ATS User Interfaces

[1]**Juan Pampin**, [2]**Oscar Pablo Di Liscia**, [1]**William 'Pete' Moss**, [3]**Alex Norman**
[1]Center for Digital Arts and Experimental Media (DXARTS), University of Washington
{pampin, petemoss}@u.washington.edu
[2]Carrera de Composición con Medios Electroacústicos, Universidad Nacional de Quilmes
odiliscia@unq.edu.ar
[3]Department of Computer Science and Engineering, University of Washington
anorman@cs.washington.edu

Abstract

ATS is a spectral modeling system based on a sinusoidal plus critical-band noise decomposition. Originally implemented in LISP [1], using the CLM synthesis language [2], ATS has now been ported to C in the form of a spectral modeling library. This library, called ATSA, implements the ATS system API which has served as foundation for the development of the ATSH graphic user interface. Written in GTK+, ATSH not only provides user-friendly access to the ATS analysis/synthesis core but also graphic data editing and transformation tools. This work has also been extended to create interfaces for two popular synthesis languages: Csound and Pd. This paper presents several ways of working with ATS data using a diversity of tools.

1 ATS Overview

ATS is a spectral modeling system developed by Juan Pampin for representing sound as a combination of sinusoidal trajectories and critical-band noise. The analysis engine uses Perceptual Audio Coding (PAC) techniques such as Signal-to-Mask Ratio (SMR) evaluation along with traditional signal processing techniques to achieve perceptually accurate sinusoidal tracking. The system's noise component is modeled using Bark scale frequency warping to evaluate sub-band noise energy, which is then distributed among the partials. The analysis model consists of temporal frames, each of which contains a set of partials having amplitude and frequency values, with optional phase information. Each frame may also contain the psychoacoustic noise information, which consists of 25 values representing the noise energy in each critical band. For detailed information on the ATS system, please visit http://www.dxarts.washington.edu/ats/

```
;;; transpose up an octave and
;;; expand four times keeping attack
(with-sound ()
(sin-noi-synth 0.0 crt-cs6
:frq-scale 2
:time-ptr '(0.0 0.0 0.025 0.1 0.5 0.5 1.0 1.0)
:duration (* (ats-sound-dur crt-cs6) 4)))
```

Figure 1: Example of ATS synthesis with CLM

2 ATS LISP

ATS first implementation was written in LISP using the CLM sound synthesis and processing language. This version of ATS is still available and is probably the most powerful interface to edit or transform data algorithmically. Spectral data is accessible through a library of LISP functions, many complex transformation algorithms are available which can be extended via macros or new functions. Multiple ATS data files can be loaded into the ATS LISP environment and hybridizing operations can create new sounds by merging their data. After being edited or transformed, data can be saved to ATS binary files (these files can be then read by the ATSH GUI, see Section 2) or synthesized using CLM.

3 ATSA

ATSA is a library of C functions implementing the ATS system's API. ATS's peak detection, peak tracking, and psychoacoustic processing algorithms are available in the API, as well as residual analysis and noise modeling tools. ATSA can be compiled in many platforms and uses the sndlib library to read and write many types of sound files.

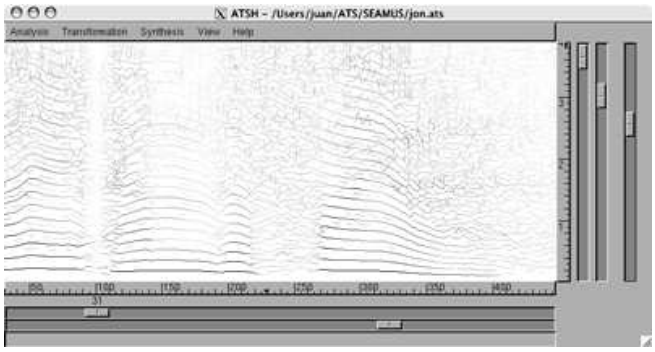

Figure 2: ATSH sinusoidal display

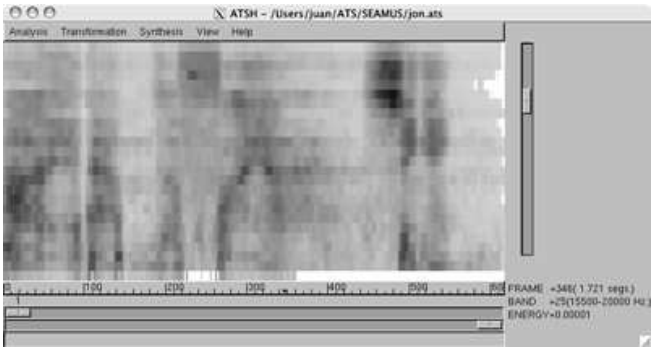

Figure 3: ATSH noise display.

3.1 Tracker

ATS' main analysis algorithm, "tracker", has been implemented using ATSA and is distributed with it as a command-line application. For more information about ATSA please visit: http://www.dxarts.washington.edu/ats/atsa

4 ATSH

ATSH is a graphical user interface for the ATS system. ATSH has been written in GTK+ using the ATSA library API. ATSH can display ATS information in many different ways, it provides a user-friendly interface to the the analysis/synthesis core of ATS as well as a variety of editing ad transformation functions. ATSH's tools are divided into three menus: Analysis, Transformation, and Synthesis.

4.1 Analysis

The Analysis window provides easy access to ATS analysis parameters. ATS' analysis algorithm (called "tracker") is extremely flexible and can be tuned in a variety of ways using up to 16 parameters. ATSH offers the possibility of saving and loading parameters via a binary *.apf file. For a complete description of ATSH's analysis parameters visit http://www.dxarts.washington.edu/ats/atsh

4.2 Transformation

Viewing Data Two types of ATS data can be viewed:

1. *Sinusoidal:* The frequency of each partial is represented on the Y axis, time runs along the X axis. Viewing the amplitude or SMR as a color value is possible, and can be toggled from the View menu. Two horizontal scrollbars control the time view (frame location and zoom), and two vertical scrollbars control frequency display (location and zoom). A third vertical scrollbar sets the contrast for the amplitude or SMR display.

2. *Noise:* The energy value of each of the 25 Critical Bands (in the Bark scale) is shown as a changing color value over time. If the mouse pointer is on the image, the frame number, time, frequency and energy values of its position are printed on the status bar.

Selecting Data There are four ways to select spectral data:

1. Using the Select All, Unselect All, Select Even, Select Odd, and Invert Selection presets from the Edit menu.

2. Using the mouse. This method allows for block selection (a spectral region), or single selection (an isolated partial).

3. Using the List View window (in the View menu). The amplitude, frequency, phase and SMR values of each frame are represented in each column of the list. Vertical (partial) and horizontal (time) selections are available in this view.

4. Using the smart selection menu item. This menu allows the user to select partials in the current view using both amplitude and/or a fix step of partial order. For example, setting from 1 to 10, jump by 2 and Amp. Threshold of -36 will select partials 1, 3, 5, 7 and 9 only if their amplitude (Peak or RMS) is above -36dB.

Editing Data After a selection is made its data can be edited. This is done by applying an envelope (linear or spline) to the amplitude or frequency values of the partials in the selected region. The envelopes can either scale the data by multiplying the envelope values, or offset the data by adding the envelope values.

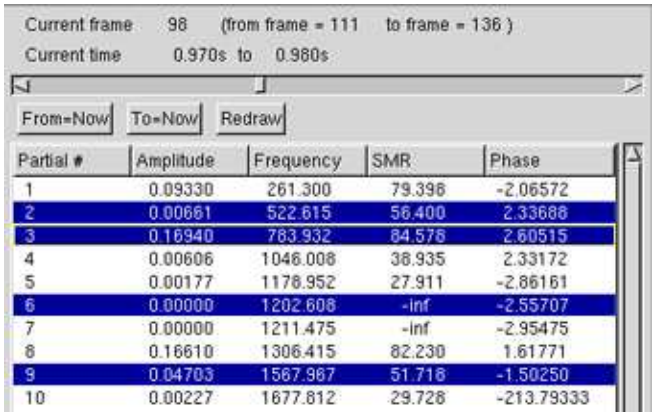

Figure 4: ATSH list display.

4.3 Synthesis

Several features concerning synthesis may be set with the Synthesis/Parameters menu item. The user may scale the overall amplitude and frequency of the original data using scalars. Note also that synthesis may use all the data, or just a selection (if any). It is possible also to use a time function which allows the user to stretch the file dynamically as well as reading it forward or backwards.

5 Csound Interface

Several Csound opcodes exist to extract and synthesize data from ATS files. Many of these opcodes work like Richard Karpen's phase vocoder opcodes for Csound (**pvread**, **pvadd**, etc.). As with **pvread**, most of the ATS-related opcodes take a time pointer that is used to index data from ATS data files. Linear interpolation is used to approximate data between analysis frames.

ATS' Csound opcodes can be broken down into two groups: those that act independently, and those that depend on the opcode **atsbufread**.

5.1 Independent Opcodes

atsread and **atsreadnz** are the most generic ATS-related opcodes. Each of these opcodes simply reads data from an ATS file and returns it for arbitrary use. **atsread** returns the frequency and amplitude information of a user specified partial while **atsreadnz** takes a noise band number and returns the corresponding noise energy data.

atsadd uses an internal oscillator to synthesize an array of sinusoids that are summed to produce a single a-rate output. The range of partials is supplied by the user. An optional frequency multiplier is used before synthesis to transpose the frequency data. Like **pvadd**, **atsadd** provides an optional amplitude "gate" function (defined in a Csound f-table) that is used to scale the amplitudes of the partials.

```
instr 1
ktime line 0, p3, 2.2
aout1 atsaddnz ktime, "cl.ats", 25
out aout1*10000
endin
```

Figure 5: Example Csound instrument using *atsaddnz* opcode

atsaddnz synthesizes a set of noise bands determined by the user. Each noise band is placed in the spectrum by modulating the band by a sine wave set to the corresponding critical-band center frequency. These bands are summed to produce a single a-rate output. An example instrument using this opcode is shown in Figure 1.

atssinnoi synthesizes both sinusoids and noise together using an internal oscillator. The noise energy for each band is distributed among the partials covered by that band, and is used to synthesize the noise component for each partial.

The final independent opcode **atsbufread** has no outputs that are directly accessible by the user. **atsbufread** takes data from an ATS file and produces a table of partial amplitude and frequency values in memory. This table can then be accessed by other opcodes. Like **atsadd**, **atsbufread** uses a time pointer and a list of partials to produce the table.

5.2 atsbufread Dependent Opcodes

atsinterpread takes a single frequency value and uses it to index a table produced by an **atsbufread** opcode and returns the corresponding amplitude value, interpolating between frames. This opcode can be useful for cross synthesizing non-ATS-derived signals with data from ATS.

atspartialtap works almost exactly like **atsread** except it reads data from a table produced by an **atsbufread**. The only argument this opcode takes is a partial number which it uses to return frequency and amplitude data. This opcode is useful if a user wants to operate on multiple partials separately using the same time pointer. While this can easily be achieved with an array of atsreads all using the same time pointer, its simplicity makes it attractive.

atscross, based on **pvcross**, allows a user to perform cross synthesis using the data from two ATS files. One of these ATS files comes from the **atsbufread**, the other is provided by the **atscross** opcode. Data is extracted from the ATS file indicated by the **atscross** opcode, and used to index the table produced by the **atsbufread** opcode. This way amplitudes of one file can be scaled by the other conserving the first file's frequency values. Independent amplitude scalars can be varied to achieve morphing from a standard **atsadd** sound to a

cross-synthesized sound.

6 Pd Interface

The object **atsread** is used to access ATS data in real time within Pd in much the same way as the Csound opcodes **atsread** and **atsreadnz**. However, the Pd object combines the functionality of the Csound **atsread** and **atsreadnz** and outputs lists of data, allowing for one object to output all the partial and noise data for one ATS file. **atsread** has three outputs: a list of frequency values, a list of corresponding amplitude values, and a list of noise energy values. These lists are always given in order from lowest partial/band to highest partial/band. The user opens an ATS file for access by sending the message "open cl.ats", or by supplying the ATS file name as an argument to the object.

The **atsread** object can also be made to only output noise or sinusoidal data by using the "nosines", "nonoise", "sines" and "noise"" messages. These messages can also be given as arguments to the object.

7 Conclusions

We have presented a variety of interfaces to work with ATS data. The LISP interface offers algorithmic processing of spectral data and synthesis extensions to the CLM language. The C library ATSA implements the analysis system API and can be used to develop spectral modeling applications such as the “tracker” command-line program. ATSH is a sophisticated graphic user interface for ATS data. It can be used as an analysis and synthesis front-end as well as for editing and transforming spectral data. This suite of tools is available in open source form at:

http://www.dxarts.washington.edu/ats/ .

References

[1] Pampin, J. 1999. “ATS: a Lisp Environment for Spectral Modeling” *Proceedings of the 1999 International Computer Music Conference.* Beijing: Computer Music Association.

[2] Lopez-Lezcano, F., and J. Pampin. 1999. “Common Lisp Music Update Report” *Proceedings of the 1999 International Computer Music Conference.* Beijing: Computer Music Association.