Geometricity and Musical Livecoding:
There are many ways to approach musical livecoding, and music-making in general. This submission discusses the notion of geometricity in the context of algorithmic composition. Geometric composition is an approach to music-making, but it reflects a perspective which strives to challenge the nature of goal-oriented art. In this sense, the use of geometric composition is a social commentary as well as a description of an artistic medium.

Music and Livecoding:

Binary polarization is often apparent in the referential language used when describing or reading about how music is made. Typically this refers to the notion that music should either be considered linearly (as in start:finish) or in a cyclical way (as in time-is-a-neverending-cycle-of-outwardly-spiraling-polyrhythms). Geometricity as such requires the perfomer-composer to think about a piece of music as a complete entity from the very beginning, instead of as a gradual aggregation of parts which eventually assembles as a whole. This process is naturally attainable and easily accessed through musical livecoding, and likely also through other forms of media not discussed here.

The aforementioned musical geometricity is based on the concept of a composition’s timeline as being a “unit length”, and each unique sound-type as a “channel”. As such, any given channel has an algorithm that describes its relationship to the entirety of the composition. Musical parameters for each channel can be assigned static values, or can vary (linearly, exponentially, stepwise etc.) between two or more values for a set duration. In this way, musical livecoding enables a strong intermixing of both *analog* (continuous) and *digital* (discrete) approaches to music-making.

In many types of digital music-making, sound characteristics are often masked inside a “black box”, where one doesn’t really know the exact [numerical] value of the parameter being modified. This is at times debilitating to creative processes. Depending on the piece, one might want to fine-tune or even drastically alter something in the music very precisely. Alas, the limitations of drag-and-drop VSTs and manipulating GUI knobs with a mouse continue to confound the user. But wait! Enter livecoding. With a clack of fingers upon keyboard, the algorithm is modified, and the programmer-performer can continue to tweak the channels like the magician they are, or sit back and enjoy the ride.

My process:
For this composition (created using FoxDot and SuperCollider), each channel is initialized with a native sound-type (e.g. a pulse or bass drum) that repeats at a regular interval. Parameters and values (chop, LPF, sustain, duration, &.c) are gradually introduced and altered while simultaneously trying to balance the frequencies and overall volume mix. This is done by comparing and re-comparing channels against each other (live monitoring) throughout the piece. During the performance, channels are modified on-the-fly by adding or deleting code and recompiling the channel, which allows for an active improvisational experience. Greater musicality can emerge when the player has a sense of how each change will affect the music. Since livecoding means manually modifying lines of code, both an experimental approach and a technical knowledge of audio programming, or DSP, can reinforce each other.

Geometricity:
So, what do I mean by geometricity? Geometricity (as used in this discussion about musical composition) is inherently a departure from the start-to-finish mindeset of linearity, and creates music which is indeed polyrhythmic, but polyrhythmic subdivision is not necessarily the focus of the task. Instead the idea is, in a sense, to superimpose squares onto triangles and circles onto hexagons while embedding them within each other. (These shapes refer to parameter values and the ways in which they geometrically relate to each other, such as in Euclidean division, scalar relationships, or indeed a 4-over-3 polyrhythm. Please refer to source code for examples.) We can think of these nested tesselations or musical textures as waves

rippling across technicolor clouds under the ocean in outer space.

In addition to being able to modify single lines of code, and therefore channels, geometricity can extend to the idea of modifying entire sets of parameters at once based upon their *parameter class*. By parameter class, I mean that instead of changing each note value or pan/volume/delay value channel-by-channel, we can be thinking of changing groups of parameter values and recompiling those instead of single channels. For example, if I want to modify the amplitudes of some channels in the piece, I go to my amplification array and raise some values, but then I need to lower some values to make sure the overall sound is balanced. (Doing this line-by-line is tedious and takes a long time, and can involve way too much mouse clicking for my personal taste, depending upon the organization of the code.)

In this composition, the values have been manually modified in the improvisational composition process. On the backend in the Atom editor for FoxDot, there is a set of files containing arrays which allow for quick manipulation of certain parameter groupings: volume, delay, and so on. The code for that particular feature is a work-in-progress and not the focus of this submission, though it is an extension of the thought processes outlined here.

Goal-oriented art and its subversion:
The subversion of goal-oriented art may lie in the use of techniques and tools themselves as creative media. Instead of having a product in mind, I argue for allowing the process of using techniques to explore possibilities as the guiding principle which shapes the end result. These can be mental, emotional, psychological, or physical techniques. Similarly to how a ceramicist uses clay manipulation techniques and social commentary or personal philosophy to guide the creation of a sculpture, text-based audio programming, an appreciation of sound, and a penchance for social justice can be used in an exploratory manner to guide the creation of sound art.

In this piece, the genre is time-based electronic music, sometimes including electronica or techno (but not necessarily excluding other similar genres). While working with this medium, I've tried many different ways to algorithmically relate independent audio channels and variables to get the 'right type' of musical results. In this case the purpose is to create a danceable piece of music that sounds cool and which has compositional principles of ebb and flow. These factors in music are usually specific to an artist's take on "how music should sound". This includes timing the entry and removal of certain sound components so as to not sound "too cluttered" or "too sparse". The idea is to capture the attention of a listener while still providing a danceable beat without being too distracting.

Goal-oriented art strives to become something. In this case, the music strives to become electronica. However, rather than try to emulate the sounds of electronica which I already know to exist, my approach has been to turn to the algorithms themselves for guidance. Experimental manipulation of parameters yields music which is at times melodic, at times glitchy, and at times raw. A geometric subdivision of signals creates the nuances of each channel, and through trial and error, a sort of livecoded-Rubik's-cube emerges.

SOURCE CODE FOR KOMBU (FoxDot)

```
kombu.py

Clock.bpm=129.3
Scale.default = "yu"
```

```
va >>
play("v",amp=0.6,amplify=0.7,dur=var([1.5,1,2,2],8),lpf=expvar([1
00,500],54),shape=0.15,room=var([0,0.1,0.2,0.3],16),pan=0.3)
oo >> play("X
",amp=0.6,amplify=0.8,room=0.05,delay=0.5,lpf=var([8000,10000],32
),echo=1,rate=linvar([0.1,1],64),fmod=linvar([-
10,10],8),pan=PSine(3))
bu >>
play("v",amp=0.6,amplify=0.5,chop=var([0,1],16),dur=var([1,1/2],1
6),rate=expvar([0.7,3],32),pan=-0.7)
xy >>
play("x",amp=var([0,0.6],8),amplify=Pvar([0.9,0.8,0.7,0.6],32),du
r=var([1/2,PDur(1,4)],32),chop=var([4,2,3,1],8),pan=-
0.3,shape=0.05,rate=linvar([1.2,0.8],8)).every(8,'stutter')

bv >> play(" v",amp=0.5,dur=2)

qa >>
pulse(amp=var([0.5,0],32),sus=1/2,dur=var([1/2,1/3,1.5],8),echo=v
ar([0.2,0.4],[24,8]),formant=0.5,bend=0.0005,pan=-0.8)
qb >>
pulse(var([3,1,2,4],64),amp=[1,0,0,1],amplify=0.7,sus=1/2,echo=0.
5,bend=0.02,formant=0.5,chop=2,pan=0.8)
qd >> pulse([0,2,4,6,8,10,12],amp=var([0,expvar([0.5,0.8],28)],
[56,28]),amplify=1,dur=1/4,lpf=3000,pan=PSine(4.7)).every(7,'reve
rse')
qd.stop()
qe >> pulse(var([3,2],32),amp=var([0.5,0],
[16,48]),dur=var([1/2,1],
[16,16]),rate=1,pan=PSine(4.3),lpf=linvar([1000,8000],32)).every(
8,'stutter')
qc >>
pulse(var([0,1,4,7,4,3,4,2],64),amp=var([1,0,0,1],128),amplify=0.
8,sus=1/2,dur=var([2/3,1/2],16),chop=var([1,4,2],
[8,24,32]),bend=0.02,formant=0.5,echo=0,delay=0,rate=var([0.5,1],
16))

#percs
ub >>
play("-",amp=0.6,dur=1,delay=0.5,sample=[3,7,1,4],rate=var([0.3,0
.5],32),pan=-0.2)
mz >>
donk(amp=var([var([[1,0,0,1],0]),0],64),amplify=0.5,echo=1/4,rate
=0.3,pan=0.2).every(8,'stutter')
va >>
play("o",amp=var([0,0.2],128),dur=1/2,delay=0,sample=[3,7,1,4],di
st=0.2).every(6,'stutter')
```

```
vb >>
play("-",amp=var([0.6,0],64),dur=1,delay=0.5,sample=[3,7,1,4]).ev
ery(5,'stutter')
vc >>
play("y",dur=2,delay=1/4,amp=var([1,0],16)).every(4,'stutter')
vx >> play("y",dur=1/2,delay=1/2,amplify=0.6,amp=var([1,0],
[4,4,4,4,2,6,2,6]),sample=([1,2,3,4,5,6,7,8])).every(3,'stutter')
xz >>
play("-",dur=3/2,amp=[0,0,1,0],delay=var([0.5,0.005],64),amplify=
var([3/4,2/3,7/8],4),chop=var([0,2,5,1,3,4],4),sample=var([1,4,5,
8,9],4))
xx >>
play("h",amp=[0,0.7],amplify=Pvar([1,0.8,0.6,0.4],4),delay=0.5,ro
om=0.1,lpf=expvar([6000,15000],16))
wu >>
play("h",amp=0.5,amplify=Pvar([1,0.8,0.6,0.4],4),delay=0,room=0.2
,echo=0.5)
vc.stop()

#bass
wx >>
sawbass([0],amp=var([0.8,0.6,0.5,0.7]),amplify=1.5,rate=0.5,dur=v
ar([1/2,1],32),chop=var([0,1,2],16),pan=PSine(2.5),lpf=linvar([30
00,9000],128))
wx >> sawbass([0],amp=0.8,amplify=1.5,dur=var([1,2],
[24,8]),chop=var([0,1,2,3],16),pan=PSine(2.5),lpf=7000)
wx >> sawbass(var([0,-2],
[56,8]),amp=1.5,amplify=1,dur=1/2,chop=Pvar([2,1,0,1,2,0,2,0],
[12,4,14,2]),pan=PSine(2.5))
wx >> sawbass([-5,-4,-3,-
2],amplify=1.5,dur=2,chop=var([0,1,2,3],8),pan=PSine(3)).every(16
,'mirror')
wx >> sawbass(var([0,[-1,1],2,
[3,4]],16),amplify=1.1,dur=2,chop=5,pan=PSine(3))
wx >> sawbass(Pvar([0,1,4,3,2],
[8,8,8,4,4]),amplify=0.8,dur=1/2,sus=1/2,chop=0,pan=PSine(3))

#sounds
p1 >> noise(dur=12/3,amp=var([0.3,0],128),amplify=Pvar([0.5,0],
[6,2]),drive=0.1,pan=PSine(5)).every(16,'stutter')
r0 >> rave(amp=var([0.8,0],64),var([0,1,2],
[16,8,8]),amplify=0.7,chop=Pvar([0,1,2,3],2),dur=1,sus=1.1,spin=0
.3,pan=[-0.8,0.8,-0.3,0.3])
r1 >>
rave(amp=var([0,0.8],64),amplify=0.7,chop=Pvar([0,1,2,3],2),dur=1
,sus=1.1,spin=0.3,pan=[-0.8,0.8,-0.3,0.3])
```

```
#melosynths
melo=
lx >>
spark(dur=1/2,chop=2,rate=1,dist=0.2,lpf=var([linvar([800,8000],3
2),0],64))
ly >>
pluck(amp=0.4,amplify=0.5,dur=1/2,chop=2,delay=0.5,drive=0.3,pan=
PSine(3))
lz >>
creep(amp=0.6,dur=Pvar([3,1,4,2,0,0,0,0],8),sus=Pvar([3,1,2,4],4)
,delay=0.5,lpf=expvar([9000,100],32))
mx >> space(amp=Pvar([1,0,0,1],[4,6,2,4,8,8,2,14,4,4,8]))
my >> ambi(Pvar([1,-
1,0,0],8),amp=Pvar([1,0],16),sus=1.2,delay=0.5,room=0.4,echo=0.2)
nx >> sitar(var([0,1,2,-
1],8),delay=var([2/3,1/2],32),amp=Pvar([0,1],
[32,32,24,8]),echo=0.2,room=0.3,dist=0.1).every(4,'stutter')
ny >> marimba(delay=0.5)
nz >> klank(var([0,2],30,2),dur=3/2)
ox >> pads(amp=var([1,0,1,0,0,0,1,1],16))
mx.stop()
```