Chapter 1

Computer Applications

In recent years, my approach to composing and performing music has reached a point were the development of computer applications runs simultaneously with my aesthetic and creative research. From the very start of composition and conception until the performance and realization, the computer applications that are developed and the aesthetic results merge within the same artwork. Therefore, the creative processes in my practice is deeply connected to computer programming and the use of computer applications. It is worth mentioning that the applications developed as part of this process are vital to the musical results of the compositions and encompass an important aspect of the compositional process. In addition, these applications were developed for an artistic purpose and understanding how they work might give an insight into my compositional output. Moreover, they do not serve a function beyond the realization of the idiosyncratic elements that constitute my creative process. That is to say, these applications do not represent a contribution to the scientific community in relationship to the technological developments of computer music but instead represent a set of tools and documentation that other artists, musicians and composers might find useful for their own practice.

In the previous chapter, I explained some of the potential that technology has brought to music and how even though technological advancements do not necessarily represent a artistic developments, they do provide with new possibilities for artistic innovation. It is because of these possibilities that I have become recently interested in using digital technology to create music. I am particularly interested in using technology for...

In this chapter, I will explain in detail a number of important computer applications that were developed together with my compositional practice. These applications were written in the SuperCollider[1] programming language. I decided to use SuperCollider as a platform to develop these

[1] James McCartney, SuperCollider, 1996. URL: http://www.audiosynth.com/

computer applications because it integrates a powerful audio synthesis server using state-of-the-art technology with the versatility and capabilities of an object-oriented-programming (OOP) language. I chose SuperCollider over other data flow programming applications like Max/MSP and Pure Data (Pd) because of its robust synthesis server and the advantages of abstraction of a high-level OOP language.[2] Another advantage of using SuperCollider is the fact that it is an open source application, which means that the code in which it is written is available for free and can be modified. Most of the computer applications I developed and which will be discussed in this chapter are written as SuperCollider classes[3] but some of them are extensions of already existing classes. The applications discussed in this chapter where used in various of the works submitted and constitute compositional strategies that reflect some aesthetic concerns that are recurrent in my work.

1.1 Spectral Tracking

In previous chapter, I briefly explained how spectralism and C. Barlow: *Synthrumentation* influenced my work.

Fast Fourier Transform (FFT)[4]

MIDI[5]

Spectrum analysis for dynamics, pitch and rhythm extraction.

1.1.1 PartialTracker

PartialTracker is a SuperCollider class for real-time partial extraction that diminishes the amount of FFT data by selecting the loudest bins and discarding the softer magnitudes with the purpose of having a limited amount of values to be returned as simple arrays for frequency and magnitude. It does so by taking an incoming audio signal, performing an FFT analysis and discarding spectral data in two ways: either by passing only the bins that are above a given threshold or by selecting a value that returns the strongest number of bins. For this purpose, I used the PV_MagAbove and PV_MaxMagN[6] phase vocoder unit generators. In order to have access to this data in the language

[2]See James McCartney, "Rethinking the Computer Language: SuperCollider", in *Computer Music Journal*, volume 25, number 4, pp. 61-68, 2002, for a discussion about the differences between SuperCollider and Max/MSP, Pd and Csound.

[3]SuperCollider classes are descriptions of the structure and implementation of a set of objects that represent the instances of the class.

[4]See http://www.wikipedia.com/FastFourierTransform for information about FFT.

[5]See http://www.wikipedia.com/MIDI for more information about MIDI

[6]Joshua Parameter, JoshUGens, 2002. URL: http://www.sourceforge.com

side of SuperCollider, I used PV_MagBuffer and PV_FreqBuffer[7] to store this information in a buffer. Ones stored in a buffer, the information can be accessed as an array and be manipulated freely. Nevertheless, the buffer stores all of the bins of the FFT and therefore the bins with the magnitudes that where not empty had to still be collected and indexed to access the corresponding frequency values. The resulting arrays therefore constitute only of the number of strongest bins, which can be defined by the user. Figure 4.1 shows and example of the frequency and magnitude arrays for the ten strongest bins. The purpose of this class is to have easy access to relevant FFT information for

```
[ 128.37791442871, 154.57292175293, 140.25003051758, 246.26268005371, 253.09353637695,
 364.92492675781, 396.52267456055, 1035.068359375, 1037.1043701172, 1241.3063964844 ];
 //array of frequencies

[ 1.8754153251648, 4.5471153259277, 5.3137745857239, 2.6146886348724, 1.2295168638229,
  2.5435922145844, 3.215939283371, 2.0944044589996, 2.6014709472656, 2.0559096336365 ];
  //array of magnitudes
```

Figure 1.1: PartialTracker: Frequency and magnitude arrays.

the purpose of converting frequencies and magnitudes either as MIDI messages or as data to be used to control synthesis definitions[8]. The incoming signal can be a live input in a performance situation, or a sound file. Lastly, this class also provides the feature of storing the spectral information that is triggered by an onset detector with the purpose of creating a MIDI file by storing time values and converting the frequency and magnitude data to MIDI notes and velocities.

1.1.2 FFTFilter

FFTFilter inherits functionality from PartialTracker and uses the information of the frequency array to control the bandwidth and center frequency of a two pole resonant filter. This FFT controlled filter is designed to be used to filter a signal with a rich spectral content, such as different types of noise, with the information given by an FFT analysis of another signal that should be more limited in its frequency range. A function is evaluated in a loop in which an argument that can be changed by the user is the time value between each iteration. This function accesses the highest and lowest

[7] Ibid.

[8] Synthesis definitions, or SynthDefs in SuperCollider represent a description of a set of Unit Generators (UGens) that perform synthesis algorithms in the SuperCollider server.

frequency values from the array calculated by the PartialTracker functionality every time the loop is evaluated. Since the purpose is to make a smooth line instead of discrete points, the signal must be lagged exponentially to produce a continuous control signal. By following this procedure, it is possible to approximately track the contour of the frequencies of a signal that have a stronger presence, given that the settings for the amount of strongest bins and magnitude threshold are appropriate for the specific spectral characteristics of the signal. Figure 4.2 shows a spectrogram of speech and how the

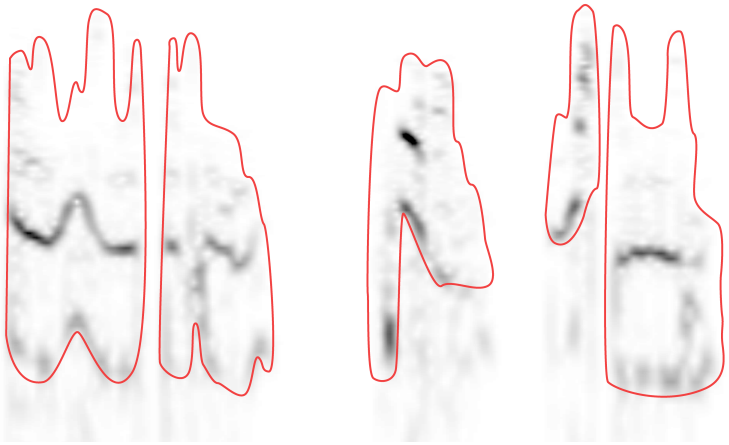

Figure 1.2: FFTFilter: Spectral mapping of vocal contour.

FFTFilter maps the contour of a vocal signal. Given that the vocal signal has a strong presence in a narrow frequency range, it is ideal to control the filter. FFTFilter therefore uses the continuos signal of the highest and lowest frequencies of the array to calculate the bandwidth and center frequency for the resonant filter. Figure 4.3 shows a visual representation of a fairly noisy signal that has been filtered by the resonant filter following the vocal contour as seen in Figure 4.2. Ones the trajectory

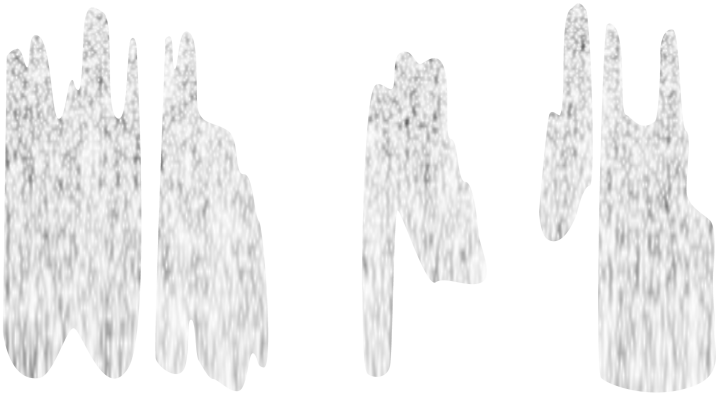

Figure 1.3: FFTFilter: Noise filtered by vocal contour.

of the filter is set by the frequency data extracted from the FFT, an envelope follower maps the amplitude of the sound that was used as the FFT input to control the amplitude of the resonant filter. Therefore, by combining the amplitude envelope and frequency contour of one sound to control a resonant filter that is applied to a second sound, it is possible to incorporate characteristics of the first sound to the filtered sound source.

1.1.3 SpearToSC and SpearToMIDI

SpearToSC is a class that takes data from the open source software application called SPEAR[9] and transfers it to an array in SuperCollider. SPEAR uses a variation of the traditional McAulay-Quartieri procedure and "attempts to represent a sound with many individual sinusoidal tracks (partials), each corresponding to a single sinusoidal wave with time varying frequency and amplitude."[10] SPEAR provides a graphical representation of a sound[11] (as seen in Figure 4.4) in which it is possible to select the individual sinusoidal tracks and allows to isolate and access the information for each individual partial. The amplitude and frequency information of each partial is given by

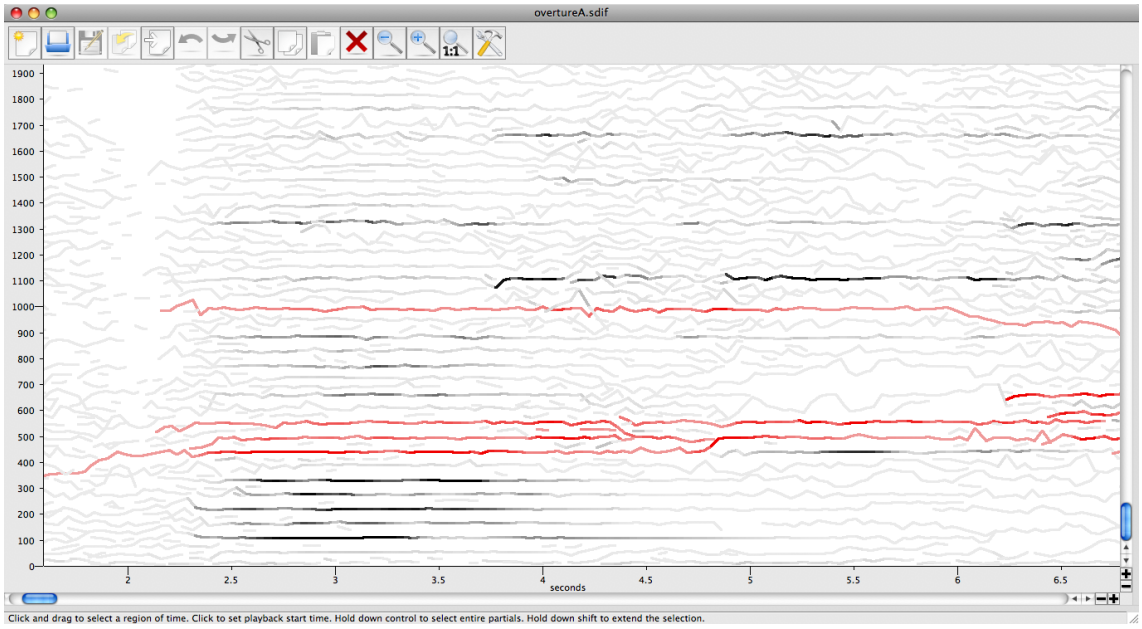

Figure 1.4: SPEAR graphical interface.

frame can be stored in a text file. SpearToSC reads text files produced by SPEAR[12] as a string

[9] Michael Klingbeil, SPEAR, 2005, URL: http://www.klingbeil.com/spear/.

[10] Michael Klingbeil, M. 2005. "Software for spectral analysis, editing, and synthesis." in *Proceedings of ICMC*, vol. 2005, 2005. URL: http://www.klingbeil.com/papers/spearfinal05.pdf.

[11] Spectral analysis where the y-axis represents frequency in hertz and the x-axis represents time in seconds.

[12] SpearToSC reads SPEAR text files in the *Text - Partials* format only.

and strips it into a multidimensional array in SuperCollider. It is therefore possible to process this data within the SuperCollider language and server and re-synthesize this information not only with sinusoidal waves, but with any type of unit generator.

SpearToMIDI is a class that inherits functionality from SpearToSC and reduces the information given by SPEAR to be used as data to produce a MIDI file or to control SuperCollider synthesis definitions. The purpose of this class is to reduce the spectral information to an amount of data that can later produce notated material for a written score, a MIDI file or a control system to be used for triggering synthesis algorithms. The data in the text file generated by SPEAR is available by frame and gives too much information for this purpose. Therefore, SpearToMIDI reduces this data in four stages: First, it takes an amplitude threshold argument which gets rid of all of the partial information that lies bellow this value (as seen in Figure 4.5). In other words, it breaks the partial

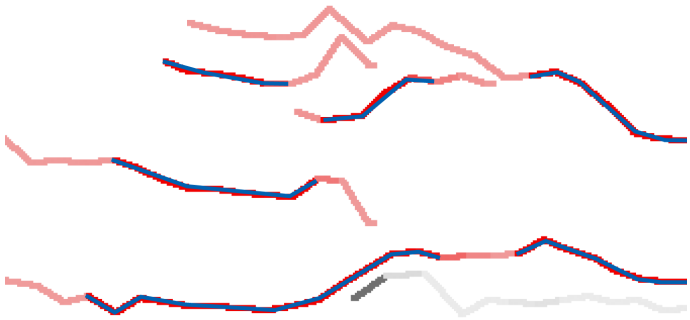

Figure 1.5: SpearToMIDI: Amplitude threshold selection.

in different groups by introducing silences instead of the data that lies bellow the threshold and at the same time keeps track of the beginning and the end of each group.

The second stage reduces data with a frequency modulation threshold. Each group is taken as a line and the computer only stores the points in the line which cross a given interval (the modulation threshold). For example, Figure 4.6 shows how the lines representing the groups in Figure 4.5 are traced by selecting the points that cross a given interval.[13] If the interval is of one semitone then the frequencies are averaged to the closest chromatic note. It is possible to make microtonal divisions of the equal-tempered scale by using floating point values for the modulation threshold. The magnitude, frequency and time values for each point are stored as a collection of data. This collection can then be accessed and used to control synthesis definitions externally by generating envelopes, which gradually

[13]The grid represents the intervals as shown in the y-axis. For the purpose of simplification, the diagram doesn't show a logarithmic representation of frequency.

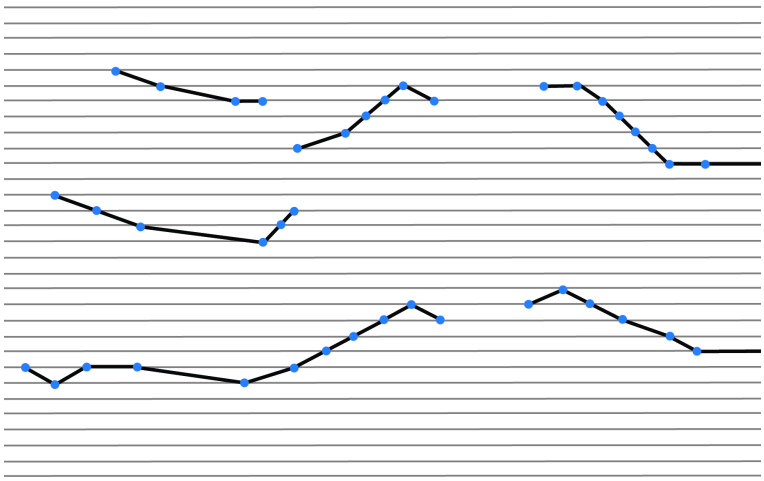

Figure 1.6: SPEARToMIDI: Point selection through frequency modulation threshold.

change frequency to produce glissandos and amplitude for gradual volume change. After these first two stages, the original data from Spear is reduced considerably by disregarding details that are not vital for the given purpose.

The third stage, takes the points of the lines that where obtained in the previous stage and translates them into single notes with a start and an end and that do not change in frequency and amplitude while playing— in other words, a format that is compatible with the MIDI note-on and note-off paradigm. The points are then considered as representing note-on messages and the note-off messages are calculated depending on whether the point is followed by another new point or if the point is the last of the group, in which case a silence would proceed. In other words, a note-off is inserted before a new note-on or just before a silence. Figure 4.7 shows the note representation

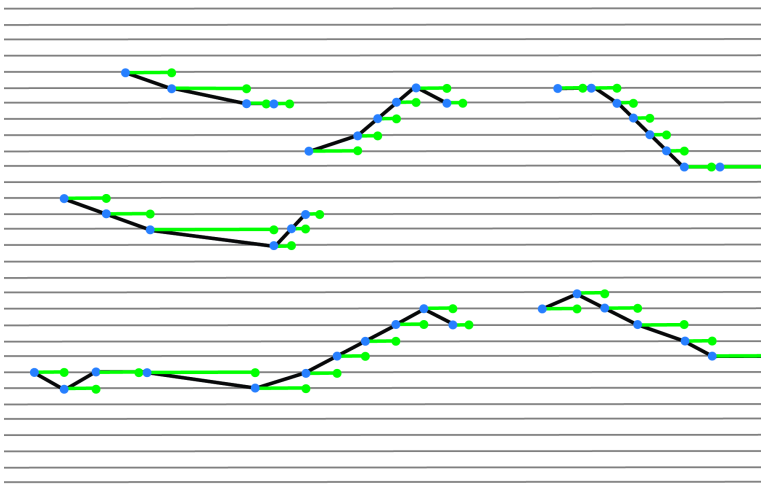

Figure 1.7: SPEARToMIDI: Note representation.

derived from Figure 4.6, where the notes are seen as green lines, the note-on messages as blue points and the note off messages as green points. The results of this stage can be used to generate a MIDI file[14] with the intention of either using it to trigger a sampler or to import it into a notation software to create a written score. The user can input the time signature and tempo for the MIDI file as well as an interval value that divides the MIDI note range into different MIDI tracks. By doing this, the notes are separated into different tracks depending on their value in relationship to each other with the purpose of not having too many notes in the same track. Furthermore, these results can be used to create a list of OSC (Open Sound Control) commands that can be sent to the SuperCollider Server for Real-Time-Synthesis and Non-Real-Time-Synthesis. Extra arguments can also be added to control other values in the synthesis definition, which can be set individually by using a function to be evaluated for each instance of the definition.

1.2 Real-Time Scoring

1.2.1 AlgorithmicScore

1.3 Pre-compositional Tools

1.3.1 MIDI Mapping

1.3.2 MIDI Triggering

[14] Using the SimpleMIDIFile class that is part of wslib by Wouter Snoei, which is can be obtained as a Quark.