GABOR, MULTI-REPRESENTATION REAL-TIME ANALYSIS/SYNTHESIS

Norbert Schnell and Diemo Schwarz

Real-Time Applications Team
IRCAM - Centre Pompidou, Paris, France
Norbert.Schnell@ircam.fr, Diemo.Schwarz@ircam.fr

ABSTRACT

This article describes a set of modules for Max/MSP for real-time sound analysis and synthesis combining various models, representations and timing paradigms. Gabor provides a unified framework for granular synthesis, PSOLA, phase vocoder, additive synthesis and other STFT techniques. Gabor's processing scheme allows for the treatment of atomic sound particles at arbitrary rates and instants. Gabor is based on FTM, an extension of Max/MSP, introducing complex data structures such as matrices and sequences to the Max data flow programming paradigm. Most of the signal processing operators of the Gabor modules handle vector and matrix representations closely related to SDIF sound description formats.

1. INTRODUCTION

1.1. Background

Max/MSP and similar graphical programming environments [12, 11, 4] are popular for rapid prototyping and composition of interactive audio and music applications. The popularity of the Max paradigm can been seen as related to the inherently modular and thus intuitive approach to programming using a metaphor of analog synthesizers. In Max, modules are connected to a unidirectional data flow graph, called a *patch*. Starting from the early Max environment allowing only for the processing of streams of integer numbers related to MIDI information, today Max/MSP integrates the processing of audio and video streams. Many other multimedia programming environments provide a similar programming paradigm.

The family of Max languages in the tradition of the Music V music programming languages distinguish event oriented *control* processing and constant rate *audio* stream processing. Max/MSP as well as some similar environments also allow for restricted multirate processing of audio streams.

The Gabor library introduces a signal processing paradigm based on the general idea of streams of atomic sound entities processed with arbitrary timing or arbitrary processing rates. Gabor modules are scheduled within the Max event (or message) processing model rather than the block-wise signal stream processing engine. This way Gabor allows for the combination of different signal processing techniques in a unified framework one could describe as *generalized granular synthesis*.

In summary Gabor relies on two basic ideas of processing sound as *atomic particles* and in *arbitrary rates*:

- Gabor processes arbitrary atomic sound particles such as grains, wave periods or frames and their description by different sets of parameters or coefficients in form of streams of generic vectors and matrices.
- Sound particles are processed at arbitrary rates or moments and thus can be adjusted to the frequency, texture or rhythm of a sound or to the required processing rate of a particular algorithm.

The name *Gabor* is a reference to Dennis Gabor[1] who first raised the idea of "acoustical quanta", particles of sound at the edge of temporal and timbal perception, and who explored their mathematical properties in relation to quantum mechanics [7, 8]. This article will extend the notion of "quanta" compared to the precise definition by Dennis Gabor for his work on information theory and the time-frequency representation named after him.

1.2. Overview

The Gabor library provides various methods to cut a continuous audio stream into a stream of vectors using different temporal schemes. It implements various analysis and synthesis operators treating vectors of sound samples, spectral representations or coefficients. A generic overlap-add buffer reconstitutes a continuous audio stream from a stream of vectors at the output of a Gabor patch (Max sub-program). Figure 1 shows the simplified flow-chart of a Max patch using Gabor.

The current version of Gabor supports the following analysis, synthesis and signal processing techniques:

- Granular synthesis
- PSOLA analysis/re-synthesis
- Phase vocoder and other STFT based techniques
- Sinusoidal analysis/re-synthesis
- Convolution, correlation, etc.
- Estimation of spectral envelopes
- Estimation of various audio descriptors

Gabor is based on FTM [5], an extension of Max/MSP supporting the handling and processing of complex data structures within the Max data flow programming paradigm. Gabor mainly uses the FTM *fmat* class, a simple two-dimensional matrix of floating-point values. The same matrix class is used to represent vectors of sound samples, complex Fourier spectra (two-column matrix) and any other representation of a frame of sound as a vector of analysis coefficients or parameters (e.g. cepstrum coefficients, partials or formants). Gabor matrix formats are kept as close as possible

[1] Nobel prize in Physics 1971 for his invention and development of the holographic method

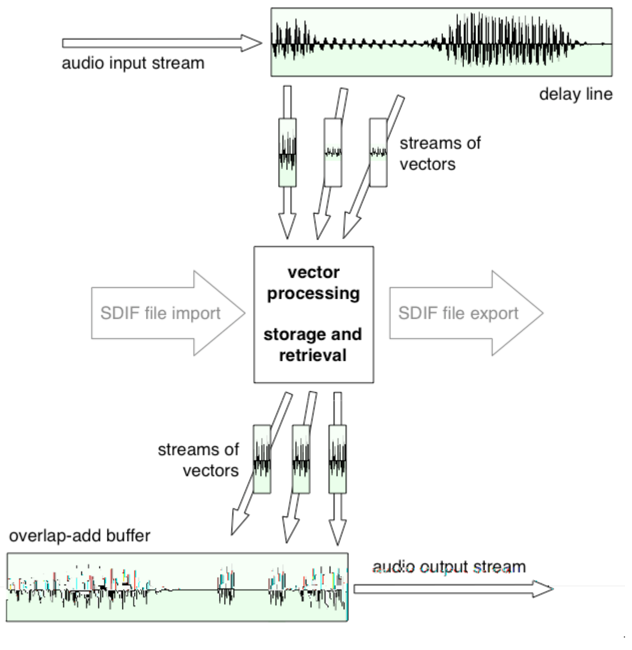

Figure 1: *General data flow of a Gabor program*

to the matrix formats specified for the SDIF Sound Description Interchange Format [16].

Other libraries based on FTM address further vector and matrix processing, statistical modeling and recognition algorithms for sound, gesture and musical data as well as real-time data base handling and information retrieval.

2. STREAMS OF "QUANTA"

Gabor currently provides two different delay line modules. While the module `gbr.dline` represents a usual delay line storing the last samples of an incoming audio stream, the module `gbr.drain` stands for the inverse case of a buffer holding the next samples to be output. For many applications these delay line modules are the interface between audio streams and streams of vectors using the modules `gbr.copy` to copy a vector of sound from a delay line and `gbr.paste` to overlap-add vectors back into a continuous audio stream. The `gbr.paste` module provides various interpolation modes to overlap the incoming stream of vectors with adequate precision.

The timing of a stream of vectors between an input delay line and an output buffer — the moment of their occurrence or their period — is arbitrary but very precisely taken into account by the Gabor modules using 64-bit floating-point precision. A vector is triggered by a message received by the `gbr.copy` module giving a position in the input delay line and size of the vector to be copied and sent from the output of the module. Alternatively `gbr.copy` can also generate a stream of vectors from another (longer) vector for example representing a pre-recorded audio stream (i.e. an imported sound file).

In addition to the generic delay line and copy/paste modules Gabor provides three optimized modules for cutting a continuous audio input stream into a stream of vectors. Each module addresses a particular signal processing technique.

`gbr.grain~` ... copies single grains out of an input stream (combining `gbr.dline` and `gbr.copy`)

`gbr.psy~` ... cuts input stream into elementary wave forms

`gbr.slice~` ... cuts input stream into overlapping vectors specified in samples (especially for STFT use)

A more complex Gabor module, `gbr.svp~`, provides the optimized analysis stage of an advanced phase vocoder [3, 9] integrating multiple FFT calculations and phase vocoder specific processing. The module directly provides matrices of frequency domain data at its output.

Finally a module `gbr.ola~` is provided as a generic sink of vector streams combining the two modules `gbr.paste` and `gbr.drain~`. Usually all analysis/re-synthesis techniques using one of the above modules to generate a stream of vectors use the `gbr.ola~` module to reconstitute a continuous audio stream from the incoming vectors by overlap-add.

2.1. Grains and Sound Segments

Granular synthesis is the most general case of composing an audio stream of or cutting an audio stream into atomic entities [18, 13]. For granular synthesis with Gabor, a stream of vectors can be generated from an audio stream using the `gbr.grain~` module specifying grain position (i.e. the onset time of the grain in a delay line or another vector) and grain duration in milliseconds.

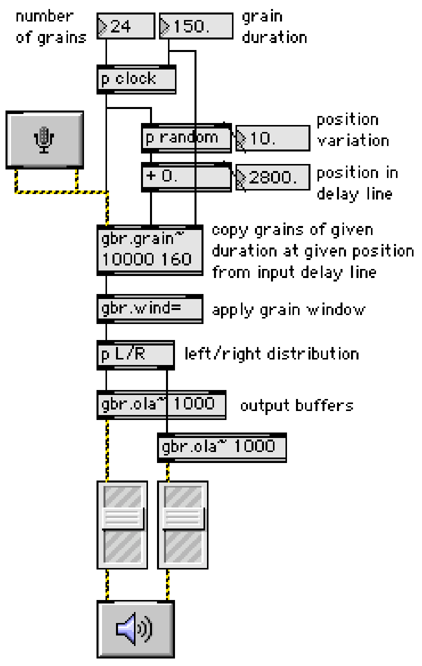

Figure 2: *Granular synthesis example using a delay line.*

Figure 2 shows a simple example applying stereo granulation to a real-time audio stream using a delay line. By varying the position in the delay line the user can continuously navigate through the last 10 seconds of the incoming sound. The stream of vectors stays unprocessed apart from a alternating distribution to the left and right audio channel at the output.

As with grains, one can work with streams of longer sound segments corresponding to beats, notes or atomic musical phrases or sound samples. Gabor proposes here a general alternative to the usual unit generator approach of Max/MSP and many other computer music languages and programming environments.

2.2. Waves and Periods

PSOLA (Pitch Synchronous Overlap-add) requires pitch synchronous processing [10], whereby a wave period may not be a multiple of the sample period. In this sense modular real-time PSOLA applications utilize *variable rate* signal processing (as opposed to *multi-rate*).

For PSOLA analysis and re-synthesis applications the module `gbr.psy˜` was created. It cuts an incoming audio stream into a stream of vectors representing successive elementary waveforms if the signal is voiced. For the unvoiced parts, arbitrary grains of a fixed period and duration are output. The module uses the *Yin* algorithm [2] and provides the current frequency (zero for unvoiced parts) as well as the periodicity factor and the energy of the current waveform.

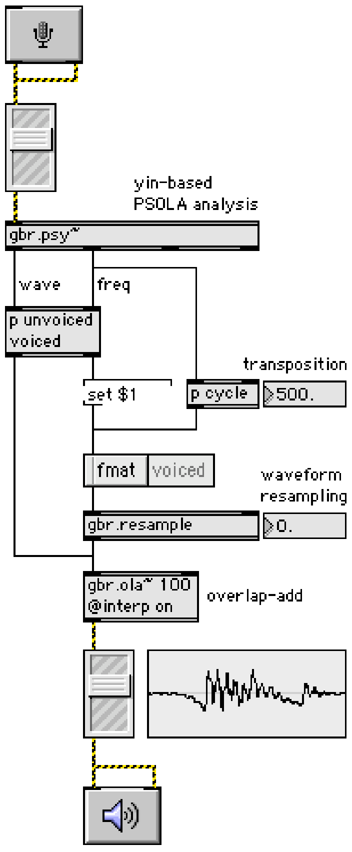

Figure 3: *PSOLA analysis/re-synthesis example.*

Figure 3 shows a simple PSOLA analysis re-synthesis example. In this example the frequency value is used to separate voiced signal periods and unvoiced grains into two streams of vectors. While the unvoiced vectors are immediately copied into the overlap-add buffer, the elementary waveforms are stored in a static vector (called *voiced* in the example). The static vector is output with a frequency depending on the original analysis frequency (i.e. allowing for transposition given in cents).

2.3. STFT and company

In order to provide an optimized interface for constant rate signal processing techniques such as those based on short-time Fourier transforms, the module `gbr.slice˜` was introduced. It cuts the incoming audio streams into slices of overlapping vectors of a fixed size specified in samples.

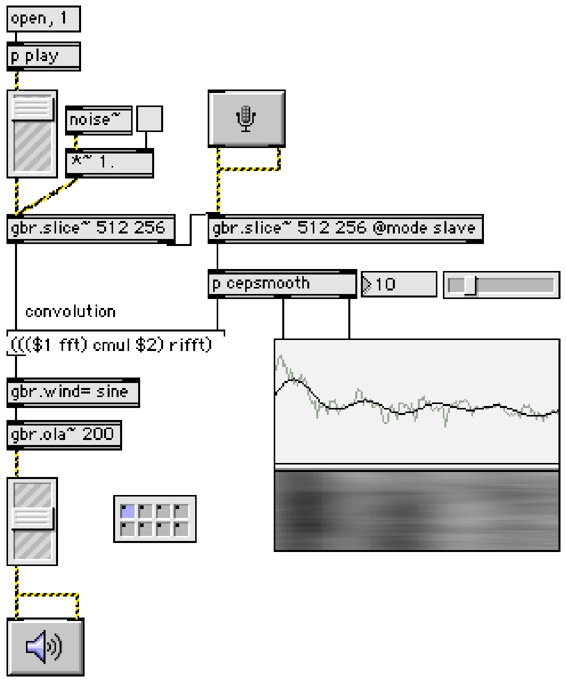

Figure 4: *STFT convolution with estimated spectral envelope.*

The example in figure 4 uses two synchronized `gbr.slice˜` modules to perform a cross-synthesis of two incoming audio signal streams. On the stream at the right of the example a spectral envelope is estimated using a cepstrum method [17] with the given number of cepstrum coefficients. The spectral envelope is applied to the stream on the left by convolution multiplying the two streams of vectors in the frequency domain before applying an IFFT and outputting the result to the `gbr.ola˜` module.

Additive synthesis integrates with Gabor as an STFT technique using the *FFT-1* method based on inverse Fourier Transform [14]. Using this technique the Gabor additive synthesis modules add partials to a short-time FFT spectrum and could be easily combined with the above convolution example of figure 4.

Gabor integrates a complex phase vocoder analysis stage which directly transforms an incoming audio stream into frequency domain data. The `gbr.svpana˜` module outputs vectors of a spectral representation giving an amplitude, a frequency, a phase and a

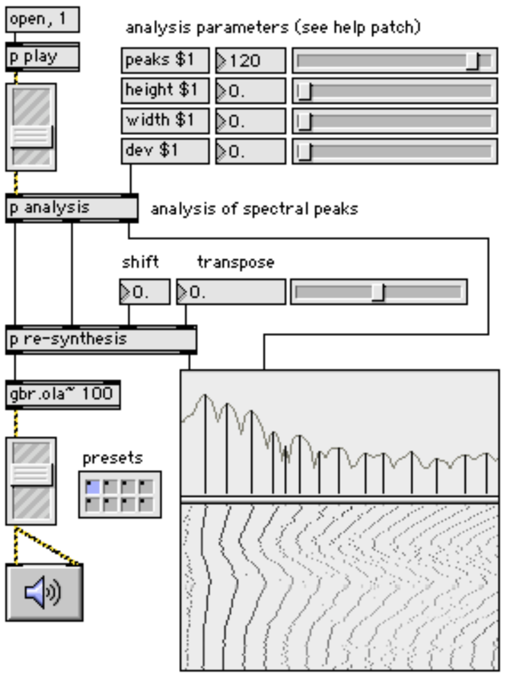

Figure 5: *Sinusoidal analysis/re-synthesis example.*

class for each FFT bin. The module internally performs a classification of the FFT bins [15] distinguishing bins related to sinusoidal spectral components (i.e. partials) and bins being part of a signal transient. The representation allows for dedicated processing of the classified bins obtaining high quality spectral domain processing such as time-stretching, pitch shifting and cross-synthesis.

3. DSP TIMING AND SCHEDULING

Usually audio processing in Max is handled by a dedicated computation engine, MSP (*Max Signal Processing*). As in similar programming environments and plug-in hosts, constant rate audio signals in MSP are processed block-wise by a chain of function calls deduced from the graph of audio processing modules of a Max patch. MSP provides an efficient interpretation of a given graph of modules with a short edit-compile-and-run loop and has the potential of execution with constant computation load.

Particular MSP sub-patches allow for restricted multi-rate processing by changing the calculation block size and block rate within a defined region of the graph. In this way, short time FFT based frequency domain processing using a constant FFT and hop size is introduced into Max/MSP.

In this context, pitch-synchronous processing or granular synthesis was only possible within integrated synthesizer modules without providing modularity for the audio processing operators.

In order to be able to combine constant rate processing with pitch-synchronous and general granular synthesis in a modular framework, Gabor modules have to be able to exchange vectors with an arbitrary constantly changing rate or, in other words, at any moment. As mentioned above the vectors exchanged between the Gabor modules are scheduled within the message system of Max based on a discrete event computation model. Gabor allows this way to create and process streams of vectors adjusted to the frequency, texture or rhythm of a particular sound or the required processing rate of a particular algorithm for analysis, transformation and synthesis.

3.1. "Logical" Time

Regarding the logic time of Max/MSP, each message or event sent from one module to another is accompanied by a 64-bit precise floating-point time. When converting an audio stream to and from a stream of vectors, these time-tags are carefully taken into account by the Gabor modules. In addition, when copying samples from a continuous sample stream (i.e. delay line or vector) to a stream of grains or wave periods, a fractional offset and exact duration can be stored within each vector. These values correspond to the precise starting position of the sound segment between two actual samples of the original audio stream and the precise duration.

The overlap-add module uses the Max message system time-tag of the incoming vector in combination with the fractional offset value for the precise reconstitution of a signal stream at the output. An optional delay time can paste the vector arbitrarily at a "later" position into the overlap-add buffer. Another module which takes into account the fractional offset and duration of a vector is `gbr.wind=~`. The module adjusts a chosen window to the exact onset and duration of the grain or wave period.

3.2. "Real" Time

The Max/MSP scheduler alternates the signal block calculations of the MSP engine and the event processing initiated by the user interface, alarm system and external inputs (MIDI etc.). For each scheduler tick, the "logical time" advances by the period of time corresponding to a block of samples. In the ideal case, all calculations related to events or messages belonging to the time period of a block are computed within this period. This allows the DSP subsystem to read the last block of incoming audio samples and to deliver the next block of outgoing samples in time to the output. The minimum delay of the logical timing between the input and the output of a Gabor application corresponds to the block size of the MSP signal processing engine. The actual delay corresponds to the size of vectors cut out of the input stream rounded to up the nearest multiple of the DSP block size.

The actual source of time and synchronisation for the MSP audio processing as for any similar signal processing program, is the underlying audio system or more precise the clock of the ADC and DAC converters. In this way, Max/MSP, as many other applications, is more or less loosely synchronised depending on the size of the i/o buffers used by the audio drivers and the Max/MSP audio interface. Max provides the signal processing block size and the audio i/o buffer size as two different parameters.

These temporal aspects of the Max/MSP scheduling have to be taken into account when Gabor is controlled by external input. In this scenario, the audio stream has to be synchronized with external input streams such as OSC (*Open Sound Control*) [19] or MIDI. Current efforts in the development of FTM target the integration of OSC allowing for the transmission of streams of vectors as used by Gabor with an exact reconstitution of the temporal aspects such as frequency, texture and rhythm at the reception [1].

4. MEMORY AND PERSISTENCE: RECORDING, SAMPLING AND SDIF

The transformation of a continuous stream of samples to a stream of events within the Gabor modules associates each vector to a precise instant of time transforming continuity of the original stream to a new continuity of a stream of events. The grains, wave periods or frames processed by the Gabor modules can be called *atomic* in the sense of being treated each as a *single event*. The internal temporal development of a vector is conserved by its indices regarding the current sample rate.

In analogy to usual audio *delay lines*, *samplers* and *sound file players*, a number of interesting applications use the possibility of recording a stream of vectors and replaying it later. Usually the FTM class *track* is used as a generic container of streams of vectors. In fact, a track can contain a time-tagged sequence of any kind of events or messages. Several modules are provided to delay, record and replay sequences of events in different ways using an FTM *track* object. Other modules allow for reading the track data at arbitrary positions, speed and direction, interpolating successive vectors or matrices.

Delay lines and recording buffers can be inserted at any stage of a complex analysis/synthesis algorithm built with Gabor in a modular way. Multiple synthesis processes can use the same data previously analyzed in real-time or imported from a file. Analysis data can be recorded in real-time and exported to a file.

The FTM *track* class allows for importing and exporting SDIF[2] files [16]. This way data can be exchanged with other analysis, editing and processing applications such as *Audiosculpt* and *OpenMusic*.

5. CONCLUSIONS

Gabor provides a novel approach to modular signal processing. It combines various vector based signal processing algorithms in a unified framework using an event processing model. Gabor integrates easily with the Max/MSP graphical programming paradigm and explores the possibilities of this approach in a coherent set of modules.

Granular synthesis, PSOLA, additive synthesis and the phase vocoder are well established and distinguished techniques of audio processing. All these techniques are clearly interrelated in terms of their granularity of calculation and provide partly similar partly complementary possibilities of sound analysis, transformation and synthesis. Using Gabor these techniques can be easily combined.

Additional new impulses for sound synthesis applications are awaited from the integration of Gabor with other libraries based on FTM. The MnM library provides modules for linear vector algebra and statistical models [6]. Further efforts in the framework of the FTM project target the integration of data base access and content based processing.

The Gabor modules are freely available within the FTM distribution for Max/MSP at `http://www.ircam.fr/ftm`.

6. ACKNOWLEDGEMENTS

We first of all have to acknowledge the researchers at IRCAM who contributed to the various algorithms meeting in the Gabor library and who provided vivid support for their implementation, especially Xavier Rodet for the FFT-1 addtiive synthesis, Geoffroy Peeters for PSOLA, Alain de Cheveign for Yin and Axel Röbel for the phase vocoder. Further on we'd like to thank the other members of the Real-Time Applications Team, Riccardo Borghesi and Frederic Bevilacqua as well as Remy Muller and Patrice Tisserand for their friendship and daily inspiration for the FTM project Gabor is part of.

[2] `http://www.ircam.fr/sdif/`

7. REFERENCES

[1] E. Brandt and R. Dannenberg. Time in Distributed Real-Time Systems. In *ICMC*, Beijing, China, 1999.

[2] Alain de Cheveign and Hideki Kawahara. YIN, a Fundamental Frequency Estimator for Speech and Music. *JASA*, 111:1917–1930, 2002.

[3] Mark Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[4] F. Dechelle et al. jMax: a new JAVA-based editing and control system for real-time musical applications. In *ICMC*, Ann Arbor, Michigan, 1998.

[5] N.Schnell et al. FTM — Complex Data Structures for Max/MSP. In *ICMC*, Barcelona, Spain, 2005.

[6] R. Muller F. Bevilacqua and N. Schnell. MnM: a Max/MSP mapping toolbox. In *NIME*, Vancouver, Canada, 2005.

[7] Dennis Gabor. Theory of communications. *Journal of the Institution of Electrical Engineers*, 93(3):429–457, 1946.

[8] Dennis Gabor. Acoustical Quanta and the Theory of Hearing. *Nature*, 159(4044):591–594, 1947.

[9] J. Laroche and M. Dolson. New Phase Vocoder Technique for Pitch-Shifting, Harmonizing and Other Exotic Effects. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, New Paltz, New York, 1999.

[10] G. Peeters et al. N. Schnell. Synthesizing a choir in real-time using Pitch-Synchronous Overlap Add (PSOLA). In *ICMC*, Berlin, Germany, 2000.

[11] M. Puckette. Pure Data. In *ICMC*, pages 269–272, Hong Kong, 1996.

[12] M. Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002.

[13] Curtis Roads. *Microsound*. The MIT Press, Cambridge, Massachusetts, 2002.

[14] X. Rodet and P. Depalle. Spectral Envelopes and Inverse FFT Synthesis. In *Proceedings of the 93rd Convention of the Audio Engineering Society, AES*, New, York, 1992.

[15] Axel Roebel. Transient detection and preservation in the phase vocoder. In *ICMC*, Singapore, 2003.

[16] D. Schwarz and M. Wright. Extensions and Applications of the SDIF Sound Description Interchange Format. In *ICMC*, Berlin, Germany, 2000.

[17] Diemo Schwarz and Xavier Rodet. Spectral Envelope Estimation and Representation for Sound Analysis-Synthesis. In *ICMC*, Beijing, China, 1999.

[18] B. Truax. Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2):14–26, 1988.

[19] M. Wright and A. Freed. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers. In *ICMC*, Thessaloniki, Greece, 1997.