Score Generation in Voice-Leading Orbifolds

Michael Gogins
gogins@pipeline.com

23rd July 2005

I describe the use of algorithms to generate musical scores in spaces whose points are chords ordered such that neighbors have the least possible voice-leading between them, the movement of one voice by one semitone. Such spaces are called voice-leading spaces; musically speaking, they are completed forms of the *Tonnetz* of Euler, Oettingen, and Reimann; mathematically speaking, they are orbifolds. In the present work, time is not represented as a dimension of any voice-leading space, but rather as the sequence of chords; in other words, a score is considered to be a function of time onto chords in some voice-leading space. This is a "natural" representation of music in the sense that generative algorithms can be constructed from primitive operations, namely moving about through voice-leading space in various ways, that have an obvious musical interpretation according to some basic rules of pragmatic music theory. In particular, shorter movements in voice-leading space sound smoother than longer movements, and avoiding "oblique" movements in voice-leading space directly avoids all parallel motions of voices. At the same time, the use of voice-leading spaces imposes no constraints on the generality or universality of compositional algorithms. All movements of voices are possible, none are privileged, none are forbidden. To illustrate the concepts, I describe the implementation of a simple $0L$ Lindenmayer system, or string rewriting grammar, that generates scores in voice-leading orbifolds, with some musical examples. I conclude by considering potential adaptations of other compositional algorithms to voice-leading orbifolds, including parametric and evolutionary algorithms.

1 Introduction

Algorithmic composition, or the use of automated procedures to generate musical scores, is as old as computer music, dating back to Hiller and Isaacson [1]. Today there are many varieties of algorithmic composition [2]. I would broadly categorize them as either "grammatical", i.e. based on some kind of generative grammer, or "mathematical", i.e. based on mathematical operations or forms such as fractals. My impression is that the mathematical approach has been more *musically* fruitful than the grammatical approach, which suffers by comparison with the works of the many composers who understand some particular grammar of musical style far better than any algorithm. The mathematical approach has the artistic advantage of readily generating scores that would be difficult for any composer to imagine but, of course, it also suffers from the overproduction of music that offends ears trained by stylistically well-formed pieces.

I think one reason mathematically-based algorithms overproduce ill-formed pieces is that such algorithms operate "outside of time." By this I mean that the space in which the algorithms operate is typically a simple Euclidean space having the same dimensions as the musical score itself: for example, time × pitch × instrument. This makes the algorithms easy to construct, but when they operate, there is nothing to prevent them from folding pitches back through time so that different pitch-class sets end up overlaying each other on the score at the same point in time, in a way that

is difficult to predict or control. Therefore, obtaining a harmonic progression or counterpoint (I use these terms here without respect to whether the music is tonal or atonal, serial or non-serial, stochastic or deterministic) with the desired effect, if possible at all, requires combining some understanding how the algorithms work with a process of trial and error.

Another obstacle is that, as discussed by Justin London [3], pitch as a dimension of music does not seem to be isomorphic to time as another dimension of music. This implies that forms produced by operations on pitch cannot be expected to resemble the forms produced by similar operations on time.

Consequently, I have been searching for mathematically-based algorithms that either prevent rotating pitch-class sets through time, or constrain operations by stylistic rules, or both. And I have been looking for ways to do this while retaining a purely mathematical basis for the algorithms — in other words, without creating hybrids in which the production of some mathematically-based algorithm is filtered or massaged by a second, grammatically-based algorithm. My experience is that such hybrids close off musical possibilities and obscure formal unity.

One way to avoid rotating pitch through time is to require that all parameters of the score be a function of time. This means that the score space must have *at least* one dimension for each voice of music. However, thanks to my limited mathematical skills, I was never able to find any mathematical or geometric representation of voice-leading or counterpoint — until recently, that is, when the work of Dmitri Tymoczko [4] on voice-leading spaces came to my attention. In a voice-leading space, each unordered n-note chord is represented as a point on the orbifold T^n/S_n, that is, the n-torus *modulo* the symmetric group S_n.

Tymoczko demonstrates that such spaces, which are a generalization of the *Tonnetz*, encapsulate basic symmetries and constraints of Western music. In particular, voice-leading spaces clarify how common chords are flexible with respect both to harmonic progression and voice-leading, and these spaces provide a very simple geometric illustration of some basic rules of voice-leading and inner symmetries of music. Figure 1 shows the voice-leading orbifold for trichords, arranged to show the "layers" composed by iterating stepwise motions from voice 1 through voice 3. Each ball represents a chord of three voices (whether the chord is a complete triad, some other chord of three voices, or a unison), and each of the lines joining the balls represents a voice-leading motion of one semitone. Therefore, the whole network of lines represents the complete *Tonnetz* joining every unordered trichord. Chords with the same interval structure have the same color. The labelled chord is C major. The complete Python source code for producing this interactive model can be found in Listing 1.

Figure 2 shows the same orbifold rotated to show how chords with the same interval structure are arranged in oblique columns, and the whole orbifold forms a prism in the voice-leading space. The white column in the middle consists of the augmented triads, the three blue columns surrounding the augmented triads consist of the minor triads, the three red columns surrounding the augmented triads consist of the major triads (again, the labelled chord is C major), and the three red columns along the edges of the prism consist of the unisons. Moving one step up or down a column amounts to tranposing a chord of that type up or down a semitone. Therefore the chords in the columns are *not* joined by *one* semitone voice-leading! Rather, the trichords in the columns are joined by *three* one-semitone voice-leadings in parallel motion. It is obvious that the smoothest voice-leadings move along the lines of the *Tonnetz*, and that avoiding oblique motions (i.e., not moving up or down the columns) avoids parallel motions of voices.

Voice-leading orbifolds for tetrachords, pentachords, and higher numbers of voices would have a similar structure, but could not be shown in their entirety using 3-dimensional graphics, which can show only a slice of a higher-dimensional orbifold.

Once I saw Tymoczko's preprint, it occurred to me that instead of mapping mathematically-based compositional algorithms onto some Euclidean "score space", it would probably be more useful to map them onto voice-leading spaces. As my initial investigation of this approach, I have written a program that generates scores using non-parametric ($0L$) Lindenmayer systems [5] in which the generating "turtle" moves about, not in a Euclidean score space, but in a voice-leading orbifold for a preset number of voices.

2 Implementation

The starting point for score generation in orbifolds is that arithmetic on orbifolds can be represented using regular matrix arithmetic, with one additional step. Because orbifolds are quotient spaces, after each operation each voice in each chord vector must be taken *modulo* the size of the space.

It is important to note that for the purposes of music theory the symmetric group of the orbifold is the *octave*, but for purposes of score generation the symmetric group is the *range of the voices*.

Lindenmayer systems are recursive functions that rewrite strings [5]. Each Lindenmayer system consists of an axiom or initial string of atoms, a table of replacement rules each specifying how an atom is to replaced with a string of atoms, an implicit rule that an atom with no replacement is replaced by itself, and the number of iterations for the recursion. In addition, some of the atoms of the system are commands for a "turtle" in a turtle graphics-like system. For example, `F` might mean move one step while drawing a line, `f` move one step without drawing a line, `+` turn right, `-` turn left, `[` push the turtle state onto a stack, and `]` pop the turtle state from the stack. (Pushing starts a branch, and popping returns the turtle to the branching point.) If the turtle operates in a multi-dimensional space, the turtle can be defined as a position vector in the space, the step as another vector, movement as adding the step vector to the turtle vector, and rotation as multiplying the step vector by a rotation matrix, so that the step points in a new direction.

The Lindenmayer system is iterated for the specified number of recursions, during which the repeated replacements usually expand the initial axiom into a very long string of atoms, called the production of the system. The production is then interpreted as a program for the turtle, which draws a figure in the space. This is the simplest type of Lindenmayer system, or OL system, in which the replacement rules do not depend on the state of the production on either side of the current atom, and do not take parameters.

OL systems have already been used for some time to generate musical scores in spaces where time is one dimension of the space [6, 7, 8]. To adapt OL systems to generating scores in voice-leading spaces, where each dimension of the space is a voice and time is simply the sequence of movements, was straightforward. I implemented the score generator using Python [9], the SciPy package which among other things provides efficient matrix arithmetic and linear algebra routines for Python [10], and CsoundVST, an extended version of Csound that includes Python scripting and some facilities for mathematically-based algorithmic composition [11]. The complete source code is shown in Listing 2.

What distinguishes this algorithm from merely doing arithmetic on notes *modulo* the range of the score is that (a) each point in the space, and therefore each position of the turtle in the Lindenmayer system, represents not a note but a chord, and (b) the chords follow each other in strict temporal sequence, all voices moving together. Consequently, every movement of the turtle represents a simple voice-leading, and every voice-leading between two chords, and so every sequence of two chords, can be represented by a series of movements of the turtle. The difference between generating *voice-leadings* and generating *notes* becomes obvious after listening to pieces generated by the present algorithm, as compared with pieces generated by other Lindenmayer system algorithms.

Some comments are in order:

- All operations on the turtle are performed using matrix arithmic from SciPy's `Numeric` modules for linear algebra, but the `stayInsideOrbifold` function is called after each operation to ensure that the result remains within the orbifold.
- The `createRotation` function creates a matrix for performing a rotation *from* one dimension *towards* another dimension, i.e. *around* the axis orthogonal to those dimensions.
- There are N voices in each chord, but I have added N additional dimensions to the space for the instrument choice, and N more additional dimensions for the loudness, of each voice. This affords a degree of control over orchestration and dynamics, in addition to pitch and time.
- Time is defined purely by succession of chords, in order of the interpretation by the turtle of the final production. There is one time step between each chord.

Command	Meaning
F	Move the turtle position one step and create a chord.
f	Move the turtle position one step but do not create a chord.
+a, b	Rotate the turtle step by angle A from dimension a towards dimension b.
-a, b	Rotate the turtle step by angle A from dimension b towards dimension a.
-a, b	Rotate the turtle step by angle A from dimension b towards dimension a.
*x	Multiply the size of the turtle step by x.
/x	Divide the size of the turtle step by x.
[	Push the turtle state and step onto a stack.
]	Pop the turtle state and step from the stack.
C	Create a chord at the current turtle position.
=td, x	Assign x to dimension d of the turtle position.
atd, x	Add x to dimension d of the turtle position.
std, x	Subtract x from dimension d of the turtle position.
mtd, x	Multiply dimension d of the turtle position by x.
dtd, x	Divide dimension d of the turtle position by x.
=sd, x	Assign x to dimension d of the turtle step.
asd, x	Add x to dimension d of the turtle step.
ssd, x	Subtract x from dimension d of the turtle step.
msd, x	Multiply dimension d of the turtle step by x.
dsd, x	Divide dimension d of the turtle step by x.
=Tx	Assign x to the size of the time step.
aTx	Add x to the size of the time step.
sTx	Subtract x from the size of the time step.
mTx	Multiply the size of the time step by x.
dTx	Divide the size of the time step by x.

Table 1: **Turtle Commands**

- The default number of voices is 4, and the default starting state for the turtle is a major 7th chord in first inversion. Of course this can be changed programmatically.

- The Lindenmayer system provides a `tieNotes` option for tying repeated notes in one voice. This option is implemented in the `createChord` function by comparing, at each time step, each voice of the current chord with the previous or continuing chord. If the pitch or instrument assignment of the voice has not changed, the duration of the continuing note is incremented by one time step. If the pitch or instrument has changed, the continuing note is written to the score, and it is replaced in the continuing chord by the current note. Tying notes in this way overcomes, to a considerable degree, the rhythmic monotony of the one time step per chord rule.

- In any case, tied or not, notes written to the score are shortened by 1/3 of a time step, in order not to be completely *legato*.

- The Lindenmayer system subclasses the `ScoreNode` class of the Silence composition system in CsoundVST, even though the Lindenmayer system is a pure Python class, while `ScoreNode` is a C++ class with a Python wrapper. This is made possible by the *director* facility in SWIG [12], the tool that I used to generate the Python bindings for CsoundVST. This enables scores generated in voice-leading orbifolds by the Lindenmayer system to be subjected to further manipulations by the Silence system, and to be rendered by Csound.

Table 1 summarizes the turtle commands and their meanings. The first group of commands is a minimal set typically found in *OL* systems. I added the second group of commands in hopes of making the system more flexible for composing. The third group of commands controls the size of the time step, i.e. tempo.

3 Results

To investigate the potential of this Lindenmayer system, I generated a number of scores. I began with a very simple example in order to develop an understanding of the system (not to mention to debug the code), and then I introduced complications, step by step, to produce more sophisticated scores.

3.1 Proof of Concept

To demonstrate the correctness of the implementation, the following Lindenmayer system was iterated.

The expected result is a score with five sections: the first (`C`) proves that a chord can be repeated without moving any voices, the second (`[ =s01 =s10 =s20 =s30 F ]`) takes the first voice and increments its pitch till it reaches the top of the orbifold, then cycles back up from the bottom; the third (`[ =s00 =s11 =s20 =s30  F ]`) takes the second voice and cycles it; the fourth (`[ =s00 =s10 =s21 =s30 F ]`) takes the third voice and cycles it; and the fifth (`[ =s00 =s10 =s20 =s31 F ]`) takes the fourth voice and cycles it. The `[` command pushes the state of the turtle position and step onto a stack, the `=s31` command (for example) assigns 1 to the 3rd dimension of the turtle step, and the `]` command pops the state of the turtle position and step from the stack. The resulting score is as expected, and is shown in Figure 3.

3.2 Tying Notes

Next, I rendered the same score, but this time with repeating notes tied (Figure 4).

3.3 Something More Like Music

Next, I created a Lindenmayer system designed to sound more like music (Listing 4, first page of score shown in Figure 5).

The axiom of this system divides the size of the turtle step by four, and starts with a seed atom `A`. The one replacement rule rotates the turtle, i.e. selects which voices to move, then moves in voice-leading space and writes a chord, or moves without writing a chord. This system also 'branches' and, inside the branch, enlarges the size of the turtle step.

3.4 Controlling Tempo, Dynamics, and Instrumentation

As a final study, I created a variant of the previous system that uses additional turtle commands to control the time step (tempo), dynamics, and instrument choice (Listing 5). The first page of the resulting score is shown in Figure 6. In this case, the changing instrument assignments are obvious because when a voice changes to a different instrument, it also moves to a different staff.

3.5 Moving Voices by Absolute Increment

I also created other studies using the turtle commands for moving one voice at a time by an absolute increment (the second group of commands in Table 1). I found that these studies were not as musically interesting as the studies mentioned above, which were created by first changing the orientation of the turtle in voice-leading space, and then moving the turtle. Therefore, I have not included code or scores for these studies here.

4 Discussion

I believe that score generation in voice-leading spaces is a promising instrument of musical composition. The advantages I hoped to gain by using voice-leading spaces all seem to be present. On the one hand, it seems easier to keep the voice-leadings closer to the bounds of good taste than with Euclidean score spaces; on the other hand, this does not in the least seem to prevent the production of patterns that would be difficult to imagine — at least, for me.

It is natural to think of mapping other compositional algorithms to voice-leading spaces. One path for future reserarch is to map iterated function systems (IFS) to voice-leading spaces. The motive for this is that I have already shown iterated function systems [13] can be used to implement a form of parametric composition, by representing each IFS code as a point in a virtual parameter space [14, 15]. The parameter space can then be tiled with thumbnails or sketches of the compositions represented by each point, or colored by some analytic number derived from the score. Once this is done, it is possible to treat the parameter space as if it was a Mandelbrot set for scores, to zoom in and out of the space, analyze how certain types of scores are located in certain regions, and to compose very quickly with some insight into the musical structure and symmetry of the space.

Another future path is to evolve compositions by applying the genetic algorithm to Lindenmayer systems or IFS codes in voice-leading orbifolds.

Finally, there may well exist other spaces even better suited than voice-leading orbifolds for simultaneously representing both musical structure and score-generating operations. For example, voice-leading spaces do not usefully encode information about scales, root progressions, and other harmonic as opposed to contrapuntal constructs. It might also be desirable to make the octave registers of the voices independent dimensions of the space.

I regard the elucidation of better spaces for algorithmic composition as one of the most important tasks of contemporary pragmatic music theory.

5 Figures

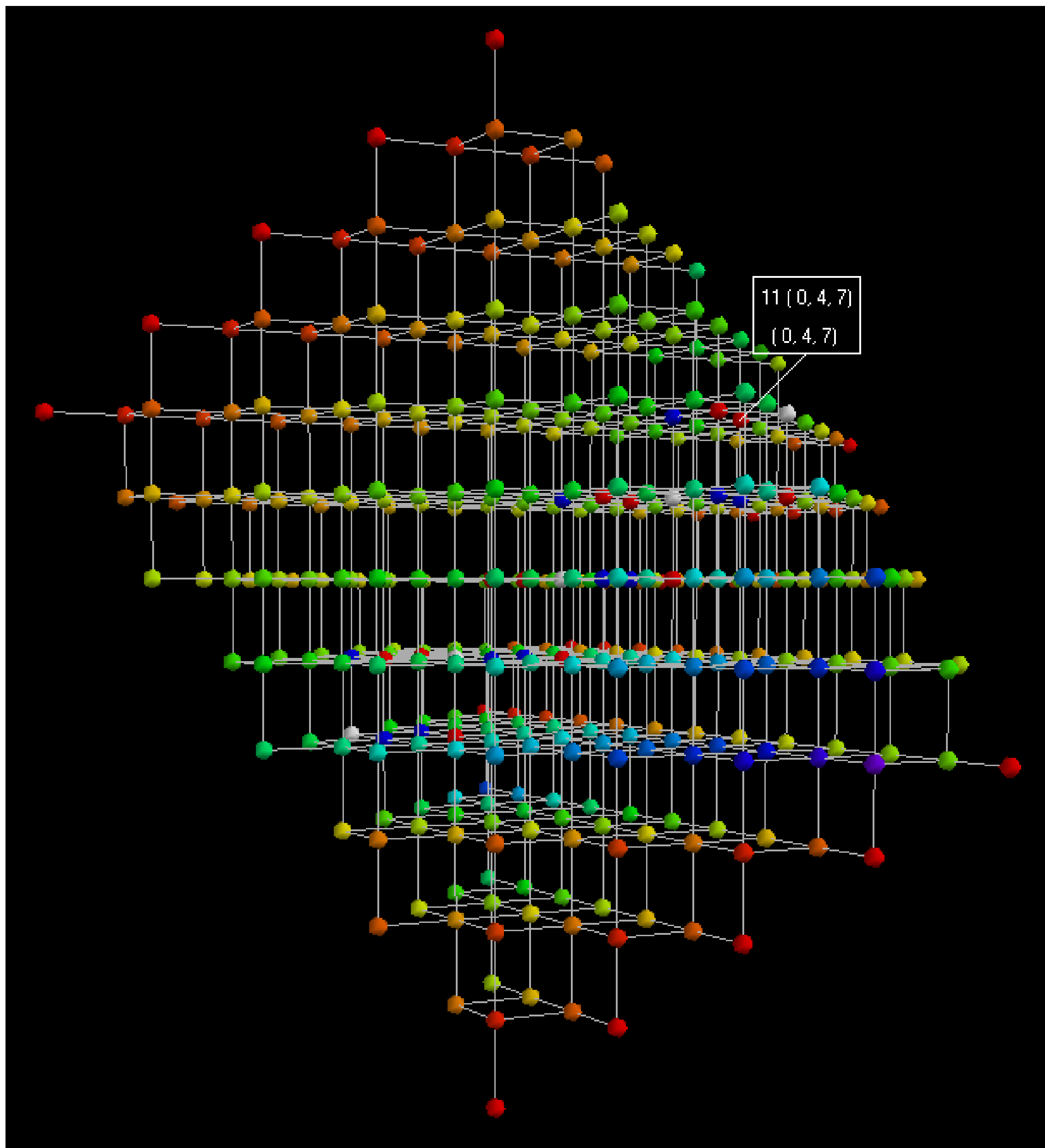

Figure 1: **Orbifold for Trichords Showing Layers**

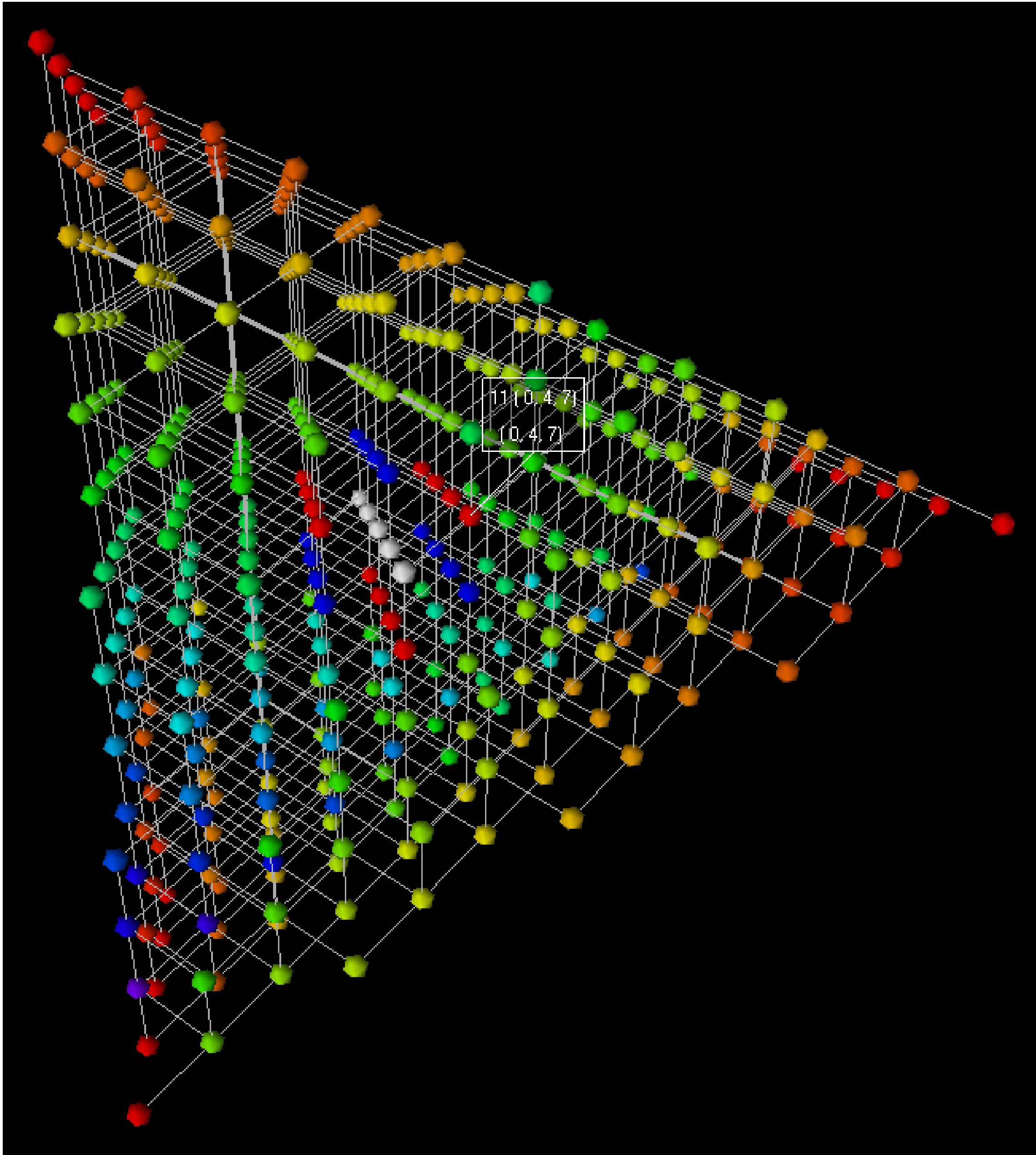

Figure 2: **Orbifold for Trichords Showing Columns**

Figure 3: **Proof of Concept**

Figure 4: **Proof of Concept with Repeating Notes Tied**

Figure 5: **Study A**

Figure 6: **First Page of Study B**

6 Program Listings

Listing 1: **Voice-Leading Orbifold for Trichords**

```
'''
V O I C E - L E A D I N G   O R B I F O L D
F O R   T R I C H O R D S   I N   1 2 T E T
Copyright 2005 by Michael Gogins.

See: http://ruccas.org/pub/Gogins/score_generation_in_voiceleading_orbifolds.pdf

The following software is used in this program:
Python 2.3 from http://www.python.org
VPython from http://www.vpython.org
CsoundVST from http://csound.sourceforge.net

N O T E
Wait for Csound to finish compiling before selecting any trichords.
You may need to change the dac number (sound card) in the code below.
And you need to have an ASIO driver for your sound card (or, ASIO4ALL).
'''
print __doc__
import gc
import sys
import time
# Change enableCsound to True if you have installed CsoundVST.
# Importing CsoundVST automatically creates a csound object.
enableCsound = False
keepPlaying = True
if enableCsound:
    import CsoundVST
from visual import *
import random
scene.title = "VOICE-LEADING ORBIFOLD: click ball=play, drag right button=spin, drag middle button=zoom, close window to stop"
scene.fullscreen = 0
scene.width = 800
scene.height = 600
scene.autoscale = 1
scene.exit = 0
def unorder(orderedTrichord):
    list = [orderedTrichord[0], orderedTrichord[1], orderedTrichord[2]]
    list.sort()
    return (list[0], list[1], list[2])
def modulusOf(trichord):
    newTrichord = ((trichord[0] - trichord[0] + 144) % 12,
                   (trichord[1] - trichord[0] + 144) % 12,
                   (trichord[2] - trichord[0] + 144) % 12)
    return newTrichord
def unorderedType(trichord):
    trichord = modulusOf(trichord)
    trichord = unorder(trichord)
    return trichord

trichords = {}
balls = {}
ballsForChordTypes = {}
def setColor(ball):
    inversion1 = unorderedType(ball.trichord)
    inversion2 = (inversion1[2] - 8, inversion1[0] + 4, inversion1[1] + 4)
    inversion3 = (inversion2[2] - 8, inversion2[0] + 4, inversion2[1] + 4)
    inversion1 = unorder(modulusOf(inversion1))
    inversion2 = unorder(modulusOf(inversion2))
    inversion3 = unorder(modulusOf(inversion3))
    inversion1Key = str(inversion1)
    inversion2Key = str(inversion2)
    inversion3Key = str(inversion3)
    if   inversion1Key in ballsForChordTypes:
        ball.color = ballsForChordTypes[inversion1Key].color
    elif inversion2Key in ballsForChordTypes:
        ball.color = ballsForChordTypes[inversion2Key].color
    elif inversion3Key in ballsForChordTypes:
        ball.color = ballsForChordTypes[inversion3Key].color
    else:
        # Color major triads red.
        if   inversion1 == (0, 4, 7) or inversion2 == (0, 4, 7) or inversion3 == (0, 4, 7):
            ball.color = (1.0,0.0,0.0)
        # Color augmented triads white.
        elif inversion1 == (0, 4, 8) or inversion2 == (0, 4, 8) or inversion3 == (0, 4, 8):
            ball.color = (1.0,1.0,1.0)
        # Color minor triads blue.
        elif inversion1 == (0, 3, 7) or inversion2 == (0, 3, 7) or inversion3 == (0, 3, 7):
            ball.color = (0.0,0.0,1.0)
        else:
            hue = (inversion1[0] + inversion1[1] * 2.0 + inversion1[2]) / 44.0
            saturation = 1.0
            value = 1.0
            ball.color = color.hsv_to_rgb((hue, saturation, value))
voices = 1
modulus = 12
octaves = 1
layers = (modulus) * voices * octaves
trichordCount = 0
for layer in xrange(0, layers + 1):
    for x in xrange(layer / voices - (2 * modulus / voices), (layer / voices + (modulus  / voices) + 1) * octaves):
        for y in xrange(x, (layer / voices + (2 * modulus  / voices) + 1) * octaves):
            for z in xrange(y, (layer / voices + (2 * modulus / voices) + 1) * octaves):
                sum = x + y + z
                if sum == layer and z <= (x + modulus):
                    trichord = unorder((x, y, z))
                    if trichord not in trichords:
                        trichordCount = trichordCount + 1
                        trichords[trichord] = trichord
                        ball = sphere(pos = trichord, radius = 0.125)
                        # Make a label for each ball, and hide it till it's needed.
                        ball.trichord = unorder(((x + 144) % modulus, (y + 144) % modulus, (z + 144) % modulus))
                        # print ball.trichord
                        balls[str(ball.trichord)] = ball;
                        if ball.trichord == (1,5,10):
```

```
                            scene.center = ball.pos
                        ball.layer = layer
                        ball.sum = ball.trichord[0] + ball.trichord[1] + ball.trichord[2]
                        setColor(ball)
                        ball.name = "%2d (%2d,%2d,%2d)\n  (%2d,%2d,%2d)" % (layer, ball.trichord[0], ball.trichord[1], ball.trichord
                            [2], ball.pos[0],ball.pos[1],ball.pos[2])
                        ball.label = label(pos = trichord, text = ball.name, height = 11, box = 1, opacity = 0.3, visible = 0, line =
                            1, xoffset=40,yoffset=40 )
    print
# Enlarge C, its closest F, and its closest G.
#balls['(0, 4, 7)'].radius = .25
#balls['(0, 5, 9)'].radius = .25
#balls['(2, 7, 11)'].radius = .25
def connect(origin, neighbor):
    if neighbor in trichords:
        curve(pos = [origin, neighbor], color = (0.67, 0.67, 0.67))
for trichord in trichords.values():
    connect(trichord, unorder((trichord[0] + 1.0, trichord[1], trichord[2])))
    connect(trichord, unorder((trichord[0], trichord[1] + 1.0, trichord[2])))
    connect(trichord, unorder((trichord[0], trichord[1], trichord[2] + 1.0)))
    connect(trichord, unorder((trichord[0] - 1.0, trichord[1], trichord[2])))
    connect(trichord, unorder((trichord[0], trichord[1] - 1.0, trichord[2])))
    connect(trichord, unorder((trichord[0], trichord[1], trichord[2] - 1.0)))
if enableCsound:
    def csoundThreadRoutine():
        csound.load('c:/utah/home/mkg/projects/csound5/examples/CsoundVST.csd')
        csound.setCommand('csound -d -m0 -b400 -B1200 -odac2 temp.orc temp.sco')
        csound.exportForPerformance()
        gc.disable()
        csound.compile()
        while keepPlaying:
            csound.performKsmps()
    csoundThread = threading.Thread(None, csoundThreadRoutine)
    csoundThread.start()
pickedBall = None
oldBall = None
while scene.visible:
    if scene.mouse.clicked:
        try:
            m = scene.mouse.getclick()
            if oldBall:
                oldBall.label.visible = 0
            if pickedBall:
                pickedBall.label.visible = 0
            if str(m.pick.trichord) in balls:
                oldBall = pickedBall
                pickedBall = m.pick
                if pickedBall:
                    pickedBall.label.visible = 1
                    note1 = "i  8 0 4 %d 70 0 -.75" % (60 + pickedBall.pos[0])
                    note2 = "i  8 0 4 %d 70 0  .0"  % (60 + pickedBall.pos[1])
                    note3 = "i  8 0 4 %d 70 0  .75" % (60 + pickedBall.pos[2])
                    if enableCsound:
                        csound.inputMessage(note1)
                        csound.inputMessage(note2)
                        csound.inputMessage(note3)
        except:
            pass
        scene.mouse.events = 0
if enableCsound:
    keepPlaying = False
    csoundThread.join()
    csound.cleanup()
print "Finished."
```

Listing 2: **Lindenmayer System**

```
'''
GENERATING SCORES WITH O-L LINDENMAYER SYSTEMS
IN VOICE-LEADING ORBIFOLDS
Copyright (C) 2005 by Michael Gogins
All rights reserved.

See: http://ruccas.org/pub/Gogins/score_generation_in_voiceleading_orbifolds.pdf
'''
import CsoundVST
from Numeric import *
import math
import sys
import copy

# To represent arithmetic on voice-leading orbifolds
# of N dimensions with regular matrix arithmetic,
# call this function after every matrix operation.
def stayInsideOrbifold(N, size, vector):
    # Take the modulus of each element by the size
    # of the orbifold.
    for i in xrange(N):
        x = vector[0, i]
        while x < 0:
            x += size
        while x > size:
            x -= size
        vector[0, i] = x

# Create a matrix of N dimensions
# to rotate from dimension 1 to dimension 2 by A.
def createRotationMatrix(N, dimension1, dimension2, A):
    matrix = identity(N, Float)
    matrix[dimension1,dimension1] =   math.cos(A)
    matrix[dimension1,dimension2] = - math.sin(A)
    matrix[dimension2,dimension1] =   math.sin(A)
    matrix[dimension2,dimension2] =   math.cos(A)
    return matrix

class ChordLindenmayer(CsoundVST.ScoreNode):
    def __init__(self):
        CsoundVST.ScoreNode.__init__(self)
        self.tieNotes = True
```

```
        self.voices = 4
        self.range = 60.0
        self.timeStep = 1.0
        self.dimensionsPerVoice = 3
        self.dimensions = self.voices * self.dimensionsPerVoice
        self.angle = math.pi / 2.0
        self.generations = 3
        self.axiom = ""
        self.production = ""
        self.rules = {}
        self.continuingChord = []
        self.score = self.getScore()
        self.score.thisown = 0
        # The turtle is a row vector.
        self.turtle = zeros((1, self.dimensions), Float)
        self.turtle[0, 0] = 30.0 +  0.0
        self.turtle[0, 1] = 30.0 +  4.0
        self.turtle[0, 2] = 30.0 +  7.0
        self.turtle[0, 3] = 30.0 + 11.0
        # Voices and loudnesses are pre-assigned but of course can change.
        for i in xrange(0, self.voices):
            self.turtle[0, self.voices + i] = float(2 + i)
            self.turtle[0, self.voices * 2 + i] = float(70)
    def generate(self):
        self.produce()
        self.interpret()
    def produce(self):
        self.production = self.axiom
        for generation in xrange(self.generations):
            tokens = string.split(self.production)
            self.production = ""
            for token in tokens:
                if self.rules.has_key(token):
                    self.production += self.rules[token]
                else:
                    self.production += token
                self.production += ' '
        # print self.production
    def interpret(self):
        self.continuingChord = []
        self.currentChord = []
        self.durations = {}
        self.step = zeros((1, self.dimensions), Float)
        self.step[0, 0] = 1
        tokens = string.split(self.production)
        turtlestack = []
        stepstack = []
        self.time = 0.0
        for token in tokens:
            command = token[0]
            if   command == 'F':
                self.turtle = self.turtle + self.step
                stayInsideOrbifold(self.voices, self.range, self.turtle)
                self.createChord(self.turtle)
            elif command == 'f':
                self.turtle = self.turtle + self.step
                stayInsideOrbifold(self.voices, self.range, self.turtle)
            elif command == '+':
                dimensions = string.split(token[1:], ',')
                rotator = createRotationMatrix(self.dimensions,
                                               int(dimensions[0]),
                                               int(dimensions[1]),
                                               + self.angle)
                self.step = matrixmultiply(self.step, rotator)
            elif command == '-':
                dimensions = string.split(token[1:], ',')
                rotator = createRotationMatrix(self.dimensions,
                                               int(dimensions[0]),
                                               int(dimensions[1]),
                                               - self.angle)
                self.step = matrixmultiply(self.step, rotator)
            elif command == '*':
                self.step = self.step * float(token[1:])
            elif command == '/':
                self.step = self.step / float(token[1:])
            elif command == '[':
                turtlestack.append(array(self.turtle))
                stepstack.append(array(self.step))
            elif command == ']':
                self.turtle = turtlestack.pop()
                self.step = stepstack.pop()
            elif command == 'C':
                stayInsideOrbifold(self.voices, self.range, self.turtle)
                self.createChord(self.turtle)
            elif command == '=':
                subcommand = token[1]
                args = string.split(token[2:], ',')
                if subcommand == 't':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.turtle[0, dimension] = value
                    stayInsideOrbifold(self.voices, self.range, self.turtle)
                elif subcommand == 's':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.step[0, dimension] = value
                elif subcommand == 'T':
                    value = float(args[0])
                    self.timeStep = value
            elif command == 'a':
                subcommand = token[1]
                args = string.split(token[2:], ',')
                if subcommand == 't':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.turtle[0, dimension] = self.turtle[0, dimension] + value
                    stayInsideOrbifold(self.voices, self.range, self.turtle)
                elif subcommand == 's':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.step[0, dimension] = self.step[0, dimension] + value
```

```
                elif subcommand == 'T':
                    value = float(args[0])
                    self.timeStep = self.timeStep + value
            elif command == 's':
                subcommand = token[1]
                args = string.split(token[2:], ',')
                if subcommand == 't':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.turtle[0, dimension] = self.turtle[0, dimension] - value
                    stayInsideOrbifold(self.voices, self.range, self.turtle)
                elif subcommand == 's':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.step[0, dimension] = self.step[0, dimension] - value
                elif subcommand == 'T':
                    value = float(args[0])
                    self.timeStep = self.timeStep - value
            elif command == 'm':
                subcommand = token[1]
                args = string.split(token[2:], ',')
                if subcommand == 't':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.turtle[0, dimension] = self.turtle[0, dimension] * value
                    stayInsideOrbifold(self.voices, self.range, self.turtle)
                elif subcommand == 's':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.step[0, dimension] = self.step[0, dimension] * value
                elif subcommand == 'T':
                    value = float(args[0])
                    self.timeStep = self.timeStep * value
            elif command == 'd':
                subcommand = token[1]
                args = string.split(token[2:], ',')
                if subcommand == 't':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.turtle[0, dimension] = self.turtle[0, dimension] / value
                    stayInsideOrbifold(self.voices, self.range, self.turtle)
                elif subcommand == 's':
                    dimension = int(args[0])
                    value = float(args[1])
                    self.step[0, dimension] = self.step[0, dimension] / value
                elif subcommand == 'T':
                    value = float(args[0])
                    self.timeStep = self.timeStep / value
        if self.tieNotes:
            # Write the final chord.
            for continuingNote in self.continuingChord:
                continuingNote[1] = continuingNote[1] - (self.timeStep / 3.0)
                self.score.append(continuingNote[0],
                                  continuingNote[1],
                                  continuingNote[2],
                                  continuingNote[3],
                                  continuingNote[4],
                                  continuingNote[5])
    def createChord(self, turtle):
        self.currentChord = []
        for i in xrange(self.voices):
            note = [ self.time,
                     self.timeStep,
                     144.0,
                     turtle[0, i + self.voices],
                     turtle[0, i],
                     turtle[0, i + self.voices * 2] ]
            self.currentChord.append(note)
        if not self.tieNotes:
            for currentNote in self.currentChord:
                currentNote[1] = currentNote[1] - (self.timeStep / 3.0)
                self.score.append(currentNote[0],
                                  currentNote[1],
                                  currentNote[2],
                                  currentNote[3],
                                  currentNote[4],
                                  currentNote[5])
        else:
            if self.time == 0:
                self.continuingChord = copy.deepcopy(self.currentChord)
            else:
                for i in xrange(self.voices):
                    currentNote = self.currentChord[i]
                    continuingNote = self.continuingChord[i]
                    if (round(currentNote[4]) == round(continuingNote[4])) and (round(currentNote[3]) == round(continuingNote[3])):
                        continuingNote[1] = continuingNote[1] + self.timeStep
                    else:
                        continuingNote[1] = continuingNote[1] - (self.timeStep / 3.0)
                        self.score.append(continuingNote[0],
                                          continuingNote[1],
                                          continuingNote[2],
                                          continuingNote[3],
                                          continuingNote[4],
                                          continuingNote[5])
                        self.continuingChord[i] = copy.deepcopy(currentNote)
        self.time = self.time + self.timeStep
```

Listing 3: **Proof of Concept**

```
'''
GENERATING SCORES WITH 0-L LINDENMAYER SYSTEMS
IN VOICE-LEADING ORBIFOLDS
Copyright (C) 2005 by Michael Gogins
All rights reserved.

See: http://ruccas.org/pub/Gogins/score_generation_in_voiceleading_orbifolds.pdf
'''
import CsoundVST
from Numeric import *
```

```
import math
import sys
import lindenmayer

print __doc__

lindenmayer = lindenmayer.ChordLindenmayer()
lindenmayer.tieNotes = False
lindenmayer.range = 24
lindenmayer.axiom = 'C [ =s0,1 =s1,0 =s2,0 =s3,0 F ] [ =s0,0 =s1,1 =s2,0 =s3,0  F ] [ =s0,0 =s1,0 =s2,1 =s3,0 F ] [ =s0,0 =s1,0 =s2,0
      =s3,1 F ]'
lindenmayer.generations = 4
lindenmayer.timeStep = 0.125
lindenmayer.rules['F'] = 'F F F'
lindenmayer.rules['C'] = 'C C'
lindenmayer.generate()

# Place the ChordLindenmayer node inside a Random node to randomize velocity and pan,
# place the Random node inside a Rescale node,
# and place the Rescale node inside the MusicModel.

model = CsoundVST.MusicModel()
model.setCppSound(csound)
random = CsoundVST.Random()
random.createDistribution("uniform_01")
random.setElement(6, 11, 1)
random.setElement(8, 11, 1)
rescale = CsoundVST.Rescale()
rescale.setRescale(CsoundVST.Event.INSTRUMENT, 1, 1,  0,       4)
rescale.setRescale(CsoundVST.Event.KEY,        1, 0, 48,       0)
rescale.setRescale(CsoundVST.Event.VELOCITY,   1, 1, 65,       6)
rescale.setRescale(CsoundVST.Event.PAN,        1, 1, -0.25,    1.5)
random.addChild(lindenmayer)
rescale.addChild(random)
model.addChild(rescale)
model.generate()
filename = 'simple'
model.getScore().save(filename + '.mid')
instruments = [8,11,9,14,14]
for event in model.getScore():
    event.setInstrument(instruments[int(event.getInstrument())])
    print event.toString()


print 'Filename:', filename
model.setConformPitches(False)
model.setTonesPerOctave(12.0)
csound.load('c:/utah/home/mkg/projects/music/library/orchestra.csd')
csound.setCommand("csound -m7 -RWdfo " + filename + ".wav " + filename + ".orc " + filename + ".sco")
csound.setFilename(filename)
duration = model.getScore().getDuration()
print 'duration =',duration
model.createCsoundScore('''
; SoundFonts
; to Chorus
i 1 0 0 100 200 0.2
; to Reverb
i 1 0 0 100 210 0.1
; to Output
i 1 0 0 100 220 1

; to Chorus
i 1 0 0 4 200 0.2
; to Reverb
i 1 0 0 4 210 0.1
; to Output
i 1 0 0 4 220 1

; to Chorus
i 1 0 0 5 200 0.2
; to Reverb
i 1 0 0 5 210 0.2
; to Output
i 1 0 0 5 220 1

; to Chorus
i 1 0 0 6 200 0.2
; to Reverb
i 1 0 0 6 210 0.2
; to Output
i 1 0 0 6 220 1

; to Chorus
i 1 0 0 7 200 0.2
; to Reverb
i 1 0 0 7 210 0.2
; to Output
i 1 0 0 7 220 1

; to Chorus
i 1 0 0 8 200 0.2
; to Reverb
i 1 0 0 8 210 0.2
; to Output
i 1 0 0 8 220 1

; to Chorus
i 1 0 0 9 200 0.2
; to Reverb
i 1 0 0 9 210 0.5
; to Output
i 1 0 0 9 220 1

; to Chorus
i 1 0 0 10 200 0.2
; to Reverb
i 1 0 0 10 210 0.2
; to Output
i 1 0 0 10 220 1

; to Chorus
```

```
i 1 0 0 11 200 0.2
; to Reverb
i 1 0 0 11 210 0.2
; to Output
i 1 0 0 11 220 1

; to Chorus
i 1 0 0 12 200 .2
; to Reverb
i 1 0 0 12 210 .1
; to Output
i 1 0 0 12 220 1

; to Chorus
i 1 0 0 13 200 .2
; to Reverb
i 1 0 0 13 210 .1
; to Output
i 1 0 0 13 220 1

; to Chorus
i 1 0 0 14 200 .2
; to Reverb
i 1 0 0 14 210 .1
; to Output
i 1 0 0 14 220 1

; to Chorus
i 1 0 0 15 200 .2
; to Reverb
i 1 0 0 15 210 .1
; to Output
i 1 0 0 15 220 1

; to Chorus
i 1 0 0 16 200 .2
; to Reverb
i 1 0 0 16 210 .1
; to Output
i 1 0 0 16 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 18 200 .2
; to Reverb
i 1 0 0 18 210 .1
; to Output
i 1 0 0 18 220 1

; to Chorus
i 1 0 0 19 200 .2
; to Reverb
i 1 0 0 19 210 .1
; to Output
i 1 0 0 19 220 1

; to Chorus
i 1 0 0 20 200 .2
; to Reverg
i 1 0 0 20 210 .1
; to Output
i 1 0 0 20 220 1

; to Chorus
i 1 0 0 21 200 .2
; to Reverb
i 1 0 0 21 210 .1
; to Output
i 1 0 0 21 220 1

; to Chorus
i 1 0 0 22 200 .2
; to Reverb
i 1 0 0 22 210 .1
; to Output
i 1 0 0 22 220 1

; Chorus to Reverb
i 1 0 0 200 210 0.05
; Chorus to Output
i 1 0 0 200 220 0.0
; Reverb to Output
i 1 0 0 210 220 0.4

; SoundFont outout.
i 100 0 [''' + str(duration) + '''+15] 80
; Chorus.
i 200 0 [''' + str(duration) + '''+15]    11     33
; Reverb.
i 210 0 [''' + str(duration) + '''+15]  0.95   0.8  15000
; Master output.
i 220 0 [''' + str(duration) + '''+15]    1      1
''')
csound.perform()
```

Listing 4: **Study A**

```
'''
GENERATING SCORES WITH O-L LINDENMAYER SYSTEMS
IN VOICE-LEADING ORBIFOLDS
Copyright (C) 2005 by Michael Gogins
All rights reserved.

See: http://ruccas.org/pub/Gogins/score_generation_in_voiceleading_orbifolds.pdf
'''
import CsoundVST
from Numeric import *
import math
import sys
import lindenmayer

print __doc__

filename = 'study.a'

lindenmayer = lindenmayer.ChordLindenmayer()
lindenmayer.axiom = '/4.0 A'
lindenmayer.generations = 3
lindenmayer.timeStep = 0.25
lindenmayer.rules['A'] = 'A +0,1 F -3,2 f +0,1 F +2,1 A [ f f F f f F *1.5 -3,0 F f f f A  F ] A'
lindenmayer.generate()

model = CsoundVST.MusicModel()
model.setCppSound(csound)

# Place the ChordLindenmayer node inside a Random node to randomize velocity and pan,
# place the Random node inside a Rescale node,
# and place the Rescale node inside the MusicModel.

random = CsoundVST.Random()
random.createDistribution("uniform_01")
random.setElement(6, 11, 1)
random.setElement(8, 11, 1)
rescale = CsoundVST.Rescale()
rescale.setRescale(CsoundVST.Event.INSTRUMENT, 1, 1,  0,       4)
rescale.setRescale(CsoundVST.Event.KEY,        1, 0, 42,       0)
rescale.setRescale(CsoundVST.Event.VELOCITY,   1, 1, 65,       6)
rescale.setRescale(CsoundVST.Event.PAN,        1, 1, -0.25,    1.5)
random.addChild(lindenmayer)
rescale.addChild(random)
model.addChild(rescale)
model.generate()
model.getScore().save(filename + '.mid')
instruments = [8,11,9,14,14]
for event in model.getScore():
    event.setInstrument(instruments[int(event.getInstrument())])
    print event.toString()


print 'Filename:', filename
model.setConformPitches(False)
model.setTonesPerOctave(12.0)
csound.load('c:/utah/home/mkg/projects/music/library/orchestra.csd')
csound.setCommand("csound -m7 -RWdfo " + filename + ".wav " + filename + ".orc " + filename + ".sco")
csound.setFilename(filename)
duration = model.getScore().getDuration()
print 'duration =',duration
model.createCsoundScore('''
; SoundFonts
; to Chorus
i 1 0 0 100 200 0.2
; to Reverb
i 1 0 0 100 210 0.1
; to Output
i 1 0 0 100 220 1

; to Chorus
i 1 0 0 4 200 0.2
; to Reverb
i 1 0 0 4 210 0.1
; to Output
i 1 0 0 4 220 1

; to Chorus
i 1 0 0 5 200 0.2
; to Reverb
i 1 0 0 5 210 0.2
; to Output
i 1 0 0 5 220 1

; to Chorus
i 1 0 0 6 200 0.2
; to Reverb
i 1 0 0 6 210 0.2
; to Output
i 1 0 0 6 220 1

; to Chorus
i 1 0 0 7 200 0.2
; to Reverb
i 1 0 0 7 210 0.2
; to Output
i 1 0 0 7 220 1

; to Chorus
i 1 0 0 8 200 0.2
; to Reverb
i 1 0 0 8 210 0.2
; to Output
i 1 0 0 8 220 1

; to Chorus
i 1 0 0 9 200 0.2
; to Reverb
i 1 0 0 9 210 0.5
; to Output
```

```
i 1 0 0 9 220 1

; to Chorus
i 1 0 0 10 200 0.2
; to Reverb
i 1 0 0 10 210 0.2
; to Output
i 1 0 0 10 220 1

; to Chorus
i 1 0 0 11 200 0.2
; to Reverb
i 1 0 0 11 210 0.2
; to Output
i 1 0 0 11 220 1

; to Chorus
i 1 0 0 12 200 .2
; to Reverb
i 1 0 0 12 210 .1
; to Output
i 1 0 0 12 220 1

; to Chorus
i 1 0 0 13 200 .2
; to Reverb
i 1 0 0 13 210 .1
; to Output
i 1 0 0 13 220 1

; to Chorus
i 1 0 0 14 200 .2
; to Reverb
i 1 0 0 14 210 .1
; to Output
i 1 0 0 14 220 1

; to Chorus
i 1 0 0 15 200 .2
; to Reverb
i 1 0 0 15 210 .1
; to Output
i 1 0 0 15 220 1

; to Chorus
i 1 0 0 16 200 .2
; to Reverb
i 1 0 0 16 210 .1
; to Output
i 1 0 0 16 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 18 200 .2
; to Reverb
i 1 0 0 18 210 .1
; to Output
i 1 0 0 18 220 1

; to Chorus
i 1 0 0 19 200 .2
; to Reverb
i 1 0 0 19 210 .1
; to Output
i 1 0 0 19 220 1

; to Chorus
i 1 0 0 20 200 .2
; to Reverg
i 1 0 0 20 210 .1
; to Output
i 1 0 0 20 220 1

; to Chorus
i 1 0 0 21 200 .2
; to Reverb
i 1 0 0 21 210 .1
; to Output
i 1 0 0 21 220 1

; to Chorus
i 1 0 0 22 200 .2
; to Reverb
i 1 0 0 22 210 .1
; to Output
i 1 0 0 22 220 1

; Chorus to Reverb
i 1 0 0 200 210 0.05
; Chorus to Output
i 1 0 0 200 220 0.0
; Reverb to Output
i 1 0 0 210 220 0.4

; SoundFont outout.
i 100 0 [''' + str(duration) + '''+15] 80
; Chorus.
i 200 0 [''' + str(duration) + '''+15]    11     33
```

```
; Reverb.
i 210 0 [''' + str(duration) + '''+15]  0.97   0.8  15000
; Master output.
i 220 0 [''' + str(duration) + '''+15]    1     1
''')
csound.perform()
```

Listing 5: **Study B**

```
'''
GENERATING SCORES WITH O-L LINDENMAYER SYSTEMS
IN VOICE-LEADING ORBIFOLDS
Copyright (C) 2005 by Michael Gogins
All rights reserved.

See: http://ruccas.org/pub/Gogins/score_generation_in_voiceleading_orbifolds.pdf
'''
import CsoundVST
from Numeric import *
import math
import sys
import lindenmayer

print __doc__

filename = 'study.b'

lindenmayer = lindenmayer.ChordLindenmayer()
lindenmayer.range = 48
lindenmayer.axiom = '/6.0 A'
lindenmayer.generations = 3
lindenmayer.timeStep = 0.16125
lindenmayer.rules['A'] = 'A +0,1 F -3,2 f +0,1 f f +2,1 f f A [ mT1.5 F mt8,1.5 F mt9,1.5 F mt10,1.5 mt11,1.5 C C f F F *1.5 -3,2 f A
      F dT1.5 at4,1 at5,1 at6,1 at7,1 f f F f f F *2.25 -3,0 f +1,2 A F A F ] A'
lindenmayer.generate()

model = CsoundVST.MusicModel()
model.setCppSound(csound)

# Place the ChordLindenmayer node inside a Random node to randomize velocity and pan,
# place the Random node inside a Rescale node,
# and place the Rescale node inside the MusicModel.

random = CsoundVST.Random()
random.createDistribution("uniform_01")
random.setElement(6, 11, 1)
random.setElement(8, 11, 1)
rescale = CsoundVST.Rescale()
rescale.setRescale(CsoundVST.Event.INSTRUMENT, 1, 0,  2,       0)
rescale.setRescale(CsoundVST.Event.KEY,        1, 0, 36,       0)
rescale.setRescale(CsoundVST.Event.VELOCITY,   1, 1, 70,      12)
rescale.setRescale(CsoundVST.Event.PAN,        1, 1, -0.25,    1.5)
random.addChild(lindenmayer)
rescale.addChild(random)
model.addChild(rescale)
model.generate()
model.getScore().save(filename + '.mid')
instruments = [8,11,9,14,14]
for event in model.getScore():
    #event.setInstrument(instruments[int(event.getInstrument())])
    print event.toString()

print 'Filename:', filename
model.setConformPitches(False)
model.setTonesPerOctave(12.0)
csound.load('c:/utah/home/mkg/projects/music/library/orchestra.csd')
csound.setCommand("csound -m7 -RWdfo " + filename + ".wav " + filename + ".orc " + filename + ".sco")
csound.setFilename(filename)
duration = model.getScore().getDuration()
print 'duration =',duration
model.createCsoundScore('''
; SoundFonts
; to Chorus
i 1 0 0 100 200 0.2
; to Reverb
i 1 0 0 100 210 0.1
; to Output
i 1 0 0 100 220 1

; to Chorus
i 1 0 0 4 200 0.2
; to Reverb
i 1 0 0 4 210 0.1
; to Output
i 1 0 0 4 220 1

; to Chorus
i 1 0 0 5 200 0.2
; to Reverb
i 1 0 0 5 210 0.2
; to Output
i 1 0 0 5 220 1

; to Chorus
i 1 0 0 6 200 0.2
; to Reverb
i 1 0 0 6 210 0.2
; to Output
i 1 0 0 6 220 1

; to Chorus
i 1 0 0 7 200 0.2
; to Reverb
i 1 0 0 7 210 0.2
; to Output
i 1 0 0 7 220 1

; to Chorus
i 1 0 0 8 200 0.2
```

```
; to Reverb
i 1 0 0 8 210 0.2
; to Output
i 1 0 0 8 220 1

; to Chorus
i 1 0 0 9 200 0.2
; to Reverb
i 1 0 0 9 210 0.5
; to Output
i 1 0 0 9 220 1

; to Chorus
i 1 0 0 10 200 0.2
; to Reverb
i 1 0 0 10 210 0.2
; to Output
i 1 0 0 10 220 1

; to Chorus
i 1 0 0 11 200 0.2
; to Reverb
i 1 0 0 11 210 0.2
; to Output
i 1 0 0 11 220 1

; to Chorus
i 1 0 0 12 200 .2
; to Reverb
i 1 0 0 12 210 .1
; to Output
i 1 0 0 12 220 1

; to Chorus
i 1 0 0 13 200 .2
; to Reverb
i 1 0 0 13 210 .1
; to Output
i 1 0 0 13 220 1

; to Chorus
i 1 0 0 14 200 .2
; to Reverb
i 1 0 0 14 210 .1
; to Output
i 1 0 0 14 220 1

; to Chorus
i 1 0 0 15 200 .2
; to Reverb
i 1 0 0 15 210 .1
; to Output
i 1 0 0 15 220 1

; to Chorus
i 1 0 0 16 200 .2
; to Reverb
i 1 0 0 16 210 .1
; to Output
i 1 0 0 16 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 17 200 .2
; to Reverb
i 1 0 0 17 210 .1
; to Output
i 1 0 0 17 220 1

; to Chorus
i 1 0 0 18 200 .2
; to Reverb
i 1 0 0 18 210 .1
; to Output
i 1 0 0 18 220 1

; to Chorus
i 1 0 0 19 200 .2
; to Reverb
i 1 0 0 19 210 .1
; to Output
i 1 0 0 19 220 1

; to Chorus
i 1 0 0 20 200 .2
; to Reverg
i 1 0 0 20 210 .1
; to Output
i 1 0 0 20 220 1

; to Chorus
i 1 0 0 21 200 .2
; to Reverb
i 1 0 0 21 210 .1
; to Output
i 1 0 0 21 220 1

; to Chorus
i 1 0 0 22 200 .2
; to Reverb
i 1 0 0 22 210 .1
; to Output
i 1 0 0 22 220 1

; Chorus to Reverb
```

```
i 1 0 0 200 210 0.05
; Chorus to Output
i 1 0 0 200 220 0.0
; Reverb to Output
i 1 0 0 210 220 0.4

; SoundFont outout.
i 100 0 [''' + str(duration) + '''+15] 70
; Chorus.
i 200 0 [''' + str(duration) + '''+15]    11     33
; Reverb.
i 210 0 [''' + str(duration) + '''+15]  0.92   0.5  15000
; Master output.
i 220 0 [''' + str(duration) + '''+15]    1     1
''')
csound.perform()
```

References

[1] Lejaren Hiller and L.M. Isaacson, editors. *Experimental Music: Composition with an Electronic Computer.* McGraw–Hill, New York, New York, 1959. 1

[2] Kristine H. Burns. *The History and Development of Algorithms in Music Composition, 1957-1993.* PhD thesis, Ball State University, 1993. 1

[3] Justin London. Some non–isomorphisms between pitch and time, april 2001. http://www.people.carleton.edu/~jlondon/some_non-isomorphisms.htm. 2

[4] Dmitry Tymoczko. The geometry of musical chords, 2005. http://music.princeton.edu/~dmitri/voiceleading.pdf. 2

[5] Przemyslaw Prusinkiewicz and Artistid Lindenmayer. *The Algorithmic Beauty of Plants.* Spring-Verlag, New York, 1996 [1991]. Available online at http://algorithmicbotany.org/papers. 2, 3

[6] S. R. Holtzman. Using generative grammars for music composition. *Computer Music Journal*, 5(1):51–64, 1981. 3

[7] Michael Gogins. Fractal music with string rewriting grammars. *News of Music*, 13:146–170, Winter 1992. 3

[8] Jon McCormack. Grammar based music composition. In R. Stocker et al., editor, *From Local Interactions to Global Phenomena*, Complex Systems 96, Amsterdam, 1996. ISO Press. 3

[9] Guido van Rossum. Python. http://www.python.org. 3

[10] Travis Oliphaunt, Pearu Peterson, and Eric Jones at al. Scipy — scientific tools for python, 2005. http://www.scipy.org. 3

[11] Richard Boulanger. csounds.com... almost everything csound. http://www.csounds.com. 3

[12] David M. Beazley et al. Welcome to swig. http://www.swig.org. 4

[13] Michael F. Barnsley. *Fractals Everywhere.* Academic Press Professional, Boston, 2nd edition, 1993 [1988]. 6

[14] Michael Gogins. Iterated function systems music. *Computer Music Journal*, 15(1):34–42, Spring 1991. 6

[15] Michael Gogins. ...how i became obsessed with finding a mandelbrot set for sounds. *News of Music*, 13:129–139, Winter 1992. 6