CATMASTER AND "A VERY FRACTAL CAT", A PIECE AND ITS SOFTWARE

Fernando Lopez-Lezcano
CCRMA, Stanford University
nando@ccrma.stanford.edu
http://ccrma.stanford.edu/~nando

ABSTRACT

In this paper I describe the genesis and evolution of a series of live pieces for a classically trained pianist, keyboard controller and computer that include sound generation and processing, event processing and algorithmic control and generation of low and high level structures of the performance. The pieces are based on live and sampled piano sounds, further processed with granular and spectral techniques and merged with simple additive synthesis. Spatial processing is performed using third order Ambisonics encoding and decoding.

1. INTRODUCTION

This series of piano pieces, starting with "Cat Walk" at the end of 2008, and currently ending with "A Very Fractal Cat, Somewhat T[h]rilled"[1] (last performed in concert in May 2010) was motivated by a desire to return to live performance of electronic music. As a classically trained pianist I was interested in exploring the capabilities of "augmented pianos", and the use of algorithms in the context of an evolving, interactive performance piece that also used virtuoso gestures from the performer (other examples include pieces by Jean Claude Risset[5] and Andy Schloss and David Jaffe[6]).

Between 1994 and (roughly) 1999 I was also involved with real-time performance of computer music but using a custom version of the Radio Drum as a 3D controller (the program was "PadMaster", written in Objective-C on the NeXT platform, see [11] and [12]). The amount of processing and algorithmic control I could use was limited by the capabilities of the NeXT, as the program could barely play two stereo sound files while controlling three external synthesizers and interfacing with the RadioDrum through MIDI. There was not much power left to create notes algorithmically and while that was the eventual goal of a next version of the program, it never happened.

This is a return to a very similar goal, with computers that can do a lot more, and using the first controller I learned to use effectively, a piano keyboard.

The piece uses an 88 note weighted piano controller as the main interface element of the system (the two lowest notes in the keyboard are used as interface elements, and the rest of the keyboard is available for the performance of the piece). The piece requires a keyboard controller with both a pitch bend and modulation wheels, four pedals (the usual sustain pedal plus three additional control pedals), and an 8 channel digital fader box (BCF2000 or similar) that is used by the performer to change the balance of the different sound streams during the performance.

A computer (either laptop or desktop) running Linux provides all the sound generation and algorithmic control routines through a custom software program written in SuperCollider ("CatMaster"), and outputs either a 3rd order Ambisonics encoded stream, or an Ambisonics decoded output for an arbitrary arrangement of speakers. The piece should be played with a diffusion system that can at a minimum support 5.1 channels of playback.

2. THE PIECE

The CatMaster program gives the performer a framework in which to recreate and rediscover the piece on each performance.

At the algorithm and gesture level the program provides a flexible and controllable environment in which the performer's note events and gestures are augmented by in-context generation of additional note events through several generative algorithms. The performer maintains control of the algorithms through extra pedals that can stop the note generation, allow the performer to "solo", and change the algorithms being used on the fly.

At the audio level the original sounds of up to five pianos (recreated through Gigasampler libraries and/or through MIDI control of a Disklavier piano) is modified, transformed and augmented through synthesis of related audio materials, and various sound transformation software instruments.

Finally the resulting audio streams are spatialized around the audience and routed to audio outputs in a flexible manner that allows for the piece to be performed in a variety of diffusion environments.

[1] The reference to "cats" in the title of the pieces refers to the proverbial cat walking and dancing on the keyboard of a piano

Copyright: © 2010 Fernando Lopez-Lezcano. This is an open-access article distributed under the terms of the Creative Commons Attribution License 3.0 Unported, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

2.1 Score vs. program vs. piece

In this particular piece there is no separation between the program itself and the piece. “CatMaster” was not designed as a general purpose program and all its code evolved from concrete performance and artistic needs of the performer (which evolved through many sessions and concert performances).

The overall form of the piece is defined in the program through “Scenes” (see below for more details). Each scene changes the overall behavior of the program and suggests certain behaviors and gestures to the performer through a text field in the GUI. The performer switches scenes manually through the two lowest keys of the keyboard controller and thus controls the timing of the musical discourse, but the overall form and behavior of the piece is pre-composed.

On the other hand the performer is not tied (yet, if ever) to a common music notation score in which all the notes are written down. While in theory this gives him/her complete freedom to improvise, in practice each scene or section of the piece has definite behaviors, gestures, and rhythmic and intervalic materials associated with it.

In future versions the program should provide more guidance to the performer than it currently does. The GUI should include a graphical score view so that each scene transition provides the performer with further instructions on the notes, intervals and gestures to perform. This would make it easier to open the piece to other performers, something which has not happened so far.

This balance between free and directed improvisation with overall control of the form is similar to the approach taken while writing the PadMaster program [11, 12].

3. CHOOSING AN ENVIRONMENT

A very important early decision was choosing an adequate computer language and development environment for writing the program. There were several requirements:

- a complete text based computer language (the author has a strong programming background, and anticipated the software would grow into a very complex program)
- preferably an integrated environment that can deal with MIDI, OSC, a GUI and audio using the same language
- support for multiples threads of execution and for multiple tempos and internal clocks
- very efficient audio generation and processing
- support for multicore processors so that audio processing and generation can use all cores when available
- has to run under the Linux operating system (the author's platform of choice)

There are no options to the author's knowledge that satisfy all requirements. The one that was finally selected was SuperCollider[8] as is the one that best matches the requirements.

Other systems were considered. Pd was discarded as it was anticipated that visual programming would not be the best fit for a very complex program (it would quickly become hard to debug and extend). ChuCK is currently not as feature-rich as SuperCollider and although its sample-by-sample audio processing is very useful for audio algorithm design, it leads to inefficiency in processing and synthesizing audio. The potentially more efficient approach of writing the software directly in C or C++ (PadMaster was written in Objective-C) was also discarded as it would involve gluing together several independent libraries to achieve the same results as SuperCollider.

Regretfully, as all other computer music languages at the time of this writing (except perhaps for the Faust compiler[14]), SuperCollider can't use multiple cores which are now standard in most computers. But a workaround is available because the SuperCollider language (sclang) is independent of the synthesis server (scsynth). They are two separate processes which communicate through OSC. It is possible to start more than one synthesis server to better utilize the capabilities of the underlying hardware and have all instances controlled through the same sclang language executable.

Tim Blechmann's supernova[13] synthesis server for SuperCollider is currently starting to provide experimental multiple core support with automatic load balancing between processors, and will hopefully be integrated into SuperCollider in the near future and used by this piece.

3.1 Other software

While SuperCollider provides most of the software needed through a custom program, several other open source software packages are used in the piece.

At the core of all the audio processing is **Jack** [4], a very low latency sound server that can connect applications to each other and to the sound card. Additional software includes:

- **Linuxsampler**: is used to generate the main ingredient of the piece, piano sounds (from four different Gigasample sound fonts) [1].
- **Jconvolver**: used as a reverberation engine with an Ambisonics first order impulse response [2]
- **Ambdec**: the Ambisonics decoder [3].

Some external utilities such as *amixer*, *jack_lsp* and *jack_connect* are also used. All external programs are automatically started and monitored by the CatMaster SuperCollider software.

4. CURRENT STRUCTURE OF THE PROGRAM

The program is event driven. Each event received from the keyboard controller, pedals or fader box activates a *Routine* (an independent thread of execution in the Su-

perCollider language) that processes the event, potentially spawns other *Routines* and eventually terminates.

4.1 High level control of the form

All the behaviors and parameters that are described in this paper can be changed dynamically by the performer. The change is done indirectly through *Scenes* that group sets of parameters and behaviors. The performer can step back and forth through the *Scenes* that make up the entire piece using the two lowest keys in the keyboard controller (which are not connected to sound generation), and change the overall response of the program to events arriving from the various controllers.

The collection of Scenes creates a predetermined (or precomposed) overall form for the piece. But the performer is free to navigate them differently in each performance and there is no fixed constraint in the duration of each section. In practice each section, through an iterative process of improvisation and discovery, has a definite feel in terms of gestures, rhythms and intervalic material that the performer uses in concert.

The gradual addition of features to the program has slowly created new sections of the piece (which have been explored through many performances), and the program itself has been modified extensively as a result of the performance experience, adding algorithms and features. It is an iterative process of refining both the artistic performance and the software being used.

4.2 NoteOn / NoteOff events

NoteOn and NoteOff events are the most important and drive most of the performance of the piece.

Every NoteOn and NoteOff event received is immediately sent to the appropriate “main” piano in Linuxsampler. Currently the two main pianos (a Steinway and a Bosendorfer) are spatialized statically int the front of the stage and each receives (statistically) 50% of the notes directly played by the performer.

A Cage prepared piano is also used sparingly is some sections of the piece, and the probability of notes being sent to it can be defined statically in each Scene or can be changed gradually when a trigger event happens.

After the performed notes are sent directly to the pianos, chords are detected with a simple timeout based algorithm, and if a given note is outside a chord an analysis function is run that trains second order Markov chains on the fly, looking for pitch intervals, duration of notes, rhythmic values and note loudness. Durations and rhythmic values are quantized to a pre-selected collection of values before training the chains, enforcing a rhythmic structure on the piece, regardless of the precision of the playing of the performer.

At the beginning of each performance the Markov chains start from an “empty” state and are filled as the performer plays notes. The program constantly learns transitions from the performer as the piece unfolds.

The Markov chains are later used as sources for various functions that generate algorithmic parameters for notes and phrases.

After the analysis is done an algorithm routine is run that determines the creation (or not) of additional note events.

4.3 Note generation algorithms

Note generation algorithms are Routines that get spawned by the NoteOn event and run asynchronously from the rest of the performance. The algorithms used for each parameter of the generated notes, the overall tempo (and tempo change) and the number of additional notes generated can all be controlled through Scene parameters, or in some cases directly by the performer (for example the modulation wheel changes by default the number of events generated in almost all algorithms).

4.3.1 Markov chains

This algorithm uses data derived from the Markov chains being trained by the performer. The pitch intervals come directly from the corresponding chain. Rhythm, duration and loudness of notes come either from a set of multiple predefined patterns or from the corresponding Markov chains, and which one is the source is determined from programmable random functions. Every note played by the performer potentially adds layers to the sound texture being generated, with a mix of in context and out of context notes. The artistic goal is to provide a feel of unity to a given segment of the piece, with additional surprises for the performer in the form of unexpected algorithmic materials being inserted into the piece.

The chains start with no content and thus the algorithms can't generate notes. As the performance progresses there is a point in which the software judges there is enough information accumulated, and starts to enable the algorithm.

4.3.2 Fractal melodies

This algorithm uses a fractal melody generator based on self similar melodies stacked in pitch and overlapping in tempo (loosely based on the Sierpinski triangle fractal curve examples in *Notes from the Metalevel* [7]).

The pitch material (a chord) for each triggered fractal melody is derived from the intervals in the intervals Markov chain, and a fractal is only triggered if the melody contains enough non-zero jumps in pitch, so this can only start happening after a fair number of notes have been played and analyzed.

4.3.3 Scales

This algorithm generates scales going up or down in pitch with parameters that determine note jump interval, direction of the scale, and total number of notes generated by the algorithm.

4.3.4 Trills

This algorithm generates a short scale that goes up or down in pitch with a programmable step from the performed note, and then a trill with a programmable interval and duration.

4.4 Controlling the algorithms

Which algorithm is active and its parameters can be selected through variables that can be defined in each Scene. One of the four performance pedals is also dedicated to algorithm control and serves a dual function.

When it is *up* the “normal” algorithm defined in the current scene is executed. When it is *down* the fractal melody algorithm is selected regardless of other parameters (and that is because the fractal melodies have an important role in the piece).

4.5 Stopping the algorithms

The up to down *transition* in the state of the algorithm pedal immediately terminates all currently running algorithms. During a typical performance this pedal is used constantly to select how additional notes are created, to control the thickness of the textures that are generated and to create abrupt transitions in the form of the piece.

An additional pedal is dedicated to a *solo* function, when pressed subsequent notes played do not spawn more note generation threads enabling the performer to play solo notes, melodies or chords without any algorithmic additions, or to play solo over a texture of algorithmically generated notes (the “solo” pedal changes in state do not stop currently running algorithms).

Between the two pedals a wide range of behaviors can be instantly controlled by the performer.

4.6 Balancing the sound

A fourth expression pedal (a continuous controller pedal) is used to control the volume balance between notes that the performer plays and are sent directly to the pianos, and all other notes generated by algorithms. In that way the performer can get the spotlight, so to speak, or the algorithms can jump to the forefront of the sound stage, all controlled live by the performer.

4.7 Pitch bend

The pitch bend wheel is also processed by the program and is used in a section of the piece to create micro tonal textures. The pitch bend wheel in the controller bends one of the main pianos up and the other down in mirror amounts, while the Disklavier and the other software pianos maintain the center pitch. Bends can create subtle beatings, or be used to play arbitrary micro tonal notes.

5. SIGNAL PROCESSING

A second dimension of the piece is the live digital signal processing of the piano sounds. This includes transformation of the sounds through granular and spectral techniques and the addition of synthetic sounds in some sections of the piece.

5.1 Recording and granulation engine

The first addition to the signal processing subsystem was a retriggerable sound recording engine that can store up to 5 minutes of sound per piano channel, and a matching granular synthesis instrument that can be triggered by incoming note events and reads its source material from the recent past of the live sound recording of the pianos.

Several parameters of the live granulation process can be controlled through Scene changes and one fader in the fader box is dedicated to controlling the loudness of the granulation instrument outputs.

5.2 Spectral processing

An instrument that implements fft based processing of the piano sounds was also written. It uses conformal mapping, bin shifting and bin scrambling unit generators in the frequency domain followed by an ifft to go back to the time domain. Several parameters of the frequency domain processing are currently controlled by the pitch bend and modulation wheels so it is possible to change the nature of the processing quite drastically in real time.

A second fader of the fader box is assigned to control the volume of the spectral processors.

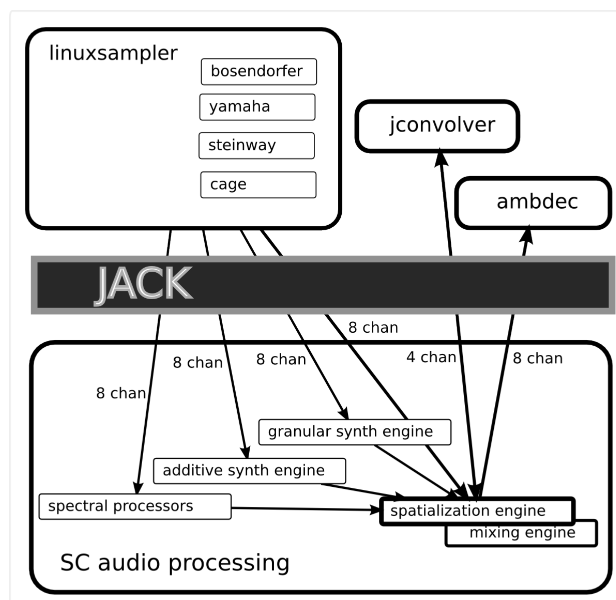

Figure 1: Audio routing overview

5.3 Fractals and sine waves

When a fractal melody is being generated, a certain definable percentage of notes will also trigger a software in-

strument that includes dual beating sine waves with a pitch envelope that will augment some of the partials of the piano sound. The sine instrument has a simple triangular envelope so that the sound does not mask the original attack of the piano notes but rather creates a “wash” of sound that prolongs the notes. A variable controls the density of the sine wave textures (ie: how often they are triggered for each new note) and can be changed through Scene changes.

As before a dedicated fader controls the overall volume of the sine generators.

6. SPATIALIZATION

Another dimension of the piece is the spatialization of all the sonic materials. All audio streams are independently rendered through 3rd order Ambisonics encoders. The spatialization engine provides static and dynamic routing of incoming audio with dedicated sends to a convolution based Ambisonics impulse response reverberation (implemented with the Jconvolver program).

The two main pianos are statically panned left and right at the front of the stage image, without reverberation.

There are four sets of autopanners that move sound streams around the audience in elliptical trajectories: 6 or 8 output channels from the sampled pianos (depending on how many sampled pianos are used), 8 channels of granular synthesis spatialization (coming from up to 48 granulators running simultaneously), 8 channels of sine wave autopanners (sine wave instrument instances are randomly assigned to one of the available panners) and finally 6 or 8channels for the spectral piano processors.

An extension planned for future versions is to allow more control (either automated or through the fader box) of the directionality of the autopanned audio feeds.

Finally the outputs are routed to their proper final destinations. This is programmable though global variables and is designed to accommodate several flexible options for the diffusion of the piece. With the current hardware the audio can feed up to 16 discrete speakers through one or two Ambisonics decoders, or can send a raw Ambisonics stream to an external decoder through either analog or digital connections.

7. USING REAL PIANOS

The program can also control MIDI controlled pianos (so far only used Disklavier Yamaha pianos have been used).

The behavior of Yamaha Disklavier pianos presents a unique challenge not yet fully tackled in the program. The Disklavier have two operating modes. A non-realtime mode can have perfect rhythmic accuracy at the cost of a 500 mSec delay between the arrival of MIDI messages and the sounding of a note. Or a realtime mode, with *almost* no additional delay. A problem is that in this mode the delay between reception of the incoming MIDI messages and the sounding of a note depends on note velocity (low MIDI velocity notes have more delay than high MIDI velocity notes, see [9, 10]). On the other hand, the sampled pianos react instantly (within the delay of the controller itself and the latency of the audio interface which is normally on the order of 5 millisec onds) while the Disklavier has a delay between the reception of the MIDI message and the sounding of any note.

For that reason in the current program the Disklavier is never sent notes played by the performer but rather receives only notes generated by algorithms. The delay is not that important for those (but would be very cumbersome for the performer as there would be a noticeable echo effect). Even then the result is less than optimal as the delay is noticeable even when only algorithms are playing through it.

In a future version of the program that delay should be compensated by the software (for example using lockup tables as in [10]) and taken into account in the scheduling of the algorithmically generated notes themselves so that the actual played notes are in better sync with the other (sampled) pianos.

7.1 The Disklavier as a controller

The Disklavier has also not been used so far as a controller but that is also contemplated for future versions of the program. The performer should be able to switch between the two keyboards (when a Disklavier is available - the piece can be played without one). Further processing of incoming and outgoing note events will have to be programmed so that algorithms and the real performer are unlikely to “play” the same note at the same time. The solution being contemplated will probably implement a dynamic guard zone around the last performed notes in the Disklavier that can't be activated by algorithms (so the algorithms will work around the human player and try to not interfere with him or her).

8. TEMPO AND TRANSITIONS

The overall tempo of all algorithmically generated textures can be controlled manually or automatically. In a section of the piece the tempo is automatically changed (rapidly or in a slower transition) by switching scenes.

Background routines can be triggered by scene changes so that parameters that control algorithm generation or any other parameter in the program can be changed continuously over a period of time.

9. GRAPHICAL USER INTERFACE

Using SwingOSC a graphical interface is presented to the performer to give feedback during the performance. Two prominent elements are a running clock that is started when the first note event is received from the keyboard, and three text areas that show the previous, current and next scene in the performance (previous and next text fields are smaller and grayed out). A notification text panel can display arbitrary text strings and is used mostly for updates to tempo and other gradual changes in internal values. Further down a GUI of the keyboard with sev-

eral subsections shows the state of all the keys and algorithms.

The first row shows which keys have active algorithms running and associated with them, and how many (the hue changes according to the total count of threads on each key). The second row shows which notes have been triggered by algorithms and the third row shows the keys that have fractal algorithms running on them and how many.

A total grand count of running algorithms, granulators and pending algorithms is shown below as well as the state of the Markov chain learning routines.

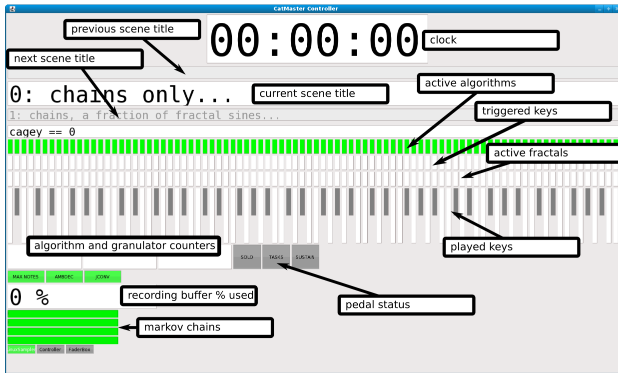

Figure 2: Graphical user interface

Three buttons show the state of the three main control pedals and several additional indicators show the state of external programs and the connection state of external hardware.

10. LIMITATIONS

After 1 ½ years of evolution the program is hitting the limits of what is possible to do with the current generation of laptops owned by the author (those limitations of course disappear when using a faster 4 core desktop machine but that is not practical when playing in concerts that involve traveling abroad).

A short term solution has been the use of two simultaneous SuperCollider synthesis engine instances to use the dual core processor of the laptop. The solution has successfully made use of more available processing power. The parallel nature of some of the processing enables it to be parceled to a separate synthesis engine (and Jack splits the processing in different parallel threads). In this piece the granular synthesis engine (plus the associated spatialization routines, see below) which turned out to be quite cpu intensive is using the second instance of scsynth and is shifted automatically by the operating system to a different core.

11. EVOLUTION OF THE PROGRAM

This section gives a very sparse chronology of the evolution of the program and the consequences of major changes in the structure and form of the piece. Some internal heavy rewriting of the code is not listed as it only had impact in the clarity of the program code and the possibility of further expansion of the program.

The first version of the program (end of October, 2008) was just a proof of concept short program with responders for note events and a first implementation of the scale algorithm. Shortly after that the first implementation of the Markov algorithm was incorporated into the piece and lead to a lot of experimentation and tuning that grew into the first versions of what would become “Cat Walk”.

In short: later added chord detection code to properly train the interval Markov chains (Oct 29). Split into two main pianos panned left and right in the stage (Nov 4). First code for granulation instruments, this enabled the first addition of synthetic sounds to the performance (Nov 7). Major work in the GUI for feedback to the performer, including the elapsed time counter and a first try at piano keyboard views that monitor activity of the algorithms (Nov 9). Added Markov chain for note duration and a pedal that stops all tasks (Nov 11) - the control pedal addition was vital for performance as a way to control the density and timing of the algorithms, and after that the piece was more dynamic and the possibility of contrast in the form was greatly enhanced.

Added quantization for training of duration and rhythm Markov chains, and modulation wheel control of the length of algorithms (Nov 12). First implementation of Scenes (Nov 13). Scenes enabled the composer to program the high level structure of the piece in the program. A lot of debugging ensued because sometimes there would be hanging notes, specially from the Disklavier (it was later discovered that the Disklavier did not really work well with lots of overlapping notes and those were programmatically forbidden). Added reverberation using the Freeverb algorithm. Finally added pitch bend code for both main pianos (Nov 19). This evolved later into a whole section of the piece in which the performer plays with detuning and micro tonal textures.

And finally a major milestone, after many rehearsals the first concert performance of “Cat Walk” on November 20th 2008. It culminated a month and a half of very intensive coding and test performances.

At the beginning of February 2009 the first implementation of the fractal melodies code was written. The capability to stop the fractals with the algorithm pedal was also added and the pedal was subsequently used to select between Markov and fractal melody algorithms. Also the spatialization code was changed by adding auto-panning functions that moved the pianos and granulators around the audience. Also added the code that supported the BCF2000 fader box to control the volume of different audio streams.

Added optional sine wave additive synthesis components to the fractal melodies (Feb 8). This changed the piece significantly as the sonic color of the fractals could be further manipulated. Converted spatialization to use VBAP and tried to use 3D VBAP code with 16 speakers, but the CPU load was too high, so the spatialization was switched to use 2nd order Ambisonics encoding. Added convolution reverberation code using Jconvolver, replacing the simpler Freeverb Schroeder reverberation that

was used before (Mar 3). Add spectral processing instruments (Mar 20). This originated another section of the piece that follows the pitch bend section. Changed the Ambisonics encoder to use 3rd order (Mar 24). As the CPU limits were approached a second SC synthesis server was added to spread DSP load between cores (Mar 28). Changed the reverberation to use Ambisonics impulse responses (Mar 30). Split spatialization into two Ambisonics rings (Apr 2).

Another important milestone concert performance on April 16th 2010. Much expanded piece that included all of the above changes in the code.

Added solo pedal (Sep 8). This allowed more freedom in the performance as the performer can now play solo. Changed the internal structure of the software to be more modular (Sep 10). Implemented more tempo change functions. Added trill algorithms. This led to the creation of a whole new section at the end of the piece in which tempo changes gradually and abruptly. Add next and previous scenes text views so the performer can anticipate the next section of the piece before transitioning into it (Sep 14).

Another major milestone On September 18th 2009, first concert performance that includes the "trill" algorithm and a whole new section of the piece at the very end.

More details about the performances, and a recording of a current performance of the work can be found at:

http://ccrma.stanford.edu/~nando/music/a_very_fractal_cat/

12. FUTURE WORK

Much work remains to be done. In reality this is an open ended project that merges programming and performance art. Currently the duration of the piece is around 15 minutes but with the palette of sounds already available it could be expanded significantly, possibly into a suite of smaller pieces that further explore the musical spaces of the different algorithms and processing techniques used.

The code needs a lot of refactoring work to be able to add more algorithm types as modules. The original algorithms and the training of the Markov chains is currently hardwired into the code and not modular.

And so far the program only responds to events generated by the performer. A major change will be creating a process that can generate events by itself and not only in response to the performer. That would open the door to a dialog between the processing routines and the performer.

Chord analysis and use is another area of future expansion, chords are being detected by nothing is done about them at this point. More and better sound processing instruments is also a goal.

At this point it is also necessary to have a detailed look at cpu usage with the goal of optimizing it, specially with regards to all the sound processing and generation code. It is becoming increasingly difficult to expand the functionality of the program without hitting the hard limit of maximum cpu usage.

13. ACKNOWLEDGMENTS

This piece would not have been possible without the many professional open source software programs available for free. Many thanks to the hundreds of developers that make it possible to use a very sophisticated environment for programming and music making.

14. REFERENCES

[1] "Linuxsampler," (an open source audio sampler) . http://www.linuxsampler.org

[2] AmbDec (open source Ambisonics decoder), by Fons Adriansen (http://www.kokkinizita.org).

[3] Jconvolver (open source partitioned convolution engine), by Fons Adriansen (http://www.kokkinizita.org).

[4] Jack, an open source sound server (www.jackaudio.org).

[5] David Jaffe, W. Andrew Schloss: *Intelligent Musical Instruments: The Future of Musical Performance or the Demise of the Performer?,* INTERFACE Journal for New Music Research, The Netherlands, December 1993

[6] Jean Claude Risset: *Three Etudes, Duet for One Pianist* (1991)

[7] Rick Taube: *Notes from the Metalevel*, Editorial Acme, Utrecht, 2004.

[8] SuperCollider, http://supercollider.sourceforge.net/

[9] Werner Goebl and Roberto Bresin, *Measurement and Reproduction Accuracy of Computer Controlled Grand Pianos*, Stockholm Musical Acoustics Conference, 2003

[10] Jean-Claude Risset and Scott Van Duyne, *Real-Time Performance Interaction with a Computer Controlled Acoustic Piano*, Computer Music Journal, Spring 1996

[11] Fernando Lopez-Lezcano, "*PadMaster, an improvisation environment for real-time performance*", ICMC 1995, Banff, Canada.

[12] Fernando Lopez-Lezcano: "*PadMaster: banging on algorithms with alternative controllers*", ICMC 1996, Hong Kong

[13] Tim Blechmann, *supernova, a multiprocessor-aware synthesis server for SuperCollider*, Linux Audio Conference 2010

[14] FAUST, a compiled language for real-time audio signal processing; http://faust.grame.fr/