing engine. In Sect. 3 there is a presentation of the new recursive clustering method; the new study and the resulting algorithm for scalable premixing are located in Sect. 4; the study and novel metric for crossmodal audio-visual clustering in Sect. 5. Some implementation issues and results are presented, concluding with a discussion of the approach.

## 1.2 Physical and digital sound

Sound is a physical phenomenon created by vibration of matter, which puts the surrounding air into oscillation by varying the pressure. The variation of low and high pressure is transmitted through the air creating wave-like motion which can be heard as sound when reaching human ears.
Elementary harmonic sound waves are defined by the *frequency* and the *amplitude*. The frequency represents the elevation of the perceived sound, the amplitude is its volume. Both may well vary through time resulting in a graphically displayable wave form (see the top of Figure 1.1). More than one wave can be present at the same time while each having different amplitudes and frequencies, superimposing each other and thus yielding e.g. a voice or music. A human is capable of hearing frequencies of about approximately 20 Hz up to 20 kHz, normally decreasing when aging.

Digital sound has two main attributes: the *sampling rate* and the *quantization*. The former is the number of samples per time-unit, the latter is the number of bits used to encode the value of the amplitude. As stated by the sampling theorem [Sha49], the sampling rate has to be twice as high as the maximum possible frequency. That is why a high quality sound's sampling rate has to be at least 40 kHz (i.e. 40,000 sample points per second) to be able to represent frequencies up to 20 kHz. The standard CD sampling rate is 44.1 kHz.

Having values for the amplitude at each time-instant, the frequencies are implicitly coded with the changes of the amplitude. They can be obtained by e.g. doing a *Fast Fourier Transformation* (FFT) over a

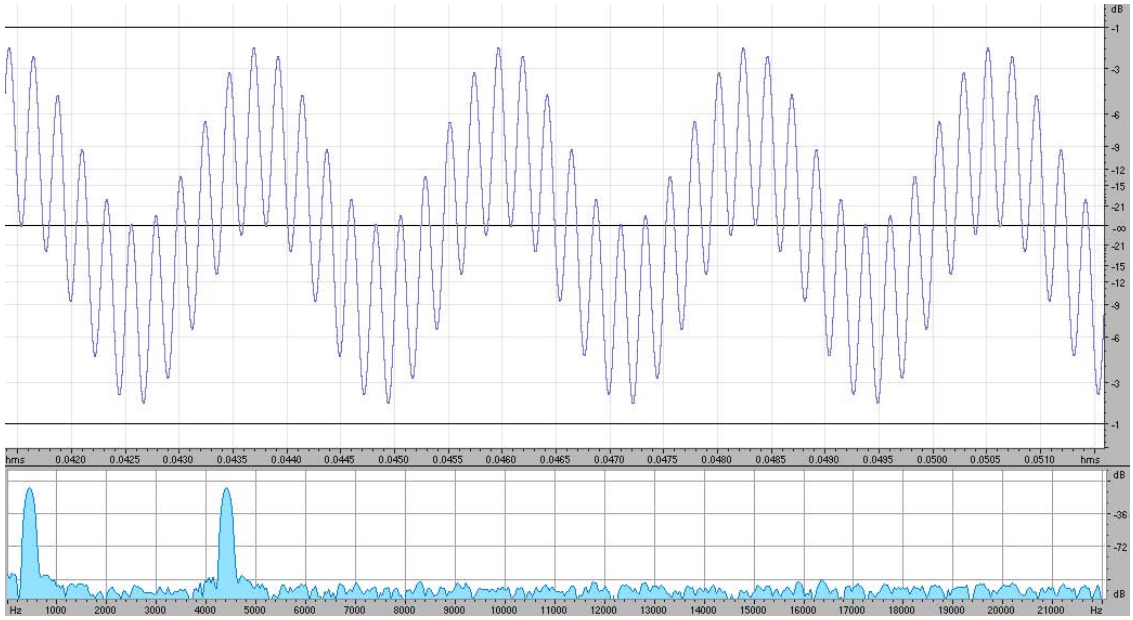


Figure 1.1: A 440 Hz wave superposed by a 4400 Hz wave. Notice the FFT representation at the bottom where the two frequencies contained in the wave are clearly visible.

number of samples, called *frame size*. With a frame size of 1024 (which is a typically used value), the FFT results in 512 linearly distributed complex frequency values with their amplitude and relative delay (phase). These values can be investigated or modified and then the (original or changed) signal can be reconstructed using the inverse FFT.

## 1.3 Previous work

Relatively little effort has been devoted to the design of scalable rendering strategies for 3D audio, which can provide level-of-detail selection and graceful degradation. Also, the concept of "saliency" which is required for this is not well-defined. In this section a short overview of the previous work most relevant to this problem will be given.

### 1.3.1 Audio Saliency

The first question coming up when dealing with progressive, level-of-detail selective audio processing is: How to select and discard the data which is not important? What is the important data in sound? Which aspects / parts of the sound are better noticed than others? What is audio saliency?

The easiest way to define this seems to be the *energy* of a sound - the amplitude of the signal. In

perceptual audio coding [PS00] the *power spectrum distribution* (PSD) is used which measures the energy present in a number of frequency sub-bands at each time frame. Comparing the PSDs of sounds would potentially give a good approximation which sound is the most salient in a mixture if human perception of sound intensity was frequency independent. But this is not the case; imagine a 1kHz tone compared to a 50Hz tone with the same volume. 50 Hz are perceived much more gently (see the Equal-loudness contours [ISO03] for more details). The energy metric would assign the same saliency to both. Also, the pure (source) energy does not take into account the distance to the listener $r$, the propagation delay $delta = r/c$, and a frequency dependent attenuation $A$, obtained from the filters used to modify the source signal (e.g. occlusion and scattering).
To come up with a solution to this problem, the *binaural loudness estimation* was introduced in [TGD04] as a saliency estimation. It was used to define perceptual importance for deriving masking thresholds and clustering weighting. Loudness values for the left and right ear could be obtained from the *sound pressure levels* at each ear using the model of [JBB97]. These are defined as:

$$P_t^k(f) = Spat(S_k) \times \sqrt{PSD_{t-\delta}^k(f)} \times A_t^k(f)/r, \tag{1.1}$$

where $Spat(S_k)$ is a frequency and direction dependent attenuation due to spatial rendering like HRTF processing.

A completely different approach was the recent adaptation of visual saliency maps [IKN98] to audio by Kayser et al. [KPLL05] thus resulting in an *auditory saliency map* depicted in Figure 1.2.
Here, the source signal is transferred to a fourier domain representation and converted into a two dimensional image, called "intensity image". Various important features are extracted at different scales and therefore image pyramids are created for intensity, frequency contrast and temporal contrast. After this step the approach is similar to the visual saliency map method [IKN98] - where intensity, colors and orientation maps were created: For each feature map the different scales are compared using center surround differences. Together with a following normalization step the method favors maps which contain view high peaks. The image pyramids are then collapsed across scales resulting in three so-called "conspicuity maps" which are combined again, yielding the final saliency map.
Kayser et al. states that the auditory saliency map approach "captures essential aspects of human judgements of auditory saliency" [KPLL05].

### 1.3.2 Encoding and rendering of spatial auditory cues

Progressive spatial sound encoding techniques can be roughly subdivided in two categories.

A first category is based on a physical reconstruction of the wavefield at the ears of the listener. For instance, several approaches have been proposed to perform progressive binaural rendering [Bla97] by decomposing HRTFs on a set of basis functions through principal component analysis [CVH95, LJGW00, JW06]. HRTF, "*Head Related Transfer Function*", is a filter which describes how a source audio signal - defined by

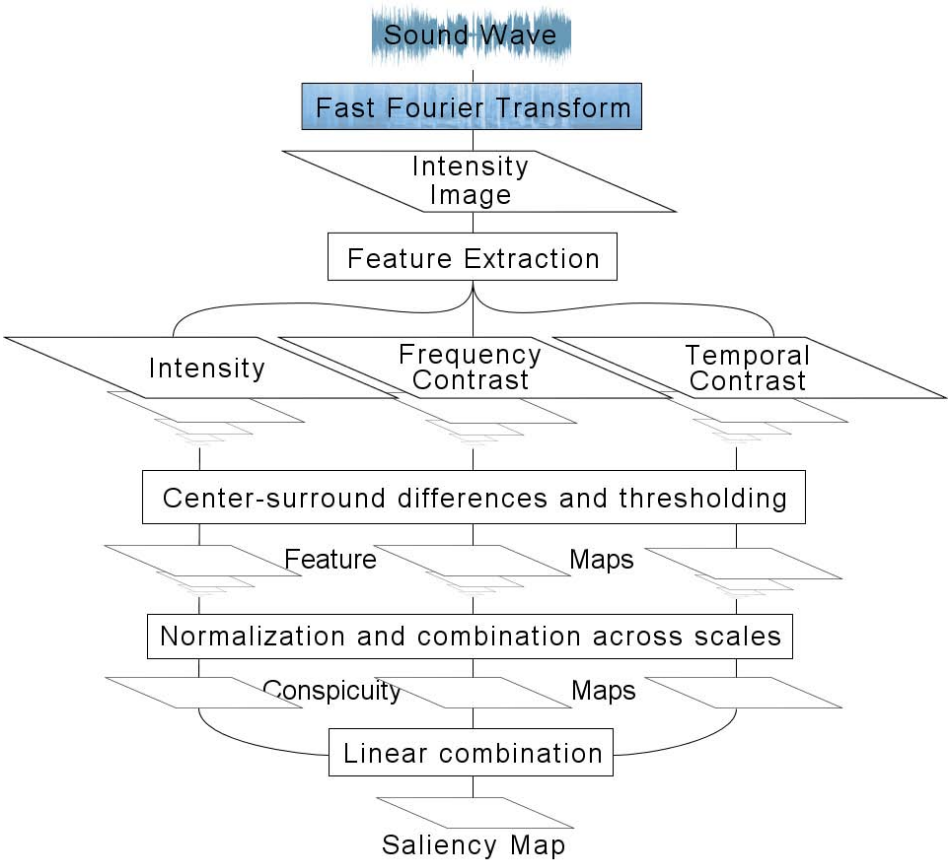


Figure 1.2: Overview of the auditory saliency map method

frequency and location - is modified by the head, torso and ear of a specific person. In nature this modification (evoked by reflection and diffraction) helps us to localize a sound. By filtering a mono input sound with HRTF, the listener can get the impression of surround sound while only using headphones. In this case, level-of-detail can be controlled by the number of eigenfilters used for the decomposition. This technique allows for efficient binaural 3D-audio processing but is limited to this unique restitution strategy. Alternative approaches decompose the incident wavefield on a set of spatial / directional basis functions (e.g., spherical harmonics) [MM95]. Such representations have the advantage of supporting level of detail (by truncation of the decomposition coefficients) and are relatively independent of the restitution setup. They can be decoded over a variety of loudspeaker configurations and also converted to binaural format for headphone listening [JLP99]. However, accurate localization usually requires high order decomposition and, consequently, encoding a large number of audio channels, which has historically limited the use of these approaches.

Another category performs world-space compression of positional cues by clustering nearby sound sources and using a unique representative position per cluster for spatial audio processing [Her99, WS04,

TGD04]. Wand et al. [WS04] propose to group sources in a hierarchical spatial data structure as used for point-based rendering. However, their approach is very efficient only in the case of fixed sources. Tsingos et al. [TGD04] recently introduced a clustering algorithm driven by a loudness-weighted geometrical cost-function (see Sec. 3) which accounts for fine-grain temporal variations in the source signals. They also use precomputed descriptors (e.g., loudness and tonality) to sort the sources by decreasing importance and perform a greedy culling operation by evaluating auditory masking at each time-frame of the simulation. Inaudible sources can then be safely discarded. The culling stage is usually very efficient and results in a significant decrease of the number of sound sources to cluster and process. Although this approach was found to perform well for environments containing a few hundred sources, it is unclear that it scales well to larger numbers of sources and clusters due to the cost of the proposed clustering algorithm which, in their case, implies a near quadratic number of evaluations of the cost-function.

### 1.3.3 Scalable audio processing

Fouad et al. [FHB97] propose a level-of-detail progressive audio rendering approach in the time-domain; by processing every n-*th* sample, artifacts are introduced at low budget levels. Wand and Straßer [WS04] introduce an importance sampling strategy using random selection, but ignore the signal properties, thus potentially limiting the applicability of this method.

In recent years, several contributions have been introduced that aim to bridge the gap between perceptual audio coding and audio processing in order to make audio signal processing pipelines more efficient. A family of approaches proposed to directly process perceptually-coded audio signals [LS97, LS99, DDS02, Tou00, TEP04] yielding faster implementations than a full decode-process-recode cycle. Although they are well suited to distributed applications involving streaming over low-bandwith channels, they require specific coding of the filters and processing. Moreover, they cannot guarantee efficient processing for a mixture of several signals, nor that they would produce an ”optimal” result. Other approaches have explored how to extend these approaches by concurrently prioritizing subparts of the original signals to process to guarantee a minimal degradation in the final result [GLT, Tsi05, KT02]. Most of them exploit masking and continuity illusion phenomena [KT02] to remove entire frames of the original audio data in the time-domain [GLT] or, on a finer scale, process only a limited number of frequency-domain Fourier coefficients [Tsi05].

### 1.3.4 Crossmodal studies

While the primary application of 3D audio rendering techniques is simulation and gaming, no spatial audio rendering work to date evaluates the influence of combined visual and audio restitution on the required quality of the simulation. However, a vast amount of literature in neurosciences suggest that cross-modal effects, such as ventriloquism, might significantly affect 3D audio perception [HWaBS$^+$03, AB04]. This effect tells us that in presence of visual cues, the location of a sound source is perceived shifted toward the visual cue, up to a certain threshold of spatial congruency. Above this threshold, there is a conflict between the perceived sound location and its visual representation and the ventriloquism effect no longer happens.

The spatial window (or angular threshold) of this effect seems to depend on several factors (e.g. temporal synchronicity between the two channels and perceptual unity of the bimodal event) and can vary from a few degrees [LEG01] up to 15° [HWaBS+03].

This kind of effect could lead us to think that we are more tolerant to localization errors when we have visual clues (i.e. in the viewing frustum) than when we don't have any visual information (outside the viewing frustum). This would lead to an algorithm which tend to cluster together more easily sound sources that are in the viewing frustum.
In an other way, one could think that since we look at the sources, having them auditory displaced could annoy us whereas for the invisible ones, the localization doesn't need to be precise. This would lead us to an opposite algorithm which favour a larger number of clusters in the viewing frustum.

## 1.4 The sound processing engine

Figure 1.3 shows a general overview of the designed audio rendering pipeline. In a preprocessing step, the FFT representation for every input sound file is computed and stored in a custom file format as well as some descriptors for each sound which are in particular:

- the spread enery for 4 frequency sub-bands [PS00],
- the tonality, calculated as a spectral flatness measure [PS00]; tonality is a descriptor in [0,1] encoding the tonal (when close to 1) or noisy (when close to 0) nature of the signal,
- the loudness [TGD04] (like described in Sect. 1.3)
- the pinnacle value; this value indicates how many frequencies are needed to reconstruct the signal with a decent quality.

These values could also be computed in realtime if needed (e.g. for synthesized sound) but if not, they should be prestored for maximum efficiency.

After loading the precalculated data, the audio rendering pipeline can be decomposed into four main steps which are repeated for each audio frame.

### 1.4.1 Sound source Masking

When playing hundreds of sounds concurrently, the human listener cannot hear each one anymore because some sounds mask others. The masking step takes advantage of this fact by figuring out which sounds are audible in the final mix. To do this, the perceptual saliency of all sources in the scene is evaluated based on their *spread energy* [PS00]. This value is based on the idea of interband masking. One masker in a band can have significant effect on masking thresholds in others.

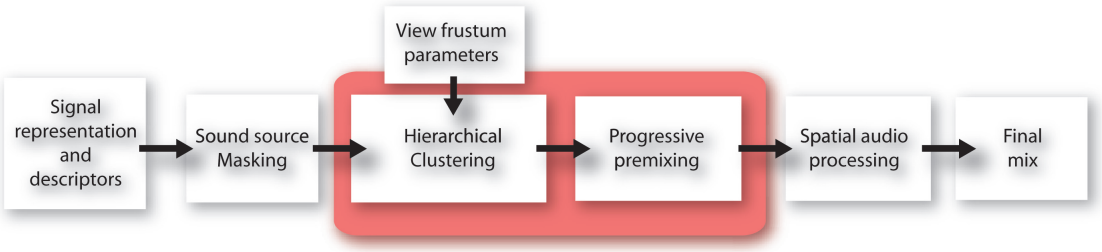


Figure 1.3: Overview of the overall sound rendering pipeline.

In this step the sources are first sorted by their spread energy at the particular time frame. Then perceptually inaudible ones are culled by progressively adding more salient ones into the mix until their combination hides the remaining ones. This is also depicted in the pseudo - code (see listing 1.1).

Listing 1.1: Sound source Masking

```
1  Mmix = 200
2  Pmix = 0
3  T = 0
4  PtoGo = sum of the spread energy of all sounds
5  while (dB(PtoGo) > dB(Pmix) - Mmix) and (PtoGo > ATH) do
6    set signal k to not-masked
7    PtoGo -= energySpread[k]
8    Pmix += energySpread[k]
9    T += energySpread[k] * T[k]
10   Tmix = T / Pmix
11   Mmix = 21 * Tmix + 6
12   k++
13 end while
```

Here, *ATH* is the *absolute threshold of hearing* [ZF99] while in the applications for this work a constant threshold of -35 dB was used. *T[k]* is the tonality index [PS00] of sound k.

Masking is very fast while also being very effective. This step may easily safely discard 50% of the sources without making it possibly to hear any difference.

### 1.4.2 Hierarchical Clustering

In this second step, all remaining sound sources are grouped together into clusters by assigning a representative position to each source - which is its cluster's centroid. The clustering algorithm is a dynamic farthest-first traversal of the point set of sound sources based on the Hochbaum-Shmoys heuristic [HS85].

While clustering, sound sources which are more salient are given more accuracy in position by weighting the importance of each source by its *loudness*.

In this work, the original sound clustering algorithm [TGD04] is improved by introducing two different recursive approaches. Furthermore a pilot study was performed to include visual aspects of a scene into sound clustering by changing the weighting of importances of sound sources by visual clues.

### 1.4.3 Progressive premixing

After grouping each audible source into a cluster and given a representative position, every cluster is premixed, i.e. an equivalent source signal is generated for each cluster. This step also includes operations on the original audio signal like filtering, re-sampling, etc. which differ for each source.

To optimize the performance of this step, only the most important frequency data is taken for the mixture which is determined by the *pinnacle value*.

### 1.4.4 Spatial audio processing

Arriving at this step, one audio signal per cluster is available, having assigned the position of the cluster centroid. Due to the fact that spatial audio processing can be very expensive regarding performance, it is done at the very end of the pipeline where there are only few signals to process. In the application for this work, HRTF (Head Related Transfer Function) processing for binaural rendering [Møl92] is used. Another possibility would be e.g. to feed the audio hardware channels with the cluster signals by using DirectSound 3D or similar APIs.

# Chapter 2

# The central processing unit

Because the audio approach described in this work runs on the CPU (*central processing unit*), it is important to know how this hardware is operating.

CPUs designed according to the *von Neumann architecture* execute a series of instructions called a *program*. The program is mostly represented as a sequence of integers stored in some kind of memory, called program memory. To execute the program, four main steps have to be passed: *fetch*, *decode*, *execute* and *writeback*.

In the *fetch* step an instruction is taken from the program memory by using the *program counter* (PC) to find the right position in memory. In fact, the program counter defines the execution location in the current program and can thus be used to compute the position in memory the next instruction can be found. In todays CPUs the former - relatively slow - program memory is replaced by various caches which are accelerated even more by pipeline architectures. Thereby following instructions can be fetched while decoding and executing others. After an instruction is fetched, the program counter is increased by the instruction's length in memory units. Consequently the program counter then points to the next instruction.

To be able to know what to do, the previously fetched instruction has to be split up into meaningful pieces. This is done in the *decode* step (see Fig. 2.1). Usually, an instruction comprises the *opcode* which defines the operation to be executed. The remaining pieces are parameters for the operation describing how it should be done, e.g. the parameters of an addition operation. These operants can be given as a constant value (also called immediate instruction) or as a reference, i.e. the location of a value (a memory adress - defined by several adressing modes - or a register - which is in principle a very small memory on the CPU which provides extremely fast access). For the decoding of instructions the *instruction set architecture* (ISA) helps as a kind of overall "dictionary" to interpret the numerical values fetched from the program memory. Unlike in older CPUs the ISA can now be programmed, too, and is thus changable even after the manufacturing of the CPU.

Having the instruction fetched and decoded, the CPU implements the *execute* step. An instruction is executed by connecting needed components to fulfill the desired operation. For example the ALU (arithmetic logic unit) is connected if an arithmetic operation has to be accomplished. This component then provides a

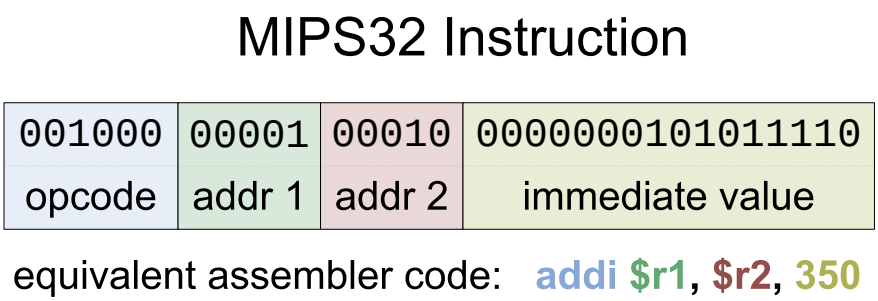


Figure 2.1: Decoding of a MIPS32 instruction. The value stored in $r2 will be added to the immediate value 350 and the result will be stored in $r1. In case of an overflow, $r1 will not be altered but the overflow flag will be set.

set of inputs and outputs which are used for the calculation. If operations fail or need to signalize something special which is not included in the standard output signals, the unit can set a flag in a register which can then be used by higher level layers.

The following *writeback* step just writes back the results of the operation to memory. The type of memory depends on the specific architecture of the CPU and also on the program itself. The results may be written to registers - which are very fast but small - or e.g. to main memory - which is slow but large. Some operations only modify the program counter (like jump operations, used for functions, loops and conditions) and don't need any memory for writeback other than the program counter.

After the writeback step, the process is finished, the result has been calculated and stored in memory or a task has been executed. Now, the procedure repeats with the next instruction the program counter is pointing at. In todays complex and high performance CPUs, multiple instructions can be fetched, decoded and executed concurrently ("pipelining") introducting some difficulties in implementing the process cycle however resulting in significant speed-ups. Acceleration of CPUs is also not only done by increasing the clock rate but also by introducing a number of caches and using them additionally. The caches also differ in the category of usage: There are specialized ones which are e.g. only responsible for one specific step in the pipeline. There are multi-level caches which provide access to faster and smaller caches as well as to slower and bigger ones. And there are caches called predictors and storing data which can be costly to compute and thus isn't needed to be recomputed if already calculated and stored.

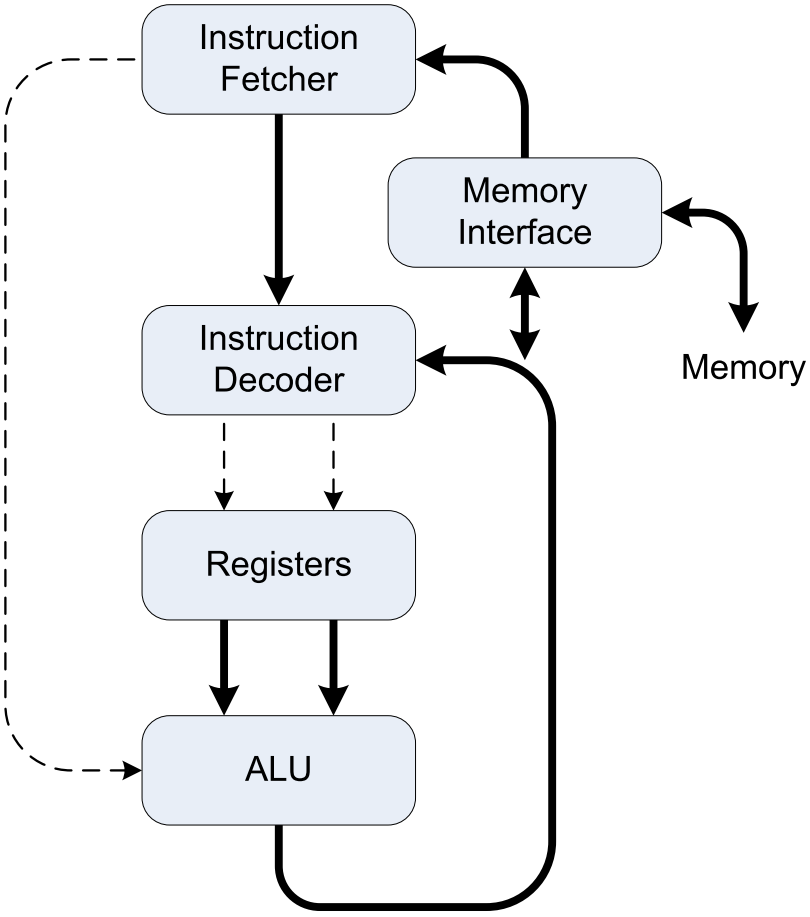


Figure 2.2: A block diagram of a simple CPU.

# Chapter 3

# Optimized recursive clustering

In previous work [TGD04], large numbers of sound sources are dynamically grouped together in clusters, and a single new *representative* point source is created. While this approach allows the treatment of several hundred sound sources on a standard platform, it does incur some computational overhead, which becomes a potential bottleneck as a function of the number of input sources / target-clusters and the available computational budget. To resolve this limitation, a new recursive clustering algorithm is presented. For improved efficiency and accuracy two recursive algorithms are proposed: One with a fixed budget of clusters and one with a variable number of clusters. The fixed budget approach allows to set a fixed computational cost for 3D audio processing and is also useful to address a fixed number of 3D-audio hardware channels when available for rendering. The variable number of clusters dynamically adjusts the computational cost to obtain the best trade-off between quality and speed in each frame, given a user-specified error threshold.

A variant of this algorithm is also discussed which has been included in the commercially available game "Test Drive Unlimited" by EdenGames / ATARI.

## 3.1 Original method

In the method of [TGD04], sources are grouped together by using a clustering algorithm based on the Hochbaum-Shmoys heuristic [HS85]. First, this strategy selects $n$ cluster representatives between the $k$ original sources by doing a farthest-first traversal of the point set. The cost-function used is a combination of angular and distance errors to the listener of the candidate representative source with respect to the sources being clustered. An additional weighting term, based on each source's instantaneous loudness value, is used to limit error for loud sources. The following weighting factor is thus used:

$$d(C_n, S_k) = L_t^k \left[ \alpha log_{10} \left( \frac{||P_{C_n}||}{||P_{S_k}||} \right) + \frac{\beta}{2} \left( 1 - \frac{P_{C_n} \cdot P_{S_k}}{||P_{C_n}||||P_{S_k}||} \right) \right] \tag{3.1}$$

This factor is the weight of the source $S_k$ with respect to the cluster representative $C_n$ where $L_t^k$ is the loudness value of the source $S_k$ at time-frame $t$. $P_{S_k}$ and $P_{C_n}$ are the positions of the sound source $S_k$ and the cluster representative $C_n$ relative to the listener's position and orientation. $L_t^k$ ensures that the error will be minimized for perceptually important sources; thus sources that are far away are grouped in bigger clusters. To balance the influence of angle and distance, values of $\alpha = 2$ and $\beta = 1$ are typically used.

To ensure minimal spatial distortion, the position of the representative should try to preserve the perceived distance and direction to the listener. To this effect, the centroid is calculated in spherical coordinates $(\rho_{C_n}, \theta_{C_n}, \phi_{C_n})$ and shifted dependent on the sources' loudness values, as follows:

$$\rho_{C_n} = \frac{\sum_j L_t^j r_j}{\sum_j L_t^j}, \theta_{C_n} = \theta(\sum_j L_t^j S_j), \phi_{C_n} = \phi(\sum_j L_t^j S_j) \tag{3.2}$$

where $r_j$ is the distance from a source $S_j$ in the cluster to the listener.

## 3.2 Recursive method

As said before, in the original clustering algorithm - which works iteratively - the computational overhead for a large number of sources is huge and can become a bottleneck because it doesn't scale well. To overcome this problem of a nearly quadratic quantity of calculations of the cost-function, a recursive approach is introduced. This ensures to scale well also with a large number of sources.

### 3.2.1 Recursive fixed-budget approach

In a first pass, the original clustering algorithm is run with a target number of clusters $n_0$. In each subsequent pass, every generated cluster gets divided into $n_k$ clusters. The total budget of clustering is thus $\prod_k n_k$. The original clustering algorithm can be considered as a special case where only $n_0$ is set. In the tests for this work, a two-level recursion is typically used.

By using this approach, a fixed budget can be defined which will be used for clustering. Thus, the computational cost for clustering in each frame is known in advance and can be set to fit the needs.

### 3.2.2 Variable cluster budget

This approach dynamically allocates the number of clusters in realtime. This is especially useful for scenes where sounds are frequently changing during time in shape, energy as well as in location. The algorithm then flexibly allocates the required number of clusters; thus clusters are not wasted where they are not needed.

First, every sound source which has not been masked [TGD04], is put in one cluster which is then recursively split into two until an appropriate condition is met. In every recursion step the error in angle relative to the listener position is computed for each sound source in a cluster relative to its centroid. If the average angle error is below a threshold, cluster splitting is terminated. In the tests, it was found that a 25° threshold value proved satisfactory.

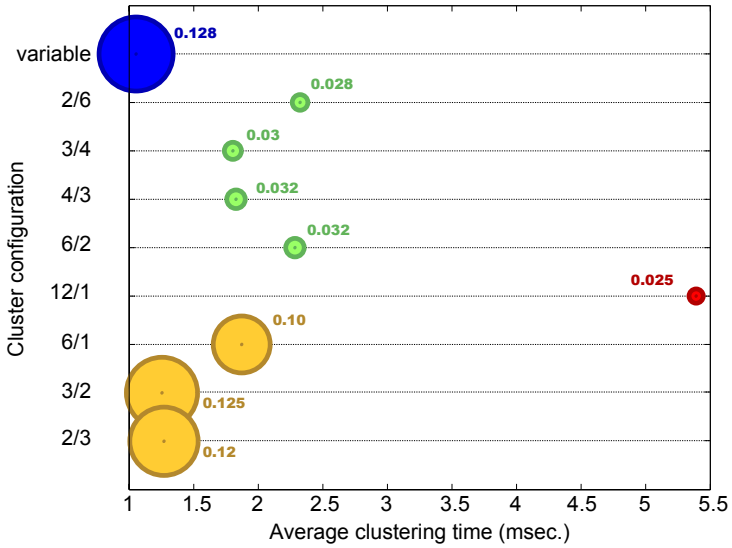


Figure 3.1: Benchmarks for hierarchical clustering of 800 sources using different 6 and 12 cluster configurations. The average clustering error is also displayed (denoted by the size of the circles and the number next to it). Note the significant speed-up compared to the non-hierarchical algorithm for the 12 cluster configurations (in red) while the errors remain similar.

## 3.3 Quantitative error / performance comparison

A quantitative comparison was performed between the previous clustering approach of [TGD04], and the two proposed recursive clustering methods. To obtain performance and error values, several thousand tests were run with random configurations of 800 sound sources, using the different algorithms and measuring the time and the error (in terms of distance and angle) for each method.

Figure 3.1 shows the results of the tests for fixed budgets of 12 and 6 clusters using different two-level subdivision strategies. For instance, line 3/4 corresponds to a clustering using 12 clusters (3 top-level clusters recursively refined into 4). The line 12/1 corresponds to the previous clustering approach of [TGD04] where all 12 clusters are top level.

As it can be seen, the performance of the recursive approaches are clearly better than the direct algorithm. For the same final budget, the 3/4 and 4/3 configurations appear to be better choices in terms of speed. As expected the error is larger for hierarchical clustering since it leads to less optimal cluster placement. However, as the number of clusters grows this effect tends to disappear.

The variable cluster method is faster on average. However, with the current settings it also created fewer

clusters (6.6 clusters created on average) and, as a consequence, has higher average error. Interestingly, the peak number of clusters created by the variable method is 22, which underlines the flexibility and adaptability of the approach.

## 3.4 Implementation in a commercial game

The audio clustering technique was used in the development of the commercially available computer game *Test Drive Unlimited*. In this car racing game, the sound of each racing vehicle is synthesized based on numerous mechanical quantities. The sound emitted by each wheel is controlled by 20 physical variables while 8 variables control the engine / transmission sounds. Four additional values control aerodynamic phenomena. These are used for real-time control and playback of a set of pre-recorded sound samples. All sound sources are then rendered in 5.1 or 7.1 surround sound. For implementation on the *XBOX360*, Eden Games adopted a variant of the recursive variable budget technique described above. In particular, a budget of 8 clusters was used at each recursion level. If the quality criterion is not met for this budget, a local clustering step is applied in each cluster. However, the local nature of clustering resulted in audible artifacts from one frame to the next, because of sources moving from one cluster to another. To resolve this problem, sources are ordered by perceptual priority, and the most important ones will prefer to be clustered with the cluster of the previous frame, effecting a form of temporal coherence.

Despite this improvement, extreme cases still presented some difficulty, notably the case of a car crashing into an obstacle. In this case, the physics engine generates numerous short-lasting sound sources in the immediate neighbourhood of the vehicle. Temporal coherence is thus useless in this context. The solution to this issue is to apply a separate clustering step to the sources generated by physics events; this results in more clusters overall, but resolves the problems of quality.

A snapshot of *Test Drive Unlimited* with a visualisation of the sound source clusters superimposed in red, is shown in Figure 3.2.

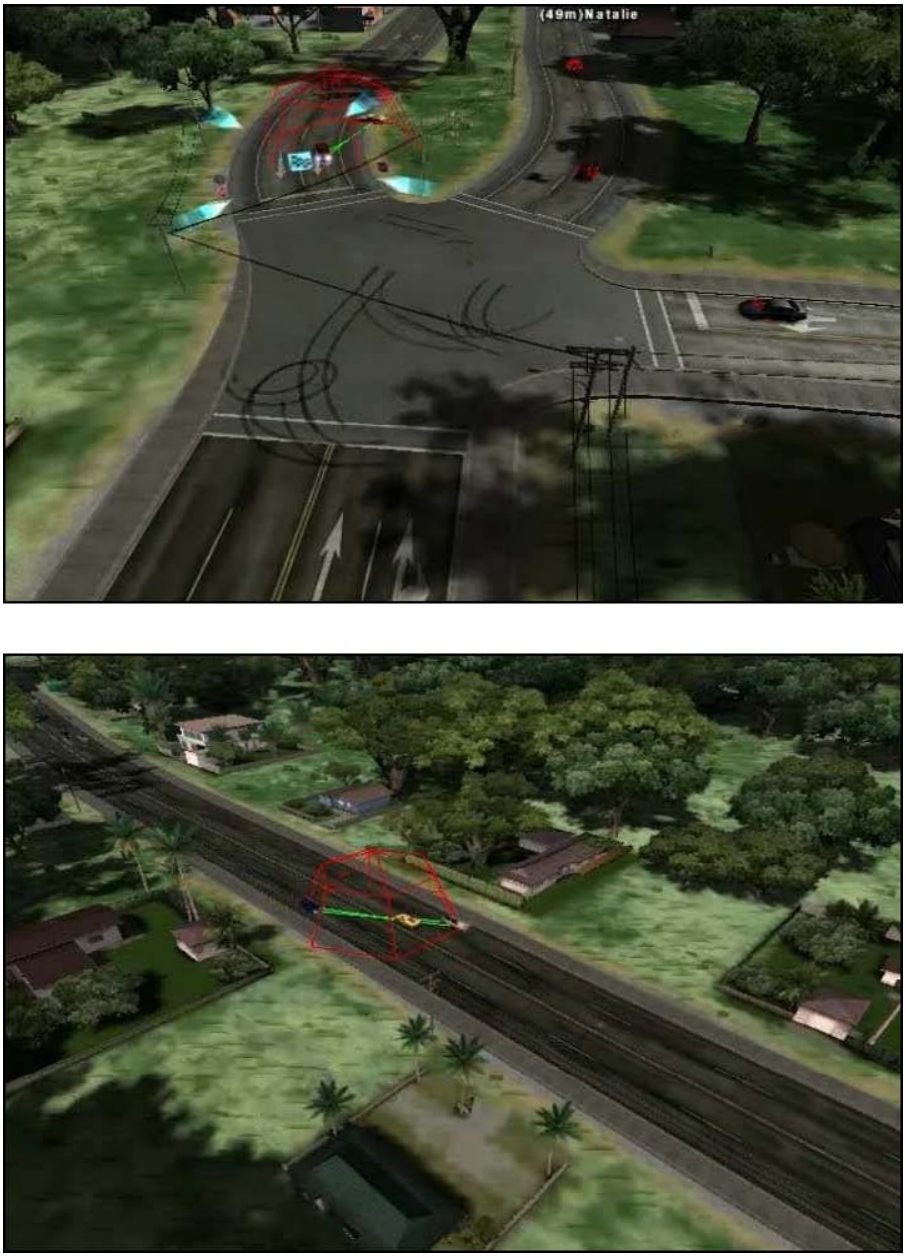


Figure 3.2: Sound source clustering in the *Test Drive Unlimited* engine. Red wireframe spheres are clusters. ©Eden Games-ATARI 2006.

# Chapter 4

# Scalable perceptual premixing

In order to apply the final spatial audio processing to each cluster (see Figure 1.3), the signals corresponding to each source must first be *premixed*. The premixing stage can be as simple as summing-up the signals of the different sources in each cluster. In addition, a number of audio effects usually has to be applied on a per-source basis. Such effects include filtering (e.g., distance, occlusions), pitch-shifting (e.g., Doppler effect) or other studio-like effects [Zöl02]. Hence, when large numbers of sound sources are still present in the scene after auditory culling (masking) or for systems with limited processing power, the cost of this stage can quickly become the bottleneck of the audio rendering pipeline.

In this section, a progressive signal processing technique is proposed that is based on perceptual information and can be used to implement scalable per-source processing. This work is based on the approach of [Tsi05] and is briefly summarized in Figure 4.1.

The approach uses a specific time-frequency representation of audio signals. At each time-frame (typically 1024 samples at 44.1KHz), the complex-valued coefficients of a short-time Fourier transform (STFT) are precomputed and stored in decreasing modulus order. In real-time during the simulation, the algorithm prioritizes the signals and allocates a number of coefficients to process to each source so that a predefined budget of operations is respected. In [Tsi05], this importance sampling stage is driven by the energy of each source at each time-frame and used to determine the cut-off point in the list of STFT coefficients. However, using only energy for importance sampling leads to sub-optimal results since it does not account for the sparseness of the representation obtained for each signal. For instance, a loud tonal signal might require fewer coefficients than a weaker noisier signal for transparent reconstruction (Figure 4.2). An additional weighting term measuring the efficiency of coding for each frame of input signal was thus proposed for budget allocation.

In the following, an improved budget allocation strategy is introduced. Furthermore, the results of a perceptual quality study are presented, aimed at evaluating the novel technique and several possible importance metrics used for prioritizing source signals.

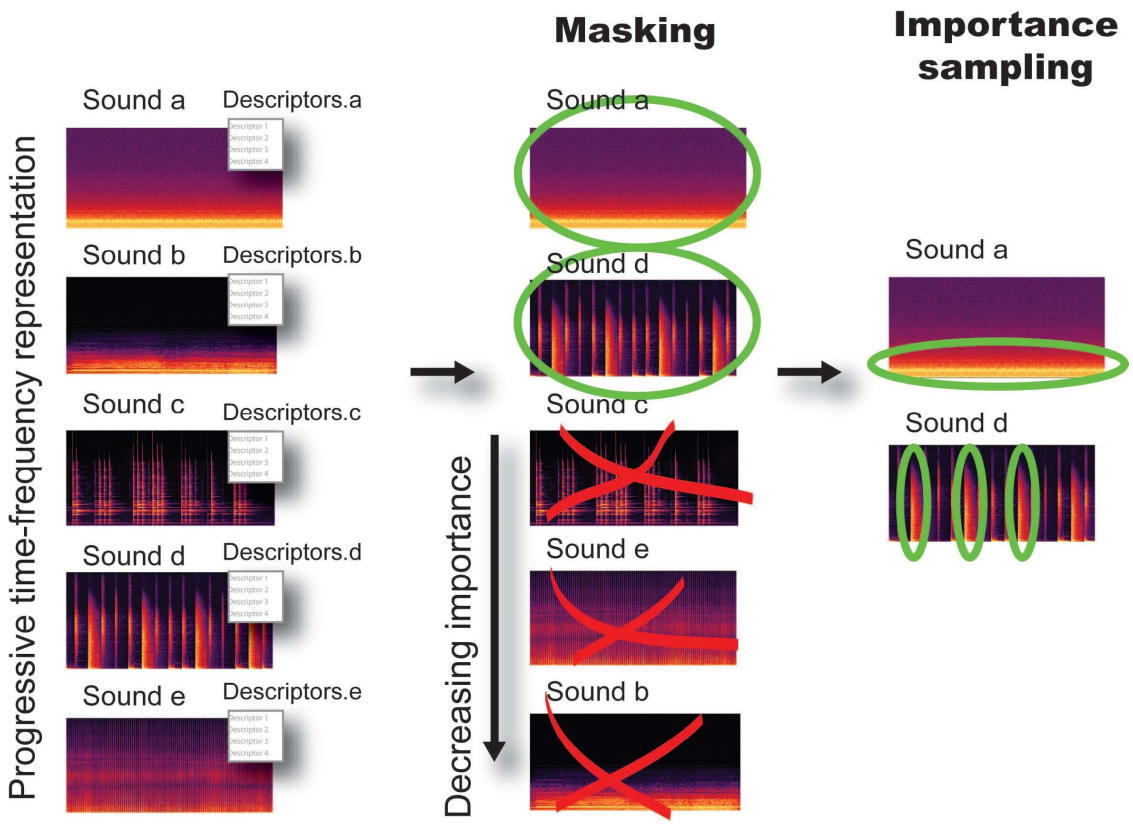


Figure 4.1: Overview of the progressive perceptual premixing.

## 4.1 Improved budget allocation

### 4.1.1 The importance value

To distribute the budget number of operations per audio frame most suitable, sound sources have to be ranked by their saliency. More salient sources are to be reconstructed in better quality and thus more operations should be used for them due to quality degradation when using viewer. E.g. reconstructing a salient sound with bad quality may be noticed quite easily while reconstructing a non-salient one with bad quality will probably be unobservable.

The saliency of a sound is determined by the importance value which can typically be the energy or the loudness of the signal as proposed in [TGD04, Tsi05]. In this work, it was also experimented with an *audio saliency map* value derived from the model recently proposed in [KPLL05]. This model is very similar to the visual saliency maps [IKN98] but it is applied on a time-frequency domain representation of audio signals (see Sect. 1.3.1). In this case, after computing the auditory saliency map, the saliency values were integrated over a small number of frequency subbands (typically 4 were used on a non-linear frequency scale).

By using one of these importance metrics, every sound source gets assigned an importance value in each audio frame which is then used in the iterative importance sampling to distribute the available budget.

### 4.1.2 The pinnacle value

Contrary to [Tsi05], the improved budget allocation pre-computes the explicit number of STFT coefficients necessary to transparently reconstruct the original signal. This value, that is called *pinnacle*, is pre-computed for each time-frame of input audio data and stored together with the progressive STFT representation. To compute the pinnacle value (see listing 4.1), the STFT coefficients are first sorted by decreasing modulus order. The energy of each coefficient is integrated until a threshold of at least 99.5% of the total energy of the frame is reached and the number of corresponding coefficients is greater than $(1 - tonality) \cdot N/2$, where $N$ is the total number of complex Fourier coefficients in the frame (i.e. the *frame size*) and $tonality \in [0, 1]$ is the tonality index of the frame [PS00, KAG+02]. This index is close to 1 for tonal signals and drops to 0 for noisier signals.

Listing 4.1: Computation of the pinnacle

```
1  function computePinnacle
2    sort all frequencies F by its energies in descending order
3    sumAll = 0
4    sumBorder = 0
5    for (i = 0 to frameSize / 2 - 1) do
6      sumAll += F[i].energy
7      sum[i] = sumAll
8    end for
9    sumBorder = sumAll * 0.995
10   for (i = 0 to frameSize / 2 - 1) do
11     if (sumi > sumBorder) and
12         (i > (1 - tonality) * frameSize / 4) then return i
13   end for
14   return frameSize / 2 - 1
15 end function
```

The result is a value indicating how many of the most energetic frequency bins are needed for the signal to be reconstructed in decent quality. The metric was tested extensively with various types of sound (white, pink and brown noise, voice, music, etc.) and gave satisfying results.

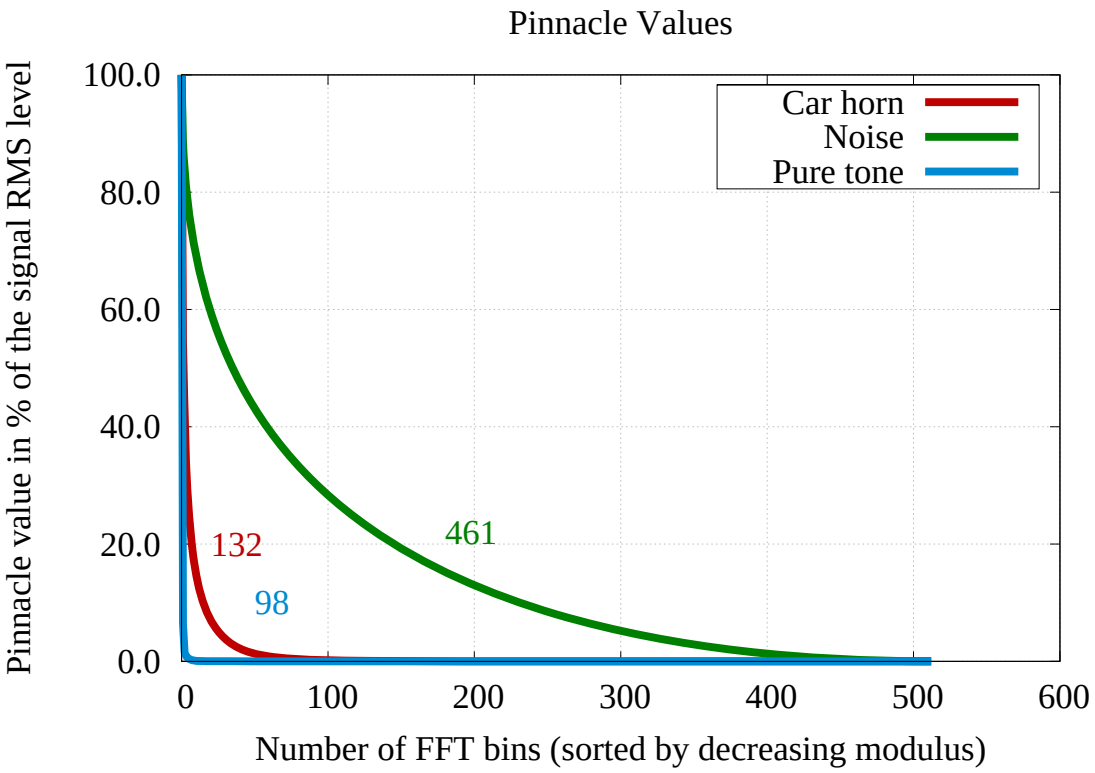


Figure 4.2: The pinnacle value as a function of target FFT bins. Tonal signals require fewer coefficients than noisier signals since their Fourier representation is much sparser. The value next to each curve corresponds to its pinnacle value, used to obtain the needed amount of frequency bins at each frame of input signal for transparent reconstruction.

### 4.1.3 Iterative importance sampling

After having calculated the indication values *importance value* and *pinnacle value*, they are used for distributing the spendable budget to the sound sources in the premixing step. A constant number of arithmetic operations is assumed to be required for each complex STFT coefficient. Hence, fitting a budget number of operations for our pipeline at each processing frame directly amounts to selecting a budget number of coefficients for each frame of input sound signal. It can be taken advantage of pre-storing the FFT in decreasing energy order by directly processing the $n_i$ first coefficients for each input signal $s_i$, so that the sum of all

$n_i$s does not exceed the total budget $N$. To determine the $n_i$s, each signal first gets assigned an importance value like described above.

Then, every input signal gets assigned a number of bins $n_i$ relative to its importance as follows:

$$n_i = I_i / \sum_i I_i \cdot \mathrm{N}, \tag{4.1}$$

where $I_i$ is the importance of the source $i$. Ideally, $n_i$ should be smaller than the signal's pinnacle value to avoid suboptimal budget allocation as can be the case with the approach of [Tsi05]. To avoid such situation, all remaining coefficients above pinnacle threshold are re-assigned to the remaining signals, that do not already satisfy the pinnacle value criterion because the importance value of these ones was too low. This allocation is again relative to the importance values of the signals:

$$n_i += I_i / \sum_i I_i \cdot \mathrm{extraCoeffs}, \tag{4.2}$$

where $extraCoeffs$ are the remaining coefficients of the first pass. The relative importance of each remaining signal is updated according to the reallocation of coefficients. If the budget is high enough, the process is iterated until all signals satisfy the pinnacle criteria or receive a maximal number of coefficients.
For all the details of the iterative importance sampling, see listing 4.2.

Listing 4.2: Iterative importance sampling

```
1  calculate percental importance of all audible sounds from its relative
2    importanceValues
3  numberOfFreqToGive is set by user (= budget number of frequencies)
4  set all sounds unsatisfied
5  numberOfUnsatisfied = numberOfSounds
6  while (numberOfFreqToGive > 0 and numberOfUnatisfied > 0) do
7    remainingAmount = 0
8    percentageLeft = 0
9    for all audible sounds i not satisfied do
10     frequencyAmount[i] += percentageValue[i] * numberOfFreqToGive
11     if (frequencyAmount[i] > pinnacleValue[i] ) then
12       remainingAmount += frequencyAmount[i] - pinnacleValue[i]
13       frequencyAmount[i] = pinnacleValue[i]
14     end if
15     if (frequencyAmount[i] > frameSize / 2) then
16       remainingAmount += frequencyAmount[i] - frameSize / 2
17       frequencyAmount[i] = frameSize / 2
18     end if
19     if (frequencyAmount[i] == pinnacleValue[i]) or
```

```
20        (frequencyAmount[i] == frameSize / 2) then
21       set sound i satisfied
22       numberOfUnsatisfied--
23       percentageLeft += percentageValue[i]
24     end if
25   end for
26   for all audible sounds i not satisfied do
27     if (percentageLeft < 1) then
28       percentageValue[i] /= (1 - percentageLeft)
29     else
30       percentageValue[i] = 1
31     end if
32   end for
33   numberOfFreqToGive = remainingAmount
34 end while
35
36 percentageValue of all sounds is taken again from its relative
37   importanceValues
38 set all sounds unsatisfied
39 numberOfUnsatisfied = numberOfSounds
40 while (numberOfFreqToGive > 0 and numberOfUnatisfied > 0) do
41   remainingAmount = 0
42   percentageLeft = 0
43   for all audible sounds i not satisfied do
44     frequencyAmount[i] += percentageValue[i] * numberOfFreqToGive
45     if (frequencyAmount[i] > frameSize / 2) then
46       remainingAmount += frequencyAmounti - frameSize / 2
47       frequencyAmount[i] = frameSize / 2
48     end if
49     if (frequencyAmount[i] == frameSize / 2) then
50       set sound i satisfied
51       numberOfUnsatisfied--
52       percentageLeft += percentageValue[i]
53     end if
54   end for
55   for all audible sounds not satisfied do
56     if (percentageLeft < 1) then
57       percentageValue[i] /= 1 - percentageLeft
```

```
    else
      percentageValue[i] = 1
    end if
  end for
  NumberOfFreqToGive = remainingAmount
end while
```

## 4.2 Quality evaluation study

To evaluate possible importance metrics and the pinnacle-based algorithm a quality evaluation study was conducted.

### 4.2.1 Experimental procedure

Seven subjects aged from 23 to 40 and reporting normal hearing volunteered for five different test sessions. For each session, a *Multiple Stimuli with Hidden Reference and Anchors* procedure was used (MUSHRA, ITU-R BS.1534) [SK00, EBU03, Int03]. Subjects were asked to simultaneously rank a total of 15 stimuli relative to a reference stimulus on a continuous 0 to 100 quality scale. The highest score corresponds to a signal indistinguishable from the reference. The reference stimuli were different mixtures of ambient, music and speech signals. In all cases, 12 of the 15 test-stimuli consisted of degraded versions of the mixture computed using the progressive mixing algorithm at various budgets (5%, 10% and 25% for music and ambient and 2%, 5% and 10% for speech), using the pinnacle-based technique, not using the pinnacle and using either loudness or saliency-based prioritization. In all cases, the processing is done using 32-bit floating point arithmetic and reconstructs signals at 44.1KHz. Two anchor stimuli, providing reference degradations, were also included. In this case, a downsampled 16KHz / 16-bit version of the mixture was chosen and a *mp3*-encoded version at 64Kbps. Finally, a hidden reference was included, too. Stimuli were presented over headphones. Subjects could switch between stimuli at any point while listening. They could also define looping regions to concentrate on specific parts of the stimuli. A volume adjustment slider was provided so that subjects could select a comfortable listening level. Figure 4.3 shows the corresponding interface.

### 4.2.2 Results

The study confirmed that the scalable processing approach is capable of generating high quality results using 25% of the original audio data and produces acceptable results with budgets as low as 10%. In the case of speech signals, for which the STFT representation is sparser, the algorithm could generate an acceptable mixture (avg. score 56 / 100) with only 2% of the original coefficients. As can be seen on Figure 4.4, the approach yields to significantly better results than a 16KHz reference signal (16KHz processing would correspond to a 30% reduction of data compared to our 44.1KHz processing). At 25% budget (or 10% in the case of speech), results comparable or better than the 64Kbps mp3-encoded reference are obtained. An

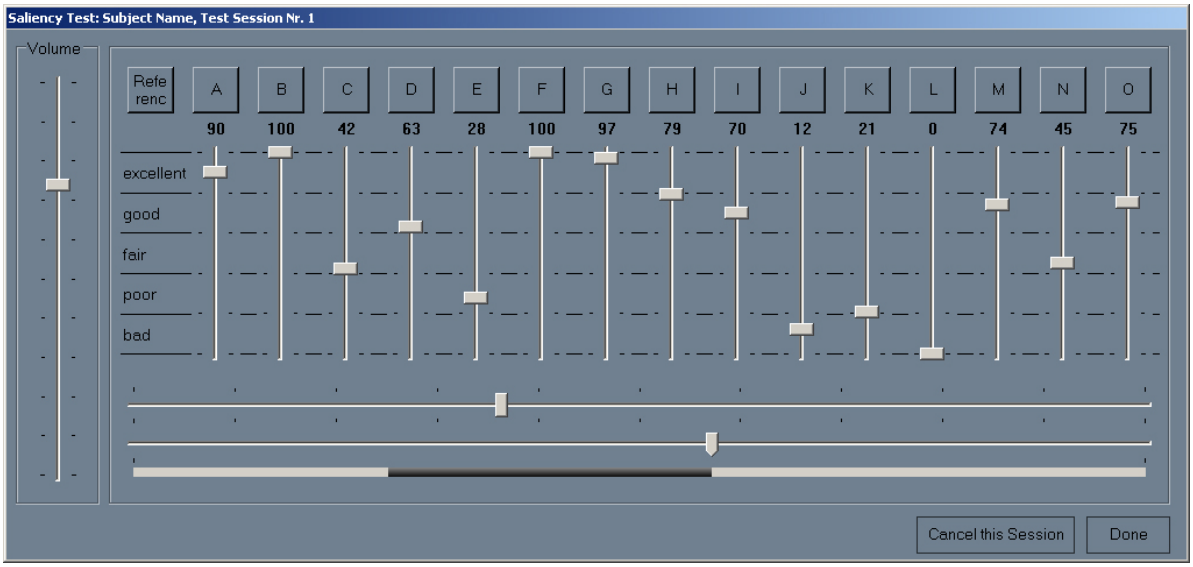


Figure 4.3: A view of the MUSHRA interface used for the evaluation of the scalable premixing.

analysis of variance (ANOVA) [How92] was performed on the results. As expected, the analysis confirmed a significant effect of the computing budget ($p<0.01$) on the quality of the resulting signal (Figure 4.5). It can be seen that the variation of perceived quality is not generally a linear function of budget, especially for more tonal signals (music, speech) which can be efficiently encoded until a sharp breakdown point is reached. Interaction between budget and importance metric was found to be significant ($0.05<p<0.01$). At low or high budgets, the two metrics lead to very similar results. However, for intermediate budgets, loudness-based prioritization improved perceived quality relative to the auditory saliency map - based alternative. Similarly, using the new pinnacle algorithm also leads to a slight improvement in the results, especially for cases where both tonal and noisier signals are present.

Noisier stationary sounds, which do not contain strong spectral features, usually receive a lower saliency map value although they might contain significant energy and require more coefficients to be properly reconstructed. This might explain why saliency map - based prioritization led to lower perceived quality in some cases.

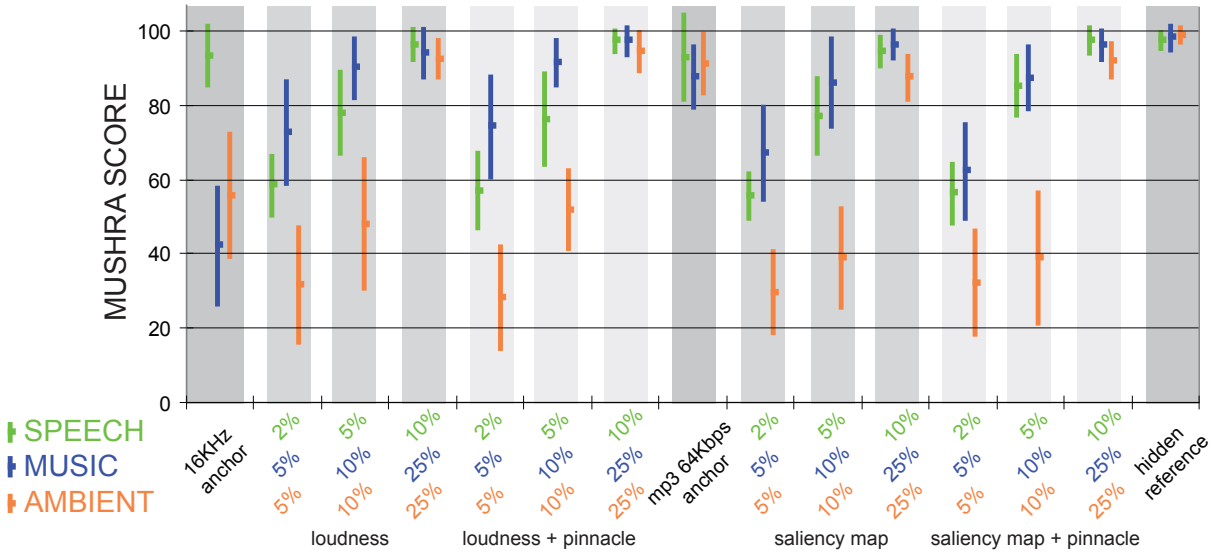


Figure 4.4: Average MUSHRA scores and 95% confidence intervals for the progressive processing tests.

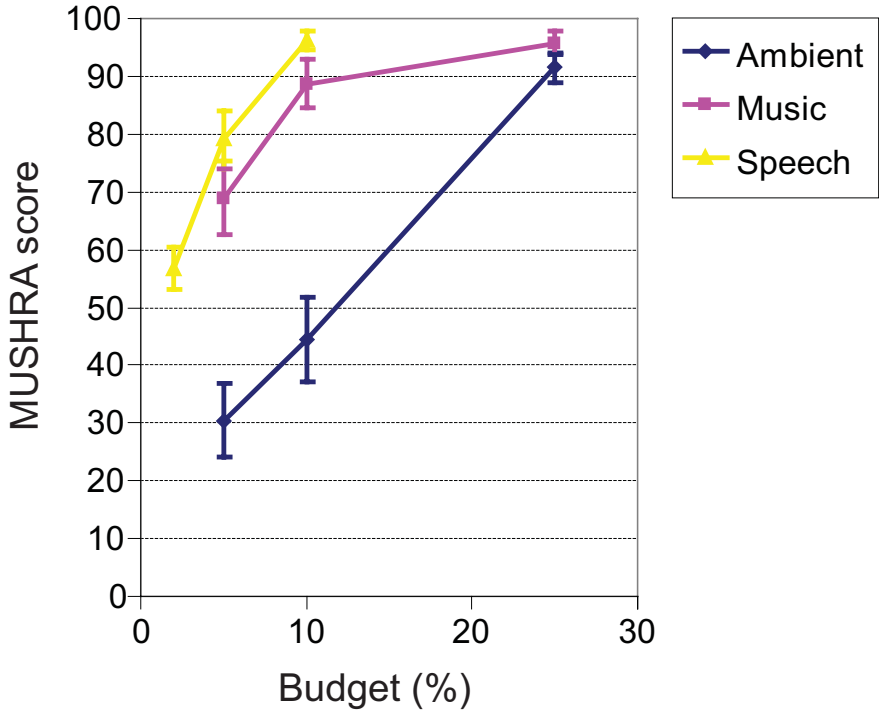


Figure 4.5: Average MUSHRA scores and 95% confidence intervals as a function of budget. Note how perceived quality does not vary linearly with the processing budget and also varies depending on the type (i.e., sparseness) of the sounds.

# Chapter 5

# Cross-modal effects for sound scene simplification

In the preceding sections, different aspects of audio rendering have been improved for complex scenes, without consideration for the corresponding visuals. Intuitively, it would seem that such interaction of visual and audio rendering should be taken into account, and play a role in the choice of metrics used in the audio clustering algorithm. A first attempt was presented in [TGD04], but was inconclusive presumably due to the difficulties with speech stimuli, which are generally considered to be a special case.

Research in ventriloquism (see Section 1.3), could imply that we should be more tolerant to localization errors for sound rendering when we have accompanying visuals. If this were the case, the weighting terms in the clustering algorithm could be changed to create fewer clusters for sound sources in the visible frustum. However, a counter argument would be that in the presence of visuals, we are more sensitive to localization, and we should favour more clusters in the viewing frustum.

The goal was to see whether some insight into this question could be provided with a pilot perceptual study. The next step was to develop and test an improved audio clustering algorithm based on the indications obtained experimentally.

## 5.1 Experimental setup and methodology

The following experimental setup was chosen to provide some insight on whether we need more clusters in the visible frustum or not.

The subjects are presented with a scene composed of 10 animated - but not moving - objects emitting "ecologically valid" sounds, i.e., a moo-ing sound for the cow, a helicopter sound for the helicopter, etc. (Figure 5.1).

There are two main conditions: audio only (i.e., no visuals) (condition *A*) and audio-visual (*AV*). Within each main condition there's a control condition, in which sources follow a uniform angular distribution, and


Figure 5.1: An example view of the experimental setup for the audio-visual pilot user study.

the condition which is tested, where the proportion of clusters in the visible frustum and outside the visible frustum is varied.

The test was run with 6 subjects (male, aged between 23-45, with normal or corrected to normal vision and reporting normal hearing). All were naive about the experiment. Five of them had no experience in audio. Prior to the test, subjects were familiarized with isolated sound effects and their corresponding visual representation.

The subject stands 1 meter away from a 136 x 102 cm screen (Barco Baron Workbench), with an optical headtracking device (ART) and active stereo glasses (see Figure 5.3). The field of view in this large screen experiment is approximatively 70 °.

Headphones are used for audio output and our system uses binaural rendering [Bla97, Møl92] using the LISTEN HRTF database (http://recherche.ircam.fr/equipes/salles/listen/). The subjects were not part of the database. Hence, they performed a "point and click" pre-test to select the best HRTFs over a subset of 6 HRTF selected to be "most representative" similar to [SWVD06]. The marks attributed for the test are given with a joystick.

The *A* condition was presented first for three candidates, while *AV* condition was presented first for the other three. No significant effect of ordering was observed.

To achieve the desired effect, objects are placed in a circle around the observer; 5 are placed in the viewing frustum and 5 outside. For both control and main conditions, four configurations are used randomly,

by varying the proportion of clusters. Condition 1/4 has one cluster in the view frustum and 4 outside, 2/3 has 2 in the view frustum and 3 outside, etc. A uniform distribution of clusters corresponds to condition 1/4, with only 1 cluster in the frustum - which has a viewing angle of approximately 70° (see above) - representing nearly 1/4 of 360°. Each condition is repeated 15 times with randomized object positions; these repetitions are randomized to avoid ordering effects.

The ITU-recommended *triple stimulus, double blind with hidden reference* technique [and93, IR94] is used: 2 versions of the scene were presented ("A" and "B") and a given reference scene which corresponds to unclustered sound rendering. One of the 2 scenes was always the same as the reference (a *hidden reference*) and the other one corresponds to one of our clustering configurations. For each condition, the subject was presented with a screen with three rectangles ("A", "R" and "B"), shown in Fig. 5.1. The subjects were given a gamepad, and were instructed to switch between "A", "B" and "R" using three buttons on the pad, which were highlighted depending on the version being rendered. The subjects were asked to compare the quality of the approximations ("A" or "B") compared to the reference. They were asked to perform a "*quality judgment paying particular attention to the localization of sounds*" for the 2 test scenes, and instructed to attribute one of 4 levels of evaluation "No difference", "Slightly different", "Different" and "Clearly different" from the reference, which were indicated in rectangles next to the letter indicating the screen (see Fig. 5.1).

## 5.2 Analysis and results

A mark for each evaluation was attributed (from 0 to 3). As suggested by this ITU-R standard protocol, only the difference was kept between the test sample and the hidden reference. The data was also normalized by dividing each mark by the mean score of the user (the average of all marks of the candidate over all his tests).

The plot of the means of the results for each clustering type and each experimental condition is shown in Fig. 5.2.

There was no significant difference between the *A* and *AV* conditions regarding the respective scores of each cluster configuration. However, the difference of quality ratings between configurations was not similar in the two conditions. In condition *A*, 1/4 and 2/3 configurations lead to a similar improvement in the perceived quality (see Figure 5.2). In condition *AV*, the best quality is perceived in configuration 2/3. While 2/3 and 1/4 configurations are not perceived differently in condition *A* (Wilcoxon test, N=90, T=640.5, Z=0.21, p=0,83), the quality scores of 2/3 configuration are higher than those of 1/4 configuration in condition *AV* (Wilcoxon test, N=90, T=306.5, Z=2.56, p=0.01).

As the field of view in this large screen experiment (see Figure 5.3) is approximatively 70° and as there are 5 clusters, a uniform distribution of clusters is given by only 1 cluster in the frustum. The lower perceived quality for 4 clusters in the frustum could be explained by the fact that in this case we only have 1 cluster outside the frustum which make a great error in lateralization (i.e., left / right differentiation).

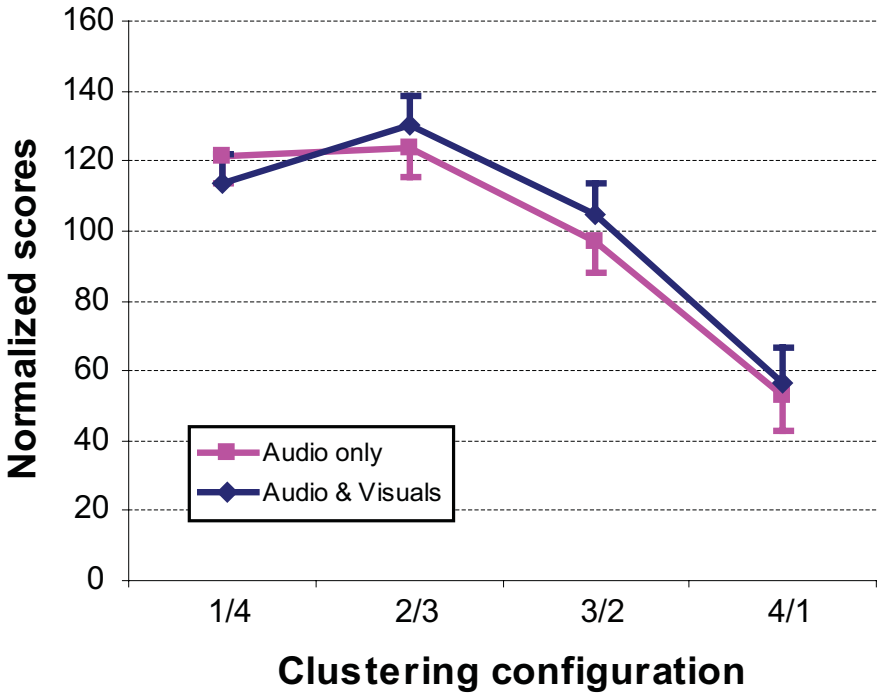


Figure 5.2: Mean values and 95% confidence intervals for the 6 test subjects in Audio only and Audio-Visual conditions as a function of the number of clusters inside / outside the view frustum.

Overall, the above results are considered as a significant indication that, when using the audio clustering algorithm with visual representation of the sound sources, it is better to have more clusters in the view frustum, compared to a uniform angular distribution.

## 5.3 An audio-visual metric for clustering

Given the above observation, a new weight in the clustering metric was developed which encourages more clusters in the view frustum. The cost-function of the clustering algorithm was modified by adding the following weigthing term:

$$1 + \alpha \left( \frac{\cos\theta_s - \cos\theta_f}{1 - \cos\theta_f} \right)^n \tag{5.1}$$

where $\theta_s$ is the angle between the view direction and the direction of the sound source relative to the observer, $\theta_f$ is the angular half-width of the view frustum and $\alpha$ controls the amplitude and $n$ decay-rate of this visual improvement factor.


Figure 5.3: A subject performing the experiment on the workbench.

# Chapter 6

# Implementation and results

## 6.1 Implementation

A key-aspect of the implementation was that the sound engine should have been pretty much independent from the graphics engine used with it. Thus, one can freely decide which scenegraph to choose. To accomplish this task, a standardized *wrapper* class was introduced serving as an interface between the user application - which uses some kind of graphics engine - and the sound engine. This wrapper was provided as an abstract class, hence one wrapper class wil have to be implemented for each graphics engine by derivating from the abstract wrapper class.

For the example programs done for this work, the system is built using the Ogre3D graphics engine. Therefore, the *SoundManager* was implemented which serves as the wrapper and is thus inherited from the abstract class. It is capable of creating sound source objects (which are inherited from *Ogre3D::MovableObject*) and seamlessly integrating them into the Ogre3D scenegraph. A coarse overview of the system is given by Figure 6.1.

Normally, when bringing together audio and graphics, audio is processed on a different thread than video. The same is true in this case. There are plenty of reasons for this. One is that a multi-core system can be fully utilized. Moreover, the audio should not depend on the graphics frame-rate nor the other way round which is difficult to accomplish when running everything in only one thread. And a third factor is the modularization. Having graphics and audio in the same thread, it's very hard to implement both independently from each other like it was done in this work.

The problem arising with multi-threading is of course synchronization. While the positions of the sounds or other attributes are updated by one thread, the other thread must not access these values because their content is undefined at that time. In principle, the sound engine "asks" the application about the camera and sound parameters at each audio frame and the application on the other hand can add sounds, start the audio or request the outcome of the sound rendering from the sound engine, e.g. how many sounds are audible, at which cursor-position a sound is, etc. All this communication happens through the SoundManager (wrapper) and is synchronized transparently by the sound engine using locks.

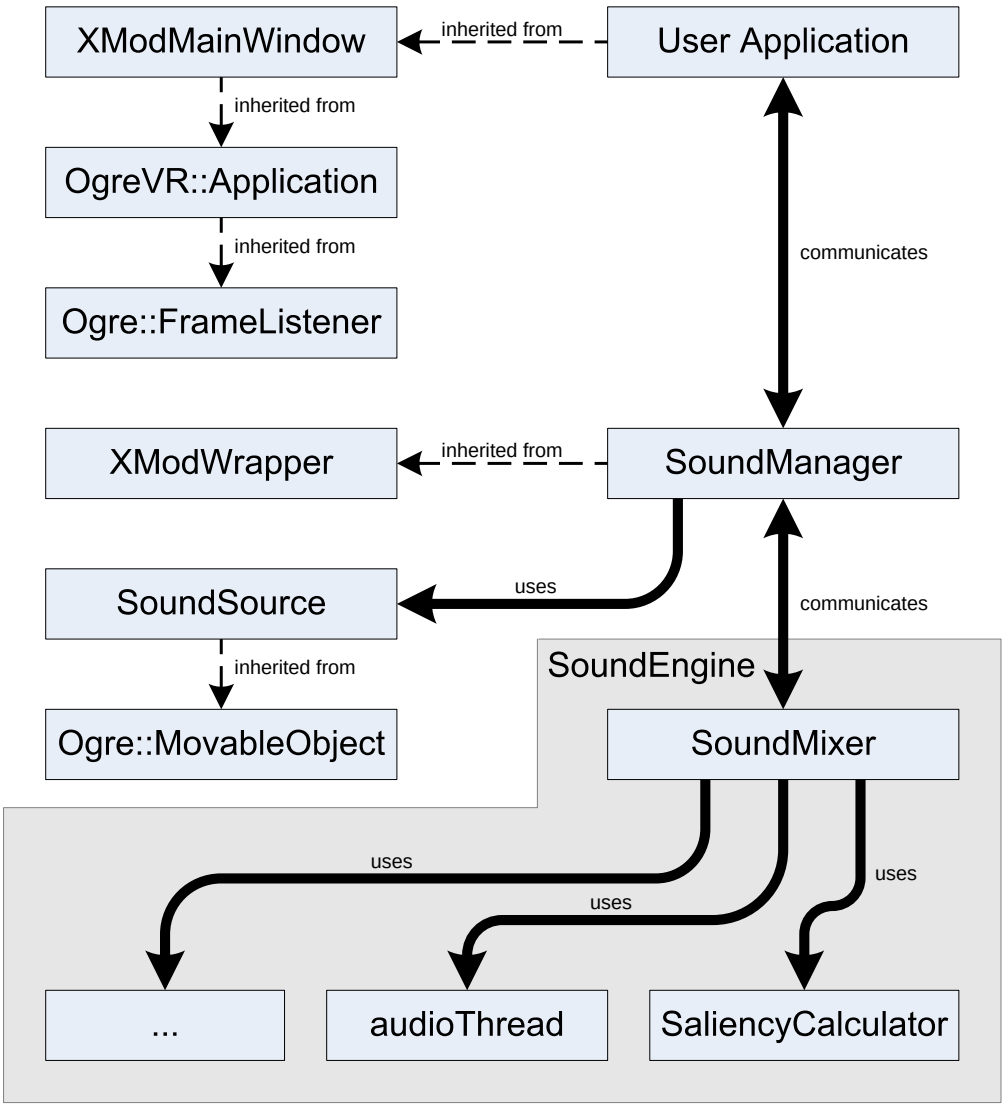


Figure 6.1: Schematic overview of the framework developed. This is a coarse scheme of the system with Ogre3D and OgreVR like it was used. OgreVR is an Ogre-based library and supports stereo rendering.

Figure 6.2 illustrates one frame of the audio processing pipeline. First the positions of the sound sources are fetched from the graphics engine and calculated relative to the listener. The distance attenuation factor for each source is calculated in this step, too. After this, the masking evaluation takes place to detect inaudible sound sources and to discard them. All audible sounds are then grouped together into clusters depending on their positions relative to the listener. At the end, the scalable premixing step follows and assigns a number of freuqencies to each source, mixes the sounds of each cluster, does the HRTF and inverse FFT, mixes the resulting signals of every cluster together and sends the final signal to the soundcard. An alternative in the

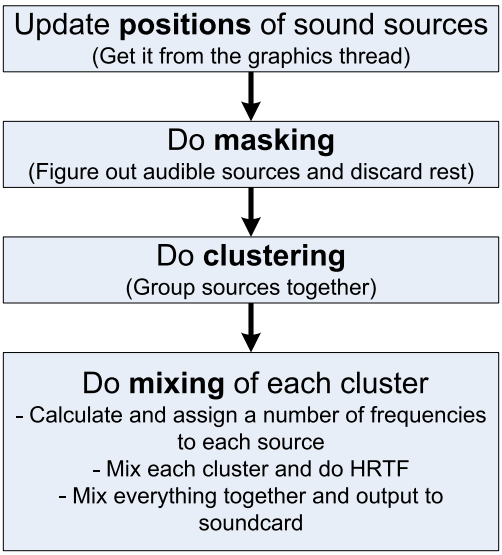


Figure 6.2: Overview of a general audio frame. These steps are performed in the audio thread.

developed system is also to send the mixed signals of each cluster to the soundcard and not mixing them manually. Each cluster could thus be mapped to one output channel of the soundcard.

A final word about the implementation goes to the ability of the CPU to cache data (see Section 2). During implementing speed test programs related to this work, it was a good practice to keep in mind this - in those cases - undesired feature of current CPUs. Because there weren't thousands of unique sound files available, tests had to be designed carefully to be sure that no caching happens when processing the same sound twice in a row and thus performance gains due to caching issues do not happen.

## 6.2 Results

Tests on two scenes were ran, one is a variant of the highway scene from [TGD04], and another is a city scene. Both scenes are shown in Figure 6.3 respectively Figure 6.4. In both cases, a platform with a dual-core 3 GHz *Xeon* processor and a *NVidia 7950 GX2* graphics accelerator was used; the system is built using the Ogre3D graphics engine, like said before, and the custom audio library developed for this work. Audio was processed at 44.1 KHz using 1024-sample-long time-frames (i.e., 23 msec.). The following tests were performed with masking disabled to get a stable performance measure.

The highway scene contains 1004 sound sources, which are car engine and car stereo music sounds, cow "mooing" sounds, train sounds, and water sounds in a stream. It was found that a scalable premix budget of 25% is satisfactory in terms of audio quality. In this scene, clustering took 4.7 msec. per frame. Premixing was very simple and only included distance attenuation and accumulating source signals for each cluster.

Figure 6.3: A city scene with 1815 mobile sound sources. Audio is rendered in realtime with the progressive lossy processing technique using 15% of the frequency coefficients and with an average of 12 clusters for 3D audio processing. Degradations compared to the reference solution are minimal.

Figure 6.4: An highway scene with 1004 mobile sound sources, running with 25% of the frequency coefficients and 12 clusters.

Figure 6.5: Two frames from the walkthrough to test the new audio-visual criterion.

Premixing using 100% of the original audio data took 6 msec. Using our scalable processing with 25% budget this cost is brought down to 1.83 msec.

The street scene contains 1800 sounds sources, which are footstep sounds and voices for the people in the crowd, car engine sounds, car stereo music sounds, bird sounds and sirens. Again, a scalable premix budget of 15% is satisfactory for this scene. In this scene, clustering took 5.46 msec. per frame while premixing using 100% of the original audio data took 6.84 msec. Using the scalable processing with 15% budget the cost of premixing is brought down to 2.17 msec.

In the commercial game *Test Drive Unlimited*, the average number of simultaneous sound sources is 130. A typical "player car" can generate up to 75 sound sources, while "AI" cars have a simplified model of a maximum of 42 sources. A maximum of 32 clusters were used in the game, although in a vast majority of cases 10 clusters are sufficient. Overall, the clustering approach discussed in Section 3.4 results in a reduction of 50-60% of CPU usage for the audio engine, which is freed for other application tasks, such as AI, gameplay etc.

To test the new audio-visual criterion, a variant of the street scene and an appropriate path was constructed, in which the positive effect of this criterion is clearly audible. For this test, the values $\alpha = 10$ and $n = 1.5$ were used, which proved to be satisfactory. The user follows a path in the scene and stops in a given location in the scene. There are 132 sources in the environments and a target budget of 8 clusters. By switching between the reference, and the approximations with and without the audio-visual metric, the improvement is clearly audible when more clusters are used in the view frustum. In particular, the car on the right has a siren whose sound is audibly displaced towards the centre with the audio-only metric.


Figure 6.6: Left: The clusters without the audio-visual metric. Right: The clusters with the new metric. It is clearly shown that the new metric separates the sources appropriately.

# Chapter 7

# Discussion and conclusion

In this work, a fast hierarchical clustering approach has been proposed that can handle large numbers of sources and clusters. Furthermore a progressive processing pipeline has been proposed for per-source effects (i.e., the premixing) that allows one to choose the best performance / quality ratio depending on the application and hardware constraints.

Combined with auditory masking evaluation, these new algorithms allow for real-time rendering of thousands of mobile sound sources while finely controlling processing load vs. quality. In fact, while designing the test examples for the thesis, the major problem faced was authoring environments complex enough and most of the performance limitations actually came from the graphics engine. However, with next-generation games making increased used of procedurally-generated audio (e.g., based on physics engines), scenes with thousands of sound events to process are likely to become common in the near future. In the examples only simple per-source processing was used. However, in most gaming applications each source is likely to be processed with a chain of various effects (e.g., occlusion filters, echoes, re-timing, pitch-shifting, etc.) that would make the scalable approach even more attractive.

Besides gaming applications another interesting utilization for this scalable mixing method would be a chat application taking place in a virtual 3D environment. Then, only audible data has to be transferred from the server to a specific client thus saving a vast amount of traffic and / or being able to increase quality.

Of course hardware is advancing very well and one could claim that a scalable level-of-detail approach is not necessary in the near future. But in fact, sound engines are not supposed to use 100% of the hardware available in a game since there are many other things to do additionally (graphics, physics, AI computations, ...). It is very common to spend only a fractional amount of resources to audio which fortifies the clustering and scalable approach done in this work.

Perceptual studies for clustering and scalable premixing were also presented. A cross-modal perceptual study aimed at determining possible influence of the visuals on the required quality for audio clustering. Although one could expect ventriloquism to allow for rendering simplifications for visible sources, the study suggest that more clusters might actually be required in this case. A possible explanation for this is that, in a complex scene, clustering is likely to simplify auditory localization cues beyond common ventriloquism

thresholds. As a consequence, a new metric was introduced to augment the importance of sources inside the view frustum. An example was demonstrated where - with a large number of sound sources outside the view frustum - it leads to improved results. Moreover a user-study of quality was performed for the scalable premixing approach and showed that it leads to high quality results with budgets as low as 20 to 15% of the original input audio data. Although auditory saliency map-based importance appeared to show some limitations for the scalable processing algorithm compared to loudness, it might still be useful for prioritizing sources for clustering. It may also be useful for a basis of future work to define perceptual saliency for applications like telephone or even audio coding algorithms.

In the future, it would be interesting to experiment with auditory saliency metrics to drive clustering and evaluate the algorithms on various combinations of A/V displays (e.g., 5.1 surround or wave field synthesis setups). Also, the influence of ventriloquism on these algorithms merits further study. It is also likely that authoring is now becoming a fundamental problem. Adapting the algorithms to handle combinations of sample-based and procedurally synthesized sounds seems a promising area of future research.

# Diskussion und Fazit

In dieser Arbeit wurde eine schnelle hierarchische Clusteringmethode vorgestellt, die in der Lage ist, eine große Anzahl von Soundquellen und Clustern zu bewältigen. Des Weiteren wurde eine progressive Pipeline zur Verarbeitung von Effekten entwickelt, die individuell für jede Soundquelle angewandt werden ("Pre-mixing"). Diese Verarbeitung erlaubt, das beste Verhältnis zwischen Performance und Qualität zu wählen, abhängig von der Anwendung und den Vorgaben der Hardware.

Zusammen mit der auditiven Maskierungsermittlung, erlauben diese neuen Algorithmen Echtzeitrendering von tausenden mobilen Soundquellen während es gleichzeitig möglich ist, die Auslastung sehr fein gegenüber der Qualität abzustimmen. In der Tat war das Hauptproblem während des Entwerfens von Testbeispielen die Entwicklung von virtuellen Umgebungen, die komplex genug sind um die Algorithmen auszunutzen. Die meisten Performanceprobleme kamen in diesem Fall von der Grafikengine.

Allerdings werden Szenen mit Tausenden von zu verarbeitenden Soundquellen in der bald kommenden nächsten Spielegeneration sehr verbreitet sein, da diese Spiele prozedural generierte Sounds enthalten werden (z.B. basiert auf der Physikengine). In den Beispielen wurde nur eine einfache pro-Soundquelle-Verarbeitung benutzt. Jedoch ist es in den meisten Spieleapplikationen üblich, dass jede Schallquelle mit einer Kette von verschiedensten Effekten bearbeitet wird (z.B. Verdeckungsfilter, Echos, Neuberechnung des Timings, Tonhöhenveränderung, usw.), was den hier vorgestellten skalierbaren Ansatz noch attraktiver macht.

Von Spielen abgesehen wäre eine andere interessante Anwendung für die skalierbare Methode, Sounds zu mischen, eine Chatanwendung welche in einer virtuellen 3D-Umgebung stattfände. In diesem Fall müssten nur hörbare Daten vom Server zum entsprechenden Client übertragen werden. Dies würde enorm viel Datenverkehr sparen und es möglich machen, die Qualität entsprechender heutiger Anwendungen zu steigern.

Natürlich wird Hardware stetig verbessert, so dass man argumentieren könnte, ein skalierbarer level-of-detail Ansatz wird in Bälde nicht mehr nötig sein. Aber Soundengines können in Spielen nicht 100% der verfügbaren Hardware benutzen, da zusätzlich viele andere Dinge zu berechnen sind (Grafik, Physik, künstliche Intelligenz, ...). Somit wird auch künftig nur ein Bruchteil der Ressourcen für Sound verwendet werden können, was das Clustering und den skalierbaren Ansatz in dieser Arbeit noch untermauert.

Des Weiteren wurden Wahrnehmungsuntersuchungen für das Clustering und skalierbare Vormischen der Sounds präsentiert. Eine transmodale Wahrnehmungsstudie zielte darauf ab, mögliche Einflüsse der

visuellen Eindrücke auf die benötigte Qualität für Soundclustering festzustellen.

Obwohl man erwarten könnte, dass es der Bauchredner-Effekt erlaubt, sichtbare Quellen vereinfacht und in schlechterer Qualität zu rendern, hat die Untersuchung gezeigt, dass in diesem Fall mehr Cluster nötig sein könnten. Eine mögliche Erklärung dafür ist, dass in einer komplexen Szene Clustering leicht die auditiven Lokalisierungsmerkmale jenseits der gewöhnlichen Bauchredner-Schwellen korrumpiert.

Folglich wurde eine neue Metrik eingeführt, um die Wichtigkeit von Soundquellen im Sichtfeld zu erhöhen. Ein Beispiel wurde demonstriert, bei dem dies - mit einer großen Zahl an Soundquellen im Sichtfeld - zu besseren Ergebnissen führt.

Überdies gab es eine Untersuchung der Qualität der Methode für das skalierbare Vormischen der Soundquellen (“Premixing”). Diese zeigte, dass ein Budget von nur noch 20 bis 15% der originären Eingangsdaten zu hochqualitativen Ergebnissen führt. Obwohl die Anwendung der auditiven Saliency Map auf die Ermittlung der Wichtigkeit der Schallquellen beim skalierbaren Verarbeitungsalgorithmus einige Schwächen im Vergleich zu Loudness gezeigt hat, könnte diese Metrik trotzdem für die Priorisierung der Sounds beim Clustering nützlich sein. Darüber hinaus könnte die auditive Saliency Map helfen, die wahrnehmbaren Attribute im Schall zu bestimmen, was als Basis für Anwendungen wie Telefon oder sogar Audio-Kodierungsalgorithmen dienen könnte.

In der Zukunft wäre es interessant, mit auditiven Saliency-Metriken zu experimentieren, um Clustering zu steuern. Außerdem könnten die Algorithmen mit verschiedenen Kombinationen von audiovisuellen Darstellungen evaluiert werden (z.B. mit 5.1 Surround oder Wave Field Synthesis Aufbauten). Darüber hinaus verdient der Einfluss des Bauchredner-Effekts auf diese Algorithmen weitere Untersuchung. Es scheint, als würde das Erstellen von Sounds in baldiger Zukunft ein fundamentales Problem werden, da sehr viele verschiedene und in Echtzeit angepasste Geräusche benötigt werden. Die Algorithmen darauf anzupassen und Kombinationen von Sample-gestützten und prozedural hergestellten Sounds zu unterstützen, könnte ein vielversprechendes Feld für zukünftige Forschung sein.

# Bibliography


[AB04] D. Alais and D. Burr. The ventriloquism effect results from near-optimal bimodal integration. *Current Biology*, 14:257–262, 2004.

[and93] C. Grewin and. Methods for quality assessment of low bit-rate audio codecs, proceedings of the 12th aes conference. pages 97–107, 1993.

[BdVV93] A.J. Berkhout, D. de Vries, and P. Vogel. Acoustic control by wave field synthesis. *J. of the Acoustical Society of America*, 93(5):2764–2778, may 1993.

[Beg94] Durand R. Begault. *3D Sound for Virtual Reality and Multimedia*. Academic Press Professional, 1994.

[BF03] Frank Baumgarte and Christof Faller. Binaural cue coding - part I: Psychoacoustic fundamentals and design principles. *IEEE Transaction on Speech and Audio Processing*, 11(6), 2003.

[Bla97] J. Blauert. *Spatial Hearing : The Psychophysics of Human Sound Localization*. M.I.T. Press, Cambridge, MA, 1997.

[CVH95] J. Chen, B.D. Van Veen, and K.E. Hecox. A spatial feature extraction and regularization model for the head-related transfer function. *J. of the Acoustical Society of America*, 97:439–452, January 1995.

[DDS02] D. Darlington, L. Daudet, and M. Sandler. Digital audio effects in the wavelet domain. In *Proceedings of COST-G6 Conference on Digital Audio Effects, DAFX2002, Hamburg, Germany*, September 2002.

[EBU03] EBU subjective listening tests on low-bitrate audio codecs. *Technical report 3296, European Broadcast Union (EBU), Projet Group B/AIM*, june 2003.

[Fal06] C. Faller. Parametric joint-coding of audio sources. *proc. of the 120th AES Conv., Paris, France*, May 2006.

[FB03] Christof Faller and F. Baumgarte. Binaural cue coding - part II: Schemes and applications. *IEEE Transaction on Speech and Audio Processing*, 11(6), 2003.

[FHB97] H. Fouad, J.K. Hahn, and J.A. Ballas. Perceptually based scheduling algorithms for real-time synthesis of complex sonic environments. *proceedings of the 1997 International Conference on Auditory Display (ICAD'97),*, 1997.

[GJ06] Michael Goodwin and Jean-Marc Jot. Analysis and synthesis for universal spatial audio coding. *In 121th AES Convention, San Francisco, USA. Preprint 6874*, 2006.

[GLT] Emmanuel Gallo, Guillaume Lemaitre, and Nicolas Tsingos. Prioritizing signals for selective real-time audio processing. In *Proc. of Intl. Conf. on Auditory Display (ICAD) 2005,*.

[GTL06] Emmanuel Gallo, Nicolas Tsingos, and Guillaume Lemaitre. 3D-Audio matting, post-editing and re-rendering from field recordings. *to appear in EURASIP Journal on Applied Signal Processing, special issue on Spatial Sound and Virtual Acoustics*, 2006.

[Her99] Jens Herder. Optimization of sound spatialization resource management through clustering. *The Journal of Three Dimensional Images, 3D-Forum Society*, 13(3):59–65, September 1999.

[Her02] Jürgen Herre. Audio coding - an all-round entertainment technology. In *Audio Engineering Society 22nd International Conference on Virtual, Synthetic and Entertainment Audio (AES'22), Espoo, Finland*, pages 139–148, June 15-17 2002.

[How92] David C. Howell. *Statistical methods for psychology*. PWS-Kent, 1992.

[HS85] Dorit S. Hochbaum and David B. Schmoys. A best possible heuristic for the ık-center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985.

[HWaBS$^+$03] W.D. Hairston, M.T. Wallace, J.W. Vaughan amd B.E. Stein, J.L. Norris, and J.A. Schirillo. Visual localization ability influences cross-modal bias. *J. Cogn. Neuroscience*, 15:20–29, 2003.

[IKN98] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, November 1998.

[Int03] International Telecom. Union. Method for the subjective assessment of intermediate quality level of coding systems. *Recommendation ITU-R BS.1534-1*, 2001-2003.

[IR94] ITU-R. Methods for subjective assessment of small impairments in audio systems including multichannel sound systems. itu-r bs 1116. Technical report, 1994.

[ISO03] ISO. Normal equal-loudness-level contours. *ISO, 226:2003 Acoustics International Organization for Standardization (ISO)*, august 2003.

Bibliography


[JBB97] Moore B. C. J., Glasberg B., and Baer. *A model for the prediction of thresholds, loudness and partial loudness*. J. of the Audio Engineering Society 45, 4, 224-240, 1997.

[JLP99] J.-M. Jot, V. Larcher, and J.-M. Pernaux. A comparative study of 3D audio encoding and rendering techniques. *Proceedings of the AES 16th international conference, Spatial sound reproduction, Rovaniemi, Finland*, april 1999.

[JW06] Jean-Marc Jot and Martin Walsh. Binaural simulation of complex acoustic scenes for interactive audio. *In 121th AES Convention, San Francisco, USA. Preprint 6950*, 2006.

[KAG+02] Evelyn Kurniawati, Javed Absar, Sapna George, Chiew Tong Lau, and Benjamin Premkumar. The significance of tonality index and nonlinear psychoacoustics models for masking threshold estimation. In *Proceedings of the International Conference on Virtual, Synthetic and Entertainment Audio AES22*, june 2002.

[KPLL05] C. Kayser, C. Petkov, M. Lippert, and N.K. Logothetis. Mechanisms for allocating auditory attention: An auditory saliency map. *Current Biology*, 15:1943–1947, November 2005.

[KT02] M.C. Kelly and A.I. Tew. The continuity illusion in virtual auditory space. *proc. of the 112th AES Conv., Munich, Germany*, May 2002.

[LEG01] Jörg Lewald, Walter H. Ehrenstein, and Rainer Guski. Spatio-temporal constraints for auditory-visual integration. *Beh. Brain Research*, 121(1-2):69–79, 2001.

[LJGW00] V. Larcher, J.M. Jot, G. Guyard, and O. Warusfel. Study and comparison of efficient methods for 3d audio spatialization based on linear decomposition of HRTF data. *Proc. 108th Audio Engineering Society Convention*, 2000.

[LS97] C. A. Lanciani and R. W. Schafer. Psychoacoustically-based processing of MPEG-I layer 1-2 encoded signals. In *Proc. IEEE Signal Processing Society 1997 Workshop on Multimedia Signal Processing*, pages 53–58, June 1997.

[LS99] Chris A. Lanciani and Ronald W. Schafer. Subband-domain filtering of MPEG audio signals. In *Proceedings of Intl. Conf. on Acoustics, Speech and Signal Processing*, pages 917–920, March 1999.

[LSW03] C.T. Lovelace, B.E. Stein, and M.T. Wallace. An irrelevant light enhances auditory detection in humans: a psychophysical analysis of multisensory integration in stimulus detection. *Cognitive Brain Research*, 17(2):447–453, 2003.

[MBT+07] T. Moeck, N. Bonneel, N. Tsingos, G. Drettakis, I. Viaud-Delmon, and D. Alloza. Progressive perceptual audio rendering of complex scenes. 2007.

[MM95] D.G. Malham and A. Myatt. 3D sound spatialization using ambisonic techniques. *Computer Music Journal*, 19(4):58–70, 1995.

[Møl92] Henrik Møller. Fundamentals of binaural technology. *Applied Acoustics*, 36:171–218, 1992.

[Moo97] Brian C.J. Moore. *An introduction to the psychology of hearing*. Academic Press, 4th edition, 1997.

[PM77] D.R. Perrot and A.D. Musicant. Minimum auditory movement angle : Binaural localization of moving sound sources. *J. of the Acoustical Society of America*, 62(6):1463–1466, 1977.

[PS00] E. M. Painter and A. S. Spanias. Perceptual coding of digital audio. *Proceedings of the IEEE*, 88(4), April 2000.

[Pul06] V. Pulkki. Directional audio coding in spatial sound reproduction and stereo upmixing. *Proc. of the AES 28th Int. Conf, Pitea, Sweden*, June 2006.

[Sha49] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[SHLV99] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen. Creating interactive virtual acoustic environments. *J. of the Audio Engineering Society*, 47(9):675–705, September 1999.

[SK00] G. Stoll and F. Kozamernik. EBU subjective listening tests on internet audio codecs. *EBU TECHNICAL REVIEW,*, June 2000.

[Sto98] Russell L. Storms. *Auditory-Visual Cross-Modal Perception Phenomena*. PhD thesis, Naval Postgraduate School, Monterey, California, September 1998.

[SWVD06] L. Sarlat, O. Warusfel, and I. Viaud-Delmon. Ventriloquism aftereffects occur in the rear hemisphere. *Neuroscience Letters*, 404:324–329, 2006.

[TEP04] Abdellatif B. Touimi, Marc Emerit, and Jean-Marie Pernaux. Efficient method for multiple compressed audio streams spatialization. In *In Proceeding of ACM 3rd Intl. Conf. on Mobile and Ubiquitous multimedia*, 2004.

[TGD04] Nicolas Tsingos, Emmanuel Gallo, and George Drettakis. Perceptual audio rendering of complex virtual environments. *SIGGRAPH*, August 2004.

[Tou00] Abdellatif B. Touimi. A generic framework for filtering in subband domain. In *In Proc. of IEEE 9th Wkshp. on Digital Signal Processing, Hunt, Texas, USA*, October 2000.

[Tsi05] N. Tsingos. Scalable perceptual mixing and filtering of audio signals using an augmented spectral representation. *Proc. of 8th Intl. Conf. on Digital Audio Effects (DAFX'05), Madrid, Spain*, September 2005.

Bibliography

[wik] Central Processing Unit. Website. Available online at `http://en.wikipedia.org/wiki/Central_processing_unit`; visited on January 19th 2007.

[WS04] Michael Wand and Wolfgang Straßer. Multi-resolution sound rendering. In *Symp. Point-Based Graphics*, 2004.

[Yee00] Yang Li Hector Yee. Spatiotemporal sensistivity and visual attention for efficient rendering of dynamic environments. Master's thesis, Cornell University, 2000.

[ZF99] Eberhard Zwicker and Hugo Fastl. *Psychoacoustics. Facts and Models*. Berlin, 2nd ed. edition, 1999.

[Zöl02] Udo Zölzer, editor. *DAFX - Digital Audio Effects*. Wiley, 2002.

# Bibliography

# Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ich bin damit einverstanden, dass die Arbeit veröffentlicht wird und dass in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Graphische Datenverarbeitung, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Erlangen, den 01.02.2007

(Thomas Moeck)