Procedural audio modeling for particle-based environmental effects

Charles Verron, George Drettakis

Procedural audio modeling for particle-based environmental effects

Charles Verron, George Drettakis
REVES-INRIA, Sophia-Antipolis, France

Draft for 133rd AES Convention, October 2012

1 abstract

We present a sound synthesizer dedicated to particle-based environmental effects, for use in interactive virtual environments. The synthesis engine is based on five physically-inspired basic elements which we call sound atoms, that can be parameterized and stochastically distributed in time and space. Based on this set of atomic elements, models are presented for reproducing several environmental sound sources. Compared to pre-recorded sound samples, procedural synthesis provides extra flexibility to manipulate and control the sound source properties with physically-inspired parameters. In this paper, the controls are used to simultaneously modify particle-based graphical models, resulting in synchronous audio/graphics environmental effects. The approach is illustrated with three models, that are commonly used in video games: fire, wind, and rain. The physically-inspired controls simultaneously drive graphical parameters (e.g., distribution of particles, average particles velocity) and sound parameters (e.g., distribution of sound atoms, spectral modifications). The joint audio/graphics control results in a tightly-coupled interaction between the two modalities that enhances the naturalness of the scene.

2 Introduction

In the last two decades advances in all fields of computer graphics (i.e., modeling, animation, rendering and more recently imaging) has resulted in impressive advances in realism, even for real-time virtual environments. Paradoxically, sound in virtual environments is still usually based on pre-recorded sound samples. Procedural sound synthesis is an attractive alternative to increase the sense of realism in interactive scenes [1]. Compared to pre-recorded sounds, it allows interactive manipulations that would be difficult (if not impossible) otherwise. In particular, procedural audio parameters can be linked to motion parameters of graphics objects [2, 3] to enhance the sound/graphics interactions. Nevertheless, the use of sound synthesis is still limited in current video games, probably because of three major challenges that are difficult to fulfill simultaneously: synthesis quality should be equivalent to pre-recorded sounds, synthesis should offer flexible controls to sound designers, and its computational cost should satisfy real-time constraints.

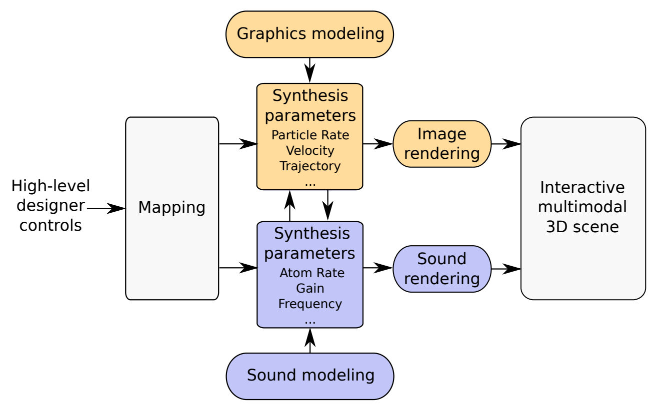

Figure 1: *Environmental sound synthesis for particle-based environmental effects. A mapping between graphics and sound parameters allows tightly-coupled audio/graphics interactions.*

Parametric sound synthesis techniques can be decomposed into two main families: physical models, aiming at simulating the physics of sound sources, and signal models aiming at reproducing perceptual effects independently of the source [4]. For environmental sounds, physical approaches are of great interest. Some authors have successfully used physical models to reproduce specific environmental sounds such as wind [5] fire [6] rolling [7] or liquids [8]. Nevertheless, the physics of these phenomena is often complicated. It requires the knowledge of various objects' characteristics and their possible interactions with surrounding gases, liquids and solids. A purely physical approach for sound synthesis is currently impossible in video games, due to the difficulty of designing a general physical model for a wide variety of environmental sounds, in addition to the high computational cost. On the other hand, studies on environmental sounds [9] suggest that synthetic signals matching relatively simple properties could give good results in terms of perceived realism.

In the computer graphics community, many models have been proposed for generating environmental effects (deformable objects, liquids, smoke, etc.) leading to impressive results (see e.g., [10] for a review). Depending on the approach, the computation cost is sometimes too high to satisfy real-time constraints. An efficient technique was introduced in [11] to simulate "fuzzy" phenomena like fire and smoke, by using dynamic collections of particles. The designer has access to meaningful parameters (e.g., number of particles, mean velocity) to control stochastic processes that define the evolution of particles over time. Recent efficient GPU and parallel implementations allow the generation of up to millions of particles in real time [12, 13]. Physical information is also included to model realistic movements of particles, and particle interactions [10]. The approach has been successfully applied for a wide range of phenomena [14, 15, 16, 17] (e.g., water, clouds, smoke, electricity, explosions, crowds, magic etc.) and is still very popular in current video games [18, 19]. Curtis Roads noticed that particle systems share many similarities with granular sound synthesis, which models a sound as a collection of short audio grains distributed in time and space [20]. To our knowledge this similarity has not yet been exploited to propose a sound synthesizer dedicated to particle-based environmental effects.

In this paper we propose a signal-based synthesis approach, and focus on the perceptual control of the generated sounds. Since most particle systems in games are based on simple stochastic laws, the associated sound models should not rely on physical solvers delivering collision detection, fluid dynamics, etc. Instead, we follow a "physically informed" synthesis approach [21, 1] and propose an efficient implementation, based on an atomic representation, suitable for several types of phenomena and in particular rain, fire and wind. This approach results in plausible sound quality and has the advantage of fulfilling real-time constraints. Furthermore, the sound models offer intuitive controls to sound designers, making the synthesizer suitable for practical artistic scenarios. A mapping is described to connect the sound synthesis parameters to particle systems and produce relevant audio/graphics interactions. The approach is illustrated in figure 1. It has been validated with several examples (see the online videos accompanying this paper [22]) in a realtime implementation.

The paper is divided into three parts. First we propose sound synthesis models based on five classes of atoms, and their parameterization to simulate rain, fire and wind effects. Then the real-time synthesis and spatialization pipeline is described. In the third part, we show how the sound models are connected to particles systems for producing multimodal environmental effects.

3 Synthesis models

In his pioneering work on synthesis and classification of environmental sounds [23] Gaver proposed three main categories depending on the physics of the source: vibrating solids, gasses and liquids. Even if these sounds refer to a wide range of physical phenomena, their acoustic morphology calls for common signal characteristics, allowing for a granular-like synthesis approach. Five "sound atoms" were defined in [24] as primitives to reproduce a variety of environmental sounds in the three categories defined by Gaver. In this paper, we rely on this set of atoms to reproduce the micro structure of rain, fire and wind sounds.

3.1 Rain sound model

Rain sounds result from drops falling on different surfaces (leaves, solid floor, water...) producing a wide variety of impact sounds [1]. Depending on the surface hit by the drops, three main types of impacts may be distinguished. Drops falling on water produce bubble sounds [25] that can be simulated by a chirped sinusoid (the chirped impact atom) with amplitude a and exponential decay α:

$$x_1(t) = a \sin\left(2\pi \int_0^t f(\nu) d\nu\right) e^{-\alpha t}$$

where the instantaneous frequency f varies linearly over time.

Alternatively, drops falling on resonant surfaces (e.g., plates, windows...) trigger an oscillating system with fixed resonant frequencies. The resulting harmonic impact sounds can be simulated by a modal impact atom

$$x_2(t) = \sum_{m=1}^{M} a_m \sin\left(2\pi f_m t\right) e^{-\alpha_m t}$$

where f_m are the modal frequencies of the impacted object and M is the number of simulated components. The amplitudes a_m depend on the excitation point, and the decay factors α_m are characteristic of the material [26].

Drops falling on rigid or deformable objects (e.g., stone, leaves) that exhibit a noisy response (high modal density) tend to produce a brief noisy sound which is perceptually different from bubbles and sinusoidal impacts. Such sounds are efficiently reproduced by a noisy impact atom, which is a sum of 8 contiguous subbands of noise $s_i(t)$, evenly spaced on the Equivalent Rectangular Bandwidth (ERB) scale, with amplitude a_i and exponential decay α_i:

$$x_3(t) = \sum_{i=1}^{8} a_i s_i(t) e^{-\alpha_i t}$$

Additionally, an equalized noise atom is used to create a rain background noise

$$x_4(t) = \sum_{i=1}^{32} a_i(t) s_i(t)$$

where $s_i(t)$ are 32 contiguous subbands of noise evenly spaced on the ERB scale, with amplitudes $a_i(t)$. This allows the simulation of a huge number of simultaneous drops with a low computational cost. The 32-subband amplitudes are extracted from rain sound samples with the method described in [27]. The rain sample is passed through a 32 ERB subband filterbank, then a_i is set as the time average energy in subband i.

Atoms x_1, x_2, x_3 and x_4 produce the basic microstructure of the rain sound model. They are distributed over time and space to simulate a wide diversity of rain situations. Three user controls $GainWater$, $GainSolids$ and $GainLeaves$ specify the maximum level of drops falling on water, solids and leaves respectively. Similarly $RateWater$, $RateSolids$ and $RateLeaves$ set the probability of falling drops per unit-time (i.e., per frame). An additional user control $GainBackground$ sets the global gain of the rain background noise.

For real-time synthesis, the synthesis parameters are initialized for a population of 100 chirped impacts with different frequencies and amplitude, following the laws proposed in [25]. Similarly, the initial parameters of 100 noisy and/or modal impacts are set to precomputed values, previously extracted from rain sound samples so as to reproduce "plausible" drop sounds. At run-time, impacts are synthesized in real time from their initial synthesis parameters. Integration of the rain user controls within the synthesis process is illustrated by the following pseudocode:

```
function PROCESSRAIN
  for each frame l do

    // Drops
    if rand()< RateWater then
      triggerOneChirpedImpact(rand().GainWater)
    if rand()< RateSolids then
      triggerOneModalImpact(rand().GainSolids)
    if rand()< RateLeaves then
      triggerOneNoisyImpact(rand().GainLeaves)

    // Background noise
    for subband i = 1 → 32 do
      a = a_i . GainBackground
      setEqualizedNoise_subband(i, a)
    end for

  end for
end function
```

where $rand()$ is a random number uniformly distributed between 0 and 1, the three trigger functions synthesize impacts with the given amplitude, and setEqualizedNoise_subband(i, a) synthesizes the i^{th} subband of the equalized noise atom with amplitude a.

3.2 Fire sound model

The fire sound is synthesized as a combination of noisy impact atoms to simulate crackling, and equalized noise to simulate the combustion (flames) noise.

Noisy impact parameters (i.e., subband amplitudes and decays) were defined to approximate real crackling sounds. Due to the complexity of these signals which are noisy and non-stationary, manual intervention was required to set the range of plausible parameters (as for noisy raindrop sounds). Five prototype spectral envelopes with eight ERB subbands were extracted from real-world crackling sound examples, along with three amplitude decays representing three categories of crack sizes: big, medium and small. For simplicity, a single decay is used in the eight subbands of each noisy impact.

To reproduce combustion noise, the 32 subband amplitudes a_i of the equalized noise are extracted from a fire sound sample (as described in [27]) and averaged over time to get a constant spectral envelope.

For real-time synthesis, 100 noisy impacts are initialized with one of the precomputed parameter sets. Then the user controls $GainCrackling$ and $RateCrackling$ to set respectively the maximum gain and the probability of crackling per unit-time. The user-control $GainCombustion$ sets the gain of the combustion noise. A low frequency noise $b(t)$ is also introduced to modulate the energy in the first four subbands, to increase the variations of the sound and continuously reproduce changing flame sizes. This modulation can be efficiently achieved by filtering a white noise by a low-pass filter with a very low cutoff frequency (around 1Hz) as suggested in [1]. The following pseudo-code illustrates the fire synthesis process:

```
function processFire
  for each frame l do

    // Crackling
    if rand()< RateCrackling then
      triggerOneNoisyImpact(rand().GainCrackling)

    // Combustion noise
    b(l) =lowPass(rand()) // random modulation
    for subband i = 1 → 4 do
      a = a_i . GainCombustion . (1 + b(l))
      setEqualizedNoise_subband(i, a)
    end for
    for subband i = 5 → 32 do
      a = a_i . GainCombustion
      setEqualizedNoise_subband(i, a)
    end for

  end for
end function
```

3.3 Wind sound model

A signal model based on time-varying bandpass filters for simulating wind sounds was proposed in [1]. We adapted this model to our architecture, producing a wind sound by the addition of several band-limited noises that simulate wind resonances. Each band-limited noise atom is defined by its time-varying spectral envelope:

$$X_5(f) = \begin{cases} A(t) & \text{if } |f - F(t)| < \frac{B(t)}{2} \\ A(t)e^{-\alpha(t)\left(|f-F(t)|-\frac{B(t)}{2}\right)} & \text{otherwise} \end{cases}$$

where $F(t)$ is the center frequency, $A(t)$ the gain, $B(t)$ the bandwidth and $\alpha(t)$ the filter slope. The amplitude A and center frequency F of each atom are set in real time by a single user control $WindSpeed$ as follows:

$$A(t) = A_i \ . \ WindSpeed \ . \ \big(1 + b(t - \tau_i)\big)$$

where A_i and τ_i are constant values (different for each atom i) that represent the relative amplitude and propagation time to the listener, and

$$F(t) = C_i \ . \ A(t) + F_i$$

where F_i and C_i are respectively the frequency offset and deviation constants. This way, the center frequency and amplitude of the band-limited components are proportional to the $WindSpeed$ user control. The modulation $b(t)$ is a low frequency noise with a cutoff frequency around 1 Hz, that introduces plausible variations in the wind sound, as described for the fire combustion noise.

To reproduce different types of wind sounds, from broadband (e.g., wind in the trees) to narrowband phenomena (e.g., wind whistling in a window) the bandwidth and slope of each atom can be adapted intuitively by the sound designer via the $WindType$ user control. This control linearly interpolates between several preset values previously created for $[\alpha_i, B_i, A_i, \tau_i, C_i, F_i]$.

The sound of rustling leaves in the trees is also simulated, as a combination of noisy impacts and equalized noise (to simulate a huge number of leaves) parameterized with the method described above for rain background noise. By default $RateWindLeaves$ and $GainWindLeaves$ (i.e., rate and gain of noisy impacts and equalized noise) are set proportionally to $WindSpeed$. The general synthesis process is illustrated by the following pseudo-code:

```
function processWind
  for each frame l do

    // Wind band-limited noises
    b(l) =lowPass(rand()) // random modulation
    for each noise i do
      [α_i, B_i, A_i, τ_i, C_i, F_i] =
                  interpolatePresets(i,WindType)
      A = A_i . WindSpeed . (1 + b(l − τ_i))
      F = C_i . A + F_i
      setBandlimitedNoise(A, F, α_i, B_i)
    end for

    // Rustling leaves
    if rand()< RateWindLeaves then
      triggerOneNoisyImpact(rand().GainWindLeaves)
    for subband i = 1 → 32 do
      a = a_i . GainWindLeaves
      setEqualizedNoise_subband(i, a)
    end for

  end for
end function
```

In summary the rain, fire and wind models require five classes of sound atoms, whose low-level parameters are listed in table 1. These atoms are the core components of the environmental sound synthesizer.

4 Spatialized synthesis engine

Rain, fire and wind sounds are created as a combination of time-frequency atoms $x(t)$. Each atom is modeled as a sum of sinusoidal and noisy components, noted respectively $s_D(t)$ and $s_S(t)$. This allows the use of efficient synthesis algorithms based on the inverse fast Fourier transform (IFFT) to generate the atoms. Additionally, IFFT synthesis is combined

Atom	Parameters
Modal impact	a_m initial amplitudes α_m decays f_m frequencies
Noisy impact	$[a_1 ... a_8]$ subband amplitudes $[\alpha_1 ... \alpha_8]$ subband decays
Chirped impact	f_0 initial frequency σ linear frequency shift α decay
Band-limited noise	$F(t)$ center frequency $B(t)$ bandwidth $\alpha(t)$ filter slope $A(t)$ amplitude
Equalized noise	$[a_1(t) ... a_{32}(t)]$ amplitudes

Table 1: The five classes of atoms used for the sound synthesis models, with their respective parameters.

with 3D audio modules in the frequency domain, following the efficient approach described in [28]. The complete synthesis/spatialization pipeline is depicted in figure 2 and each part of the process is described in more detail below.

Time-frequency synthesis The synthesis of each source is realized by blocks in the frequency domain. At each block l, an approximation of the short-time Fourier transform of the atoms is built by summing the deterministic S_D^l or stochastic S_S^l contributions. Real and imaginary parts of the deterministic short-time spectrum (STS) are computed by convolving the theoretical ray spectrum formed by the M sinusoidal components of amplitude a_m^l, frequency f_m^l and phase Φ_m^l, with the "spectral motif" W which is the Fourier transform of the synthesis window $w[n]$ as described in [29]:

$$S_D^l[k] = \sum_{m=1}^{M} a_m^l e^{j\Phi_m^l} W(\frac{k}{N} - f_m^l) \quad (1)$$

N being the number of frequency bins (i.e., the synthesis window size) and k the discrete frequency index (i.e., $W[k] = W(\frac{k}{N})$). We use two synthesis window sizes N in parallel to produce the impulsive impacts ($N = 128$) and the continuous atoms ($N = 1024$). The real and imaginary parts of the stochastic STS are computed by summing the subband spectral envelopes of the atoms, multiplying by two noise sequences, and circularly convolving the result with the spectral motif W. The final STS $X^l[k]$ is obtained for each source by summing the deterministic and stochastic contributions in the frequency domain.

Integrated spatialization The architecture of the synthesizer is designed for easily extending the perceived width of sound sources. Rain, fire and wind sounds are formed by a collection of secondary sources spatialized around the listener. Two approaches are used to distribute the synthesized atoms into secondary sources STS. Impulsive atoms (i.e., chirped, modal and noisy impacts) correspond to phenomena typically localized in space (i.e., raindrop sounds and fire crackling). For this reason, each of them is integrated into one single STS. On the other hand, continuous noisy atoms (i.e., band-limited and equalized noises) correspond to naturally diffuse phenomena (i.e., rain background noise, fire combustion noise and wind band-limited noises). Consequently, decorrelated copies of these atoms are integrated into all the secondary sources STS, producing a diffuse noise around the listener. The decorrelation is achieved by using different sequences of noise when building the stochastic component of each atom.

A supplementary STS, common for all sources is provided for the reverberation, which is efficiently implemented by a multichannel Feedback Delay Network [30]. Spatialization of each secondary source is based on a multichannel encoding/decoding scheme (see [28] for more details). The C-channel encoding consists in applying real-valued spatial gains $(g_1, \ldots, g_C)$ to the STS $X_i^l[k]$, simulating the position (θ_i^l, Ψ_i^l) of the i^{th} point source. The gains are computed with an amplitude-based panning approach such as VBAP [31] or Ambisonics [32, 33]. The encoded multichannel STS are then summed together, channel by channel, producing a single block Y^l with

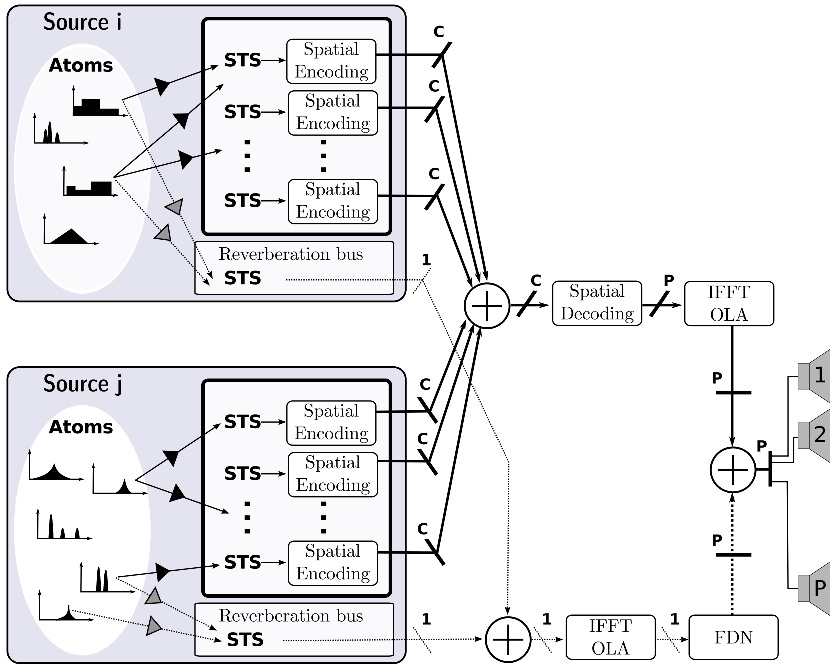

Figure 2: Architecture of the synthesis/spatialization engine. Deterministic and stochastic components of the atoms are added in the frequency domain to form a collection of spectra (STS) that are spatialized around the listener (spatial encoding/decoding with C intermediate channels). Inverse fast Fourier transform (IFFT) and overlap-add (OLA) are performed to get the multi-channel signal for an arbitrary setup of P loudspeakers (or headphones). A supplementary short-time spectrum is dedicated to the reverberation, common for all sources in the scene and implemented by a Feedback Delay Network (FDN) in the time domain (1 input channel, P output channels).

C channels:

$$Y_c^l[k] = \sum_{i=1}^{I} g_c(\theta_i^l, \Psi_i^l) X_i^l[k]$$

where g_c is the c^{th} position-dependent gain and I is the total number of sources. Spatial decoding performs linear combinations of the channels of Y^l to get the signals for the P loudspeakers. It depends on the chosen panning method and on the loudspeaker setup. Finally, the decoded channels are inverse fast Fourier transformed and overlap-added to reconstruct the synthetic signals $x_p[n]$ for the P output channels:

$$x_p[n] = \sum_{l=-\infty}^{\infty} g_p(\theta_i^l, \Psi_i^l) w[n-lL] \Big(s_D^l[n-lL] + s_S^l[n-lL] \Big)$$

where s_D^l and s_S^l are the sum of all atomic components (deterministic and stochastic) at block l and L is the synthesis hop size. For rendering over headphones ($P = 2$) we use Head Related Transfer Functions (HRTF) from the Listen[1] database to simulate $C = 18$ virtual loudspeakers in the spatial decoding stage.

[1] http://www.ircam.fr/equipes/salles/listen

5 Coupling with particle systems

The advantage of sound modeling compared to sample-based approaches lies in the flexibility of transformations offered by the synthesis parameters. Stochastic control of sound atoms were defined in section 3 to provide high-level physically-inspired manipulation of the rain, fire and wind phenomena. Here we present the mapping of audio and graphics parameters for particle-based effects.

5.1 Principles of particle systems

At each frame of the animation sequence, the basic steps of a particle system are as follows [11]: new particles are generated with a set of attributes, particles that have existed for a certain amount of time are destroyed, and remaining particles are transformed and moved according to their dynamic attributes. Initial attributes of particles are their position, velocity and acceleration, along with their size, color, transparency, shape and lifetime. To change the dynamics and appearance of the particle system, the designer has access to a set of controls that affect the mean m and maximum deviation δ of particle initial attributes. Typically, a particle attribute $p \in \mathbb{R}^N$ is defined as:

$$p = m + rand().\delta$$

where m and $\delta \in \mathbb{R}^N$ (respectively the mean and maximum deviation) are the designer controls, while rand() is a random number uniformly distributed between -1 and 1. Simple uniform stochastic processes have the advantage of being intuitive to manipulate. As an example, if the designer sets $m = [10, 0, 0]$ and $\delta = [0.5, 0.5, 0.5]$ for the initial position parameter, then the particles are randomly created inside a unit cube centered at $[10, 0, 0]$.

With these simple principles, particle systems are used as real-time building components for a wide range of environmental phenomena. The size and shape of each particle can be designed to approximate individual raindrops to simulate complex rain environments [34, 35, 36, 37, 38]. Similarly, flame and smoke particles are used to simulate fire [39, 40, 41] and leaf-like particles to simulate wind [42].

5.2 Audio/graphics controls

To decrease computational cost, particle systems often do not provide collision detection for individual particles. For this reason we do not use such information for coupling audio and graphics, and focus on high-level manipulations of sound and graphics components. This approach leads to flexible audio/graphics controls that are suitable for practical artistic scenarios.

Rain Intensity In our implementation, rain is produced by particles with initial positions randomly distributed at the top of the virtual world (concentric circles around the player) with vertical initial velocities. The $RainIntensity$ parameter dynamically controls the number of graphics particles (raindrops) emitted by the particle system per unit-time (particle spawn rate). A linear mapping between $RainIntensity$ and the rain sound parameters provides a simple way to match graphics and sound rendering. Specifically, $RainIntensity$ is linked to the raindrop rate via $RateWater$, $RateSolids$ and $RateLeaves$, along with their gain $GainWater$, $GainSolids$ and $GainLeaves$. It also controls the background noise level via $GainBackground$ (see section 3.1). For more flexibility the mapping can be edited by the sound designer to achieve the desired result (piece-wise linear mapping, exponential, etc.). The expected number of drops falling on water, leaves or solids can be set separately, which allows the specification of zones with different characteristics (e.g., more water or more foliage).

Fire Intensity For the fire simulation, a bunch of flame-like and smoke-like particles are projected above the fireplace, with varying initial positions and velocities. $FireIntensity$ controls the spawn rate of these particles (i.e., the expected number of particles per unit-time). Simultaneously, we map this parameter to control in real time the rate and gain of crackling via $RateCrackling$ and $GainCrackling$ and the gain of flame noise via $GainCombustion$ (see section 3.2). As for the rain, the mapping can be edited by the designer to adjust the desired behavior. The joint audio/graphics control is illustrated in figure 3.

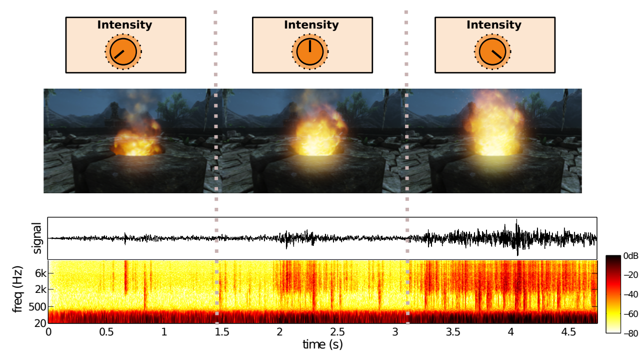

Figure 3: *Audio/graphics high-level control of a fire. The control Intensity changes the rate and gain of noisy impacts, and the combustion noise of the fire sound model. Simultaneously, it controls the flame/smoke particle spawn rate for the graphics simulation.*

Wind Speed In our system, wind is simulated by leaf-like particles moving around the player. The parameter $WindSpeed$ controls the spawn rate and velocity over the lifetime of the particles. Additionally, it controls the wind sound model by changing the resonance frequency and amplitude of the band-limited noise atoms (see section 3.3). To improve the simulation, the trees in the scene are also slightly moved with the same average wind speed.

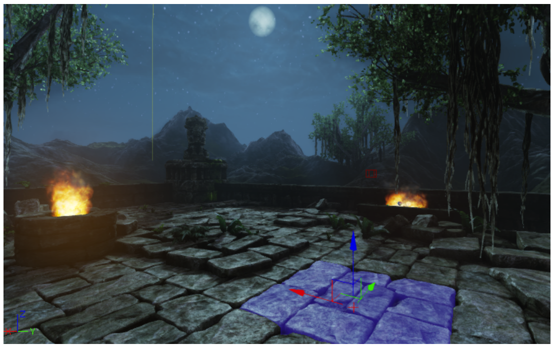

Figure 4: *Interactive scene with fire, wind and rain. Graphics are generated with UDK[2] and audio with custom external libraries implemented for Max[3]. Bidirectional communication between the audio and graphics engines is set with UDKOSC[4].*

5.3 Implementation

An interactive scene with rain, fire and wind was designed to illustrate the audio/graphics interactions (see figure 4). We used the UDK[2] game engine for graphics rendering, while sound rendering was performed with our custom synthesis/spatialization engine (see section 4) implemented as a Max[3] external library. Network communication between the graphics and sound engines was realized with the UDKOSC[4] library [43, 44], allowing the two engines to run on separate machines. High-level audio/graphics controls were edited by the designer and saved as automation parameters in the game engine. At runtime, these controls and the position of sound sources relative to the player are sent in real time (via UDKOSC functions) to the sound engine (see figure 5).

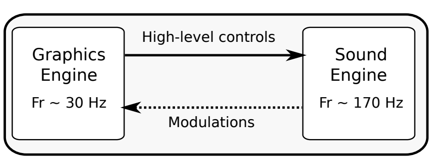

Figure 5: *Audio/graphics interactions. The graphics engine sends high-level designer controls (e.g., fire intensity and position) at the frame rate $F_r = 30$ Hz. For specific effects, the audio engine may also compute modulation parameters (e.g., flame energy) that are sent back to update the graphics.*

[2]www.UDK.com
[3]www.cycling74.com
[4]https://ccrma.stanford.edu/wiki/UDKOSC

5.4 Spatial sound distribution

We use two approaches for simulating spatial properties of sound sources in the virtual environment: localized volumetric sound sources, such as fire or wind in the trees are simulated as collections of point-like sources while completely diffuse sources like background wind and rain are created as collections of plane waves attached to the listener. These two strategies are illustrated in figure 6. Point-like sources and plane waves are both simulated with the technique described in section 4.

To compose a scene, the sound designer attaches localized sources to objects in the virtual environment. Several point-like sources can be attached together to form a volumetric source (e.g., a fire, wind in a tree). The location of each point-like source is continuously updated according to the listener position and orientation. The source-player distance d is simulated by a gain $\frac{1}{d}$ that attenuates the direct sound (not the reverberation). On the other hand, diffused sources (surrounding sounds) are simulated as a sum of eight plane waves, automatically distributed around the player. The plane wave analogy comes from the fact that surrounding sounds are virtually at an infinite distance from the listener and thus have no distance attenuation. The eight plane waves incident directions are evenly positioned on a horizontal circle, and their gain is weighted according to the desired source width as proposed in [28]. Surrounding sounds are attached to the player, i.e., they follow the player's position and orientation (all plane waves have a fixed orientation in the player's coordinate system). Consequently their position (orientation and distance) does not need to be updated in real time, saving some computation time compared to volumetric sources.

Several controls allow the sound designer to adjust the perceived width of volumetric and surrounding sounds (please see [28] for more details on the width gain computation). $RainWidth$ sets the angular distribution of rain components around the listener: individual raindrops are randomly distributed in the plane waves, while background noise is duplicated. $FireWidth$ sets the spatial distribution of crackling and combustion noise among their point-like sources. Finally $WindWidth$ sets the spreading of band-limited noises in the wind plane waves.

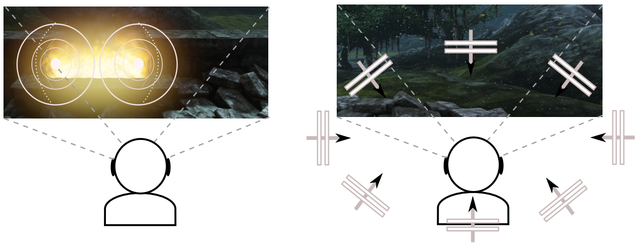

Figure 6: *Spatial distribution of sound sources in the virtual environment. Left: localized sounds (e.g., fire) are produced as a collection of point sources (spherical waves) represented as white circles. Right: diffuse sounds (e.g, rain and wind backgrounds) are simulated by plane waves with fixed incoming directions surrounding the player.*

6 Conclusion and future work

Atomic signal modeling of environmental sounds has been presented for the generation of audio in interactive applications. The proposed synthesizer is able to generate realistic sounds evoking a wide variety of phenomena existing in our natural environment. It provides an accurate tool to sonify complex scenes composed of several sound sources in 3D space. The position and the spatial extension of each source are dynamically controllable. High-level intuitive controls have been presented for simultaneous transformations of sound and graphics in interactive applications. Our approach results in a strong interaction between graphics and sound which increases the immersion in the scene. Demonstration videos are available online [22].

The main limitation of this study is the absence of a fully automatic analysis method to extract the synthesis parameters from recorded sounds. Currently the sound designer can modify the parameters of the sound models to approximate target sounds. Further research is needed to automatically decompose a given sound sample (provided by the sound designer) as a set of atomic elements. This decomposition would allow independent control of atom distribution in time and space, for high-level transformation of the original sound.

Automatic generation of soundscapes is another interesting research direction. For the moment, synthesis parameters (i.e., *RainIntensity*, *FireIntensity* and *WindSpeed*) are manually controlled by the designer, which allows him to produce the desired behavior. It would be interesting to provide a fully automatic weather system, where the correlation between each element could be specified by simple rules. As an example, the wind speed would influence fire intensity, rain intensity would be inversely proportional to fire intensity, etc. The synthesizer proposed in this paper is a good starting point for such research.

Acknowledgements

This work was partially funded by the EU IP project VERVE (www.verveconsortium.org). Part of this work has been done during the first author's Ph.D. [45] at Orange Labs and CNRS-LMA. We thank G. Pallone, M. Aramaki, R. Kronland-Martinet and C. Gondre for fruitful contributions to this work, and the anonymous reviewers for helpful comments.

References

[1] A. Farnell, *Designing sound*, MIT Press, 2010.

[2] J .K. Hahn, H. Fouad, L. Gritz, and J. W. Lee, "Integrating sounds and motions in virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 7, no. 1, pp. 67–77, 1998.

[3] C. Picard Limpens, *Expressive Sound Synthesis for Animations*, Ph.D. thesis, Université de Nice, Sophia Antipolis, 2009.

[4] D. Schwarz, "Corpus-based concatenative synthesis," *Signal Processing Magazine, IEEE*, vol. 24, no. 2, pp. 92–104, 2007.

[5] Y. Dobashi, T. Yamamoto, and T. Nishita, "Real-time rendering of aerodynamic sound using sound textures based on computational fluid dynamics," *ACM Transactions on Graphics (Proc. SIGGRAPH 2003)*, vol. 22, no. 3, pp. 732–740, 2003.

[6] J. N. Chadwick and D. L. James, "Animating fire with sound," *ACM Transactions on Graphics (Proc. SIGGRAPH 2011)*, vol. 30, no. 4, 2011.

[7] C. Stoelinga and A. Chaigne, "Time-domain modeling and simulation of rolling objects," *Acustica united with Acta Acustica*, vol. 93, no. 2, pp. 290–304, 2007.

[8] C. Zheng and D. L. James, "Harmonic fluids," *ACM Transaction on Graphics (Proc. SIGGRAPH 2009)*, vol. 28, no. 3, pp. 37:1–37:12, 2009.

[9] J. H. McDermott and E. P. Simoncelli, "Sound Texture Perception via Statistics of the Auditory Periphery: Evidence from Sound Synthesis," *Neuron*, vol. 71, no. 5, pp. 926–940, 2011.

[10] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, no. 4, pp. 809–836, 2006.

[11] W. T. Reeves, "Particle Systems - a Technique for Modeling a Class of Fuzzy Objects," *ACM Trans. Graph.*, vol. 2, pp. 91–108, April 1983.

[12] A. Kolb, L. Latta, and C. Rezk-Salama, "Hardware-based simulation and collision detection for large particle systems," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 123–131.

[13] C. Reynolds, "Big fast crowds on PS3," in *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, 2006, pp. 113–121.

[14] K. Sims, "Particle animation and rendering using data parallel computation," *Computer Graphics (Siggraph 90 proceedings)*, vol. 24, pp. 405–413, September 1990.

[15] M. Muller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[16] N. Bell, Y. Yu, and P. J. Mucha, "Particle-based simulation of granular materials," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2005.

[17] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2005.

[18] J. Lander, "The ocean spray in your face," *Game Developer Magazine*, July 1998.

[19] J. V. der Berg, "Building an advanced particle system," *Game Developer Magazine*, March 2000.

[20] C. Roads, "Introduction to granular synthesis," *Computer Music Journal*, vol. 12, no. 2, pp. 11–13, 1988.

[21] P. R. Cook, *Real Sound Synthesis for Interactive Applications*, A. K Peters Ltd., 2002.

[22] `http://www-sop.inria.fr/reves/Basilic/2012/VD12/`.

[23] W. W. Gaver, "What in the world do we hear? an ecological approach to auditory event perception," *Ecological Psychology*, vol. 5(1), pp. 1–29, 1993.

[24] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "Controlling a spatialized environmental sound synthesizer," in *Proceedings of the 2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2009.

[25] K. van den Doel, "Physically-based models for liquid sounds," in *Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display*, 2004.

[26] M. Aramaki, M. Besson, R. Kronland-Martinet, and S. Ystad, "Controlling the perceived material in an impact sound synthesizer," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 2, pp. 301–314, 2011.

[27] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "Analysis/synthesis and spatialization of noisy environmental sounds," in *Proceedings of ICAD 09 - 15th International Conference on Auditory Display*, 2009.

[28] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "A 3D Immersive Synthesizer for Environmental Sounds," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1550–1561, 2010.

[29] X. Rodet and P. Depalle, "Spectral envelopes and inverse fft synthesis," in *Proceedings of the 93rd AES Convention*, 1992.

[30] J.-M. Jot and A. Chaigne, "Digital delay networks for designing artificial reverberators," in *Proceedings of the 90th AES Convention*, 1991.

[31] V. Pulkki, "Virtual sound source positioning using vector base amplitude panning," *JAES*, vol. 45(6), pp. 456–466, 1997.

[32] D.G. Malham and A. Myatt, "3-d sound spatialization using ambisonic techniques," *Comp. Music Jour.*, vol. 19(4), pp. 58–70, 1995.

[33] J. Daniel, *Représentation de Champs Acoustiques, Application à la Transmission et à la Reproduction de Scènes Sonores Complexes dans*

un Contexte Multimédia, Ph.D. thesis, Université Paris 6, 2000.

[34] K. Garg and S. K. Nayar, "Photorealistic rendering of rain streaks," *ACM Transactions on Graphics (Proc. SIGGRAPH 2006)*, vol. 25, no. 3, pp. 996–1002, 2006.

[35] L. Wang, Z. Lin, T. Fang, X. Yang, X. Yu, and S. B. Kang, "Real-time rendering of realistic rain (msr-tr-2006-102)," Tech. Rep., Microsoft Research, 2006.

[36] N. Tatarchuk and J. Isidoro, "Artist-directable real-time rain rendering in city environments," in *Proceedings of the Eurographics Workshop on Natural Phenomena*, 2006.

[37] S. Tariq, "Rain," NVIDIA Whitepaper, 2007.

[38] A. Puig-Centelles, O. Ripolles, and M. Chover, "Creation and control of rain in virtual environments," *The Visual Computer*, vol. 25, pp. 1037–1052, 2009.

[39] O. E. Gundersen and L. Tangvald, "Level of Detail for Physically Based Fire ," in *Theory and Practice of Computer Graphics*, 2007, pp. 213–220.

[40] C. Horvath and W. Geiger, "Directable, high-resolution simulation of fire on the GPU," in *ACM SIGGRAPH 2009 papers*, 2009.

[41] W. Zhaohui, Z. Zhong, and W. Wei, "Realistic fire simulation: A survey," in *12th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, 2011.

[42] M. Yi and Q. Froemke, "Ticker Tape: A Scalable 3D Particle System with Wind and Air Resistance," 2010.

[43] R. Hamilton, "UDKOSC : An Immersive Musical Environment," in *Proceedings of the International Computer Music Conference*, 2011.

[44] R. Hamilton, J.-P. Caceres, C. Nanou, and C. Platz, "Multi-modal musical environments for mixed-reality performance," *Journal on Multimodal User Interfaces*, vol. 4, pp. 147–156, 2011.

[45] C. Verron, *Synthèse Immersive de Sons d'Environnement*, Ph.D. thesis, Université de Provence, 2010.