Where's The Beat? Tools for Dynamic Tempo Calculations

Jan C. Schacher, Martin Neukom,

Institute for Computer Music and Sound Technology,
Zurich School of Music, Drama and Dance
jan.schacher@hmt.edu, martin.neukom@doz.hmt.edu

Abstract

"Timegrid" is a set of methods and tools geared towards working with polyphonic tempo structures. Common tempo operations such as linear and exponential accelerandi and more uncommon ones such as arbitrarily complex functions are used, which are converted to simple time and tempo pairs. Aimed at the composer these processes produce a variety of files useful for audition, graphical scoring and direct use in algorithmic composition. The methods and tools described here can be used to solve a variety of problems. As an additional example a method for the generation of random numbers from an arbitrary distribution is shown. "Timegrid" is implemented as an external and an application in MaxMSP and made publicly available together with its source code.

1 Introduction

Polyphonic tempo structures or so called *tempo canons* have been known since Lew Termen's "Rhythmicon" built for Henry Cowell (Schedel, 2002) and Conlon Nancarrow's work for player pianos (Gann, 1963). In computer music and more specifically in algorithmic composition this technique has been used with varying success and to different degrees of complexity. We clearly differentiate between tempo polyphony and polyrhythmic structures. In tempo polyphony voices don't share an underlying grid of pulsation, as is the case in polyrhythm. On the contrary, each voice runs at an independent tempo and is based on subdivisions derived from this basic pulse. Ratios, superposition, and synchronization between voices are achieved only in an absolute time domain and not on an abstract structural level.

To use this technique a composer faces several challenges in the domain of notation and audition as well as with the more practical problems of playability. Today's main notation and sequencer applications are very mature but lack one essential feature. The use of independent, per-voice tempo is not possible and spatially synchronized notation cannot be generated from such structures.

We are addressing the tasks of defining functions necessary to generate evolving tempo grids and have developed several simple means of visualizing and auditioning the resulting poly-tempo patterns. The primary target audience for this work is not the musician with aptitudes in developing and programming ideas in symbolic systems such as Common Lisp[1], Open Music[2] or PWGL[3]. Our intention is much rather to give those composers a succinct set of tools who can conceptualize and define rhythmic processes such as tempo polyphony but don't want to or cannot invest the time necessary to learn a proper programming language. The goal is to bridge this gap and provide a simple modular tool chain within an application-like framework with a simple and intuitive graphical interface.

2 Methods and Tools

The Framework we implemented is similar to a tool chain providing a modular approach. Each stage of development is separated and hands its result to the following module in a neutral format. This enables the workflow of the composer to be extended and customized in various ways without the need to continuously rebuild existing functionalities. The basic information generated and transported between modules is a simple list consisting of timestamp and tempo pairs that can be stored and manipulated as plain text files and expressed or applied in numerous ways. The output in a graphical file format is compatible with all current vector graphics packages such as Inkscape[4] and is intended to be used as a basis for paper-based score layout.

2.1 Definitions

Let's begin by defining the basic terms used in this paper. The beat is the basic pulsation or temporal subdivision upon which the music is based. Musical tempo T is defined as beats b per time unit t

$$T = \frac{b}{t} \qquad \text{with t in minutes} \qquad (1)$$

The duration of a certain number of beats is

$$t = \frac{b}{T} \qquad (2)$$

[1] http://common-lisp.net/ -> all URIs Valid in May 2007
[2] http://recherche.ircam.fr/equipes/repmus/OpenMusic/
[3] http://www2.siba.fi/PWGL/
[4] http://www.inkscape.org/

and the number of beats occurring within a specific time is

$$b = Tt. \tag{3}$$

Since tempo is a function of time T = T(t), it must be defined as derivative of the beat function b(t)

$$T(t) = \frac{d}{dt} b(t) \tag{4}$$

and the beat function (the number of beats as function of time) as the integral of the tempo function

$$b(t) = \int_0^t T(\tau) d\tau \tag{5}$$

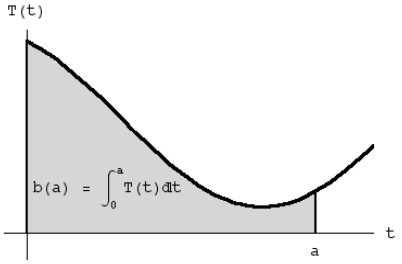

Figure 1. Beat function as the integral of the tempo function

The inverse function to the beat function b(t) is the time function t(b) which gives the time at which beat b occurs.

2.2 Linear and Exponential Functions

The most commonly used tempo variations are linear or exponential functions. An interesting fact relating to the use of these functions is the way we perceive tempo change. Much as in the frequency domain we perceive changes in tempo on a logarithmical scale – when hearing the difference between linear and exponential accelerando we tend to find the latter more natural. (Figure 1.) In *Timegrid* we give several ways of controlling linear and exponential accelerando, allowing the user to choose which of the defining variables length, number of beats, starting or ending tempo to leave undefined according to the constraining criteria arising from his compositional ideas.

2.2.1 Linear Tempo Functions

If we denote the tempo at time 0 with T_0 and at time a with T_a the tempo function is

$$T(t) = ct + T_0 \tag{6}$$

with

$$c = \frac{T_a - T_0}{a} \tag{7}$$

and the beat function

$$b(t) = \int_0^t (c\tau + T_0) d\tau = \frac{ct^2}{2} + T_0 t \tag{8}$$

With these formulas the tempo and the exact time of every beat can be calculated.

2.2.2 Exponential Tempo Functions

Given T_0 and T_a the tempo function is

$$T(t) = T_0 e^{ct} \tag{9}$$

with

$$c = \frac{\log\left(\frac{T_a}{T_0}\right)}{a} \quad \text{and} \quad a = \frac{\log\left(\frac{T_a}{T_0}\right)}{c} \tag{10}$$

the beat function is

$$b(t) = \int_0^t (T_0 e^{c\tau}) d\tau = \frac{T_0 (e^{ct} - 1)}{c} \tag{11}$$

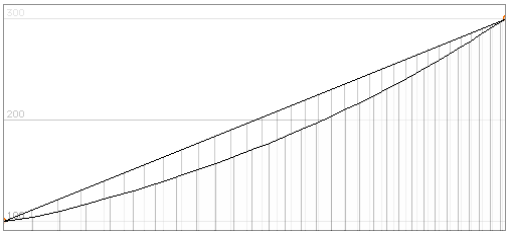

Figure 2. Linear and exponential accelerando evolving from T100 to T300 over the duration t of 10 seconds.

2.3 Arbitrarily Complex Functions

The manipulations described above can only be applied in a closed form to very simple functions. Arbitrarily complex or manually generated functions must be treated numerically. These functions can originate from tables drawn manually or from complex functions evaluated to a certain number of points. Basically they can be obtained from any dataset imaginable. In order to obtain a maximal temporal resolution these functions are first rendered or interpolated to a new set of data based on the smallest time unit used, in our case the millisecond, and then integrated. The integration is achieved with a discretization of time (with sampling rate sr) and the following summation.

$$b(t) = \frac{1}{sr} \sum_{k=0}^{t*sr} T\left(\frac{k}{sr}\right) \tag{12}$$

The integrated function can now be evaluated for points where an integer value is reached. This point corresponds to the resulting beat-position.

We choose the spline as graphical element used in a graphical editor, mainly for its versatility and ease of use. The spline polynomial function generates fairly complex curve behaviors from very little control point information and lets one rapidly approximate the shape of other functions. (Figure 2.) For a composer the visualization and superposition of these graphics is a very powerful means of

exploring the relationships and learning about the properties of his ideas concerning tempo changes.

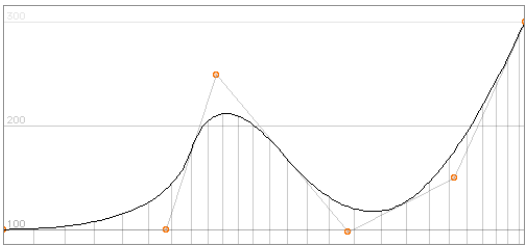

Figure 3. Graphical editor for tempo functions based on splines of arbitrary complexity with a display of the resulting beats.

2.4 Synchronizing Voices

In order to generate musically poignant figures, tempo evolutions at times need to coincide on a single event. To achieve this synchronization of voices a different approach is necessary. Instead of deciding on variables such as number of beats per segment of start tempo / end tempo combinations, the use of curves going through specific time/beat locations produces tightly synchronous moments with less strict time grids in between. (Figure 4.)

The following procedure can be used to get beats from voices with differing tempo functions to coincide. The starting point is a list of times of the coincidences and the number of beats of every voice at these times.

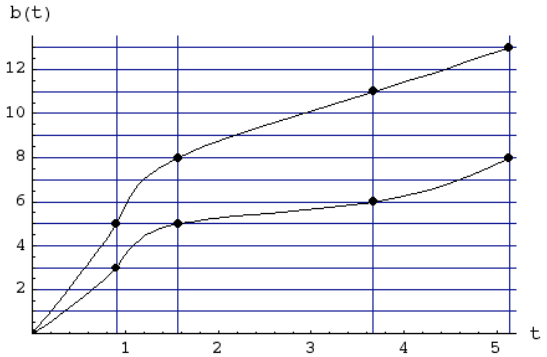

Figure 4. Graph representing points to be synchronized between two voices and the corresponding tempo function as a cardinal spline through the given points.

The beat functions are created as spline functions through these points. Since the spline must include the given points we implemented this as a cardinal spline, a subset of the hermite spline[1]. From these beat functions the remaining functions and the specific time and tempo values can be calculated.

2.5 Other Functionalities

To make the results of these processes more usable for the composer a few auxiliary functions were created, such as adding an offset, concatenating and merging different time / tempo sets. The idea is that those blocks of tempo lists can be manipulated and previewed inside the framework before being exported to one of the resulting file formats.

One essential feature is the facility to audition the polyphony immediately from within the framework. This is achieved by adding a basic MIDI-playback engine that can play any general MIDI instrument provided by the operating system's built-in synthesizer. With the possibility of exporting the voices to a type I MIDI-file the information can also be used in a traditional sequencer, even if that program cannot handle the independence of tempo between the voices since the standard MIDI format can only contain one tempo-map per file.

One of the most useful exportable file formats for composers is certainly the graphical score. This is done via an encoding to the SVG-format[2], which is in essence a xml-structured text format and can even be viewed in web-browsers, but is intended for use in a full-blown vector-graphics application. It's fairly easy to implement different page-layouts and displays of the time-information in a spatial notation and a number of options for placing the time-grids and different paginations are given.

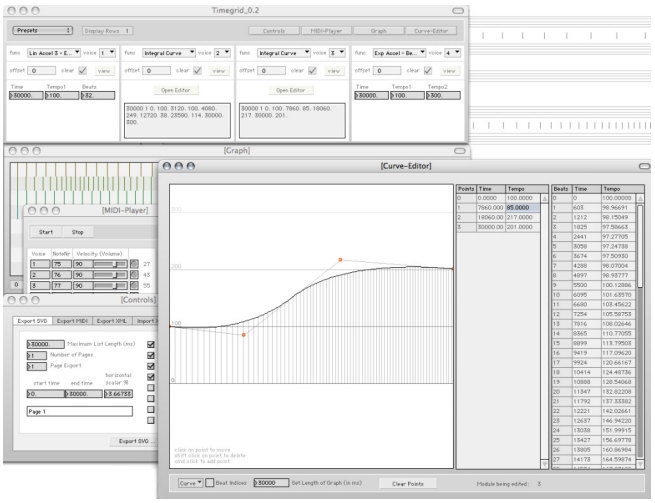

Figure 5. Screenshot of the *Timegrid* Application

3 Random Distributions

An additional example of a method using arbitrarily complex functions is the generation of random distributions. Random number generators exist only for certain distributions. In order to generate random numbers with an arbitrary distribution the following general approach can be used.

A random variable is characterized by its probability density function or by its cumulative distribution function. The probability density function f(x) shows the probability with which x occurs. More precisely: the probability that the random variable is in the interval [a, b] equals the area

[1] http://www.cubic.org/docs/hermite.htm
[2] http://www.w3.org/TR/SVG/

under f(x) from a to b.

$$P(a < x < b) = \int_a^b f(x)dx \qquad (13)$$

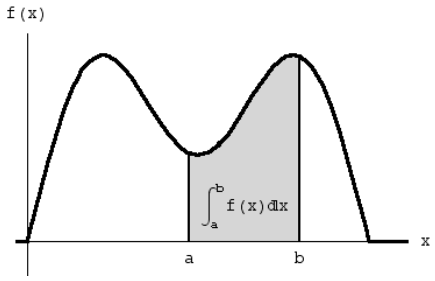

Figure 6. Probability density function.

The whole area under f(x) must be 1:

$$\int_{-\infty}^{\infty} f(x)dx = 1 \qquad (14)$$

The cumulative distribution function F(x) of the variable X is defined by

$$F(x) = P[X \le x] = \int_{-\infty}^{x} f(u)du \qquad (15)$$

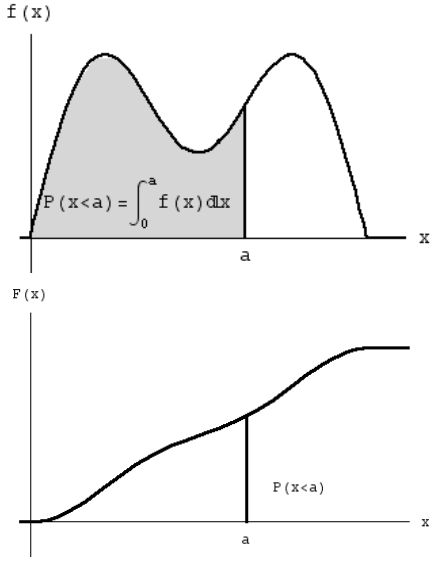

Figure 7. Probability density function f(x) and cumulative distribution function F(x).

The inverse function F^{-1} of the cumulative distribution function F applied on uniformly distributed random numbers produces random numbers with the density f(x) as figure 8 shows.

The *Timegrid* external calculates each function from a vector and finally generates the inverse function F^{-1} through interpolating stepwise from one axis to the other. All these intermediate functions can be exported as lists of discrete values and can be reused for other processes, such as synthesizing the waveform in Xenakis' (Xenakis 1963) dynamic stochastic synthesis.

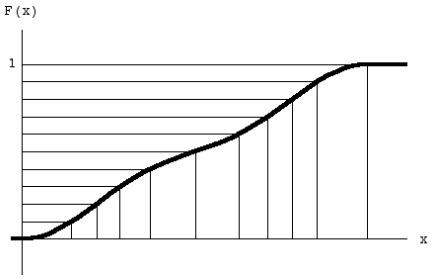

Figure 8. Uniformly distributed random values in F(x) correspond to random numbers x with the density f(x).

The same graphical interface can be used to generate the arbitrarily complex random distribution function.

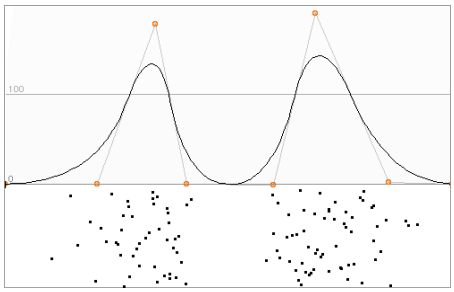

Figure 9. Random distribution graphically manipulated by a spline.

4 Conclusion

The *Timegrid* methods and tools are geared towards the composer. It is obvious that in the field of algorithmic composition other more powerful methods for generating varying time patterns exist. However none of these is user-friendly enough to be used by non-programming musicians. The biggest challenge and the weak spot in all existing software is how to generate music notation that both accurately represents the independent time flows and doesn't compromise readability of the resulting score. *Timegrid* is a step towards that goal. Future work will go into exploring the possibilities offered by linking to an open music notation packages such as Lilypond[1] for proper high-quality score typesetting.

The *Timegrid* external, the MaxMSP application and their sources can be freely downloaded from: http://www.icst.net/downloads/ (URI valid in May 2007)

5 References

Gann, K.; 1996/2006. The Music Of Conlon Nancarrow. *Cambridge University Press* ISBN-13: 9780521028073

Schedel, M., Anticipating interactivity: Henry Cowell and the Rhythmicon. *Organised* Sound (2002), 7: 247-254 Cambridge University Press

Xenakis, I.; 1963/1992. Formalized Music p. 289-293, *Pendragon Press, Hillsdale NY* ISBN-1-57647-079-2

[1] http://www.lylipond.org