View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Huddersfield Repository

Descriptor driven concatenative synthesis tool for Python

Sam Perry
E-mail: u1265119@unimail.hud.ac.uk

Abstract

A command-line tool and Python framework is proposed for the exploration of a new form of audio synthesis known as 'concatenative synthesis', a form of synthesis that uses perceptual audio analyses to arrange small segments of audio based on their characteristics. The tool is designed to synthesise representations of an input target sound using a source database of sounds. This involves the segmentation and analysis of both the input sound and database, the matching of input segments to their closest segment from the database, and the re-synthesis of the closest matches to produce the final result.

The project aims to provide a tool capable of generating high-quality sonic representations of an input, to present a variety of examples that demonstrated the breadth of possibilities that this style of synthesis has to offer and to provide a robust framework on which concatenative synthesis projects can be developed easily. The purpose of this project was primarily to highlight the potential for further development in the area of concatenative synthesis, and to provide a simple and intuitive tool that could be used by composers for sound design and experimentation. The breadth of possibilities for creating new sounds offered by this method of synthesis makes it ideal for digital sound design and electroacoustic composition.

Results demonstrate the wide variety of sounds that can be produced using this method of synthesis. A number of technical issues are outlined that impeded the overall quality of results and efficiency of the software. However, the project clearly demonstrates the strong potential for this type of synthesis to be used for creative purposes.

Keywords: Concatenative synthesis; Python; audio descriptor; audio analysis; command line tool; Python framework; Python sound;

Acknowledgments

I would like to thank A Harker for his advice and guidance as a mentor throughout the project, and A Harker and P Chen for access to their vocal samples database. Thanks also to D Chaplin for his creative input in generating results.

Background

The concept of constructing a new sound by arranging collections of smaller sounds has gained popularity in the past 30 years through the introduction of granular synthesis, which works on the theory that any sound can be described through the arrangement of smaller samples (referred to as 'grains'). This representation of sound allows for the temporal decomposition and re-arranging of real-world samples, with the potential to create new 'complex, dynamically-evolving sounds' (Roads, 1988, p.1).

Concatenative synthesis (CS) is a form of synthesis that has developed significantly over the past 15 years, driven by recent advancements in technology. The key advancements have been in ease of access to large databases of audio and the development of methods for extracting useful information from these databases automatically (Schwarz, Beller, Verbrugghe, & Britton, 2006, p. 1). CS utilises these technologies to provide a content-based extension to granular synthesis; analysis of a database of source grains enable them to be differentiated based on their characteristics. These characteristics can then be used for grain selection in the process of synthesising output for a wide range of applications (Schwarz, 2007, p. 102).

Related works

A number of programs utilise CS to achieve various goals. The process has been used for applications in areas such as speech synthesis, instrument synthesis and creative sound design.
The wide range of applications demonstrates the versatility of this synthesis technique. It differs from traditional synthesis methods as it uses real recorded samples, as opposed to traditional methods that focus on defining sets of rules for emulating real sounds. By transforming samples that have been directly recorded from a source, the subtle nuances of the source's sound are preserved. These would be difficult to reproduce using other synthetic methods for modelling an instrument (Maestre, Ramrez, Kersten, & Serra, 2009, p. 24).

Speech synthesis

Creating a natural and intelligible realisation is an important factor when developing a speech-synthesis system. The Talkapillar project is one such example of how highly convincing results are possible with CS. Through careful analysis of a vocal database, the project aims to impose the qualities of the database voice on an input voice. This would result in the words of the input speaker being transformed to appear as if they were spoken by the voice in the database. (Beller, Hueber, Schwarz, & Rodet, n.d.)

Instrument Synthesis

Progress has also been made in improving the quality of instrumental synthesis. As with speech synthesis, the use of samples directly allows for natural-sounding results, which provides a method for reproducing real instruments convincingly. Another important aspect of instrumental synthesis is that of performer expression. The reproduction of performance qualities such as dynamics, timbre and timing is essential when emulating a real instrument and CS has been used to effectively reproduce these aspects. This is achieved through splicing of grains based on their expressive characteristics to form musical phrases. For example, just as a violinist might transition seamlessly from one articulation to the next, the CS software will join grains to produce the variation in articulations. This contrasts with the traditional approach to sampling, where samples are played in isolation, resulting in a discontinuity between adjacent samples (Lindemann, 2007, p. 82). The Catapillar project is one such example of this use of CS. By using a Viterbi algorithm, the project is able to calculate the smoothest overall transition between grains across the output, resulting in convincing synthesis of orchestral instrument performances (Schwarz, 2003, p. 5).

Creative sound design

The flexibility of CS allows for creativity in a broader context than simply emulating real-world instruments and speech. It can also be used as a tool to explore the possibilities for synthesising new abstract sounds for creative purposes.

A prominent project in this area of CS is IRCAM's CataRT project (Schwarz et al., 2006). The project focuses on the playback of source grains based on their proximity to a target in multi-dimensional descriptor space. Providing a target point in the descriptor space enable the user to navigate the database, playing selections of samples that are nearest to the target. This allows the user to explore the database intuitively through a graphic user interface, selecting a point in 2-dimensional space with the mouse. Grains are then played back in real-time to create an 'audio mosaic'.

Alternatively, target audio can be provided and analysed to create a target location based on it's location in the descriptor space. Tremblay and Schwarz's (2010) use of CataRT to explore electroacoustic sample banks demonstrates the creative potential of this method. CS is used in this context as a means of synthesising matches in a corpus database to real-time input from an electric bass. Significance is placed on linking the playback of grains to the expressively of the performer. The use of perceptually based audio descriptors to match the source to the target allows the performer to navigate the database naturally based on factors such as the pitch and timbre of the bass guitar. The result is a performance that mixes characteristics of both the bass guitar output and the qualities of the corpus database to create a hybrid of the two.

This is by no means an exhaustive overview of the projects and techniques that explore the vast possibilities of CS. Further information can be found in the article by (Schwarz, 2006)

Concatenator

The Concatenator project aims to provide an open source tool that allows composers to generate a variety of CS driven realisations for sound design purposes. In addition, the project aims to provide an intuitive API that Python programmers might use as the fundamental building blocks on which to build further CS applications. The result is a framework and command-line interface, built in Python, for easy access to basic CS techniques. All relevant material including source code, results, and documentation can be found in the official online project repository (Perry, 2016). The current implementation can be used for the concatenation of a source database onto target audio files, using a range of perceptual audio descriptors for matching. Database management, simple matching and synthesis algorithms are used to achieve this, and are described in the following sections.

The features and uses of this tool are most comparable to those of the MATConcat project (Sturm, 2004), which was developed to provide an open source tool for generating similar representations of audio in MATLAB. Although there are technical differences such as the number of descriptors available for each project, both share a similar focus on the electro-acoustic compositional applications of CS. Results produced for the MATConcat project are comparable to those of the Concatenator project, and both work offline to produce results. The Concatenator project builds on this by providing a wider variety of descriptors and the ability to artificially enhance matches (as discussed in the Synthesis and Transformations section).

Program design and implementation

The Concatenator project consists of a number of components that work together to produce the final output. A complete description of all components and there usage in the Concatenator project can be found in it's documentation.

Output is generated by analysing overlapping segments of audio (known as grains) from both the target sound and the source database, then searching for the closest matching grain in the source database to the target sound. Finally, the output is generated by applying a hanning window and overlap-adding the best matches. Each component is discussed in detail in the following sections.

When designing the Concatenator framework, ease of development, use and extensibility were primary considerations. It was for these reasons that the framework was written in the Python programming language. Python has grown in popularity in the scientific community recently, primarily due to its focus on productivity, readability and the large number of efficient numeric processing libraries available (Pedregosa et al., 2011; Fangohr, 2014; Jones, Oliphant, Peterson, et al., 2001). This makes Python a good choice for quickly developing ideas in the context of audio signal processing. Unfortunately, the language does sacrifice processing speed for simplicity, and as a result, is not suitable for real-time signal processing. Other performance-focused languages such as C++ are better suited to this type of processing. However, it was decided that the increase in productivity, lack of prior CS research in Python and the author's previous experience made it the most suitable choice for this project.

The choice to limit the project to offline processing has both positive and negative implications for the function of the project. A key disadvantage of this type of processing is the lack of possibility for any live performance aspect. This method provides no way of exploring the feedback between performer and system in a live environment, as in the work of Tremblay and Schwarz (2010). However, there are advantages to offline processing that would not be possible in a real-time context.
One significant advantage is that databases can afford to be far larger than they could be in real time. Without the requirement to process output in a short period of time, more time can be taken to search vast databases in the hope that the closest match to a target will be found.
Another advantage is in the global view of a target that can be taken in an offline approach. Because the complete audio file is available from the start of processing, techniques can be applied that consider the output as a whole, rather than on a grain-by-grain basis. This allows for algorithms such as the Viterbi algorithm to find the sequence of grains that provide the best continuity, as demonstrated in the Catapillar project (Schwarz, 2003, p. 4) This would not be possible in real-time, as audio is processed 'on the fly'.

An additional consideration was the method to be used for controlling the target to which the grains would be matched. It was decided that the most interesting results would be produced through the matching of grains to a target audio file, as opposed to other approaches such as matching to MIDI scores. In this sense the project is a form of offline audio-mosaicking tool similar to that of CataRT.

Descriptor Implementation

In order to differentiate between grains, a number of audio descriptors were implemented. Audio descriptors are used to measure a specific characteristic of a signal (Lerch, 2012, p. 31). For example, a root mean square (RMS) descriptor was implemented to give an indication of the overall intensity of the grain. Another example is the fundamental frequency (F0) descriptor, which was implemented to give a value relating to pitch for harmonic grains. These values could then be used by the matching algorithm in order to find the best match between the source and target grains.
Owing to time constraints on the project, only a limited number of basic descriptors were implemented. For this reason, the project was designed so that new descriptors could easily be added. The object-oriented design of the descriptors provides the potential for quick development of any future descriptors to be added.

Database design

When generating descriptors for large databases, large amounts of data are produced and so an efficient method of storing and retrieving the data was needed in order to manage this. The Python interface to the HDF5 filesystem (Collette, 2016) was chosen for it's simplicity and ability to compress the data automatically. Storing Numpy arrays of descriptors in groups allowed for quick and easy access to analyses from a single, organised source.

Matching algorithms

In order to match grains using the descriptor values, a matching algorithm was required. Initially a brute-force matcher was used to compare each descriptor value in the target to all values of the same descriptor type in the source. However, it quickly became apparent that this approach would be far too slow, particularly for a larger database.
For this reason, a k-dimensional tree search algorithm was used in an effort to improve matching efficiency. This approach produced the same results as the brute force matcher, but by arranging descriptors in a tree structure, a far more efficient search to find the best match was possible. This reduced matching time considerably.

Synthesis and transformations

The final step in the program was to synthesise the matched output. This process consisted of:

1. Retrieving the best grain matches returned by the matching algorithm
2. Applying a window function
3. Overlapping the grains
4. Transforming grains to match the target
5. Saving the result to a file

Initially, grains were not transformed to better match the target. This worked effectively for large databases; however, it was observed that results synthesised using small databases were of a lower quality, as the chance of a closely matched grain was lower. To account for this, methods for altering grains to better match their target were implemented. It was decided that the two most significant characteristics to alter were the pitch and intensity of the grains. By scaling the grains by the difference between the source and target RMS, it was possible to impose a closer intensity on a grain. Likewise, by shifting the pitch of a grain by the difference, it was possible to better match the pitch contour of the output to that of the target audio. This improved the results significantly in smaller databases, as poor matches could be improved to match the target more convincingly.

Command-line interface

In order to make the framework accessible to users, a command-line interface was developed. By supplying arguments to the program, users could alter parameters and experiment freely with the tool. Although this interface was sufficient for testing and experimentation, it quickly became apparent that there were too many parameters to pass to the program via the command line interface on each run. A configuration file parser was created to address this issue, allowing users to specify default parameters that would be used by the program on each run. The combination of these interfaces provided an effective means for accessing all of the framework's features.

Documentation and API

Complete documentation for the project was created in order to make the project as user friendly as possible for both developers and users. As a result, a full API is available alongside examples of use and instructions for command-line operation. This was created in the hope that it might form a usable package that developers can build on quickly and effectively to build other CS projects, allowing for easier access to Python-based CS than is currently available. The command-line interface is equally documented to allow users to create their own realisations quickly and easily so that this project may be used for creative sound design purposes.

Results and evaluation

Overall, the results generated by this project showed promise; a variety of transformations were generated using open source instrument databases to demonstrate the projects potential for sound design application. This tested the project's ability to convincingly impose qualities of an instrument onto target sounds. A variety of examples are provided that outline the style of synthesis aimed for. These range from imposing acoustic guitar qualities on an electric guitar to imposing stringed instrument qualities on vocal melodies. Current results have a clear synthetic nature, but still clearly exhibit some of the main characteristics of the database used.

Research Limitations/Potential Development

In retrospect, a great deal of time was spent trying to improve the efficiency of the project. Although this was necessary, as initial tests were not feasible on most databases, it had a negative impact on the time available for developing perceptual qualities of the output. As a result of this, the overall quality of output might not perhaps be as natural as that of other projects in this area. This is apparent in the vocal → string instrument examples. Phrases tend to begin and end abruptly, failing to replicate any defined attack or decay of the string instruments, as would be expected when hearing a string

instrument naturally. Conversely, this does give output it's own synthetic characteristic, which may be desirable as perfect reproduction of an instrument may not be the reason for using this tool.
In addition, the amount of computation required results in large amounts of time needed to produce high quality results. An end user may not have the patience required to reach the quality of results that might be possible. This is in part a drawback of the Python language, and could be better accounted for with further work on profiling the performance of the tool.
However, the fundamental concepts such as descriptor matching and transforming matches to better fit the target, which are used in the most sophisticated CS projects, have been implemented in this project to satisfying creative effect. As a proof of concept, this project displays the possibilities for CS in Python and there is evidently potential for further development in this area.

There are a number of further improvements that could be made to this project in order to improve the quality of results and extend it's overall usefulness. These range from reasonably simple modifications that could not be implemented purely due to time constraints, to more complex ideas that may take a considerable amount of work. The following is a list of some initial ideas for improvements.

- The current implementation uses only a small and relatively basic subset of the audio descriptors available. This limits the analysis of audio and thus the quality of matches. Using a larger set of more advanced descriptors may improve quality from this perspective. One way would be to incorporate the open source Essentia audio descriptors (Universitat Pompeu Fabra, 2016) giving the project access to a vast quantity of descriptors for analysis.

- Replacing the hanning window function used for grain windowing with a short cross-fade at grain overlaps should reduce amplitude modulation, resulting in smoother transitions between grains. This might be further improved through calculating the point of maximum similarity by cross-correlating overlapping sections, as described by Zolzer (2011, p.191-193) in the Synchronus OverLap Add (SOLA) algorithm.

- A lack of continuity between grains was observed in results, most likely owing to the lack of any comparison of selected grains. A Viterbi algorithm could be used to account for this, allowing for a search to be done amongst the top matches to find the optimal set of grains. This takes advantage of the offline nature of the project and has been shown to work effectively in the Talkapillar project (Beller et al., n.d.).

- Although the HDF5 filesystem allows for easy storage of descriptor values, it also has drawbacks that limits the functionality of the project. One significant problem is that it is difficult to implement parallel processing using the library and for this reason asynchronous processing was not implemented in the project. An alternative method of storage may accommodate this more easily, allowing for the speed-ups possible through asynchronous processing. The overall design of the database management was also relatively naive and may benefit from being replaced by a technology such as an SQL database or similar. This has been shown to work effectively in work such as the CataRT project (Schwarz et al., 2006, p.3).

Conclusion

This project has provided a functioning Python based CS project with much potential for further development. Given the number of technical issues faced with this style of synthesis (from the big data issues faced with analysis storage, to high efficiency requirements for processing the large quantities of data), overall this project appears to work effectively. It provides a new and accessible means for tapping some of the vast amount of potential that concatenative synthesis has to offer.

With the ever increasing quality of technology, it is predicted that new techniques such as concatenative synthesis may grow further in popularity, leading to an increasing number of possibilities in this area of sound synthesis. It is hoped that this project might aid in the highlighting the possibilities offered by this form of synthesis and demonstrate some of the technical obstacles that must be addressed to design a CS project successfully.

References

Beller, G., Hueber, T., Schwarz, D., & Rodet, X. (n.d.). An Overview of Talkapillar, 2–6. Retrieved from http://recherche.ircam.fr/anasyn/concat/doc/Talkapillar_overview.pdf

Collette, A. (2016). HDF5 for Python. Retrieved August 28, 2016, from http://www.h5py.org/

Fangohr, H. (2014). Introduction to Python for Computational Science and Engineering, 162. Retrieved from https://www.southampton.ac.uk/~fangohr/training/python/pdfs/Python-for-Computational-Science-and-Engineering.pdf

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy.org. Retrieved from https://www.scipy.org/

Lerch, A. (2012). *An Introduction to Audio Content Analysis*. doi:10.1002/9781118393550

Lindemann, E. (2007). Music synthesis with reconstructive phrase modeling. *IEEE Signal Processing Magazine*, *24*(2), 80–91. doi:10.1109/MSP.2007.323267

Maestre, E., Ramrez, R., Kersten, S., & Serra, X. (2009). Expressive Concatenative Synthesis by Reusing Samples from Real Performance Recordings. *Computer Music Journal*, *33*(4), 23–42. doi:10.1162/comj.2009.33.4.23

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay,. (2011). Scikit-learn: Machine Learning in Python Gal Varoquaux. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perry, S. (2016). Concatenator Source Code Repository. Retrieved from https://github.com/Pezz89/PySoundConcat

Roads, C. (1988). Introduction to Granular Synthesis. *12*(Summer), 11–13.

Schwarz, D. (2003). The Caterpillar System for Data-Driven Concateantive Sound Synthesis. *Proceedings of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, 1–6.

Schwarz, D. (2006). Concatenative Sound Synthesis: The Early Years. *Journal of New Music Research*, *35*(1), 3–22. doi:10.1080/09298210600696857

Schwarz, D. (2007). Corpus-based concatenative synthesis. *IEEE Signal Processing Magazine*, *24*(2), 92–104. doi:10.1109/MSP.2007.323274

Schwarz, D., Beller, G., Verbrugghe, B., & Britton, S. (2006). Real-time Corpus-Based Concatenative Synthesis with CataRT. *Proceedings of the 9th Int. Conference on Digital Audio Effects (DAFx-06), Montreal, Canada*, 1–7. Retrieved from http://articles.ircam.fr/textes/Schwarz06c/index.pdf

Sturm, B. L. (2004). MATConcat: an application for exploring concatenative sound synthesis using MATLAB. *Proceedings of DAF-x04*, *2*, 323–326.

Tremblay, P. A. & Schwarz, D. (2010). Surfing the Waves : Live Audio Mosaicing of an Electric Bass Performance as a Corpus Browsing Interface. *In Proceedings of the 2010 International Conference on New Interfaces for Musical Expression (NIME10), Sydney, Australia*, 447–450. Retrieved from http://eprints.hud.ac.uk/7421/

Universitat Pompeu Fabra. (2016). Essentia. Retrieved from http://essentia.upf.edu/

Zolzer, U. (2011). *DAFX: Digital Audio Effects* (2nd ed.). Chichester, England: Wiley.