Perceptual Audio Rendering of Complex Virtual Environments

Nicolas Tsingos, Emmanuel Gallo, George Drettakis

Perceptual Audio Rendering of Complex Virtual Environments

Nicolas Tsingos — Emmanuel Gallo — George Drettakis

N° 4734

Février 2003

THÈME 3

ISSN 0249-6399 ISRN INRIA/RR--4734--FR+ENG

Perceptual Audio Rendering of Complex Virtual Environments

Nicolas Tsingos , Emmanuel Gallo , George Drettakis

Thème 3 —Interaction homme-machine,
images, données, connaissances
Projet REVES

Rapport de recherche n° 4734 —Février 2003 —21 pages

Abstract: Including spatialized audio in virtual environments significantly enhances the sense of realism and immersion. With the constant increase in complexity of virtual scenes, rendering spatialized audio is currently often impossible, due to the prohibitive signal processing costs of rendering a large number of sound sources.

We present a novel solution to this problem permitting audiovisual rendering of environments containing hundreds of moving sound sources with complex frequency-dependent effects and reverberation processing on an consumer grade system. Our approach introduces three algorithms, based on perceptual culling, a clustering strategy allowing for rendering a large number of sound sources with only a limited number of available hardware 3D audio channels, and the use of graphics hardware to perform required pre-mixing operations on the audio signals. With little overhead to the graphics resources, our method leverages the use of existing audio rendering hardware while introducing additional controls and processing capabilities.

We have implemented our approach, and we present results for a complex, highly dynamic virtual environment containing hundreds of sound sources, animated characters and moving objects. The results show that our approach permits high-quality interactive audio-visual rendering of such scenes.

Key-words: Spatialized Sound, Virtual Environments, Rendering, Graphics hardware, Audio hardware

Rendu audio perceptif d'environnements virtuels complexes

Résumé : L'ajout de son spatialisé dans un environnement virtuel améliore de manière significative le sens de réalisme et d'immersion. Les environnements virtuels devenant de plus en plus complexes, effectuer un rendu sonore spatialisé pour un grand nombre de sources sonores est souvent impossible en raison du coût élevé du traitement du signal.

Nous présentons une solution à ce problème, permettant le rendu audio-visuel d'environnements contenant des centaines de sources sonores, avec réverbération et effets dépendants de la fréquence, sur des systèmes grand-public.

Notre approche introduit trois algorithmes utilisant une élimination perceptive des sources sonores inaudibles, un regroupement permettant de rendre un grand nombre de sources sur un nombre limité de canaux audio cablés ainsi que le hardware graphique pour effectuer les opérations de pré-mixage nécessaires.

Avec peu d'impact sur les performances graphiques, notre méthode permet d'exploiter les ressources matérielles des cartes sonores existantes tout en introduisant des possibilités de contrôle et traitement supplémentaires.

Nous avons implémenté notre approche et présentons des résultats pour un environnement virtuel complexe comprenant des centaines de sources mobiles, de personnages et d'objets animés. Ces résultats montrent que notre approche permet un rendu audio-visuel de qualité pour de telles scènes.

Mots-clés : Son spatialisé, Environnements virtuels, Rendu, Hardware graphique, Hardware sonore

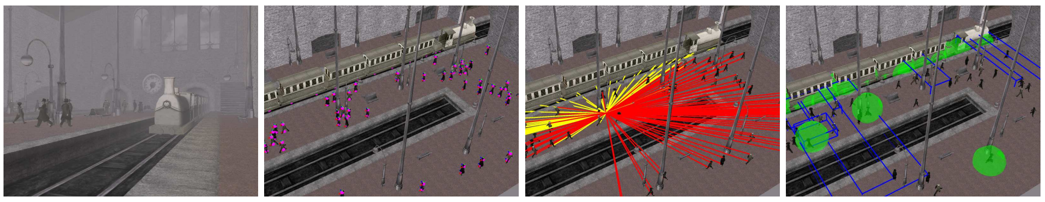

Figure 1: Left, an overview of our test virtual environment, containing 162 sound sources. The people in the crowd and the train are moving. Mid-left, the purple spheres indicate the positions of the sound sources. Notice how the train is an extended source modeled by a collection of point sources. Mid-right, the ray-paths in red correspond to the perceptually masked sound sources. Right, the blue boxes are clusters of sound sources with the representatives of each cluster in green, allowing audio rendering of complex auditory scenes.

1 Introduction

Virtual environments (VE's) are rapidly becoming very complex, in terms of the amount of geometry and texture contained in the scenes, but also in terms of the complexity of animation and interactivity. Including spatialized audio is a key aspect in producing realistic complex VE's. Several studies have shown that the combination of auditory and visual cues enhances the sense of immersion [34, 26, 25]. Current techniques for audio rendering usually assume that sound is emitted by point sources for which direct and low-order indirect contributions to the listening point can be interactively evaluated using geometrical techniques [19, 35]. Indirect contributions are then modeled as secondary "image-sources" [7, 35] that are rendered individually, requiring heavy signal processing resources even for a small number of sources and moderately complex geometry.

Moreover, as can be seen in Figure 1, real-world sound sources such as a train cannot be properly modeled as point-emitters (for instance, a train is better represented as a line source). Such simplifying assumptions considerably reduce the realism of audio simulations. One solution to this problem is to represent an extended sound source in space by a collection of point sources as proposed in [4]. However, this solution further increases the number of sources to render.

Proper modeling of complex source directivity patterns [32], 3D positional audio [5] and artificial reverberation [13, 32] are key aspects of audio rendering and also require significant additional processing cost which limits the realism of audio simulations. Despite advances in commodity audio hardware (e.g., [1]), only a small number of processing channels (i.e, 16 to 64) are usually available.

For all the reasons presented above, current state-of-the-art solutions [35, 32, 37], cannot provide high-quality audio renderings for complex virtual environments which respect the mandatory real-time constraints. Consider the example of Fig. 1, which contains 60 humans, moving trains, various sounds in the station (clock, *etc.*), resulting in 162 sound sources. Previous solutions cannot render spatialized sound for such scenes in real time.

To address this shortcoming, we propose new algorithms permitting high-quality audio rendering for complex VE's, such as that shown in Figure 1. These algorithms should gracefully *scale* and *degrade* to fit specific hardware or processing power constraints whilst limiting audible artefacts; the algorithms should apply *adaptive psychoacoustic refinement*, by searching for the best possible

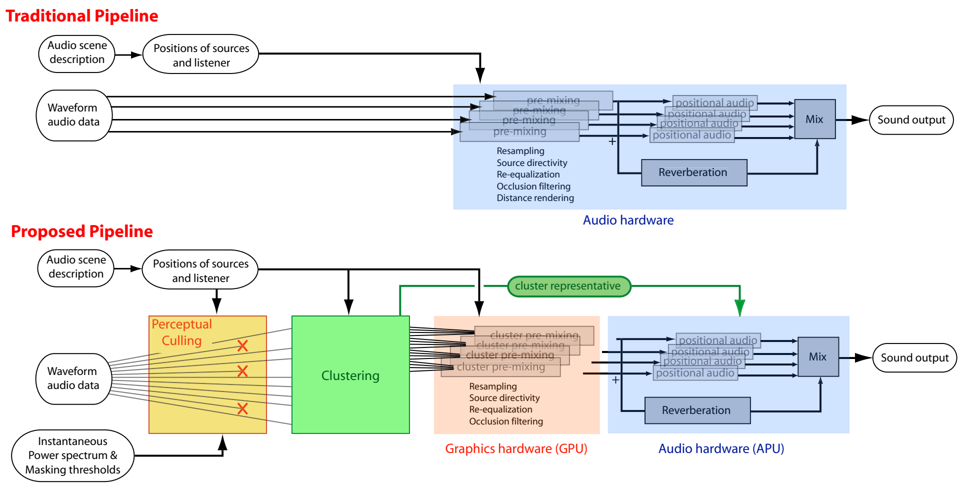

Figure 2: A comparison of a traditional hardware-accelerated audio rendering pipeline (top) and our novel approach combining a perceptual culling/clustering strategy with the use of commodity graphics and audio rendering hardware (bottom). Graphics hardware is used to speed-up per-source pre-mixing operations while specialized audio processing (i.e., positional audio, reverberation) is performed using cluster representatives only.

efficiency vs. accuracy tradeoffs based on existing psychoacoustic knowledge of human hearing and input audio data. The approach should also attempt to *minimise error*, compared to direct rendering of all sources independently, while maintaining *interactivity*, which for audio processing typically requires frame rates of between 20 and 50Hz.

To meet these goals, we present three novel methods which greatly improve the ability to perform high-quality audio rendering for complex scenes, containing a large number (hundreds) of audio sources. This is achieved by combining a dynamic perceptual culling algorithm, a dynamic clustering technique and the use of graphics hardware to speed up audio processing.

We first present a new perceptual sound-source culling technique based on dynamic auditory masking evaluation. This involves preprocessing the input signals so that an efficient on-line evaluation of masking can be performed. The number of sound sources passed onto the next stage of the algorithm is thus significantly reduced.

We next propose an original perceptual clustering strategy. We derive novel metrics exploiting psychoacoustical knowledge of the input signals and human hearing to fit clusters to a dynamic source distribution and construct the representative for each cluster. Our approach best-fits a predefined number of clusters and is locally optimal according to the clustering metric. The number of clusters can be dynamically adapted through time to optimize available computational power.

Finally, to make best use of standard resources available on off-the-shelf platforms, we also show that graphics rendering hardware can be used to perform pre-mixing operations for each individual source. This choice is further justified by the fact that GPU processing power is likely to grow faster than that of the CPU, as discussed in [30], and that it is very unlikely that commodity audio hardware will ever compete in capabilities (e.g., programmability) nor processing power with commodity graphics hardware.

We have implemented these three approaches and ran the resulting system on a moderately complex virtual environment. The results of our tests show that our solutions can render highly dynamic audio-visual virtual environments comprising hundreds of point-sound sources on modest platforms using only off-the shelf, standardized tools. We also present a short objective performance evaluation providing a first validation of our choices. Being able to render such complex combined auditory/visual scenes shows a significant improvement in the quality and realism of the virtual experience. It also suggests that complex soundscapes have little perceptual entropy and can be successfully represented as a small collection of point sources.

2 Previous Work

While psychoacoustic knowledge has been aggressively and successfully exploited in the field of audio compression [28], very few solutions to date have been proposed which reduce the cost of an audio rendering pipeline. Most of them specifically target the filtering operations involved in binaural (headphone based) audio rendering and do not usually rely upon psychoacoustic rules. Chen et al. [12, 11] proposed the use of principal component analysis of Head Related Transfer Functions (HRTFs) to speed up the signal processing operation. Fouad et al. [16] propose a general multi-resolution rendering approach for spatialized audio where the input sound samples are progressively processed, but such approaches might lead to excessive low pass filtering at low resolution. One approach, however, optimizes HRTF filtering by avoiding the processing of psychoacoustically insignificant spectral components of the input signal [24].

Even when using such approaches, the number of sources which can be rendered in software remains limited to a few tens on off-the-shelf systems. Thus, rendering more complex auditory scenes requires dedicated multi-processor systems or distributed audio servers [10].

An alternative to software rendering is to use additional resources such as dedicated DSP systems (*Tucker Davis*, *Lake DSP*, *etc.*) or commodity audio hardware (e.g., *Sound Blaster* [1]). The former are usually high audio fidelity systems but are not widely available and usually support ad-hoc APIs. The latter provide hardware support for game-oriented APIs (e.g., *Direct Sound 3D* [2], and its extensions such as *EAX* [3]). These solutions are widely available, inexpensive and tend to become *de facto* standards. They provide specialized 3D audio processing for a variety of listening setups and dedicated effects such as reverberation processing. In both cases, however, only a small number of sources (typically 16 to 64) can be rendered using hardware channels. Automatic adaptation to resources is available in Direct Sound 8 [2] but is based on distance-culling (i.e., far-away sources are simply not rendered) which can lead to discontinuities in the generated audio signal. Moreover, this solution would fail in scenes where many sources are close to the listener.

A solution to the problem of rendering many sources using limited software or hardware resources has been presented by Herder [21, 22] and is based on a clustering strategy. Similar approaches have also been proposed in computer graphics for off-line rendering of scenes with many lights [29]. In [21, 22], a potentially large number of point-sound sources can be downsampled to a limited number of representatives which are then used as actual sources for rendering. In theory, such a framework is general and can accommodate primary sound sources and image-sources. However, Herder's solution suffers from several drawbacks which limits its practical use. First, his clustering scheme is based on fixed clusters (corresponding to a non-uniform spatial subdivision) which actually cannot be easily adapted to fit a pre-defined budget. Hence, the algorithm cannot be used as is for resource management purposes. Second, the choice of the cluster representative (the cartesian centroid of all sources in the cluster) is not optimal in the psychoacoustical sense. Finally, perhaps the major drawback of such a clustering strategy is that audio streams for each sound source still have to be mixed in order to obtain an aggregate audio stream which can be rendered from the position of the cluster's representative. Such pre-mixing would have to be performed in software and could only consist of simple operations in order for the overall process to remain efficient.

3 Overview of our approach

Our approach can be decomposed into four steps (see Figure 2) repeated for each audio processing frame through time:

- First, we cull perceptually inaudible sources by evaluating instantaneous power spectral densities and masking thresholds for each source. This first stage requires some masking information to be pre-computed for each input sound signal.
- Next, we generate a pre-defined number of clusters from the remaining source distribution. We use a dynamic clustering strategy based on global $k-$ means [17, 18, 27]. Such techniques have been widely studied in the field of vector quantization [20] and appear to provide the best time-*vs*-quality tradeoff [18, 27], in the sense of being fast but not globally optimal. All sound sources in a given cluster are grouped together and a representative point source is constructed for each non-empty cluster.
- Then, a source signal is generated for each cluster in order to feed the available audio hardware channels. This phase involves a number of operations on the original audio data (filtering, re-sampling, mixing, *etc.*) for which parameters (directivity effects, propagation delay, *etc.*) are different for each source. We show that such pre-mixing can be efficiently mapped to commodity graphics hardware in order to optimize processing resources.
- Finally, the pre-mixed signals for each cluster together with their representative point source can be used to feed standard audio rendering hardware through standard APIs (e.g., Direct Sound 3D), or can be rendered in software.

The following sections describe these four stages in more detail.

4 Perceptual culling

Our processing pipeline starts by evaluating a perceptual importance for each source and culling inaudible sources. Contrary to basic distance-culling strategies we want to ensure that all culled

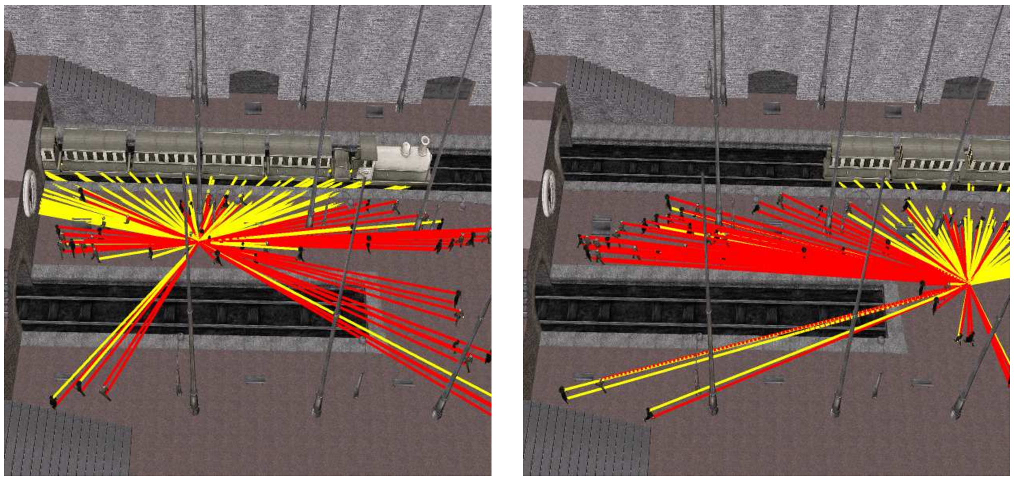

Figure 3: Two frames of our station scene, containing 162 sound sources, distributed on the people, train, station noises, *etc.* visualizing the culling approach. Culled sources are represented with red ray-paths to the listener. For the two different locations of the observer, we can see that large numbers of sources (45 and 75 respectively) are culled, without audible difference. Please listen and watch the corresponding tape/CD rom using headphones to appreciate this effect.

sources are indeed inaudible. The problem of evaluating auditory maskings has been widely studied, especially for perceptual audio coding (PAC) applications. As a result, several expressions are readily available to evaluate masking thresholds of audio signals, for instance in the MPEG-1 layer III (mp3) standard [8, 31]. However, PAC encoders can only process a limited number of signals at a time due to the cost of evaluating the required perceptual information. In our case, the scene is dynamic and masking criteria have to be recomputed at each processing frame for a large number of sources. However, for a variety of applications, the input audio samples are still known in advance (i.e., do not come from a real-time input or are not synthesized in real-time).

4.1 Pre-processing

Based on this assumption, we can precompute masking thresholds of our input signals through time (using small chunks of input signal) and dynamically access them when required. In order to retain efficiency, we evaluate masking thresholds for three frequency bands f corresponding to low, medium and high frequency components of the audible audio spectrum (i.e, 0-500 Hz, 500-2000 Hz, 2000+ Hz). We also estimate the instant power spectral distribution **PSD** over the three bands and store it together with the masking threshold information **M**. Finally we also pre-compute three band-passed copies of our original signals that will be used in the premixing stage of our pipeline (see Figure 2). This pre-processing can be done off-line or when the application is started. We refer the reader to appendix A for additional details on how we pre-compute this information.

4.2 On-line processing

At a given time-frame T of our audio rendering simulation, each source S_k is characterized by : 1) its distance to the listener r, 2) the corresponding propagation delay $\delta = r/c$, where c is the speed of sound, 3) a frequency-dependent attenuation $\mathbf{A}$, resulting from the source's directivity pattern, occlusion, scattering, *etc.*. To be consistent with the pre-processing step, $\mathbf{A}$ is also evaluated for the same three frequency bands.

Our culling algorithm first estimates the perceptual *loudness* L_k^T at time T, of each sound source k as follows:

$$\begin{aligned} L_k^T &= \sum_f \alpha(f)\, \mathbf{P}_k^{T-\delta}(f), \\ \mathbf{P}_k^{T-\delta}(f) &= \mathbf{PSD}_k^{T-\delta}(f) \times \mathbf{A}_k^T(f)/r^2, \end{aligned} \tag{1}$$

where $\mathbf{P}_k^{T-\delta}(f)$ is an estimate of the power level in each frequency band f at time $T-\delta$ and $\alpha(f)$ is a weight that controls the relative perceptual importance of the three components. We used $\alpha(low) = 0.05$, $\alpha(medium) = 0.80$ and $\alpha(high) = 0.15$, based on a simplification of equal loudness contours [38]. Note that this loudness could also be tuned to provide direct user-control of the individual importance of each sound source.

We also smooth the loudness function by averaging it over the last 10 frames. We then sort the sources by decreasing order according to their combined loudness and masking power threshold (i.e., $L_k^T\ ||\mathbf{60} - \mathbf{M_k^{T-\delta}}||$). We also compute the total power level of our scene

$$\mathbf{P_{TOT}} = \sum_k \mathbf{P}_k^{T-\delta}(f).$$

We assume that sound power adds up which is a crude approximation but works reasonably well with real-world signals, which are typically noisy and decorrelated. Moreover, we use conservative masking thresholds that compensate for this assumption. Conservative masking thresholds also ensure that we do not mask a source that could actually be perceived due to our binaural separation cues [23]. Such *binaural unmasking* should be accounted for but little computationally exploitable data is available.

At each frame, we maintain the sum of the power of all sources to be added to the mix, $\mathbf{P_o}$, which is initially equal to $\mathbf{P_{TOT}}$. We then progressively add sources to the mix, maintaining the current mix masking threshold $\mathbf{T}$, as well as the current power $\mathbf{P_{mix}}$ of the mix. To effect the perceptual culling, we apply the following algorithm, where **ATH** is the absolute threshold of hearing [38]:

```
$\mathbf{P_o} = \mathbf{P_{TOT}}$
while $\mathbf{P_o} > \mathbf{P_{mix}} - \mathbf{T}$
    and $\mathbf{P_o} < \mathbf{ATH}$ do
    add source $S_k$
    $\mathbf{P_o} -= \mathbf{P}_k$
    $\mathbf{P_{mix}} += \mathbf{P}_k$
    $\mathbf{T} += \mathbf{M}_k$
end
```

Since we always maintain an overall estimate for the power of the entire scene our culling algorithm behaves well even in the case of a scene composed of many low-power sources. This is the case for instance with image-sources resulting from sound reflections. A naive algorithm might have culled all sources while their combination is actually audible.

In Figure 3 we demonstrate the effectiveness of our culling approach on a our example virtual environment.

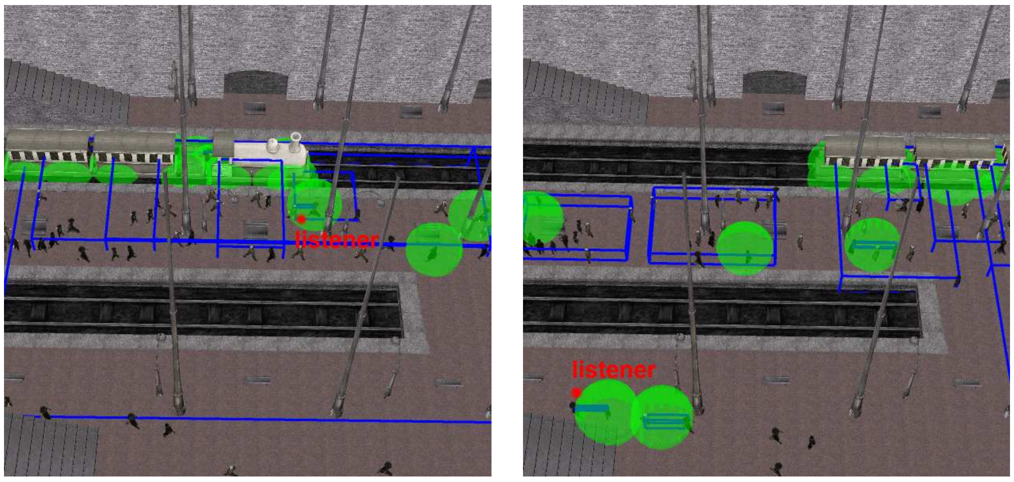

Figure 4: Two frames of our station scene, as in Fig. 3, visualizing the corresponding clusters for two listener positions and different positions of the train. The listener is shown in red. Cluster representatives are visualized as green spheres. Note how the scene is highly dynamic, and that the clusters change dramatically over short periods of time. Please listen and watch the corresponding tape/CD rom using headphones to appreciate this effect.

5 Clustering Sound Sources

Sources that have not been culled by the previous stage are then grouped by our dynamic clustering algorithm. Each cluster will act as a new point source representing all the sources it contains. Our goal is to ensure minimal perceivable error between these auditory impostors and the original auditory scene.

Perception of multiple simultaneous sources is a complex problem which has received a lot of attention in the acoustics community [6]. This problem is also actively studied in the community of auditory scene analysis (ASA) [9, 15]. However, ASA attempts to solve the inverse and more complex problem of segregating a complex sound mixture into discrete, perceptually relevant components.

5.1 Clustering metrics and cluster representative

Our error metric is based on the sum of two spatial deviation terms from a source S_k to the cluster representative C: a distance deviation term and an angular deviation term.

$$d(C,S_k) = L_k^T \left(\beta log_{10}(||C||/||S_k||) + \gamma \frac{1}{2}(1 + C.S_k) \right), \tag{2}$$

where L_k^T is the source loudness presented in the previous section Eq. (1). This weighting ensures error is minimal for perceptually important sources. In our experiments we used $\beta = 2$ and $\gamma = 1$, to better balance distance and angle errors.

The global error E_n for the cluster is simply the sum of the error of all sources S_j in the cluster:

$$E_n = \sum_j d(C,S_j) \tag{3}$$

The representative for the cluster must ensure minimal acoustic distortion when used to spatialize the signal. In particular it must preserve the overall impression of distance and incoming direction on the listener. Thus, a good starting candidate is the centroid of the set of points in (distance, direction) space. Since we are not using a fixed spatial subdivision structure as in [21], the cartesian centroid would lead to incorrect results for spatially extended clusters. For instance, as can be seen in Figure 5, the cartesian centroid (red) of the two sources would lead to a point source very close to the listener. Hence, both sources would sound louder than they really are. Using the centroid in polar coordinates (green) yields a better representative since it preserves the average distance (and thus propagation delay) to the listener.

Moreover, source loudness will affect spatial perception of sound [6]. Hence, we can actually use the loudness to affect the location of the representative. Based on these observations, the location of the representative C is defined in spherical coordinates as:

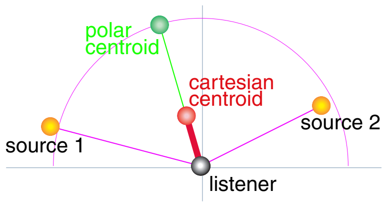

Figure 5: Cartesian versus polar centroid. In this configuration the cartesian centroid leads to a representative too close to the listener while the polar centroid preserves the distance.

$$\begin{aligned} \rho_C &= \sum_j L_j^T r_j / (\sum_k L_k^T), \\ \theta_C &= \theta(\sum_j L_j^T \mathbf{S}_j / (\sum_k L_k^T)), \\ \phi_C &= \phi(\sum_j L_j^T \mathbf{S}_j / (\sum_k L_k^T)). \end{aligned} \tag{4}$$

Note that in the equations presented, all source positions $\mathbf{S}_j$ are assumed to be expressed in a cartesian coordinate system relative to the listener position and orientation.

5.2 Dynamic clustering algorithm

Our clustering strategy starts with a single cluster and successively “splits" the cluster, until the specified maximum number of clusters is reached. Of course, if fewer sources are present than specified clusters, the algorithms returns one cluster per input source.

In a first phase, the algorithm selects and splits the cluster with maximal error by creating two new representatives based on the current representative location. The two points are chosen to be: 1) the previous representative for the cluster and 2) one of the sources in the cluster chosen in order to minimize the error function.

We test all possibilities amongst the sources in the cluster and retain the source leading to the lowest overall error (Eq. 3), resulting in the creation of two clusters. The distance (Eq. (2)) of each of these two points to each of the sources in the cluster is evaluated and sources are assigned to the cluster corresponding to the point they are closer to.

In a second step, the location of the two new representatives and their set of sources is optimized by iteratively recomputing the representative for the two groups using Eq. (4) and re-assigning the sources to these newly obtained points. This phase stops when the global error for the two new clusters (the sum of the two errors) reaches a local minimum. This process typically required 4-5 iterations in our tests.

Figure 4 shows several examples of clusters and representatives obtained with this approach. Note how the algorithm adapts well to a variety of situations, even in complex, highly dynamic scenes.

6 Audio processing with the GPU

The third stage of our pipeline is to compute an aggregate signal for an entire cluster based on the individual signals emitted by each individual sound source it contains. This signal will then be spatialized in the final stage of the pipeline.

Computing this pre-mix of all sources involves a number of operations such as filtering (to account for scattering, occlusion, *etc.*), resampling (to account for the variable propagation delay and Doppler shifting) and $1/r$ distance attenuation of the input signals.

Filtering depends on the source directivity pattern and thus must be performed on each source individually. In our case, we use a simple frequency dependent attenuation to account for all filtering effects. We implement such filtering as a simple equalization over our set of three frequency bands.

To avoid artefacts, propagation delay also has to be accounted for on a per source basis. Not doing so would result in clicking artefacts as noticed in [21]. A simple method to account for time-varying non-integer delays is to re-sample the input sound signal. Simple linear interpolation gives good results in practice, especially if the signals are oversampled beforehand [37].

Since they must be performed on a per-source basis, pre-mixing operations can quickly become the bottleneck of a clustering approach, especially on low-end mono-processor platforms.

To further optimize all available resources, we use commodity graphics hardware processing to perform all necessary audio pre-mixing operations.

In this section we show that graphics hardware is well suited to re-sampling, mixing and calculations on separate frequency bands required for audio rendering. The processing pipeline we present can be implemented with standard graphics processing operations, using standard APIs such as *OpenGL*™, which makes the method usable on a wide range of platforms.

6.1 Loading audio into texture memory

As a result of our preprocessing stage we obtain three band-passed copies of the original input audio waveforms (see Section 4 and Appendix A). Note that this is still a time domain representation of the original signals where each of the three signals only contains a sub-part of the audible frequency-range.

These three copies are loaded into graphics texture memory as a collection of one-dimensional (R,G,B) textures corresponding to (low, medium, high) frequency-components (see Figure 6). This ensures efficient parallel processing of the three components through the graphics pipeline.

Due to hardware limitations, texture width must be a power of two and is typically limited to 2048 pixels. Hence we have to generate a collection of small texture chunks for each signal. We also have to separate positive and negative parts of the signal since the frame buffer only supports positive values.

6.2 Mapping pre-mixing to graphics hardware

Pre-mixing operations typically involve re-equalization, re-sampling and mixing audio streams together. To achieve this in hardware, the pre-mixed audio data for each cluster is created by rendering a colored, textured line for each sound source in the cluster.

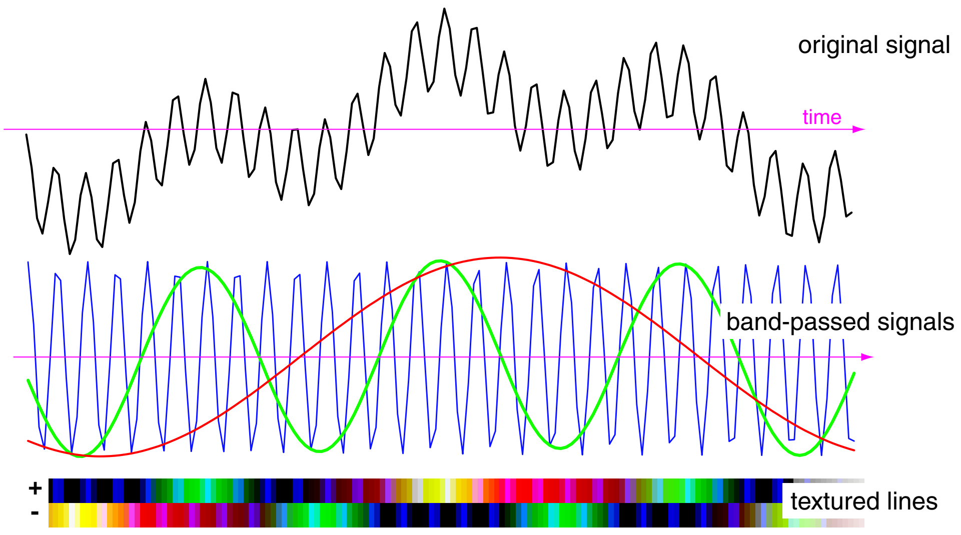

Figure 6: Input audio signals are pre-processed in order to be premixed by the GPU. They are split into three frequency bands (low, medium, high), shown as the red, green and blue plots on the lower graph, and stored as one-dimensional RGB texture chunks (one pair for the positive and negative parts of the signal).

We first use texture resampling to render Doppler shift. Due to the variable propagation delay δ (discussed in Sec. 4.2), the signal may be time-compressed or dilated. Consequently the signal for a current frame might require data from several texture blocks (see Figure 7). In this case, care must be taken to render a separate line-segment for each texture block to build an entire frame of resampled audio data. This is illustrated in Fig. 7.

To achieve the effect of the attenuation **A** (Sec. 4.2), we use color modulation of the three frequency components. The signals are also attenuated by a distance factor accounting for the $1/r$ pressure attenuation.

We activate blending to accumulate the different consitutant sources of the cluster together in the frame-buffer.

Usually, the final stage of the pipeline will already include distance processing based on the location of the cluster's centroid (for example when using audio hardware). Hence, we use an attenuation factor relative to the centroid's distance, i.e., ρ_C/r. This also contributes to preserving the dynamic range since we are accumulating signals into a limited resolution frame-buffer.

As mentioned previously, for each cluster, positive and negative parts of the signal are rendered separately as a pair of lines in the frame buffer. Hence for a 1200 sample long audio processing frame, we need a $1200 \times 2 \times N$ frame-buffer, where N is the number of non-empty clusters after the clustering algorithm converged.

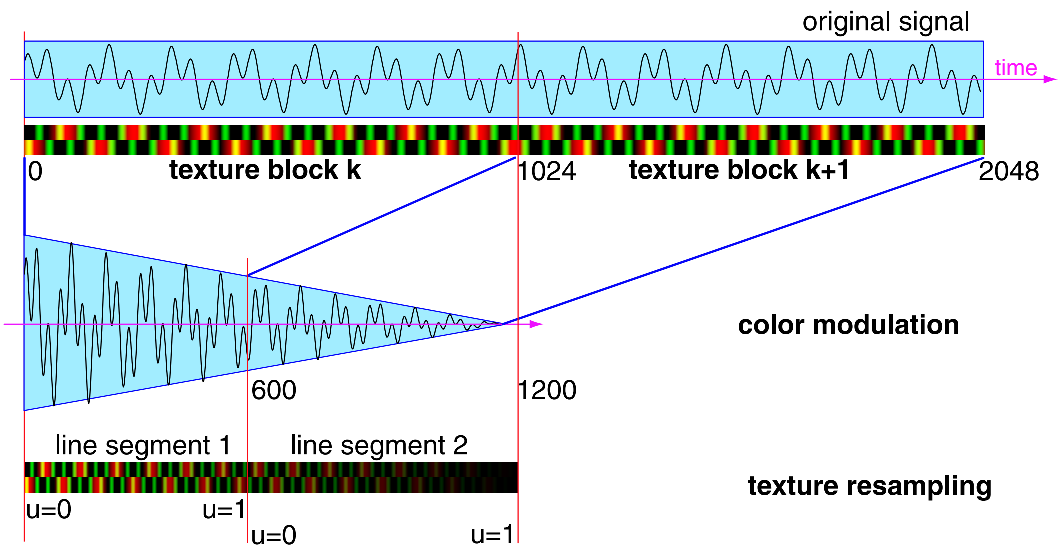

Figure 7: Pre-mixing audio signals with the GPU. Signals are rendered as textured line segments. Resampling is achieved through texturing operations and re-equalization through color modulation. Signals for all sources in the cluster are rendered with blending turned on, resulting in the desired mix.

Once all signals for all clusters have been accumulated, we read back the frame buffer and recombine positive and negative parts of the signal (by adding (R,G,B) components together) and add the recombined signals together to form the pre-mixed waveform.

6.3 Discussion

Being able to premix each source individually has several advantages. First, we can preserve the delay and attenuation of each source, ensuring a correct distribution of the energy reaching the listener through time. Doing so will preserve most of the spatial cues associated with the perceived size of the cluster. Fig. 8 shows a comparison of the energy distribution through time reaching a listener (echogram). The top echogram is obtained when all sources are treated individually. In the lower echogram, all sources in each cluster have been delayed by the delay corresponding to the cluster's representative, which would be the naive solution, avoiding resampling for each source. Note how different the time structure is. In the second case, the impression of spatial "width" associated with the cluster would be lost. A second advantage of performing pre-mixing prior to audio hardware rendering is that we can achieve more effects currently not (or poorly) supported (arbitrary directivity patterns, time delays, *etc.*) in existing audio hardware or API's. For instance, in our video, we used custom directivity patterns for the wheels of the train and voice of the pedestrians.

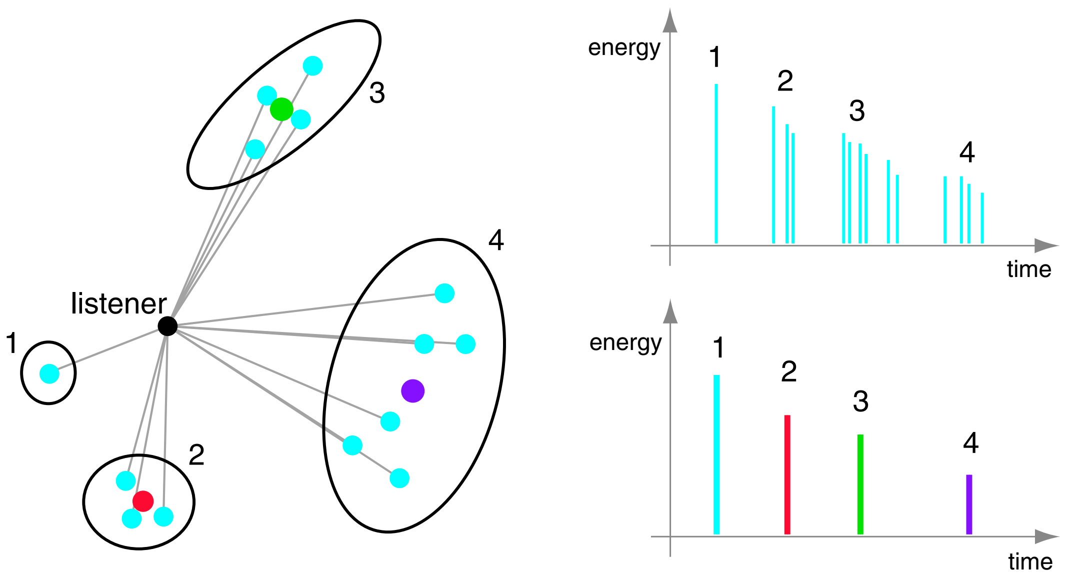

Figure 8: Left: a configuration of 4 clusters. Right (top): echogram of individually premixed sources; (bottom) echogram using cluster representative for delay and attenuation.

7 Implementation and Results

Our complete system, including our three-step algorithm and the spatialization back-end, have been implemented on *Windows* and *Linux* platforms with both software and hardware spatialization for clusters. Software spatialization processing was carried out using our own API. For hardware spatialization, we used *Direct Sound 3D*.

We have implemented a simple graphics viewer in OpenGL, but no acceleration techniques (such as view-frustum or occlusion culling, *etc.*) have been included.

7.1 Spatializing clustered signals

The pre-mix signals generated by our three-step approach, for each cluster along with cluster representatives, can be used to auralize the audio scene in a standard spatialized audio system.

Each cluster is considered as a point-source located at the position of its representative. Any type of spatial sound reproduction strategy (panning, binauralization, *etc.*) applicable to a point source model can thus be used.

Spatialization can be done in software, limiting the cost of HRTF processing to the number of clusters. More interestingly, it can be done using standard "game-audio" APIs such as *Direct Sound*. In this case an hardware 3D audio buffer can be created for each cluster and fed with the pre-mixed signals. The sound buffer is then positioned at the location of the representative (e.g., using Direct Sound's *SetPosition* command).

We can also benefit from additional processing offered by current consumer audio hardware, such as artificial reverberation processing.

7.2 Statistics and Timings

Figure 9 shows the performances of the clustering algorithm for several input sources and cluster number. The positions of the sources were randomly generated in a 100mx100mx100m cube centered around the listener.

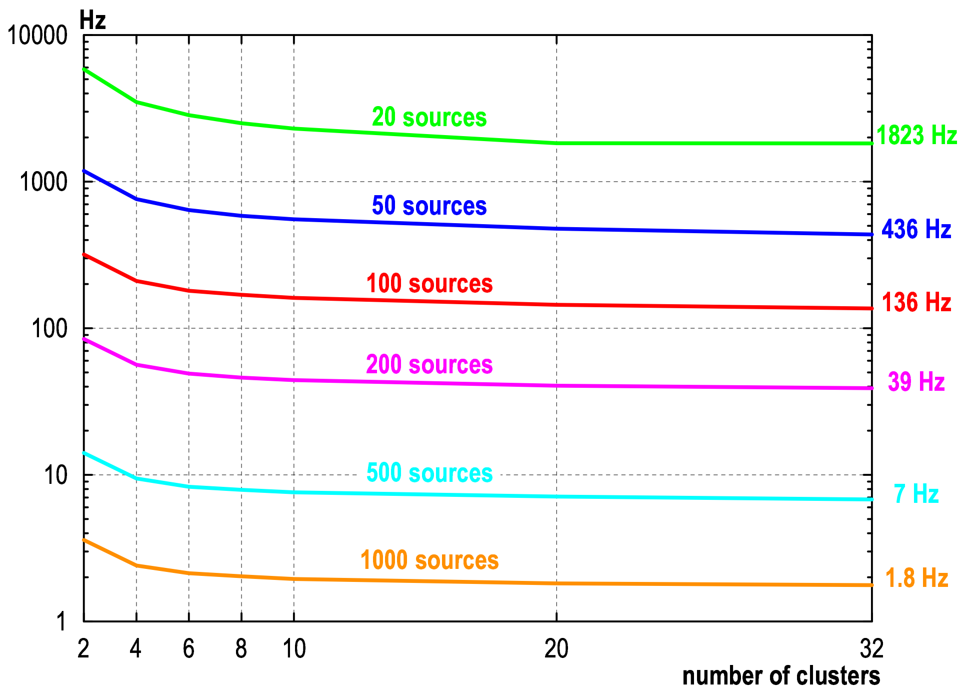

Figure 9: Performance plots for our clustering algorithms. We plot the clustering averaged refresh rate versus the number of clusters for different numbers of sources. Note that the refresh rate rapidly becomes almost constant when cluster number increases.

For perceptual masking, preprocessing takes 0.33 seconds for 1 second of audio input sampled at 44.1 KHz (CD quality). The online culling process takes between 1.5-2.2 msecs in the examples tested. For a scene of 162 sources, clustering takes between 7 to 10 msec per frame on average. To estimate the load of our premixing on the graphics hardware we turned off audio processing on the GPU for our station scene. The video frame rate with audio processing was 38 Hz and without our approach was 30Hz, which gives an indication of the load of the method on the graphics pipeline.

If we do not use the graphics hardware, our tests indicate that we can render on the order of 50-60 sound sources, with simple stereo panning. In the results we show, we have rendered up to 162 sources.

7.3 Example Scene

As can be seen on the video, our technique can be used to render complex audio-visual environments. The video was shot directly out of a dual *Xeon* 1.8 GHz PC with *GeForce 4600 Ti* graphics and a *Soundblaster Audigy* audio board. We also ran our demos on a laptop with a built-in *ATI Radeon Mobility 5700* graphics board and a *SoundMax* audio board. Our test scene is composed of a station model, two trains, and 60 animated models of humans walking. The scene contains around 70,000 polygons, uses 17Mb of texture memory, and 31.5 Mbs of audio samples, corresponding to about 4 minutes of CD quality audio. In the tests shown on the video, we have between 142 and 162 sound sources. We can achieve rendering of about 100 sources on a laptop computer. As can be heard on the video, the quality of the audio rendering is still quite high.

8 Discussion

Unlike previous clustering strategies, our algorithm adapts to a specified number of clusters in a locally optimal way. Thus it can be used for dynamic resource allocation.

As presented above, we have seen that the hardware-mapped audio processing does not present a significant overhead for the graphics engine. This is clearly demonstrated by the visual complexity of the environment used for our tests. Nonetheless, the fact that premixing happens on a different computer component than the audio hardware does add a certain amount of overhead in terms of data transfer on the system bus. Our tests suggest that this limits our ability to use hardware premixing to about 20-30 clusters, depending on the hardware used.

Informal tests using our system indicate that for complex auditory environments, even a small number of clusters result in high quality results. The most significant problem with audio quality currently stems from the fact that current graphics hardware uses uses 8 bits per pixel frame-buffers (in particular for blending operations). For audio data this is somewhat insufficient and results in audible quantization noise. Nonetheless, we do not consider this to be a fundamental drawback of our approach, since next-generation graphics cards will support extended resolution frame buffers.

Finally, it would be possible to use randomized *k*-means [18], to randomly choose the cluster centroids in the splitting phase, allowing the treatment of a higher number of sources. This would speed up the clustering algorithm but would result in a less optimal solution. Early experiments conducted with this approach show that, increasing the number of pedestrians, about 225 sources can be rendered in our test example. Although the obtained clusters can lead to a greater error, the audio quality remained very close to the original solution. For more sources, the video frame rate decreases significantly. However, we suspect that graphics rendering is the limitation in this case.

8.1 Relevance to image rendering

The two first aspects of our approach, namely perceptual masking and clustering have their analogues in image rendering. Notably, [14], use a similar approach to precompute elevation maps of the Visual Difference Predictor masking operator, so that an on-line evaluation can be performed.

More interestingly, our clustering approach is in essence an algorithm for the on-line construction of a spatial subdivision. Spatial subdivision structures are useful in ray-tracing or radiosity algorithms, but are typically restricted to static scenes. With the appearance of interactive ray-tracing approaches [36], the ability to dynamically build a spatial subdivision structure on-line with an approach similar to our could prove extremely interesting.

Finally, this work also suggests that hardware graphics and audio pipelines could share resources such as memory or processing. One could think of an integrated *A/V rendering chip* that could process audio or graphics textures and render directly to memory accessible to the dedicated audio processing unit. Integrated chipsets such as the *NVidia Nforce*™ could be a starting step in this direction.

9 Conclusion and future work

We have presented a three-step approach which significantly accelerates spatialized audio rendering for complex virtual environments. The approach first applies perceptual masking to cull unimportant sources, then clusters sources using perceptual criteria, and finally leverages graphics hardware to accelerate audio processing operations.

Our results suggest that processing for audio rendering could be distributed in a different manner between the different hardware components: pre-mixing operations for audio could be done using the graphics oriented hardware while dedicated audio hardware would handle specific effects such as artificial reverberation. Thus graphics hardware could be used as an efficient DSP engine for high quality audio calculations based on pixel shader implementations. This could be used by professional audio programs (e.g., sequencers) to support hardware accelerated real-time effects on consumer-grade workstations without the need for expensive ad-hoc DSP boards.

As demonstrated in our results section, extended sources are often present in real environments and significantly impact the complexity involved. We would like to investigate automatic modeling of extended sources through measurements or simulations.

Finally, other clustering metrics should be evaluated, taking into account other parameters of the signals (pitch, harmonicity, *etc.*). A more thorough psychophysical evaluation of our rendering pipeline is of course required. We plan to conduct this study with experts in the field in the near future.

References

[1] Creative Labs Soundblaster©. http://www.soundblaster.com.

[2] Direct X homepage, ©microsoft.
http://www.microsoft.com/windows/directx/default.asp.

[3] Environmental audio extensions: EAX 2.0 Creative©. http://www.soundblaster.com/eaudio.

[4] ZoomFX, MacroFX, Sensaura©. http://www.sensaura.co.uk.

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

[6] J. Blauert. *Spatial Hearing : The Psychophysics of Human Sound Localization*. M.I.T. Press, Cambridge, MA, 1983.

[7] J. Borish. Extension of the image model to arbitrary polyhedra. *J. of the Acoustical Society of America*, 75(6), 1984.

[8] K. Brandenburg. mp3 and aac explained. *AES 17th International Conference on High-Quality Audio Coding*, September 1999.

[9] A.S. Bregman. *Auditory Scene Analysis, The perceptual organization of sound*. The MIT Press, 1990.

[10] H. Chen, G. Wallace, A. Gupta, K. Li, T. Funkhouser, and P. Cook. Experiences with scalability of display walls. *proceedings of the Immersive Projection Technology (IPT) Workshop*, March 2002.

[11] J. Chen. *Auditory Space Modeling and Virtual Auditory Environment Simulation*. PhD thesis, University of Wisconsin-Madison, Madison, WI, 1992.

[12] 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.

[13] L. Dahl and J.M. Jot. A reverberator based on absorbent all-pass filters. *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy*, December 2000.

[14] Reynald Dumont, Fabio Pellacini, and James A. Ferwerda. A perceptually-based texture caching algorithm for hardware-based rendering. In *Eurographics Workshop on Rendering*, 2001.

[15] D.P.W. Ellis. A perceptual representation of audio. *Master's thesis, Massachusets Institute of Technology*, 1992.

[16] 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), Xerox Palo Alto Research Center, Palo Alto, USA*, 1997.

[17] P. Fränti, T. Kaukoranta, and O. Nevalainen. On the splitting method for vector quantization codebook generation. *Optical Engineering*, 36(11):3043–3051, 1997.

[18] P. Fränti and J. Kivijärvi. Randomised local search algorithm for the clustering problem. *Pattern Analysis and Applications*, 3:358–369, 2000.

[19] T. Funkhouser, P. Min, and I. Carlbom. Real-time acoustic modeling for distributed virtual environments. *ACM Computer Graphics, SIGGRAPH'99 Proceedings*, pages 365–374, August 1999.

[20] A. Gersho and R.M. Gray. *Vector quantization and signal compression, The Kluwer International Series in Engineering and Computer Science, 159*. Kluwer Academic Publisher, 1992.

[21] 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.

[22] Jens Herder. Visualization of a clustering algorithm of sound sources based on localization errors. *The Journal of Three Dimensional Images, 3D-Forum Society*, 13(3):66–70, September 1999.

[23] I.J. Hirsh. The influence of interaural phase on interaural summation and inhibition. *J. of the Acoustical Society of America*, 20(4):536–544, 1948.

[24] F. Filipanits Jr. Design and implementation of an auralization system with a spectrum-based temporal processing optimization. *Master thesis, Univ. of Miami*, 1994.

[25] P. Larsson, D. Västfjäll, and M. Kleiner. Better presence and performance in virtual environments by improved binaural sound rendering. *proceedings of the AES 22nd Intl. Conf. on virtual, synthetic and entertainment audio, Espoo, Finland*, pages 31–38, June 2002.

[26] P. Larsson, Daniel Västfjäll, and Mendel Kleiner. Ecological acoustics and the multi-modal perception of rooms: real and unreal experiences of auditory-visual virtual environments. *proceedings of the 2001 International Conference on Auditory Display (ICAD2001), Espoo, Finland*, pages 245–249, 2001.

[27] A. Likas, N. Vlassis, and J.J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

[28] E. M. Painter and A. S. Spanias. A review of algorithms for perceptual coding of digital audio signals. *DSP-97*, 1997.

[29] Eric Paquette, Pierre Poulin, and George Drettakis. A light hierarchy for fast rendering of scenes with many lights. *EUROGRAPHICS'98*, 17(3), 1998.

[30] T.J. Purcell, Ii. Buck, W.R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[31] R. Rangachar. Analysis and improvement of the MPEG-1 audio layer III algorithm at low bit-rates. *Master thesis, Arizona State Univ.*, December 2001.

[32] 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.

[33] Ken Steiglitz. *A DSP Primer with applications to digital audio and computer music*. Addison Wesley, 1996.

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

[35] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom. Modeling acoustics in virtual environments using the uniform theory of diffraction. *ACM Computer Graphics, SIGGRAPH'01 Proceedings*, pages 545–552, August 2001.

[36] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3), 2001. ISSN 1067-7055.

[37] E. Wenzel, J. Miller, and J. Abel. A software-based system for interactive spatial sound synthesis. *Proceeding of ICAD 2000, Atlanta, USA*, april 2000.

[38] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer, 1999.

A Precomputing masking information

As a starting point to our perceptual audio rendering pipeline we must precompute frequency-dependent information about the content and masking capabilities of our signals. We chose to precompute information over 3 frequency bands corresponding to low, medium and high frequencies. Although this is far less than the 25 critical bands used in audio compression, we found it worked well in practice for our application. For each input signal, we generate: 1) three band-passed copies of the signal, 2) the instantaneous short-time power spectrum and masking thresholds. In our implementation we used short (256 taps) band-pass finite impulse response filters to generate the three band-passed versions of the signal [33].

Based on techniques used in perceptual audio coding [28, 8, 31], we compute for each of the three bands a short time Fourier transform. We used 1024 sample long Hann-windowed frames with 50% overlap. We store for each band f its instantaneous power spectrum distribution, $\mathbf{PSD}_t(f)$, for each frame t. From the **PSD**, we estimate a logscale *spectral flatness measure*, $\mathbf{SFM}_t(f)$, of the signal as:

$$\mathbf{SFM}_t(f) = 10\ log10\left(\frac{\mu_g(\mathbf{PSD}_t(f))}{\mu_a(\mathbf{PSD}_t(f))}\right),$$

where μ_g and μ_a are respectively the geometric and arithmetic mean of the PSD over all frequencies. We then estimate the *tonality index*, $\mathbf{T}_t(f)$, as:

$$\mathbf{T}_t(f) = \frac{max(\mathbf{SFM}_t(f), 0)}{-60} - 1.$$

We clamp this value in the interval $[0,1]$. The tonality index is an indication of the signal's noisiness: low $\mathbf{T}_t(f)$'s indicate a noisier component. Finally, similar to the *mp3* standard [31], we estimate the masking threshold, $\mathbf{M}_t(f)$ (in dBs) as:

$$\mathbf{M}_t(f) = 31 * \mathbf{T}_t(f) + 12 * (1 - \mathbf{T}_t(f)),$$

This masking threshold represents the limit below which a maskee is going to masked by the considered signal.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399