Analysis, Data Reduction, Composition and Re-Synthesis in KLANGPILOT

Johannes Kretz
Institute of Composition and Electroacoustics,
ZiMT, University of Music and
Performing Arts Vienna, Austria
zimt@mdw.ac.at
http://www.mdw.ac.at/zimt

Ádám Siska
ZiMT, University of Music and
Performing Arts Vienna, Austria
sales@sadam.hu
http://www.sadam.hu

KLANGPILOT is an environment for sound analysis as well as for control of sound synthesis. Currently additive, subtractive and formantic synthesis are supported in parallel through optimized externals for Max/MSP with multi-processor support. Sound samples can be analyzed for their spectral content as well as for their noise components. These data are then submitted to a process of data reduction, abstraction and simplification in order to turn them into human readable models of sounds. Alternatively the user can define such models from scratch using a highly intuitive set of editors for frequencies, amplitudes, envelopes and other musical parameters. Equally one can modify the sound models obtained by analysis in the same interface. These sound abstractions can be arranged into a 'timbral score', an extension to the classical piano roll capable of displaying detailed spectral information as well. This new paradigm may ease composing music, where the visibility of sound details is crucial for the artistic process. Furthermore one can create and handle hybrid sounds through a morphing algorithm, allowing the interpolation between given sound models. Differently to the classical analysis/re-synthesis approach, KLANGPILOT aims to reduce the complexity of sound description to a minimum: instead of seeking the (almost) perfect re-synthesis of original sources, the main focus is the idea of providing a spectral score language by extending classical music notation. This new score language — which also can be connected to tools for computer aided composition — could have a strong impact on electronic composition, comparable to the impact of music notation and music printing in earlier centuries.

Introduction

When extending the creation of music with the use of electronics and even when using non-standard playing techniques on acoustical instruments, we are still, in some sense, in a state similar to 'oral culture' (Bennett, 1996). The articulation of an expressive vocabulary, which cannot be represented well by the traditional music score, often forces composers to use confusing or vague verbal descriptions and/or the distribution of heterogeneous performance material, like a paper score plus digital media. Both verbal descriptions and mixed performance material lack standardization. This makes the creative process difficult and performances nearly impossible to pull off without the composer's presence. Furthermore, we have to be aware that on one hand pitch, dynamic level, and rhythm can be notated in a way that accurately/reasonably represents the aural result, allowing the experienced musician to imagine the sound by looking at the notation. On the other hand timbral characteristics of music cannot easily be notated with this degree of precision. We must either use symbolic or textual description of actions (fingerings, playing techniques etc.) in the case of paper scores, or save technical parameters for computer programs in some abstract file format. Both methods are not intuitive and efficient working methods for composers.

Especially when working with sound synthesis, the need for an extended graphical score language (that is human-readable but still can handle the complexity of all needed sound parameters) becomes obvious. While several attempts have been made, ranging from the graphical score for Ligeti's 'Articulations' (Ligeti & Wehinger, 1958) to software like *l'Acousmographe*[1], none of these truly solves the problem, since they provide post-facto symbolic representations of analysis put together after the creative process, and are not suitable as production tools. Other graphical scores like those by John Cage or Roman Haubenstock–Ramati (Karkoschka, 1972) leave much ambiguity of interpretation to the performer and are far less specific than traditional scores, which restricts their usefulness to specific aesthetic approaches.

General Design

From the very beginning the design of KLANGPILOT (Kretz, 1999, 2002) was inspired by the work of Marco Stroppa (Agon, Stroppa, & Assayag, 2000), Jonathan Harvey (Harvey, 1981; Machover, 1984), Jan Vandenheede (Vandenheede, 1991; Vandenheede & Harvey, 1985) and Steven McAdams (McAdams, 1982, 1989) and experiences with the Patchwork environment (Laurson & Duthen, 1989). Also the idea of accessing sub-parameters of timbre like brightness, spectral flux, percussivity and harmonicity as described by Grey and Moorer (Grey & Moorer, 1978) was essential.

The current version of KLANGPILOT is under development at the Centre for Innovative Music Technology (ZiMT) at the University for Music and Performing Arts Vienna[2]. It is realized completely within the Max/MSP environment and uses highly optimized externals for sound synthesis (programmed in C++ by Ádám Siska). The GUI, programmed by Johannes Kretz, consists of a *Score Editor* (see Figure 1),

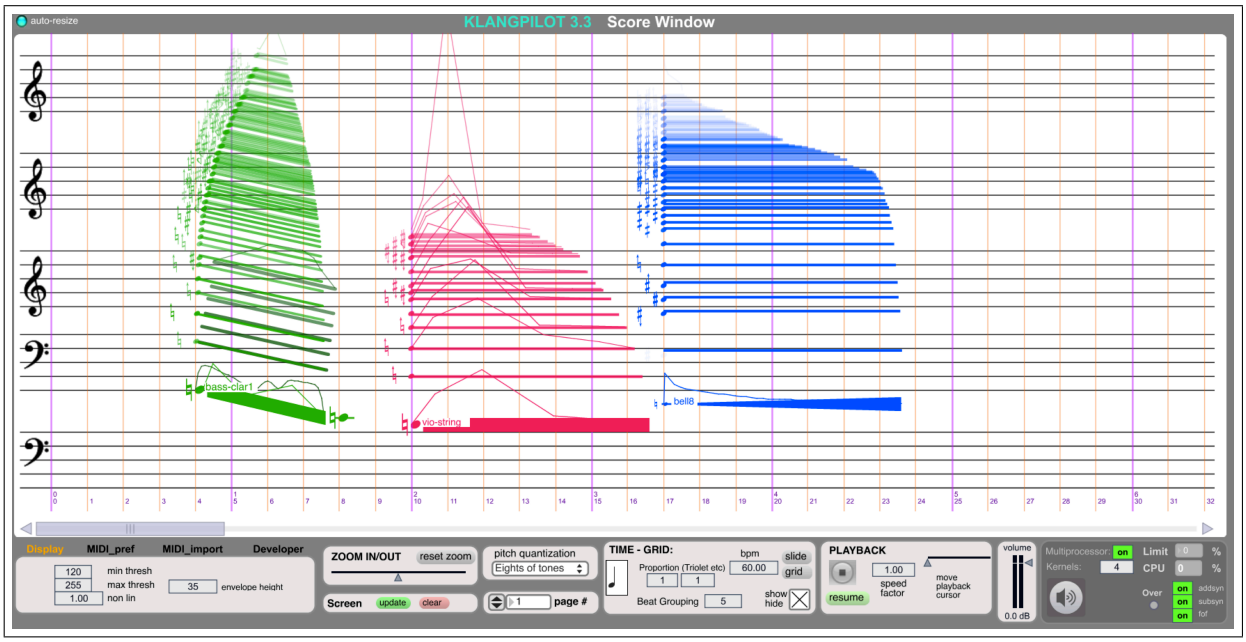

Figure 1. An example score in KLANGPILOT, revealing as well the main control elements of the software on the bottom of the screen.

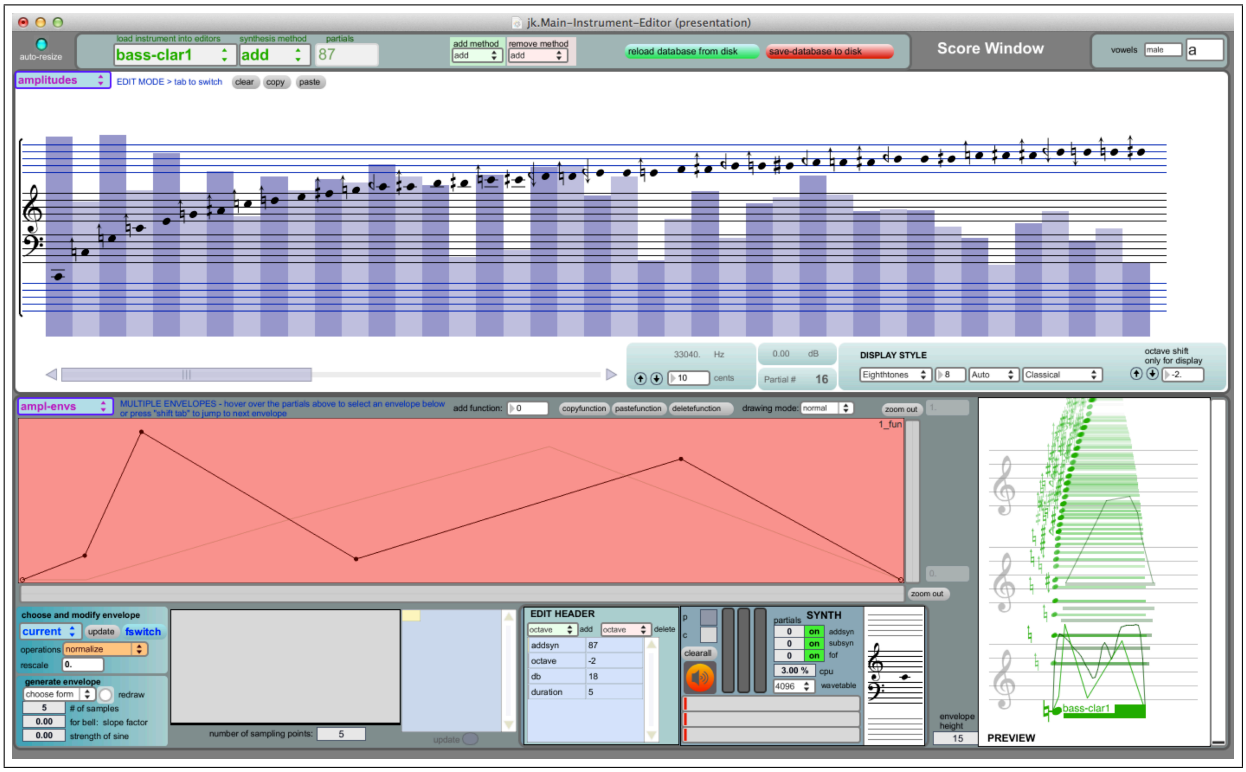

Figure 2. The Instrument Editor, where the base components and the envelopes of the instruments can be set.

an *Instrument Editor* (see Figure 2) and an *Analysis Tool* (see Figure 3). The Score Editor gives a timeline view allowing the arrangement and playing back of 'notes' (sound events) performed by KLANGPILOT instruments. In the Instrument Editor a KLANGPILOT instrument can be created/edited. These instruments can use either one or several of the supported synthesis methods simultaneously.

Figure 3. The Analysis Tool.

A KLANGPILOT instrument can also be obtained from analysis of any given sound file. Nevertheless it is recommended that short sounds (a few seconds) containing a single note or sound event are used (see Figure 3).

Data Representation and Interpolation

In order to define, edit and store synthesis parameters in a generalised way, we developed a data format shared by all objects of KLANGPILOT. The atomic element of our data representation is the *Parameter*, which may either have the form π_n or $p_n(t)$. Here, n denotes the *Channel Number* (the channel to which the actual parameter belongs to) and t is a *Timecode*: if the Parameter is part of a time-dependent envelope, the Timecode defines the temporal position of the Parameter within that envelope. Parameters in the form π_n are collected into *Static Data Sets* (*SDS*), while those in the form $p_n(t)$ are organised into *Dynamic Data Sets* (*DDS*).

Amplitudes, Frequencies and Durations of the different channels are all examples of Data Sets for different synthesis methods. We illustrate the difference between an SDS and a DDS through the following example: in additive synthesis, each oscillator has an instantaneous frequency value, changing over the time. We may define the instantaneous frequency on the channel n as the product of the (constant) base value ω_n and the (time-dependent) envelope $f_n(t)$. In this case, $\{\omega_n\} \equiv \Omega$ is an SDS while $\{f_n(t)\} \equiv F$ is a DDS.

The synthesizers expect a well-defined collection of SDSs and DDSs for their operation. Moreover, these sets must be dense. To understand what we mean by this, let us suppose that a synthesis method expects the SDSs $\Sigma^1, \Sigma^2 \ldots \Sigma^\sigma$ and the DDSs $S^1, S^2 \ldots S^s$. Let $n_{\max}$ denote the highest Channel Number occurring in any of the sets $\Sigma^1 \ldots \Sigma^\sigma, S^1 \ldots S^s$ and let T^i denote the set of every Timecode occurring in the set S^i. Then, the SDS Σ^i is said to be dense if it contains a valid Parameter for every possible Channel Number $n \in \{0 \ldots n_{\max}\}$, while the DDS S^i is said to be dense if it contains a valid Parameter for every possible Channel Number $n \in \{0 \ldots n_{\max}\}$ *and* Timecode $t \in T^i$. Roughly speaking, synthesizers expect every descriptor of every channel to be 'fully defined' in order to work.

As an example, a simple additive synthesis of 50 oscillators, each one having an amplitude and a frequency described by two constants and two separate envelopes (each of these having, for example, 9 breakpoint values), would require $50 \times 2 \times (1+9) = 1\,000$ parameters, which is beyond the 'human-readable'. To overcome this problem, we developed two separate interpolation methods for SDSs and DDSs, allowing the users to enter only a few key parameter values and let KLANGPILOT generate the rest.

For SDSs, the engine simply interpolates every missing Parameter based on the (sparse) set of values $\{\pi_n\}$ provided by the user. This interpolation may either be linear or exponential-like; however, if the lowest Channel Number (denoted ℓ) in the user-supplied Data Set is bigger than 0, then the Parameters of the lowest channels would be defined as $\pi_i = \pi_\ell \ (0 \leq i < \ell)$. The same applies if the highest user-provided Channel Number (denoted h) is smaller than $n_{\max}$, in which case $\pi_i = \pi_h \ (h < i \leq n_{\max})$ applies. The process is depicted in Figure 4.

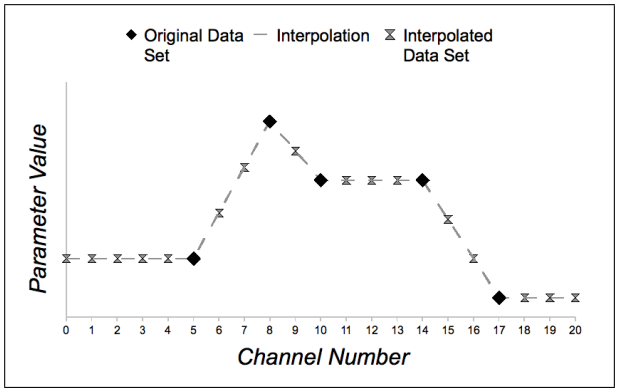

Figure 4. Interpolation of a SDS (using linear interpolation). The four black points indicate the user-defined 'sparse' data. The dashed line is the result of piecewise linear interpolation based on the original data set. The gray points show the 'dense' SDS that we get by evaluating the interpolated line at each Channel Number.

For DDSs, the interpolation is slightly more complex.

Firstly, we split the user-provided DDS (denoted as $\{p_n(t)\}$) into subsets according to the Channel Numbers of the Parameters: $P_i^{\mathrm{ChN}} = \{p_n(t)\}|_{n=i}$. This way we get the disjoint sets P_i^{ChN}, where each Parameter $p \in P_i^{\mathrm{ChN}}$ has a different Timecode (but the same Channel Number). Then, for each subset P_i^{ChN}, we interpolate the missing parameters for every $t \in T$, where T denotes the set of Timecodes appearing in the user-supplied DDS (formerly introduced as T^i for the specific DDS S^i). At this point, we always use (piecewise) linear interpolation.

Secondly, we take the DDS generated by the previous step

(denoted as $\{\tilde{p}_n(t)\}$) and split it again into subsets, this time according to the Timecodes: $P_\tau^{\mathrm{T}} = \{\tilde{p}_n(t)\}|_{t=\tau}$. Then, for each subset P_τ^{T}, we interpolate the missing parameters for every $n \in \{0 \ldots n_{\max}\}$. This interpolation may either be linear or exponential-like.

We may see the above method as a process that first computes the full envelopes of those channels which already have at least one user-defined point, and then generates the rest by interpolating between the full envelopes. We refer to Figure 5 for an illustration of the full process.

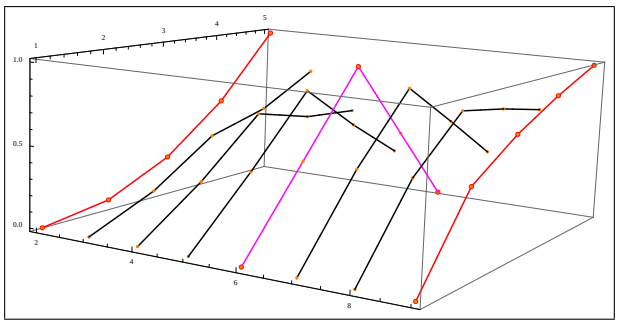

Figure 5. Interpolation of a DDS. The original data is shown with red dots. The red envelopes contain only data which was already specified by the user. We get the pink envelope after the first interpolation step, in which we interpolate two missing data points on that same envelope based on the three initial user-defined data points specified for that envelope. In the second interpolation step, we interpolate the rest of the envelopes, shown in black. The final results of the interpolation are depicted as orange points.

Synthesis Tools

In KLANGPILOT, synthesizers are defined as multi-channel devices. The channels of the same synthesizer — although running the same algorithm — operate independently from each other, with separate sets of parameters. These parameters are defined through SDSs and DDSs: if $\{\Sigma^\sigma\} \cup \{S^s\}$ denotes the set of every parameter, the parameters belonging to the i^{th} channel would be $\{\pi_i^\sigma\} \cup \{p_i^s(t)\}$, where $\pi_i^\sigma \in \Sigma^\sigma$ and $p_i^s(t) \in S^s$.

Regardless of the synthesis method, two SDSs are defined by every synthesizer: the *Offsets* and *Durations* of the channels. The former defines the time when the actual channel turns on, and the latter defines the time frame until which the channel is active. To achieve this, the Timecodes belonging to the i^{th} channel of each DDS are scaled linearly to extend between the Offset and the sum of the Offset and the Duration of the i^{th} channel. Furthermore, the synthesizers normalise internally every time value to a phase between 0 and 1 (so that the biggest $\mathrm{Offset} + \mathrm{Duration}$ value would be normalised to a phase of 1).

Synthesizers may have an overall duration, in which case they would play a sound with the given duration, from beginning to end, when triggered. However, they have an (optional) phase input as well. If this latter is being used, the synth would jump to the specified phase and 'freeze' until a new phase value is received; when 'frozen', the instantaneous values of parameters defined by DDSs are kept constant, which allows the synthesizer to 'freeze' the timbre.

Three synthesis methods have already been implemented using the above principles:

Additive. An oscillator bank; channels correspond to oscillators.

Subtractive. A filter bank; channels correspond to biquadratic band-pass filters.

Additive formant. A method inspired by the CHANT synthesizer (Rodet, Potard, & Barrière, 1984); channels correspond to formant wave functions (*fonction d'onde formantique*, *FOF*).

Table 1 presents the core descriptors (except for Offsets and Durations) of the above methods.

The core parameters of the synthesizers may be modulated in two different ways: on the one hand, with a sinewave oscillator (which we call 'modulation') and, on the other, with band-limited noise (which we denote as 'jitter'). The (sinusoidal) modulation may be described with an amplitude and a frequency value, whereas the jitter is parametrized with its amplitude and bandwidth[3]. Each of these modulation parameters are described by separate DDSs. However, in contrast to core values, modulation parameters may be omitted, in which case the synthesizers would automatically turn modulation off.

One may see that the maximum number of Data Sets describing the synthesis methods of KLANGPILOT could be quite high: additive synthesis may use up to 14, while subtractive and formant syntheses up to 20 Data Sets, although not every possible combination of modulations and core values has been implemented due to efficiency reasons (e.g. modulation of the Q–factors of subtractive synthesis is not yet possible). Table 2 lists all parameters of additive synthesis; the other two cases are very similar.

Analysis Engine

The purpose of our analysis engine is to convert an incoming signal into a set of SDSs and DDSs which can be understood by the synthesizers. Our model follows the approach of Spectral Modeling Synthesis (Serra, 1997), where the signal is represented as a sum of sinusoids and noise. The analysis comprises the following steps:

1. Decomposition of the signal, using Short-Term Fourier Transform (STFT).

	Channels correspond to…	Core Parameters
Additive	…(sinusoidal) oscillators	Oscillators' Amplitude & Frequency
Subtractive	…biquadratic band-pass filters	Bands' Gain, Centre Frequency & Q–Factor
Formant	…FOF generators	Formants' Amplitude, Centre Frequency & Bandwidth

Table 1. *Most important parameters per synthesis channel for the different synthesis methods. Each of the above parameters is obtained as the product of a constant base value and a time-dependent envelope. The formers are derived from SDSs while the latters from DDSs.*

SDS	DDS
	Amplitudes*
	Frequencies*
	Modulator Amplitudes (AM)
Offsets*	Modulator Frequencies (AM)
Durations*	Modulator Amplitudes (FM)
Amplitudes*	Modulator Frequencies (FM)
Frequencies*	Jitter Amplitudes (AJ)
	Jitter Frequencies (AJ)
	Jitter Amplitudes (FJ)
	Jitter Frequencies (FJ)

Table 2. *A list of every allowed SDS and DDS for additive synthesis. 'A' stands for Amplitude, 'F' stands for Frequency and 'J' stands for Jitter. Starred Data Sets are mandatory.*

2. Partition of the Fourier-components: we identiy the *Tonal Peaks* and the *Noise Bands* within the result of the previous step and isolate them from the rest of the data. Tonal Peaks are the spectral components describing pure sine waves with high likelihood. Noise Bands are the 'flat regions' of the spectra which can be interpreted as white noise filtered by single biquadratic band-pass filters.
3. Data aggregation: Tonal Peaks are organised into envelopes — each envelope describing the time-dependent parameters of a single sinusoidal oscillator — by means of partial tracking. Our partial tracking method approaches the problem by distinguishing between the short-time and long-time behavoiur of a partial: firstly, it creates short-time 'envelope chunks' and secondly, these chunks are merged into long-time envelopes. The same procedure is applied to the Noise Bands[4], although these envelopes describe filters' parameters instead of oscillators'.
4. Envelope reduction: the envelopes are organised into 'dense' DDSs, containing every breakpoint value obtained in the previous step. Then, we reduce the number of actual Parameters contained by the DDSs by means of piecewise linear regression.

The algorithm that finds the Tonal Peaks as well as our two-step partial tracking method was presented in (Siska, 2012). In the rest of this section, we concentrate on the last step of our analysis engine, that is, Parameter reduction.

Normally, a DDS obtained after the 3rd step contains much more information than what we consider 'human-readable' — the number of Parameters in such a DDS is normally well over a thousand! However, much of this information can be eliminated by removing those Parameters which can be reconstructed by our interpolation engine. Note that this is a lossy compression of the data, as the Parameters interpolated by our interpolation tool will differ a little bit from the originals in most cases; this is the price that we need to pay in order to efficiently reduce our Data Sets to a 'human-readable' size.

The reduction of a DDS happens in two steps, which act as if they were the inverses of the steps involved in the DDS interpolation method, presented in Section . For this algorithm, the user needs to supply an error percentage, describing the maximum allowed deviation between a Parameter obtained from analysis and the one reconstructed by interpolation:

1. We split the DDS generated by the analysis (denoted as $\{p_n(t)\}$) into subsets according to the Channel Numbers of the Parameters: $P_i^{\mathrm{ChN}} = \{p_n(t)\}|_{n=i}$. Then, we apply the following algorithm, starting with $i = 0$:
   (a) Let $j = i + 1$, $\Lambda^{\min} = \{\forall t \in T : \lambda^{\min}(t) = -\infty\}$ and $\Lambda^{\max} = \{\forall t \in T : \lambda^{\max}(t) = \infty\}$. Here, T denotes the set of Timecodes appearing in the DDS generated by the analysis.
   (b) We compute, for every value $t \in T$, the estimated Parameter subset $\tilde{P}_{j+1}^{\mathrm{ChN}}$ using linear extrapolation, based on the respective Parameters of P_i^{ChN} and P_j^{ChN}.
   (c) We compute the allowed minimum and maximum deviations (based on the user-defined error percentage) for every Parameter in $\tilde{P}_{j+1}^{\mathrm{ChN}}$ (we denote these limits $\tilde{p}_{j+1}^{\min}(t)$ and $\tilde{p}_{j+1}^{\max}(t)$)
   (d) For every $t \in T$, we compute the values $\tilde{\lambda}_{i,j+1}^{\min}(t)$ and $\tilde{\lambda}_{i,j+1}^{\max}(t)$, which are the inclinations of the lines defined by $\left(p_i(t), \tilde{p}_{j+1}^{\min}(t)\right)$ and $\left(p_i(t), \tilde{p}_{j+1}^{\max}(t)\right)$,

respectively.

(e) We update the sets $\Lambda^{\min}$ and $\Lambda^{\max}$ according to $\lambda^{\min}_{\text{new}}(t) = \max\left(\lambda^{\min}_{\text{old}}(t), \tilde{\lambda}^{\min}_{i,j+1}(t)\right)$ and $\lambda^{\max}_{\text{new}}(t) = \min\left(\lambda^{\max}_{\text{old}}(t), \tilde{\lambda}^{\max}_{i,j+1}(t)\right)$ for every $t \in T$.

(f) If $\lambda^{\min}(t) \leq \lambda_{i,j+1}(t) \leq \lambda^{\max}(t)$ holds for each $t \in T$ — where $\lambda_{i,j+1}(t)$ denotes the inclination of the line connecting $p_{j+1}(t)$ and $p_i(t)$ —, we remove P^{ChN}_j from the DDS and increase j by 1.

(g) Otherwise, we set i to the current value of j and start over.

We repeat these steps as long as we don't reach the highest Channel Number, with some additional considerations on the boundaries (these are $i = 0$ and $j = n_{\max}$).

2. We execute a similar reduction algorithm on each of the remaining sets P^{ChN}_i independently, always starting from $t = \inf(T)$:

(a) Let $\tau = t'$, where t' is the *successor* of t within the set T, $\lambda^{\min} = -\infty$ and $\lambda^{\max} = \infty$.

(b) We compute the estimated Parameter $\tilde{p}_i(\tau')$ using linear extrapolation, based on $p_i(t)$ and $p_i(\tau)$.

(c) We compute the allowed minimum and maximum deviations (based on the user-defined error percentage), denoted $\tilde{p}^{\min}_i(\tau')$ and $\tilde{p}^{\max}_i(\tau')$. Then, we calculate the inclinations of the lines connecting these values with $p_i(t)$.

(d) If the computed minimal and maximal inclinations are bigger or smaller than $\lambda^{\min}$ or $\lambda^{\max}$, we substitute these with the new values, respectively.

(e) If the inclination of the line connecting $p_i(t)$ and $p_i(\tau')$ lies within the range $[\lambda^{\min}, \lambda^{\max}]$, we remove $p_i(\tau)$ from P^{ChN}_i and set τ to τ'.

(f) Otherwise, we set t to the current value of τ and start over.

We repeat these steps (for each remaining set P^{ChN}_i) as long as we don't reach the highest Timecode, with some additional considerations on the boundaries (these are $t = \inf(T)$ and $\tau = \sup(T)$).

Figure 6 depicts how the allowed minimum and maximum inclinations are computed for a specific reference point.

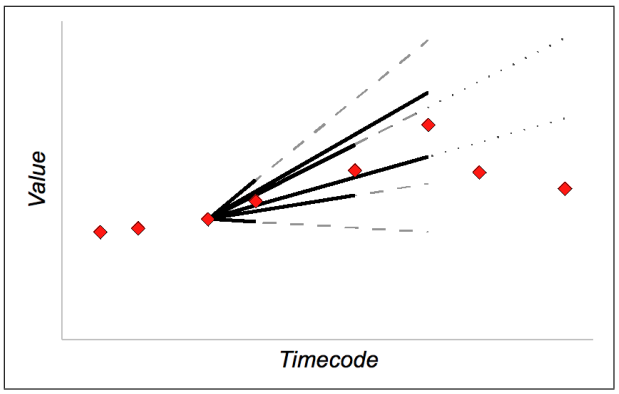

Figure 6. Finding the points that fit into the same line segment. Red dots indicate the original values. The current 'reference point' is the 3rd dot from the left. The black lines indicate the computed minimum and maximum inclinations for each subsequent data. The dotted lines indicate the inclinations defined by $\lambda^{\min}$ and $\lambda^{\max}$. As we can see, the 4th, 5th and 6th would fit on the same line.

We could summarize the above procedure as follows. Firstly, we remove those full-envelopes from the analysis results which can be fully interpolated by the neightbouring envelopes. Secondly, we take the remaining envelopes and remove all those data points which can be interpolated by the neighbouring data points. At the end, we get a sparse Data Set which only contains the Parameters that are crucial in order to reproduce the original information within the error constraing given by the user.

User Interface

The graphical paradigm of the KLANGPILOT score language can be seen as an extension of the classical piano roll (see Figure 1). Unlike MIDI files and normal piano roll representation KLANGPILOT also supports:

- Microtones — graphically represented at the maximum precision of eight notes, internally stored as floating point numbers allowing almost arbitrary precision.
- Polyphonic microtonal glissandi.
- The size of the note head and the thickness of the beam representing the duration of the note give an impression of the loudness at the beginning and the end of each note.
- The color of events can be used to indicate different instruments.
- Labels showing the instrument's name above each note can be activated.
- If instruments are contained in the KLANGPILOT instrument database, their spectrum and their envelopes can be displayed.

Rhythm is represented as position of the notes in the x-axis. A user defined grid can be used to quantify time into beats and subdivisions of beats in a given tempo for entering metric music.

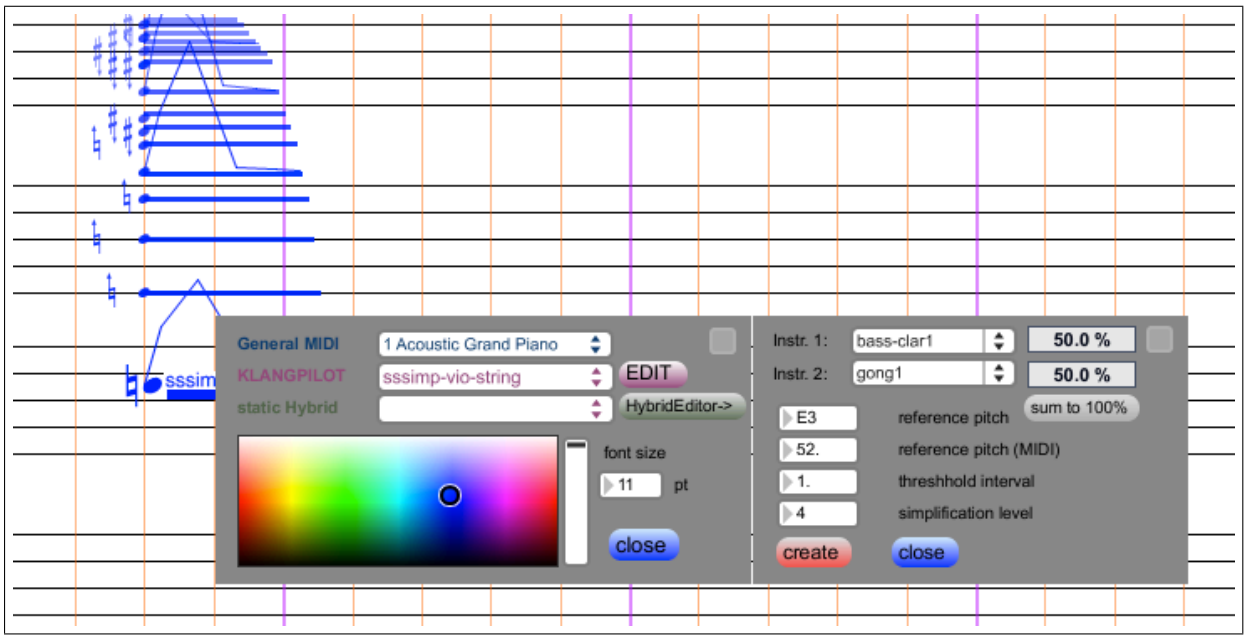

Figure 7. An instrument whose spectral properties were derived by interpolating two existing instruments in the KLANGPILOT instrument database.

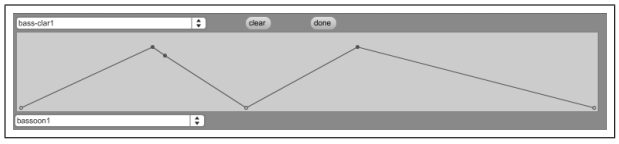

Figure 8. Dynamic interpolation between the two instruments. Horizontal position denotes time while the vertical one sets the ratio between the two instruments.

In general, the design philosophy of the KLANGPILOT score language is that the display of certain information such as spectrum, envelopes, instrument names etc. is optional and can be enabled/disabled. The user can decide about the complexity with which the musical information is represented at a given moment (see Figure 1).

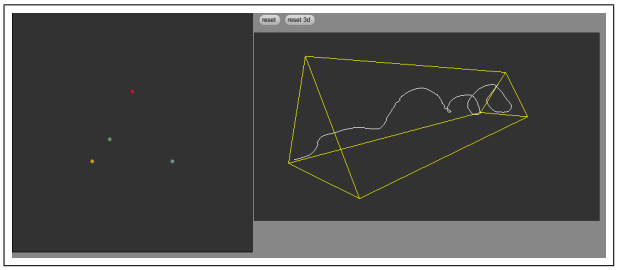

Figure 9. Dynamic interpolation between multiple instrument timbres.

The instrument editor is designed to give access to all parameters of sounds synthesis, while making a big effort to reduce the complexity of representation to a minimum (see Figure 2)

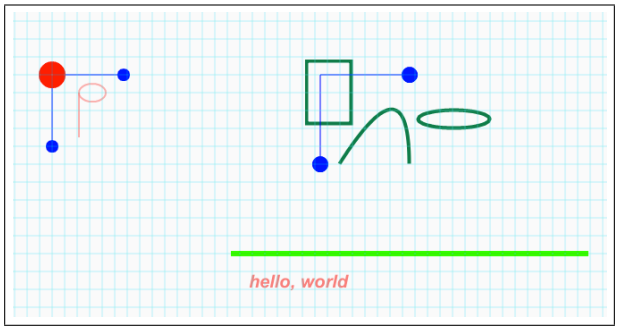

Figure 10. Example drawings on the new canvas.

Currently a hybrid editor is under development, which will allow the merging of two or three instruments statically or dynamically (see Figures 7–9).

Future Work

At the moment we are developing a new canvas object as a replacement of Max/MSP's LCD object for graphical representation and user interaction (see Figure 10). It directly supports the touching, moving, and resizing of graphical objects in the canvas without the need of recalculating them in the frame of Max's message objects. In addition, new algorithms for efficient sound analysis and reduction of complexity of the data for synthsis/artistic editing are explored.

References

Agon, C., Stroppa, M., & Assayag, G. (2000, Aug.). High level musical control of sound synthesis in openmusic. In *Proceedings of the international computer music conference.* Berlin, Germany.

Bennett, G. (1996). Music since 1945 and oral culture. In H. Dufourt & J.-M. Fauquet (Eds.), *La musique depuis 1945. matériau, esthétique et perception.* Belgium: Mardaga.

Grey, J. M., & Moorer, J. A. (1978). Perceptual evalutations of synthesized musical instrument tones. *Journal of the Acoustical Society of America*, *62*, 454–462.

Harvey, J. (1981, Winter). Mortuos plango, vivos voco: A realization at ircam. *Computer Music Journal*, *5*(4), 22–24.

Karkoschka, E. (1972). *Notation of new music*. London/New York: Universal Edition. (Original: Das Schriftbild der Neuen Musik)

Kretz, J. (1999). Klangpilot – a software environment for control and composing synthetic sounds. In H. G. Feichtinger & M. Dörfler (Eds.), *Computational and mathematical methods in music* (Vol. 133, pp. 255–266). Vienna: Österreichischen Computer Gesellschaft.

Kretz, J. (2002). Composing sounds – developing tools for refined control of timbre in music. In G. Johannsen & G. De Poli (Eds.), *Human supervision and control in engineering and music* (Vol. 62, pp. 454–462). Vienna.

Laurson, M., & Duthen, J. (1989, Nov.). Patchwork: A graphical language in preform. In *Proceedings of the international computer music conference* (pp. 172–175). Columbus, USA.

Ligeti, G., & Wehinger, R. (1958). *Articulation – eine farbige hörpartitur.* Schott Music. (Reprint of original edition)

Machover, T. (Ed.). (1984). *Musical thought at ircam* (Vol. 1) (No. 1). Harwood Academic.

McAdams, S. (1982). Spectral fusion and the creation of auditory images. In M. Clynes (Ed.), *Music, mind and brain: The neuropsychology of music.* New York: Plenum Press.

McAdams, S. (1989). Psychological constraints on form-bearing dimensions in music. *Contemporary Music Review*, *4*, 181–198.

Rodet, X., Potard, Y., & Barrière, J.-B. (1984, Fall). The CHANT–Project: From the Synthesis of the Singing Voice to Synthesis in General. *Computer Music Journal*, *8*(3), 15–31.

Serra, X. (1997). Musical sound modeling with sinusoids plus noise. In C. Roads, S. T. Pope, A. Picialli, & G. De Poli (Eds.), *Musical signal processing* (pp. 91–122). Swets & Zeitlinger.

Siska, Á. (2012, Sep.). Partial tracking in two steps. In *Proceedings of the international computer music conference* (pp. 559–562). Ljubljana, Slovenia.

Vandenheede, J. (1991). *Jonathan harvey's ritual melodies.* (Manuscript. Copy available from the authors upon request.)

Vandenheede, J., & Harvey, J. (1985, Aug.). Identity and ambiguity. the construction and use of timbral transitions and hybrids. In *Proceedings of the international computer music conference* (pp. 97–102). Vancouver, Canada.

---

[1] `http://www.ina-entreprise.com/entreprise/activites/recherches-musicales/acousmographe.html`

[2] See the 'Downloads' section at `www.mdw.ac.at/zimt`.

[3] For jitter, we use the terms 'frequency' and 'bandwidth' interchangeably.

[4] This feature is not integrated into the publicly available version of KLANGPILOT yet.