RASTA — Real-time Audio Spectral Toolbox for Analysis in Unity3D

Razvan Paisa[1], **Signe Toftgaard Henriksen**[1], and **Stefania Serafin**[1]

[1]**Multisensory Experiene Lab, Aalborg University**, Copenhagen, Denmark

ABSTRACT

This article describes the design and implementation of RASTA, a Real-time Audio Spectral Analysis Toolbox for Unity3D. Written in C#, RASTA provides an asynchronous, multi-threaded solution for extracting eight spectral and two time-domain audio descriptors in real time. Users have control over key parameters, including windowing type, FFT resolution, frequency bin exclusion threshold, and optional pre-filtering, enabling fine-tuned spectral analysis. Unlike Unity's built-in spectral analysis methods, RASTA allows pre-filtering, adaptive frequency range selection, and customizable descriptor calculations, ensuring greater flexibility and efficiency. Designed for applications in interactive media, reactive visuals, musician-AI interaction, speech analysis, and machine learning, it seamlessly integrates into Unity's C# ecosystem. By optimizing computational performance through multi-threading, RASTA enables low-latency spectral analysis without impacting frame rates, making it a powerful tool for developers working in game development, VR, digital art, and real-time sound-driven applications.

1. INTRODUCTION

In the world of interactive media and digital art, real-time audio-to-visual mapping is a fundamental technique used to create immersive and dynamic experiences. Whether in music-driven concert visuals, sound-reactive installations in museums, or real-time generative art, the ability to analyze and respond to audio in real-time is essential for interactive content creation.

Platforms such as Max with it's Jitter framework [1] and TouchDesigner [2] have long been the go-to tools for artists and developers exploring audiovisual synthesis. Max's Jitter provides a highly flexible environment for real-time video processing and manipulation based on the plethora of audio processing tools available in the Max/MSP (Max Signal Processing) toolbox, while TouchDesigner offers a node-based workflow that makes it easy to build complex audio-driven generative graphics. Both platforms have set a high standard for audio-reactive visuals, allowing creators to map frequency content, amplitude, and other spectral properties to animations and visual transformations.

With the growing adoption of virtual reality (VR) in training and rehabilitation, real-time audio analysis is becoming an essential component in VR-based interactive systems. Fields such as medical training, speech therapy, cognitive rehabilitation, and auditory feedback for physical therapy rely on precise, real-time analysis of sound features to enhance user engagement and provide adaptive learning experiences, but there is a lack of standardization in equipment and tools [1], which extends to the audio processing pipeline. For example, VR speech therapy applications can benefit from spectral analysis to assess pronunciation, speech intelligibility, or vocal effort in real time. Voice activity detection (VAD) and speaker identification are crucial for VR-based language learning environments and multi-user VR training simulations. Similarly, in physical rehabilitation, audio-reactive feedback systems can be used to encourage movement by dynamically adjusting soundscapes or visual elements based on the user's speech or movement patterns.

However, despite its superiority in interactive 3D creation and rendering, Unity3D lacks built-in, high-level spectral analysis tools, thus 3rd party audio engines (Wwize [3], FMOD [4], etc) are frequently used when advanced signal manipulation is required. Unlike TouchDesigner or Max/MSP, Unity's audio processing pipeline is primarily designed for playback and spatialization, rather than real-time analysis and reactivity. As a result, developers working on interactive, audio-driven applications within Unity must either rely on external libraries or implement complex custom solutions to extract spectral features.

This paper addresses this gap by introducing a real-time spectral analysis toolbox for Unity3D, streamlining the process of extracting and utilizing spectral descriptors for interactive applications. The toolbox provides a set of commonly used audio features that can be leveraged for dynamic audio visualization, procedural audio visualization, sonic interactive environments, and machine learning applications in Unity, that integrates seamlessly into the C# programming paradigm of Unity. The repository hosting the latest version of the toolbox can be accessed on GitHub [5].

[1] https://cycling74.com
[2] https://derivative.ca

[3] https://www.audiokinetic.com
[4] https://www.fmod.com
[5] https://github.com/razvysme/Unity3D_Spectral_Analysis

2. AUDIO DESCRIPTORS

In digital audio signal processing, spectral descriptors are commonly used to analyze and characterize the frequency content of a signal. These features play a crucial role in music information retrieval (MIR), speech processing, audio classification, real-time reactive systems, and interactive media [2]. By capturing key frequency characteristics, spectral descriptors enable the automatic extraction of meaningful audio features, making them fundamental in various signal analysis and machine learning applications.

Spectral descriptors have been applied in a wide range of fields, including speaker identification and recognition, where they help distinguish between different speakers based on voice characteristics [3]. In acoustic scene recognition, a suite of descriptors can be used to classify environments such as indoor spaces, urban streets, or natural landscapes based on their spectral profile [4, 5]. In instrument recognition tasks, spectral analysis can be used to assist in the automatic identification of musical instruments in mono or polyphonic scenarios [6]. Besides instrument recognition, musical gender discrimination is also a common use case, enabling algorithms to categorize music into predefined genres based on spectral similarities [7, 8]. Spectral descriptor are not only used for music, but also for human voice detection and classification or mood recognition, where spectral features contribute to estimating the emotional content of music or speech [9, 10].

Beyond their use in traditional audio analysis, spectral descriptors are widely employed in machine learning and deep learning frameworks to facilitate automatic sound recognition and perceptual analysis. Their ability to model frequency distribution and time-varying spectral properties makes them essential for both supervised and unsupervised learning approaches in audio-related tasks. In recent years, neural networks, convolutional architectures, and recurrent models have leveraged spectral descriptors to improve classification, recognition, and segmentation of audio signals.

The following sections provide an overview of the spectral descriptors implemented in this toolbox. For most of the equations the following definitions apply:

f_k is the frequency in Hz corresponding to bin k.

s_k is the spectral value at bin k.

b_1 and b_2 are the band edges (in bins) over which to calculate the spectral centroid.

2.1 Spectral Centroid

The spectral centroid is a fundamental feature in audio signal analysis, representing the "center of mass" of a sound's spectrum [2]. It is calculated as the weighted mean of the frequencies present in the signal, where the magnitude of each frequency bin acts as its corresponding weight as seen in equation 1.

$$\text{centroid } \mu_1 = \frac{\sum_{k=b_1}^{b_2} f_k s_k}{\sum_{k=b_1}^{b_2} s_k} \tag{1}$$

A higher spectral centroid indicates that the energy of the sound is concentrated at higher frequencies, often associated with brighter, sharper sounds. Sounds rich in high-frequency components, such as a cymbal crash or an electric guitar with distortion, have a high spectral centroid. while, ones with predominantly low frequencies, such as a bass guitar or a deep male voice, have a lower spectral centroid [2]. Because it captures a perceptual quality of sound — it's brightness [11], the spectral centroid is widely used in timbre analysis, helping distinguish between different musical instruments, speech tones, and audio textures, as well as evolution of musical phrases or genres [12]. In speech analysis, the spectral centroid helps differentiate between voiced and unvoiced speech segments. Unvoiced sounds, such as consonants like "s" or "f," typically have higher spectral centroids due to their higher frequency content, whereas voiced sounds like vowels have lower centroids [13, 14].

2.2 Spectral Spread

Another fundamental descriptor that is frequently used along the spectral centroid is the analysis of spectral spread. It represents the *instantaneous bandwidth* of the spectrum, measuring the dispersion of spectral energy around the spectral centroid [2]. It provides an indication of how concentrated or dispersed the frequency content of a sound is, offering insights into its timbre and tonal characteristics. Mathematically, it is defined as the standard deviation of the frequency spectrum, weighted by its magnitude, as shown in equation 2.

$$\text{spread } \mu_2 = \sqrt{\frac{\sum_{k=b_1}^{b_2} (f_k - \mu_1)^2 s_k}{\sum_{k=b_1}^{b_2} s_k}} \tag{2}$$

μ_1 is the spectral centroid as described by equation 1

A low spectral spread indicates that most of the signal's energy is clustered around a narrow frequency range, as seen in tonal sounds such as a flute note or a pure sine wave, which have concentrated spectral energy near the fundamental frequency. In contrast, a high spectral spread suggests that the energy is widely distributed across the frequency range, typical of percussive sounds like white noise, cymbals, or unvoiced speech sounds [2].

From a perceptual standpoint, spectral spread helps differentiate between sustained, harmonic sounds and noisy, percussive sounds [11]. Musical instruments with stable tonal content, such as violins or flutes, tend to have lower spectral spread, whereas noisy or transient sounds, such as snare drums or cymbals, exhibit higher spread due to their broadband energy distribution [15].

2.3 Spectral Flatness

The spectral flatness measures the peakiness of the spectrum, indicating whether a signal is tonal or noise-like [2]. It is defined as the ratio of the geometric mean to the arithmetic mean of the spectrum's magnitudes, as seen in equation 3.

$$\text{flatness} = \frac{\left(\prod_{k=b_1}^{b_2} s_k\right)^{\frac{1}{b_2-b_1}}}{\frac{1}{b_2-b_1}\sum_{k=b_1}^{b_2} s_k} \tag{3}$$

A higher spectral flatness suggests that the energy is evenly distributed, characteristic of noise-like or percussive sounds (e.g., white noise, snare drums). In contrast, a lower spectral flatness indicates tonal sounds with dominant frequency peaks, such as a violin note or sustained speech.

One of the major applications of spectral flatness is in singing voice detection, where it helps to distinguish sung melodies from spoken words [16]. In speech and music classification, a higher spectral flatness often indicates unvoiced speech or percussive sounds, while a lower value suggests tonal speech segments or sustained instrumental notes. Additionally, spectral flatness has been successfully applied in audio scene recognition, helping to classify environmental sounds such as urban noise, nature sounds, or indoor environments [4].

2.4 Spectral Tilt

The spectral tilt, also known as spectral skewness, measures the asymmetry of the frequency distribution around the spectral centroid [2]. It describes whether the higher or lower frequencies dominate the spectral shape, providing insight into harmonic balance and timbral characteristics. Spectral tilt is computed as the third-order spectral moment [17], as shown in equation 4.

$$\text{tilt} = \frac{\sum_{k=b_1}^{b_2} (f_k - \mu_1)^3 s_k}{(\mu_2)^3 \sum_{k=b_1}^{b_2} s_k} \tag{4}$$

μ_1 is the spectral centroid as calculated in equation 1
μ_2 is the spectral spread as calculated in equation 2

In phonetics, spectral tilt is used alongside other spectral moments to distinguish the place of articulation in speech sounds [18]. It helps analyze how energy is distributed across harmonics, with positive skew indicating dominance of lower frequencies and negative skew indicating stronger high frequencies. For example, in a four-tone signal, a positive tilt occurs when the lower tone is more prominent, while a negative tilt suggests dominance of higher harmonics.

2.5 Spectral Slope

The spectral slope measures how energy decreases with frequency, providing insight into the balance between low and high frequencies in a signal [2], and is calculated as the amount of decrease of the spectrum [19], as seen in equation 5.

$$\text{slope} = \frac{\sum_{k=b_1}^{b_2} (f_k - \mu_f)(s_k - \mu_s)}{\sum_{k=b_1}^{b_2} (f_k - \mu_f)^2} \tag{5}$$

μ_f is the mean frequency
μ_s is the mean spectral value

Spectral slope is directly related to the resonant characteristics of the vocal folds and is often used in speech analysis, speaker identification, and modeling speaker stress [3, 9]. A steeper spectral slope occurs when lower frequencies dominate, while a flatter slope indicates greater energy in higher frequencies. It is particularly important in phonetics and speech processing, where it has been used to analyze vocal effort and emotional expressiveness. Studies have shown that spectral slope discrimination occurs in early childhood development, highlighting its social and perceptual significance in human communication [10]. It is most pronounced when the energy in the lower formants is much greater than in the higher formants, which is characteristic of strongly resonant vocal sounds.

Beyond speech, spectral slope plays a key role in music timbre classification, distinguishing mellow, warm sounds from brighter, sharper ones, and it's usually used in combination with other descriptors like centroid and spread.

2.6 Spectral Crest

The spectral crest measures the peakiness of the spectrum, providing insight into the tonal versus noise-like characteristics of a signal [2]. It is defined as the ratio between the maximum spectral magnitude and the average spectral magnitude, as shown in equation 6.

$$\text{crest} = \frac{\max(s_{k\in[b_1,b_2]})}{\frac{1}{b_2-b_1}\sum_{k=b_1}^{b_2} s_k} \tag{6}$$

A higher spectral crest indicates that the spectrum contains prominent peaks, meaning the signal is more tonal. This is typical for harmonic sounds, such as a sustained violin note or a pure sine wave. In contrast, a lower spectral crest suggests that the spectral energy is more evenly distributed, characteristic of noisy or percussive sounds, such as white noise or a snare drum hit. While both spectral flatness and the crest measure aspects of peakiness of the signal, the former measures how evenly the spectral energy is distributed (or how noise-like it is) while the latter measures how much a spectrum is dominated by tonal peaks. For example a flute sound has high crest but low flatness, while the sound of rain has high flatness, but low crest.

Spectral crest is widely used in speech, music, and environmental sound classification [20,21]. It helps distinguish between tonal speech sounds and fricatives and is valuable in instrument classification, where tonal instruments like flutes exhibit high crest values, while drums and noise-based sounds have lower values.

2.7 Spectral Entropy

Another measure of the peakiness of the spectrum is spectral entropy. It measures the uniformity or randomness of the spectral distribution and it quantifies how spread out or concentrated the spectral energy is, treating the frequency spectrum like a probability distribution [2]. Spectral entropy is calculated as shown in equation 7.

$$\text{entropy} = \frac{-\sum_{k=b_1}^{b_2} s_k \log(s_k)}{\log(b_2 - b_1)} \tag{7}$$

Since entropy is a measure of disorder, spectral entropy is particularly useful in speech processing. In automatic speech recognition, it has been successfully applied to voiced/unvoiced decisions, where voiced speech regions have lower entropy due to their structured harmonic content, while unvoiced speech regions exhibit higher entropy due to their noise-like characteristics [22, 23]. For example, a vowel sound like "a" has low spectral entropy, while an unvoiced fricative like "s" has high spectral entropy due to its broadband energy distribution.

Compared to spectral crest, which captures the contrast between peak and average energy, and spectral flatness, which compares geometric and arithmetic means, spectral entropy directly quantifies spectral randomness. While white noise exhibits high entropy due to its even energy distribution, a harmonic musical note has low entropy, as most of its energy is concentrated in a few frequency bins.

2.8 Spectral Flux

Spectral flux measures the rate of change in spectral energy over time, capturing how much the frequency content of a signal fluctuates between consecutive frames [2]. Unlike spectral crest, flatness, or entropy, which focus on the static characteristics of a spectrum, spectral flux is a temporal feature, analyzing how the spectrum evolves over time. It is computed by comparing the magnitude spectrum of consecutive frames, as shown in equation 8.

$$\text{flux}(t) = \left(\sum_{k=b_1}^{b_2} |s_k(t) - s_k(t-1)|^p \right)^{\frac{1}{p}} \tag{8}$$

Because it reflects changes in spectral content, spectral flux is widely used in onset detection and audio segmentation. In music analysis, it plays a crucial role in identifying note attacks and rhythmic structures, making it particularly effective for detecting beats and percussive events [24]. For example, in a rythmical music piece, spectral flux exhibits sharp peaks at beat locations, where sudden spectral changes occur due to the transient nature of drum hits.

In speech processing, spectral flux can help differentiate between steady-state phonemes and transient sounds, aiding in speech segmentation and phoneme classification. Additionally, it is used in environmental sound analysis, helping distinguish between stationary sounds (e.g., sustained background noise) and dynamic events (e.g., footsteps, door slams) [25].

2.9 RMS Energy

Unlike the spectral descriptors discussed so far, Root Mean Square (RMS) energy is not a spectral analysis measure per se, but rather a time-domain feature that quantifies the overall loudness or intensity of an audio signal over an given time window. It is computed as the square root of the average squared amplitude values of a signal over a given time frame, as shown in equation 9.

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} x[n]^2} \tag{9}$$

N is the size of the analysis window
$x[n]$ is the value of the n-th sample

Since RMS energy does not analyze frequency content, it differs fundamentally from spectral features such as spectral centroid, spread, or flux, which characterize the distribution and evolution of frequencies in a signal. Instead, RMS energy measures the power of a signal over time, making it useful for loudness estimation, dynamic range analysis, and envelope tracking [2].

In music and speech processing, RMS energy helps distinguish between loud and quiet segments. In speech, for example, RMS energy is higher during voiced speech (e.g., vowels) and lower during unvoiced consonants (e.g., "s" or "f"). In beat detection and onset tracking, RMS energy is often combined with spectral flux to improve detection accuracy, particularly for percussive sounds, where sudden increases in energy correspond to rhythmic events. While it does not provide spectral detail, it is usually used in a complementary fashion with other spectral descriptors.

2.10 Zero Crossing Rate

Zero Crossing Rate (ZCR) measures the rate at which a signal changes sign, indicating how frequently the waveform crosses the zero amplitude axis, as seen in equation 10.

$$\text{ZCR (Hz)} = \frac{f_s}{2N} \sum_{n=1}^{N-1} \begin{cases} 1, & \text{if } x[n-1] \times x[n] < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

f_s is the sampling frequency

It is primarily a time-domain feature, rather than a spectral one, and is commonly used as a simple indicator of signal frequency content [2]. While ZCR can differentiate between tonal and noisy sounds—with higher ZCR values for noise-like signals (e.g., unvoiced speech, cymbals) and lower values for harmonic content (e.g., vowels, sustained notes)—it is often considered redundant to spectral centroid in real-world applications, and was mostly used by the authors as a validation of spectral centroid and the performance of the spectral descriptors.

3. IMPLEMENTATION AND USAGE

The toolbox is implemented as a single C# script that has to be associated as a component with an Unity *Game Object* containing an *Audio Source*. This way, multiple instances of the analysis system can be managed dynamically, by addressing the parent game object.

The core system follows these steps:

1. Audio Signal Acquisition – The Unity *Audio Source* feeds real-time audio data into the processing pipeline.
2. Filtering – An optional State Variable Filter with low-pass, band-pass and high-pass filters focus the signal before spectral analysis.
3. Fast Fourier Transform Computation – The signal is transformed into the frequency domain using the *MathNet.Numerics*[6] FFT library.
4. Spectral Feature Extraction – Spectral descriptors (e.g., centroid, spread, crest, entropy, etc.) are computed from the FFT spectrum.
5. Data Display – Extracted features are displayed on canvas, exemplifying the complete use case.

3.1 Audio Signal Acquisition

The system operates by continuously analysing audio input from the *Audio Source* component, by calling the function *OnAudioFilterRead(float[], int)*. This function practically exposes the content of the output buffer for the given *Audio Source*, returning both the number of channels and the audio data, stored as a one-dimensional interleaved array of size: *nr. of channels * length of the audio buffer* (e.g., in the case of stereo audio with a buffer length of 1024 samples, the resulting array size will be 2 * 1024 = 2048 samples). It is important to note that this function operates independently of Unity's graphics rendering frame rate (commonly dictated by the *Update()* function), as it is tied to the audio DSP clock, thus it runs recursively at a rate of $f_s/bufferSize$, unaffected by the graphical performance.

3.2 Filtering - Optional

The audio data is processed through a digital state variable filter (SVF) [26], enabling the user to isolate a specific frequency range for analysis. This approach serves two main purposes: 1. it allows the user to focus the analysis on a targeted spectral region, improving the relevance of extracted features and 2. it enables auditioning of the analysed signal. The SVF filter can be bypassed if the entire spectrum is desired. While restricting the analysis to a narrower spectral bandwidth after Fourier transformation could theoretically result in faster processing, it would compromise the ability to audibly assess the analyzed signal, making this filtering approach a more flexible solution, at least in the prototyping stages.

A circular buffer of user defined size (limited to integer multiple of the frame size) is used to store incoming samples outputted from the filter, allowing for a flexible analysis windows that exceeds the audio frame size. Larger analysis windows increase latency but result in better spectral accuracy, especially at lower frequencies, and vice-versa.

3.3 Fourier Transformation

The Fast Fourier Transform (FFT) from the Math-Net.Numerics library [27] is used to convert time-domain audio signals into their frequency-domain representations, enabling spectral feature extraction. The library provides several real and complex FFT implementations, including options optimized for real time applications. Since the FFT assumes that input signals are periodic and continuous, applying a window function is crucial to minimize spectral leakage. The toolbox supports multiple window types (rectangular, triangular, Hamming, Hanning, Blackman, Blackman-Harris), each affecting the resolution, smoothing, and spectral energy distribution [27]. The choice of window influences the normalization process, as each window type has a coherent gain factor that must be applied to correctly scale the spectral magnitudes, as described by the authors of [28].

For this project the function *void ForwardReal(Single[] data, int n, FourierOptions options)* was used, as the real-valued time-domain audio samples are transformed using a *Packed Real-Complex* FFT, which optimizes memory usage by leveraging the conjugate symmetry property of real-valued signals [27]. Since the complex spectrum of a real signal is conjugate-even, the full spectral information can be reconstructed using only the positive frequency components (i.e., the first half of the spectrum). . While Unity's built-in function *AudioSource.GetSpectrumData()* provides a simple way to obtain spectral magnitudes, it was not chosen for this implementation due to the following limitations: 1. does not allow for pre-filtering or custom modifications to the audio buffer before FFT computation; 2. it uses a fixed, predefined Unity window functions limiting the flexibility of the system.

After the FFT is computed, the bin magnitudes (1 - N, excluding the DC bin) are normalized to maintain consistent feature scaling across different input levels, with gain coefficient respective of the gain output of selected window type [28].

3.4 Descriptor Calculations

With the FFT data available, the extraction of spectral descriptors can be implemented, as described in section 2. The anatomy of all calculation function can be exemplified by the spectal centroid one: *float CalculateSpectralCentroid(float[] spectrum, float[] freqs, int b1, int b2)* that requires as input the magnitude spectrum obtained from

[6] `https://numerics.mathdotnet.com/`

the FFT, the corresponding frequency values for each bin (which has been pre-computed according to the analysis windows size), and user-defined frequency bounds, which allow the calculation to be restricted to a specific frequency range.

While the SVF filter allows for pre-filtering in the time domain, restricting the analysis range directly in the frequency domain (via b1 and b2) offers additional flexibility and redundancy. Even if the SVF is bypassed, the function can ignore undesired frequency bins, reducing computation on irrelevant spectral regions; if the SVF has already limited the signal, frequency-domain bandwidth constrains further refine the analysis. Furthermore, an adaptive band selection can be adjusted in real time without altering the raw input signal, offering a more efficient way to track specific frequency regions in the case of dynamic applications.

3.5 Asynchronous implementation

To ensure real-time performance without disrupting Unity's rendering and physics loops, the entire spectral analysis system is designed to execute asynchronously, meaning that audio processing occurs in a separate thread, independent of graphics frame update cycle. Instead of performing blocking operations that could introduce latency, the system dynamically checks whether the previous computation task has completed before launching a new analysis task. This ensures that audio frames are processed continuously and without delays, even if some frames take longer to analyze than a single graphical frame duration. By avoiding blocking operations on the main thread, the system minimizes frame drops, UI lag, and unwanted performance spikes, which are common when performing intensive real-time DSP operations within the *Update()* loop. Since more and more computers have several cores, this multi-threaded approach considerably improves performance. Furthermore, additional computations, such as adaptive filters or machine learning-based feature extraction, can be added without significantly affecting real-time performance. Now, while asynchronous execution greatly improves performance, it also introduces potential challenges, especially related to task synchronization, due to the fact that there is potential for delay and jitter in the rate of updating the spectral descriptors.

3.6 Usage

The RASTA toolbox is designed to be intuitive and easily configurable, allowing users to dynamically adjust analysis parameters through publicly exposed variables. To use the toolbox, developers must first attach the provided C# script to a *Game Object* containing an *Audio Source*. This enables real-time spectral analysis for any audio input source, including microphone input, in-game sound, or pre-recorded audio clips.

Before using the system, the MathNet.Numerics library must be installed via *NuGet* to enable FFT computation. This can be done by opening the NuGet Package Manager in Unity's C# project, searching for MathNet.Numerics, and installing the latest version. For using NuGet package manager in Unity, we recommend the Patrick McCarthy's *NuGetForUnity* [7]

Once the system is active, users can modify public parameters in Unity's *Inspector Panel* to customize the analysis process. Key adjustable parameters include the window type, analysis window length, frequency bin amplitude threshold, and the SVF frequency and operation mode (including bypass). In the included example, all the descriptors from section 2 and displayed on using Unity's TextMeshPro data system, but users can exclude unwanted descriptors from the analysis, by ommiting calculations from the *CalculateSpectralAnalysisAsync* function. An optional moving average filter class is present in the toolbox to reduce fluctuations in the analysis parameters.

4. APPLICATIONS & USE-CASES

Spectral analysis in Unity3D opens up a wide range of applications, ranging from interactive art and multimedia performances to speech training, therapy and AI-driven sound modeling. In interactive art and live performances, the toolbox allows audio-driven visuals, animations, and projection mapping, where spectral features control movement, color, or other structural elements in real time. Furthermore, a novel discipline in the human-computer co-creation environment — AI (artificial intelligent systems) and musician interaction, can benefit greatly from spectral descriptors as discussed in [29] or presented in the *SH4DOW* experimental theatre performance [8] .

Beyond artistic use, the toolbox is valuable for auditory scene analysis, helping classify environmental sounds, speech activity, and acoustic environments. In speech training and rehabilitation, spectral analysis can be used to provides real-time articulation, assisting in voice therapy and language learning. Additionally, it enables lip-syncing for virtual avatars, where speech-driven mouth animation enhances VR interactions and game character expressiveness.

A key future application is in AI model training, where real-time spectral analysis can be used to develop personalized, adaptive audio systems. By continuously analyzing input, AI can refine speech recognition, music generation, and sound classification, leading to user-specific models tailored to different voices, environments, or musical styles. The RASTA's flexibility, user friendliness and real-time capabilities make it a powerful tool for both creative and functional sound-driven applications.

5. CONCLUSION

The Real-Time Audio Spectral Analysis Toolbox for Unity3D (RASTA) bridges the gap between real-time audio processing and interactive media applications, offering an efficient and flexible solution for spectral analysis within Unity. By integrating custom spectral descriptor extraction, configurable analysis parameters, and real-time visualization, the toolbox enables both creative and functional applications, from reactive art and musician-AI in-

[7] `https://github.com/GlitchEnzo/NuGetForUnity`
[8] `https://cec.dk/contributions/sh4dow/`

teraction to speech training, environmental sound analysis, and adaptive AI models.

Unlike Unity's built-in AudioSource.GetSpectrumData(), RASTA uses the MathNet library to provide greater control over signal processing, offering pre-filtering capabilities, adaptive frequency range selection, and optimized FFT computation. The asynchronous implementation ensures that spectral analysis runs in parallel with Unity's core rendering engine, maintaining low-latency performance without affecting frame rates.

With its configurable analysis parameters, RASTA allows users to fine-tune windowing functions, FFT resolutions, and feature extraction settings, making it adaptable to varied use cases. Whether used for audio-driven visuals, interactive VR environments, real-time machine learning, or computational music applications, the toolbox provides a versatile foundation for integrating real-time audio analysis into game engines and creative coding workflows.

Looking ahead, the toolbox can serve as a starting point for training AI models that rely on localized, real-time sound features, enabling intelligent sound classification, voice activity detection, and adaptive music generation. With further development, it could extend to support deep learning frameworks, enhancing Unity's capabilities in AI-driven audio processing.

By making advanced spectral analysis accessible within Unity's C# ecosystem, RASTA affords new possibilities for interactive sound design, music visualization, and immersive multimedia experiences, positioning it as a valuable tool for developers, artists, and researchers working at the intersection of audio analysis and interactive technology.

6. REFERENCES

[1] P. P. Tuominen and L. A. Saarni, "The use of virtual technologies with music in rehabilitation: a scoping systematic review," *Frontiers in Virtual Reality*, vol. 5, p. 1290396, 2024.

[2] G. Peeters *et al.*, "A large set of audio features for sound description (similarity and classification) in the cuidado project," *CUIDADO Ist Project Report*, vol. 54, no. 0, pp. 1–25, 2004.

[3] H. Murthy *et al.*, "Robust text-independent speaker identification over telephone channels," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 5, pp. 554–568, 1999.

[4] Y. Petetin, C. Laroche, and A. Mayoue, "Deep neural networks for audio scene recognition," in *23rd European Signal Processing Conference (EUSIPCO)*, 2015.

[5] A. Eronen *et al.*, "Audio-based context recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 1, pp. 321–329, 2006.

[6] S. Essid, G. Richard, and B. David, "Instrument recognition in polyphonic music based on automatic taxonomies," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 1, pp. 68–80, 2006.

[7] T. Li and M. Ogihara, "Music genre classification with taxonomy," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005.

[8] J.-M. Ren, M.-J. Wu, and J.-S. R. Jang, "Automatic music mood classification based on timbre and modulation features," *IEEE Transactions on Affective Computing*, vol. 6, no. 3, pp. 236–246, 2015.

[9] J. H. L. Hansen and S. Patil, "Speech under stress: Analysis, modeling and recognition," *Lecture Notes in Computer Science*, vol. 4343, pp. 108–137, 2007.

[10] C. D. Tsang and L. J. Trainor, "Spectral slope discrimination in infancy: Sensitivity to socially important timbres," *Infant Behavior and Development*, vol. 25, no. 2, pp. 183–194, 2002.

[11] J. M. Grey and J. W. Gordon, "Perceptual effects of spectral modifications on musical timbres," *The Journal of the Acoustical Society of America*, vol. 63, no. 5, pp. 1493–1500, 1978.

[12] E. Schubert, J. Wolfe, A. Tarnopolsky *et al.*, "Spectral centroid and timbre in complex, multiple instrumental textures," in *Proceedings of the international conference on music perception and cognition, North Western University, Illinois*. sn, 2004, pp. 112–116.

[13] E. Scheirer and M. Slaney, "Construction and evaluation of a robust multifeature speech/music discriminator," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1997, pp. 1331–1334 vol.2.

[14] P. N. Le, E. Ambikairajah, J. Epps, V. Sethu, and E. H. Choi, "Investigation of spectral centroid features for cognitive load classification," *Speech Communication*, vol. 53, no. 4, pp. 540–551, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639311000069

[15] B. Kostek, "Musical instrument classification and duet analysis employing music information retrieval techniques," *Proceedings of the IEEE*, vol. 92, no. 4, pp. 712–729, 2004.

[16] B. Lehner *et al.*, "On the reduction of false positives in singing voice detection," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[17] K. L. Davidson and P. J. Loughlin, "Instantaneous spectral moments," *Journal of the Franklin Institute*, vol. 337, no. 4, pp. 421–436, 2000.

[18] A. Jongman *et al.*, "Acoustic characteristics of english fricatives," *The Journal of the Acoustical Society of America*, vol. 108, no. 3, pp. 1252–1263, 2000.

[19] A. Lerch, *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*. Piscataway, NJ: IEEE Press, 2012.

[20] M. K. Nandwana, H. Bořil, and J. H. Hansen, “A new front-end for classification of non-speech sounds: a study on human whistle,” *International Speech and Communication Association*, 2015.

[21] P. Herrera-Boyer, G. Peeters, and S. Dubnov, “Automatic classification of musical instrument sounds,” *Journal of New Music Research*, vol. 32, no. 1, pp. 3–21, 2003.

[22] H. Misra, S. Ikbal, H. Bourlard, and H. Hermansky, “Spectral entropy based feature for robust asr,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004.

[23] A. M. Toh, R. Togneri, and S. Nordholm, “Spectral entropy as speech features for speech recognition,” *Proceedings of PEECS*, vol. 1, p. 92, 2005.

[24] S. Dixon, “Onset detection revisited,” in *International Conference on Digital Audio Effects*, vol. 120, 2006, pp. 133–137.

[25] G. Tzanetakis and P. Cook, “Multifeature audio segmentation for browsing and annotation,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 1999.

[26] J. O. Smith III, “Digital state-variable filters,” *Center for Computer Research in Music and Acoustics*, 2024.

[27] M. Project, “Mathnet.numerics - open-source numerical library for .net,” 2025, available at: https://github.com/mathnet/mathnet-numerics. [Online]. Available: https://www.mathdotnet.com

[28] F. J. Harris, “On the use of windows for harmonic analysis with the discrete fourier transform,” *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1987.

[29] E. T. Kaspersen, D. Górny, C. Erkut, and G. Palamas, “Generative choreographies: The performance dramaturgy of the machine.” in *VISIGRAPP (1: GRAPP)*, 2020, pp. 319–326.