Demonstration of Different Functions in Terms of Fractal, Frequency Modulation and Vibrato based on Nyquist

Yanxi Li *

Guangzhou Foreign Language School, Guangzhou, China

* Corresponding author Email: liyanxi@chgzfls.com

Abstract. Nyquist is a compiler that was developed by computer researcher Roger B. Dannenberg, and it is frequently used in music composition and synthesis programming. The significance of investigating these three functions (fractal, frequency modulation and vibrato) in Nyquist is to introduce the basic principles behind each function, which could be then applied to other software as well. This paper presents three perspectives that could be achieved by the functions in Nyquist and illustrates their functions from illustrating the similarities and differences through graphs and descriptions. By combining pseudo codes and description, the distinction could be more apparently and comprehensively presented. During my research and development phase of Nyquist and its application, Nyquist Reference Manual is referred, with which scholars can basically achieve any effects in their scores or pieces by referring to it. These results can surely help substantially understand more about Nyquist, the principles behind its functions, and possible application the concepts into any other programming environments.

Keywords: Nyquist; Computer Music Composition & Synthesis; Vibrato; Frequency; Fractal.

1. Introduction

With the increasing popularity of composing computer composition and related composing software, musicians have started relying on user-friendly and intuitive platforms [1]. However, since computer experts have visualized the complex code behind these have been visualized by computer experts into buttons inside the software for all users, it is more intriguing for computer researchers interested in music to figure out the principles and fundamental operating logic behind them. It is also necessary to go in-depth inside the functions in order to obtain a more comprehensive understanding [2].

Computer music has a long history, which could be traced back to Ancient Greeks, when the concept of “harmony of the spheres” was claimed [3, 4]. Originally, combining mathematics and music was more popular than combining music with computer at that time, for people were more familiar with this application in practice while already combining mathematics with other fields. The first contemporary computational musical melody was created in 1950 in Australia. This was a huge progress for computer scientists and musicians since it was a completely brand-new concept. Afterwards, lots of shows and assemblies were held in towns to propagate their own work. In the 2000s, FM synthesizers such as Yamaha DX7, NEC PC-88 and so on began to emerge. Subsequently, more simplified programming languages began to appear and experts had made them into software that could be utilized by everyone. Computer music was developing drastically during these decades, including software and intact systems, with which even people who were not computer scientists could easily compose and manipulate their own music.

Sound can be interpreted through programmings in several ways, including using analogs, table clippings, and frequency tables etc. In Nyquist, it is based on an interactive lisp interpreter, and by providing SAL commands in the environment [4, 5]. Besides, the sole motivation of having this topic is to illustrate the detailed functions, and operating method in this field (inside Nyquist).

In order to improve the understanding and application scopes of the basic effects used in music, Nyquist, which is introduced in this paper, will be described specifically to help understanding better. In this paper, vibrato, frequency modulation and fractal music will be investigated based on Nyquist. The rest part of the paper is organized as follows: section 2 will demonstrate fractal music; section 3

will be demonstrating the various ways of frequency modulations; section 4 will present the use of vibrato in Nyquist. The reason for choosing these three aspects is that by combining practical uses of musical technique, these are the three most common and useful processing modes no matter in real performances or in signal processing. During the research and development phase of this paper and the investigation of Nyquist, primarily all the typical functions and types regarding these three aspects are listed for comparisons.

2. Demonstration of Fractal Music

2.1 Definition

Fractals are self-similar geometric shapes containing detailed structures at infinitely small scales in mathematics [7]. In music and composition with computer, the most important relating inspiration is the Sierpinski Triangle which was created by Rick Taube. To be more specific, he created layers that are recursively divided and transposed to imitate the geometry of the triangle. In this case, one can create Sierpinski triangles, squares, pentagons or even more impressing shapes. Since there is no range and time limits, people can build their music in Nyquist with more flexibility. In Nyquist, fractal music consists primarily of loops and judging statements about the sides of the shapes set initially. For instance, by creating a random score based on fractal ideas, one must first judge the number of the shapes and layers.

2.2 Random Fractal Music

Table 1. Random Fractal Pseudo Code.

Step 1	Define an “append-chord” function with parameters (chords, num)
Step 2	Judging statement: if num > 1, then set chord as variable with parameters.
Step 3	Set rand = {0}
Step 4	Define a “polygon-fractal” function with parameters (dur, pitch, layer, tim, num)
Step 5	Set score as a variant with parameters
Step 6	Judging statement: if layer > 1, then set transpose-pat and dur as new variables
Step 7	Looping from i to 0 under num, set score as a new variable with parameters
Step 8	Play score-play

Table 2. Harmonious Fractal Pseudo Code.

Step 1	Define an “append-chord” function with parameters (chords, num)
Step 2	Judging statement: if num > 1, then
Step 3	Loop from var to 1
Step 4	Judging statement: if var divided by 2 is larger than 0
Step 5	Set odd and chord as two new variables with parameters
Step 6	Judging statement: if var divided by 2 is smaller than 1
Step 7	Then set even and chord as two variables with parameters
Step 8	Set chord = {0}
Step 9	Define function “polygon-fractal” with parameters (dur, pitch, layer, tim, num)
Step 10	If layer > 1, then set score with new variables
Step 11	Play score-play

Table. 1 gives the code flow for the random function. The first aim is to create a random scale score. The chords in this piece might be pleasant, but they might weaken the connection between fractals and the music sample. To create a sample completely generated by fractals, random algorithms are imported in the rand-append function to make the pitches of the notes random. In this case, a tiny bit of manipulation is added to make the first note the base note so that we can identify the period of the music sample, which is a prominent relation between our music and fractals. In

addition, it ensures that the overall melody goes from the base note all the way to the peak by a small manipulation to imitate the fractals, from a basic geometric shape to infinitely small structures.

2.3 Harmonious Fractal Music

Table. 2 gives the code flow for this function. The second example is about the harmonic fractal music, which is primarily constitutes of chords and other harmonic elements in conventional music. The first attempt was to try to model the sequence with harmonious chords. In the harmonics cod e sample, a new function called append-chord was added. This function distinguishes odd and even inputs in a "for loop" to create harmonious seventh and octave chords above the base notes, so that no matter whatever the melody the fractal creates, the overall effect is always as harmonious as that produced by conventional music .

2.4 Combination of Fractal Music

Table. 3 lists the code flow for this function. The last example would be the combinations between these harmonics and randoms. In this fraction of code, it distinguishes the odd and even parts to form the first five notes in a layer similar to the structure of a major 9 chords as well. Moreover, it does not follow any music patterns or theorem rules, and the melody becomes dependent only on the fractal. Combination of fractal music might generate inharmonic music from conventional music aspect, but computer musicians would not pursuit the so called "harmonic music" blindly. Instead, they would generate music that has real meanings with code than simply producing music like harmonic classical or Baroque music.

Table 3. Combination of Fractal Music.

Step 1	Define an "append-chord" function with parameters (chords, num)
Step 2	Judging statement: if num > 1, then
Step 3	Loop from var to 1
Step 4	Judging statement: if var divided by 2 is larger than 0
Step 5	Set odd and chord as two new variables with parameters
Step 6	Judging statement: if var divided by 2 is smaller than 1
Step 7	Then set even and chord as two variables with parameters
Step 8	Set chord = {0}
Step 9	Define function "polygon-fractal" with parameters (dur, pitch, layer, tim, num)
Step 10	If layer > 1, then set score with new variables
Step 11	Loop from i to 0 under num
Step 12	Set a new variable score-merge with parameters
Step 13	Play score-play

3. Demonstration of Frequency Modulation

3.1 Defination

Frequency modulation is extremely important in both composing and performing in music, particularly used for dynamic modulation. In regular music performances and compositions, there are specific notes like *forte*, *piano*, *crescendo*, *diminuendo* etc. in scores in order to convey the idea of dynamics modulation. In Nyquist, people must get familiar with the concept unit generator just to reach the various effects mentioned above. Unit generators are initially created to construct synthesis and signal algorithms in Nyquist and other composing software. Examples of unit generators could be oscillator, envelope, and multiplier, which is frequently used in music composition in order to simulate the real effect in real performances. At this point, lazy evaluation can reveal its advantage. Nyquist consists a clever implementation to deal with large commands, which is to store the commands and statements into small chunks that are utilized by other unit generator, while they can

be quickly deleted to conserve memories after the transformation. If information or memories are needed to recall, they could be found inside the chunks and this would only use up little memories. To be more specific, if one writes f (c, d) in Nyquist, then Nyquist would evaluate c and d and convey the resulting value to f. Unit generator is not a single proper noun in Nyquist, but this concept could also be applied in other language and operating environment such as MAX MSP. The following demonstration and illustration of code are mostly all based on unit generator.

3.2 Waveform

According to the description of unit generator above, there are several functions in Nyquist to reach the effect of dynamics modulation. The first method will be inserting an waveform. Those specific waveform sums up the harmonics. Table. 4 presents the code flow for this function.

Table 4. Waveform Pseudo Code.

Step 1	Define a “mkwave” function
Step 2	Set *form* with parameters (For instance, the sample demo is shown with code: 0.5 * build-harmonics (1, 1024) + 0.25 * build-harmonics (2, 1024) + 0.125 * build-harmonics (3, 1024) + 0.0625 * build-harmonics(4, 1024)).
Step 3	Convert form into a list
Step 4	Play the mkwave function

In this pseudo code example, function mkwave() calls upon build-harmonics to generate the four amplitudes {0.5 0.25 0.125 0.0625}. Then, they are added and scaled in *form* temporarily. In addition, since the default signal is 1 second, the hz-to-step() function could convert the alternating signals into pitches. The last two lines of the code first build and run the mkwave() function as a side effect, then use the *form* to play it as a whole.

3.3 PWL Function

The second method of generating dynamics alternation in notes and pieces is to insert pwl function (piece-wise linear function) in Nyquist. In pwl functions, it consists with various time and value breakpoints which generates a smooth note. Computer scientists often utilize the concept of pwl (not exactly call pwl in other operating environment or software, but is applied for similar effects) to make their notes sound more professional in practice. Initially, the sound without pwl function would end with a beep, which sounds amateur. However, with the pwl function, sounds could quietly begin and smoothly end in the end. The specific code example is shown with code: plot pwl (0.5, 1, 0.75, 0.4, 0.95, 1, 1), which the corresponding graph is shown in Fig. 1. This is a constant function with the value 0 over the whole-time interval. Sounds would start to increase in time 0.5 to the intensity 1, then at time 0/75, the dynamics of the sound would alter to 0.4 in intensity. Next, it reaches a full intensity which is 1 in time 0.95, then the sound disappear thoroughly in time 1. When this signal is visualized, it looks like the case given in Fig. 1.

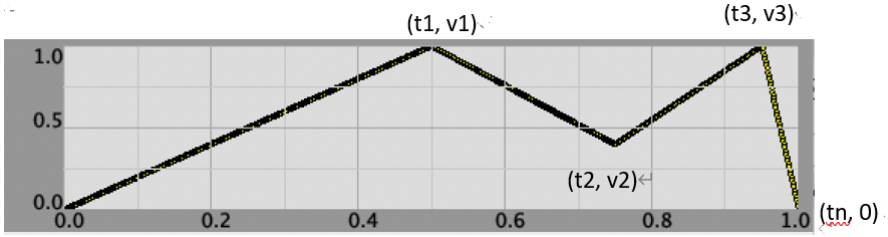

Fig 1. Plot pwl function

Pwl has three other variants as well, including pwlv, pwev, pwlr. According to Nyquist Reference Manual, the syntax of pwlv is (v0, t1, v1, t2, v2, ..., tn, vn). This function differing from the pwl function is that it is used to conduct signals that is non-zero startings and endings. and one specific example is shown in Fig 2 with code: plot pwlv (0.3, 0.3, 0.6, 0.7, 1). Seen from the results, the

starting point and the ending point are not zero, at time 0.3, the dynamics is 0.3 and at time 0.6, the dynamic intensity is 0.7. At last, this function ends with the upper bound 1.

Another function could be pwev. The syntax of pwev is (l1, t2, l2, t3, t3, ..., tn, ln), which is an exponential interpolation. The specific code example is shown below with code: plot pwev (0.2, 0.2, 0.4, 0.5, 0.9, 1), which the corresponding results are illustrated in Fig. 3. The last variant of pwl is the pwlr function. Its syntax is pwlr (i1, l1, i2, l2, ..., in), which is similar to the previous ones. Pwlr functions have relative intervals rather than absolute intervals. The following code example is a diminuendo example rather than crescendo one. The specific code example is shown below with code: plot pwlr (0.3, 1, 0.7, 0.7, 1), where the corresponding results is given in Fig. 4.

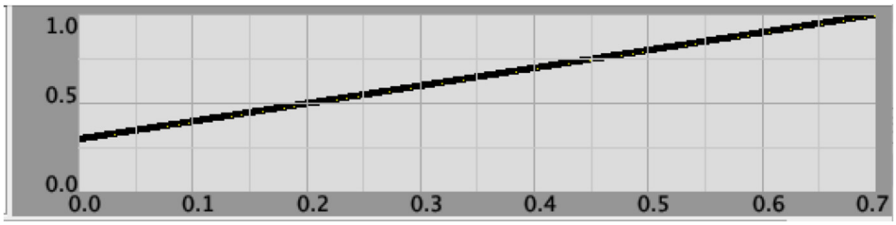

Fig 2. Plot pwlv function.

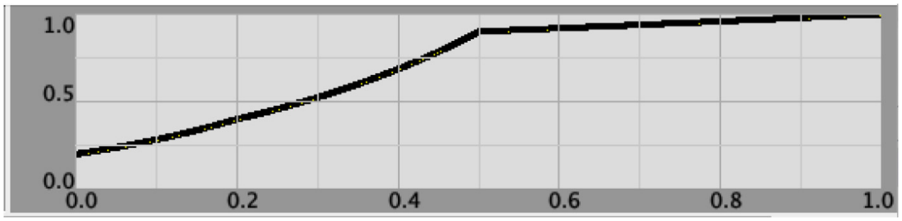

Fig 3. Plot pwev function.

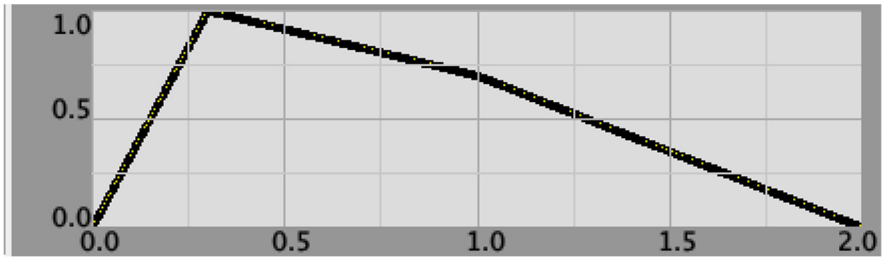

Fig 4. Plot pwlr function

3.4 Envelope

Envelope (env) is also a function based on unit generator as mentioned before. Its syntax is env (t1, t2, l1, l2, l3, duration). The most significant advantage of envelope over pwl and the similar variants is that it allows unchanged attack and decay time, which would not alter with the change of duration. According to the Fig. 5, the first-time breakpoint is 0.3 seconds, the second one is 0.4 seconds, the last (fourth breakpoint) is 0.8 seconds. The l1 is 1, l2 is 0.8 and l3 to be 0.4. The whole piece lasts for 2 seconds. The specific code example is shown below with code: plot env (0.3, 0.4, 0.8, 1, 0.8, 0.4, 2) as depicted in Fig. 5.

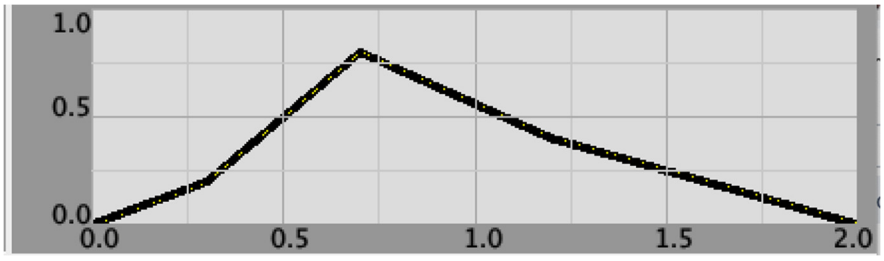

Fig 5. Plot env function

4. Demonstration of Vibrato

4.1 Defination

Vibrato is a common performing method in music, including instrumental performances and vocal performances. Each instrument groups in orchestra or bands have their own methods of producing sound with vibratos, even in vocal performances. In woodwind, instruments produce vibrato through using breaths to vibrate the air column inside the instrument. In fact, the principle behind the bronze instruments generating vibrato is similar. However, in string groups, they modulate their vibratos by pressing the strings in different intensities. The vibrato would change as long as the intensity alters.

4.2 Vibrato in Width

In order to generate vibratos in single pieces, it is easy for people to achieve it by using the concept amplitude modulation. Thus, the first way is to add lfo, which stands for low-frequency oscillator. However, one of the largest features of lfo is that it does not start from zero, so the fluctuation would be huge and the vibrato would sound super unnatural. In code, the object cannot simply times lfo to the parameters, otherwise it would occur the effect mentioned before. Besides, the signal expressions could be multiplied by the envelope expressions mentioned above to create more natural sound. The specific code example is exhibited in Fig. 6 (left panel) with code: play (pluck(c2)×lfo(6))~4.

This command makes a pluck sound and multiply by lfo (6). It produces a rapid amplitude vibrato, it is 12 Hz because the 6 Hz. The sinusoid generated by lfo (6) alternates to the positive and negative scope. Each "swing" acts as a short envelope. There are 12 of these so called envelopes per second, so this generates a 12 Hz vibrato. At the end of this command, it means stretching the whole note 4 times. In contrast, "@" means compressing the notes. Sometimes people use to achieve certain audition effect, but at other times, those are used for debugging, for finding the problems more directly by stretching the whole note and listening to the pieces more intact and neat by compressing it. Those are the two functions that are frequently used in Nyquist.

Now, as mentioned above, it would sound unnatural if simply multiplying pluck notes to lfo. Thus, the following code is the modification version after a series of attempts. By adding one to the lfo function, the piece will sound more peaceful. The specific code example is shown in Fig. 6 (right panel) with code: play (pluck(c2) × (2 + lfo(6))) ~ 4.

This line of code turns out to output a slower amplitude vibrato comparing to the one above. It is 6 Hz but the expression 2 + lfo (6) varies in amplitude from 1 to 3, and the variation happens 6 times per second. Hence, the vibrato is only 6 Hz. Comparing to the code written above, the vibrato is much more slower which sounds more natural. After the illustration of each two code demos by using the same function, the comparison has to be more directly shown by using graphs. The following first graph is corresponding to the first code demo while the second graph corresponds to the second demo.

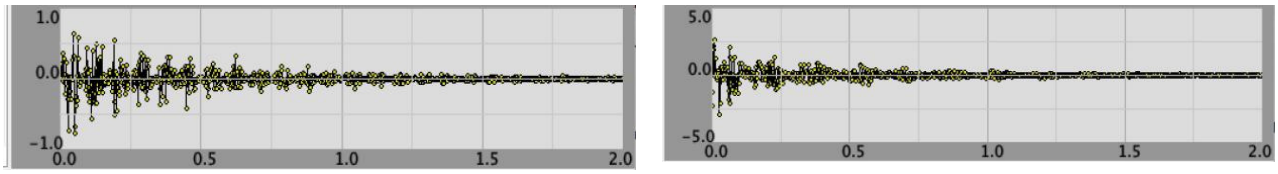

Fig 6. Plot low-frequency Oscillators.

For instance, vibrato could happen among every timbres in Nyquist, including oscillators, just as shown below. The specific code example is shown in Fig. 8 with code: play osc(c5) ×(0.4 + 0.2×lfo (5)). This command creates a sinusoidal piece in geometry thanks to the characteristic of oscillators. The output from this code demonstration (signal) vibrates at 5 Hz and does not return back to zero due to the vibrato expression contains an offset of 0.4.

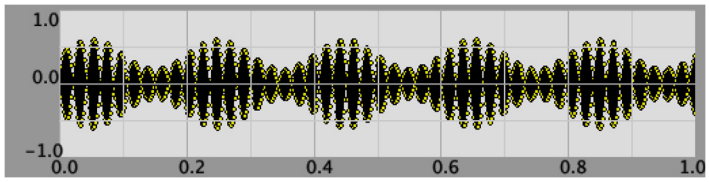

Fig 7. The output from osc(c5) × (0.4 + 0.2×lfo (5)).

4.3 Vibrato in Depth

Vibrato could be altered through crosswise, as well as through its vertical dimension. In this case, people can set up a depth function to alter the depth of vibrato. After creating a depth, one only needs to ensure that the depth is less than 1 and time depth to the vibrato expression. The example results are given in Fig. 8 with the code: function depth() return pwl (0.5, 0.9, 1, 0.9, 1); exec s-plot(depth()); play pluck(c2)×(1 + depth() × lfo(6))) ~ 4. The first code shown below shows shallower vibrato comparing to the second one: (1)play (pluck(c2)×(1 + 0.15× lfo(6))) ~ 4; (2)play (pluck(c2)×(1 + 0.9×lfo(6))) ~ 4.

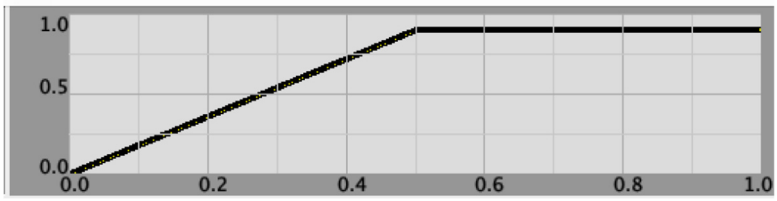

Fig 8. Plot depth of vibrato

To sum up, amplitude modulations and frequency modulations are frequently used in daily lives. The frequency modulation is a modulator whose amplitude controls the depth of the modulation. To be more specific, people always need a large amount of modulation, and they will need scale modulation as well as by inserting some particularly categories of envelopes.

5. Limitations & Prospects

Nyquist is the main programming language throughout my whole research and development. The disadvantage of using Nyquist is that since Nyquist is a unique programming language, the syntax and command are slightly different comparing to programming in other languages (e.g., MAX, MSP). In addition, since Nyquist is a self-built software with limited resources and plugins, it shows the lack of timbre comparing to normal composing software. Particularly for self-conducting software, Nyquist requires a strong ability to build up one's own timbre, which is hard to unify if the timbre is needed to be recalled in the future or shared as a standard timbre among people. Moreover, with the emergence of NIME [8], which stands for "New Interfaces for Musical Expression", computer scientists have started exploring its principles and algorithms behind instead of physical compositions by the 2000s. However, Nyquist could only satisfy needs to composite designated notes and scores based on simple code that could be referred in the reference menu. Advanced effects like NIME cannot be achieved through Nyquist. For instance, all of the pieces and notes were built by commands and statements, without those statements, however, the melody would automatically stop and break. With NIME technique, melodies could continue to be built on based on the previous code with analysis rather than the exact commands with precise parameters. In order to reach this effect, Nyquist requires more complex algorithms which could be improved in the future. ISMI (Music Information Retrieval) [9], so as NIME, is also a popular topic nowadays that Nyquist should work on. In the future, Nyquist should work on more automation production instead of repetitive commands typed by people with code. The most optimal expectation is to generate music with NIME inside Nyquist, and also with other technique such as ISMIR, from investigating timbre to algorithms [10].

6. Conclusion

In conclusion, this paper investigates vibrato, fractal music and frequency modulation based on Nyquist. Specifically, three pseudo code demos are demonstrated in generating fractal music, three function examples used to described frequency modulation, and two used to illustrate vibrato. Inside the analysis, different methods and their effects were distinguished and compared in order to make clearer distinctions and their optimal places to apply. According to the analysis, fractal music could be separated into random fractal, harmonious fractal and combined fractal music with mathematical analysis and proves. Vibrato could be categorized through two dimensions, i.e., horizontal (in width) and vertical (in depth). Before introducing frequency modulation, unit generator was introduced to make clearer clarifications of the following command and their meanings. Frequency could be adjusted from envelopes, pwl functions including its four variants, and also waveform to make adjustments of sound. In the future, ISMIR, NIME and unified timbre using more algorithms are the striving directions for Nyquist. Overall, these results offer a guideline for music composition with code in general.

References

[1] Gareth Loy, Curtis Abbott. Programming Languages for Computer Music Synthesis, Performance and Composition. ACM Computing Surveys, 1985.

[2] Joseph Akins. An Overview of Electronic Music History.

[3] Naotoshi Osaka. Cyberworlds: Sound Synthesis in Computer Music. NTT Basic Research Laboratoriesm, 1998.

[4] Ge Wang. Cambridge Companion to Electronic Music: A History of Programming and Music. Cambridge University Press, 2008.

[5] Roger B. Dannenberg, Retrieved from: http://www.cs.cmu.edu/~rbd/.

[6] Roger B. Dannenberg, Nyquist Reference Manual. Carnegie Mellon University, School of Computer Science, 2022.

[7] Brothers Technology, LLC. Fractal music, 2021.

[8] Dobrian, Christopher, Koppelmann, Daniel. The “E” in NIME: Musical Expression with New Computer Interfaces, UC Irvine ICIT, 2006.

[9] Jouni Paulus, Meinard Müller, Anssi Klapuri. State of the Art Report: Audio-based Music Structure Analysis, Tempere University of Technology, Finaland, 2010.

[10] Quanwei Fu. Research on the Use of Computer Music in Modern Musical Composition. Journal of Physics Conference Series 1820(1): 012153, 2021.