EarGram: an Application for Interactive Exploration of Large Databases of Audio Snippets for Creative Purposes

Gilberto Bernardes[1], Carlos Guedes[1], and Bruce Pennycook[2]

[1] Faculty of Engineering of the University of Porto, Portugal
{g.bernardes, cguedes}@fe.up.pt
[2] University of Texas at Austin, USA
bpennycook@mail.utexas.edu

Abstract. This paper outlines the creative and technical considerations behind earGram, an application built as a Pure Data patch for real-time concatenative sound synthesis. The system encompasses four generative strategies that automatically re-arrange and explore a database of descriptor-analyzed sound snippets (corpus) by rules other than its original temporal order into musically coherent outputs. Of notice are the system's machine-learning capabilities that reveal musical patterns and temporal organizations, as well as several visualization tools that assist the user in making decisions during performance.

Keywords: Concatenative sound synthesis, recombination, and generative music.

1 Introduction

Composing music using audio samples can become a very laborious task. Current solutions that usually involve the use of a music sequencer demand a considerable amount of time to segment and assemble a collection of samples together. During the last decade, a technique called concatenative sound synthesis (CSS) [1] eases the process of synthesizing new sounds based on preexisting audio samples that was extremely difficult and time-consuming when drawn by hand. Concisely, CSS uses a large database of segmented and descriptor-analyzed sound snippets to assemble a target phrase according to a proximity measure in the descriptors space. It was originally intended for text-to-speech synthesis [2], but, later, it was introduced to several other fields that use sound synthesis techniques. CSS is beginning to find its way in musical composition and performance since 2000 [3, 4]. However, the vast majority of literature about this technique still focuses on solving technical problems that enhance the efficiency of these systems, paying very little attention to its musical applications.

The application reported here, i.e. earGram, is a Pure Data (PD) patch that implements a CSS engine and several exploratory tools for musical creative practices. earGram automatically re-arranges sound snippets and permit rapid prototyping of interactive music systems. Particular attention was given to the definition of target phrases to be synthesized, by designing GUIs that allows the user to specify targets quickly and intuitively. Four methods to recombine automatically the units are

9th International Symposium on Computer Music Modelling and Retrieval (CMMR 2012)
19-22 June 2012, Queen Mary University of London
All rights remain with the authors.

proposed. They approach two different generative music strategies. The first uses the corpus to synthesize targets defined by an imposed metric and harmonic templates selected beforehand by the user. The second creates a novel music output while retaining the time-varying acoustic morphologies of the audio source(s). Of particular interest is the system's ability to cluster units into representative groups (sub-corpus). The user can control the system in real-time and adjust several structural elements of the output such as the meter, the key, the number of voices, and tempo.

Compared to other CSS software implementations, earGram has three more algorithms related to visualization of the corpus. The common visualization tools that CSS software implementations offer, such as 2d-plots and similarity matrices, depict the distribution and relation amongst units. Eargram offers two more methods that have never been used on the music field and focus on the visualization of high-dimensional data, such as the feature vectors that represent the units, and a third one that aims at depicting the long-term structure of the audio sources(s). They are respectively parallel coordinates [5], star coordinates [6], and arc diagram [7].

Our approach to CSS is inspired on T. Jehan's Skeleton [8] and D. Schwarz's cataRT [9]. The architecture and the conceptual approach of the two systems was our fundamental basis. The analysis-synthesis models presented by Jehan [8] and implemented in Skeleton, especially the perceptual and structural modeling of the music surface, was of seminal importance for the development of the machine listening and learning in earGram. Schwarz's cataRT was equally important due to the similarities of the programing environment used, and its real-time capabilities. earGram extends previous research on generative strategies that recombine descriptor-analyzed units into coherent musical outputs suitable for both studio and live experimentations. In addition, the system offers visual representations of the corpus that were never used in CSS software.

2 System Design

In this section we provide an overview of the design scheme of earGram that is also shown in figure 1.

The user must first feed the system with audio data, either from a live audio input or from pre-recorded material. The first and left-most block in figure 1 (analysis) is responsible for 3 tasks: (1) segmenting the audio material, (2) reducing the content of each unit to a feature vector, and (3) model the harmonic, timbre and metrical structures of the audio source(s) over time. At this stage, a list of pointers to audio segments and their respective feature vector are stored in a database. Subsequently, the system groups the units using one of the available clustering algorithm, and displays the result on a 2d-plot on the main interface of the system (see figure 2).

After importing the audio and analyzing it, the user must choose a generative method for the performance. The available generative methods are responsible for defining a target phrase and retrieving the units that best match this target. At runtime, a signal-processing block enhances the concatenation and emphasizes the artistic expression (right-most bock on figure 1). Among the available audio processing techniques are adaptive filtering, reverberation, chorus, and spectral shift (see sections 7 and 8 for a detailed description).

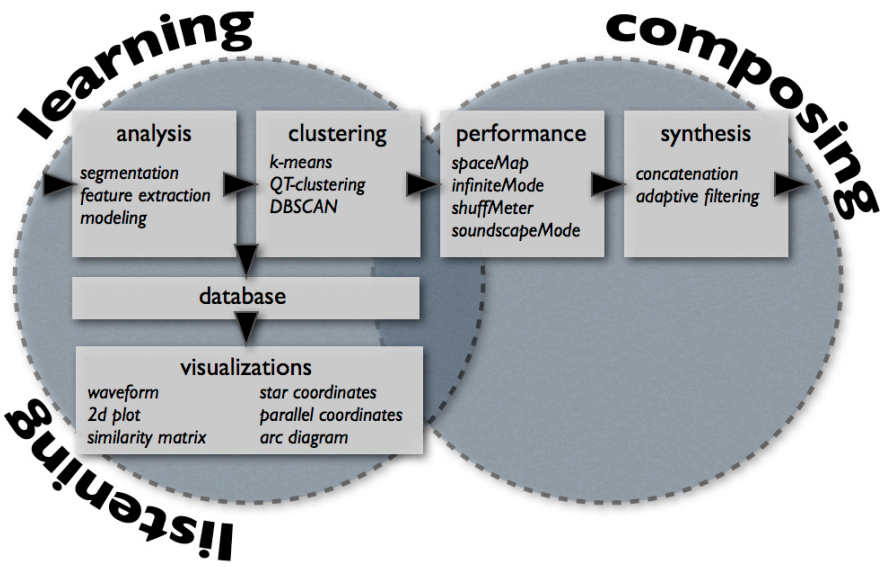

Fig. 1. Design scheme of earGram.

3 Initialization

Initially the user must either create a new project or open a previously saved one, and specify the audio source(s) that will feed the system. There are three options available: (1) single audio track; (2) multiple audio files in a folder; or (3) a live input signal.

The functionality and interface of the proposed system was designed so that musicians that aren't familiarized with MIR research or technology can easily generate some consistent musical results. By default, the system assumes an automatic configuration that needs little to no fine-tuning. However, most settings can be configured via the preferences panel reachable through the main interface. In the following sections, we will describe the system in detail pointing out the differences between the auto-assigned and user-defined settings.

4 Analysis

The analysis block is responsible for three tasks: (1) **segmenting** the audio into units according to a predefined method, (2) defining a **feature vector** that characterizes

each unit, and (3) and **modeling** the units' structure over time regarding harmony, timbre and meter.

In order to **segment** the audio samples using the default settings, the system will inspect the audio input for peaks with harmonic relationships on the spectral flux auto-correlation function to define a regular pulse and segment the audio accordingly. If no such peaks are found, the system will segment the source(s) on every detected *onset*. The user can also assign the segmentation mode manually overwriting the default configuration. Besides the *beat*, and *onset* segmentation methods, there are more methods available: the *uniform* method that segments the audio uniformly based on the length of the window size specified in the system preferences; and the *pitch* method that segments the audio based on the presence of different fundamental frequencies (to be used on monophonic sound sources only). The beat-tracker algorithm used is largely based on S. Dixon [10], and the onset segmentation algorithm is based on P. Brossier [11].

The second purpose of the analysis block is to create for each unit a **feature vector** that represents it. We rely on the *timbreID* library developed by W. Brent [12] for PD to describe the low-level spectral qualities of each unit. We chose this library for its robustness, efficiency, and ability to work in both real time and non-real time. It implements a vast collection of low-level spectral audio descriptors available, such as bark, bfcc, cepstrum, centroid, kurtosis, flatness, flux, irregularity, mfcc, rolloff, skewness, spread, and zero-crossing rate. Two additional low-level features, loudness and fundamental frequency, are extracted by *sigmund~* a PD built-in object created by M. Puckette. Additionally, we built some PD abstractions that define some mid-level features of the input signal, such as tempo, meter, harmonic progressions, and key. Based on this set of descriptors, we represent the units in two ways: (1) static, i.e. constant over the length of the units or (2) dynamic, i.e. varying over the length of the unit.

In the third part of the analysis, the system also creates statistical **models** that represent the harmonic and timbre temporal evolution of the audio source(s). For both music characteristics (harmony and timbre) a transition probability table is created that represents the probability of going from unit *i* to unit *i+1*. The set of all states and transition probabilities completely characterizes a Markov chain, which later allows the generation of new sequences based on stochastic processes. In order to create a transition probability table for harmony and timbre we needed to classify each unit into a finite number of predefined classes. The unit's harmonic content is characterized by the pitch class profile (0-11) of the fundamental bass. The timbre is characterized by a single integer that represents the 3 highest bark spectrum bins, out of a total of 24 bins. Initially, the 3 highest bins are ordered from the lowest to the highest and converted into binary representation. Then the second and the third bins numbers are shifted left by 5 and 10 cases respectively. The three numbers are re-converted to decimal and summed.

Finally, if the input signal was segmented on a beat basis, we build a template that represents the distribution of the units' noisiness for the length of a measure. Zero-crossing rate is a good indicator of the signal noisiness. Very high values denote a very noisy signal while speech or music signals tend to have a very low value. Given the estimated meter with *n* beats per measure, we built a template with *n* bins that represents the noisiness of each beat within a measure. We fill the template by finding the mean value of the zero-crossing rate of all units labeled with a particular pulse,

and repeat the operation for all pulses within the measure. At last, the template is normalized to the range 0-1.

4 Database

A database is created to store the data produced during analysis. It is implemented in PD as a collection of arrays. Each individual array stores the data correspondent to a particular feature for all units in the corpus.

The database and the variables used for the analysis of the sources(s) can be saved as a text file and loaded later, in order to not repeat the time consuming tasks of the database construction, especially if we are dealing with hundreds or thousands of units.

5 Clustering

Clustering aims at grouping similar segments together to form collections of units whose centroid or representative characterizes the group, revealing musical patterns and a certain organization of sounds in time that can be applied in various manners during performance. The current implementation comprises three non-hierarchical clustering algorithms: *k*-means, quality-threshold clustering (QT-clustering), and DBSCAN. We chose this set of algorithms because we considered that they form a good collection to explore the database. If the user wants to have a concise number of clusters defined a priori and consider all units in the corpus, in order to create sub-corpus for different layers or sections, the choice should fall on *k*-means. On the other hand, if the user wants to define the quality of the clusters based on threshold of similarity or neighborhood proximity between units, he/she should choose either QT-clustering or DBSCAN, respectively. The distance metric used to calculate the similarity amongst units in all clustering methods is the Euclidian distance. Even if the clustering algorithms implemented in earGram can deal with arbitrary long vectors, to convey a clearer and more understandable visualization in two dimensions, the algorithms process only two-dimensional vectors selected from the available bag of descriptors.

K-means is one of most popular clustering algorithms available. It partitions the corpus into clusters by allocating each unit to the cluster with the nearest centroid. The total number of clusters *k* needs to be defined a priori. However, the *k*-means implementation in earGram suggests to the user the optimum number of clusters using a technique known as ‘elbow method’. Our implementation of the technique follows two steps. First, we calculate the distortion, i.e. sum of the squared distances between each unit and its allocated centroid for each different value of *k*, ranging from 2 to 9 clusters. Second, we assign the parameter *k* to the number of clusters that doesn't give much better modeling of the data according to a threshold.

QT-clustering was developed by L. Heyer, S. Kruglyak, and S. Yooseph [13] to cluster gene expression patterns. Quality is defined by the cluster diameter and the minimum number of units contained in each cluster. The two parameters are assigned

initially by the user. However, the user does not need to define the number of clusters. All possible clusters are considered: a candidate cluster is generated with respect to every unit and tested in order of size against the quality criteria. In addition, it points out the outliers that should be treated differently (notably excluded) at runtime.

DBSCAN defines the clusters based on the neighborhood proximity and the density of the units in a cluster. Our implementation follows the algorithm described in [14] by M. Ester, H. Kriegel, J. Sander, and X. Xu. The user must define initially two parameters. They are respectively the neighborhood proximity threshold and the minimum density within the radius of each unit. Similarly to the QT-clustering algorithm, DBSCAN avoids defining a priori the number of clusters. However, the algorithm finds arbitrarily shaped clusters very diverse from the ones found by the QT-clustering. It can even find clusters surrounded by (but not connected to) a different cluster.

6 Visualization

Given the huge amount of information that the software produces during analysis, we appended a visualization block to communicate clearly and effectively the information concerning the audio source(s). It is aimed to assist the decision-making during the performance. Most of the visualization tools are interactive and besides displaying the information to the user, they assist him in defining regions of the source material that they want to work with based on structural similarities. The implemented tools and algorithms for data visualization focus on different musical hierarchical levels that demonstrate structural properties of different types. We can roughly divide them in four categories organized from the lowest level, which consists of continuously variable expressive properties to the top levels, which encompass discrete canonical properties: (1) the waveform display is one of most common visualizations tools for audio data, and it provides the user with a general overview of the source(s)' content, and its segmentation; (2) the similarity matrix and the arc diagram [7] aims at presenting the long-term structure of the corpus; (3) the 2d-plots and star coordinates [6] reveal a concise representation of the units based on various combinations of descriptors; and (4) parallel coordinates [5] examines the high-dimension descriptors space.

The waveform plot helps the user to identify and browse through the segmented units.

The self-similarity matrix and arc diagram displays give the user a better understanding of the long-term structure of the audio data by finding similar patterns along the source(s). They depict pairwise similarity between the units of the corpus. The user can group and select different sections on each representation that are treated as different layers during the performance. Non-uniform units require special attention when being compared, because most audio features aren't insensitive to the length of the unit. Thus, a high variability in the unit's length can lead to misleading results on both self-similarity matrix and arc diagram.

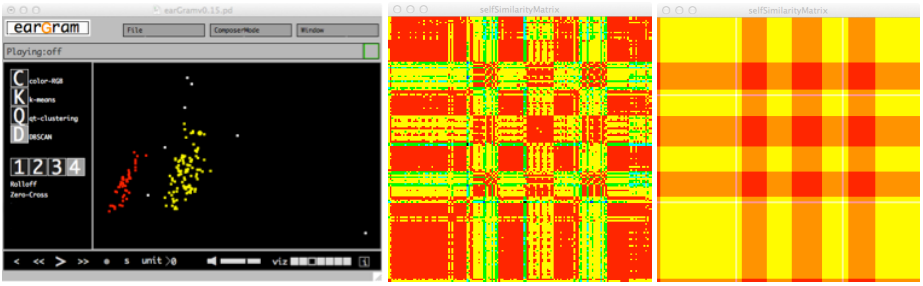

Fig. 2. Different visualizations of a single-track source corpus – *4* by Aphex Twix. From left to right: earGram interface depicting the corpus clustered by a DBSCAN algorithm on a 2d-plot (using spectral rolloff and zero-crossing rate as variables). Middle and right images are self-similarity matrices plotting the same corpus. The middle image depicts the similarity using all available descriptors, and the right most uses the color scheme gathered from the cluster representation on the interface.

The 2d-plot is one of most common visualizations adopted by CSS software. It is especially suitable for navigating and exploring the corpus intuitively. Similar units are plotted together, and its representation along the axis reveals characteristics related to the axis' variable. Additionally, another layer of information concerning the units' color is also available. The color of each unit is defined by a list with three elements that correspond to the red, the green, and the blue values of an additive (RGB) color model. The values that compound the list that defines the units' color are chosen from the available audio descriptors. Star coordinates is a dimensionality reduction algorithm presented by E. Kandogan [6]. It maps high-dimensional data linearly to 2d or 3d using their vector sum. Here we used this algorithm to represent multiple features on a 2d representation. We chose this algorithm for its understandability (each dimension still preserves the same meaning), contrary to approaches such as multidimensional scaling or principal component analysis. One disadvantage of star coordinates is the need to explore the representation by weighing the variables and assigning the axis to different angles to find interesting patterns. Parallel coordinates [5] is barely used in the music domain but is a known procedure to visualize high-dimensional data and analyze multivariate data. By default, all descriptors are taken in account to create the projection, although the user can select the features he/she wants to include.

Figure 2 depicts three representations of the same corpus that comprises a singe audio track – *4* by Aphex Twin. The structure of the song is clarified by the matrix in the rightmost image, which represents the units by the colors resulting from the DBSCAN clusters (leftmost image). We can clearly notice that the song comprises two sections that alternate throughout the track.

7 Performance

The main drive behind the analysis is primarily synthesis. On the following sections, we present four methods that re-arrange in a structured and musical meaningful way

the collection of units that form the corpus based on the analysis described in the previous sections. The recombination processes cover the generation of three specific music results: (1) sonic textures / soundscapes (space-Map and soundscapeMode) either by browsing the navigable 2d visualizations or by defining targets according to audio qualities, (2) extending indeterminately the length of a particular audio sample avoiding repetitions (infiniteMode), and (3) defining targets that reflect a particular meter (shuffMeter).

The methods described bellow are both responsible for defining the target phrases and selecting the units that best matches the target queries.

7.1 SpaceMap

This method is meant to function as a tool that allows intuitive and interactive exploration of the units on the 2d-visual representation. It can be seen as an extended granular synthesis engine where grains are organized in a meaningful visual representation. It aims at creating sonic textures with controllable nuances. It is a very powerful method when playing along with a live input source particularly when improvising, because besides the automatic and meaningful segmentation that the software produces, after a segment is defined it is consequently plotted in the interface, creating an almost instantaneous representation of the input signal during performance.

It has three playing modes: (1) mouseOver – continuously maps the mouse position on screen to the granulator's parameters; (2) pointerClick – the same effect as mode 1, but only when the mouse button is pressed a unit is played; and (3) colorPicker – selects units based on their RGB color values that are retrieved from a navigable grid of colors.

Several parameters can be changed during performance and affect each unit separately, such as gain, density of events, pitch deviations, and panning. All parameters can have a certain degree of random variability. The software also allows the creation of several bus-channels that may incorporate audio effects. At runtime, the representation of the units in the interface can be changed without affecting the synthesis, except when a live input source is fed to the system.

7.2 InfiniteMode

The second synthesis method implemented in earGram aims at generating a musical result of indeterminate length that doesn't repeat, while it retains the time-varying acoustic morphologies of the audio source(s). It is primarily suitable for single-track audio input, or for groups of sound files that share commonalities at the metrical, harmonic and timbre level.

Each new unit is triggered and defined at the end of the previous one and defined at that stage based on the harmonic, metrical, timbre and noisiness models created beforehand during analysis. The interface allows us to select and use up to three sets of characteristics that will be responsible for defining the target. On the interface, two sets of characteristics are predefined: one for soundscapes (timbre) and a second for polyphonic music (meter, harmony and timbre).

At runtime, the algorithm selects a new unit to synthesize by finding all units that both satisfy the assigned group of characteristics and that best matches the spectrum of the previous unit, i.e. when a new unit is triggered, the algorithm examines all selected characteristics, and for each of them defines a group of units that match the query. Then, it finds the units that are common to all groups of characteristics, and finally, from the remaining units is selected the one that minimizes the distance on the bark spectrum representation to the previous selected unit. If the algorithm doesn't find any unit that satisfies all the assigned characteristics the algorithm will ignore sequentially characteristics until finding candidates by the contrary order of the interface. If we have three selected characteristics, and any unit is found for a specific query, the algorithm eliminates the third characteristic and examines again the number of remaining units, if nothing is retrieved it eliminates then the second and so on.

Harmony and timbre aims at preserving the temporal evolution of chord progressions and audio spectra from the original source(s). At every new query a group of units is outputted for each element, according to the transition probability table elaborated during analysis.

To preserve the metrical accents' distribution over the length of a bar during synthesis, the algorithm retrieves for each metrical accent the units that were previously labeled accordingly during analysis. In other words, initially, while at the original temporal units order, every unit is labeled with their respective position over the length of a bar, in a sequence that goes from 1 to number of units per bar. E.g., assuming we got a time signature with four units every bar, we would split the source in groups of four units, and label each sequentially. At runtime, for each new metrical accent, the algorithm retrieves all units that were labeled with that metrical accent.

The noisiness characteristic attempts to replicate the zero-crossing rate configuration over the length of a measure that was encoded in a template elaborated during analysis. At each query, successive values are retrieved from the template and the algorithm looks at the database to find units that present a similar zero-crossing rate value. Given the template value x the algorithm retrieves all units that fall on the interval $[x\text{-}0.1, x\text{+}0.1]$.

7.3 ShuffMeter

Clarence Barlow's metric indispensability principle [15] has been successfully applied as a metrical supervision procedure when generating drum patterns in a particular style [16] as well as a model for constraining a stochastic rhythmic generation algorithm given a particular time signature [17]. The two algorithms work with symbolic music representations. shuffMeter extends previous research by applying Barlow's principle to drive the definition of targets that reflect a particular meter.

Given the scope of this paper and space restrictions, we cannot detail all the implementation of Barlow's metric indispensability. However, we follow the implementation described in [16].

After assigning a meter and a specific metrical level, the algorithm defines a hierarchical organization of the strong and weak beats of the meter to be better perceived by a listener. We mapped the weights into two audio descriptors: loudness

and spectral flux, by assuming that loudness and spectral changes are most likely to occur on the strongest meter accents. To simplify the computation we merged both descriptors into a single descriptor defined as their mean value. For each query the algorithm gathers the metrical weight *w* for that specific accent, and retrieves all units from the corpus with a value of *w*±0.1 for that descriptor.

We can apply this principle either on the whole corpus or on separate clusters, allowing as many layers as the number of existing clusters. The user can navigate in real time in a two dimensional map in the form of a square. Two pairs of variables mapped to each of the vertices of the square will adapt the configuration of the weights. The horizontal direction, from rough to smooth, will regulate the variability between all accents. The vertical direction, from loud to soft, will increase or diminish the weights proportionally.

Each concatenated unit is triggered by a timer assigned to the duration correspondent to the current beats per minute (bpm). This method was adopted here instead of a more natural strategy implemented in the previous section (7.2. infiniteMode), given the need to synchronize several units with slightly different lengths. If the units' length doesn't match the specified duration, they are consequently scaled in time by recurring to a time-stretch algorithm, which changes the speed of the audio signal without affecting the pitch.

7.4 SoundscapeMode

The soundscapeMode is a recombination method that was specially designed to recreate and work with environmental sound sources. It is a valuable and easy tool to design sound for film or installations, since it can structurally arrange on a map the units according to their perceptual qualities. The map has the form of a square divided into four main regions arranged in pairs of interconnected variables. The user can navigate in real time on the map as if he would navigate through a sound cartography. The first variables' pair controls the density of events (dense and sparse) and the second the roughness of the events (smooth and sharp).

The horizontal variable is density, i.e. the number of units played simultaneously, and ranges from 1 to 5. Smooth-sharp dichotomy, the second variables' pair represented vertically, aims at regulating and organizing the corpus in terms of diversity and stability and it is driven by the spectral flux descriptor. Spectral flux is a frequency domain feature, and describes the fluctuations in the spectrum of the signal. It was chosen because it is powerful in denoting attacks and sudden changes in the spectrum and thus for showing how stable the audio is along the unit. It is prudent to note that the application is highly dependent on the source file(s). If we feed the system with varying texture samples, the difference between smooth and sharp will be almost imperceptible.

As in infiniteMode, we added a block at the end of the target's definition that intends to maintain the best possible continuation between concatenated units, in terms of loudness and spectral changes. It is done by gathering all units' candidates for a specific query and finding the one that minimizes the distance on the bark spectrum representation to the previous selected unit.

8 Synthesis

Synthesis is done by concatenating units with a slight overlap. Each unit is played with amplitude envelope in the shape of a bell curve.

Most recombination methods make sure that the best possible continuity between concatenated units is guaranteed – i.e. if more than one unit matches the target at a certain point of the phrase, the system will select the unit that best matches the spectrum of the previous one. However, discontinuities and gaps still occur. To improve the quality of the synthesis an additional feature is added at the end of the chain in order to filter certain transition discontinuities on the audio flow. This is done with the help of an external object from the soundhack plugins bundle [18] named +spectralcompand~, which is a spectral version of the standard expander/compressor, commonly known as compander. It divides the spectrum in 513 bands and processes each of them individually. The algorithm computes iteratively the spectrum every 50 ms and applies it as a mask during synthesis.

9 Applications

The four recombination algorithms detailed in section 7 are suitable for a variety of music situations, spanning from sound installations to concert music. The design of the system doesn't reflect any particular music style. Our main purpose was to design a music system that learn from the music it draws its database from, and define coherent target phrases to be synthesized. Thus, the music output is highly dependent on the sound source(s), which are entirely selected by the user. In addition, some user supervision is needed to select certain recombination methods over others given the nature of the sound source(s). For example, if we fill the database with polyphonic music signals segmented on a beat basis it will be highly implausible that this collection of units will produce a consistent result when using soundscapeMode, which is mainly intended to recombining environmental sounds.

The system is easily adjustable to the context of interactive performance. All recombination methods have some degree of variability that can be easily controlled on the GUI. The interface for all recombination methods is intuitive and built as navigable maps that are almost self explainable. Instead of adjusting manually the several variables, the user can map characteristics extracted from an ongoing performance, whether they are motion, or sonic characteristics, or even any other measurable features extractable from a particular setting to any controllable variable of the interface.

The main purpose of the machine listening and learning techniques implemented in earGram is to drive the synthesis part of the software. However, the system may be useful for other applications domains outside this realm. The analytical and visualization tools that the software provides may constitute a valuable resource for the purpose of analyzing music under several fields such as computational musicology and cognitive musicology.

9 Conclusions and Discussion

This paper presents earGram, a novel CSS application built in Pure Data that comprises four generative music strategies that re-assign the original temporal order of the corpus for interactive music contexts, focusing on the user interface, MIR techniques of data analysis, mining and retrieval, and probabilistic modeling of timbre and harmony.

The visual representations offered in earGram gives the user a better understanding of the entire collection of units and the similarity amongst them. Most visualizations also allow interactive and guided exploration of the corpus, suitable for creating soundscapes and elucidating some decision-making concerning performance. The use of Barlow's indispensability algorithm proved to be an efficient method to ensure metrical coherence during the recombination process by providing a template that guides the definition of targets according to a predefined meter. A Markov chain algorithm was successfully applied to generate an infinite number of variations on the original signal with a minimum amount of interaction, while retaining the time varying morphologies of the source(s) modeled by a transition probability table between units.

The software together with many sound examples for all the recombination methods detailed in the paper and their respective project template used to create the examples are available at: https://sites.google.com/site/eargram/.

10 Future Work

The audio source(s) used during analysis are determinant for the possible outputs the system can consistently offer. CSS is only as good as the database from which it draws its sound units. Thus, besides requiring a rich database of sounds, more research should be addressed towards a better understanding of the source(s), which would contribute for more refined way of using the descriptors and help restricting the application field.

Concerning analysis further additions that center on the rhythmic content of the audio source(s) are under development. We believe that a better understanding of the source's rhythmic structure will enhance solidity during performance, particularly by avoiding gaps in the synthesis continuum, and by favoring typical articulations and textures of the corpus. Still regarding rhythmic instabilities, when layering different clusters of units, as it is done most notably in shuffMeter, but also in soundscapeMode, some rhythmic incongruence arouse due to the lack of alignment between overlapping units. Besides the valuable contribution that the before-mentioned rhythmic descriptions can add, some experiments will envisage to time-stretch units to align their rhythmic content.

Flexible methods for sequencing and mixing different recombination methods, various clusters, and diverse corpuses (notably combining corpus from different sources, e.g. live and pre-recorded sound) are under development.

Finally, concerning the applications domain, we consider that evolutionary methods could help at orienting the system, notably by defining larger targets that

consider more than the transition between consecutive units and by allowing control over the evolving process.

Acknowledgments. This work was partly supported by the European Commission, FP7 (Seventh Framework Programme), ICT-2011.1.5 Networked Media and Search Systems, grant agreement No 287711 (MIReS). We would like to thank George Sioros for his careful review of this paper.

References

1. Schwarz, D.: Current Research in Concatenative Sound Synthesis. In: Proceedings of the International Computer Music Conference, Barcelona, Spain (2005)
2. Hunt, A. J., Black, A. W.: Unit Selection in a Concatenative Speech Synthesis System Using a Large Speech Database. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (1996)
3. Zils, A., Pachet, F.: Musical mosaicking. In: Proceedings of the COST G-6 Conference on Digital Audio Effects, Limerick, Ireland, December (2001)
4. Schwarz, D.: Musical Applications of real-time corpus-based concatenative synthesis. In: Proceedings of the International Computer Music Conference (2007)
5. Inselberg, A.: Parallel Coordinates: Visual Multidimensional Geometry and Its Applications. Springer (2009)
6. Kandogan, E.: Visualizing Multi-dimensional Clusters, Trends, and Outliers using Star Coordinates. In: Proceedings of the Knowledge and Data Mining (2001)
7. Wattenberg, M. Arc Diagrams: Visualizing Structure in Strings. In: Proceedings of the IEEE Information Visualization Conference (2002)
8. Jehan, T.: Creating Music by Listening. Ph.D. Thesis, M.I.T., MA (2005)
9. Schwarz, D., Cahen, R., Britton, S.: Principles and Applications of Interactive Corpus-based Concatenative Synthesis. In: Journées d'Informatique Musicale, GMEA, Albi, France (2008)
10. Dixon, S.: An interactive beat tracking and visualization system. In: Proceedings International Computer Music Conference (2001)
11. Brossier, P.: Automatic Annotation of Musical Audio for Interactive Applications. Ph.D. thesis, Queen Mary, University of London (2006)
12. Brent, W.: A Timbre Analysis and Classification Toolkit for Pure Data. In: Proceedings of the International Computer Music Conference, New York, EUA (2010)
13. Heyer, L., Kruglyak S., Yooseph, S.: Exploring Expression Data: Identification and Analysis of Coexpressed Genes. Genome Research, 9:1106-1115 (1999)
14. Ester, M., Kriegel H., Sander, J., Xu, X.: A density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Proceedings of the Knowledge Discovery and Data Mining. AAAI Press, pp. 226–231 (1996)
15. Barlow, C.: Two essays on theory. Computer Music Journal, 11, pp. 44-60 (1987)
16. Bernardes, G., Guedes, C., Pennycook, B.: Style Emulation of Drum Patterns by Means of Evolutionary Methods and Statistical Analysis. In: Proceedings of the Sound and Music Computing Conference, Barcelona, Spain (2010
17. Sioros, G., Guedes, C.: Automatic Rhythmic Performance in Max/MSP: the kin.rhythmicator. In: Proceedings of the International Conference on New Interfaces for Musical Expression, Oslo, Norway (2011)
18. SoundHack Plugins Bundle, http://soundhack.henfast.com/