Spring 5-2016

Applications of the Sierpiński Triangle to Musical Composition

Samuel C. Dent
University of Southern Mississippi

The University of Southern Mississippi

Applications of the Sierpiński Triangle to Musical Composition

by

Samuel Dent

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science
in the Department of Mathematics

May 2016

Approved by

Jiu Ding, Ph.D., Thesis Adviser
Professor of Mathematics

Bernd Schroeder, Ph.D., Chair
Department of Mathematics

Ellen Weinauer, Ph.D., Dean
Honors College

Abstract

The present paper builds on the idea of composing music via fractals, specifically the Sierpiński Triangle and the Sierpiński Pedal Triangle. The resulting methods are intended to produce not just a series of random notes, but a series that we think pleases the ear. One method utilizes the iterative process of generating the Sierpiński Triangle and Sierpiński Pedal Triangle via matrix operations by applying this process to a geometric configuration of note names. This technique designs the largest components of the musical work first, then creates subsequent layers where each layer adds more detail.

Key Words: Sierpiński Triangle, Sierpiński Gasket, pedal triangle, music composition, matrix

Acknowledgements

I thank my adviser, Dr. Jiu Ding, for his patience and guidance while helping me prepare this thesis. I would like to thank Dr. Danny Beard for my music theory education and for his helpful critique. I would also like to thank Dr. John Perry for his aid in devising the implementation of the computer code in SAGE. Finally, I thank Dr. Richard Perry for his support and critique of the compositions.

Table of Contents

Figures

Algorithm Pseudocode

Definitions

Abbreviations

Control Unit - CU

Iterated Function System – IFS

Input Unit - IU

Major Key – maj.

Minor Key – min.

Output Unit - OU

Sierpiński Pedal Triangle – SPT

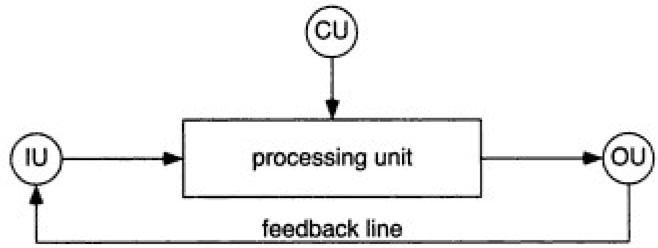
Chapter 1: Introduction

Description of the Problem

This project intends to explore one of the relationships between mathematics and music, specifically through the fractals known as the Sierpiński Triangle and the Sierpiński Pedal Triangle. A **fractal** is a geometric construct that is self-similar throughout its structure. Fractals can appear complex, but they are often generated by the simple process of iterating a pattern, where each iteration reduces the size of the pattern. Music, in essence, is an organized collection of sounds and silences. Composers frequently write music based on a pattern or motif of some kind, resulting in a self-similarity where a musical work can be broken down into sections, then into phrases, and so on. Each level of the music often reflects the nature of the work in its entirety. The self-similarity and patterns are evident in both fractals and music composition, indicating that there is a possible relationship between them. It is this relationship that this project means to explore. The music produced should be more than a series of random notes, but a series that pleases the ear. Additionally, this project seeks to generalize methods of producing the Sierpiński Triangle to the Sierpiński Pedal Triangle.

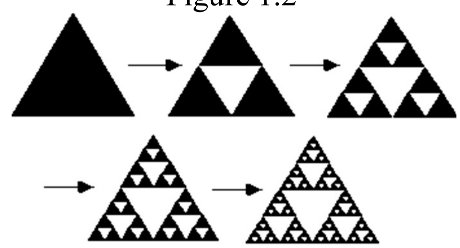
Relevant Background

Figure 1.1



Process of feedback
IU = input unit **CU** = control unit **OU** = output unit
Peitgen, H. et al., 2004 [6]

Figure 1.2



Sierpiński Process
Devaney, R. L., 1995 [2]

Fractals largely depend on the principle of **feedback**: an operation that is repeated many times with the output of one iteration becoming the input for the next iteration. This process is shown in Figure 1.1, where the control unit is a set of parameters to produce a desired result. The **Sierpiński Triangle** is a particular fractal produced through the feedback process shown in Figure 1.2. For the classic Sierpiński Triangle, the process begins with an equilateral triangle. Form an interior triangle by connecting the midpoints of the sides of the original triangle, and remove this interior triangle to leave three congruent equilateral triangles similar to the original. Take the output as the input and repeat this process indefinitely to produce the Sierpiński Triangle. Beginning with a different type of triangle also generates a Sierpiński Triangle where the n^{th} iteration produces 3^n congruent triangles that are similar to the initial triangle. As a result of the process removing a triangle equal to a fourth the area of the previous triangle, the area of the construct on the n^{th} iteration is $(0.75)^n$. So, the Sierpiński Triangle itself has an area of 0 because

$$\lim_{n \to \infty} 0.75^n = 0$$

yet the Sierpiński Triangle can be seen by the human eye [5].

One way to produce fractals such as the Sierpiński Triangle is with an iterated function system, abbreviated IFS. Before discussing the IFS, further background information must be made clear. Lasota and colleagues define a sequence of random variables $\xi_1, \xi_2, \dots, \xi_n$ to be **independent** if, for a sequence of Borel sets $B_1, B_2, \dots, B_n$, the events $\{\xi_i \in B_i\}$ are independent for all i. That is,

$$\begin{aligned} &\text{prob}\{\xi_1 \in B_1, \xi_2 \in B_2, \dots, \xi_n \in B_n\} \\ &\qquad = \text{prob}\{\xi_1 \in B_1\} \times \text{prob}\{\xi_2 \in B_2\} \times \dots \times \text{prob}\{\xi_n \in B_n\} \end{aligned}$$

A family of transformations $S_t: X \to X$, for $t \in \mathbb{R}$ on a set X is a **dynamical system** $\{S_t\}_{t \in \mathbb{R}}$ if it satisfies the following properties [5]:

1. $S_0(x) = x, \forall x \in X$.
2. $S_t(S_{t'}(x)) = S_{t+t'}(x), \forall x \in X$ with $t, t' \in \mathbb{R}$.
3. The mapping $(t, x) \to S_t(x)$ from $\mathbb{R} \times X$ into X is continuous.

Now let X be a closed, non-empty subset of $\mathbb{R}^d$. Consider N continuous transformations

$$S_i: X \to X \text{ for } i = 1, \dots, N,$$

the probabilistic vector

$$(p_1, \dots, p_N), \text{ where } p_i \geq 0 \text{ and } \sum_{i=1}^{N} p_i = 1,$$

and the sequence of independent random variables $\xi_1, \xi_2, \ldots$ such that

$$\mathrm{prob}\{\xi_n = i\} = p_i, \text{ for } i = 1, \ldots, N.$$

The dynamical system defined by the formula

$$x_{n+1} = S_{\xi_n}(x_n) \text{ for } n = 0,1, \ldots$$

is called an **iterated function system** [5]. For $A \subset X$, we define

$$F(A) = \bigcup_{i=1}^{N} S_i(A) \qquad \text{and} \qquad A_n = F^n(A_0)$$

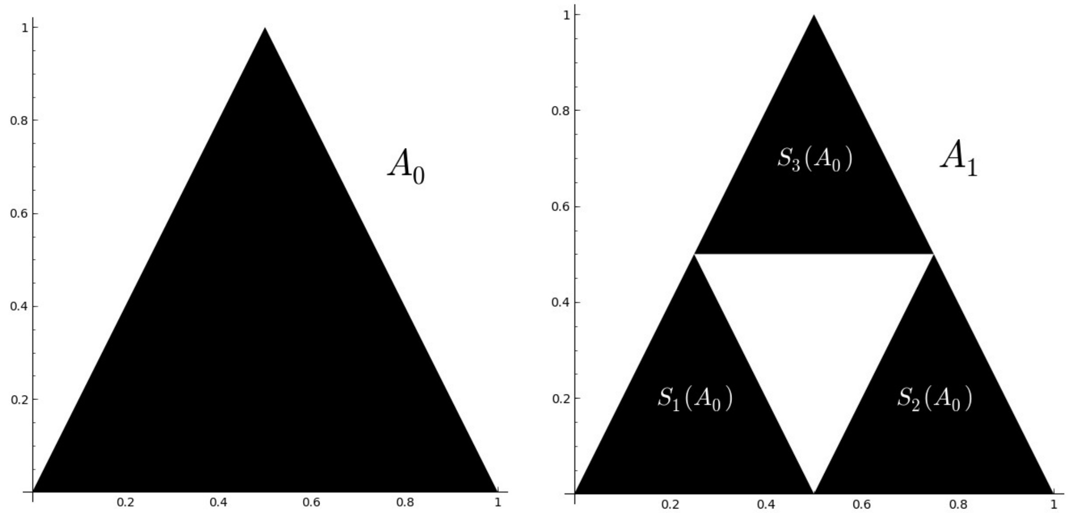
where $A_0 = A$. We also define the **limiting set** $A_* = \lim_{n\to\infty} F^n(A_0)$ [5].

The following is an IFS given by Lasota and colleagues that generates the Sierpiński Triangle: let $X = \mathbb{R}^2$ and

$$S_i(x) = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} x + \begin{pmatrix} a_i \\ b_i \end{pmatrix}, \text{ for } i = 1,2,3 \text{ where}$$

$$a_1 = b_1 = 0;\ a_2 = \frac{1}{2}, b_2 = 0;\ a_3 = \frac{1}{4}, b_3 = \frac{1}{2}$$

Figure 1.3



First iteration of the Sierpiński Triangle IFS

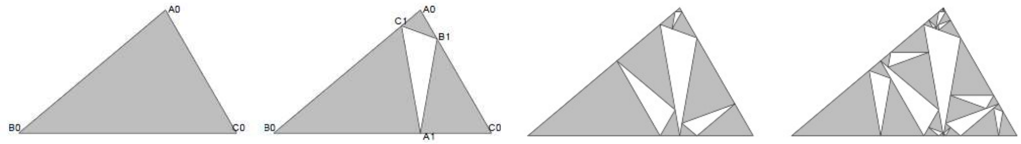
Lasota, A. & Mackey, M. C., 1994 [5]

We choose A_0 to be the isosceles triangle with vertices $(0,0), (1,0), \left(\frac{1}{2}, 1\right)$. Because this is a triangle and the transformations produce a similar triangle, each transformation can be calculated on the vertices to find the transformed triangle's vertices, rather than performing the transformations on every point in the triangle. Then $S_1(A_0)$ is a triangle with vertices $(0,0), \left(\frac{1}{2}, 0\right), \left(\frac{1}{4}, \frac{1}{2}\right)$. $S_2(A_0)$ and $S_3(A_0)$ are congruent to $S_1(A_0)$ but translated to the right, and up and to the right, respectively. Then the output is

$$A_1 = F(A_0) = S_1(A_0) \cup S_2(A_0) \cup S_3(A_0).$$

This first iteration is shown in Figure 1.3. The limiting set $A_* = \lim_{n\to\infty} F^n(A_0)$ is the Sierpiński Triangle.

Figure 1.4



(80º-40º-60º) - triangle 1^{st} iteration 2^{nd} iteration 3^{rd} iteration

Sierpiński Pedal Process

Ding, J. et al. [3]

Related to the Sierpiński Triangle is the **Sierpiński Pedal Triangle (SPT)**, which utilizes the pedal triangle rather than the triangle connecting the midpoints. For an initial triangle T_0, the **pedal triangle** is the triangle formed by connecting the three feet of the altitudes of T_0. If T_0 is a right triangle, then the pedal triangle is a straight line, and if T_0 is an obtuse triangle, the pedal triangle reaches outside T_0. For an acute T_0, the pedal triangle remains inside T_0 and can, therefore, be removed in the Sierpiński process as shown in Figure 1.4. Indefinite iteration results in the SPT. The SPT also produces a total of 3^n triangles similar to the original on the n^{th} iteration, but they are scaled, rotated, and reflected individually [3]. This key difference leads us to believe that the SPT may allow more flexibility concerning applications to music.

Fractal music is the application of methods used to generate fractals in the field of music [4]. An example of this crossover from fractals to music is the L-System defined by Hazard and colleagues. An **L-System** is a repetitive process that transforms a short string or axiom into a longer, more complicated string through a set of production rules. Each symbol in the string has a respective production rule, and with each iteration, the symbols are replaced by their production rule. Figure 1.5 exhibits an example of an L-

Figure 1.5

Axiom: AB

Production Rules:

- A → ABC
- B → CAD
- C → DC
- D → BDB

Axiom: AB

A B
1st iteration: (ABC)(CAD)

A B C C A D
2nd iteration: (ABC)(CAD)(DC)(DC)(ABC)(BDB)

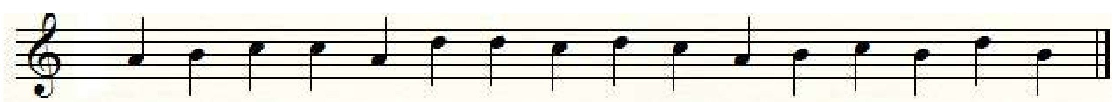
Etc.

Hazard, C. et al., 1999 [4]

System. The resulting string is intended to mimic the self-similarity of fractals. The string can then be interpreted musically as a string of notes, chords, or other objects [4].

A direct approach is to assign each symbol of the string to its corresponding note name, i.e. "A" → A, "B" → B, and so on, and a symbol such as "R" for a rest. The second iteration string from Figure 1.5 would then become the note string in Figure 1.6. The axiom and production rules must be chosen carefully to produce the desired music and effect [4].

Figure 1.6



String of notes from Figure 1.5
Hazard, C. et al., 1999 [4]

With respect to chords, the production rules can replace each symbol (a Roman numeral representing a triad based on that number's location in a scale) with a short chord progression to create a larger progression. Hazard and colleagues provide the example in Figure 1.7. For maximum effect, the axiom and production rules should be strongly influenced by the guidelines delineated by music theory. Hazard and colleagues do note that this L-System is not particularly effective on its own and should be combined with other methods. For instance, the resulting chordal string can be used as background chords for a melody or to constrain the melody to sound more like traditional Western music [4].

Figure 1.7
An L-System for a chord progression

Axiom: I

Production Rules:

- I → I IV V I
- ii → ii V I V
- IV → IV V I ii
- V → V I ii V

Axiom: I

I
1st iteration: (I IV V I)

I IV V I
2nd iteration: (I IV V I)(IV V I ii)(V I ii V)(I IV V I)

Etc.

Hazard, C. et al., 1999 [4]

Chapter 2: Matrix IFS

Method

First, analyze the matrix IFS for the Sierpiński Triangle and generalize it to the SPT. Then, create an algorithm or algorithms to carry out the matrix IFS on an arbitrary acute triangle to generate a given number of iterations in the SPT process. Graph several examples for various choices in initial vertices and iteration levels.

Results

In order to generate the SPT for an arbitrary acute triangle, I broke the task into four algorithms: the first reorganizes the vertices' coordinates, the second performs the first iteration of the Sierpiński Pedal Process, the third performs the Sierpiński Pedal Process a given number of times, and the fourth graphs the appropriate triangles. A description of each is given below.

Algorithm 1

algorithm *redo_points*
  inputs
    $a_1, a_2, b_1, b_2, c_1, c_2$, the respective *x*- and *y*-components of the vertices of an acute triangle
  outputs
    A, B, C, the vertices in the desired order
  do
    let A be the vertex with the lowest *y*-value.
      if more than one vertex shares this *y*-value,
        then let A be the vertex that also has the lowest *x*-value
    of the remaining points, let B be the vertex with the highest *x*-value
      if both vertices have this *x*-value,
        then let B be the vertex that also has the lowest *y*-value
    let C be the remaining vertex
    return A, B, C

Algorithm 1 reorganizes the points so that the other algorithms can work with the vertices of the triangle regardless of their input order. The particular choice of order of output from Algorithm 1 is explained under Algorithm 2. For the purpose of using matrix operations, the implementation of the algorithm in computer code formats each vertex in the output as a 2×1 matrix.

Algorithm 2

algorithm *Pedal_Triangle_Matrix_3*
  inputs
    $a_1, a_2, b_1, b_2, c_1, c_2$, the respective *x*- and *y*-components of the vertices A, B, C of an acute triangle
  outputs
    $A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$, the vertices of the triangles produced in one iteration of the Sierpiński Pedal Process
  do
    use *Algorithm 1* to reorganize the points
    for $X \in \{A, B, C\}$,
      let $X_0 = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \left[X - \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \right]$ with θ such that side $\overline{A_0 B_0}$ will be horizontal
    for $X \in \{A_0, B_0, C_0\}$,
      for $i \in \{1,2,3\}$,
        Dilate: let $X' = \begin{pmatrix} D_i & 0 \\ 0 & D_i \end{pmatrix} X$

        Reflect: let $X'' = \frac{1}{1+m^2} \begin{pmatrix} 1-m^2 & 2m \\ 2m & m^2-1 \end{pmatrix} X'$

        Rotate: let $X''' = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} X''$, where θ_i is the angle to rotate the triangle to the correct position

        Translate: let $X'''' = X''' + \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, where x_i, y_i translate the triangle to the proper vertex on the initial triangle

        let $X_i = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} X'''' + \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$
    return $A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$

Algorithm 2 uses Algorithm 1 to reorder the points, then subtracts the coordinates of the new A from all of the vertices and rotates the triangle so that side $\overline{AB}$ is horizontal.

Due to the order of the order of the vertices in the output form Algorithm 1, the triangle should now be in the first quadrant and an acute angle θ is relatively easy to find by

$$\theta = \tan^{-1} - \frac{b_2 - a_2}{b_1 - a_1}.$$

With $\overline{AB}$ horizontal, the vertices of the "bottom left" triangle in the Sierpiński Pedal Process do not need to be rotated, and the rotations for the other vertices are easier to visualize and to determine. The remaining steps are repeated three times to make the three dilated triangles from the Sierpiński Pedal Process. The algorithm dilates the triangle by multiplying each vertex by a dilation matrix with a dilation factor D_i. D_i is the ratio of a side of the resulting to the side of an initial triangle which, according to Ding and colleagues can be found by

$$D_i = \cos(angle)$$

where *angle* is the measure of the angle on the initial triangle (A, B, C) to which the dilated triangle will eventually be translated. The algorithm reflects the points about the bisector of angle A where the slope of the bisector is m. To find m, use the fact that $\tan\theta$ is the slope of a line that is θ degrees/radians counterclockwise from the positive *x*-axis. By the tangent difference identity

$$\tan(\alpha - \beta) = \frac{\tan\alpha - \tan\beta}{1 + \tan\alpha\tan\beta}$$

the tangent of an angle between two lines with slopes m_1 and m_2, respectively, is equal to

$$\frac{m_1 - m_2}{1 + m_1 m_2}.$$

Thus, the bisector will have a slope m that satisfies

$$\frac{m - k_1}{1 + mk_1} = \frac{k_3 - m}{1 + k_3 m}$$

where k_1, k_3 are the slopes of the intersecting lines or, in this case, the sides of the triangle. Because the algorithm makes $\overline{AB}$ horizontal, $k_1 = 0$, and the equation becomes

$$m = \frac{k_3 - m}{1 + k_3 m}.$$

According to Andrilli and Hecker, the vertices can then be reflected by the matrix operation shown in Algorithm 2 because the sides of the triangle are part of lines that pass through the origin. Essentially, this operation makes the two sides of the triangle swap locations. The algorithm then rotates the triangle to match the angle to which it will be translated. The "bottom left" triangle does not need to be rotated as stated earlier. The "top" triangle must be rotated $\pi - B$ radians and the "bottom right" triangle must be rotated $C - \pi$ radians. The algorithm then subtracts the coordinates of the vertex (of the dilated triangle) which will be matched to a vertex on the original triangle and adds the coordinates of the proper vertex on the initial triangle. In the algorithm, this is shown as a net matrix addition of $\begin{pmatrix} x_i \\ y_i \end{pmatrix}$. At this point, the angle associated with the vertex of the dilated triangle that was just translated should equal the angle associated with the vertex to which the triangle was moved. The sides of the dilated triangle should also line up with the sides of the initial triangle. At the very end, the list of coordinates is returned as the nine vertices of the dilated triangles in matrix form.

Algorithm 3

algorithm *SPT_Matrix*
 inputs
  $a_1, a_2, b_1, b_2, c_1, c_2$, the respective *x*- and *y*-components of the vertices A, B, C of an acute triangle
  n, the number of iterations
 outputs
  coordinates for the triangles in the next iteration of the Sierpiński Pedal Process
 do
  use *Algorithm 2* to find $A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$
  let T be the list of these vertices and assign them level n.
   if $n - 1 > 0$,
    use *Algorithm 3* on A_1, A_2, A_3 with iteration number $n - 1$ and concatenate the list to T
    use *Algorithm 3* on B_1, B_2, B_3 with iteration number $n - 1$ and concatenate the list to T
    use *Algorithm 3* on C_1, C_2, C_3 with iteration number $n - 1$ and concatenate the list to T
  return T

Algorithm 3 enforces n iterations of the Sierpiński Pedal Process. It uses Algorithm 2 to produce one iteration, then calls upon itself to perform the Sierpiński Pedal Process on the generated triangles while decrementing the number of iterations. The decrementation ensures that the process will terminate and not perform an infinite loop. The algorithm returns a list of lists of points and iteration levels.

Algorithm 4

algorithm *SPT_Graph*
 inputs
  $a_1, a_2, b_1, b_2, c_1, c_2$, the respective *x*- and *y*-components of the vertices A, B, C of an acute triangle
  n, the number of iterations
 outputs
  a graph of the Sierpiński Pedal Triangle with n iterations
 do
  use Algorithm 3 to produce a list T of lists
  let SPT be an empty graph
  for every list of coordinates $P \in T$,
   if the level of iteration (the last entry in P) is 1,
    then add the triangle with those vertices to the graph SPT
  return SPT

Algorithm 4 sifts through the list of lists produced from Algorithm 3 and produces a graph that only includes the triangles with iteration level 1. These triangles should be the smallest triangles produced from the Sierpiński Pedal Process so no triangles should overlap or cover each other on the graph.

Discussion

Algorithm 2 is named *Pedal_Triangle_Matrix_3* because I created two previous versions. The first version required the computer to compute exact values for the vertices. With multiple instances of trigonometric functions and the introduction of π, the computer attempted to keep track of all of the values exactly and the program was very slow. In combination with Algorithm 3, the slow speed compounded over many uses of Algorithm 2, and the graphs took many hours, especially for more than three iterations. The second version rounded results to several decimal places, losing a small amount of accuracy but exponentially increasing the speed of the program. However, this version retained some redundancies in calculation, and so this third version replaced some of the calculations using simpler equations from Ding and colleagues' work. The entire program is relatively quick and is reasonably accurate for small numbers of iterations. The order of input for the vertices does not affect the process, as desired.

If the given triangle is acute, no operations involve division by zero or other inconsistencies, and the algorithms correctly execute and produce an appropriate graph. However, there are no safeguards to ensure that the triangle is acute. So, if the entered vertices form a right triangle or an obtuse triangle, the program does not properly execute. A graph may be produced, but the triangles do not form a picture with any significance or meaning. Additionally, it is recommended that the vertices' coordinates are entered with decimal points, even if the numbers do not require them, to safeguard against the program trying to use exact numbers. Although the program is relatively quick, the IFS is generally more complicated than simply calculating the vertices of the pedal triangle and removing the pedal triangle.

The computer code used in SAGE for these algorithms can be found in the Appendix.

Chapter 3: Music Composition

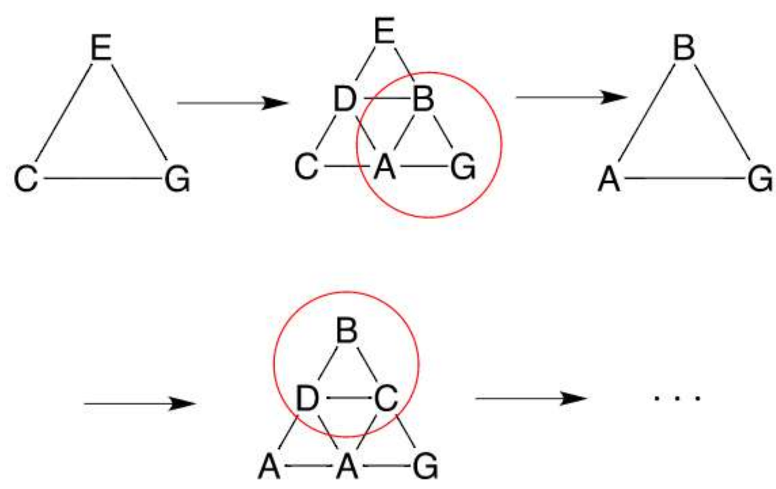
Methods

Analyze the matrix IFS of the Sierpiński Triangle and SPT for applications to music composition. Also analyze the properties, such as the relevant isometries, of the Sierpiński Triangle and SPT themselves.

Results

Initially, I created a process of composition that combined the L-System with the Sierpiński Triangle, shown in Figure 3.1. The idea was to arrange three notes into the configuration of a triangle, then create an inner triangle like in the original Sierpiński Process. However, I had difficulty creating rules for choosing notes in this method. The most promising version started with the three notes of a tonic chord in a given key. Then

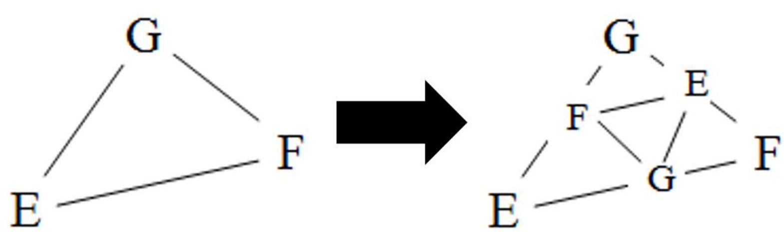
Figure 3.1



Sierpiński L-System on a C maj. tonic chord

the inner notes were chosen as notes that could "appropriately" connect the vertices on that side. A random vertex was chosen and the process was performed again on that triangle. However, this did not prove very effective. So I turned to the Sierpiński Pedal Triangle. Although the matrix IFS is more complicated than other methods, it did emphasize how the initial triangle is related to the dilated triangles through reflections and rotations. To incorporate this idea and mimic the SPT itself, I modified the new Sierpiński L-System by beginning with three chosen notes, then making an interior pedal triangle. This meant the foot of altitude from each vertex would create the same vertex note name, as shown in Figure 3.2. Iterating the process on the three dilated triangles produces more reflections and rotations. I decided to use the three starting notes as a three note pattern, read left-to right as E-G-F. Then, after performing the Sierpiński Pedal Process, any other permutation of the three note pattern present in the L-System

Figure 3.2



Sierpiński Pedal L-System on E-G-F

could also be used. In fact, after two iterations, all possible permutations of notes are available. Additionally, sharps or flats can be added to the pattern, depending on the key of the music and the intended effect. For the actual composition of music, the idea is to spread the permutations of the three-note pattern throughout a piece at various levels. "Various levels" refers to having the pattern occur in different parts at different times, across multiple parts at the same time, as part of the chord structure, and other techniques. The intent is to mimic the self-similarity of the SPT. Then between these patterns, compose the rest of the music using musicality and the rules of music theory.

The musicality gives the composer freedom to produce music with desired effects, feelings, and sounds, while the music theory gives some constraints on this freedom. However, for my purposes, I allowed for some instances of musicality to supersede music theory if there was a discrepancy. The impetus behind this decision is that at least one work produced from this project is to be performed by my tuba quartet *Sort of Voce*. As such, the music needed to be interesting to play and hear in addition to interesting to compose and analyze. Furthermore, this situation dictated that at least one work be in four parts, for two tubas and two euphoniums, all of which are in bass clef. The musical work located in the Appendix, titled *Sierpiński Pedal Quartet* is the result of this method under these conditions.

Discussion

The *Sierpiński Pedal Quartet* is divided into three movements and is based on the E-G-F pattern. The first movement, titled *Energy*, is in the key of E^b maj. It utilizes the permutations of the three-note pattern to create quick, moving lines that are full of vigor. The focus on the movement conveys an energetic feel, rather than a focus on chord structure to produce harmonious progressions. As a result, many of the phrases primarily use only three chords, such as I (E^b maj.), IV (A^b maj.), and V (B^b maj.). However, the vast majority of the music is euphonious, rather than many clashing lines of notes.

Energy begins with a Euphonium 2 solo explicitly using the original E-G-F pattern, and in measures 5-16, the other instrument parts are added in layers. Measures 17-28 introduce the main theme of the piece in the Euphonium 1 part. Euphonium 2 and Tuba 1 utilize straight eighth notes to produce the feeling of movement throughout the phrase with a short break from eighth notes in measure 24 so that the players can breathe. Tuba

2 uses a slightly syncopated rhythm that is popular in pop music to generate an even greater feeling of motion. The phrase is essentially repeated in measures 29-40 but with a countermelody in Tuba 1. At measure 41, the movement enters into a bridge-like section which augments the instrument parts from measures 1-16 and adds a more rhythmic aspect in Euphonium 1. Beginning at measure 49, Euphonium 1, Euphonium 2, and Tuba 2 essentially perform the same overall division of eighth notes (dotted quarter note + dotted quarter note + quarter note) every other measure, but in each division, Euphonium 1 or Euphonium 2 (or both) play a grouping of eighth notes, creating a cascade effect. Tuba 1 plays straight eighth notes similar to measures 17-28 while the other parts use the division described previously. Then when the other parts have a whole note, Tuba 1 interjects with the E-G-F pattern or a similar pattern, continuing the motion. Measure 57 reverts back to the last four measures of the theme (e.g. measures 25-28) but with a Euphonium 1 harmony line. All parts are essentially in unison rhythm in measures 61-65, accenting the E-G-F pattern with harmonies. Measures 66-77 are a restatement of the theme from measure 17-28. Measures 78-89 also use the theme. However, the tuba parts wait a measure to come in with an arpeggio-like figure and also introduce different chords than the other theme phrases. Then the tubas, in unison rhythm, accompany the Euphonium 1 part beginning in measure 82. During all of this, Euphonium 2 performs the E-G-F pattern and rhythm from the beginning as straight eighth notes, again keeping the momentum of the piece going. At measure 86, Tuba 1 takes over this line while Euphonium 2 plays a harmony line in unison rhythm. Tuba 2 plays a harmony line to the theme in Euphonium 1. Then measure 90 begins the closing section. The euphoniums trade sixteenth note runs, maintaining the energy of the piece. The tuba parts begin with

four quarter notes, then Tuba 1 switches to a three-eighth-note-pattern in the next measure, and Tuba 2 changes to a two-eighth-note-pattern, intended to make the ending feel like it is speeding up without actually changing tempo. All parts play an eighth note run in octaves in measure 93. In the next measure, Euphonium 1 and Tuba 1 progress to the next note in the run (B^b) while Euphonium 2 and Tuba 2 delay this note with one E-G-F pattern from the introduction. Then all parts end on a tonic E^b major chord on beat four of the last measure.

The second movement is titled *Grief* and exhibits vignettes of some stages of grief a person might experience after a traumatic event. It is broken into three sections, mimicking the overall structure of the whole work. The first section represents the melancholy stage of grief, and so it is in the key of E min. It contains gloomy, flowing lines, reflecting the person's dejected spirit while still maintaining an appearance of control. The middle section, still in E min, features much more dissonant harmonies, polyrhythm, and meter changes to convey a growing anger and intensity. This section represents anger building in the victim as they fight the reality of the traumatic experience until he/she finally loses control and explodes. The final section represents the person finally facing their internal struggle with the traumatic experience, eventually coming to an acceptance of reality and triumphing over grief. The section begins with quiet solo melodic lines that harken back to the melancholy attitude of the first section. Then it transitions into the key of G maj. and builds back intensity, not of anger, but of hope. After reaching the climax of this accumulation of intensity, the piece gradually softens and ends in a sweet whisper.

In measures 1-8 of *Grief*, the instrument parts have staggered entrances that result in dissonance between the three notes E, $F^{\#}$, and G. The staggered entrances convey the feeling of loneliness of the victim while the dissonance represents the pain of grief. At the end of each short phrase, the dissonance partially resolves, representing the juxtaposition of the victim's collected appearance with his/her interior aching. Beginning in measure 9, the tuba parts form the accompaniment for the forthcoming euphonium melodic material. They maintain the pattern of Tuba 2 downbeat with Tuba 1 eighth notes on beats two and three until measure 30 to keep a moving, flowing feeling that is slower and more lyrical than in *Energy*. Euphonium 2 begins the melodic material in measure 11, and Euphonium 1 takes the mantle in measure 17 while Euphonium 2 provides a harmony. Between each phrase of melody is a measure of rest in the euphonium parts intended to maintain the isolated feeling introduced in the first measures. Permutations of the E-G-$F^{\#}$ pattern can be readily identified in the melodic material. Tuba 1 picks up the melody in measure 30 while the other parts provide accompaniment. The lugubrious feel is carried all the way to measure 45, where a more intense dissonance emerges, representing the first streak of anger in the victim.

The middle section begins with the euphonium parts continuing the previous dissonance in measures 46-47 while Tuba 1 plays an E-G-$F^{\#}$ pattern in measure 47. Beginning in measure 48, the euphonium parts clash with tritone dissonance and polyrhythm where Euphonium 1 focuses on the dissonant notes and Euphonium 2 plays a pattern on E-G-$F^{\#}$. The tuba parts play a sinister melody in octaves (and briefly in fifths in measure 53) allowing the lower voicing to dominate this section and giving it a darker, angrier tone. In measure 56-57, the tuba parts create a sense of unease by playing the first note of groupings of three sixteenth notes, rather than on the downbeats of the meter. Meanwhile, the euphonium parts trade sixteenth note runs that are mostly based on thirds and that somewhat matches the rhythm of the tubas. That is, whenever a euphonium starts a sixteenth note run, the tubas also play on the euphonium's starting note. Then in measure 57, the euphoniums play the sixteenth note run simultaneously a second (interval) apart from each other, adding more cacophony to the uneasiness of the measures. Measures 58-59 change the meter to further destabilize the meter, and the euphonium parts continue to clash with one another. Tuba 1 plays a monothematic line in measure 60 which can be broken down into four groupings of three sixteenth notes and a quarter note. The first grouping starts on $F^{\#}$, uses thirds travel up to the beginning E^{b} of the next grouping, which uses a second and thirds to reach a G. Once this grouping ends, the final grouping displays a G-F-E pattern. These groupings reflect the three-note pattern with the starting notes, use of thirds, and the direct use of the final G-F-E pattern. The quarter note at the end of the measure with all parts playing functions to inject intense harmonic dissonance back into the measure. Measures 61-62 function the same as measures 58-59. Then, in measure 63, the euphoniums return with the sixteenth note runs from previous measures while Tuba 1 plays a pattern on E-G-$F^{\#}$. By measure 64, anger has consumed the victim, represented in the driving E-G-$F^{\#}$ pattern in Tuba 2. In measures 64-87, Tuba 2 maintains the same driving pattern while the Tuba 1 part becomes more detailed with shorter note values to continue increasing the momentum of the piece. During these phrases, the euphoniums play melodic material that becomes more dissonant as time progresses. The final build to the climax occurs in measures 88-91, where the euphoniums accent every fourth beat while the tubas, staggered two beats from the euphoniums, also accent every fourth beat. Tuba 1 also plays an E-G-F$^{\#}$ eighth note pattern while Tuba 2 rests to maintain motion. The climax of anger is reached in measures 92-97, with a final dissonant chord in measure 95 pushing into measure 96. Here Tuba 2 gives a final, explosive pedal E to symbolize the last outburst of anger in the victim, followed by a decay in volume to represent the anger fading away from the victim. A quick moment of silence between measures 97 and 98 returns the feeling of isolation and reflection in the victim. Then Euphonium 2 provides a solo melodic line similar to the melancholy material from measures 11-16, bringing back the lugubrious tone. This is followed by another short pause and a similar line in Tuba 1 that references material from measures 30-45. Another short pause, and Euphonium 1 introduces new material that hints at the key change to G maj. that shortly follows.

Measure 111 marks the beginning of the victim's progress of overcoming his/her grief. Euphonium 1 carries the lyrical melody while the tuba parts focus on using chord changes to convey the feeling of change and hope. Euphonium 2 joins in on a harmony line in measure 115, and in measure 118, Euphonium 2 introduces an E-F$^{\#}$-G motif that will prevail the remaining portion of the piece. This motif uses the same notes that conveyed pain and anger in the first two sections of *Grief*, but now it intends to inspire positive feelings. This represents the internal change within the victim as he/she transitions from sadness and ire to a more positive outlook. In measures 119-130, the euphonium parts switch roles approximately every four measures, one playing the melody and the other playing the motif, so that the players may catch their breath. Measures 119-122 rebuild the intensity from the middle angry section, but this time the intensity is not found in anger, but in hope and strength. In measures 123-130, the victim has overcome his/her grief and finds the courage to carry on. The euphoniums play the motif and melody while the tubas provide accompaniment. Tuba 1 joins the melodic material as a harmony line beginning in measure 126. In measures 131-136, the instrument parts retain some dissonance, but now there is more beauty than pain in these sonorities. This represents the victim looking back on the traumatic experience and his/her recent grief with the knowledge that he/she has triumphed over the situation, becoming stronger for it. During this phrase, the Euphonium 1 continues playing the melody from the preceding measures while Euphonium 2 provides a countermelody every other measure. A decay in volume in measure 136 brings all instrument parts to tender whisper for measure 137 to the end, which represents the victim reaching the end of the stages of grief.

The final movement of the work, in F min., is titled *Funk-ish*. It incorporates blues scales and syncopated rhythms to create a funk-like groove throughout. In general, the tubas lay down a bass line with some embellishment while the euphoniums provide the melody and harmony. For some sections, however, no particular part has a “melody,” but instead, the ensemble grooves together as one single unit. That is, every part is of equal “importance” melodically and harmonically. The focus of this movement is to create music where the audience feels the urge to tap their feet or bob their heads to the music.

The first four measures of the movement are introductory material. Measures 1 and 3 have an F in octaves for the first two eighth notes, while measures 2 and 4 have the F-G^b-E^b pattern as the notes distributed across all four parts for the first two eight notes. Tuba 1 plays for the remaining beats in the first two measures, introducing the rhythm and theme in measure 2 that will occur at the end of every main phrase for the rest of the movement. Euphonium 1 solos in measure 3 based on the G^b-F-E^b pattern, and in measure 4, Euphonium 2 and Tuba 2 carry the ensemble into measure 5. In measures 5-8, Tuba 2 provides a bass line partially based on the F-G^b-E^b pattern while Tuba 1 focuses on the upbeats. Measure 9 begins the first non-introductory section. For measures 9-12, Tuba 2 continues with a similar bass line as before while Euphonium 2 plays a rhythmic melody. Euphonium 1 and Tuba 1enter in layers for the last two measures. In measures 13-16, Tuba 2 focuses on playing a C to give the feeling of being on a dominant, rather than tonic, chord. Euphonium 1 takes the melody, and Tuba 1 plays upbeats. Euphonium 2 joins Euphonium 2 as harmony in measure 15, and in measure 16, the Euphoniums play two sixteenth notes on alternating eighth note beats (Euph. 1 on the first eight note, Euph. 2 on the second eighth note, etc.) so that sixteenth notes are always being played. This builds the intensity going into measure 17 where the upper three instruments play a staggered F-B^b-C^b-E^b theme that reoccurs throughout the piece. In measure 18, the upper three voices play a harmonized version of the line from the Tuba 1 part in measure 2. During these two measures, Tuba 2 plays the F-F-G^b-E^b pattern in quarter notes.

The next phrase, beginning in measure 19, is one of the sections that focuses less on melodic material and more on the ensemble grooving together. In measures 19-22, the groove is generally created by the tubas playing the first two eighth notes with the euphoniums playing a quarter note on beat 2. Then the tubas play a bass rhythm in tandem during the remaining two beats. The tubas enter a sixteenth note apart from each other in fifths to mimic an electric bass playing on two different strings. The exception is measure 20, where the tubas play a different funk line in octaves. The F-G^b-E^b pattern is found in these measures with the F in the tuba parts on beat one and the E^b and G^b in the euphonium parts on beat two. Measures 23-26 are similar but focus on the dominant chord again. In these measures, Tuba 1 plays four sixteenth notes on beat one, Euphonium 2 plays four sixteenth notes on beat two, and Euphonium 1 plays a sixteenth note run in the remaining beats. The sixteenth notes in Tuba 1 focus on a G, and Tuba 2 plays E^b and F in beats three and four, forming a G-E^b-F pattern. Measures 27-30 resemble measures 19-22 but with Tuba 1 playing the F-B^b-C^b-E^b theme on beat one, and the euphoniums playing sixteenth notes on beats three and four. Measures 31-32 mimic measures 17-18 but with a modified Tuba 2 resembling the bass material from measures 19-30.

The next phrase in measures 33-46 follows the same general form as the previous phrase with a section in F, a section in B^b instead of C, another section in F, and the closing two measures. It maintains constant sixteenth notes scattered throughout the Euphonium 1, Euphonium 2, and Tuba 2 parts. In measures 33-36, Tuba 1 provides the basic bass line while the other parts provide the constant sixteenth notes. Tuba 2 picks up the bass line in measure 37 while the euphonium parts maintain the sixteenth notes; Euphonium 2 plays two sixteenth notes, then Euphonium 1 plays two sixteenth notes, and they continue alternating. Tuba 1 contrasts the quick sixteenth notes with a melodic line made of longer note values, namely half notes and quarter notes. This melody also contains instances of the F-E^b-G^b pattern. In measures 41-44, Tuba 1 continues the melody and Tuba 2 continues the bass line. The euphonium parts play the F-B^b-C^b-E^b pattern as sixteenth notes in a general arch shape. That is, the parts follow the pattern going up in pitch the first two beats of each measure, and follow the pattern down the last two beats. However, in measure 44, the euphoniums mimic the material from measures 33-36 to build intensity going into the concluding two measures of the phrase. Again the last two measures resemble measures 17-18 with a modified bass line.

Measure 47 marks the final phrase of the movement. The euphonium parts play a rhythmic pattern based on a repetition of an eighth note followed by a sixteenth note with the Euphonium 1 part containing an F-E^b-G^b pattern. The tubas play F in octaves as two strong eighth notes on beat one of each measure, then fill in the rest of the measure while the euphoniums rest. Tuba 1 plays the F-B^b-C^b-E^b pattern on beat four. At measure 51, the euphoniums switch parts, and the tubas echo in fifths the pattern in the euphoniums. Tuba 2 echoes the pattern three times: the first beginning on E^b, the second on G^b, and the third on F. This forms an E^b-G^b- F pattern across the three measures. The final measure follows the form of the other phrases and resembles measure 18.

The overall structure of the movements also reflects the three-note pattern. *Energy* is in the key of E^b maj., *Grief* is in E min./G maj., and *Funk-ish* is in F min. This resembles Figure 3.2 in that the triangle on the E vertex is the largest, so most of the musical work is in E^b maj. or E min. The next largest triangle is on the F vertex, so F min. covers the next largest section of the work. Finally, the triangle on the G vertex is smallest, so the smallest portion of the work is in G maj. All of the movements use similar permutations of E-G-F from the Sierpiński Pedal L-System at various levels of the music, but the effects created by these permutations vastly differ from each other between the individual movements.

Bibliography

1. Andrilli, S. and Hecker, D (2016). Computer Graphics. *Elementary Linear Algebra*. Elsevier Inc.
2. Devaney, R. L. (1995, April). *The Sierpinski Triangle*. Retrieved from Boston University, http://math.bu.edu/DYSYS/chaos-game/node2.html.
3. Ding, J., Hitt, L. R., Wang, B., & Zhang, X. *Sierpinski Pedal Triangle.*
4. Hazard, C., Johnson, D., & Kimport, C. (1999). *Fractal Music*. Retrieved from http://www.tursiops.cc/fm/.
5. Lasota, A. & Mackey, M. C. (1994). Iterated Function Systems and Fractals. *Chaos, fractals, and noise* (2nd ed.). New York: Springer-Verlag New York, Inc.
6. Peitgen, H., Jürgens, H., & Saupe, D. (2004). The Backbone of Fractals. *Chaos and fractals: New frontiers of science* (2nd ed.). New York: Springer-Verlag,New York, Inc.

Appendices

Appendix A: SAGE Codes

Algorithm 1: *redo_points*

```
def redo_points(a1,a2,b1,b2,c1,c2):
    if a2 < b2 and a2 < c2:
        if c1 < b1:
            A = matrix([[a1],[a2]])
            B = matrix([[b1],[b2]])
            C = matrix([[c1],[c2]])
        elif b1 < c1:
            A = matrix([[a1],[a2]])
            B = matrix([[c1],[c2]])
            C = matrix([[b1],[b2]])
        elif b1 == c1:
            if b1 < a1:
                if b2 < c2:
                    A = matrix([[a1],[a2]])
                    B = matrix([[c1],[c2]])
                    C = matrix([[b1],[b2]])
                else:
                    A = matrix([[a1],[a2]])
                    B = matrix([[b1],[b2]])
                    C = matrix([[c1],[c2]])
            else:
                if c2 < b2:
                    A = matrix([[a1],[a2]])
                    B = matrix([[c1],[c2]])
                    C = matrix([[b1],[b2]])
                else:
                    A = matrix([[a1],[a2]])
                    B = matrix([[b1],[b2]])
                    C = matrix([[c1],[c2]])
    elif a2 == b2 and a2 < c2:
        if a1 < b1:
            A = matrix([[a1],[a2]])
            B = matrix([[b1],[b2]])
            C = matrix([[c1],[c2]])
        else:
            A = matrix([[b1],[b2]])
            B = matrix([[a1],[a2]])
            C = matrix([[c1],[c2]])
    elif a2 == c2 and a2 < b2:
        if a1 < c1:
```

```
            A = matrix([[a1],[a2]])
            B = matrix([[c1],[c2]])
            C = matrix([[b1],[b2]])
        else:
            A = matrix([[c1],[c2]])
            B = matrix([[a1],[a2]])
            C = matrix([[b1],[b2]])
    elif b2 < a2 and b2 < c2:
        if c1 < a1:
            A = matrix([[b1],[b2]])
            B = matrix([[a1],[a2]])
            C = matrix([[c1],[c2]])
        elif a1 < c1:
            A = matrix([[b1],[b2]])
            B = matrix([[c1],[c2]])
            C = matrix([[a1],[a2]])
        elif a1 == c1:
            if a1 < b1:
                if a2 < c2:
                    A = matrix([[b1],[b2]])
                    B = matrix([[c1],[c2]])
                    C = matrix([[a1],[a2]])
                else:
                    A = matrix([[b1],[b2]])
                    B = matrix([[a1],[a2]])
                    C = matrix([[c1],[c2]])
            else:
                if a2 < c2:
                    A = matrix([[b1],[b2]])
                    B = matrix([[a1],[a2]])
                    C = matrix([[c1],[c2]])
                else:
                    A = matrix([[b1],[b2]])
                    B = matrix([[c1],[c2]])
                    C = matrix([[a1],[a2]])
    elif b2 == c2 and b2 < a2:
        if b1 < c1:
            A = matrix([[b1],[b2]])
            B = matrix([[c1],[c2]])
            C = matrix([[a1],[a2]])
        else:
            A = matrix([[c1],[c2]])
            B = matrix([[b1],[b2]])
            C = matrix([[a1],[a2]])
    elif c2 < a2 and c2 < b2:
        if a1 < b1:
            A = matrix([[c1],[c2]])
            B = matrix([[b1],[b2]])
            C = matrix([[a1],[a2]])
```

```
        elif b1 < a1:
            A = matrix([[c1],[c2]])
            B = matrix([[a1],[a2]])
            C = matrix([[b1],[b2]])
        elif a1 == b1:
            if a1 < c1:
                if a2 < b2:
                    A = matrix([[c1],[c2]])
                    B = matrix([[b1],[b2]])
                    C = matrix([[a1],[a2]])
                else:
                    A = matrix([[c1],[c2]])
                    B = matrix([[a1],[a2]])
                    C = matrix([[b1],[b2]])
            if c1 < a1:
                if a2 < b2:
                    A = matrix([[c1],[c2]])
                    B = matrix([[a1],[a2]])
                    C = matrix([[b1],[b2]])
                elif b2 < a2:
                    A = matrix([[c1],[c2]])
                    B = matrix([[b1],[b2]])
                    C = matrix([[a1],[a2]])
    return A,B,C
```

Algorithm 2: *Pedal_Triangle_Matrix_3*

```
def Pedal_Triangle_Matrix_3(a1,a2,b1,b2,c1,c2):

    A0,B0,C0 = redo_points(a1,a2,b1,b2,c1,c2)
    j = A0[0,0]
    k = A0[1,0]

    B0 = matrix([[B0[0,0]-A0[0,0]],[B0[1,0]-A0[1,0]]])
    C0 = matrix([[C0[0,0]-A0[0,0]],[C0[1,0]-A0[1,0]]])
    A0 = matrix([[A0[0,0]-A0[0,0]],[A0[1,0]-A0[1,0]]])
    rotation = arctan((B0[1,0]-A0[1,0])/(B0[0,0]-A0[0,0]))
    #R(theta) = matrix([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])
    A0 = matrix([[cos(-rotation),-sin(-rotation)],[sin(-rotation),cos(-rotation)]])*A0
    B0 = matrix([[cos(-rotation),-sin(-rotation)],[sin(-rotation),cos(-rotation)]])*B0
    C0 = matrix([[cos(-rotation),-sin(-rotation)],[sin(-rotation),cos(-rotation)]])*C0
    #return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])]),
    #aspect_ratio = 1)
```

```
m1 = ((B0[1,0]-C0[1,0])/(B0[0,0]-C0[0,0])) #for finding A1 (of pedal triangle)
m2 = ((C0[1,0]-A0[1,0])/(C0[0,0]-A0[0,0])) #for finding B1
#m3 should be zero since we rotated the triangle


a0 = sqrt((B0[0,0]*1.-C0[0,0])**2+(B0[1,0]*1.-C0[1,0])**2) #distance from B0 to C0
b0 = sqrt((A0[0,0]*1.-C0[0,0])**2+(A0[1,0]*1.-C0[1,0])**2) #distance from A0 to C0
c0 = sqrt((A0[0,0]*1.-B0[0,0])**2+(A0[1,0]*1.0-B0[1,0])**2)#distance from A0 to B0

theta_A = arccos((a0**2 - b0**2 - c0**2)/(-2*b0*c0)) #the measure of angle A
theta_B = arccos((b0**2 - a0**2 - c0**2)/(-2*a0*c0)) #the measure of angle B
theta_C = arccos((c0**2 - a0**2 - b0**2)/(-2*a0*b0)) #the measure of angle C

a_1 = a0*cos(theta_A)
b_1 = b0*cos(theta_B)
c_1 = c0*cos(theta_C)

#producing the bottom left triangle

#We shrink the original triangle
Shrink1 = matrix([[a_1/a0, 0],[0,a_1/a0]])
A_1 = Shrink1*A0 # this should result in [[0],[0]]
A_2 = Shrink1*B0
A_3 = Shrink1*C0

#We reflect the triangle about a line First we need to find the slope of the line
#about which we will reflect the triangle
k1 = 0. #the slope between A0 and B0
k3 = m2 #the slope between A0 and C0
k2 = var('k2')
sols3 = solve([(k2-k1)/(1.+k2*k1) == (k3-k2)/(1.+k3*k2)],k2)
k2 = round(sols3[1].right_hand_side(),10)
M = 1/(1. +k2**2)*matrix([[1-k2**2,2*k2],[2*k2,k2**2 -1]])
A_1 = M*A_1
A_2 = M*A_2
A_3 = M*A_3
#return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])])+
#polygon([(A_1[0,0],A_1[1,0]),(A_2[0,0],A_2[1,0]),(A_3[0,0],A_3[1,0])], color =
#'red',zorder = 5), aspect_ratio = 1) due to the way we repositioned the original
#triangle,this smaller triangle is done


#producing the bottom right triangle

#Shrink
Shrink2 = matrix([[b_1/b0, 0],[0,b_1/b0]])
B_1 = Shrink2*A0 # this should result in [[0],[0]]
B_2 = Shrink2*B0
B_3 = Shrink2*C0
```

```
#Reflect
B_1 = M*B_1
B_2 = M*B_2
B_3 = M*B_3
#return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])])+
#polygon([(B_1[0,0],B_1[1,0]),(B_2[0,0],B_2[1,0]),(B_3[0,0],B_3[1,0])], color =
#'red',zorder = 5), aspect_ratio = 1)

#Now we need to rotate the triangle to the correct position
B_1 = matrix([[round(cos(theta_C - pi),10),round(-sin(theta_C  -
pi),10)],[round(sin(theta_C - pi),10),round(cos(theta_C - pi),10)]])*B_1
B_2 =matrix([[round(cos(theta_C - pi),10),round(-sin(theta_C  -
pi),10)],[round(sin(theta_C - pi),10),round(cos(theta_C - pi),10)]])*B_2
B_3 = matrix([[round(cos(theta_C - pi),10),round(-sin(theta_C  -
pi),10)],[round(sin(theta_C - pi),10),round(cos(theta_C - pi),10)]])*B_3

#Now we translate the triangle to the correct position
B_1 = B_1 - B_2 + B0
B_3 = B_3 - B_2 + B0
B_2 = B_2 - B_2 + B0
#return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])])+
#polygon([(B_1[0,0],B_1[1,0]),(B_2[0,0],B_2[1,0]),(B_3[0,0],B_3[1,0])], color =,
#'red',zorder = 5), aspect_ratio = 1)


#producing the top triangle

#Shrink
Shrink3 = matrix([[c_1/c0, 0],[0,c_1/c0]])
C_1 = Shrink3*A0 # this should result in [[0],[0]]
C_2 = Shrink3*B0
C_3 = Shrink3*C0

#Reflect
C_1 = M*C_1
C_2 = M*C_2
C_3 = M*C_3
#return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])])+
#polygon([(B_1[0,0],B_1[1,0]),(B_2[0,0],B_2[1,0]),(B_3[0,0],B_3[1,0])], color =
#'red',zorder = 5), aspect_ratio = 1)

#Now we need to rotate the triangle to the correct position

C_1 = matrix([[round(cos(pi - theta_B),10),-round(sin(pi - theta_B),10)],
[round(sin(pi - theta_B),10),round(cos(pi - theta_B),10)]])*C_1
C_2 = matrix([[round(cos(pi - theta_B),10),-round(sin(pi - theta_B),10)],
[round(sin(pi - theta_B),10),round(cos(pi - theta_B),10)]])*C_2
C_3 = matrix([[round(cos(pi - theta_B),10),-round(sin(pi - theta_B),10)],
```

```
    [round(sin(pi - theta_B),10),round(cos(pi - theta_B),10)]])*C_3


    #Now we translate the triangle to the correct position
    C_1 = C_1 - C_3 + C0
    C_2 = C_2 - C_3 + C0
    C_3 = C_3 - C_3 + C0
    #return show(polygon([(A0[0,0],A0[1,0]),(B0[0,0],B0[1,0]),(C0[0,0],C0[1,0])])+
    #polygon([(C_1[0,0],C_1[1,0]),(C_2[0,0],C_2[1,0]),(C_3[0,0],C_3[1,0])], color =
    #'red',zorder = 5), aspect_ratio = 1)

    #rotate all points to the original triangle's position
    A_1 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*A_1
    A_2 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*A_2
    A_3 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*A_3
    B_1 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*B_1
    B_2 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*B_2
    B_3 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*B_3
    C_1 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*C_1
    C_2 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*C_2
    C_3 = matrix([[cos(rotation),-sin(rotation)],[sin(rotation),cos(rotation)]])*C_3

    #translate triangles to position of the original
    A = matrix([[j],[k]])
    A_1 = A_1 + A
    A_2 = A_2 + A
    A_3 = A_3 + A
    B_1 = B_1 + A
    B_2 = B_2 + A
    B_3 = B_3 + A
    C_1 = C_1 + A
    C_2 = C_2 + A
    C_3 = C_3 + A

    #return show(polygon([(A_1[0,0],A_1[1,0]),(A_2[0,0],A_2[1,0]),(A_3[0,0],
    # A_3[1,0])], color = 'black') +polygon([(B_1[0,0],B_1[1,0]),(B_2[0,0],B_2[1,0]),
    #(B_3[0,0],B_3[1,0])],color = 'black')\
    #+ polygon([(C_1[0,0],C_1[1,0]),(C_2[0,0],C_2[1,0]),(C_3[0,0],C_3[1,0])], color =
    #'black'), aspect_ratio = 1)
    return A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3
```

Algorithm 3: *SPT_Matrix*

```
def SPT_Matrix(a1, a2, b1, b2, c1, c2, level):
    P = Pedal_Triangle_Matrix_3(a1,a2,b1,b2,c1,c2)
    p11 = P[0][0][0]
    p12 = P[0][1][0]
    p21 = P[1][0][0]
    p22 = P[1][1][0]
```

```
    p31 = P[2][0][0]
    p32 = P[2][1][0]
    p41 = P[3][0][0]
    p42 = P[3][1][0]
    p51 = P[4][0][0]
    p52 = P[4][1][0]
    p61 = P[5][0][0]
    p62 = P[5][1][0]
    p71 = P[6][0][0]
    p72 = P[6][1][0]
    p81 = P[7][0][0]
    p82 = P[7][1][0]
    p91 = P[8][0][0]
    p92 = P[8][1][0]
    #print (p11, p12), (p21, p22), (p31, p32), (p41, p42), (p51, p52), (p61, p62),
    #(p71, p72), (p81, p82), (p91, p92)
    T = [(p11,p12,p21,p22,p31,p32,level),(p41,p42,p51,p52,p61,p62,level),
    (p71,p72,p81,p82,p91,p92,level)]

    if level - 1 > 0:
        print "Processing.   ",level
        T += SPT_Matrix(p11,p12,p21,p22,p31,p32,level-1)
        print "Processing..  ",level
        T += SPT_Matrix(p41,p42,p51,p52,p61,p62,level-1)
        print "Processing... ",level
        T += SPT_Matrix(p71,p72,p81,p82,p91,p92,level-1)
        print "Moving to next one"

    return T
```

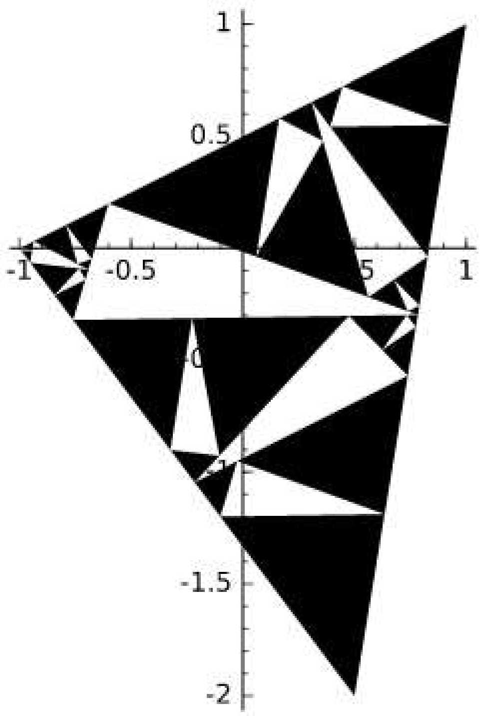
Algorithm 4: *SPT_Graph*

```
def SPT_Graph(a1,a2,b1,b2,c1,c2,level):
    T = SPT_Matrix(a1,a2,b1,b2,c1,c2,level)
    #print "I have a matrix"
    p = Graphics()
    for P in T:
        if P[6] == 1:
            p += polygon([(P[0],P[1]),(P[2],P[3]),(P[4],P[5])], color = 'black')
    #print "I have a polygon"
    return show(p, aspect_ratio = 1)
```

Appendix B: Example Graphs

```
SPT_Graph(-1.,0.,1.,1.,1./2., -2., 3)
```

```
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
```



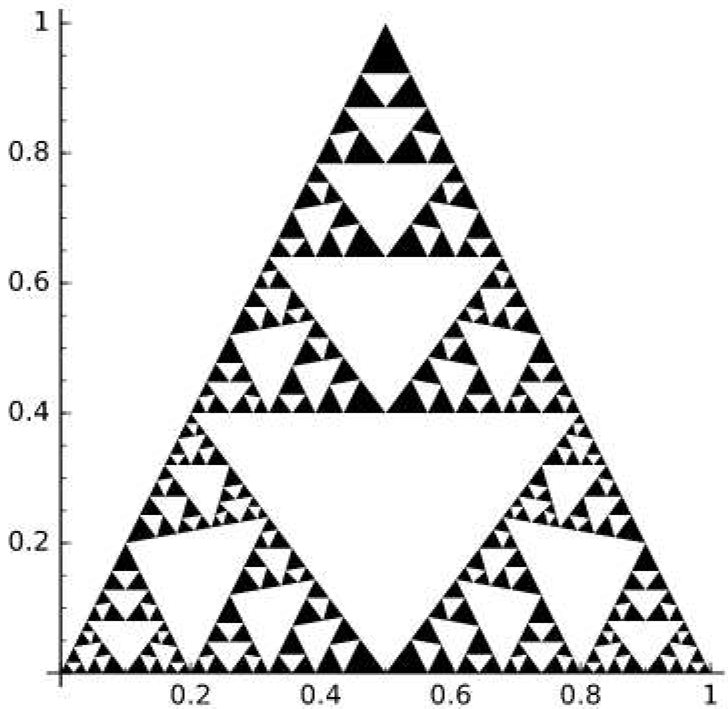
SPT_Graph(0.,0.,1.,0.,1./2., 1., 5)

```
Processing.    5
Processing.    4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Processing..   4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Processing...  4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
```

```
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Moving to next one
Processing..   5
Processing.    4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Processing..   4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Processing...  4
Processing.    3
Processing.    2
```

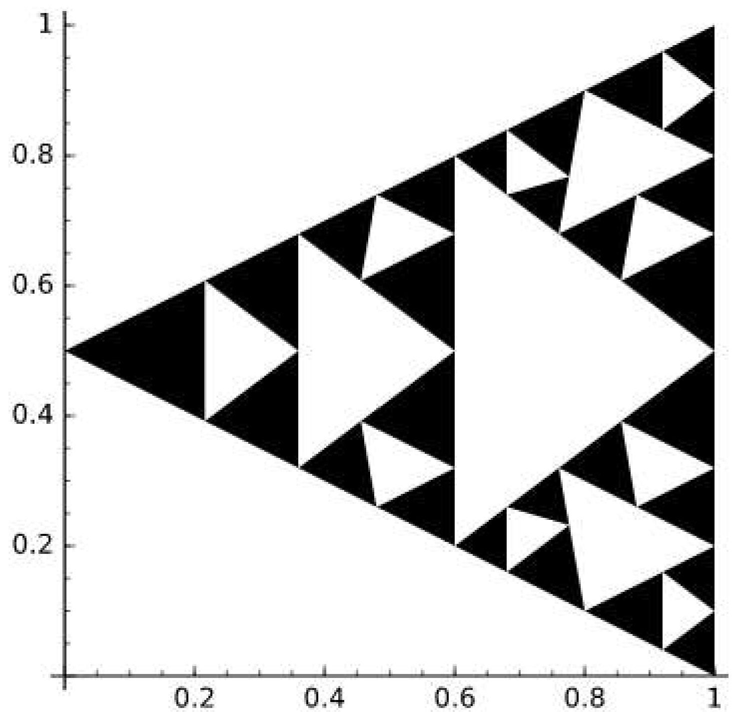
```
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Moving to next one
Processing...  5
Processing.    4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Processing..   4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
```

```
Processing...  2
Moving to next one
Moving to next one
Processing...  4
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
Moving to next one
Moving to next one
```



```
SPT_Graph(0.,1./2,1.,0,1., 1., 3)
```

```
Processing.    3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing..   3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Processing...  3
Processing.    2
Processing..   2
Processing...  2
Moving to next one
Moving to next one
```



Appendix C: *Sierpiński Pedal Quartet*

The following pages contain the *Sierpiński Pedal Quartet*. At the beginning of each movement, the lines of the staff system are labeled with the beginning letter of the instrument part and the part number (e.g. E 1 = Euphonium 1, T 1 = Tuba 1).

Score

Energy

Sam Dent


8
f
12
13
f


15
fp
f
fp
mp
fp
mp
fp
mf
18

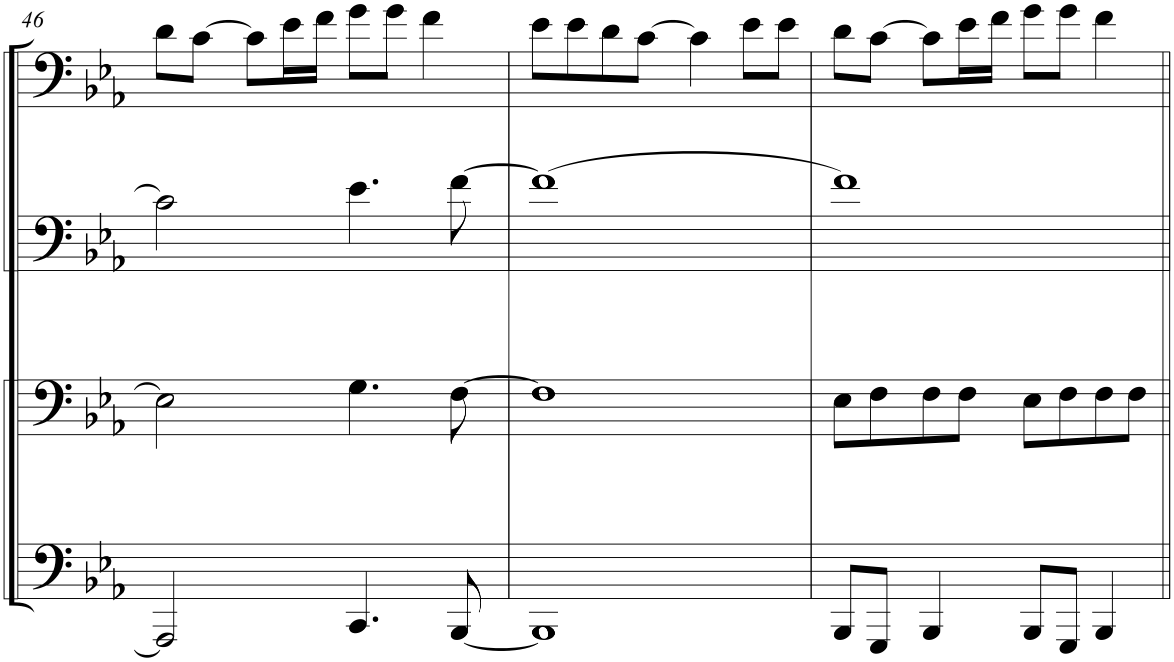

21
24


27
29
f
f
30


33


36


41
39
mf
p
cresc.
mf
p
cresc.
cresc.
p
43
f


46


49
f
f
mf
f


52
55


58
61


65
fp
f
fp
mp
fp
mp
fp
mf
69


72
75
fp
fp
fp


78
f
f


81


84
mf
87


90
ff
ff
ff
ff
92

Score

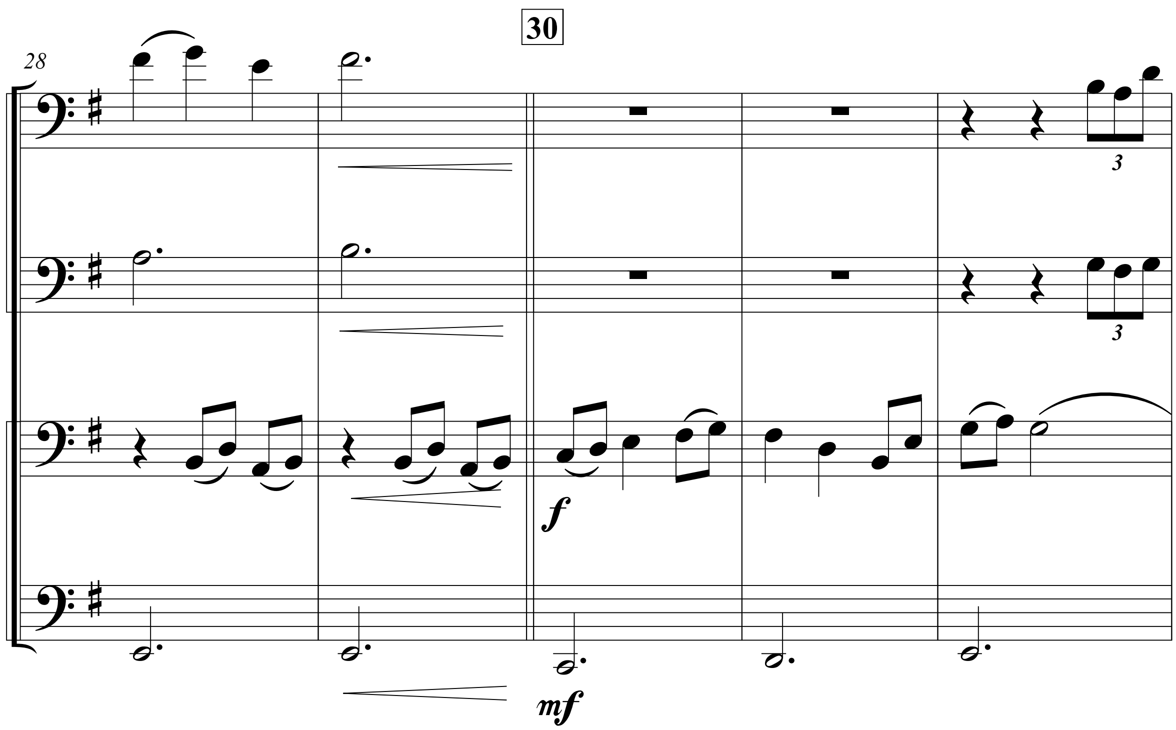
Grief

Sam Dent


13
mf

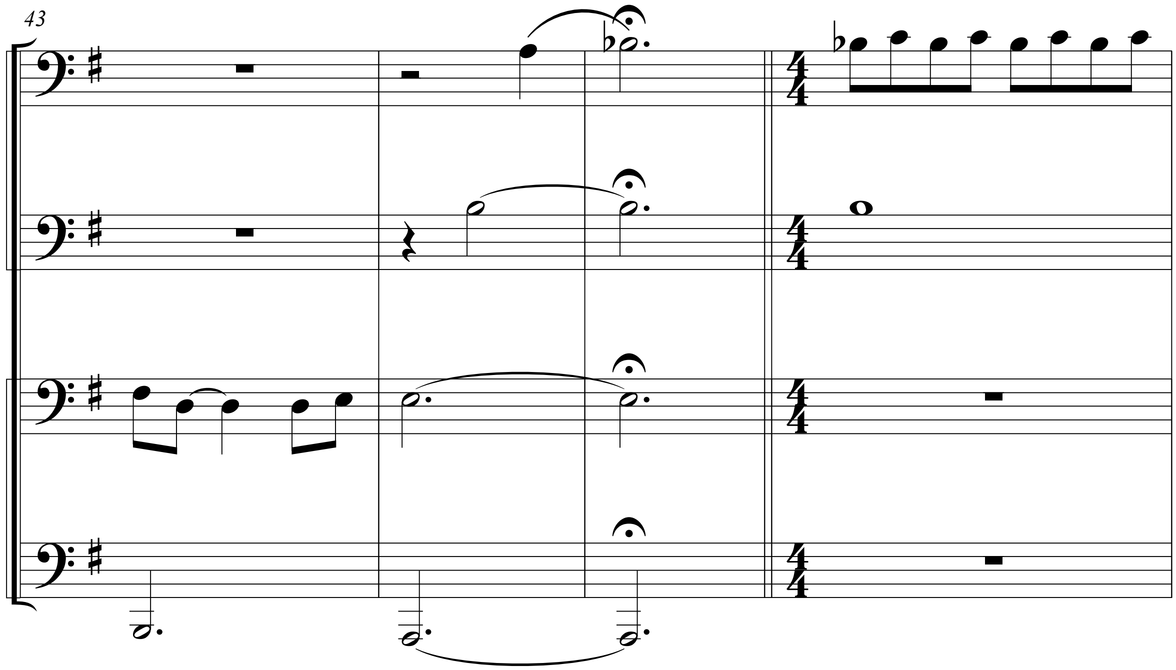

18

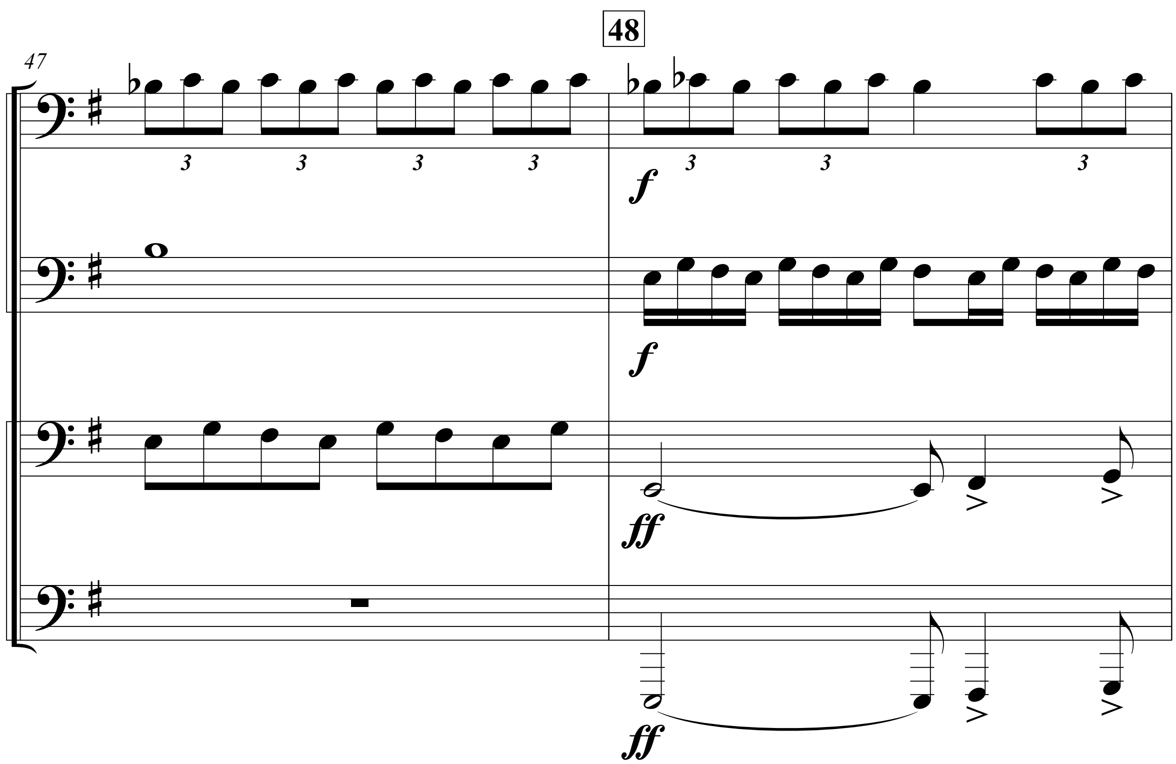

23


28
30
3
3
f
mf


33


38


43


47
48
f
f
ff
ff


49
3
3
3
3
3
3
3
3
51
3
3
3
3
3
3


53


55
ff
ff

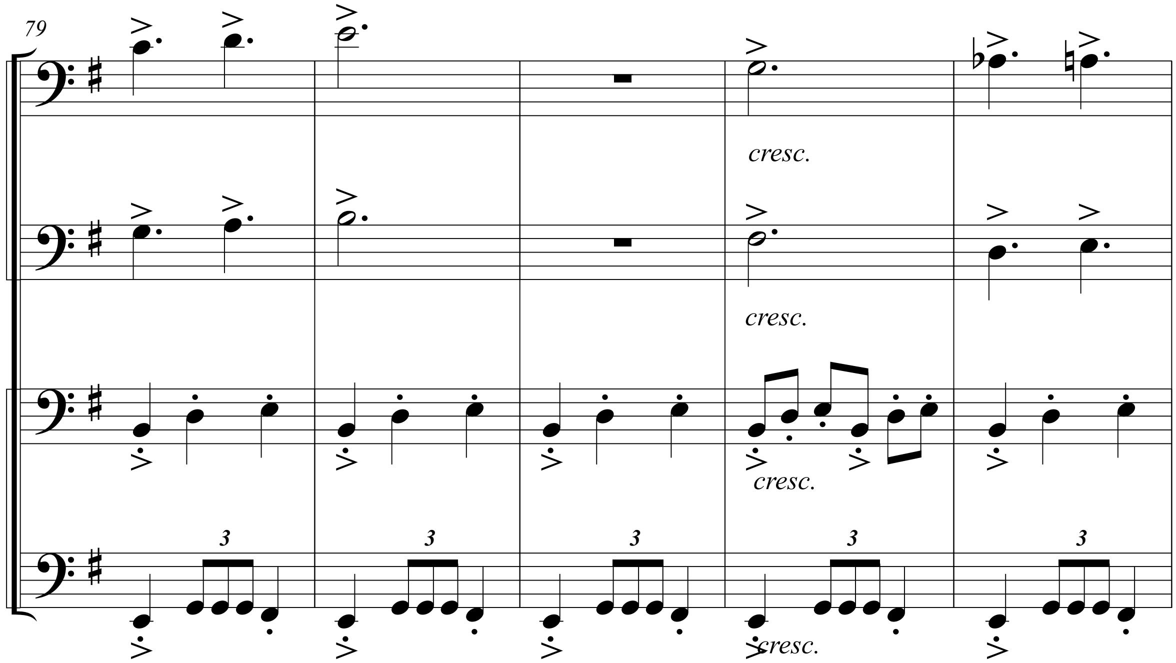

57
59
f
ff


62

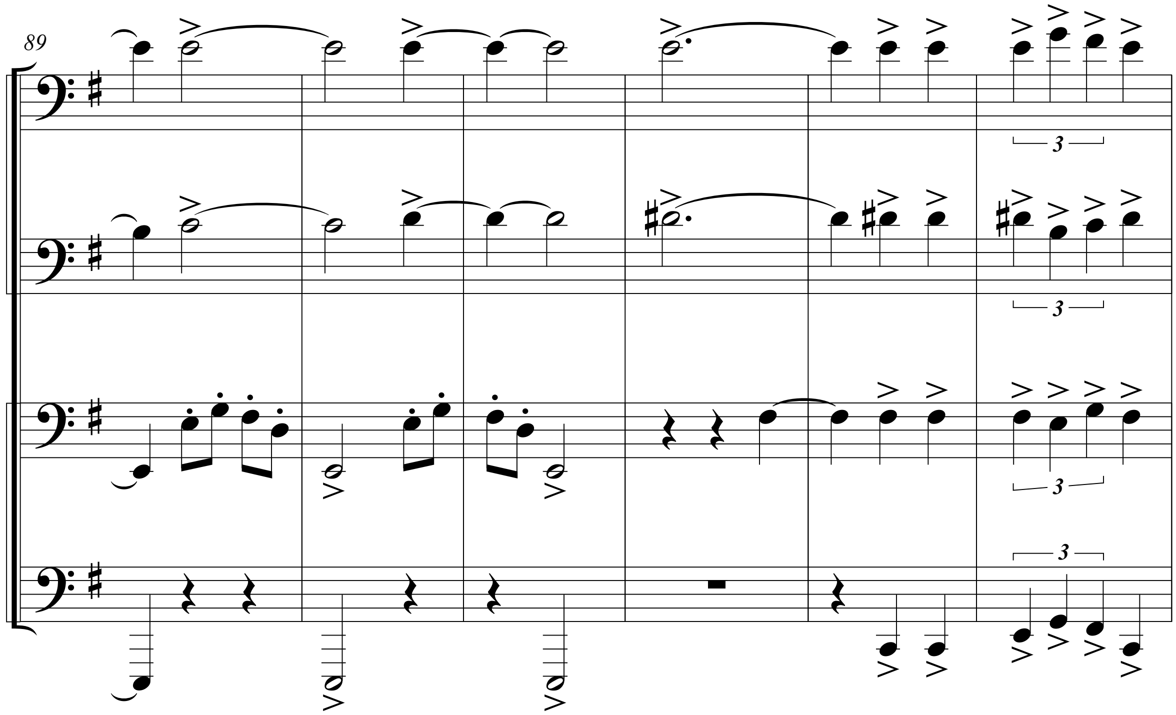

64
(♪=♩)♩ = 160
f
f
f
mf
f
mf

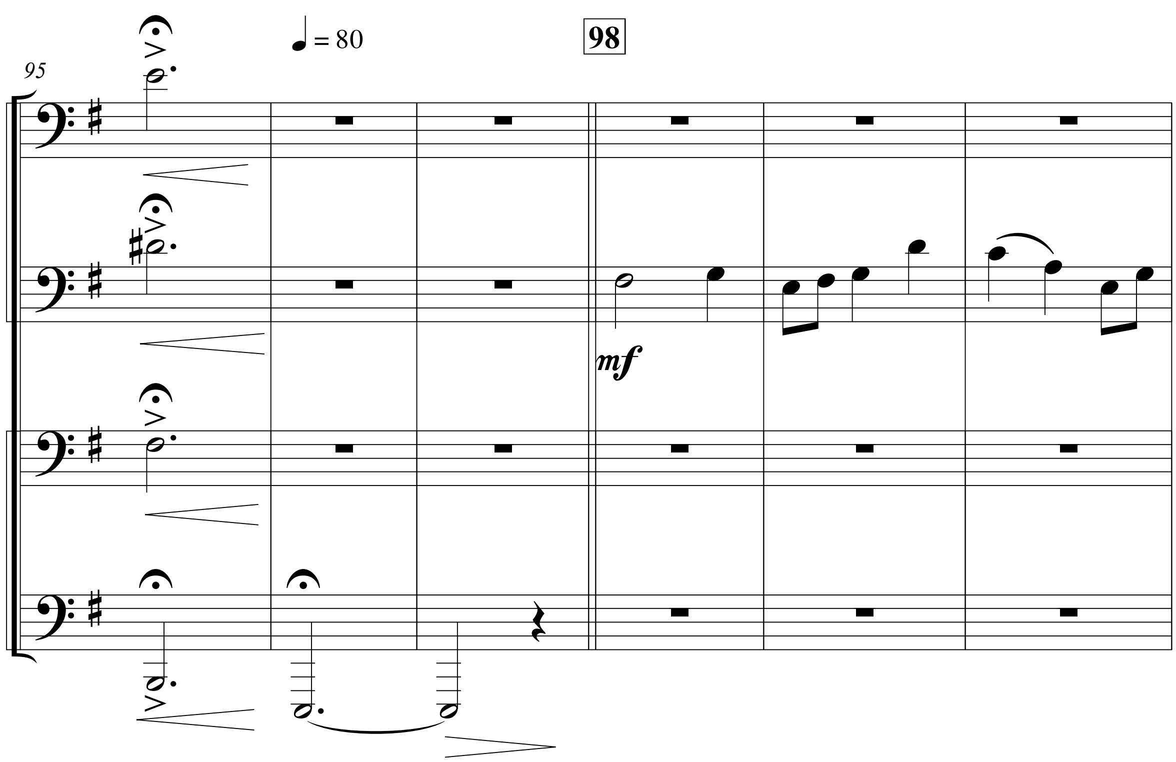

69


74

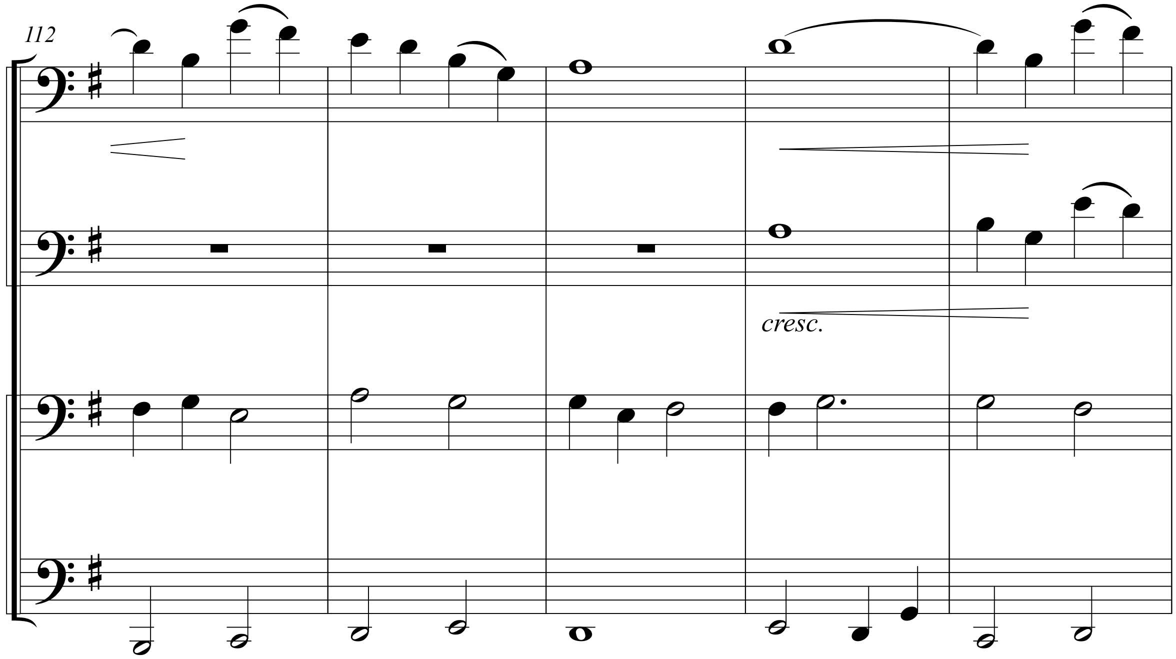

79
cresc.
cresc.
cresc.
cresc.
3

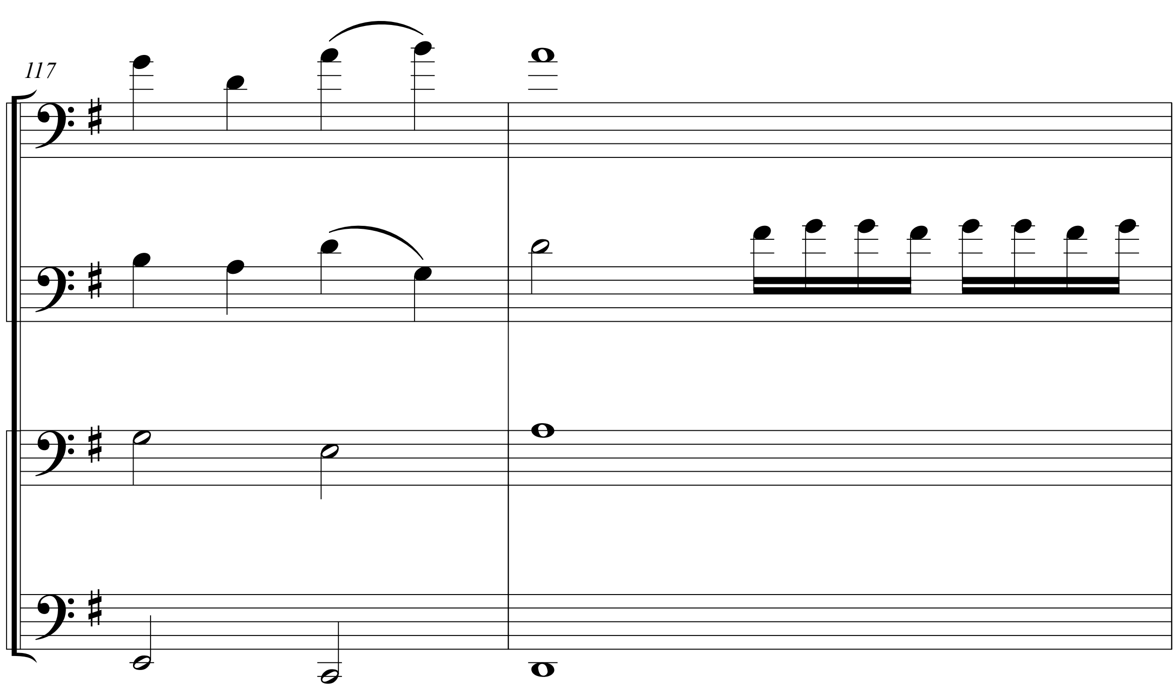

84
ff
ff
ff
ff
3


89
3
3
3
3


95
= 80
98
mf

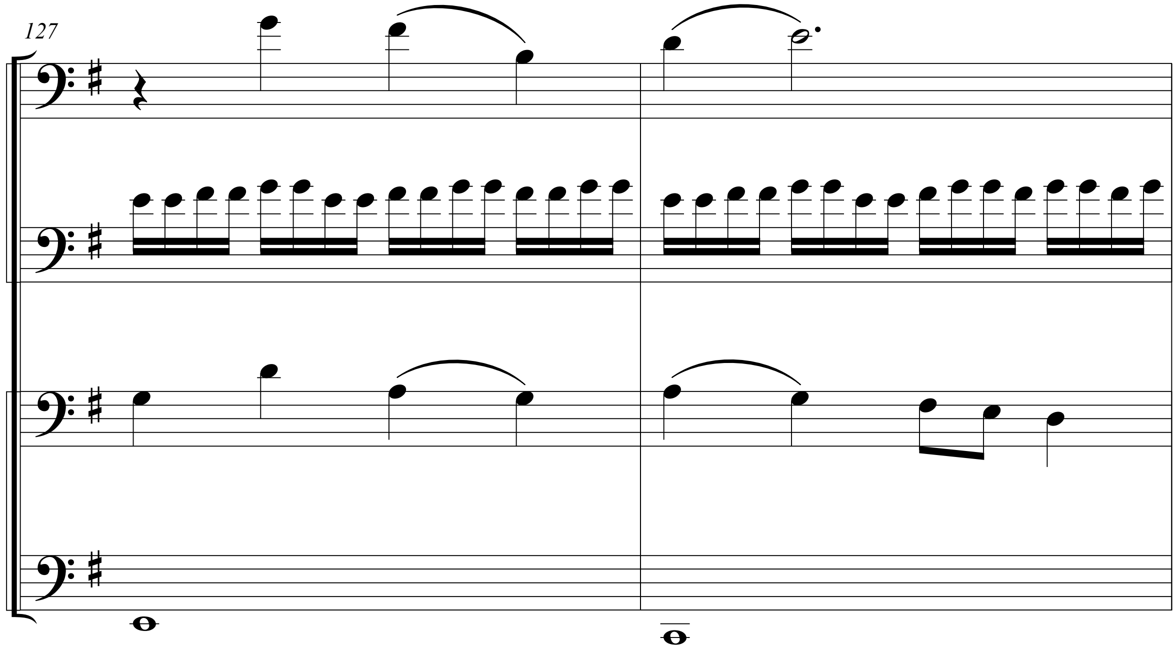

101
mf
mf
107
rit.
111
= 80
cresc.
pp
cresc.
pp
cresc.


112
cresc.


117

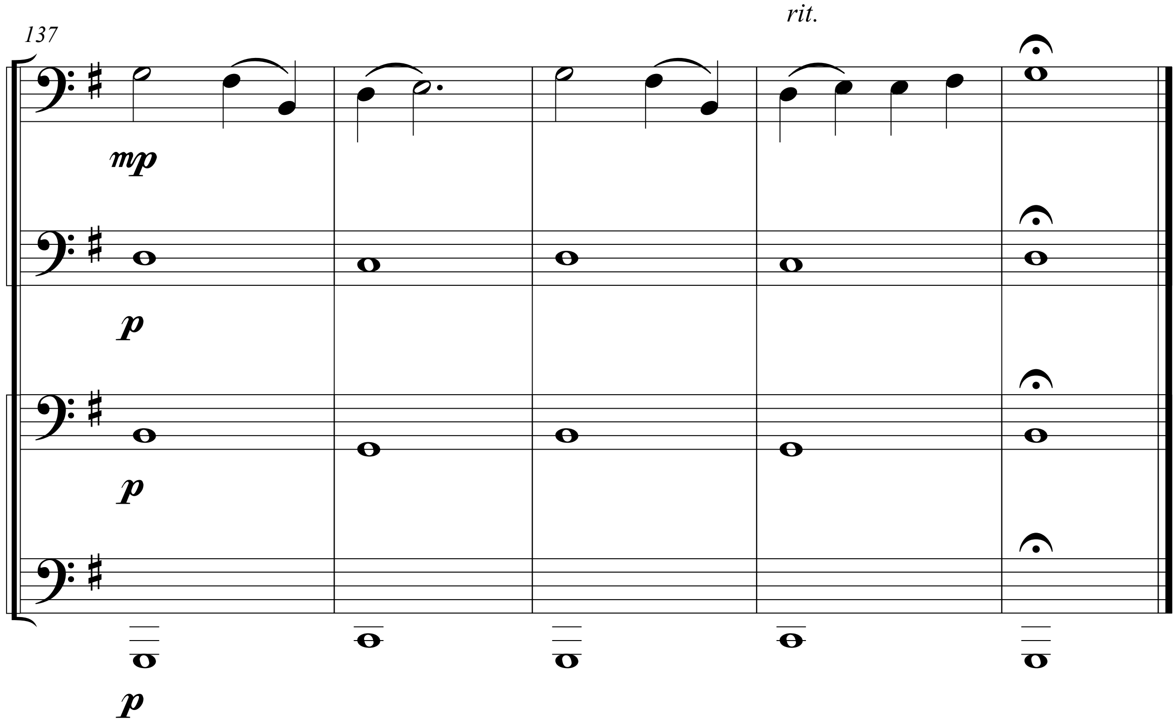

119
3
f
f
3
f
3
f
121
3
3
3


123
ff
ff
ff
ff
125


127


129


131
134
3
3


137
rit.
mp
p
p
p

Score

Funk-ish

Sam Dent


5


8
9
f
mf


10
mf
12
f
mf


14
16


18
19
mf
mf
mf
mf
20


22
fp
f
fp
f
f
f
24


26
mf
cresc.
mf
cresc.
mp
cresc.
mp
cresc.
28


30
f
f
f
f
32
33
mf
mf
mf


34
mf
36
f


38


40
f
f
f


42
44


46
47
mp cresc.
mp cresc.
p cresc.
p cresc.
48


50
ff
ff
ff
ff


52


53
54