ATS: a Lisp Environment for Spectral Modeling

Juan Pamphlín

Center for Computer Research in Music and Acoustics (CCRMA), Stanford University
juan@CCRMA.stanford.edu, http://www-CCRMA.stanford.edu/~juan

Abstract

ATS is a library of Lisp functions for spectral Analysis, Transformation, and Synthesis of sound. It provides a variety of tools for spectral modeling including different analysis front-ends, complex transformation algorithms, and several synthesis techniques. This paper presents a snapshot of ATS' current state, as well as an overview of recent research work related with this on-going project.

1 Introduction

ATS is a library of Lisp functions for spectral Analysis, Transformation, and Synthesis of sound. The Analysis section of ATS implements two complementary partial tracking algorithms. This allows the user to decide which strategy is the best suited for a particular sound to be analyzed. Analysis data is stored as a Lisp abstraction called sound. A sound in ATS is a symbolic object representing a spectral model that can be sculpted using a wide variety of transformation functions. ATS sounds can be synthesized using different target algorithms, including additive, subtractive, granular, and hybrid synthesis techniques.

2 Symbolic Processing

ATS is written in Common Lisp [1]. Using Lisp's listener the user can interact with the system in many ways, performing analysis of sounds, visualizing and transforming spectral data, and running synthesis in real time. The use of a high level language like Common Lisp presents the advantage of a symbolic representation of spectral qualities [2]. For instance, high level traits of a sound, such as global spectral envelopes, frequency centroids, formants, vibrato patterns, etc., can be treated as symbolic objects and used to create abstract sound structures or spectral classes. In a higher layer of abstraction, the concept of spectral class is used to implement predicates and procedures, conforming spectral logic operators. For instance, in terms of this logic, sound morphing becomes a union (a dynamic exchange of features) of spectral classes that generates a particular hybrid sound instance. Spectral information is stored in a data abstraction called sound. Sounds can be treated symbolically like any other Lisp object. This means that the user can map functions or closures through lists of sounds or include sound objects in algorithmic composition procedures in a transparent way, spectral and musical data being interchangeable.

3 Analysis

Sound analysis in ATS is based on the sinusoidal model [3]. In its present stage, ATS has two analysis front-ends, one using a pitch synchronous algorithm (sieve), and a more complex one performing partial tracking and extraction (tracker). Both functions have their foundation on the Short-Time Fourier Transform (STFT) and perform sinusoidal parameter detection (i.e. frequency, amplitude, and phase). The output of these analysis functions is an ATS sound structure.

3.1 Sieve

This algorithm performs harmonic tracking of partials. The parameters for the STFT are computed as a function of the fundamental of the analyzed sound. This function is useful for stable isolated harmonic tones, it places detected peaks
into a frequency sieve that is used to create sinusoidal trajectories. This algorithm is based on a pitch synchronous strategy where each track of the sieve has a controllable frequency bandwidth dependent on the fundamental.

3.2 Tracker

Tracker implements a more robust analysis algorithm than sieve that is suitable for the analysis of non harmonic tones. After performing peak detection and interpolation, peaks are continued across frames, and sinusoidal trajectories are extracted [4]. Tracker uses also psychoacoustic information to determine the salience of each detected trajectory. This information is based on the masking effects produced within critical bands and accounts for the audibility of the sinusoidal trajectories. Salience information can be used to reduce the number of partials using a psychoacoustic metric. In a post-processing stage to analysis, the user can decide to keep just a certain number of trajectories, filtering out the less salient ones. Salience can be also used to decide which partials to synthesize in a real-time scenario where synthesis resources are limited [5].

4 Transformations

A sound in ATS is an intermediate representation of the spectral evolution of an analyzed signal [6]. The user can manipulate the parameters of a sound to operate spectral transformations on it. Transformations can be destructive (i.e. the original sound structure is changed), or generative (i.e. the transformation generates a new instance of the sound, keeping the original sound untouched). An ATS sound can cumulate several transformations before being resynthesized.

Parameters passed to the transformation functions can be, of any of the following types:

1. A number: transformation is parallel and synchronous, this numeric value is used to transform all partials over all the time frames of the sound.

2. A list of numbers: each partial is transformed with a different value.


|let* ([partials (ats-sound-partial my-sound)])
| (transp-env
| (loop for i from 0 below partials
| with even-env = '[(0.1.0 1.3.0)]
| with odd-env = '[(0.1.0 1.3.0)]
| collect
| (if (oddp i) odd-env even-env)])
| (trans-sound 'my-sound transp-env
| formants T
| name 'my-new-sound
| :simp T?)

Figure 1: Example Lisp code using trans-sound. Even and odd partials are transposed using different envelopes. The loop macro is creating a list of envelopes transp-env that is used in the call to the function.

4. A list of envelopes: each partial is transformed with a different dynamic value.

5. A list of numbers and envelopes: some partials are transformed with constant values and others with dynamic ones.

Transformation functions operate on frequency, amplitude, and time parameters. ATS has at present about 20 built-in transformation functions that can be combined using macros to design more complex algorithms. In the following sections three of the transformation functions are presented as an example.

4.1 Transposition

trans-sound sound transposition &key formants name simp

The function trans-sound performs frequency transposition. This function takes the following arguments:

- sound: sound instance to transform
- transposition: transposition factor (being of any of the formats described above)
- formants: if T (true), formants of the sound are kept after transposition. The amplitude of the partials are scaled according to the spectral envelope of the original sound.
- name: optional name for the new generated sound (if NIL the function is destructive)
- simp: if T (true) partials with a main frequency over half sampling rate or below zero Hertz after transformation, are eliminated from the sound structure.
4.2 Time Stretching

\texttt{stretch-sound sound stretch &key name}

This function performs time stretching over the partials of a sound:

- \textit{stretch}: time stretching factor (being of any of the formats described above). In ATS each partial can be stretched by a different constant or dynamic factor. This can produce spectral structures where vertical relationships between partials are completely altered (diachronic transformation). During synthesis, parameters are interpolated between windows according to this (altered) time information.

4.3 Amplitude Gate

\texttt{gate-sound sound &key limit scaler predicate name}

This function performs a selective filtering of the partials of a sound. The user can define an amplitude threshold in dB and a selection predicate. This predicate compares the mean signal-to-mask ratio (SMR) of the partials with the specified threshold. For example if the predicate is > only partials with mean SMR greater than the threshold are transformed. Relevant parameters are:

- \textit{limit}: SMR threshold in dB

- \textit{scaler}: amplitude scaler. The amplitudes of the selected partials are multiplied by this value. This parameter can be a number (constant scaling) or an envelope (dynamic scaling).

- \textit{predicate}: numeric Lisp predicate used for the partial selection. The predicate takes two parameters: a threshold in dB and the average SMR of the tested partial.

5 Spectral User Interface (SUI)

Besides the described transformation functions, ATS provides a Spectral User Interface (SUI) for real-time spectral transformations. Using real-time CLM capabilities [8], the SUI provides the user with a set of graphic sliders that control transformation parameters during re-synthesis in real-time.

![Figure 2: Spectral User Interface real-time controls](image)

In its present version the SUI has the following controls:

- \textit{Amplitude}: amplitude scaling of the spectral components

- \textit{Transposition}: frequency transposition. All partials of the sound are transposed in frequency as in a variable-speed tape recorder without changing the length of the source.

- \textit{Shift}: frequency shifting. This procedure adds a fixed amount of frequency to all the partials of the sound, creating a spectral translation or compression. If we consider a harmonic spectrum generated by the formula \( y = a \cdot x \), where \( y \) is the frequency of the partial, \( x \) its rank, and \( a \) the frequency value of the fundamental, the spectral shift can be expressed as: \( y = a \cdot x + b \), where \( b \) is the shift factor. The user controls the amount of shift in terms of a percentage of the fundamental frequency of the sound (the default range goes from 0% to 100%).

- \textit{Distortion}: this transformation considers that the source has a harmonic structure (linear spectrum) and lets the user exponentially distort it. Spectral distortion can be expressed as: \( y = a \cdot x^b \), where \( y \) is the frequency of the transformed partial, \( x \) its rank, \( a \) the frequency value of the fundamental, and \( b \) the distortion factor. If the value of \( b \) is 1.0 we obtain a harmonic structure, if we increase its value we get a non-linear frequency structure that is perceived as inharmonic.

- \textit{Proportional Time}: this slider acts as a time-frame "scrubber". The user can move across the frames of the spectral structure
during synthesis or even freeze the synthesis at a given frame. Using the play toggle button the SUI can be set into "scrubbing" mode or into a loop synthesis mode.

6 Synthesis

The synthesis engine of ATS is implemented using the CLM (Common Lisp Music) synthesis and sound processing language [7][8]. ATS has many target synthesis algorithms, the most important ones are:

1. **Additive Synthesis**: Real-time additive synthesis can be performed with the Spectral User Interface (SUI) or the ifft-synth function, both using an IFFT overlap-add algorithm. ATS has also an oscillator-bank additive synthesizer (osc-synth) that allows the use of phase information during resynthesis.

2. **Subtractive Synthesis**: two subtractive algorithms are available using spectral information to control a filter-bank. The function ftt-synth implements the filter bank using an IFFT overlap-add approach, and fmt-synth uses an array of robust bandpass filters (resonators). Both functions can filter white noise or a sound file specified by the user.

3. **Granular Synthesis**: spectral information can be used as a control grid for granular synthesis. In this terms a sound becomes a time-frequency grid that specifies the evolution of sound grains over time. The function grn-synth lets the user design spectral grids to be used in grain generation. The density, shape and duration of the grains can also be controlled dynamically.

4. **Hybrid**: as for transformations, synthesis functions can be combined using macros to create hybrid synthesis methods. For instance, granular synthesis can be combined with subtractive synthesis to perform some kind of granular filtering on a sound file using spectral data to control the filters.

Written in a high level language like Common Lisp, ATS allows for the symbolic processing of spectral traits. Analysis data is stored as a Lisp abstraction called sound, a symbolic Lisp object representing a spectral model, that can be sculpted using a wide variety of transformation functions. ATS sounds can be synthesized using different target algorithms written in CLM. A Spectral User Interface offers graphic-oriented spectral transformation capabilities during real-time resynthesis. ATS provides an environment for sound design and composition that allows the user to explore the possibilities of spectral modeling in a very flexible way.

References


7 Conclusions

ATS is an on-going research and development project, this paper discussed its current state.