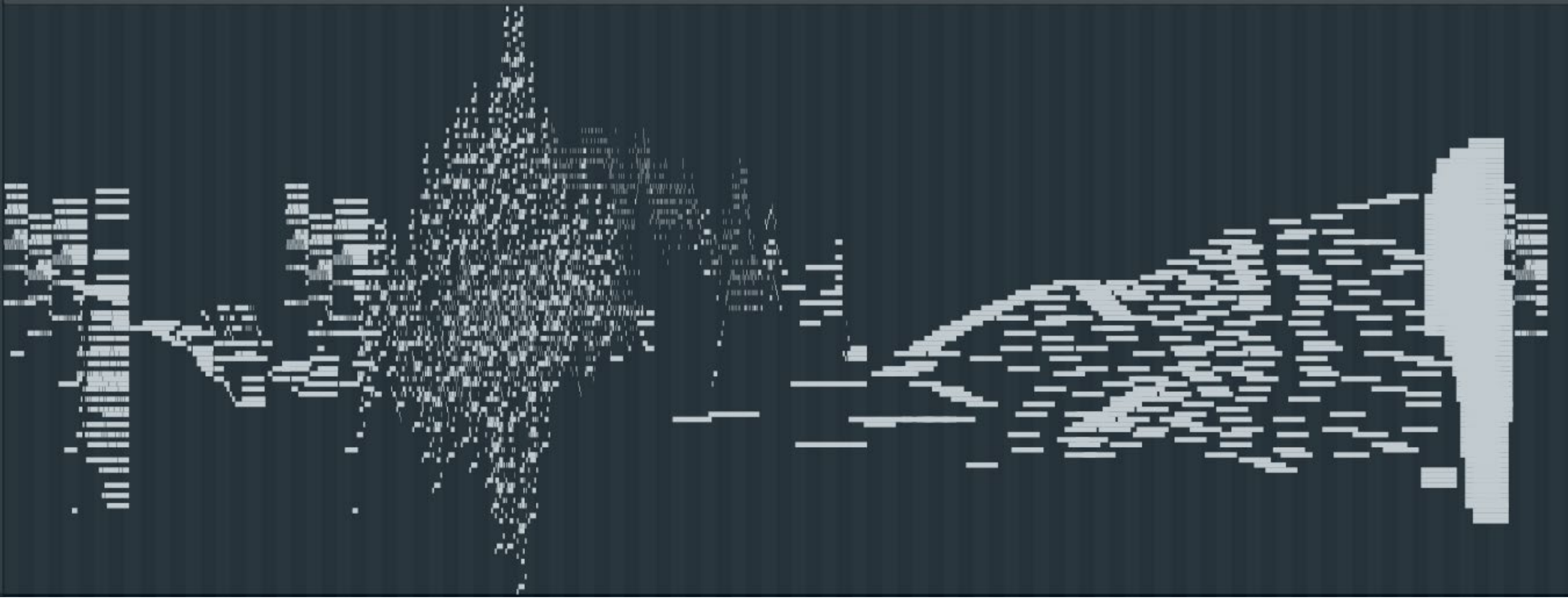
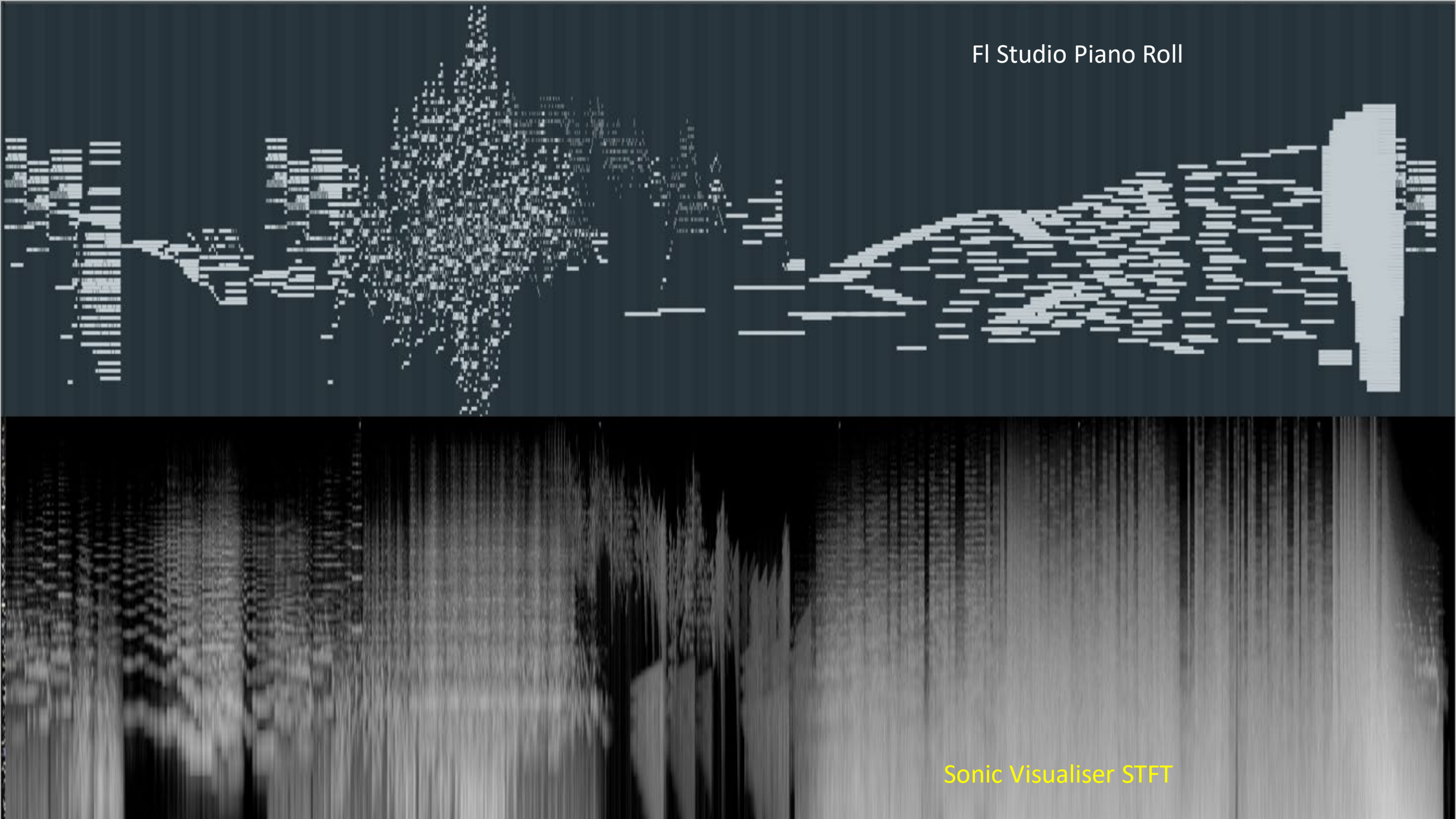
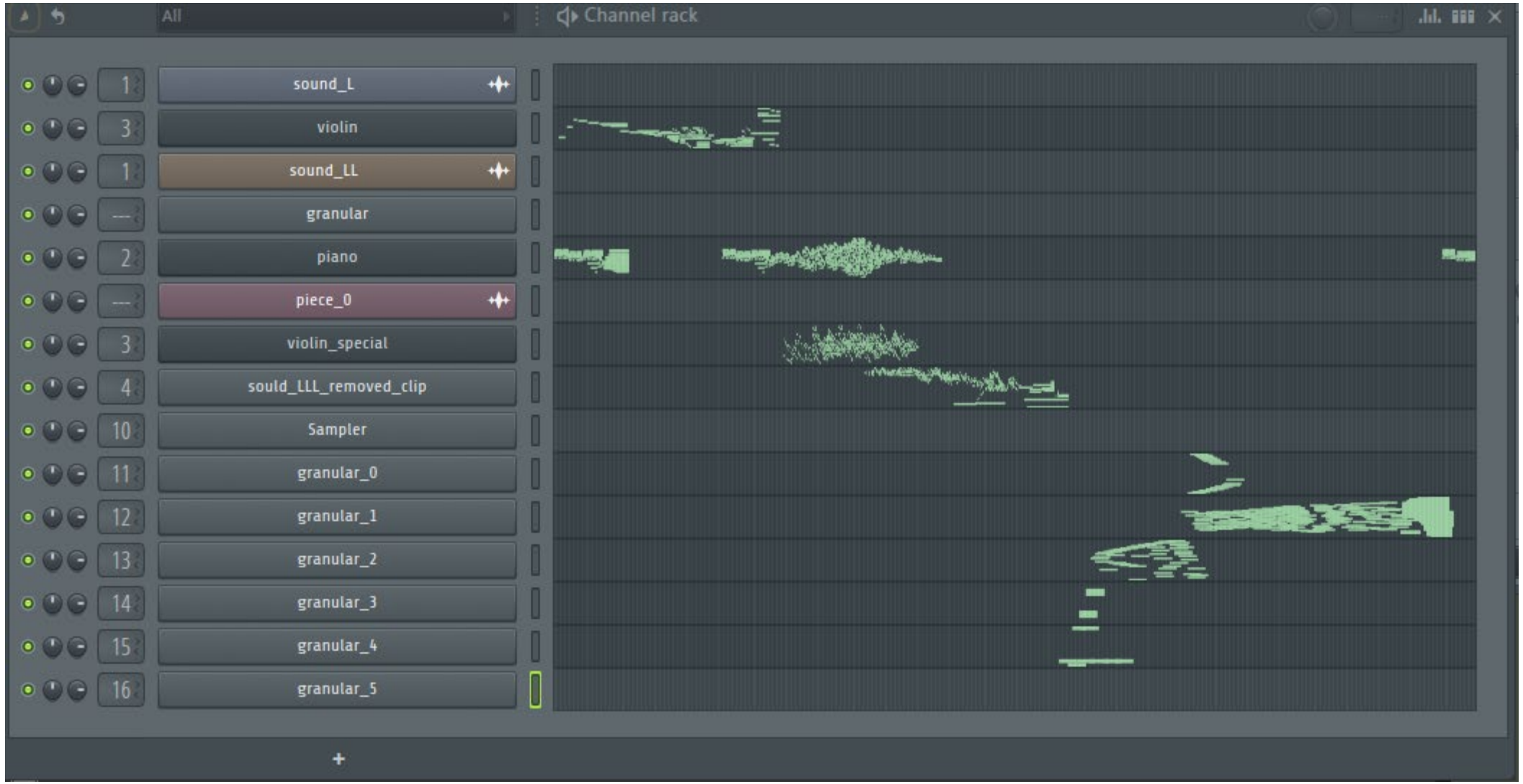


Evolved

Benny Zhang

Symbolic & Audio view

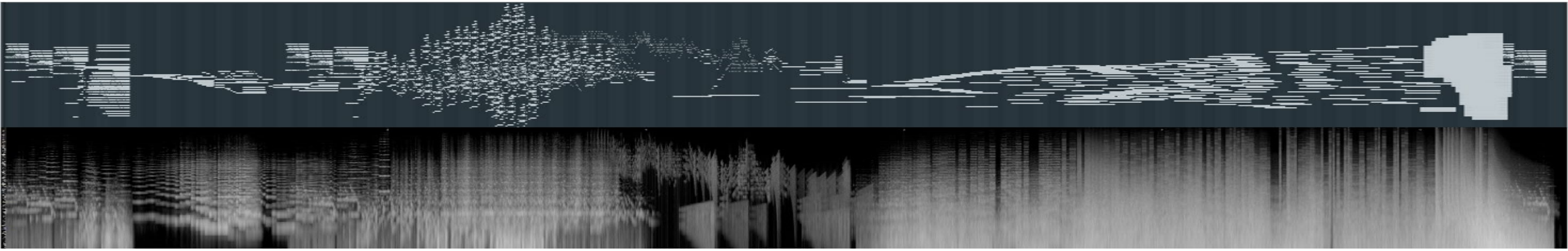

Fl Studio Piano Roll
Sonic Visualiser STFT


All
Channel rack
1
sound_L
3
violin
1
sound_LL
granular
2
piano
piece_0
3
violin_special
4
sould_LLL_removed_clip
10
Sampler
11
granular_0
12
granular_1
13
granular_2
14
granular_3
15
granular_4
16
granular_5

Abstract

- Evolved is a piece exploring the idea of being ordered to non-ordered in different dimensions. The philosophy behind this work expresses the idea of surrealism. From the tonality perspective, it starts from ordered impressionistic ensembles music to chaotic re-sampling of the impressionistic part. From the Symbolic-Spectral perspective, it starts with mainly sampled acoustic sounds to artificially generated spectrum music. From the POV of Information theory, it evolves from the most ordered state to the most chaotic states. From the cognitive neuroscience perspectives, It serves as representation the clear semantic representation at the early age of brain to a dying chaotic brain potentially suffered from Alzheimer disease.

Evolution Logistics



Time

Tonal	→	Atonal
Sparse	→	Dense
Symbolics-Oriented	→	Spectrum-Oriented
low Information Entropy	→	high Information Entropy
Cognitive Functional (Normal brain)	→	Cognitive Dysfunctional (Neuro-degenerated brain)

Keywords

- Improvisation
- Non-tertian Harmony
- Sampling
- Additive Synthesis
- Granular Synthesis
- Spectralism
- Sound Mass
- Midi Sequencing
- Music Information Theory

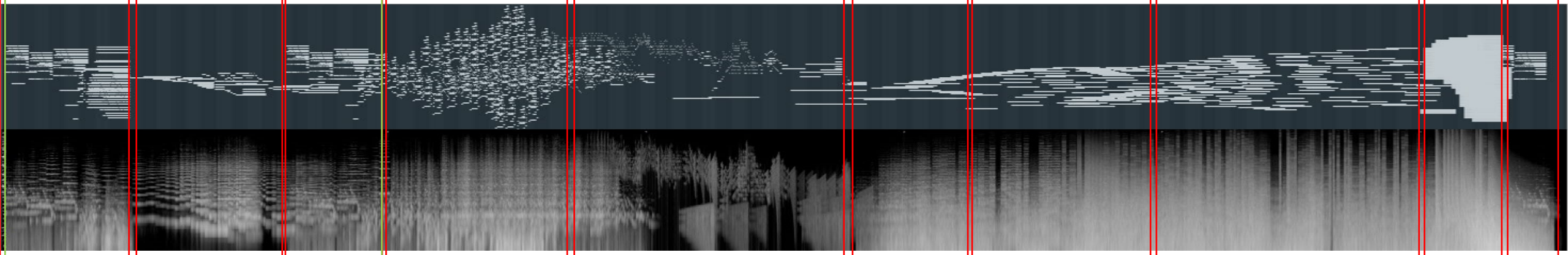
Sound Source used

- Instruments & Effectors:
  - Acoustics:
    - Grand Piano (Kontakt NI Grandeur)
    - Violin (Kontakt Chris hein Violin – Italian violin Clean Start)
  - Synthesized:
    - Sine Wave Pattern Generator (Python DIY)
    - Granular Synthesizer (Python DIY)
  - Sample used:
    - The {A + B + A'} part for granular synthesis
    - Then load it to fl sampler for midi sequence with different playback speeds
  - Effectors:
    - Fl Reverb 2
    - Fl Delay 2
    - (Mastering)Waves – C6 Stereo

Music Form & Techniques Detail

Saved as Grains.wav for later sections' granular synthesis

A → B → A' → C → D → E' → E'' → E''' → E'''' → A''



Instrumentation:
Piano + Long Expressive Chris hein violin

Production Process:
This section is majorly created by the improvisation on keyboard.

For the **piano** part improvisation logic, first I create a bar of music by playing a polychord in the C phrygian mode. And then I select the midi notes I improvise and then modulate to other starting pitch by copy-paste.

It is the same improvisational logic for the **violin** part. When I write long melody line, I prevent any appearance of cyclic rhythm or obvious pattern in melodic interval.

This part serves as a **tonal introduction** of the piece, and it provides a comparison baseline for the later sound mass section.

(For the mixing process, I just assign each instrument to a single bus, and then put a single default setting digital reverb to make it sounds good. Then a FL compressor called Soundgoodizer is applied.

Instrumentation:
Piano + Long Expressive Chris hein violin

Production Process:
This section is majorly created by the improvisation on keyboard.

I improvise the semi-monophonic voice and the non-tertian harmony in this **Strings** part. When I write long melody line, I prevent any appearance of cyclic rhythm or obvious pattern in melodic interval. When I improvise non-tertian harmony, I try to place them into close voicing as possible.

This part serves as a **tonal cinematic continuation** of the piece, and it provides a comparison baseline for the later sound mass section.

(For the mixing process, I just assign each instrument to a single bus, and then put a single default setting digital reverb to make it sounds good. No compression is being added for violin.)

Instrumentation:
Piano + Long Expressive Chris hein violin

Production Process:

This section serves for **recapitulation** purpose.

The piano in this part is majorly copy paste the fragment from the A section, with same midi-sequencing modulation technique.

I also improvise on the violins in this part, just to make the music not that boring.

This part serves as the last part of **tonal cinematic development** of the piece, and it provides a comparison baseline for the later sound mass section.

(For the mixing process, I just assign each instrument to a single bus, and then put a single default setting digital reverb to make it sounds good. No compression is being added for violin.)

Instrumentation:
Piano + Ricochet Chris hein violin

Production Process:

This section serves for **recapitulation** purpose.

The piano in this part is majorly copy paste the fragment from the A section, with same midi-sequencing modulation technique.

I also improvise on the violins in this part, just to make the music not that boring.

This part serves as the last part of **tonal cinematic development** of the piece, and it provides a comparison baseline for the later sound mass section.

(For the mixing process, I just assign each instrument to a single bus, and then put a single default setting digital reverb to make it sounds good. No compression is being added for violin.)

Instrumentation:
Sine Wave Addition Synth (Python)

Production Process:

This section serves for **transition from acoustic sound to surreal sound.**

This section only uses the sinewave Triangle pattern I created in python by simple DSP frequency shifting method, which is '3 vertically arranged triangle-shaped sound' in Short Time Fourier Transform View mode.

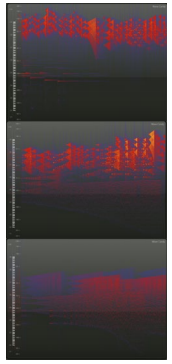
Then I feed the exported waveform from python and put that into a FL sampler with no ADSR applied.

Then I improvise the triangle sound in high register on the midi keyboard I have, try to see how interesting the sound might be.

It serves as the first atonal exploration with surreal synthesized sound in the piece.

(For the mixing process, I assign this effectors plugins in sequential wiring:
FL Reverb2 → FL Reverb2 → FL delay 2

This serves to spatialize the sound synths sound and make spatially consistent to the previous acoustic section.)



Instrumentation:
Granular synth with Grains.wav

```
# Params: source, source_start, source_end, layer_num, grain_size_range, playback_speed_range, length,sr
o0 = benny_granular_synth(y.T, 0, 60, 10, (20,40), (1,1.1), length=6,sr=44100)
o1 = benny_granular_synth(y.T, 0, 60, 10, (200, 400), (1,1.1), length=6,sr=44100)
o2 = benny_granular_synth(y.T, 0, 60, 10, (2000, 4000), (1,1.1), length=6,sr=44100)
o3 = benny_granular_synth(y.T, 0, 60, 10, (4000, 6000), (1,1.1), length=6,sr=44100)
o4 = benny_granular_synth(y.T, 0, 60, 10, (6000, 8000), (1,1.1), length=6,sr=44100)
o5 = benny_granular_synth(y.T, 0, 60, 10, (20000, 30000), (1,1.1), length=6,sr=44100)
```

Production Process:

This section serves for **the introduction stage of granular synthesis.**

This section only uses the Grains.wav as input, and then we select the first 60 seconds of the sound with 10 layering of additive process for output 6 seconds. The grains are randomly selected from any (6000, 8000) or (4000, 6000) intervals and then playback through the midi sequencer.

The point should be noticed here is that, I want to pertain of feeling of tonal ordered in this introduction stage by making the grains size very large. Because doing so may maintain some spectral contour of the previous tonal sections.

For the sequencing, improvisation again.

(For the mixing process, I just add a reverb on it)

Instrumentation:
Granular synth with Grains.wav

Production Process:

This section serves for **the first development stage of granular synthesis.**

This section only uses the Grains.wav as input, and then we select the first 60 seconds of the sound with 10 layering of additive process for output 6 seconds. The grains are randomly selected from any (6000, 8000) or (4000, 6000)or (2000,4000) intervals and then playback through the midi sequencer.

The addition of (2000,4000) grained wave are serves for adding more abstract and surreal sound into the piece, and foreshadow the late coming pure spectral art.

For the sequencing, improvisation again.

(For the mixing process, I just add a reverb on it)

Instrumentation:
Granular synth with Grains.wav

```
# Params: source, source_start, source_end, layer_num, grain_size_range, playback_speed_range, length,sr
o0 = benny_granular_synth(y.T, 0, 60, 10, (20,40), (1,1.1), length=6,sr=44100)
o1 = benny_granular_synth(y.T, 0, 60, 10, (200, 400), (1,1.1), length=6,sr=44100)
o2 = benny_granular_synth(y.T, 0, 60, 10, (2000, 4000), (1,1.1), length=6,sr=44100)
o3 = benny_granular_synth(y.T, 0, 60, 10, (4000, 6000), (1,1.1), length=6,sr=44100)
o4 = benny_granular_synth(y.T, 0, 60, 10, (6000, 8000), (1,1.1), length=6,sr=44100)
o5 = benny_granular_synth(y.T, 0, 60, 10, (20000, 30000), (1,1.1), length=6,sr=44100)
```

Production Process:

This section serves for **the second development stage of granular synthesis.**

This section only uses the Grains.wav as input, and then we select the first 60 seconds of the sound with 10 layering of additive process for output 6 seconds. The grains are randomly selected from any (6000, 8000) or (4000, 6000)or (2000,4000) or (200,400) or (20,40) intervals and then playback through the midi sequencer.

The addition logic is same here for (200,400) and (20,40), just make it sounds more 'low information entropy'.

(The concept of information entropy resulting a genre called chaos theory)

For the sequencing, improvisation again.

(For the mixing process, I just add a reverb on it)

Instrumentation:
Granular synth with Grains.wav

Production Process:

This section serves for **the climax of the granular synthesis.**

At this stage, I only played the (20,40) with different pitches **simultaneously. This may be heard as a filter shifting effect due to the different playback rate of the same sample.**

For the sequencing, improvisation again.

(For the mixing process, I just add a reverb on it)

Instrumentation:
Piano

Production Process:

Just copy paste from Section A.

It serves as a coda and makes listener spirit goes back to the conscious world.

DIY Sinewave Chirp Generator (Triangle waveform)(Function)

```
def benny_trans(sr, bus, start_time, end_time, start_freq, end_freq, start_amp, end_amp, f_linearity, a_lineraity):
    '''
    This function is a chirp with different phrase.

    amplitude is normalized

    start_time is absolute second
    '''
    amp = int(np.iinfo(np.int16).max)

    # frequency
    temp_bus = []
    for i in range(1, (end_time - start_time) * sr + 1):
        frequency = start_freq + (end_freq - start_freq) * (i/(sr * (end_time - start_time)))
        temp_bus.append(math.sin((2. * math.pi * (i/sr) * frequency)))

    # amplitude
    for i in range(len(temp_bus)):
        amplitude = amp * (start_amp + (end_amp - start_amp) * (i/len(temp_bus)))
        temp_bus[i] = amplitude * temp_bus[i]

    # now add on the bus
    for i in range(start_time * sr, end_time*sr):
        bus[i] = bus[i] + temp_bus[i - start_time*sr]

    return bus
```

DIY Sinewave Chirp Generator (Triangle waveform)(Function call)

```
# realization of the sound
sr = 44100; sec = 60
linearity = ['linear', 'exp', 'polynomial_1', 'polynomial_2']
master_bus_L = [0 for i in range(sr*sec)]; master_bus_R = [0 for i in range(sr*sec)]

spread = [-261 + i*26 for i in range(20)]
for item in spread:
    master_bus_L = benny_trans(sr, master_bus_L, 0, 10,261 * 3,\
                               261*3 + item, 0.01 / len(spread),0.08 / len(spread), 'linear', 'linear')
for item in spread:
    master_bus_L = benny_trans(sr, master_bus_L, 0, 10,261 * 4,\
                               261*4 + item, 0.01 / len(spread),0.08 / len(spread), 'linear', 'linear')
for item in spread:
    master_bus_L = benny_trans(sr, master_bus_L, 0, 10,261 * 2,\
                               261*2 + item, 0.01 / len(spread),0.08 / len(spread), 'linear', 'linear')

# normalie amplitude
biggest = -100000
for i in range(len(master_bus_L)):
    if abs(master_bus_L[i]) > biggest:
        biggest = abs(master_bus_L[i])
for i in range(len(master_bus_L)):
    master_bus_L[i] = master_bus_L[i] * (int(np.iinfo(np.int16).max)/biggest)

sound(master_bus_L, rate=sr, label='Noise')
sound(master_bus_R, rate=sr, label='Noise')
write("sound_LLL.wav", sr, np.asarray(master_bus_L).astype(np.int16))
#write("sound_R.wav", sr, np.asarray(master_bus_R).astype(np.int16))
```

DIY Granular Synthesis (Functions)

```
def benny_granular_synth(source, source_start, source_end, layer_num,\
                         grain_size_range=(500, 1500), playback_speed_range=(0.1,10), length=1,sr=44100):
    '''
    source                is the complete 2-D array of raw sound
    source_start          is the start point of the grain sampling from source
    source_end            is the end   point of the grain sampling from source
    layer_num_range       is number of layers in the synthesis process
    grain_size_range      is the range of how many sample points should be used for small grains
    play_back_speed_range is the speed of the playback of each grains
    '''
    # Parameters
    source               = source;
    source_start         = source_start
    source_end           = source_end
    layer_num            = layer_num
    grain_size_range     = grain_size_range
    playback_speed_range = playback_speed_range
    length               = length
    sr                   = sr

    #1  Step One separate the source in two L and R channel
    source_l = source[0]; source_r = source[1]

    #2  Create a length * sr blank canvas
    output_l = [0 for i in range(sr * length)]; output_r = [0 for i in range(sr * length)]

    #3  Now get the fragment of source bounded by source_start and source_end
    source_frag_l = source_l[source_start * sr : source_end * sr]
    source_frag_r = source_r[source_start * sr : source_end * sr]
```

```
#   After getting the fragment of source, now we can go to the real granular synth!

#4  Now Do the granular synth with while loop!
for i in range(layer_num):
    temp_voice_l = []
    temp_voice_r = []
    while True:
        start_grain          = int(len(source_frag_l) * random.random())
        grain_increment      = int(random.uniform(grain_size_range[0], grain_size_range[1]))
        speed = random.uniform(playback_speed_range[0],playback_speed_range[1])
        #print(start_grain);print(grain_increment)
        end_grain            = start_grain + grain_increment
        if end_grain > len(source_frag_l):
            end_grain = len(source_frag_l)
        origin_speed_grain_l = source_frag_l[start_grain: end_grain]
        origin_speed_grain_r = source_frag_r[start_grain: end_grain]
        temp_length = len(origin_speed_grain_l)
        speeded_l = [origin_speed_grain_l[i] for i in range(0, temp_length)]
        speeded_r = [origin_speed_grain_r[i] for i in range(0, temp_length)]
        temp_voice_l = temp_voice_l + list(np.asarray(speeded_l) * np.kaiser(len(speeded_l),4))
        temp_voice_r = temp_voice_r + list(np.asarray(speeded_r) * np.kaiser(len(speeded_l),4))

        if len(temp_voice_l) > (length*sr):
            temp_voice_l = temp_voice_l[:length*sr]
            temp_voice_r = temp_voice_r[:length*sr]
            break

    for i in range(length * sr):
        output_l[i] = output_l[i] + temp_voice_l[i]
        output_r[i] = output_r[i] + temp_voice_r[i]

# 5 Now normalize the volume as the trans function did for both output l and r
biggest = -100000
for i in range(len(output_l)):
    if abs(output_l[i]) > biggest:
        biggest = abs(output_l[i])
for i in range(len(output_l)):
    output_l[i] = output_l[i] * (int(np.iinfo(np.int16).max)/biggest)

biggest = -100000
for i in range(len(output_r)):
    if abs(output_r[i]) > biggest:
        biggest = abs(output_r[i])
for i in range(len(output_r)):
    output_r[i] = output_r[i] * (int(np.iinfo(np.int16).max)/biggest)

# now return the output_l and output_r and everything done!!
return np.asarray([output_l, output_r]).T
```

Kaiser window is applied here for reduction high freq clips

DIY Granular Synthesis (Function Calls)

```
# This is the section of handcrafted granular systhesis
sr, y = read("piece_0.wav")
# Params: source, source_start, source_end, layer_num, grain_size_range, playback_speed_range, length,sr
o0 = benny_granular_synth(y.T, 0, 60, 10, (20,40), (1,1.1), length=6,sr=44100)
o1 = benny_granular_synth(y.T, 0, 60, 10, (200, 400), (1,1.1), length=6,sr=44100)
o2 = benny_granular_synth(y.T, 0, 60, 10, (2000, 4000), (1,1.1), length=6,sr=44100)
o3 = benny_granular_synth(y.T, 0, 60, 10, (4000, 6000), (1,1.1), length=6,sr=44100)
o4 = benny_granular_synth(y.T, 0, 60, 10, (6000, 8000), (1,1.1), length=6,sr=44100)
o5 = benny_granular_synth(y.T, 0, 60, 10, (20000, 30000), (1,1.1), length=6,sr=44100)
write("granular_0.wav", sr, np.asarray(o0).astype(np.int16))
write("granular_1.wav", sr, np.asarray(o1).astype(np.int16))
write("granular_2.wav", sr, np.asarray(o2).astype(np.int16))
write("granular_3.wav", sr, np.asarray(o3).astype(np.int16))
write("granular_4.wav", sr, np.asarray(o4).astype(np.int16))
write("granular_5.wav", sr, np.asarray(o5).astype(np.int16))
```