Chaos Melody Theory™

by Elaine Walker

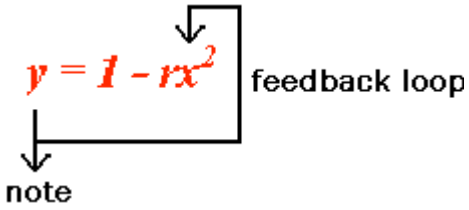
Introduction

Chaos Theory applies to many things in nature. The idea of chaos is that one simple seed can bloom into a very complex fractal structure. Trees are born out of seeds that grow into very complicated shapes based off of a few simple shapes. With a constant dichotomy that is true to their fractal structure, each branch of the tree resembles the main trunk, each root splits off into smaller and smaller roots, and even the entire tree itself may resemble the shape of one leaf. Most everything that is not man-made has a chaotic origin and is fractal in structure, including the universe itself. Perhaps man-made structures would be more welcomed in nature if they were also based off of chaotic methods.

This paper is about a new type of music theory that is chaotic in origin, and organic and human in feeling. This music is derived from a simple mathematical function whose output is fed back into the input over and over, out from which flows a stream of numbers that form extremely interesting organic melodies. It is called "*Chaos Melody Theory*". This type of algorithmic music is the result of a recursive process in which a simple iterative function, such as $f(x) = 1 - rx^2$, or $f(x) = rx(1-x)$, is used to generate a stream of numbers that is scaled and mapped to MIDI note numbers and directly applied to the pitch parameter. The generic term "chaos music" will refer to such music, where pitch is being controlled by a mathematical iterated function. *Chaos Melody Theory* will be compared to certain aspects of traditional Western music theory, however, this paper will not come to any absolute and profound conclusions.

The terms "chaos melody" and "melody sets" will be used quite often in this paper to describe the melodic patterns that result from iterating an equation, either in the real or complex domains. One member of the melody set is defined as "the scaled output of an iterated function, provided it does not escape to infinity or tend towards zero". A chaos melody is one such member of the set. To further note, the absolute frequencies that result are somewhat arbitrarily scaled. For this reason, it is the up and down patterns of the melodies, not the exact frequencies that are the subject of discussion. The basis of *Chaos Melody Theory* is that the beauty lies in the patterns of notes, as they leap up and down in a signature manner, as a result of the particular equation that is being iterated.

A simple process is used to calculate a stream of numbers that are mapped to pitches to create musical compositions, starting with a simple formula such as $y = 1 - rx^2$. The equation is solved for y, again and again, each time replacing x with the y value from the previous computation. This process is repeated over and over, feeding the output back into x for each successive iteration, like a musical feedback loop.



A better way to write the formula would be:

$$f(x_{n+1}) = 1 - rx_n^2$$

Complete musical works can be shaped and molded by the performer by dynamically varying the ***r*** parameter and reinitializing the ***x*** value in real-time, while iterating the function. The rate at which the function is iterated will be determined by the performer. With *Chaos Melody Theory*, the only parameter being controlled with the mathematics is the pitch, and the performer has complete control over the phrasing, dynamics, and tempo. Desirable ranges for ***x*** and ***r*** and they're unique effect on the output will be explored in this paper. In general, however, depending on what numbers are plugged in for ***x*** and ***r,*** the resulting stream of numbers will have one of three possible outcomes:

1. The output will eventually tend towards zero
2. The output will eventually settle into a repetitive cycle
3. The output will eventually escape towards infinity

Simple Chaos Melodies

A chaos melody is defined as "the scaled output of an iterated function, provided it does not escape to infinity or tend towards zero". In other words, the mathematical iteration is only considered a true chaos melody if it eventually settles into a repeating pattern (number 2 above). This means that a function will only result in a chaos melody when iterated if the parameters ***x*** and ***r*** are kept within a specified range. In this case, the range for ***x*** would be between -1 and 1. If ***x*** is outside of that range, the output would rapidly get larger as a result of being squared over and over. The best range for ***r*** in this case is between 0 and 2.

The ***r*** parameter determines, to an extent, how chaotic the output will be. The ***r*** parameter is a constant, although there are many cases when it should be varied dynamically for musical purposes, moving from areas of more predictable melodies to areas of a more chaotic dimension and vice versa. The "bifurcation" diagrams below show how gradually increasing the value of ***r*** over time causes the output to be more and more chaotic. In Figures 1 and 2, the ***r*** value is superimposed over the diagram to show how the level of chaos increases with the value of ***r***. (The exact range of pitch and time are arbitrary in these diagrams.)

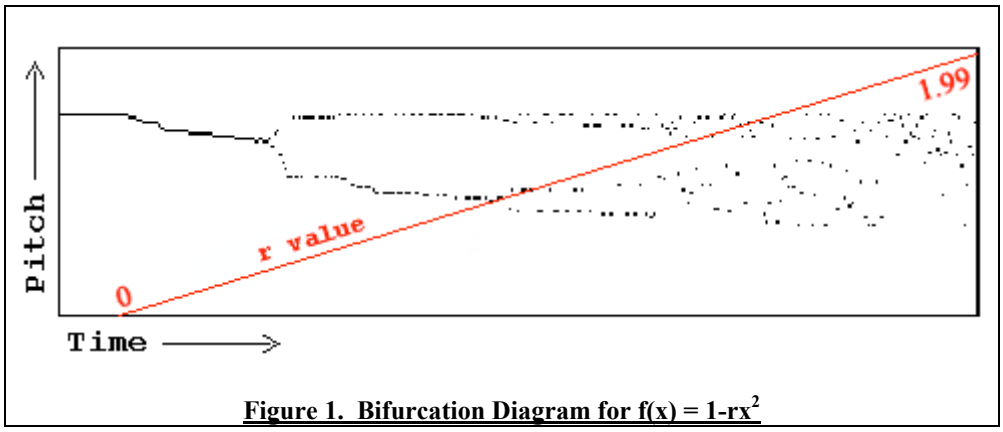


Figure 1. Bifurcation Diagram for $f(x) = 1-rx^2$

Imagine each pixel in the bifurcation diagram as one discrete musical note, somewhat like a piano scroll, and each pixel corresponds to one iteration of the mathematical function. Towards the left end of the diagram, when ***r*** is at a low value, the output would sound like a series of notes that each are continually lower in pitch. When the value of ***r*** goes above a certain threshold, the output bifurcates, or splits in two, and the output bounces between two notes. Eventually, when ***r*** crosses the next threshold, each branch bifurcates again, resulting in a melodic pattern bouncing between four notes in a particular order, etc. Eventually, as ***r*** gets even higher, the melody becomes so complex, it begins to sound more and more random. A similar outcome happens with another equation, ***f(x) = rx(1-x)*** as the value of ***r*** is increased over time. However, notice that the pattern has a different shape, resulting in different sounding melodic structures. In other words, the function itself defines what the range of melodic patterns will sound like.

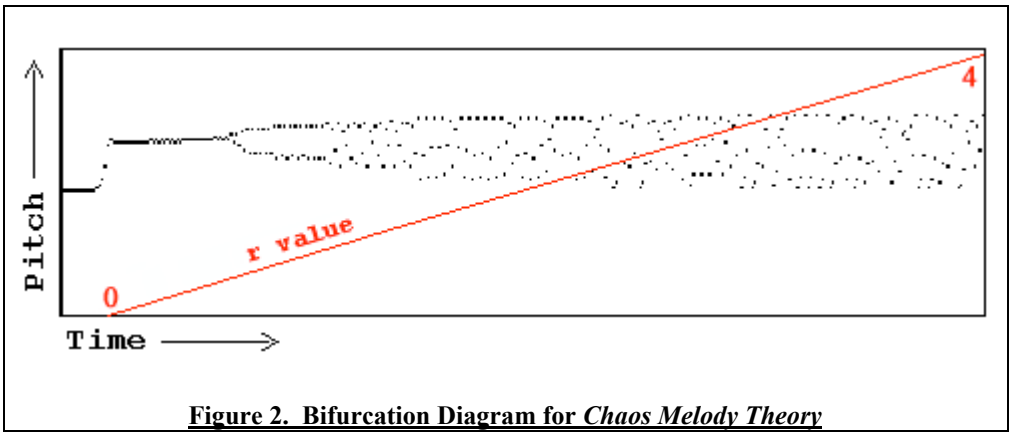


Figure 2. Bifurcation Diagram for *Chaos Melody Theory*

In Figure 1, if ***r*** is raised past 1.99, the notes would trail upward and tend towards infinity. The same would happen in Figure 2 if ***r*** is raised past 4. It hasn't been specified what starting value of ***x*** was used in these diagrams, but for now it will suffice to say that a value was chosen that would not result in an escape to infinity, or a tendency towards zero. The above are typical examples of how continually raising the value of ***r*** for each iteration creates a bifurcation diagram that exhibits a coherent structure that

is unique to the equation being used. The bifurcation diagrams above are like signature patterns for the respective equations. If this technique is used to create a melodic structure in *Chaos Melody Theory*, it is simply called "bifurcating". In practice, the value of ***r*** is rarely raised continually like this, and instead, ***r*** would be set to some value within the desired range and left static long enough for a complete chaos melody to run it's course.

The ***r*** value (or chaos value) has the effect of determining the complexity and shape of each chaos melody's infinite repeating pattern or "orbit". Observing the bifurcation diagram of Figure 1, it is possible to get an idea of what an orbit for $1-rx^2$ may look like for various values of r. If ***r*** is less than .5, the orbit will consist of only one note, over and over. Between .5 and 1, the orbit will consist of 2 notes bouncing back and forth. If ***r*** is set at 1.2, the orbit will be a 4 note pattern, and if ***r*** is set close to 2, the orbit will be very complex. As will be shown in this paper, even a very complex orbit is sometimes recognizable as belonging to a certain "melody set". In other words, a complex orbit may contain small, recognizable patterns that are characteristic of the function that it originated from.

The significance of the ***x*** value (or seed value) is the unique meandering pattern or "phrase" that begins each melody, which get pulled into an orbit. Three chaos melodies are shown below, each derived from the same function, and with ***r*** set at the same constant value. However, each melody was initiated with a different value of ***x,*** resulting in a different phrase for each melody. All three melodies eventually gravitate to the same orbit. The orbits below are fairly straightforward 6 note patterns that repeat over and over, but the phrases appear to be haphazard and almost random in nature. Observing Figure 3 closely, small recurring patterns appear to emerge within the phrases, and it should be apparent that they are not random, and rather are organic in nature, with similar repetitive shapes recurring from time to time. Phrases can be a very interesting and musically pleasing in their own right.

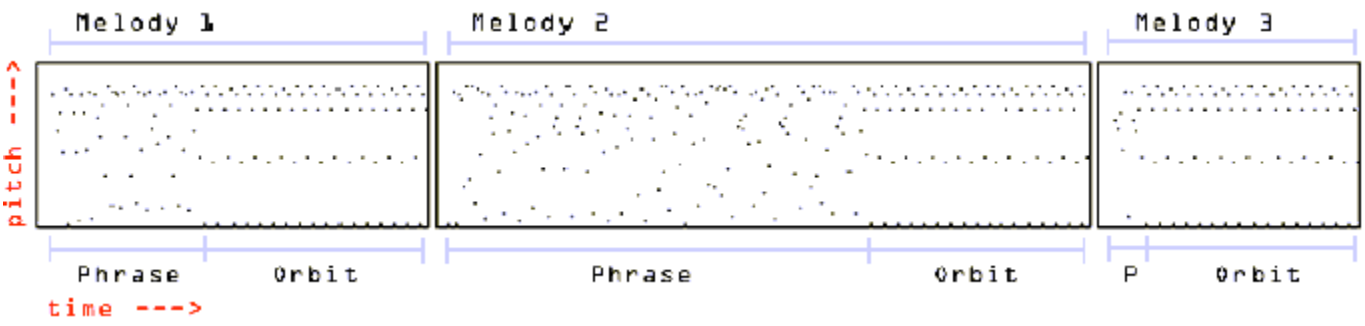


Figure 3: A function being iterated, starting with 3 different values of x

The challenge of composing satisfying music with *Chaos Melody Theory* is one of finding specific values (or ranges of values) for ***x*** and ***r*** that will result in phrases and orbits that are musically pleasing. A virtuoso of chaos melodies would know the art of initiating new ***x*** values, and have a feel for how one orbit will flow into a new orbit when a parameter is changed.

Traditional Music vs. Chaos Melody Theory

Musical compositions normally consist of building blocks such as themes, phrases and ideas all tied together as a coherent unit. What is needed for interesting music, traditional and untraditional alike, is a satisfying mix between the orderly and predictable and the surprising and unpredictable. It should also follow some simple grammatical rules, and exhibit patterns of stepwise motion mixed with small intervallic leaps. *Chaos Melody Theory* meets these criteria. Within each simple function is an infinite set of melodies, some of which are simple and repetitive, and some which are explosive and complex in form. Even the most chaotic patterns found here don't just seem random, leaping up and down in a completely haphazard manner. To the contrary, the complex patterns have an eerily human feel. The stepwise meandering mixed within the leaps and bounds goes beyond the commonly used Brownian or Drunken noise patterns. There are organically recurring phrases and patterns that can be thought of as "motifs" or melodic building blocks.

Chaos Melody Theory actually feels more like clay than blocks, in that it is malleable, yet even more fluid in its dynamics. One melody will "flow" into a new melody as a parameter is nudged in one direction, as if a spacecraft were nudged into a new planetary orbit. The spacecraft would not instantly assume its new orbital trajectory, but would gradually be captured by the new gravity basin, taking several orbits to fall into its new orbit. It is necessary to carefully choose limits for the seed and chaos parameters that will not ever allow the notes to fly off into infinity (get ejected into interplanetary space) or tend towards zero (fall into the planet's gravity well, crash landing on the surface). Like computing a spacecraft trajectory, computers can be used to find limits and combinations of parameter values that will keep the equation from iterating into infinity. However, only years of careful listening and experimentation will yield the most pleasing musical results, in the same sense that our traditional harmony we are taught today is based off of centuries of composition and careful studies. Today we could easily look at chord combinations and compute exactly how the harmonics will acoustically affect one another. However, only with human effort are most musical progressions composed.

Although the chaos melodies tend to sound very organic and even humanlike at times, to really produce expressive and convincing music with chaos melodies, it is necessary to not only have dynamic control over the "chaos" parameter and to be able to initiate new "seed" values at will, but to also have the ability to modify loudness, note duration and rhythmic content in real-time. In a sense, the performer or composer of chaos melodies is "conducting" the dynamics and phrasing of the music as one would conduct an orchestra. However, *Chaos Melody Theory* goes one step further, in that the performer is also conducting the mathematics, and is making creative choices that affect the melodic content. Like an orchestral conductor, it is not necessary to be a virtuoso of any particular instrument to be able to play this music, but it certainly helps to have a good musical sense of phrasing, dynamics, and musical emotion.

Consonance and Dissonance:

Now, in order to get into another area of comparison, it will be necessary to look at a more detailed bifurcation diagram that has dynamics that are more complex than the previous examples. The bifurcation diagrams of $1-rx^2$ and $rx(1-x)$ shown in Figures 1 and 2 are fairly straightforward, getting continually more complex as the $\boldsymbol{r}$ parameter is raised. Other functions such as $\boldsymbol{x} = x^2 + \boldsymbol{r}$ in the diagram below have far more complex dynamics as the $\boldsymbol{r}$ value is incremented. Instead of simply branching off

into a more and more chaotic structure, there are "pockets" where, in the midst of chaos on either side, very stable areas emerge.

In the bifurcation diagram below, the ***x*** axis represents pitch. Time is along the vertical axis, as well as the ***r*** value. The ***r*** value is raised (actually lowered into the negative, in this case) continually along with time, from top to bottom. Arbitrary values along the ***r*** axis are labeled with the letters A through G. B, C, D, E, F and G represent the reservoirs or "pockets" where the output is simple. H, I, J and K represent the areas of chaos. Viewing the diagram, it is apparent that if $\boldsymbol{x} = x^2 + \boldsymbol{r}$ is iterated with ***r*** as a constant, staying at a value around A, it would result in a melody of one note playing over and over. Around the areas of H, I, J, or K, it would be a rather chaotic melody. B,C, and D are stables areas that one would expect from a bifurcation diagram, simply splitting off into more and more branches. However, E, F, and G are areas of stability in the midst of the chaos that don't follow a simple bifurcation pattern.

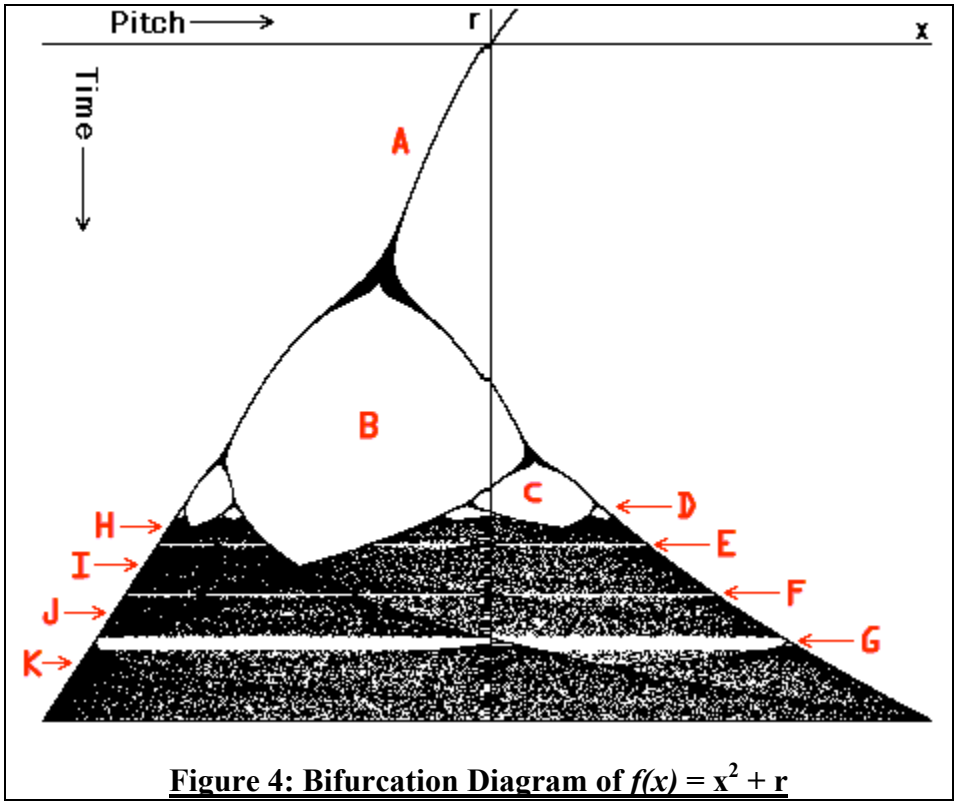


Figure 4: Bifurcation Diagram of *f(x)* = x^2 + r

In other words, continually raising the ***r*** value causes the bifurcation to go through areas of stability and chaos in a non-linear fashion. Other functions also have non-linear bifurcations, where there may be pockets of consonance in the midst of chaos on either side. This can be loosely related to the concept of "dissonance curves" in musical scales. Intervallic dissonance has a bumpy curve, where playing consistently larger intervals repeatedly passes through cycles of consonant and dissonant intervals, in a very nonlinear fashion. In *Chaos Melody Theory*, it is more useful to think in terms of consonance and dissonance of melody sets, rather than consonance and dissonance of intervals and chords in traditional music.

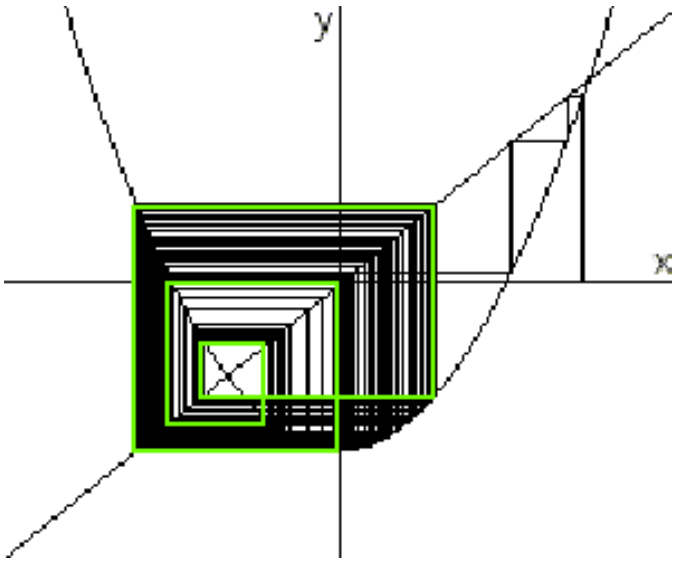
A Deeper Look Into Chaos Melodies: Phrases and Orbits

This section will frequently refer to the bifurcation diagram above. The function diagrams that follow show melodies derived from the same function, $y = x^2 + r$, as the bifurcation diagram. Each graph has an ***x*** and ***y*** axis, a diagonal line corresponding to $y = x$, and a parabola that is a graphical representation of the function for a specific value of *r*. The perpendicular black lines represent the phrase portion of a melody, and the light green lines represent the orbit that the phrase gravitates to.

This is a fun technique for computing the outcome of an iteration, and it is one that can even be done by hand to get a rough idea of whether certain values will cause the melody to escape into infinity, tend towards zero, or get caught in an orbit. Starting on the ***x*** axis with the desired seed value for ***x,*** a line is drawn vertically to the parabola. Then a line is drawn horizontally to the $y = x$ line, then vertically to the parabola, horizontally to the $y = x$ line, and so on and so forth. Each time the line touches $y = x$, a note is represented, according to it's position along the ***x*** axis (to the right of the ***y*** axis would be higher notes, and to the left of the ***y*** axis would be lower notes).

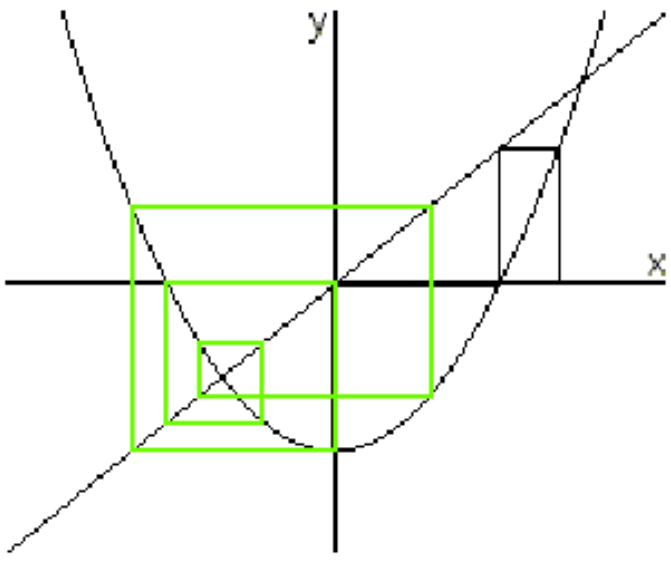
The following four graphs ***1.*** through ***4.*** use a value of $r = -1.475$, which corresponds to the E section of the diagram above. E refers to the horizontal cross section where there are basins, or "pockets of consonance" where a pattern of 6 notes would result. Using the technique below to derive melodies with that same value of ***r,*** the results indeed show 6 notes along the $y = x$ line. Four different starting ***x*** values are shown, resulting in four very different phrases as they are pulled into the orbit. The phrases range from simple to complex, but each fall into the exact same simple orbit (in green). Following the horizontal and vertical black lines from the starting point on the ***x*** axis, it is interesting to see how the lines meander their way into the orbit.

1.



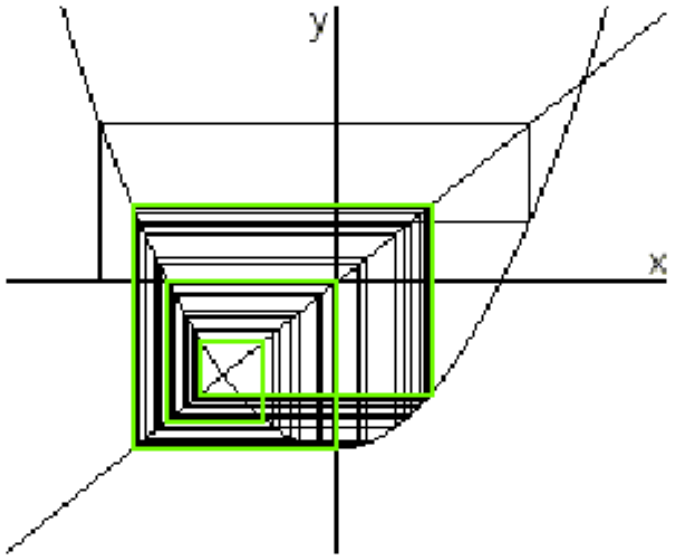
E: ***r = -1.475, x = 1.76777***

2.



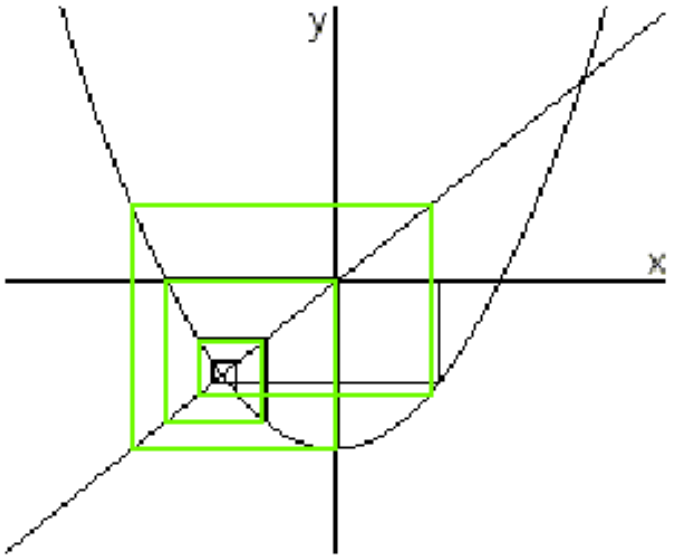
E: ***r = -1.475, x =1.63763***

3.



E: r = -1.475, x = -.70270

4.



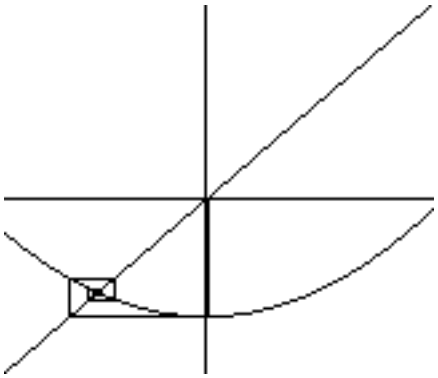
E: r = -1.475, x = .77001

The following diagrams ***5.*** through ***14.*** show orbits that correlate to different areas (labeled with capital letters) of the above bifurcation diagram. Each was iterated with ***x*** = *zero*. With the exception of ***5.***, which gravitates from the initial seed value of zero to a single point on the function line, starting *with* ***x*** = *zero* results in melodies that have no phrase. They immediately assume their final orbit without any initial meandering. It is apparent, when comparing the bifurcation diagram to the orbits derived from the different ***r*** values, for different areas of consonance and dissonance in the bifurcation diagram, the orbits are correspondingly simple or complex.

5. is derived from the area of A where ***r*** = *-.33*. In the bifurcation diagram it is apparent that the orbit would only consist of one note over and over, and graph ***5.*** shows the melody indeed gravitating to one point. Graph ***6.*** is derived from the area around B, and it is apparent from both the bifurcation diagram and the graph, that the orbit would consist of only two notes. The orbit of graph ***7.*** is composed of 4 notes, and ***8.*** has 8 notes, which is also apparent in the bifurcation diagram. Graph ***9.*** and ***10.*** are derived from ***r*** values that are further down in the bifurcation diagram (area F and G respectively), however, they have simpler orbits than graphs ***7.*** and ***8.*** This is because they are from basins that have a high amount of melodic consonance. It is quite clear that area G in the bifurcation diagram would result in an orbit of three notes.

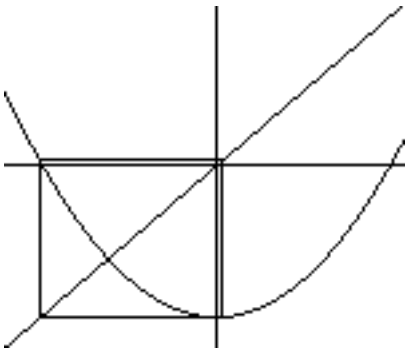
11., 12., 13. and ***14.*** are derived from the areas H, J and K, which are chaotic areas. ***11.*** and ***13.*** might make nice melodies, but ***12.*** and ***14.*** would most likely sound like a bit of a mess. Although ***13.*** looks rather complicated, there is an obvious pattern to the orbit. Even 14. may be useful in the right context, for it certainly has a similar pattern imbedded in its orbit as well.

5.



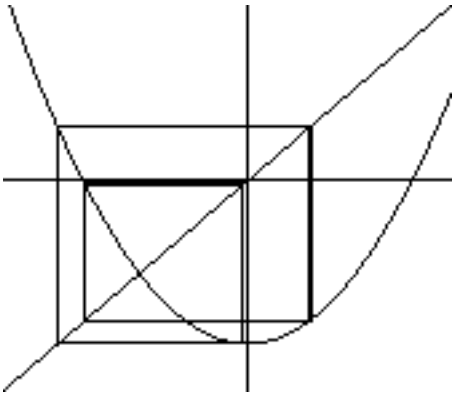
A: r = -.33, x = 0

6.



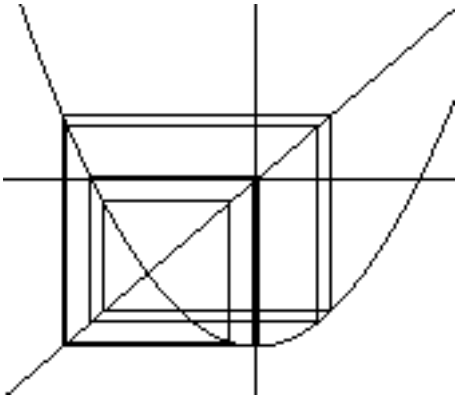
B: r = -1.03, x=0

7.



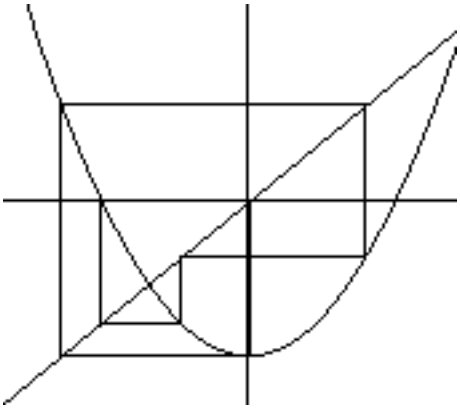
C: r = -1.33, x=0

8.



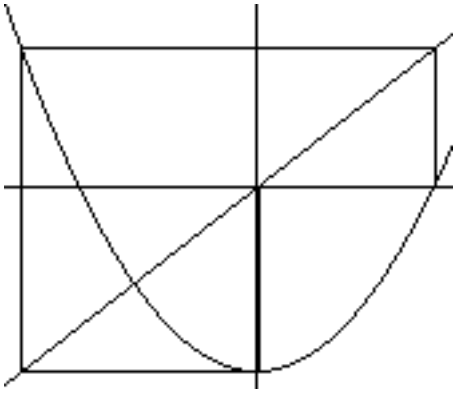
D: r = -1.39, x=0

9.



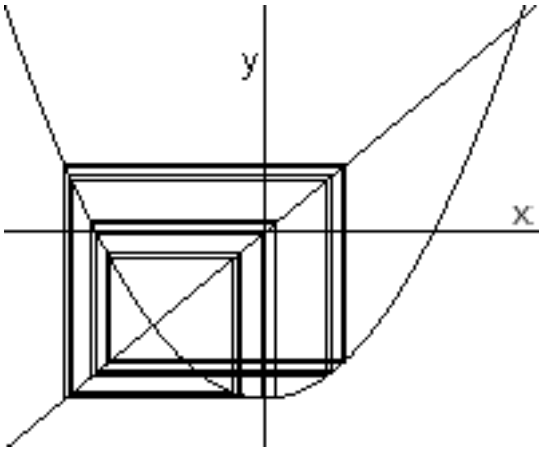
F: r = -1.627, x=0

10.



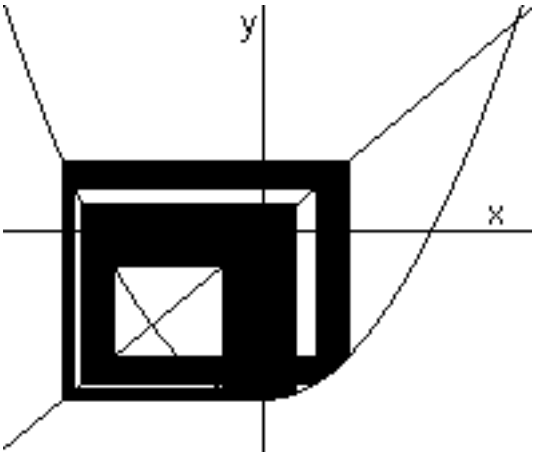
G: r = -1.75694, x=0

11.



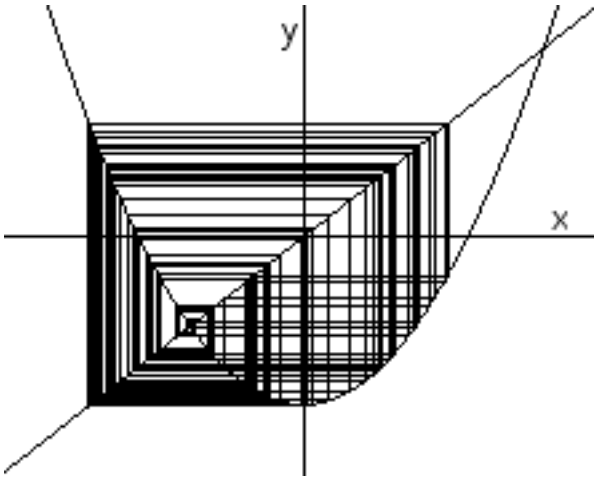
H: r = -1.4, x=0

12.



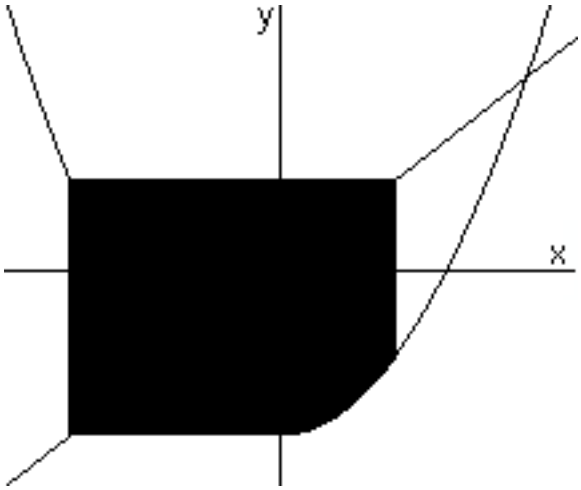
H: C = -1.42, x=0

13.



J: r = -1.674, x=0

14.



K: r = -1.79, x=0

Complex Chaos Melodies

It is possible to expand the idea of chaos melodies into the complex domain, resulting in a counterpoint between two different, yet thematically related, melodies. In other words, a function that is being iterated to produce chaos melodies may contain complex variables. Two examples of extending simple functions into the complex domain are shown below. When a function is extended into the complex domain, it gets divided into two parts: a "real" part and an "imaginary" part. One melody is the result of iterating the "real" part of the complex function, and the other the result of iterating the "imaginary" part. The output of the real portion is fed back into both the real and imaginary portions, and the output of the imaginary part is fed back into both parts as well. The real and imaginary parts of the function are therefore dependent on the output of each other, simultaneously. The phrase and orbit portions of both melodies happen synchronously, but are not identical. The up and down patterns vary, but they always exhibit similar recurring motifs (little recognizable melodic patterns) throughout.

Chaos Melody Theory applies in either the real or complex domain. However, in the complex domain, the results can be wildly more chaotic and less predictable. An example of a very simple melody in the complex domain is shown in Figure 13, overlaid on top of a Julia set. In this example, each dot represents two notes: one related to the ***x*** axis (the "real" axis), and one to the ***y*** axis (the "imaginary" axis). The blue dots make up the phrase of the two melodies, and it takes seven iterations for the phrase to get pulled into the orbit. The orbit consists of only four notes in this example, which would repeat indefinitely. Complex melodies can be much more complicated than this simple example, however. When a phrase gets pulled into a repeating orbit, it may suddenly switch to different orbits, back and forth in a less predictable manner. Therefore, a "repeating orbit" in the complex domain includes the possibility that the orbit may suddenly switch to a different orbit and back again, over and over, an infinite number of times. For this and other reasons, *Chaos Melody Theory* is more straightforward and robust in the real number domain than it is in the complex domain.

For those unfamiliar with complex numbers, a brief explanation of complex numbers follows.

A complex number has two parts; a "real" part and an "imaginary" part. Just as the real number system is the union of the rational and the irrational number sets, the complex number system is a union of the real and the imaginary numbers. An imaginary number is any real number multiplied by the square root of -1. This square root of -1 has a special name, i. A complex number which has a real component 8.5 and an imaginary component of 3.2 could be written as 8.5 + 3.2i. You can perform arithmetic with complex numbers; addition, subtraction, multiplication and division are possible, and follow the rules of basic algebra with 'i' being treated as a variable.

In the same sense that a pair of real number values such as (x,y) can be plotted as a point on a Cartesian plane, one complex number, which is made up of two parts (x,iy) can be plotted as a point on a complex plane. In the complex plane, the horizontal axis corresponds to the real number line while the vertical axis corresponds to the imaginary number line. Any particular complex number, therefore, can be plotted as a point on the plane, and any point on the plane has a complex number that corresponds to it. The real part of the number determines the horizontal placement of the point, and the imaginary number determines the vertical. The origin of the graph (the place where the axes cross) is 0 + 0i.

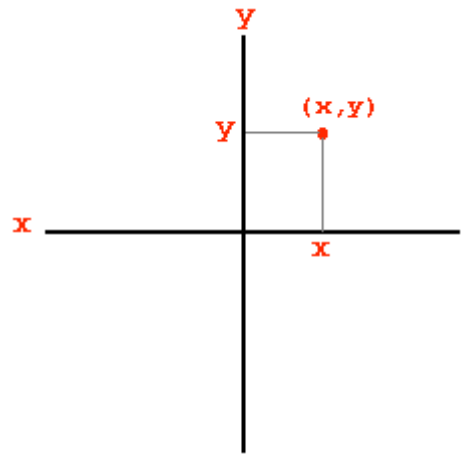


Figure 5: Cartesian plane - the point is representing two numbers, *x* and y

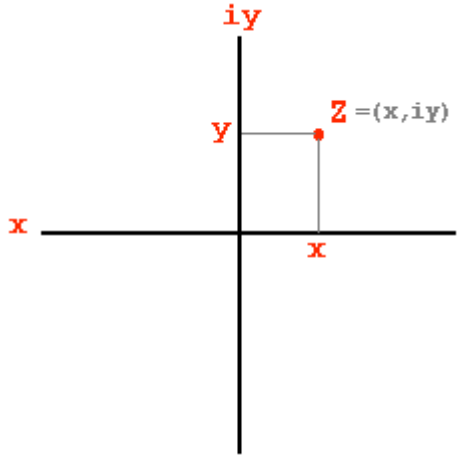


Figure 6: Complex plane - The point is representing one complex number made up of *x* (the real part) and iy (the imaginary part)

Following are two examples of extending simple functions into the complex domain:

Firstly, if the function $f(x_{n+1}) = 1 - rx_n^2$ is iterated in the real domain, a series of values that bounce back and forth along the real number line will result:

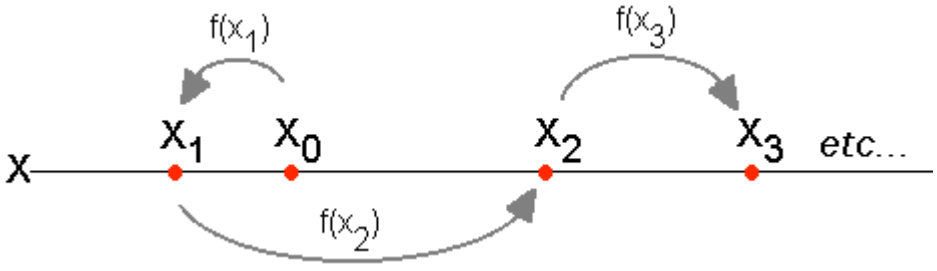


Figure 7: An iteration jumping along the real number line

This can be extended into the complex domain by letting:

r = a + ib , and x = x + iw .

This can now be pieced together into a new complex equation. x^2 is now written as:

(x + iw) ^{2} ***or (x + iw) * (x + iw)***

Conveniently, addition and multiplication of complex numbers obey all of the usual rules of algebra, including the commutative, associative, and distributive laws. Remember that the real and imaginary parts of the equation are added separately.

When multiplied we get:

$(x + iw)(x + iw) = x^2 + iw^2 + 2ixw = x^2 - w^2 + 2ixw.$ (Remember that $i^2 = -1$!)

And $r * x^2$ is now written as: $(a + ib) * (x^2 - w^2 + 2ixw)$

Now, multiply this out: $(a + ib)(x^2 - w^2 + 2ixw) = ax^2 - aw^2 + 2iaxw + ibx^2 - ibw^2 + 2i^2bxw$

And simplify the equation, remembering that $i^2 = -1$: $a(x^2 - w^2) * i[2axw + b(x^2 - w^2)] - 2bxw.$

Let's not forget to add in the "*1* +" at the beginning! :

$1 + a(x^2 - w^2) * i[2axw + b(x^2 - w^2)] - 2bxw.$

Now equation is rearranged to separate the real and imaginary parts. (Anything with an ***i*** in front of it is imaginary.)

$1 + a(x^2 - w^2) - 2bxw + i[2axw + b(x^2 - w^2)]$

Lastly, it is written out as two separate equations:

the real part: $x = 1 + a(x^2 - w^2) - 2bxw$
the imaginary part: $iw = b(x^2 - w^2) + 2axw$

To compute chaos melodies, each ***x*** and *iw* output are fed back into both the real and imaginary parts of the function simultaneously each time the function is iterated. This is shown in the algorithm below:

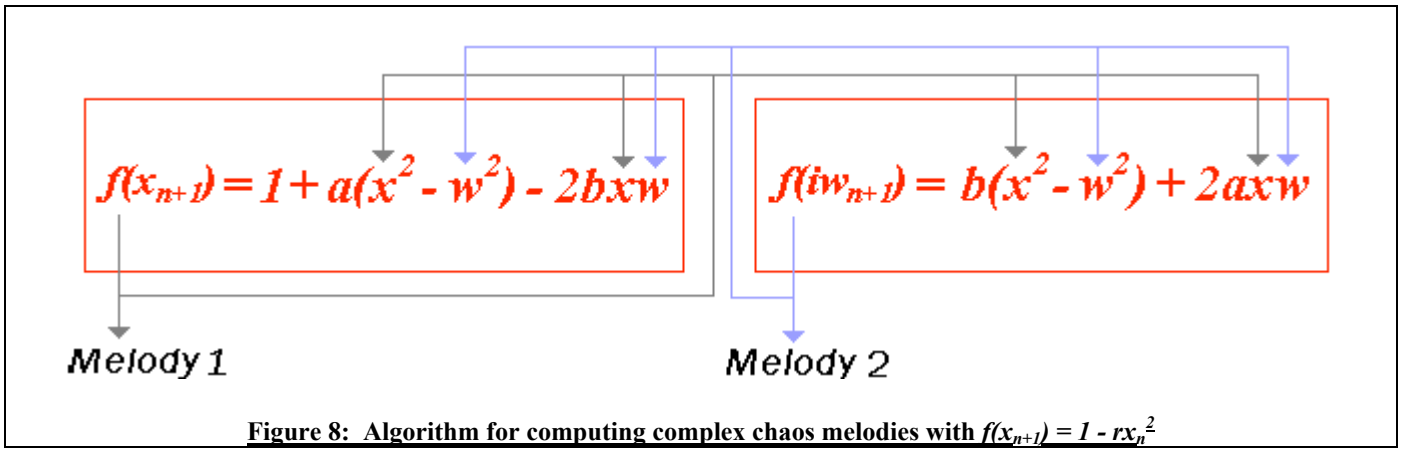


Figure 8: Algorithm for computing complex chaos melodies with $f(x_{n+1}) = 1 - rx_n^2$

Now, imagine the outputs of each part being plotted simultaneously as individual points on a complex plane:

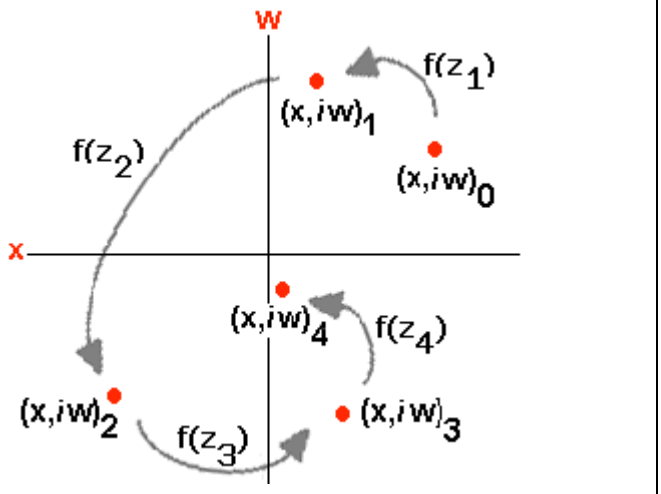


Figure **9: An iteration jumping around in the complex domain**

Looking at a bifurcation diagram of the complex function, there is a separate diagram for the real part and the imaginary part, resulting in a counterpoint between the two. They are similar, but as is common, the imaginary output is somewhat like a compressed version of the real part.

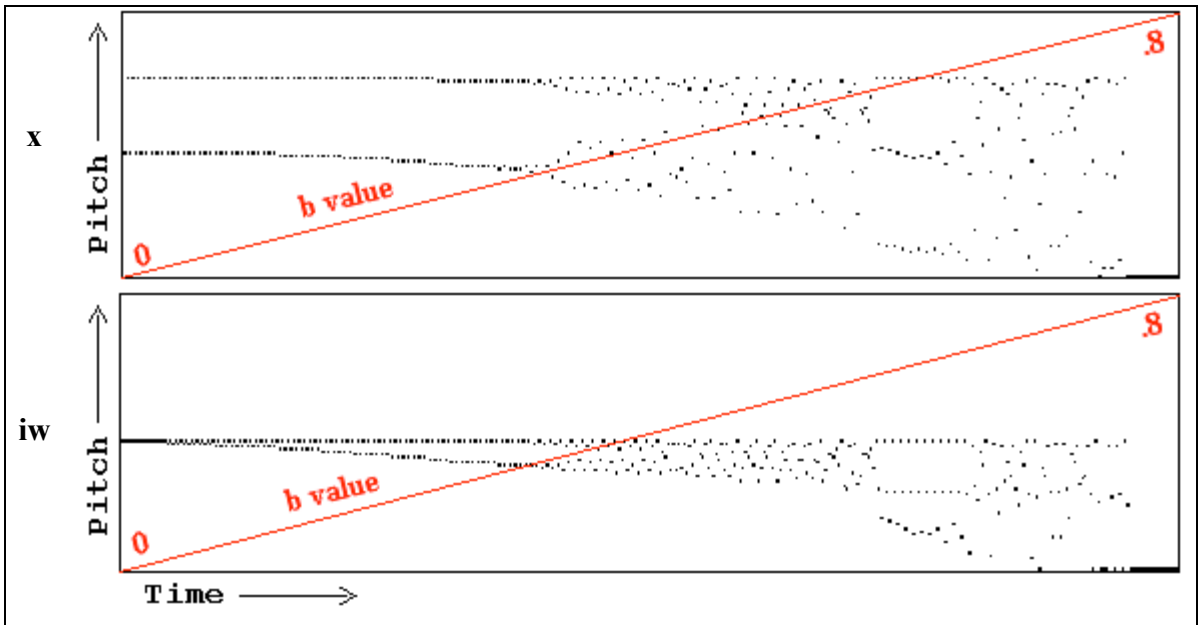


Figure 10: Bifurcation Diagram for $f(x_{n+1}) = 1 - rx_n^2$ in the complex domain
a = 1, b = 0 to .8, *x* = -.99, w = -.99

Interestingly, unlike in the real number domain, where the bifurcation diagram is a "fingerprint" of the function was seeded from, in the complex domain, there can be many different bifurcation diagrams, if one of the chaos parameter values (*a* or *b*) remains constant and the other is varied upward. In these two examples, the *b* value is constantly increasing, causing the bifurcation, but the *a* value stays constant. Above, we see what happens when *a* is set to a value of 1,and below a is set to a value of .75, resulting in a much more simple bifurcation that doesn't get very far before collapsing. Trying out several bifurcations for various values of *a* and *b* is a good way to quickly become somewhat familiar with the dynamics of a complex function.

This has shown that for a = 1, many interesting melodic patterns are possible. In the diagram above, basins or pockets of consonance happen around *b* = .6. Also, it is obvious from the diagram below that for *a* = .75, not many interesting orbits will happen. For *a* = .75 it is not possible for an orbit to go beyond a four note pattern. However, bifurcation diagrams don't tell all about what phrases will happen as an iteration gravitates towards the orbit.

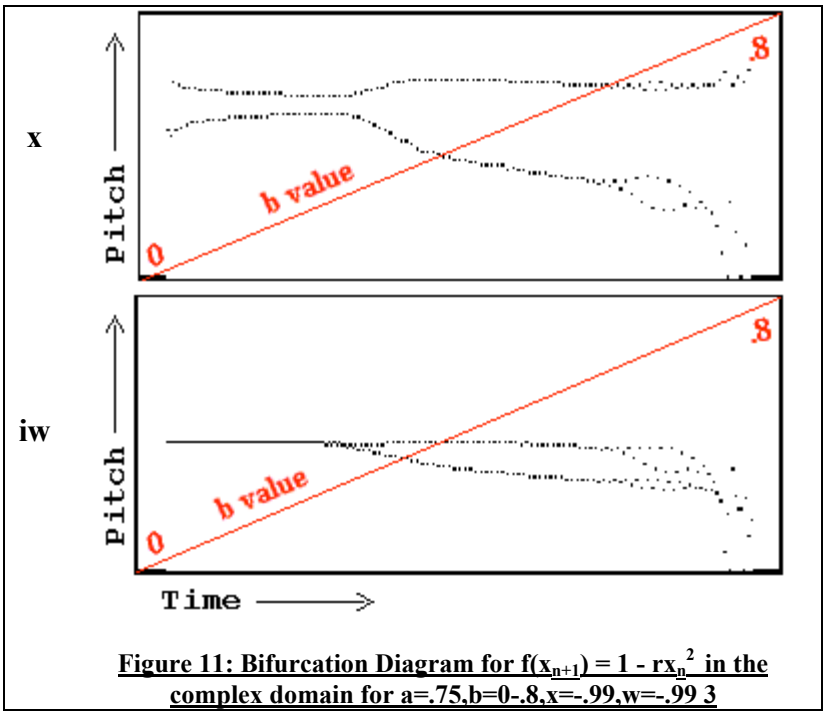


Figure 11: Bifurcation Diagram for $f(x_{n+1}) = 1 - rx_n^2$ in the complex domain for a=.75,b=0-.8,x=-.99,w=-.99 3

Next, to show a second example, the equation $f(x) = r * e^z$ is extended into the complex domain by letting $r = a + ib$ and $z = x + iw$. This equation works out rather elegantly and makes wonderful complex melodies. However, the equation is rather hard to control and can be very delicate. It helps to pre-program distinct finite ranges for the values of the parameters so that the melodies don't race off to infinity.

It should be noted that the number ***e*** is approximately 2.78 and is a special number that occurs frequently in nature. Now the equation looks as follows: $x + iw = (a + ib) * e^{(x + iw)}$.

Now several steps will be taken to separate the complex equation into its real and imaginary parts. No detailed mathematical explanations are given here, however, it can still be appreciated that the equation evolves into an elegant form.

$(a + ib) * e^{(x + iw)} = (a + ib) * [(e^x)(e^{iw})]$

And when multiplied out we get: $x + iw = (a * e^x * e^{iw}) + (ib * e^x * e^{iw})$

We now have to take into account that there is a rule: $e\text{^}iw = cosw + isinw$

The equation is rewritten as:

$$\begin{aligned} x + iw &= ae^x(cosw + isinw) + ibe^x(cosw + isinw) \\ &= ae^x * cosw + ae^x * isinw + ibe^x * cosw + ibe^x * isinw \\ &= ae^x * cosw + ae^x * isinw + ibe^x * cosw + i^2 * be^x * sinw \\ &= ae^x * cosw + ae^x * isinw + ibe^x * cosw - be^x * sinw \end{aligned}$$

The last step was observing the $i^2 = -1$ rule.

Lastly, it is rearranged into the separate real and imaginary parts:

$x + iw = ae^x * cosw - * be^x * sinw + ae\text{^}x * isinw + ibe^x * cosw$

Now get rid of the *i*:

$$\begin{aligned} x &= ae^x * cosw - be^x * sinw \\ iw &= ae^x * sinw + be^x * cosw \end{aligned}$$

In one very last step e^x is factored out, ending up with:

the real part: $x = e^x (acosw - bsinw)$
the imaginary part: $iw = e^x (asinw + bcosw)$

And now the same algorithm for computing chaos melodies is shown, with the real and imaginary parts of the new function plugged in:

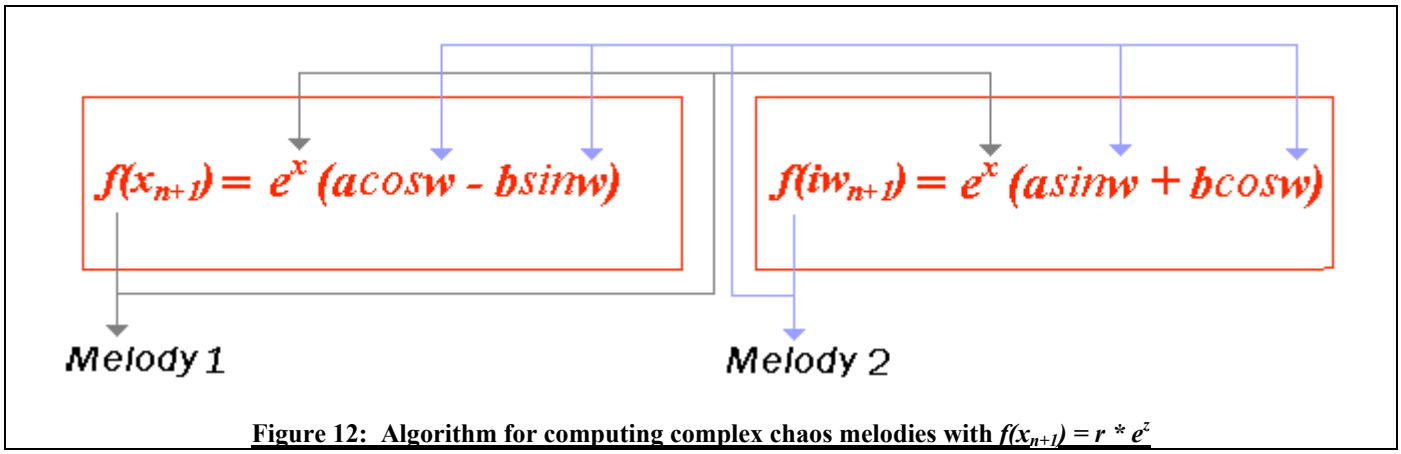


Figure 12: Algorithm for computing complex chaos melodies with $f(x_{n+1}) = r * e^z$

Melody 1 and Melody 2 form a counterpoint of two different, yet inherently related melodies. Some "piano scroll" examples of complex melodies are shown below. The first two examples are derived from $f(x) = 1 - rx^2$ in the complex domain.

Melody 1 derived from the real part of $f(x)1 - rx^2$.

Melody 2 derived from the imaginary part of $f(x)1 - rx^2$.

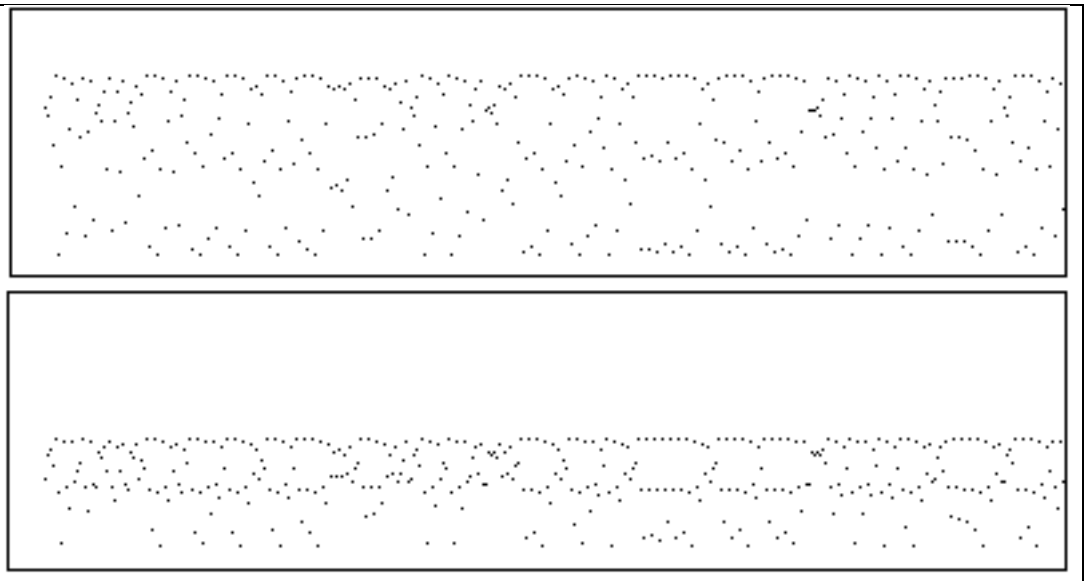


Figure 13: 1-rx^2 Complex Melody Example: a=1,b=.68,x=-.57,w=-.9

Only the 1st is shown in this example. B is set to 0, which resulted in Melody 2 only consisting of 1 note over and over. Melody 1 is periodically switching between two unstable orbits. It never escapes to infinity, but doesn't seem to be able to decide which orbit to stay in. In the complex domain, this is a common trait.

Melody 1 derived from the real part of $f(x)1 - rx^2$.

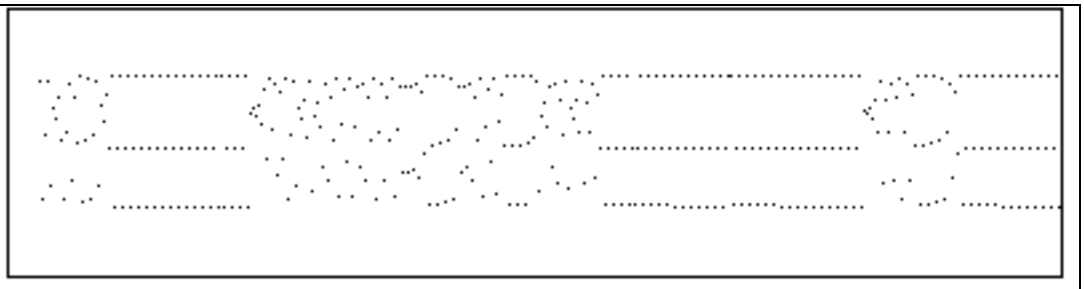
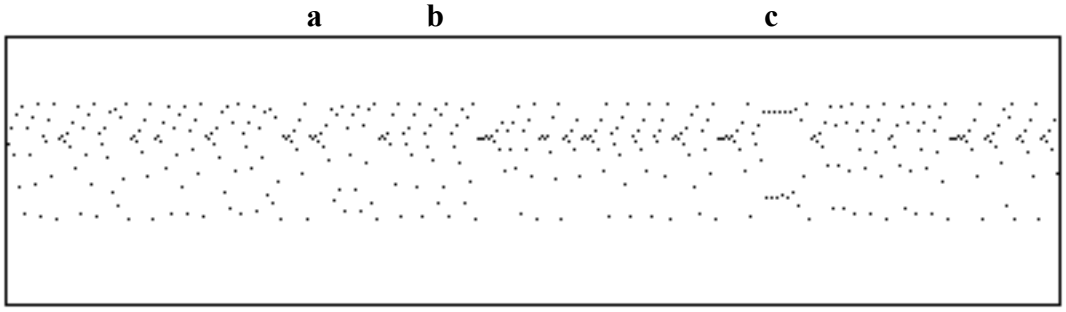


Figure 14: 1-rx^2 Complex Melody Example: a=1,b=0,x=-.57,w=-.9

The following melody also has b set to zero, so only Melody 1 is shown. This melody shows clearly the signature characteristic of the function $f(x) = 1 - rx^2$. The sideways V shaped patterns reoccur often in this melody, and even fractal versions of it. One of the many sideways V patterns is labeled with an a. The area under the b is an example of how the melody takes twice as long to play a V shape, due to notes near the bottom that happen in between each of the notes inside the V shape. The area under c shows an example of two very compressed V's are happening at the same time.



Melody 1 derived from the real part of $f(x)1 - rx^2$.

Figure 15: 1-rx^2 Complex Melody Example: a=1.55,b=0,x=-.99,w=-.99 2

These are all fractal characteristics. They are examples of how *Chaos Melody Theory* is a source of motifs, or building blocks, that organically produce copies and larger and smaller versions of themselves. This technique of modifying simple patterns to create larger musical works with a coherent theme throughout is common in many types of traditional music, both ancient and modern. It is a musical trait that composers and performers naturally produce, and here, it is produced naturally with elegant mathematics, molded and shaped by the musician, like clay in the hands of an artist.

The next two examples show the real and imaginary melodies of the function $f(x) = r * e^z$. The melodies exhibit an interesting pattern that goes between two unstable orbits in the first example, and between three unstable orbits in the second example. A very short phrase section can be seen at the beginning of the first example. As long as the melody gets pulled into an orbit, or a series of interchanging orbits, and does not race off to infinity or tend towards zero, it is considered a chaos melody.

Melody 1 derived from the real part of $f(x) = r * e^z$.

Melody 2 derived from the imaginary part of $f(x) = r * e^z$.

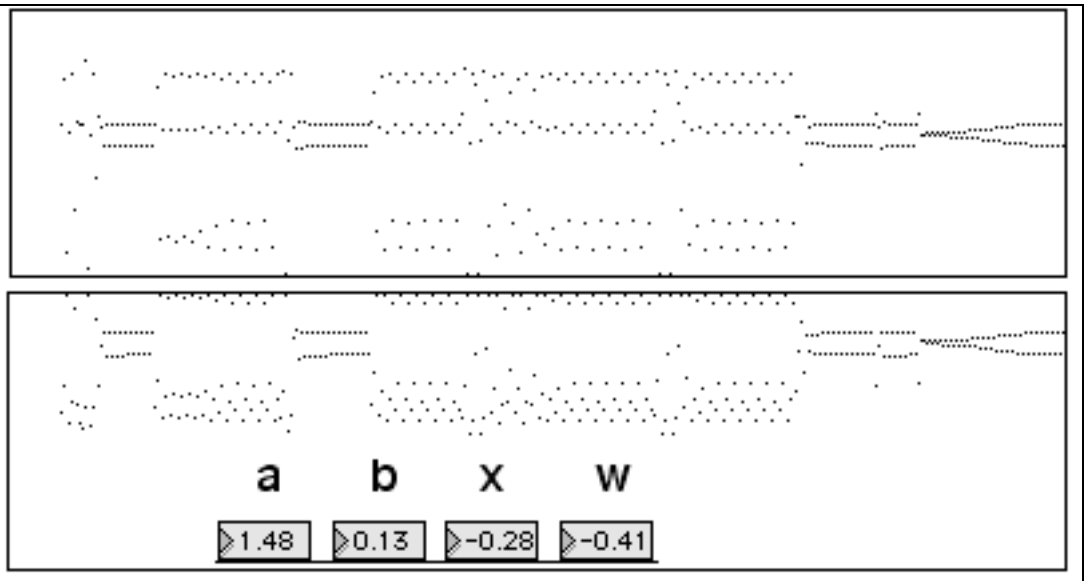


Figure 16: $f(x) = r * e^z$, a=1.48,b=.13,x=-.28,w=-.41

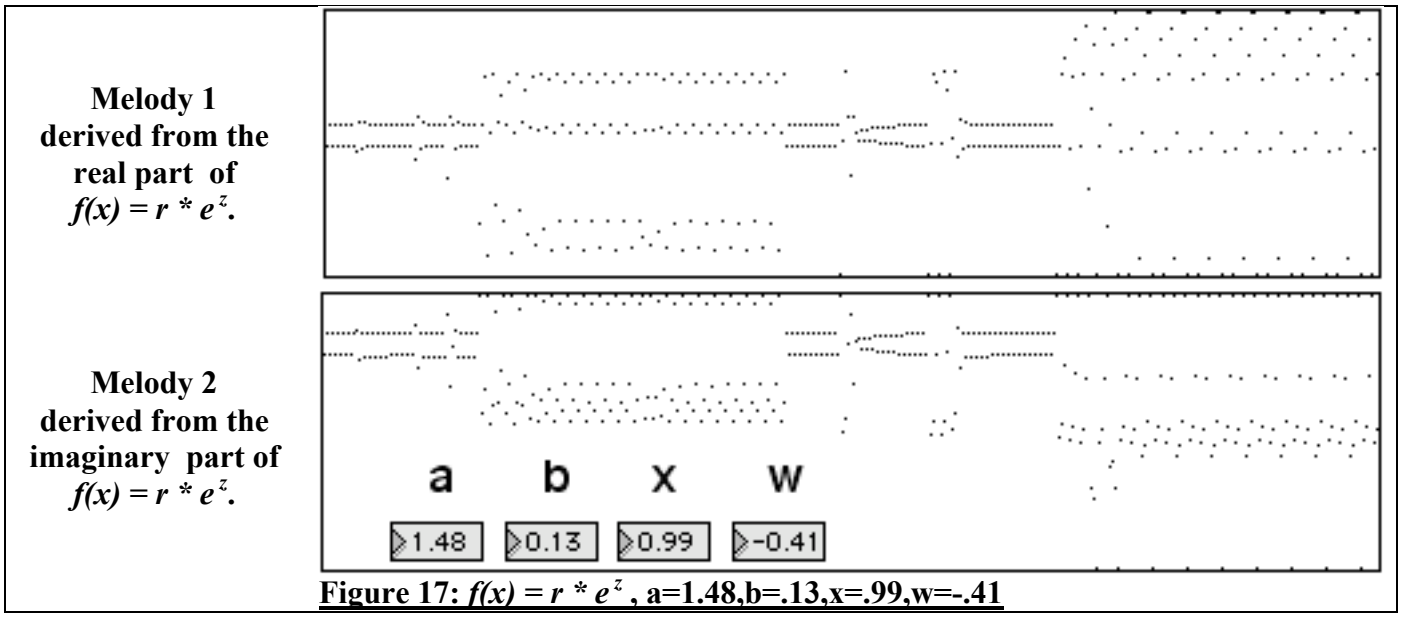


Figure 17: $f(x) = r * e^z$, a=1.48,b=.13,x=.99,w=-.41

The example is in the real domain, and shows both the chaos and seed values being modified, creating a fragment of a musical work. It is labeled with an ***x*** every time a new seed value (or ***x*** value) is initiated. When the notes tend towards zero (labeled with [zero] or [z]), a new ***x*** is initiated shortly thereafter. This happens at least four times in this example. When the fourth ***x*** is initiated, the chaos value (or ***r*** value) is raised continually resulting in a bifurcation, and then lowered. After that, the iteration stops momentarily, for a rest in the piece, and re-initiated with an ***x.*** An unstable phrase starts, which gets pulled towards zero again. The ***x*** is initiated a couple of times, and then the phrase finally gets pulled into a repeating orbit.

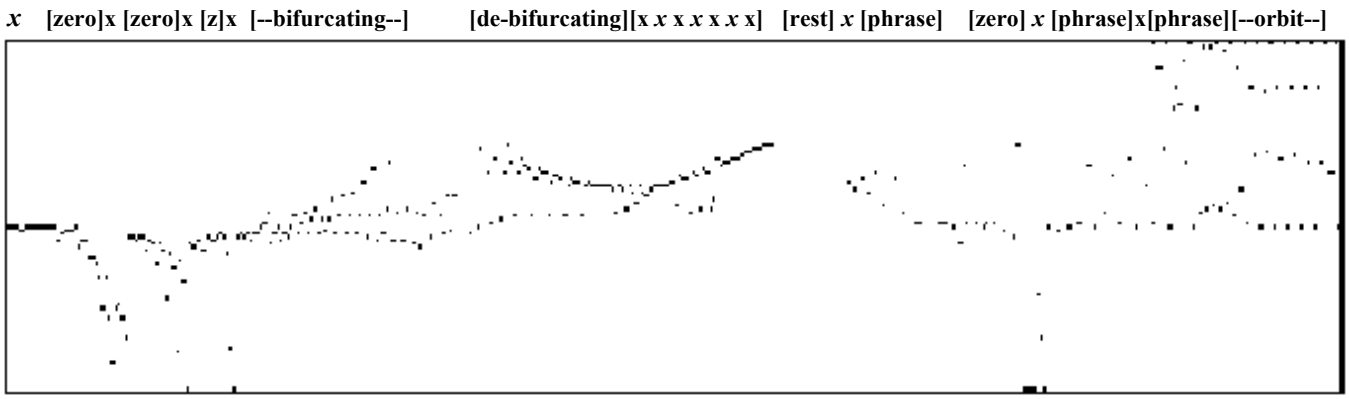


Figure 18: Creating a Musical Passage from $f(x)=1-rx^2$

This is just one example of how the parameters might get manipulated to create a piece or music. The seed and chaos values are pushed and nudged in different directions, to form an organically morphing pattern. It is not an easy task to create a coherent piece of music, and it helps to become familiar with the dynamics of the function being iterated. Preset values and ranges help to control the process as well.

Fractals - The Mandelbrot Set and Julia Sets

Chaos theory and fractals are closely related, and it may be of interest to the reader to relate chaos melodies to pictures of fractal structures. Before going much deeper into how these melodies can be molded into actual works of chaos music, it may be useful to diverge for a moment in order to get a bigger (or smaller?) picture of where these melodies are coming from. Pictures of fractals are plotted on a complex plane. Below is a picture of the Mandelbrot set.

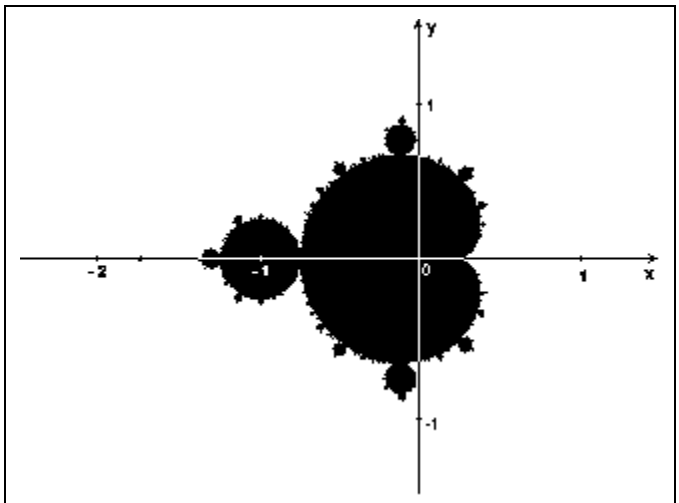


Figure 19: The Mandelbrot Set

The Mandelbrot Set is thought by some to be the most beautiful and complex structure in mathematics. However, this black and white depiction of the Mandelbrot set actually contains very little data. This picture is a far cry from some of the colorful pictures that have been produced. Only a low resolution set of black pixels has been computed to create this picture. One could also imagine extracting the information in this picture from a more detailed picture of the Mandelbrot set as a type of data compression. Whether it was initially computed with lower resolution, or whether this data was selectively extracted from a higher resolution image, in a sense, it is only a loose representation of the Mandlebrot set. However, this low resolution black and white picture still contains a lot of extremely interesting and useful data.

In the same way that the infinite detail of the Mandelbrot set (or any fractal image) can not be truly represented on a two dimensional plane, it is unimaginable to use all of the data in one coherent piece of music. There would simply be too much information! In the same sense, however, a composer could simply compute an extremely low resolution fractal image and use that data for musical purposes. Alternatively, the composer could extract data from a higher resolution structure and assign that data to musical parameters. Unlike *Chaos Melody Theory*, other composers of algorithmic music use this method.

The function used to compute the Mandelbrot set (and Julia sets) *is* $f(Z) = Z^2 + C$. In this function, Z and C are complex numbers. This means that where $Z = x + iy$, x is the real part of Z and iy is the

imaginary part of ***Z*** (Note that ***C = a + ib***). Another type of set produced by iterations is called a Julia set. A different Julia set is represented by every point on a complex plane.

In the function used to compute the Mandelbrot set and Julia sets, $f(Z) = Z^2 + C$, ***Z*** and ***C*** are complex numbers. ***Z=x + iy,*** where ***x*** is the real part of ***Z*** and ***iy*** is the imaginary part of ***Z,*** etc. A Julia set is represented by each point on the complex plane. The following series of pictures show how Julia sets associated with points outside of the Mandelbrot set are disconnected in quality, where not all of the black portions are touching, and Julia sets associated with points well inside of the Mandelbrot set are more solid and uninteresting. Julia sets that are associated with the extremely complex outline (or the basin boundary) of the Mandelbrot set are the most visually interesting. It is these points on the complex plane that are near the edge of the Mandelbrot set that exhibit the most exotic patterns.

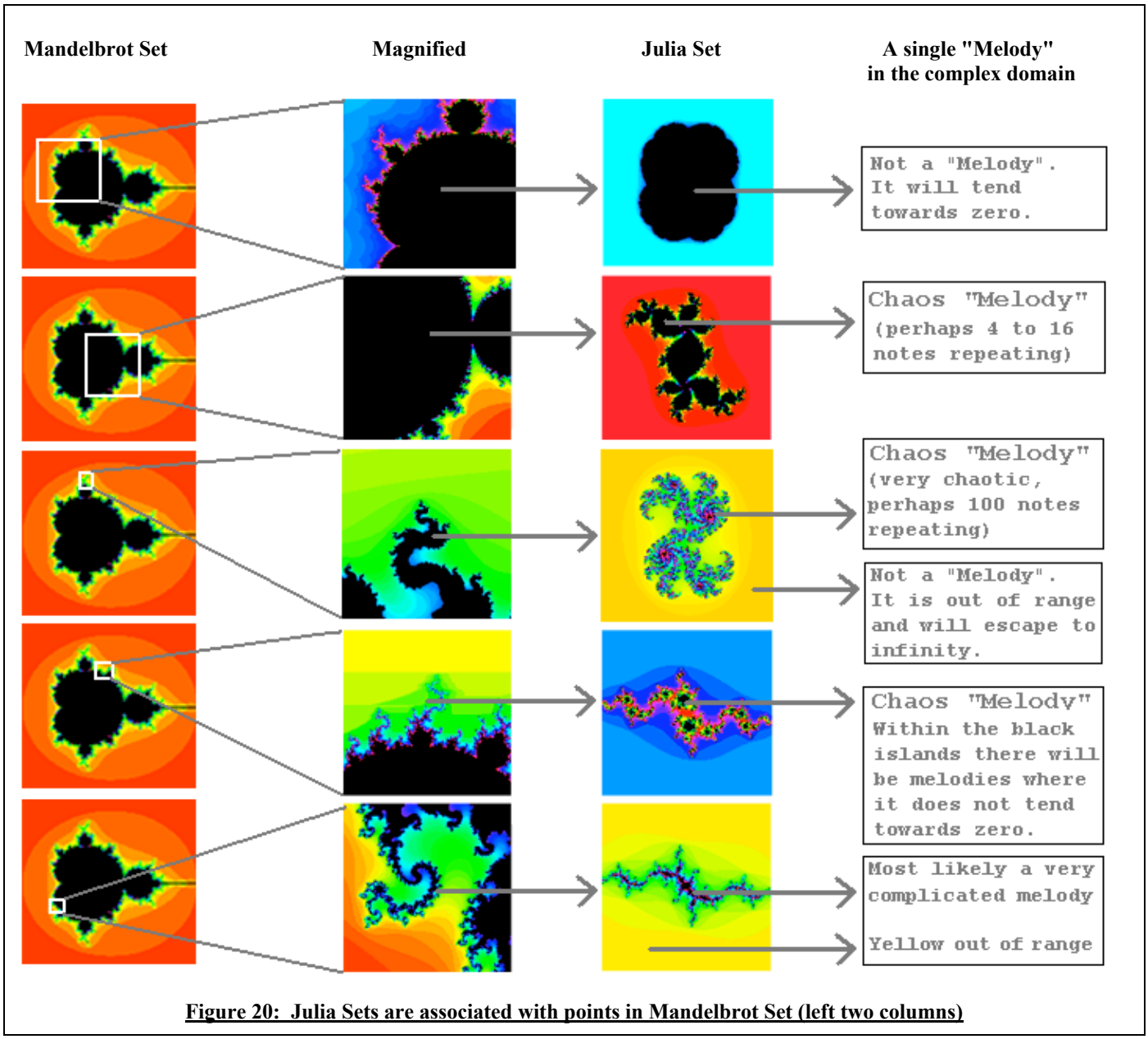


Figure 20: Julia Sets are associated with points in Mandelbrot Set (left two columns)

Just as one Julia Set is represented by each point on the complex plane, and closely related to the Mandelbrot set, a chaos melody is represented by each point on a Julia Set, and is closely related to that Julia Set. Melodies derived from points well inside the black areas, or "basins", are very consonant and perhaps uninteresting. These pockets of consonance are referred to as "basins". Melodies derived from points in the colored areas escape to infinity and are not even considered true chaos melodies. It is extremely complicated around the edges of the basins, and this is where the most interesting and useful melodies are derived from. These areas are called the "basin boundaries".

To further understand how chaos melodies are connected to pictures of Julia Sets, it is necessary to take a closer look at how the Mandelbrot and Julia sets are drawn. Then it will be clearer that chaos music only uses a tiny fraction, yet such an incredibly detailed portion, of a Julia set, which is in turn an extremely small yet fabulously detailed portion of the Mandelbrot set!

Computing the Mandelbrot Set:

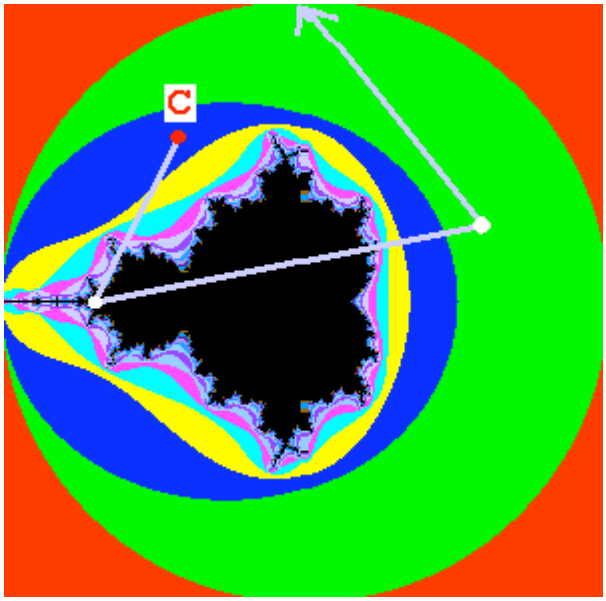


Figure 21: Escape cycle of a value of C that is outside of the Mandelbrot set.

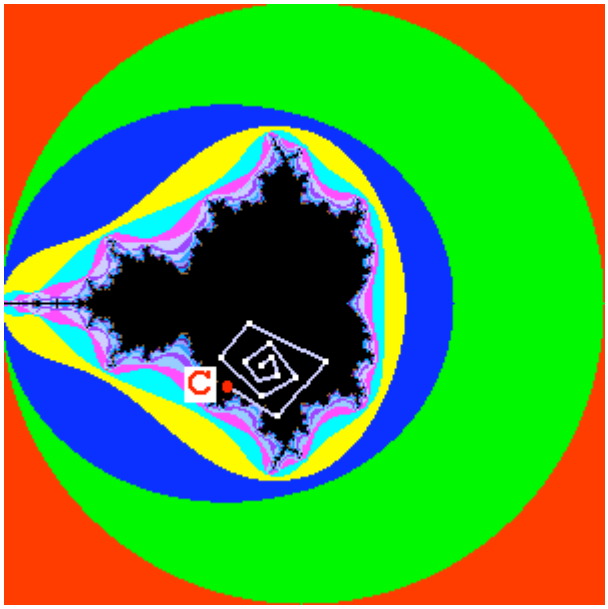


Figure 22: Repeating cycle of a value C contained within the Mandelbrot set.

The Mandelbrot set is computed by iterating the function **f(Z)=Z^2 + C** in the complex domain. However, to compute the Mandelbrot set, the **Z** value always starts at zero. To make a detailed picture, the function is iterated with a starting value of **Z=0**, for every possible **C** value. In other words, **C** is the set of all points on the complex plane. The values of **C** that cause **Z** to eventually fall into a repeating cycle are usually plotted with black pixels. The values of **C** that cause **Z** to eventually escape out of a

circle of radius 2 (and eventually escape towards infinity) are usually either plotted with white pixels, or assigned to a color palate that relates to the amount of iterations necessary to escape out of the circle of radius 2. Once a specified number of iterations for each value of **C** takes place (depending on the software and desired resolution of the final picture), the fate of the pixel associated with the initial value of **C** is determined. Either it has escaped outside of the circle of radius 2, and is colored accordingly, or it has fallen into a repeating cycle or fixed point and is colored black. In other words, to draw the picture, every pixel must be tested and then colored accordingly. And only a limited number of points on the plane can be tested, depending on the computing power available and the resolution desired for the final picture of the Mandelbrot set.

In the pictures above, the Mandelbrot set is in black, inside the green circle of radius 2. Figures 21 and 22 show two points being tested to see if they belong in the Mandelbrot set. The red dot in Figure 8 is a point on the complex plane that was found to not be contained in the Mandelbrot set since it escaped outside of the circle of radius two with only four iterations. This point would be colored blue, according to the color palette used for this image. The red dot in Figure 22 represents a point on the complex plane that was shown to belong to the Mandelbrot set. After several iterations it is quite apparent that it will not escape outside of the circle of radius 2.

Computing Julia Sets and Chaos Melodies:

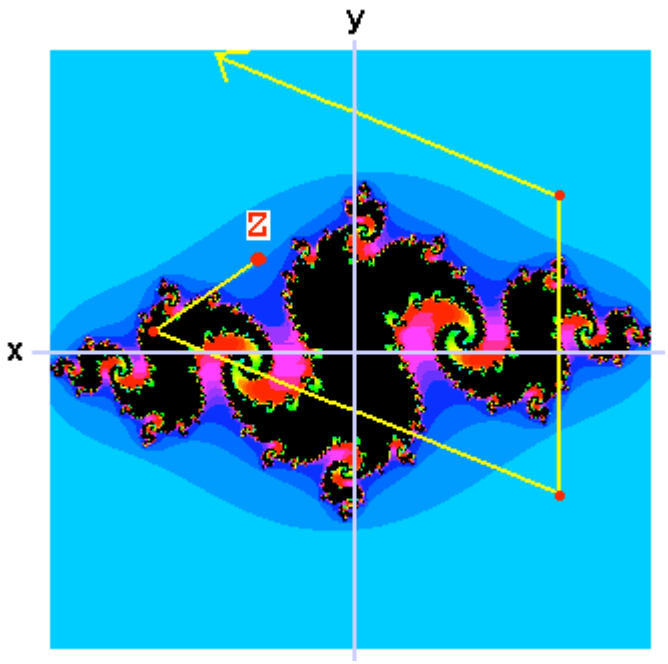


Figure 23: Not a "melody"; the values escape towards infinity after only four iterations.

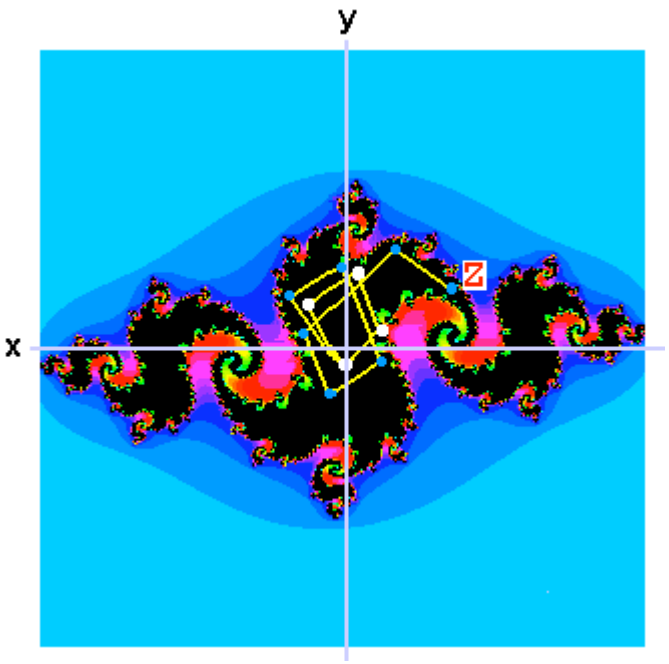


Figure 24: An example of a simple "melody" with a seven note phrase and four note orbit

Figure 20 shows how Julia sets are visually connected with the Mandelbrot set. Mathematically, the iterative function that is used to produce them is also*: **f(Z) = Z^2 + C***. What is different is the way this formula is used. As was shown above, in order to draw a picture of the Mandelbrot set, the formula is iterated separately for each point ***C*** of the complex plane, always starting with ***Z0 = 0***. When making a picture of a Julia set, ***C*** remains fixed during the whole generation process, while different values of ***Z*** get iterated. The value of ***C*** determines the shape of the Julia set. In other words, each point of the complex plane is associated with a particular Julia set.

To create one discrete chaos melody basically involves the same process that is used to compute one pixel at a time of a Julia set image. However, instead of using this process as a test to decide what color to make one pixel on a computer screen, all of this bouncing around on the complex plane is used to create a musical melody (actually two simultaneous melodies). ***C*** is left at a constant value and the equation is iterated with an initial starting value of ***Z***. As the function is iterated, the output will either jump out of range (see the red dotes in Figure 24), fall towards zero, or get pulled into a repeating cycle or orbit (see Figure 24). Figure 24, shows a real chaos melody, where the initial meandering 7 note pattern (shown with blue dots) gets pulled into a four note orbit (shown with white dots).

In the case of Figure 23, the starting ***Z*** value would have been colored a shade of blue, according to the color palette because it escaped with only four iterations. The initial value of ***Z*** in Figure 23 would have been colored black because it is apparent it will not get pushed out of bounds, and instead it is trapped in a four-cycle orbit. In other words, chaos melodies are completely lost in a picture of a fractal, because an entire melody has just become one black pixel, lost within the black basins of the fractal image. In *Chaos Melody Theory*, rather than extracting data from an entire fractal image to create music, each chaos melody explores the very find detail of one wondrous pixel.

This is a very simple example of a melody. More often than not, the phrase and orbit will be much more complex. As we have seen in the previous section, a phrase may get pulled into a series of unstable orbits, switching orbits from time to time. It is very hard to predict when an orbit will suddenly become unstable enough to get pulled into a new trajectory, and this is one reason that it may be more desirable to create chaos melodies in the real domain. The saving grace is that this music is not random, and therefore can have preset parameter values stored and recalled. In addition, musical passages can be sequenced, simply by recording the parameter changes. If the software is written correctly, each time a function is iterated with exactly the same parameter changes, same melody should result.

A realistic conclusion can be drawn, that when doing live improvisation of chaos melodies, it may be preferred stay within the real domain, creating only one melody at a time. Melodies in the real domain tend to be more robust and predictable. When sequencing an entire composition, it may be preferred to use the complex domain in order to create a counterpoint of two melodies, spending quite a bit of time molding and shaping the parameters into a fluid composition, sequencing and saving fragments as they are composed, until a complete piece is finished.

Controlling Chaos Melodies

Due to the recursive mathematical nature of how chaos melodies are generated, it is necessary to have dedicated software or hardware as a performance tool in order to improvise and compose complete works of music. The concepts discussed in this paper have been incorporated into software called the "Chaos Melody Maker" and a hardware device called the "Chaos Controller". Both are still being developed and perfected but for the most part are fully functional. They are examples of the kinds of tools under development in the realm of *Chaos Melody Theory*. The accompanying audio CD contains a 20 minute, four movement piece, created with the "Chaos Melody Maker" using the function $f(x)=1-rx^2$ in the real domain, one five minute improvisation on the "Chaos Controller" instrument using the function $f(x)=1-rx^2$ in the real domain, and several examples of phrases and orbits generated with the "Chaos Melody Maker" using the function $f(x) = e^z$ in the complex domain. A track listing is included with the audio CD.

How does the performer actually go about controlling and performing chaos melodies? Firstly, it may help to simply get into the right frame of mind. One must get sufficiently far enough away from their years of formal music training and notions of traditional counterpoint and chord progressions, or the process will only be a frustrating one. In fact, there is no specific tuning (such as the widely used 12 tone equal temperament) specified for *Chaos Melody Theory*. Although it does regard pitch modification, the streams of numbers output from the iterated functions must be scaled, and the amount of scaling is up to the composer and/or programmer. Traditional music training may be helpful, however, regarding a performer's sense of phrasing, dynamics, and the general shape of a coherent musical work. People with no formal music training whatsoever can enjoy producing wonderful music using these tools, and seasoned musicians may find it a refreshing change and a creative stimulus.

Figuratively speaking, the mathematics is the engine, which the performer steers down a pathway or network of roads that are predetermined by the real mathematical possibilities themselves. The roads can be thought of as the set of all possible melodies and bifurcations, and these roads lead to many interesting places. The performer also has free control over speed (tempo), acceleration (accelerando/ritardando), and starting and stopping (rhythm and note duration), within the limits of the mechanics of vehicle (the hardware or software). In other words, the mathematics does not control any of the dynamics or rhythmic content, but leaves that up to the performer so that in a sense they are "driving" the mathematics.

Literally speaking, the performer will initiate a mathematical feedback loop with initial seed and chaos values. The numbers output are scaled to a range somewhere in between 0 and 127 and mapped to MIDI note numbers, which are sent to a MIDI sound module. Further "pitch scaling" can be done in the MIDI sound module that is being triggered, in the sense that the sound being played may be microtonal. The output will continue it's orbital cycle until the performer initiates new seed values or chaos values, these parameters are dynamically re-initiated or swept up and down at will. The performer will simultaneously be controlling the dynamics and phrasing, tempo and rhythm with their arms, hands, feet, or any control device that has been invented for this purpose.

"Chaos Melody Maker™" Software

The "Chaos Melody Maker™" (see Figure 25) was programmed in Opcode's Max; a visual programming environment in which math, text, and other "objects" are connected together with virtual cables to create an endless variety of algorithmic music programs and other music related virtual gizmos. This is one example of how software can be created, not only as a live performance and improvisation tool, but also as a sequencing environment specifically for chaos melodies.

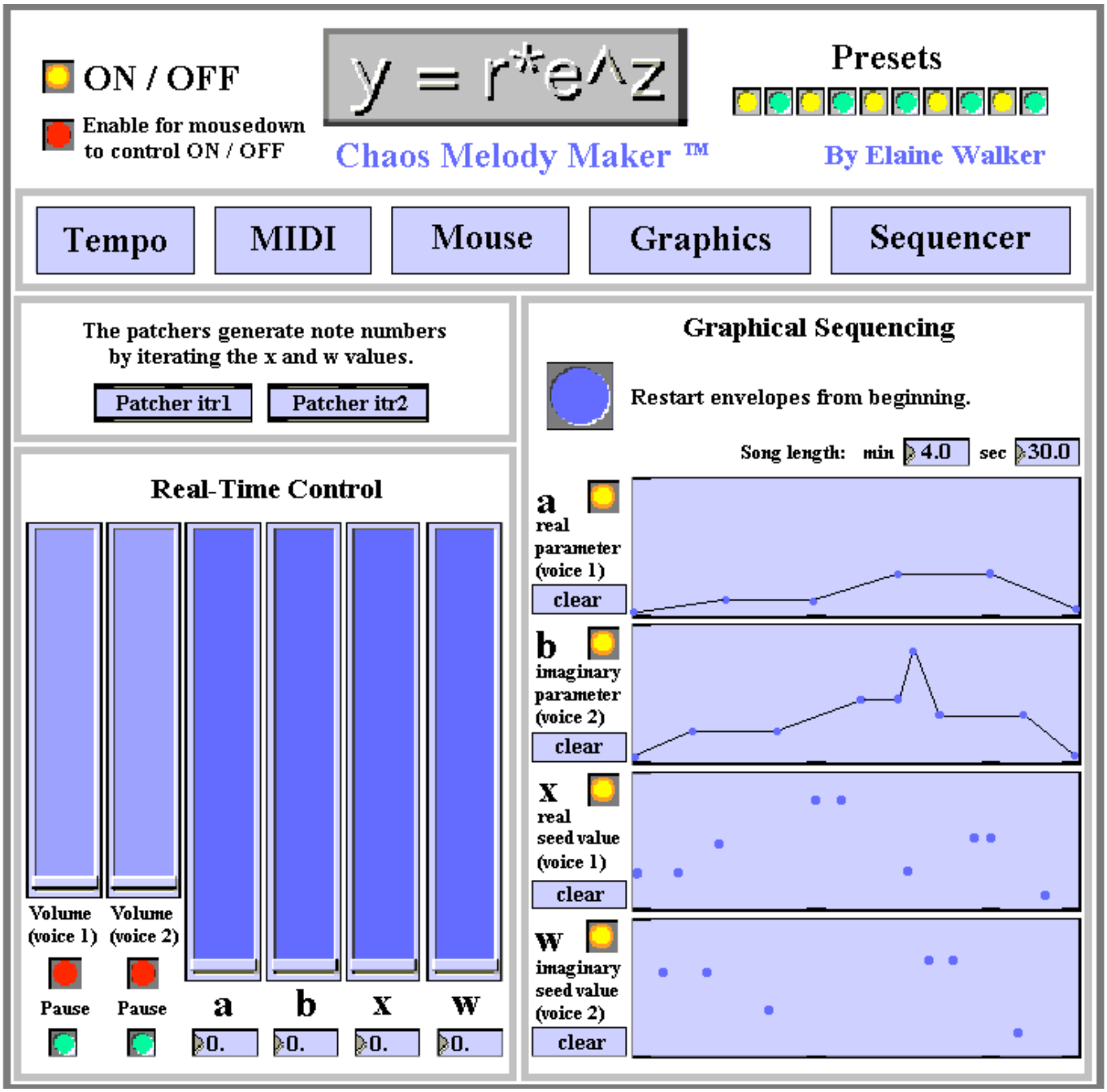


Figure 25: The front page of the "Chaos Melody Maker" ™ MAX patch

In the "Chaos Melody Maker", objects named "patcher itr" and "patcher itr2" contain the mathematical feedback loops that are the underlying machine behind the music being created. There are a few versions of the "Chaos Melody Maker". One iterates a function in the real domain with the option of having two melodies simultaneously playing, one being scaled in time to create interesting counterpoint. The example on the audio CD (tracks 1 through 4) was composed with that particular version using the function $f(x)=1-rx^2$ in the real domain. Any function can easily be plugged into the software, simply by pasting over the old function with the proper syntax.

The version of the "Chaos Melody Maker" shown in this section uses the function $f(x) = r * e^z$, extend it into the complex domain, as was shown in the section on Complex Chaos Melodies. The real portion of the function, $e^x (a\cos w - b\sin w)$, is incorporated in the "patcher itr" (see Figure 26), and the imaginary portion, $e^x (a\sin w + b\cos w)$, is incorporated in the "patcher itr2" (see Figure 27). The expressions below were written in the "expr" object in a way that Max understands. Each variable that is to be controlled must be represented as **$f1**, **$f2**, etc. In this case, a = **$f1**, b = **$f2**, x = **$f3** and w = **$f4.** There is an inlet for each variable, with a virtual cable connected to each. (Max uses the expression **exp(x)** to represent the number e^x.)

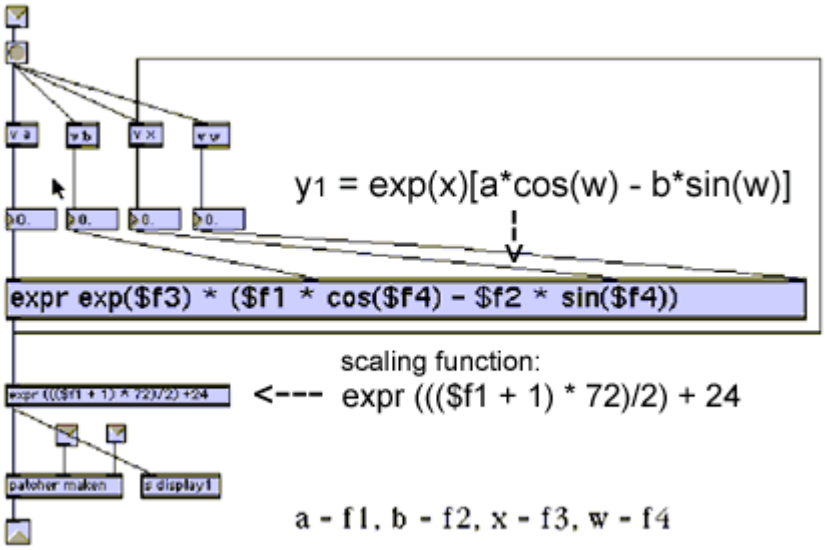


Figure 26: Inside the "patcher itr" object. Max feedback loop for the "real" part of the function

In the "patcher itr", the equation is solved for y_1 and the answer is sent out the outlet. The values are then scaled with the "expr" object that contains the expression $(((\$f1 + 1) * 72)/2 + 24$. This puts the numbers in a proper range for a 72 note keyboard (somewhere in between 0 and 127 which is the range that MIDI devices recognize). It then goes directly to the "makenote" patcher, which turns the stream of numbers into MIDI "note on" (pitch) information, and then is sent out of the computer to the desired MIDI module. However, before being scaled, it is also fed right back into the x value in BOTH patchers.

Consequently, the "patcher itr2" is solved for y_2 and fed back into the ***w*** value in both patchers. y_2 is also scaled and turned into MIDI note information and sent to the MIDI module on a separate MIDI channel.

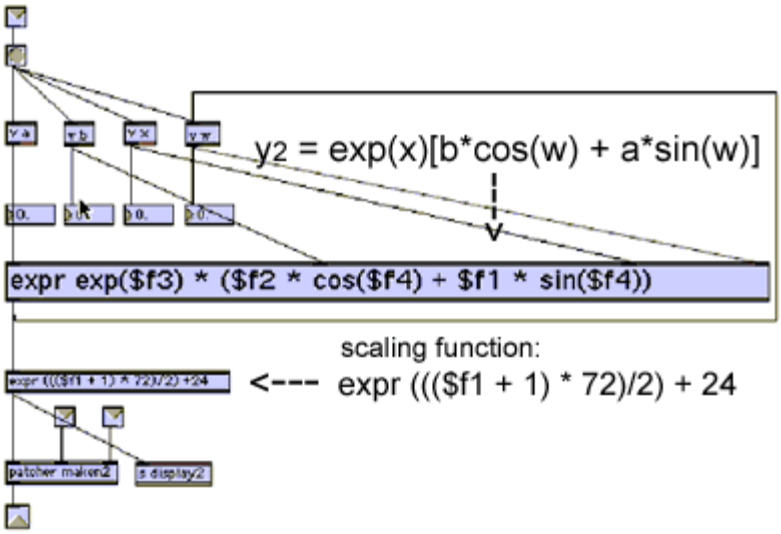


Figure 27: Inside the "patcher itr" object; The feedback loop for the "imaginary" part of the function

Simply stated, the two feedback loops are feeding into each other continuously, while at the same time are outputting values that are mapped to MIDI note numbers. The first feedback loop is generating a melody with the "real" part of the complex equation, and the other with the "imaginary" part. The result is two simultaneous melodies that are similar and exhibit the same general behavior, but are slightly different in melodic content. This creates a kind of chaotic counterpoint, and works well when one sound is mapped to a low range of pitches and one to a higher register. Changes to either ***a, b, x,*** or ***w*** that the performer initiates are dynamically fed into the "patcher itr" and "patcher itr2" objects overriding any values that are presently there. These values are initiated with input devices such as the mouse, computer keyboard, external MIDI sliders, a MIDI keyboard, or any such input device imaginable that can communicate through the Macintosh serial port.

When an iteration is initiated with a new value of ***x*** or ***w***, the phrases of both the real and imaginary melodies start with similar patterns, usually one a compressed version of the other, then simultaneously fall into similar orbits, or cycles of orbits. The orbital cycles will repeat until the performer intervenes with a new value of ***a, b, x*** or ***w***. A new value of ***x*** or w will start the entire iteration over, with a different phrases, and these phrases eventually fall into the same orbit as before, as long as the ***a*** and ***b*** values are left alone. If the same value of ***x*** or w is fed in as the previous iteration, then exactly the same phrase and orbit should result. If an interesting pattern is found that the user wants to recreate in the future, the current parameters can be stored by shift-clicking on one of the "preset" buttons towards the top (see Figure 25). Every time the same preset button is pressed, the same music should result. In fact, since the chaos melodies are not random, the chaos music can be sequenced.

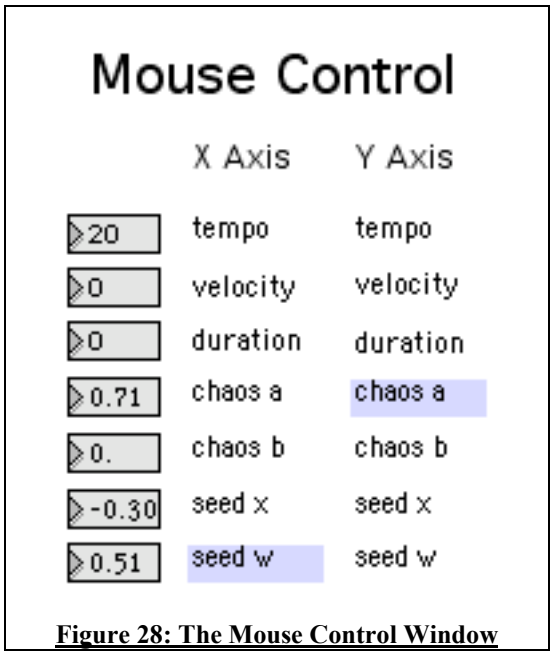


Figure 28: The Mouse Control Window

By double-clicking on the "Mouse" button near the top of the screen in Figure 25, the "Mouse Control" window to the left appears. The user of the program can use the two axes of the mouse to vary the ***a*** and/or ***b*** parameters over time, having continuous control over the chaos parameter of the melodies. New ***x*** and **w** seed values can be initiated at any time as well.

If the sound that is being triggered is programmed to have MIDI velocity control the loudness of the sound, then one way to control phrasing and dynamics would be to simply assign one of the axes of the mouse to "velocity". In Figure 28 to the left, MIDI velocity is being controlled by the ***y*** axis of the mouse. Therefore, every time the mouse is moved up or down on the screen along the ***y*** axis, the velocity value will be raised and lowered accordingly, and the sound will get louder and softer.

The feedback loops in the patchers can be started and stopped at any time by clicking the mouse down or by enabling the ON/OFF switch (see Figure 25, top left corner). The user can easily improvise with the function by riding the ***a*** and ***b*** parameters up and down to control the chaos, starting and stopping the feedback loop for rhythmic variety, and initiating new ***x*** and ***w*** values from time to time to start new phrases. Tempo, velocity and duration can also be dynamically controlled with the mouse. Computer keyboard commands have been set up to start and stop the loops, and to initiate the presets, which would normally be different combinations of ***a, b, x*** and ***w*** values, and tempo settings.

Sequencing a Complete Chaos Music Composition

It is quite common to use a standard MIDI sequencer to record the MIDI events in a musical composition. Many such sequencers exist on the market such as Emagic's "Logic", Mark of the Unicorn's "Performer" or Opcode's "Vision" software. In the future, composers may want to sequence their chaos music so that they can play back the MIDI events from their composition or improvisation. In much the same way as standard MIDI sequencers are actually used as a compositional tool, beyond simply recording MIDI events, a chaos melody sequencer can be used as a compositional tool. On the front page of the "Chaos Melody Maker" (Figure 25), it is possible to set a time limit for a composition (in this case it is set at 4 minutes and 30 seconds), and to draw in continuous changes for a, b, and to set ***x*** and w values that will be initiated throughout the piece. The changes occur every time the iteration is started. This concept is a little bit different than a standard sequencer, because it is not the actual MIDI note or MIDI controller information that is being sequenced, but only the parameter and variable changes over time. It is possible to capture the MIDI note events on the fly in the "Sequencing" window, while playing the sequence that was made by setting parameter and variable changes, and even to initiate additional values in real time along with the preset values in the graphical sequencer.

The "Chaos Controller™" Instrument

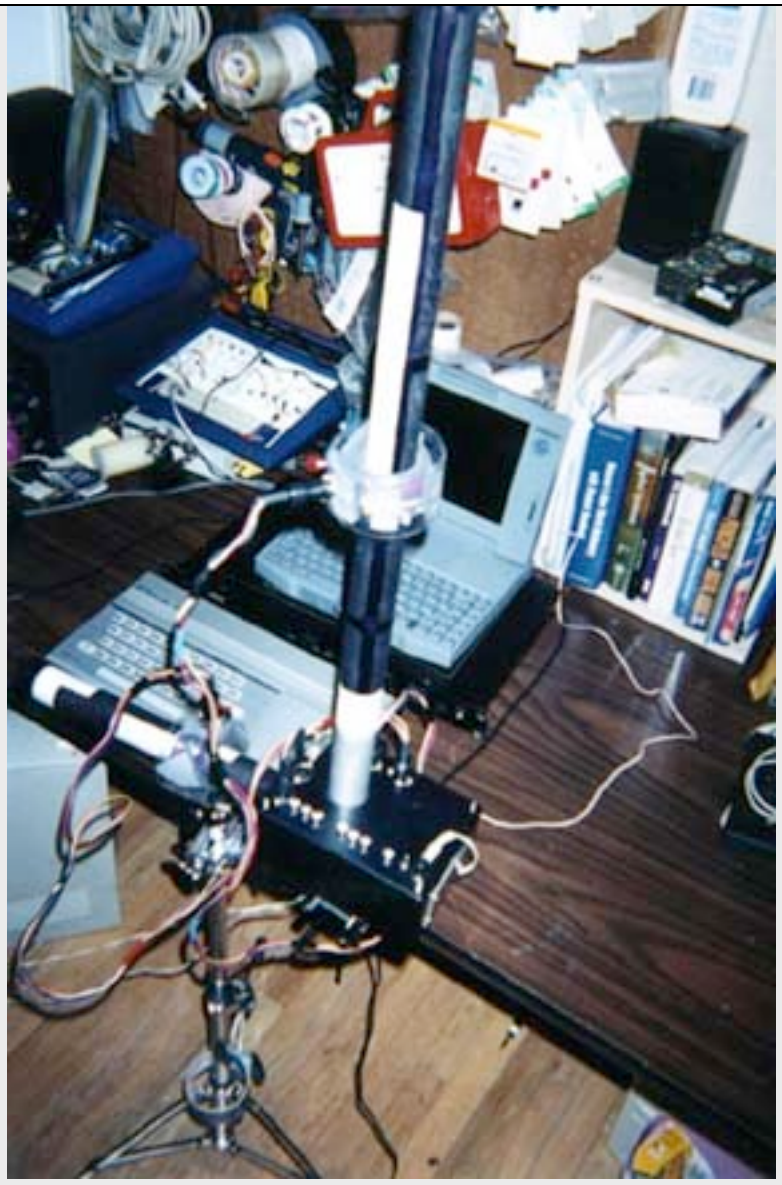
The Chaos Controller is a stand-alone hardware MIDI performance instrument that uses the Basic Stamp microcontroller to store and process a simple program that roughly emulates the Chaos Melody Maker software. As of this writing, the Chaos Controller is currently loaded with a program to iterate

$$f(x_{n+1}) = 1 - rx_n^2$$

in the real domain. One melody at a time is sent out the MIDI output as a stream of note on and note off events. Other MIDI control information can be mapped to the continuous control pedal. The device consists of a black box with two poles attached, one vertical and one horizontal, each equipped with a slider. The vertical axis controls the ***r*** value, and the horizontal axis controls the ***x*** value. Each slider can read eight positions on the pole, and each position is mapped to preset values for ***x*** or ***r*** respectively. Preset values are carefully selected and pre-programmed into the software before the Chaos Controller is used in performance. All possible combinations of the preset ***r*** and ***x*** values result in 64 possible melodies derived from the formula. Each ***r*** value will result in the same repeating orbit, with eight possible phrases, as a result of the eight preset ***x*** values. 64 possible patterns offer plenty of variety to make up one discrete musical composition, or live improvisation.

The performer plays the instrument by placing their left hand on the vertical slider, and their right hand on the horizontal slider, arms outstretched. One finger on the left hand presses a button on the vertical slider to start the music. This button starts a loop in the software that iterates the function repeatedly at a specified tempo. The tempo can currently be controlled by a foot pedal, or by repeatedly pushing the button for each note. If the button is pushed quickly, one iteration will happen each time, and if the button is held in, it will continually iterate the function. The slider can be moved up and down the pole through the eight different positions, dynamically feeding the function with the new ***r*** values. The new values only get sent if the button is pressed in. The horizontal slider has a button that resets the seed value (the ***x*** value) to the value corresponding to the current position of the horizontal slider.

It is possible to memorize the patterns fairly easily, by remembering the eight different orbits, and getting a feel for what each value of ***x*** will do at the beginning of each orbit. The performer normally makes small, subtle movements with both sliders, moving up and down through the eight possible positions. Visually, the playing style can be compared to that of a theremin. The sliders can also be twisted to 3 different positions around the pole that correspond to one of the eight possible values. This is possible due to the way the sliders actually read 3 bit code on the poles. The sliders each have three led's and photoresistors on the inside. The light of the led's is reflected off of black and white code on the pole into the photoresistors which in turn cause changes in voltage. The voltages changes are detected by the Basic Stamp chip which maps the 3 bit values to the presets for ***x*** and ***r***. The three bits around each pole are in a greycode pattern, so that moving up the pole will only result in one bit changing at a time.

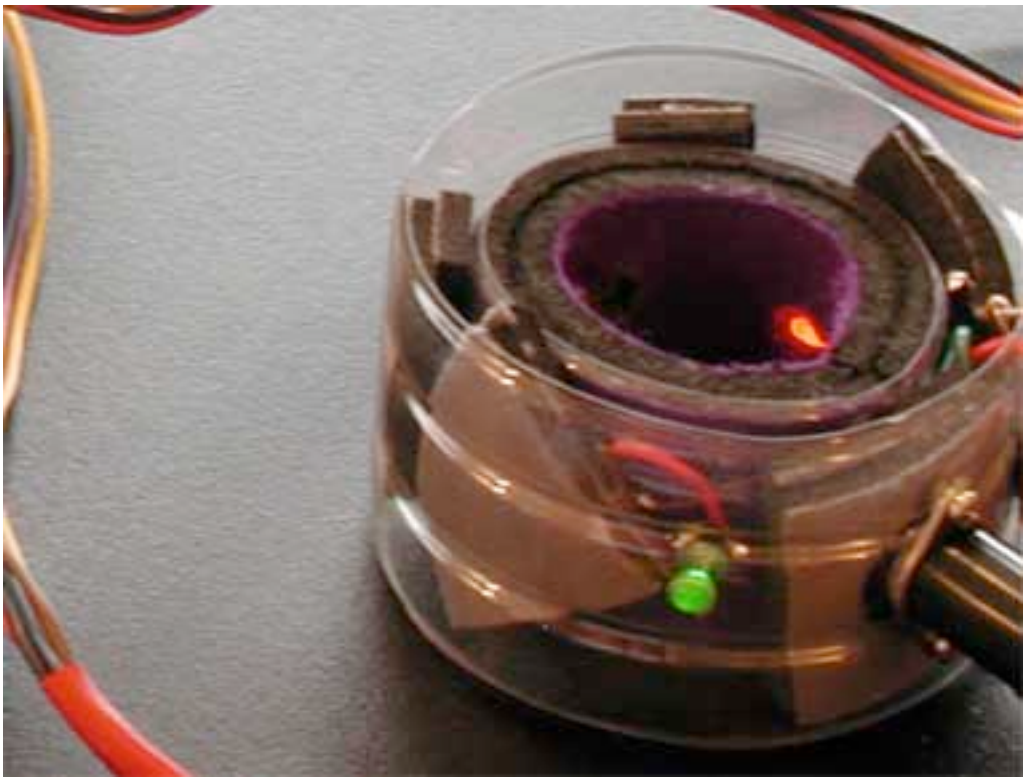


The Chaos Controller

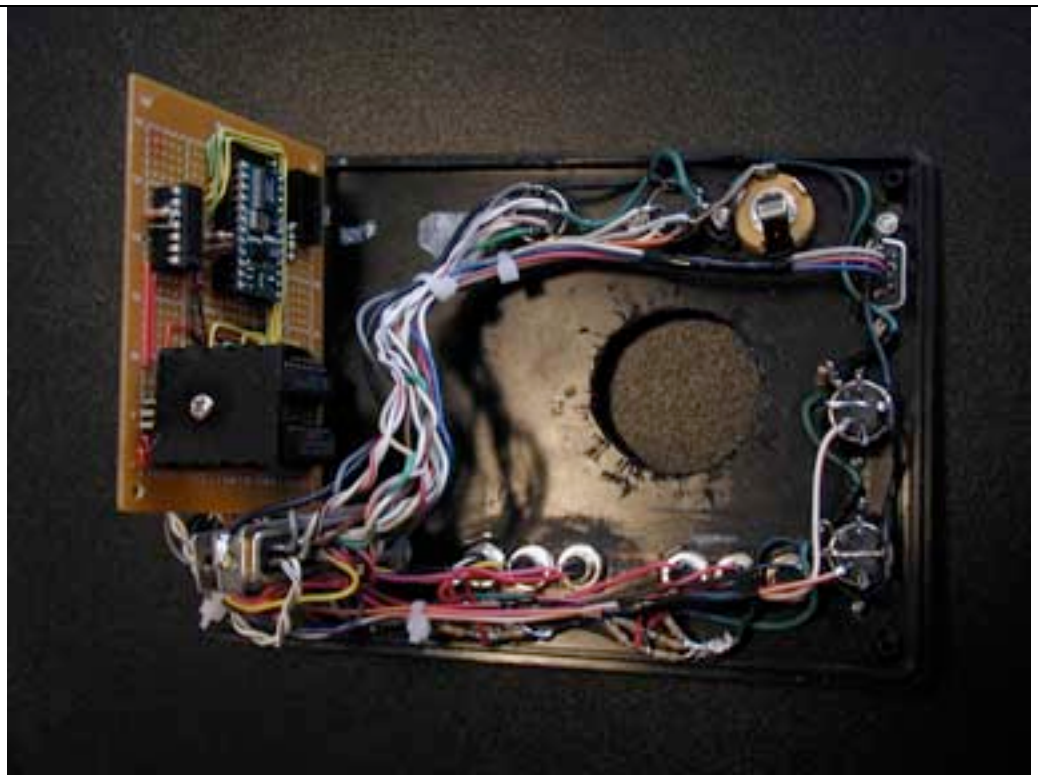
1	1	1					
1	0	1					
1	0	0					
1	1	0					
0	1	0					
0	1	1					
0	0	1					
0	0	0					
			0	0	0	0	1	1	1	1
			0	0	1	1	1	0	0	1
			0	1	1	0	0	0	1	1

Figure 29: Greycode pattern on the poles of the Chaos Controller (1's are white and 0's are black)

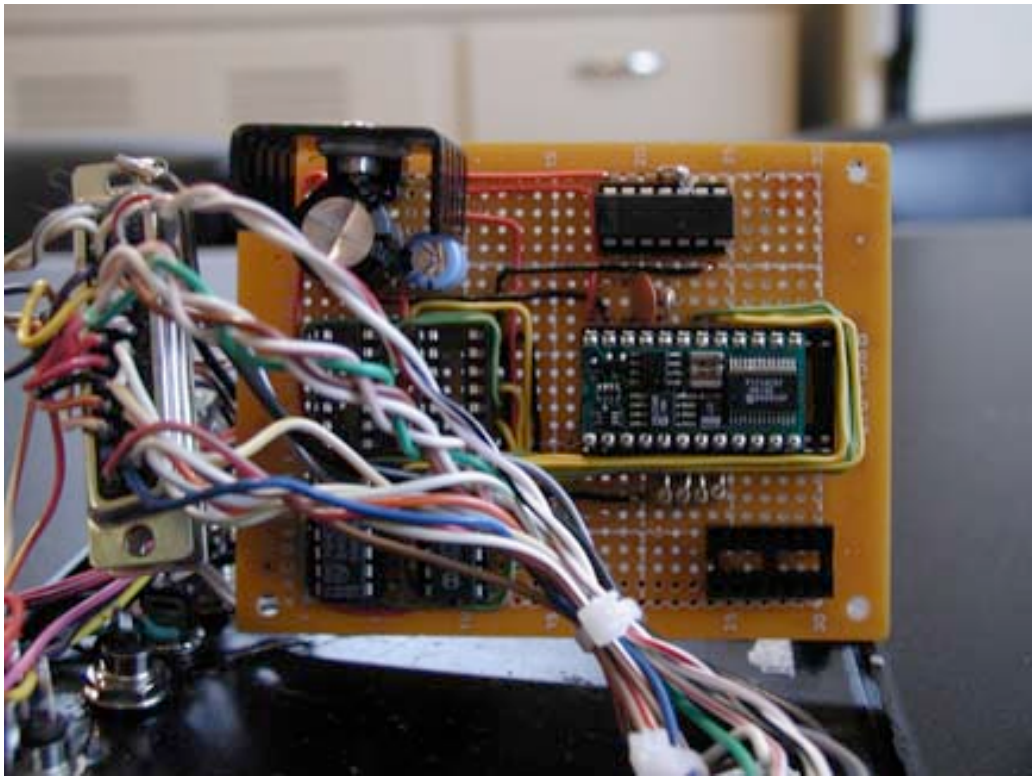
The black box containing the circuitry



One of two sliders



Inside the lid of the black box



Up close view of the circuit board

Lights on the black box turn green each time a slider passes over a white section of the code, and turn red when they pass over black sections. This can help the performer to keep track of what 3 bit number the sliders are reading, as it can get confusing since the sliders can also be rotated. Being able to rotate the sliders makes it possible to jump from one value to another that normally wouldn't be possible, simply sliding straight up and down the pole. It may be hard to memorize all of the rotational slider values, on top of remembering which melodic patterns correspond to the eight positions of the pole, and the lights are there to make the task a little bit easier. They also make a nice light display for the audience watching the performance.

Below is the code which is currently loaded in the Chaos Controller. It is written in P-Basic for the Basic Stamp microcontroller chip from Parallax (the largest chip in the picture above). It starts with an initialization procedure to set inputs and output pins, and variables. The lights blink on startup, and the program waits for the performer to press the red button on the vertical slider to start the mathematical iteration loop. Once the button is pressed, the program reads the 3 bit code from the vertical slider or "chaos slider". The Basic Stamp does not do floating point math, so some low resolution bit shifting binary math is necessary to make the computations. A new seed value can be set at any time with the horizontal slider, either while the iteration is looping, or while the iteration is paused. The output values are scaled, and sent as MIDI note data out the MIDI out jack, which can be connected to the input of any standard MIDI sound module.

```
'Chaos Controller, by Elaine Walker 2000
'This program is currently loaded into the Chaos Controller.


'================================INITIALIZATION=====================================

'-----------DATA FROM SLIDERS------------
input 0       'START button (starts the music)
input 1       'SEED button (sets seed value)

input 8       'photoresistor LSB (chaos value)
input 9       'photoresistor vld
input 10      'photoresistor MSB

input 11      'photoresistors for SLIDER 2 (seed value)
input 12
input 13

'-------------LIGHTS ON BOX--------------
output 7      'light indicator on box for SLIDER 1
output 6      'light indicator
output 5      'light indicator

output 4      'same for SLIDER 2 indicator lights
output 3
output 2

'--------------MIDI OUT PORT-------------
output 14     'serial out for MIDI
```

```
'-------------READ GREY CODE-------------
n var nib           'reads chaos slider 3bit grey code
code var nib        'puts code in the right order, 0 through 7
m var nib           'reads seed slider 3bit grey code
mcode var nib       'puts code in the right order, 0 through 7
hibits var word
medbits1 var word
medbits2 var word
product var word

'---------------MIDI CHAOS---------------
seed var word
chaos var word
noteout var byte
seedsquared var word
tempo var word
pedal var word
'duration var byte
'velocity var byte

seed = 45875
chaos =  55000
chaos = chaos - 32768
'============================BLINK LIGHTS AT STARTUP============================
startup:
     pause 1000
     out2=1
     pause 200
     out3=1
     pause 200
     out4=1
     pause 200
     out5=1
     pause 200
     out6=1
     pause 200
     out7=1
     pause 200
     outs=%00000000
     pause 200
     outs=%11111100
     pause 500
     outs=%00000000

'==============WAIT FOR PLAYER TO PRESS BUTTON ON VERTICAL SLIDER==============
waitforstart:

     high 15
     pause 1
     rctime 15, 1, pedal
     pedal = pedal/150
     lookup pedal, [1000, 250, 250, 75], tempo

 IF in1=1 THEN main
 IF in0=1 THEN resetseedpaused

goto waitforstart
```

```
'=========================READ DATA FROM CHAOS SLIDER==========================
main:

     IF in1=0 THEN waitforstart

     n = (in8) + (in9 * 2) + (in10 * 4)           'read inputs 8,9,10 for chaos
     lookup n, [50000, 62259, 60000, 56000, 62000, 59000, 55000, 58000], chaos
     chaos = chaos - 32768                        'ABOVE ARE PRESET CHAOS VALUES

 IF in0=1 THEN resetseed

'------------CHECK IF SEED IS NEGATIVE, AND SQUARE THE SEED VALUE-----------------

continue:

    if(seed<32768) then negativeseed  'if seed is negative... (below 32768)

positiveseed:
     seed = seed - 32768       'make seed range 0 to 32768 (0 to 2, absolute value)
                               'however, the seed never exceedes the range of 0 to 1
                               'which is 1 to 16384 at this point
                               'the sign doesn't matter, since seed gets squared
     goto keepgoing

negativeseed:
     seed = 32768 - seed

keepgoing:
     hibits = (seed/128) * (seed/128)                        'high bits * high bits +
     medbits1 = ((seed*128)/256) * (seed/128)                'low  bits * high bits +
     medbits2 = (seed/128) * ((seed*128)/256)                'high bits * low  bits +
     seedsquared = hibits + (medbits1/256) + (medbits2/256)

'-----------------------MULTIPLY CHAOS * SEED SQUARED---------------------------

 hibits = (chaos/128) * (seedsquared/128)                'high bits * high bits +
 medbits1 = ((chaos*128)/256) * (seedsquared/128)        'low  bits * high bits +
 medbits2 = (chaos/128) * ((seedsquared*128)/256)        'high bits * low  bits +
 product = hibits + (medbits1/256) + (medbits2/256)

'---------------------- 1 - rx^2 ------ (rx^2 is "product") -----------------

 seed = 49152 - product         '1-rx^2
                                '49152 is 1 now instead of 16384 to make 0 = 32000
                                'so that negative numbers can be represented

'--------------------------------MIDI OUTPUT-------------------------------------

 noteout = seed / 744                          'put in range of 88 note keyboard

 serout 14, 12, [144, noteout, 120]            'play MIDI notes
 pause tempo
 serout 14, 12, [128, noteout, 0]
 pause 1

'debug ?tempo
```

```
goto main

'=======SET NEW SEED VALUE WITH BUTTON ON HORIZONTAL SLIDER DURING ITERATION=======

resetseed:
     m = (in11) + (in12 * 2) + (in13 * 4)     'read inputs 11,12,13 for seed value
     lookdown n, [0,1,3,2,6,4,5,7], mcode     '[000,001,011,010,110,100,101,111]


 seed = mcode * 4681 + 16384             'scale seed to -1 to 1 (or 16384 to 49152)

goto continue

'=======SET NEW SEED VALUE WITH BUTTON ON HORIZONTAL SLIDER WHILE PAUSED=========

resetseedpaused:
     m = (in11) + (in12 * 2) + (in13 * 4)     'read inputs 11,12,13 for seed value
     lookdown n, [0,1,3,2,6,4,5,7], mcode     '[000,001,011,010,110,100,101,111]

     seed = mcode * 4681 + 16384         'scale seed to -1 to 1 (or 16384 to 49152)

goto waitforstart
```

Why only Pitch and not Timbre, Rhythm, or Dynamics?

Why not use the dynamical systems discussed above to modify parameters other than pitch? Firstly, the author is personally more in tune (no pun intended) with pitch relationships than with timbre, while admitting that both are equally important in composition. Furthermore, it is much more apparent to hear what is going on with the mathematics when the pitch is modulated rather than timbre, dynamics, or rhythm. Each different function has its own distinct melodies and thematic motifs that reoccur, although there are seemingly endless variations of the melodies for each function. After improvising and composing with various functions, it actually becomes possible to recognize which function is being heard.

A benefit of *Chaos Melody Theory* is that practically anyone - non-musicians and any non-mathematicians alike - can produce beautiful and satisfying music using hardware and software that was created for this purpose. The performer is only required to vary a few parameters that control the phrasing, dynamics and the amount of chaos. People can explore their sense of musical dynamics and phrasing, even with no prior musical training, and they can empower themselves by controlling elegant mathematics, hearing instant feedback. Most importantly, it allows people with no formal musical training to get personally involved with music again, using technology and math, which many argue have taken away from musicality in the past.

What have other people done?

Joseph Schillinger was a pioneer in the area of chaos music whose ideas in the 1920's and 1930's about generating music by recursive and chaotic means were far ahead of their time. Many people throughout the late 80's and early 90's have experimented with iterative "chaos music", commonly using the logistic equation ***f(x)=rx(1-x).*** Martin Guertner, born 1946, has worked on fractal music since 1984. He Created the soundtrack for the *Scientific American videotape: Fractals: An animated discussion* (1991). In 1992 he founded the Fractal Music Lab Stuttgart. He's member of *Circulus Creativus*, a group of people who joined their forces to create and perform fractal music. He mainly uses the logistic equation, ***f(x) = rx(1-x).*** Gary Lee Nelson, has also used the logistic equation, ***f(x) = rx(1-x)***, as a source for his music. In his piece called *The Voyage of the Golah Iota*, he varied ***r*** between 1.0 and 4.0 over a nine minute period. The piece starts out with ***r*** = 1.0, then ***r*** rises to 4.0, then falls back again to 1.0. Values of ***x*** were iterated during this time. This piece of music would fall under the realm of *Chaos Melody Theory*, yet he is "bifurcating" during the entire piece, never letting a complete "chaos melody" run it's course.

Robert Greenhouse wrote a series of programs called *Music From the Fringe*. These were experimental programs designed to produce sounds from fractals. Another program, *The Well-Tempered Fractal*, contains ten built-in families of fractals for the user to choose from. *The Well-Tempered Fractal* is a program which composes musical melodies by mapping fractal images to musical scales. Ten different fractal types may be mapped to twenty-one different musical scales. The fractal algorithms and techniques used in this program are using different methods than discussed in this paper. Unlike *Chaos Melody Theory*, which in it's pure form, is meant to produce complete and seamless musical works, this program is only meant to produce fractally generated melodies to be used by composers as the starting point for larger musical compositions. Many others have used the MAX software to create chaotic recursive systems to produce chaos music, however, they too always seem to use the logistic equation, ***f(x) = rx(1-x),*** and never seem to go very deep into the dynamics of the function, nor to document the results in any formal manner. *Chaos Melody Theory* is an attempt to better define a very specific type of chaos music, and to define the commonly used terms.

There is a small but steady community which can be found easily on the internet, with an interest in fractal music and chaos music composition (often these terms are used interchangeably). Several websites have been well-developed that contain explanations, tutorials, music examples, fractal graphics generators, and even bulletin boards to further communication between fans of fractal and chaos music. *The Fractal Music Lab* contains music examples, downloadable software for composing original fractal music, and information and activities designed to help in understanding fractals and their possible roles in music composition. *The Fractal Music Project* website is a place where fractal music enthusiasts are encouraged to publish their works. A lot of chaos music and fractal music use different techniques, such as creating a musical piece derived from extracting data from pictures of fractals, using various fractal algorithms, and commonly the output is mapped to a parameters other than pitch, such as timbre or rhythm.

Future Implementation: Graphics, MIDI Controllers and Software

In the future, certain variations of graphics should be linked to the music. Having this visual input would help greatly in getting a sense of the dynamics of the mathematical system, and could be an exciting visual addition to a performance that would directly correlate with the music being performed. One way this could be implemented as a performance tool for simple melodies in the real domain, is to start with a visual representation of a bifurcation diagram corresponding to the function being iterated. The performer would click in or near a basin or other desirable area of the diagram and instantly create a melody from the corresponding ***x*** and ***r*** values. An ***x,y*** function diagram could instantly be projected, showing each iteration as perpendicular lines as in the function diagrams shown earlier in this paper. Perhaps sliding up and down the ***r*** axis would produce bifurcations that would be heard as melodic passages.

When working in the complex domain, Julia sets could be calculated with each change in the chaos values (the a and b values). It would be fantastic to have the Julia set morph as the chaos values changed, and this would take an immense amount of computing power. As the melodies play, the bouncing around of the iteration could be shown overlaid onto the Julia Set as in Figure 24. These graphics would be a nice visual addition to the music to stimulate the imagination. As complex melodies are experimented with in the future, desirable ranges for the parameters, and new functions with musically pleasing results should be logged in a formal manner, so that this accumulative data can be implemented in future *Chaos Melody Theory* performance and sequencing software. If a standard is set, new functions, ranges and values can be easily imported and automatically implemented and called upon in the software.

Many different variations of software and hardware MIDI controllers could be created to perform chaos music, and would only need, at the very least, control over ***x*** and ***r*** values. The Chaos Controller is just one example of the endless shapes and sizes that could work for this type of performance. Different ways of controlling parameters such as tempo and dynamics should still be explored, as the Chaos Controller is still rather limited in this area. Perhaps special graphics generating software could be written that would take data from the Chaos Controller or other performance device to create the aforementioned diagrams and Julia Sets in real time. Realistically, *Chaos Melody Theory* may not see it's time for years to come. However, if proper research and experimentation is done by the few and far between in the meantime, the tools to compose and perform this music will be well developed and matured if and when these ideas do finally catch on to the mainstream.

Conclusion

With *Chaos Melody Theory*, it is possible even for the non-musician to improvise musically, and to compose complete musical works that are dynamic and melodically pleasing. It is possible for the performer to become familiar with the dynamics of an equation, to the extent that one could become a virtuoso of this type of music. Molding and shaping phrases and orbits by delicately nudging parameters this way and that way, and creating explosive sweeps of chaos can be very empowering. There is an art to controlling mathematics in such a fluid manner, and in this case, controlling chaos. The performer must be equipped with the tools to perform music derived from *Chaos Melody Theory*, since it uses a recursive mathematical algorithm, and there are only a few hardware and software tools in existence developed for this purpose. There is a lot of room for exploration and development of new tools for the field of *Chaos Melody Theory,* and a lot of room for further research into the dynamics of different equations.

References:

Books:

Basic Stamp
Claus Kuhnel and Klaus Zahnert (Newnes 1997)

Interactive Music Systems: Machine Listening and Composing
Robert Rowe (MIT Press, 1993)

Composing Music with Computers
Eduardo Reck Miranda (Focal Press, 2001)

Chaos and Fractals: New Frontiers of Science
Peitgen, Jurgens, and Saupe (Springer-Verlag, 1992)

Chaos, Fractals, and Dynamics: Computer Experiments in Mathematics
Robert L. Devaney (Addison-Wesley, 1990)

Fractals for the Macintosh
Jesse Jones (Waite Group Press, 1993)

Fractal Creations (for the PC)
Timothy Wegner and Mark Peterson (Waite Group Press, 1991)

MacMath: A Dynamical Systems Software Package for the Macintosh
John H. Hubbard and Beverly H. West (Springer-Verlag, 1992)

Nonlinear Differential Equations and Dynamical Systems
Ferdinand Verhulst (Springer-Verlag, 1990)

Numerical Methods: Real-Time and Embedded Systems Programming
Don Morgan (M&T Books, 1992)

Programming and Customizing the Basic Stamp Computer
Scott Edwards (McGraw-Hill, 1998)

Websites:

The Fractal Music Project
http://www-ks.rus.uni-stuttgart.de/people/schulz/fmusic/

Fractal Music Lab
http://members.aol.com/strohbeen/fml.html

Fractal Explorer
http://www.geocities.com/CapeCanaveral/2854/

Well Tempered Fractal v3.0 by Robert Greenhouse
http://www-ks.rus.uni-stuttgart.de/people/schulz/fmusic/wtf/

Glossary

Bifurcating: a melodic passage that is created by raising at least one chaos parameter continually to cause the melodic pattern to bifurcate.

Chaos Controller™: a stand-alone hardware MIDI controlling device with limited control over parameters and variables in an iterated function; used for live performance of chaos melodies.

Chaos Melody™: the scaled output of an iterated function, either in the real or complex domain, provided the output does not escape to infinity or tend towards zero.

Chaos Melody Maker™: a software program with unlimited control over parameters and variables in an iterated function; capable of producing chaos melodies in the real or complex domains, and capable of sequencing changes in parameters and variables over time to be stored for later retrieval.

***Chaos Melody Theory*™:** the collection of documented research, definitions and thoughts, contained in this and other documents, pertaining to the production of organic melodic patterns for the purpose of music composition and performance based on controlling the dynamics of an iterated function in the real or complex domain.

Melody Set™: the set of all possible chaos melodies that result from all possible static combinations of variables and parameters for one particular function.

Thanks go to Dr. Richard Boulanger for being the first to inspire me to open my mind to microtonality, alternate forms of computer music composition, alternate controllers, to learn MAX and to explore fractal music. Many thanks also to Dr. Rowe for his guidance, many years later, with my MAX patch upgrades. Thanks to James Carpino for teaching me analog and digital electronics that changed my life, to Dr. Tom Igoe for teaching the Basic Stamp microcontroller in one of the most valuable classes I took during my stay at NYU. Thanks to John Klos for some very valuable help with C and Basic programming, endless electronics tips, and a lesson in binary floating point math. Many thanks to Dr. Edward Belbruno for expert teachings in chaos theory, his urgings to expand my ideas into the complex domain, and his special brand of inspiration. Thanks to Mark Torres for lending me some great math and celestial mechanics books that I will keep studying into the future and use to add one more very important control mechanism to *Chaos Melody Theory*, to blast through some of the chaos - velocity. Thanks to Mike Hudack for proofreading this paper and correcting grammar, all while actually comprehending the topic. Thanks to Dr. Peacock for his general support and advice on my thesis. Thanks to my roommate, bandmate, and friend, Liz Lysinger, for her many cups of strong coffee. Thanks to my mother, Carol Walker, for her never-ending help with mathematics, and the inspiration that makes my never-ending need for mathematics as a computer musician less painful by factors of 10. Thanks again to her and to my father, Elbert Walker, for putting me through school once again.