From Symbolic Representation to 2D Spatial Manipulation: A Transcription of Fractal Models from OpenMusic to Max

Said Bonduki[1*], **Danilo Rossetti**[13], **Ivan Simurra**[12]

[1]IA - Unicamp
R. Elis Regina, 50 - Cidade Universitária, Campinas - SP

[2]NICS - Unicamp
R. da Reitoria, 165 - Cidade Universitária, Campinas - SP

[3]FCA - UFMT
Av. Fernando Correa da Costa, 2369-2455 - Cuiabá - MT

sabonduki@gmail.com, d.a.a.rossetti@gmail.com, simurra@unicamp.br

Abstract. *This article presents a project focused on the transcription of patches for musical composition and analysis originally developed in the OpenMusic environment into Max/MSP, with particular emphasis on the implementation of a compositional model based on fractals proposed by Mikhail Malt. The model employs Iterated Function Systems (IFS) to generate fractal structures that control frequency and temporal parameters in music, initially applied in the composition Six Fractal Contemplations. The transcription aims to explore the capabilities of Max/MSP to integrate symbolic notation (through the Bach library) and graphical visualization (via the CNMAT library), thereby enhancing the accessibility and expanding the functionalities of historically significant tools in Computer-Assisted Composition (CAC). It is concluded that the model offers a bridge between mathematical abstractions and creative processes, reinforcing the relevance of CAC in both preserving and innovating computational tools for music.*

1. Introduction

Computer-Assisted Composition (CAC) is a field within music technology that has been developing since the 1950s. It encompasses two widely disseminated branches: sound manipulation and the manipulation of symbolic representations related to musical scores [1]. The first computational experiments using computers involved methods based on probabilistic analysis of existing music corpora, random generation of musical data using Markov chains derived from these analyses, and the possibility of selecting materials among the results. A fully 'computerized' composition emerged with the *Illiac Suite*, created by L. Hiller and L. Isaacson in 1957 [2].

Computational Musicology (in short CM) has developed in parallel with CAC as a branch of systematic musicology, alongside a variety of independent disciplines that form a network for understanding the musical phenomenon [3]. As with CAC, CM can rely on methods for the analysis and classification of audio signals—known as Music Information Retrieval (MIR) [4]—as well as tools for symbolic analysis of musical data through digital methods that assist in understanding the information present in musical scores [5].

This paper presents a project that is part of a broader initiative launched by the authors in 2024. The aim of the project is to transcribe support *patches* for musical composition and analysis—originally developed in the OpenMusic environment—into the Max/MSP platform. This paper specifically discusses the transcription of a *patch* created by Mikhail Malt for constructing a compositional model based on fractals that generate similar forms (variations) and control the musical temporal and frequency dimensions [6]. Malt used this model in the composition of six solo instrument and fixed media pieces titled *Six Fractal Contemplations*. The main tool used for fractal generation was Iterated Function Systems (IFS), available in the *Om-Chaos* library of OpenMusic. IFS are functions applied repeatedly, each time using the result from the previous iteration as the input. Through iteration, the function traces an orbit that is relevant to constructing chaotic systems. Some nonlinear functions contain attractors with fractional dimensions, forming fractals. IFS can produce connected or disconnected, self-similar or non-self-similar fractals, which may also contain non-fractal elements [7]. This is a technique for constructing fractals by repeating the same figure at different scales.

The transcription of tools that assist in computer-aided composition and computational musicology is relevant today, as computer systems are constantly updated and new platforms emerge. This facilitates access for new users to tools developed in the past and ensures their continued availability [8]. It should be noted that this is not precisely the case for Malt's *patch* discussed in this article, as it still functions properly in current versions of OpenMusic, along with its required external libraries. However, the current versions of Max/MSP, through certain external libraries, offer additional functionalities important for integrating musical notation, symbolic operations, and real-time audio synthesis. These are precisely the functionalities we aim to explore through our transcription, which we will detail in this text.

In the following sections, we will first provide a brief history of CAC and its various approaches. We will then present the features of the Max implementation of the fractal-based compositional model, using symbolic nota-

*Supported by Capes.

tion supported by the external Bach library. Next, we discuss the results achieved thus far, particularly the possibilities of rotation, expansion/compression, and displacement along the X (spatial/temporal position) and Y (pitch) axes. Finally, we present the partial conclusions of this ongoing project.

2. Computer-Assisted Composition

Computer-Assisted Composition, also known as CAC, refers to the use of computational systems and algorithms to support, enhance, or automate the processes of generating musical material. One of CAC's primary purposes is to reduce cost and time while optimizing the production of musical material, whether in symbolic form or directly in audio, regardless of the style, genre, or intended audience [9]. CAC platforms also contribute to the development of programming languages and graphical user interfaces (GUIs), creating environments for projecting, processing, transforming, and interacting with musical data using event- or demand-driven paradigms [10, 11]. These systems not only assist the compositional workflow but also allow for the exploration of alternative possibilities through interaction and hybrid experiences with computational support.

William Buxton, when discussing the context of CAC, argues that it should support the development of projects by explicitly revealing 'musical decision-making processes' [12]. Otto Laske identifies a defining feature of CAC as the interaction between the composer and the machine [13]. For Laske, CAC is also characterized by a balance between intuition and the role of the 'computational machine,' as well as by flexibility in both approaches [13, p. 01]. Drawing from fields such as architecture and instrument design through Computer-Aided Design (CAD), Tristan Murail argues that a CAC environment should be open and capable of accommodating various compositional perspectives and project-specific requirements [14].

One development within CAC involves various approaches to musical production, including both 'coding' and 'graphical representation.' These include software for music notation, such as Finale, Sibelius, Lilypond, MuseScore, among others. While many CAC systems support notation, others focus on interacting directly with recorded or produced audio material, especially for analyzing frequency spectra. Tools such as AudioSculpt[1], SPEAR[2], and Sonic Visualiser[3] are also considered 'composition assistants,' although their primary focus is not on score representation but on sound analysis. Nonetheless, this paper focuses on the use of CAC in the context of symbolic music generation, which will be explored in more detail.

As noted in Section 1, computational interaction via CAC was already underway by the mid-twentieth century. A pioneer in developing graphical composition interfaces, Xenakis created a machine for manipulating sound frequencies and intensities called the *UPIC – Unité Polyagogique Informatique du CEMAMu* [15]. Regarding CAC environments dealing with symbolic musical material, we must mention platforms such as PWGL[4] and OpenMusic[5]. There are also CAC environments that support real-time audio processing and synthesis. Beyond *Pure Data* or PD[6], the most notable example—and the one discussed in detail in Section 3—is Max/MSP[7]. With the development of various libraries and external applications, such as the *Bach* project [8], Max/MSP is well-positioned within the CAC context by offering a toolkit focused on symbolic data, in-environment graphical music notation, and interaction with MusicXML and MIDI file formats.

In summary, CAC, within the context of our study, represents a dynamic interaction between symbolic material generation, technology, and the creative process, using diverse techniques and technologies to expand the scope of interaction among the agents involved in our project [16, 17].

3. Considerations on the Implementation of the Patch

To implement the intended model — one that explores the territory of symbolic notation through transformations in a parametric space — it was necessary to select a software environment capable of supporting such operations. The choice of software involved a critical evaluation of the features available for symbolic notation, graphic manipulation, and mathematical operations. Another important factor was the availability of tools and functionalities that could support future extensions of the proposed model, such as sound synthesis and spatialization. Guided by these criteria, we opted for Max/MSP/Jitter, as it integrates all the necessary capabilities within a single software. This choice is not without limitations, as it is proprietary software. However, the existence of an analogous model implemented in OpenMusic — a free software — and the possibility of interfacing with other open-source software such as PureData ensures that the methods presented here are not restricted to the chosen environment. The adoption of Max was motivated by the desire to consolidate both the visualization and implementation within a single, cohesive software. [9].

[1] Available at: http://anasynth.ircam.fr/home/english/software/audiosculpt. Accessed on: 05/26/2025

[2] Available at: http://www.klingbeil.com/spear/. Accessed on: 05/26/2025

[3] Available at: https://www.sonicvisualiser.org/. Accessed on: 05/26/2025

[4] More information at: https://en.wikiversity.org/wiki/Music/Software/PWGL. Accessed on: 05/26/2025

[5] Available at: http://repmus.ircam.fr/openmusic/home. Accessed on: 05/26/2025

[6] Available at: http://puredata.info/. Accessed on: 05/26/2025

[7] Available at: http://cycling74.com/products/max/. Accessed on: 05/26/2025

[8] For more information, see https://www.bachproject.net/. Accessed on: 05/26/2025

[9] The patch is avaiable in: https://files.catbox.moe/d68bgj.maxpat

3.1. Specific Aspects of Malt's Patch Implementation in OpenMusic

The OMChaos library in OpenMusic offers four sets of functions: Orbitals, Iterated Function System (IFS), Fractus, and UTILS. Malt's work specifically employs objects from the IFS set to generate fractal proliferation. Within this set, the *make-w* object performs linear transformations over the x and y axes, enabling expansion and contraction, vertical and horizontal transposition, and rotation around both axes. In Malt's compositional patch, within the complete chain of objects and functions, the *make-w* functionality has no direct equivalent among Max's native objects. Therefore, replicating the same operational model in Max required the development of an analogous process. Other aspects of Malt's patch, however, can be implemented and even enhanced using Max's features, as discussed below.

3.2. Challenges in Implementing the Patch in Max

Considering the information extracted from Malt´s work, the proposed model can be summarized as the manipulation and transformation of the relationship between points in space and their symbolic notation equivalents, through three main operations:

1. A **Expansion** and **compression** of the x and y axes;
2. **Displacement** of points along the x and y axes;
3. A **Rotation** by 360 degrees around a defined center.

To efficiently implement these operations, objects from two external libraries — CNMAT and Bach — were employed. The *xydisplay* object from CNMAT is used to visualize the points and their manipulations graphically, based on their x and y coordinates. The Bach library is used to render the symbolic notation that is equivalent to the spatially manipulated points. In Figure 1, both the graphical and symbolic notations can be seen. Mind that the *bach.roll* object, used to generate symbolic notation, produces a proportional notation score, where the distances between notes are given by their durations in milliseconds. This object was chosen for its flexibility, real-time responsiveness, and extensive quantization options, which allow rewriting a given score in proportional notation into another one that makes use of traditional rhythmic notation. Additionally, *bach.roll* allows playback of the resulting pitch set, enabling immediate auditory feedback of the harmonic and rhythmic outcomes.

The operational chain of events begins by defining the set of pitches to be distributed graphically as points, along with their temporal distribution specified in milliseconds. Once the pitches are converted into spatial points, the transformations mentioned above can be applied and then converted back into symbolic notation — a cycle updated in real time for immediate visualization and evaluation. For this chain of events to occur, it was necessary to specify the parametric space for this conversion to take place on. We chose the *Argand–Gauss* plane — the complex plane — a geometric representation where the real

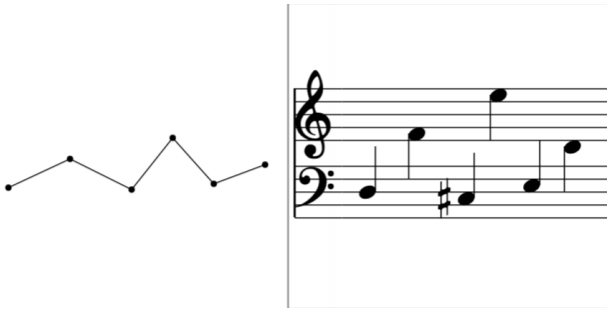

Figure 1: Visualization on the *xydisplay* and *bach.roll* objects

part (x-axis) and the imaginary part (y-axis) correspond to two complex mathematical components. This model was chosen because it offers an intuitive visualization of sums, products, and rotations in geometric form.

Visualizing the effect of operations in the graphical–mathematical domain and their impact on symbolic notation is central to the proposed model. To convert the graphical coordinates and operations into symbolic notation, the *xydisplay* object is configured to continuously send its coordinates to the processing chain, which then maps them to pitches within harmonic and rhythmic sieves. The model also supports the implementation of any harmonic and rhythmic sieves for generating point distributions and symbolic notation. In the examples shown here, we used a whole-tone harmonic sieve and a non-retrogradable rhythmic sieve, specified in microseconds. The choice of six pitches was arbitrary and maintained throughout the project, but the patch places no inherent limits on the number of pitches or points that it can utilize and process.

The initial graph is constructed by inputting the pitches and their relative temporal positions in separate lists, which the patch converts into Cartesian coordinates. This data is then sent to the *xydisplay* object, which renders the graph. Subsequently, the Cartesian coordinates from *xydisplay* are converted into polar coordinates — defining each point by its distance from the origin (r), its amplitude, and its angle ($\angle$) — which enables rotation. After applying angular rotation, the points are reconverted into Cartesian coordinates and sent back to the visualizer. This chained sequence of operations enables real-time actualization of the graphic visualizer and its symbolic counterpart.

The displacement of the points is performed along the x and y axes by adding or subtracting values within the same numeric metric, functioning as either pitch transposition or rhythmic offset, preserving the harmonic and metric structure of the set. The expansion and contraction are achieved by scaling the distances between points in a proportional order, resulting in a uniform and continuous displacement in opposite directions. Notably, the interplay of rotation and scaling can align the points horizontally or vertically, in which case certain axis-specific transformations will no longer affect the orthogonal dimension — for example, operations on the x-axis would not alter y-

coordinates when points are horizontally aligned. All the operations can be applied in a complementary way.

3.3. Patch Programming

The *xydisplay* object operates exclusively with Cartesian coordinates, as its name suggests. In order to implement the visualization of a set of notes, defined by pairs of MIDI pitch values and their corresponding onsets, these values must first be converted into Cartesian coordinates. Only then can they be represented as points within the *xydisplay* object. To achieve this, it was necessary to map them onto a scalar range between -1 and 1. This conversion requires the minimum and maximum values of the harmonic and rhythmic sieves, which, once the user defines the sequence of notes for the patch, are automatically computed within the patch's operator chain. However, to keep the system flexible and malleable, users also have the option to manually define these limits, allowing for deeper exploration of the possibilities offered by the operator set.

For a continuous graphic rotation to occur, a constant feedback loop between Cartesian and polar coordinate conversions is required to take place. This is achieved using the *poltocar* and *cartopol* objects, through which the information they receive is converted into appropriate messages for the *xydisplay* object to produce the graphic information it displays. For this to occur, the workflow implemented must comply with the following sequence of operations:

1. Input of Cartesian coordinate data into the patch;
2. Extraction of the exact same data;
3. Conversion to polar coordinates;
4. Manipulation of the data;
5. Conversion back to Cartesian coordinates to feed the viewer again.

This ensures that all operations are performed in a continuous flow, enabling real-time visualization. When working with polar coordinates, it is essential to represent angles in a minimal, unambiguous, and standardized way. Therefore, to avoid ambiguity and prevent numerical instability, it is required to normalize the angle within the range between π and -π. Hence, the following expression is used:

$$(\$f1 >= 0.) * (\$f1 - \pi) + !(\$f1 >= 0.) * (\$f1 + \pi)$$

The implementation was carried out using the *vexpr* object, which enables the application of mathematical operations on lists. This feature allows for the normalization of all points within the visualizer. The specific expression employed performs an angle adjustment operation that transforms the value $f1 into an equivalent value centered around zero by shifting it by $\pm$, -π, depending on its sign. This operation ensures that angular values, expressed in radians, are constrained within the range of $+\pi$ to $-\pi$. The resulting values are then routed back to the beginning of the data processing chain, enabling the rotation of each point's parameters in radians. For the patch interface, degrees were selected as the preferred unit for point rotation

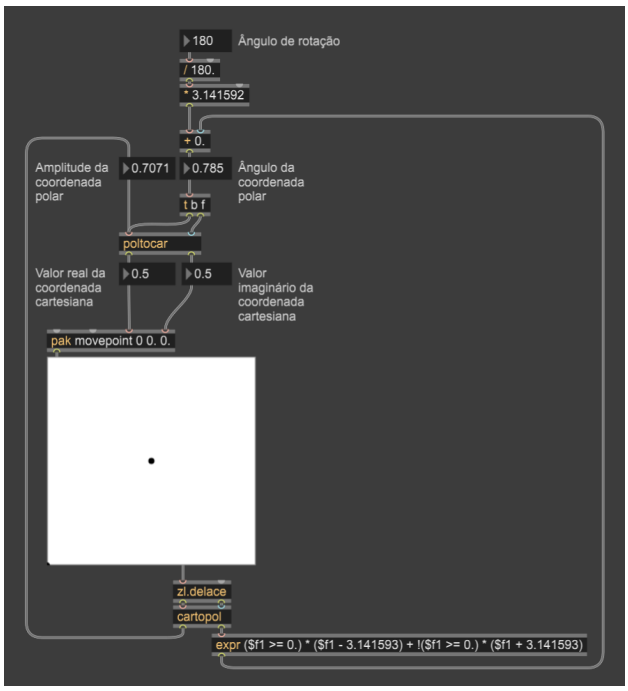

Figure 2: Conversion chain between polar and Cartesian coordinates

operations. To facilitate this, a *slider* object is used to select a value in degrees, which is subsequently converted to radians using the following expression:

$$\text{radians} = \text{degrees} \times \frac{\pi}{180}$$

The obtained value is then added to the previously registered value, thereby performing the rotation of the points. In order for the *xydisplay* object to transmit the coordinates of the points, as illustrated in Figure 2, which subsequently feeds the *zl.delace* object, it is necessary for *xydisplay* to receive numerical messages corresponding to each point inserted into it. This numerical sequence begins at zero, and a simple mechanism employing the *uzi* and *counter* objects is sufficient to carry out this task. To reference the points used in the examples in this article, the *xydisplay* object must receive a list of numbers between 0 and 5, thereby triggering the output of the coordinates of each point.

4. Discussion and Results

The final patch organization is susceptible to two modes of use:

1. **Transformation and manipulation of point and pitch positions for deferred-time use**, in which the data are extracted and reused according to the user's intent;
2. **Real-time application**, in which the generated transformations can be directed, for example, to sound synthesis or spatialization.

The following section presents examples of possible variations performed by the patch and their corresponding results.

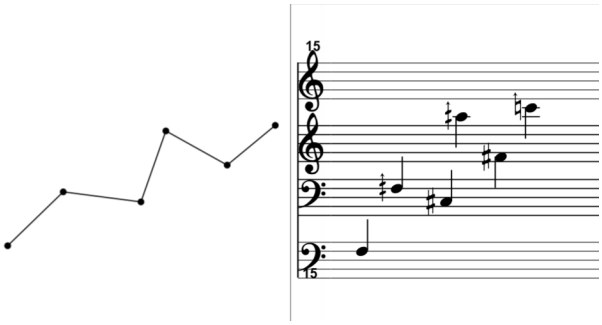

Figure 3: Rotation by 50°

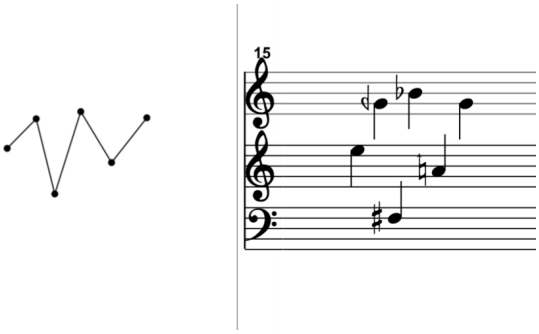

Figure 4: Expansion and compression

4.1. Rotation on X

As illustrated in Figure 2, the rotation of the set of points results in variations in both the pitch of the notes and their spatial positioning, wherein compression and expansion occur as inverse processes along the X and Y axes, respectively.

4.2. Expansion and compression of the spatial position and pitch

By isolating the processes of expansion and compression, it becomes possible to operate on one axis without affecting the other. In this way, compressions and expansions applied to the harmonic domain do not alter the rhythmic aspect, and vice versa. This can be observed in the example shown in Figure 3, where the X-axis is compressed to half of its original values, while the Y-axis is expanded to 1.5 times its original values.

4.3. Displacements in X and Y axes

The translation process of the point set operates linearly across all points, resulting in harmonic transpositions and rhythmic shifts. Figure 4 illustrates two distinct moments, in which the highlighted set demonstrates horizontal and vertical displacement around another set that remains static.

4.4. Example of non-Real-Time Use: Fractalization

Exploring a compositional process based on fractal models is also a core possibility of the patch presented here. The proposed model centers around the idea of juxtaposing and embedding sets of notes—lines constructed from

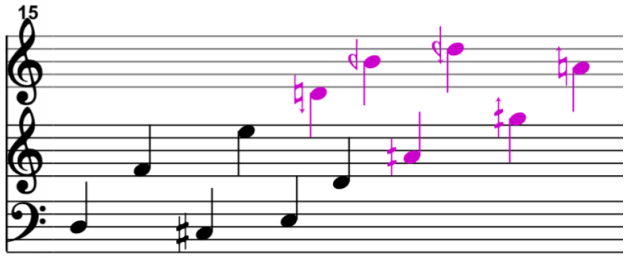

Figure 5: Vertical and horizontal displacement

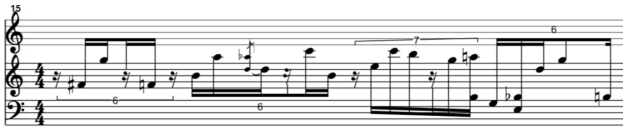

Figure 6: Fractalization quantization

a complex geometric structure—designed to preserve self-similarity across different scales. By varying the parameters available and exposed in the patch, it becomes possible to generate a series of configurations of pitch sets derived from a fractal logic.

As in Malt's original work, variations of the initial set are transposed into relative positions within the structure of the geometric figure itself, in which each of the six pitches from the initial set serves as a starting point for a new variation applied to that set. In other words, six new pitches are inserted from each pitch of a previous set. Figure 6 displays a section of a quantized construction based on the aforementioned approach.

5. Perspectives and Final Considerations

Throughout this article, we have presented different uses and possibilities for the proposed patch, aimed at musical creation. Based on the operations and models introduced, it is possible to outline preliminary considerations and identify promising avenues for future research. The model presented here demonstrates the feasibility of operations in the symbolic domain through graphical transformations. Although we consider the practical application of our proposal in compositional experiments to be essential, we also recognize the importance of addressing technical and implementation aspects. It is worth noting that the current workflow already incorporates technical development, the implementation of the programming architecture, and testing procedures carried out at each stage discussed in the context of this article. In future work, we aim to focus on the discussion of these developments within practical and creative musical projects.

The inclusion of a visual support component to complement mathematical operations such as rotation, compression, expansion, and displacement offers an alternative approach to compositional processes, going beyond purely mathematical abstractions.

Regarding the next stages of the patch's development, we highlight our interest in implementing nonlin-

ear progressions of compression and expansion, in order to provide greater plasticity to the operations available within the patch. A significant consideration for future developments lies in the patch's ability to operate transformations in real time, qualifying it as a versatile tool for use in sound synthesis techniques—whether through the continuous modulation of pitch or through jumps between distinct parametric positions. Temporal distances between points may be mapped onto parameters relevant to sound synthesis, thereby enabling control over envelope, amplitude, modulation indices, and timbre. Another anticipated functionality concerns the use of the patch for sound spatialization, where the set of points would control the movement of virtual sound sources within an ambisonic spatialization model.

References

[1] Miller Puckette. Preface: The om composer's book vol 1. *Editions Delatour/IRCAM Centre Pompidou*, 2008, 2006.

[2] Gérard Assayag. Computer assisted composition today. In *1st symposium on music and computers, Corfu*, 1998.

[3] Richard Parncutt. Systematic musicology and the history and future of western musical scholarship. *Journal of interdisciplinary music studies*, 1(1):1–32, 2007.

[4] Markus Schedl, Emilia Gómez, Julián Urbano, et al. Music information retrieval: Recent developments and applications. *Foundations and Trends® in Information Retrieval*, 8(2-3):127–261, 2014.

[5] Carlos Agon, Gérard Assayag, and Jean Bresson. The om composer's book vols. 1, 2, and 3. *Editions Delatour/IRCAM Centre Pompidou*, 1, 2, and 3, 2006.

[6] Mikhail Malt. Fractals and writing, six fractal contemplations. in: Agon, c.; assayag, g; bresson, j. (org.). the om composre's book. *Editions Delatour/IRCAM Centre Pompidou*, 2008, 2006.

[7] Michael Gogins. Iterated functions systems music. *Computer Music Journal*, 15(1):40–48, 1991.

[8] Alain Bonardi. Rejouer une œuvre avec l'électronique temps réel: enjeux informatiques et musicologiques. *Genèses musicales*, pages 239–254, 2015.

[9] Bill Manaris, Patrick Roos, Penousal Machado, Dwight Krehbiel, Luca Pellicoro, and Juan Romero. A corpus-based hybrid approach to music analysis and composition. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 839. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[10] Jean Bresson and Jean-Louis Giavitto. A reactive extension of the openmusic visual programming language. *Journal of Visual Languages & Computing*, 25(4):363–375, 2014.

[11] Jean Bresson. Reactive visual programs for computer-aided music composition. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 141–144. IEEE, 2014.

[12] William Buxton. A composer's introduction to computer music. *Journal of New Music Research*, 6(2):57–71, 1977.

[13] Otto Laske. Composition theory in koenig's project one and project two. *Computer Music Journal*, pages 54–65, 1981.

[14] Tristan Murail. Spectra and sprites 1. *Contemporary Music Review*, 24(2-3):137–147, 2005.

[15] Jean-Baptiste Thiebaut, Patrick GT Healey, Nick Bryan Kinns, and Q Mary. Drawing electroacoustic music. In *Proc. ICMC*, volume 8, 2008.

[16] Palle Dahlstedt and Peter McBurney. Musical agents: toward computer-aided music composition using autonomous software agents. *Leonardo*, 39(5):469–470, 2006.

[17] Mikhaïl Malt. La composition assistée par ordinateur, modèles et calcul, quelques éléments de réflexion. *Le calcul de la musique*, pages 163–224, 2009.