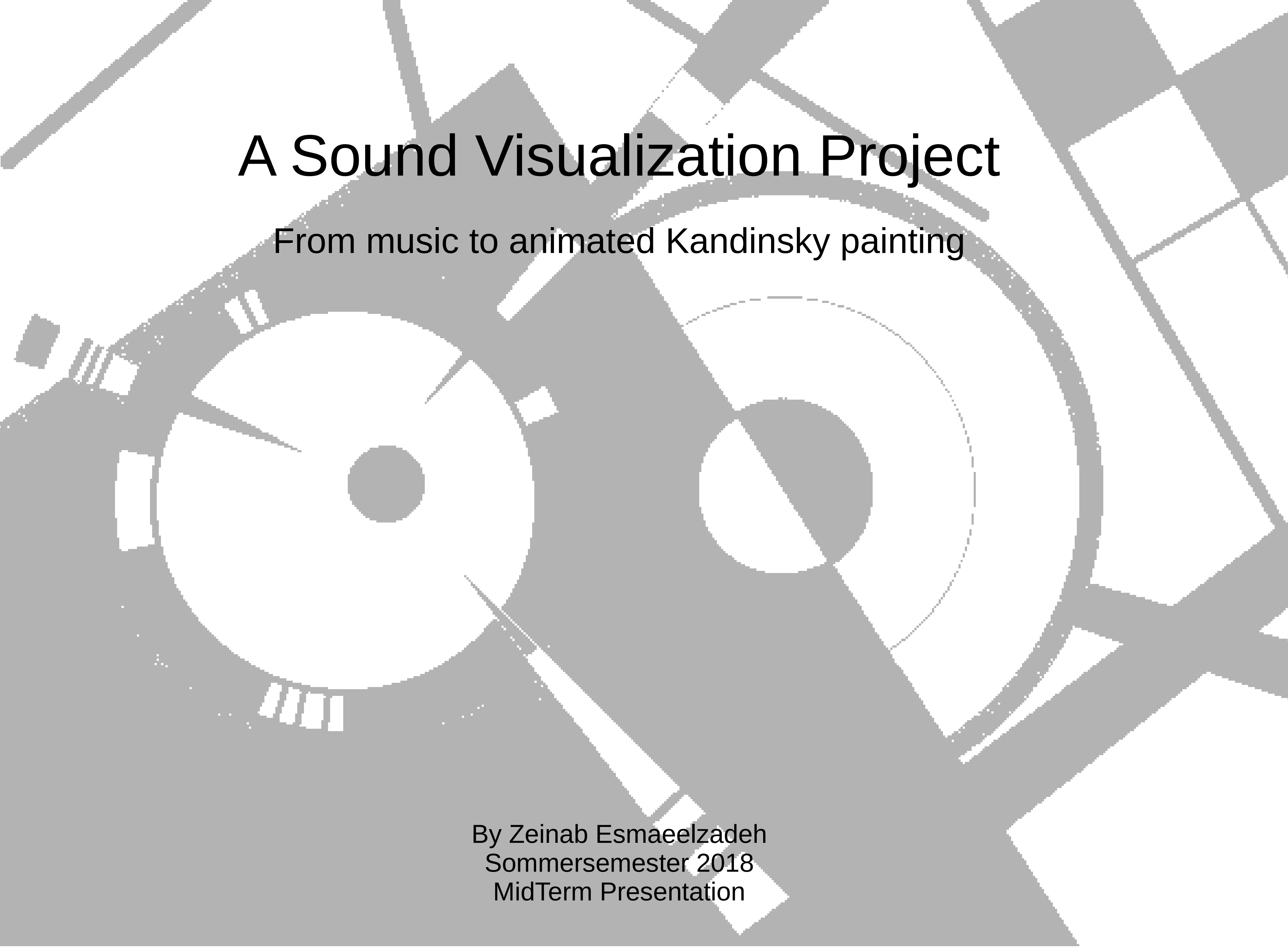


A Sound Visualization Project

From music to animated Kandinsky painting

By Zeinab Esmaeelzadeh
Sommersemester 2018
MidTerm Presentation

The project is divided in 2 parts:

Part I :
Sound analysis/Signal processing

Input :

- Analogue
  Sound
  Noise
  Microphone
  (AUX input)

- Digital:
  Midi

Output:
Matrices of numbers:
- Columns: Selected feature/Properties:
 Pitch,ADSR,Velocity,Selected frequency,etc
- Rows: Slice of time considering beat or feasibility of CPU

Part II:
Visualization in R language

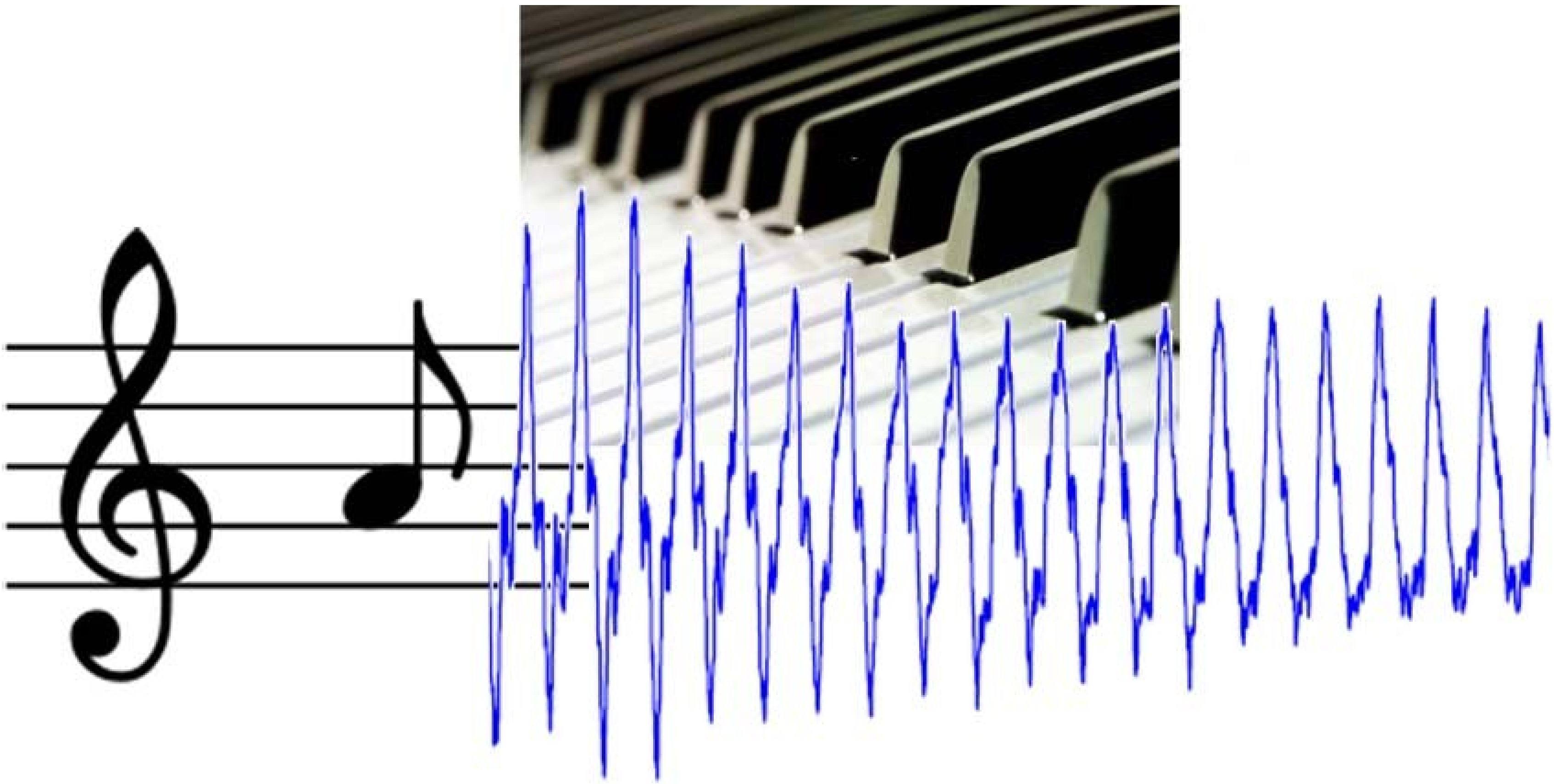
Input:
Matrices → R data frame

Output:
Pictures similar to Kandinsky’s Paintings for each feasible slice of time.
(modified Kandinsky R package)

And then animating pictures in video

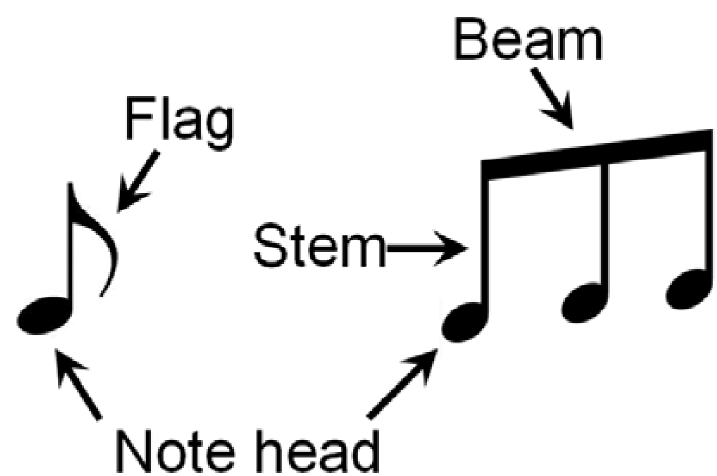
Introduction to music representation

1.1 Sheet Music Representations
1.2 Symbolic Representations
1.3 Audio Representation



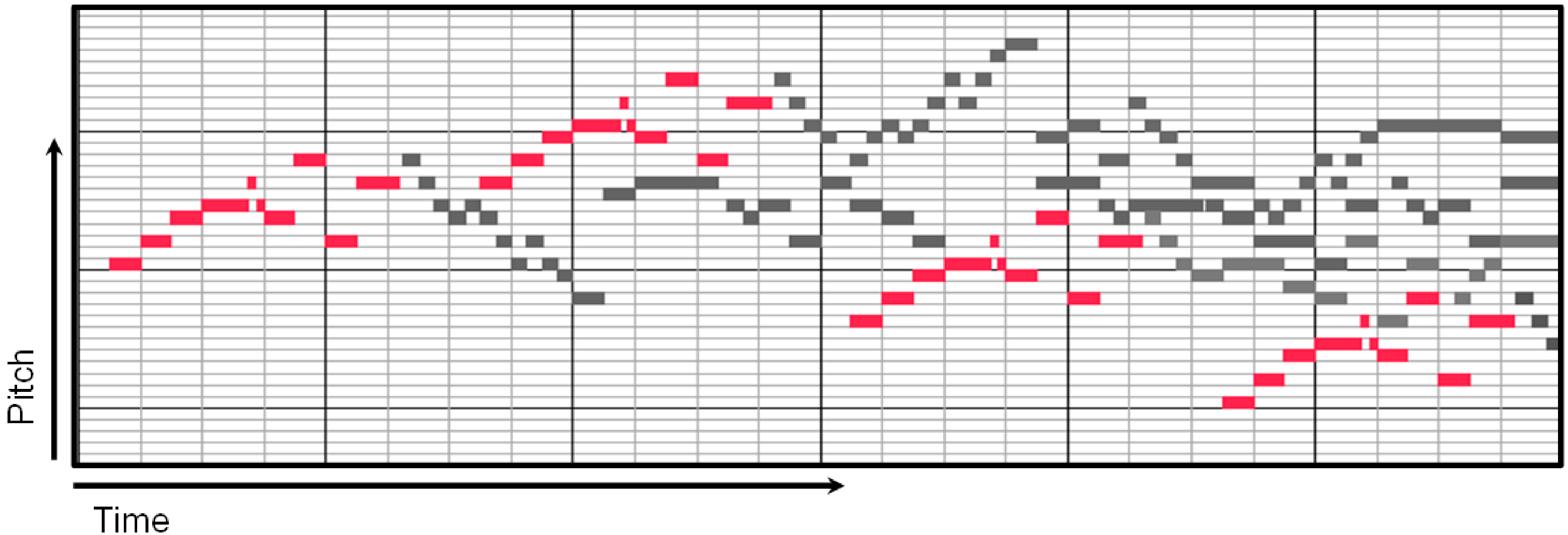
1.1 Sheet Music Representations

1.1 Sheet Music Representations

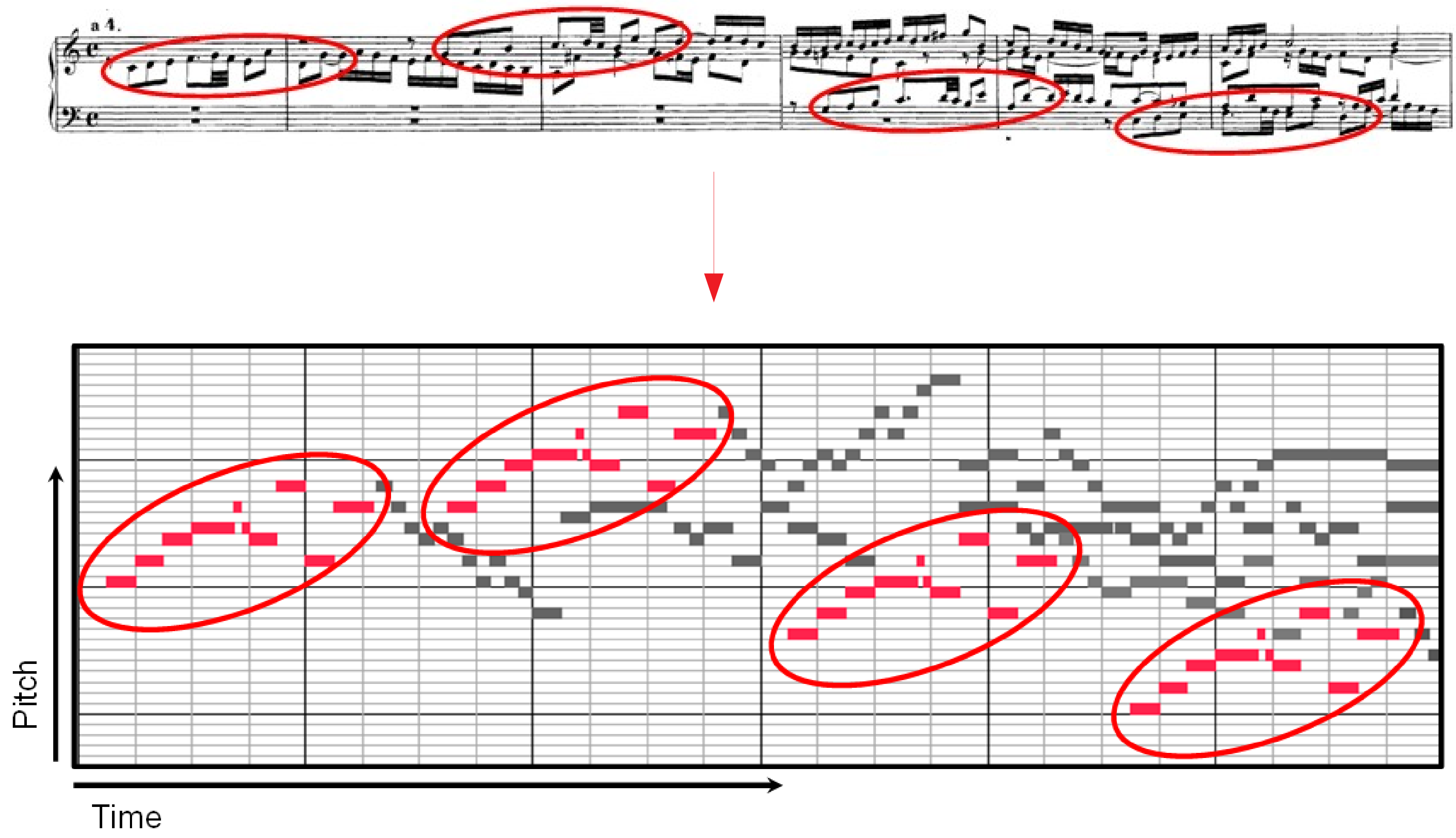


1.1 Sheet Music Representations

1.2 Symbolic Representations

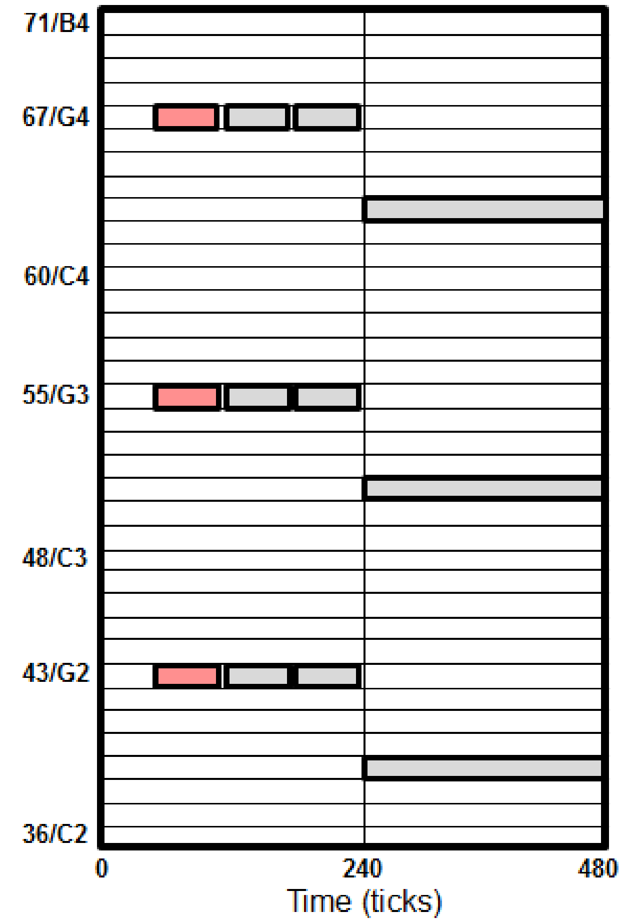


1.2 Symbolic Representations

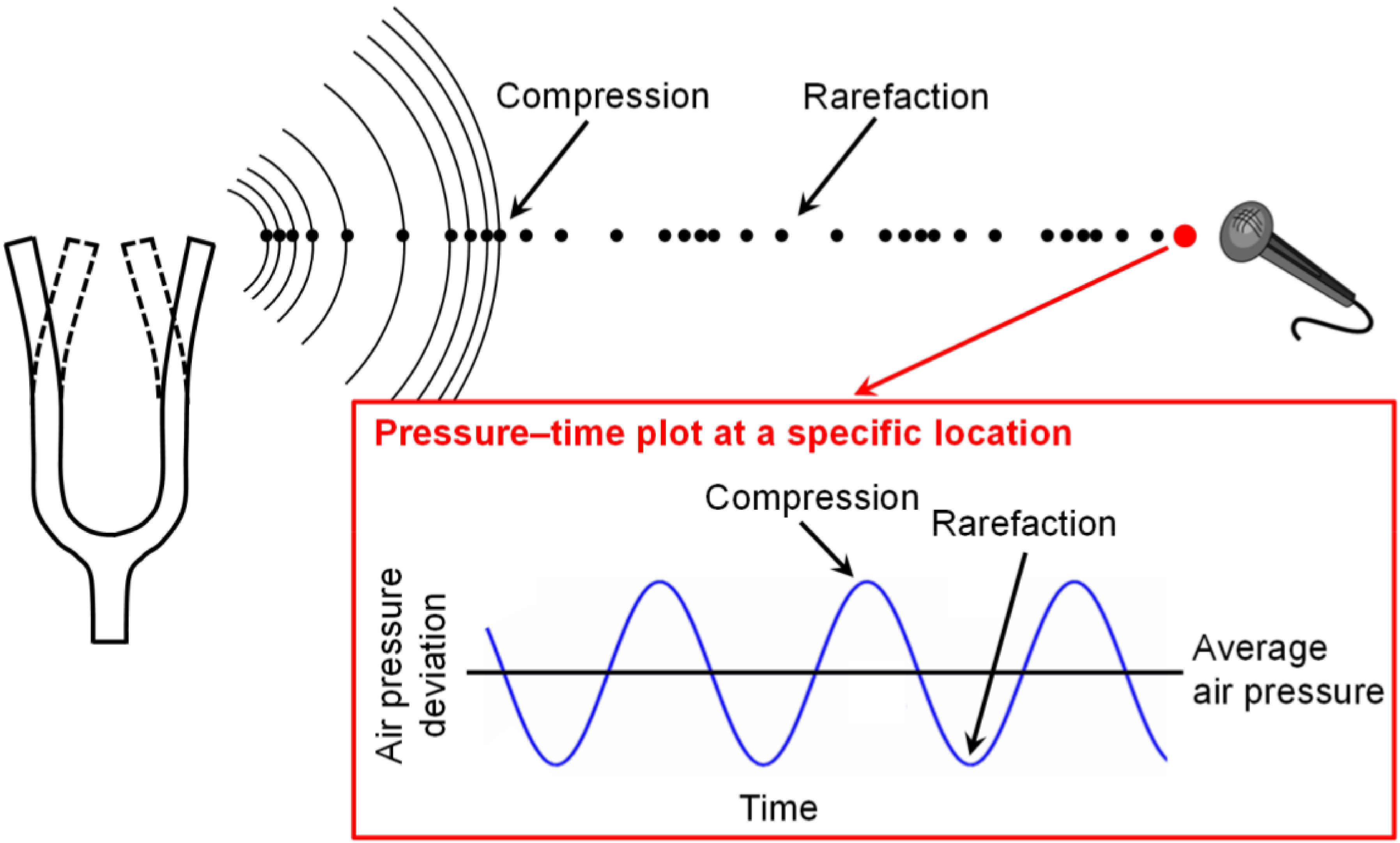


1.2 Symbolic Representations

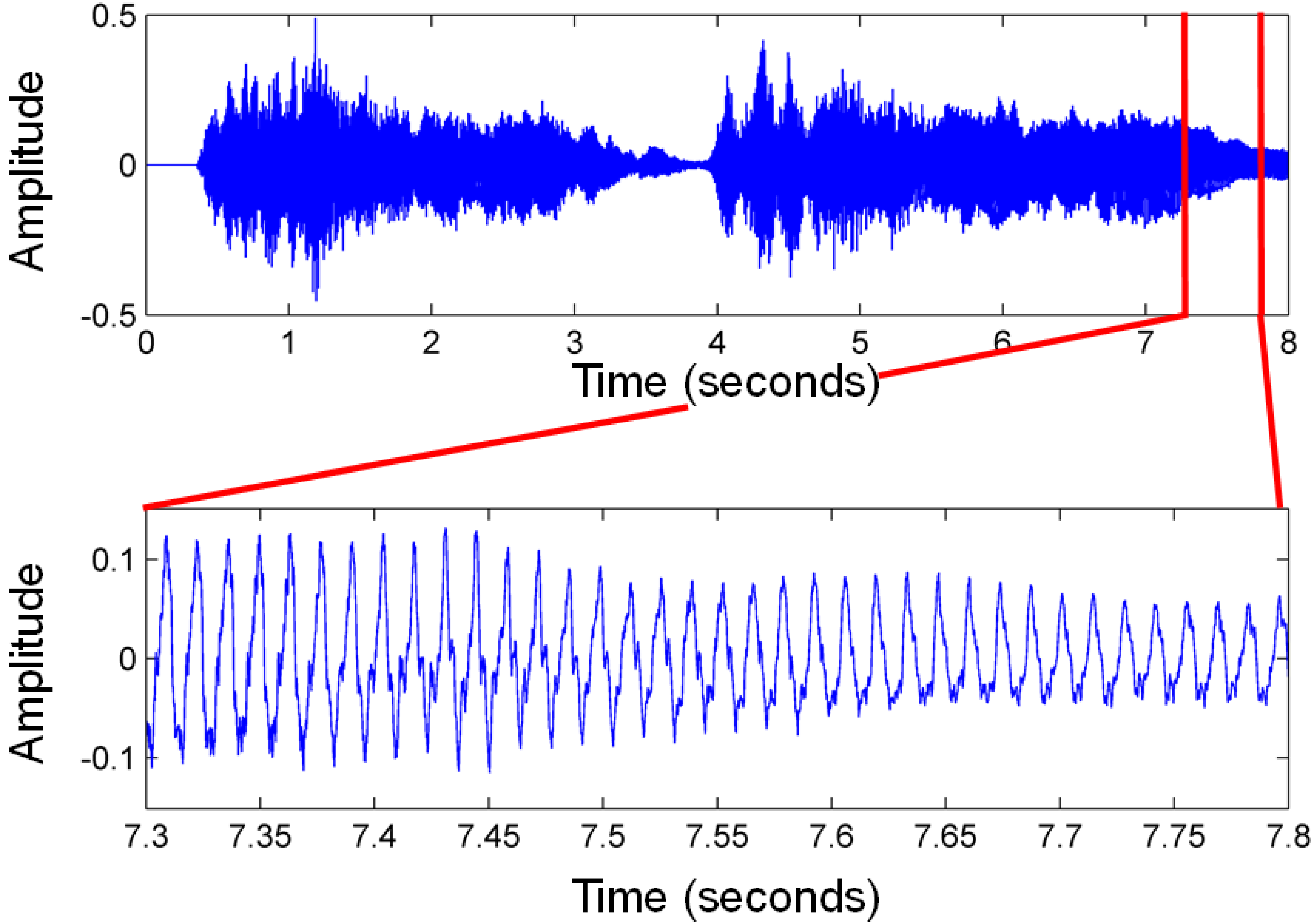
Time (Ticks)	Message	Channel	Note Number	Velocity
60	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	63	100
0	NOTE ON	2	51	100
0	NOTE ON	2	39	100
240	NOTE OFF	1	63	0
0	NOTE OFF	2	51	0
0	NOTE OFF	2	39	0



1.3 Audio Representation

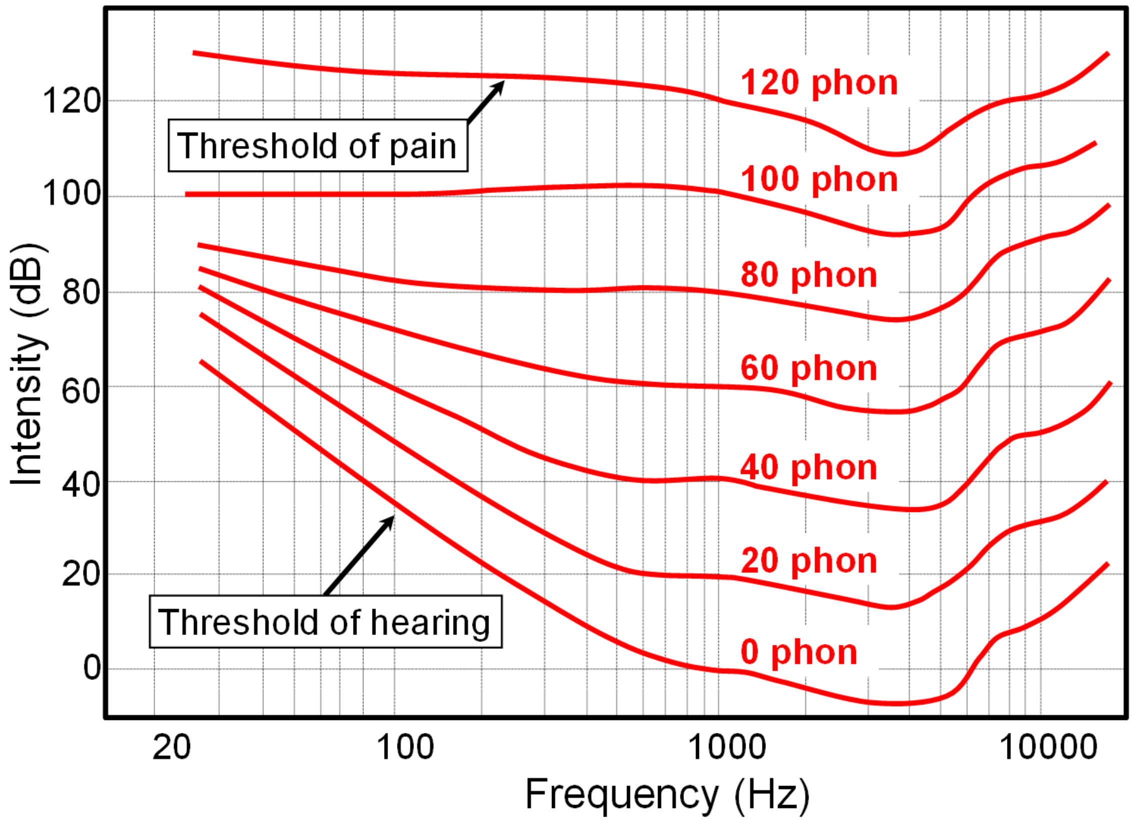


1.3 Audio Representation

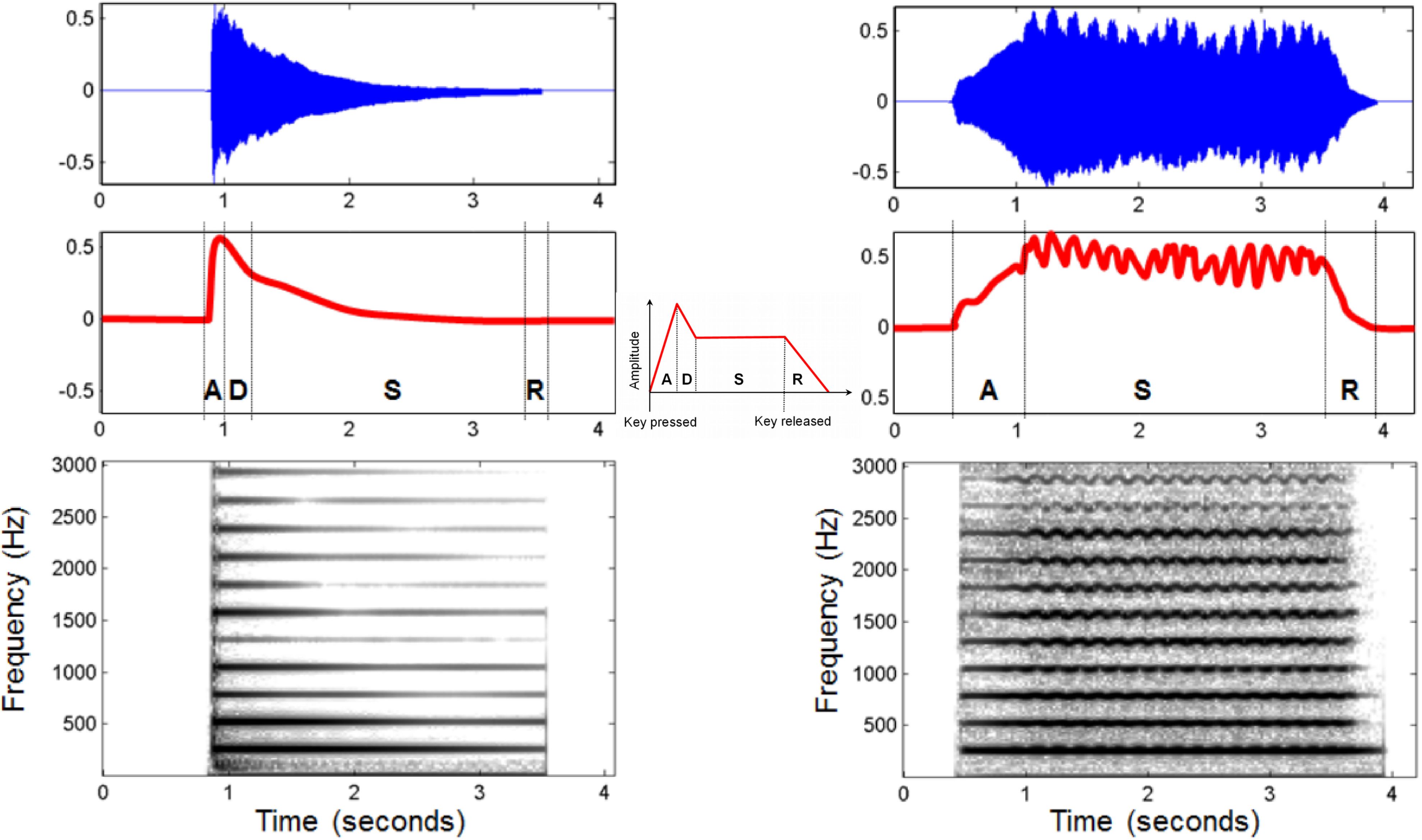


1.3 Audio Representation

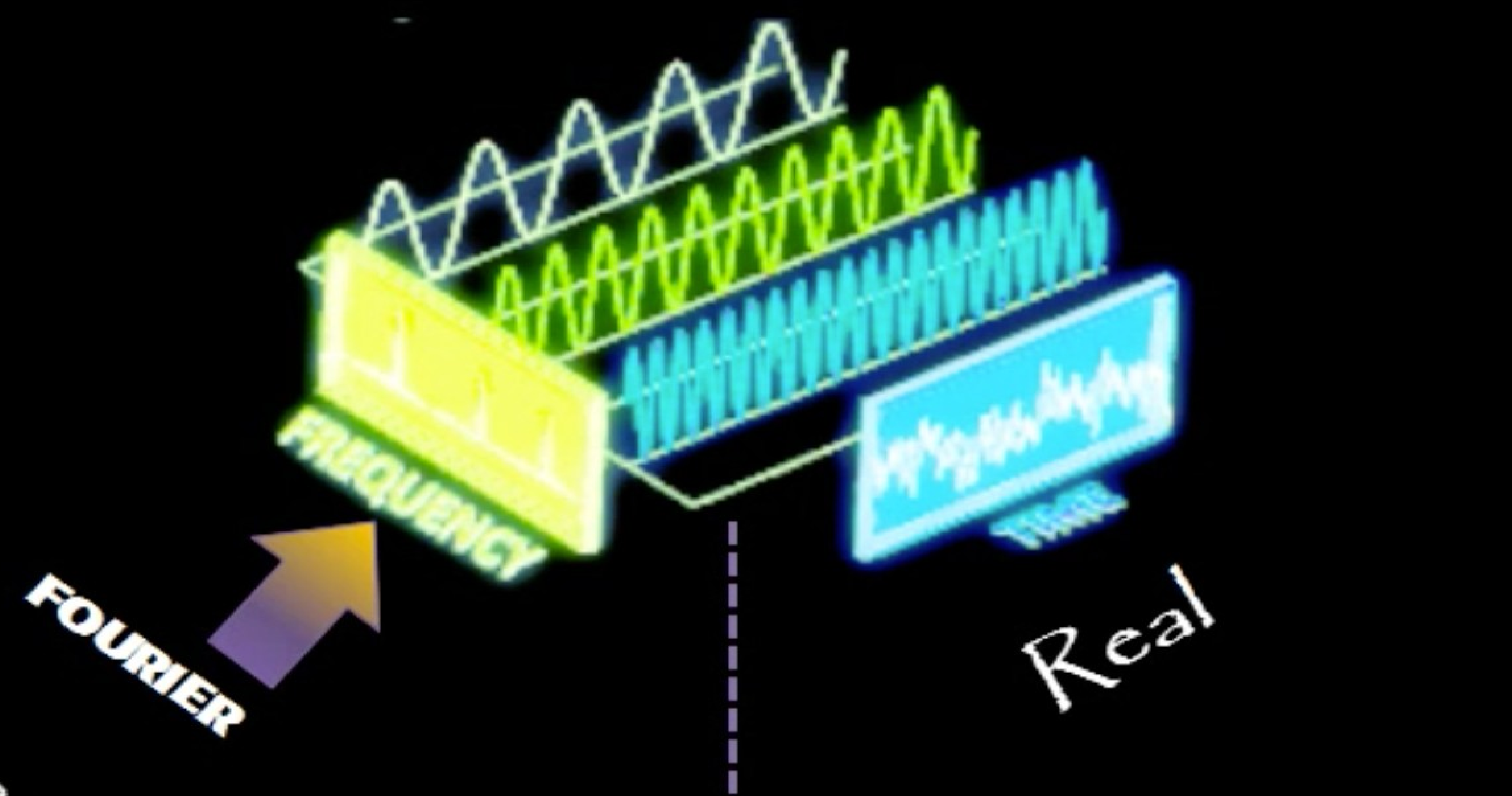
Source	Intensity	Intensity level	× TOH
Threshold of hearing (TOH)	10^{-12}	0 dB	1
Whisper	10^{-10}	20 dB	10^{2}
Pianissimo	10^{-8}	40 dB	10^{4}
Normal conversation	10^{-6}	60 dB	10^{6}
Fortissimo	10^{-2}	100 dB	10^{10}
Threshold of pain	10	130 dB	10^{13}
Jet take-off	10^{2}	140 dB	10^{14}
Instant perforation of eardrum	10^{4}	160 dB	10^{16}



1.3 Audio Representation



1.4 Fourier Analysis

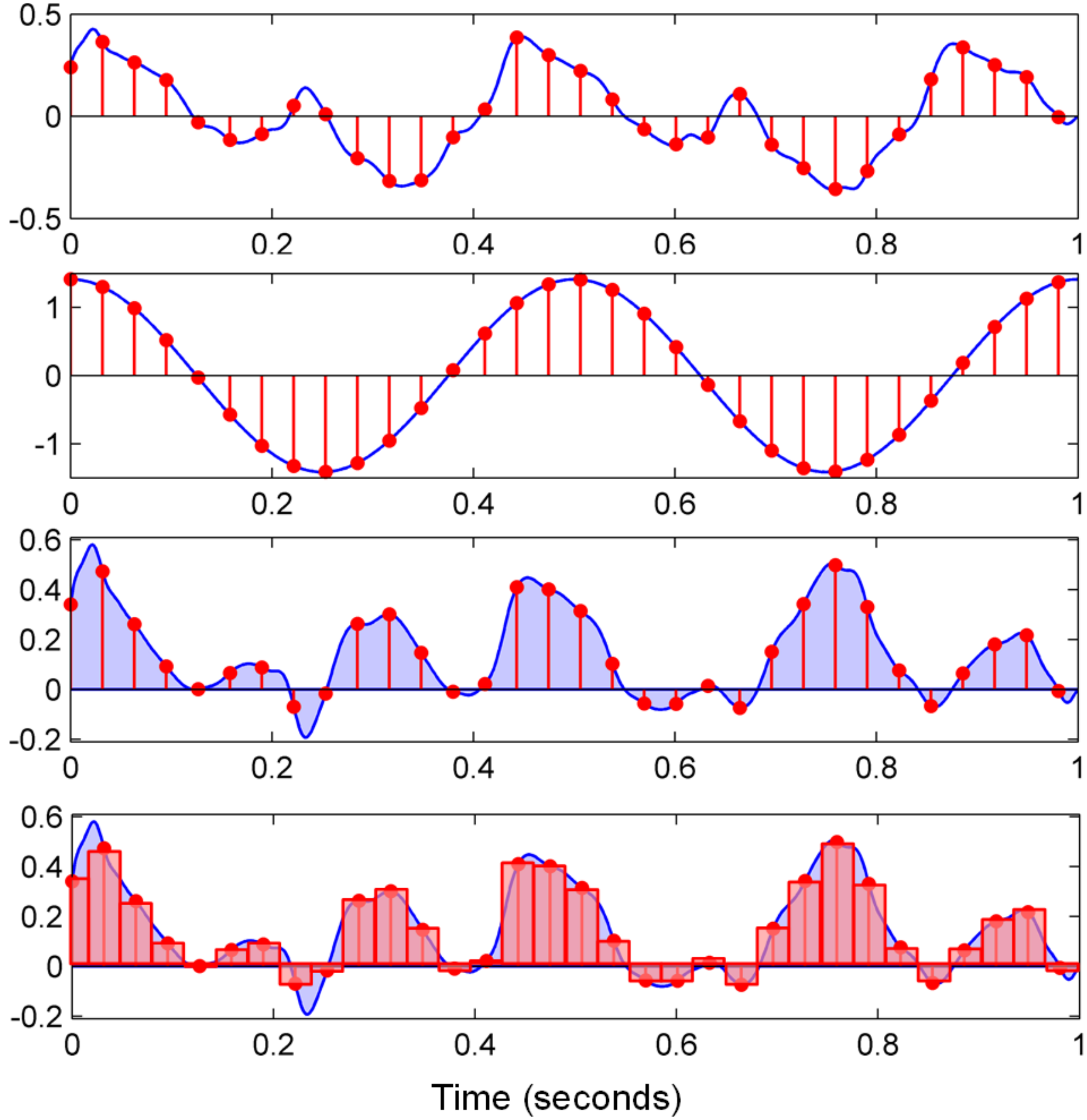


$$\sin\left(\frac{2\pi nx}{P}+\phi_n\right) \equiv \sin(\phi_n)\cos\left(\frac{2\pi nx}{P}\right)+\cos(\phi_n)\sin\left(\frac{2\pi nx}{P}\right)$$

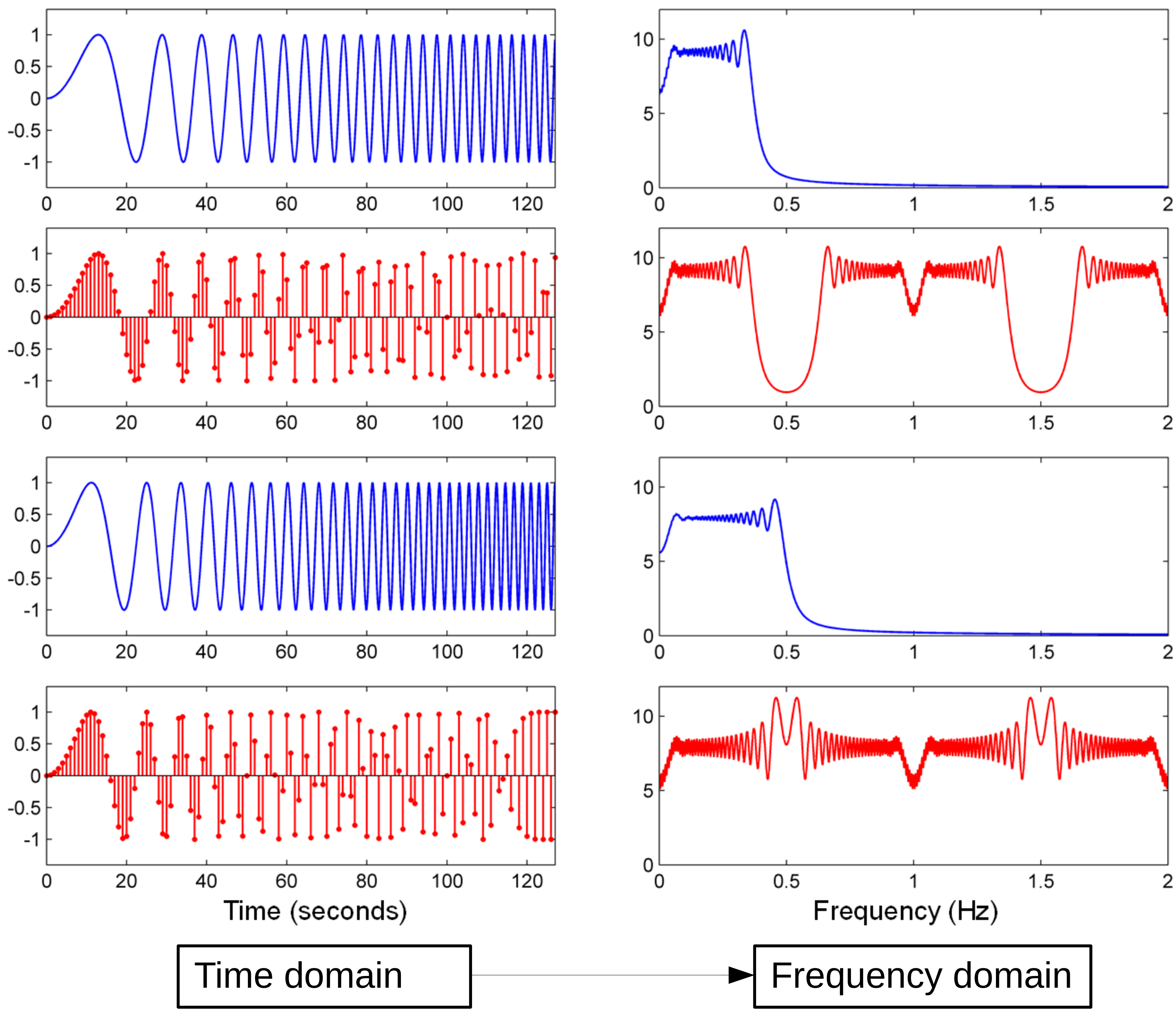
$$\sin\left(\frac{2\pi nx}{P}+\phi_n\right) \equiv \mathrm{Re}\left\{\frac{1}{i}\cdot e^{i\left(\frac{2\pi nx}{P}+\phi_n\right)}\right\} = \frac{1}{2i}\cdot e^{i\left(\frac{2\pi nx}{P}+\phi_n\right)} - \left(\frac{1}{2i}\cdot e^{-i\left(\frac{2\pi nx}{P}+\phi_n\right)}\right),$$

$$s_N(x) = \overbrace{a_0}^{A_0}/2 + \sum_{n=1}^{N}\left(\overbrace{a_n}^{A_n\sin(\phi_n)}\cos\left(\frac{2\pi nx}{P}\right) + \overbrace{b_n}^{A_n\cos(\phi_n)}\sin\left(\frac{2\pi nx}{P}\right)\right)$$
$$= \sum_{n=-N}^{N} c_n\cdot e^{i\frac{2\pi nx}{P}},$$

1.4 Fourier Analysis



1.4 Fourier Analysis



PART I : Sound analysis/Signal processing

Input :

- Analogue:
  Sound
  Noise
  Microphone
  (AUX input)

- Digital:
  Midi

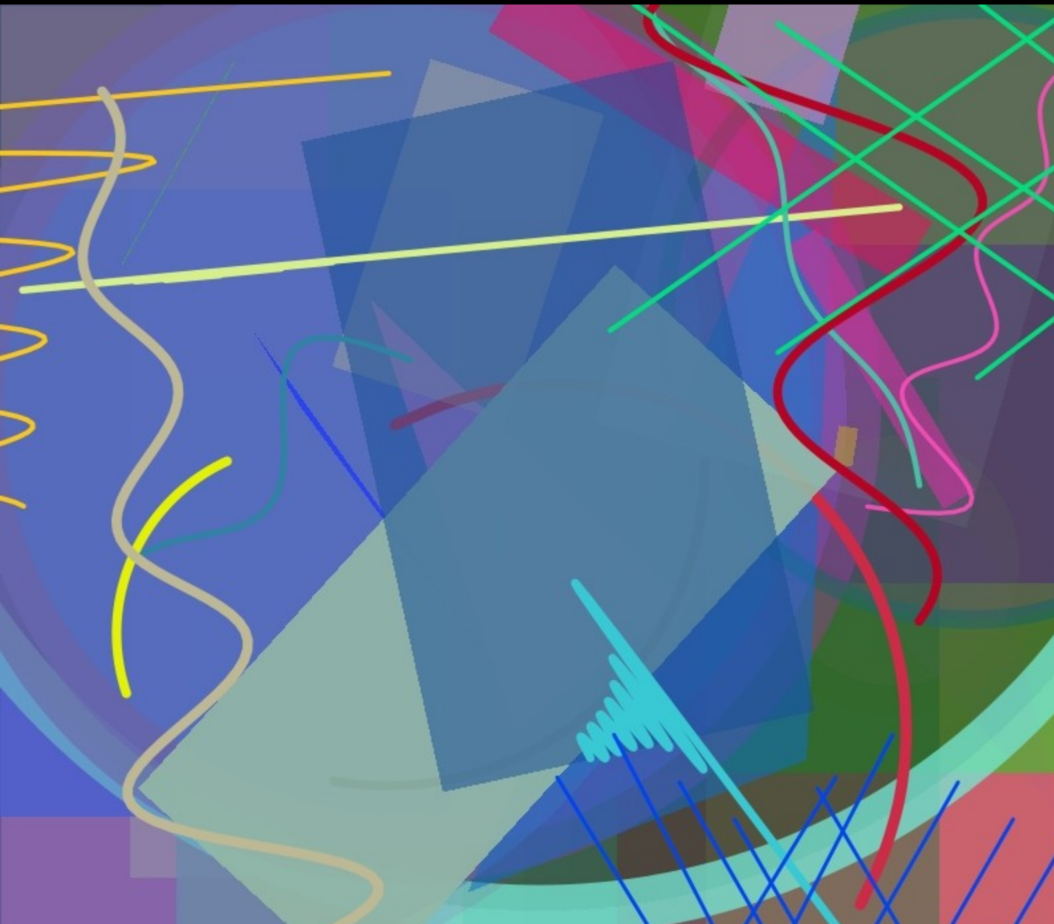
Output:
Matrices of numbers:

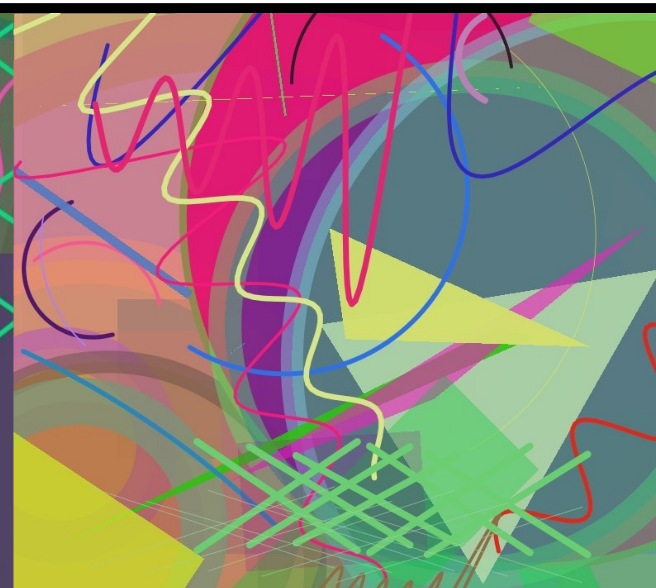
- Columns: Selected feature/Properties:
  Pitch,ADSR,Velocity,Selected frequency,etc
- Rows: Slice of time considering beat or feasibility of CPU

How it is done:

With Processing , R , Python (Scripting Languages)

PART II : Visualization in R language





Input:

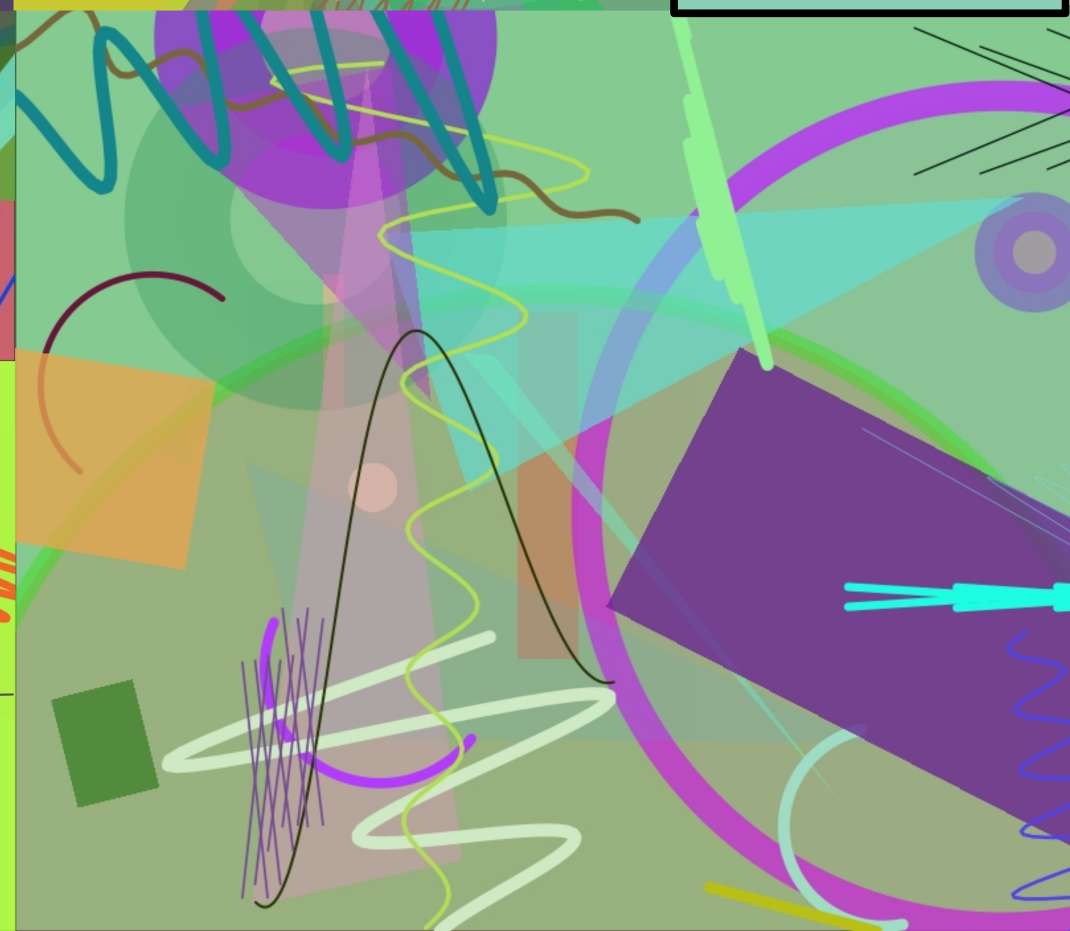
Matrices

↓

R data frame

Output:

Pictures similar to Kandinsky’s Paintings for each feasible slice of time.
(modified Kandinsky R package)



Visualization in R language

```
require(grid)
randomKandinsky <- function(n = 10) {
  grid.newpage()

  grid.rect(gp=gpar(fill=rgb(runif(1),
                             runif(1),
                             runif(1),
                             runif(1))))

  for (i in 1:n) {
    grid.rect(x = runif(1), y = runif(1), width = runif(1), height = runif(1),
              gp = gpar(col = NA,
                        fill=rgb(runif(1),
                                 runif(1),
                                 runif(1),
                                 runif(1))))
    grid.circle(x = runif(1), y = runif(1), r = runif(1),
                gp = gpar(
                  lwd = runif(1, 0, 100),
                  col = rgb(runif(1),
                            runif(1),
                            runif(1),
                            runif(1)),
                  fill=rgb(runif(1),
                           runif(1),
                           runif(1),
                           runif(1))))
    grid.polygon(x = runif(3), y = runif(3),
                 gp = gpar(col = NA,
                           fill=rgb(runif(1),
                                    runif(1),
                                    runif(1),
                                    runif(1))))

    grid.curve(runif(1), runif(1), runif(1), runif(1),
               curvature = runif(1, -1, 1), square = FALSE, ncp = sample(100, 1),
               gp = gpar(lwd = runif(1, 0, 10),
                         col = rgb(runif(1),
                                   runif(1),
                                   runif(1),
                                   1)))

    vp1 <- viewport(x = runif(1), y = runif(1), width = runif(1), height = runif(1), angle = runif(1) * 360)
    grid.rect(x = runif(1), y = runif(1), width = runif(1), height = runif(1),
              vp = vp1,
              gp = gpar(col = NA,
                        fill=rgb(runif(1),
                                 runif(1),
                                 runif(1),
                                 runif(1))))

    vp2 <- viewport(x = runif(1), y = runif(1), width = runif(1), height = runif(1), angle = runif(1) * 360)
    gCurve(sin(x)/(x), sample(5, 1), sample(10:50, 1), vp = vp2,
           gp = gpar(lwd = runif(1, 0, 10),
                     col = rgb(runif(1),
                               runif(1),
                               runif(1),
                               1)))
  }
  vp3 <- viewport(x = runif(1), y = runif(1), width = runif(1), height = runif(1), clip = "off")
  gCrissCross(vp = vp3,
              gp = gpar(lwd = runif(1, 0, 10),
                        col = rgb(runif(1),
                                  runif(1),
                                  runif(1),
                                  1)))

}
```

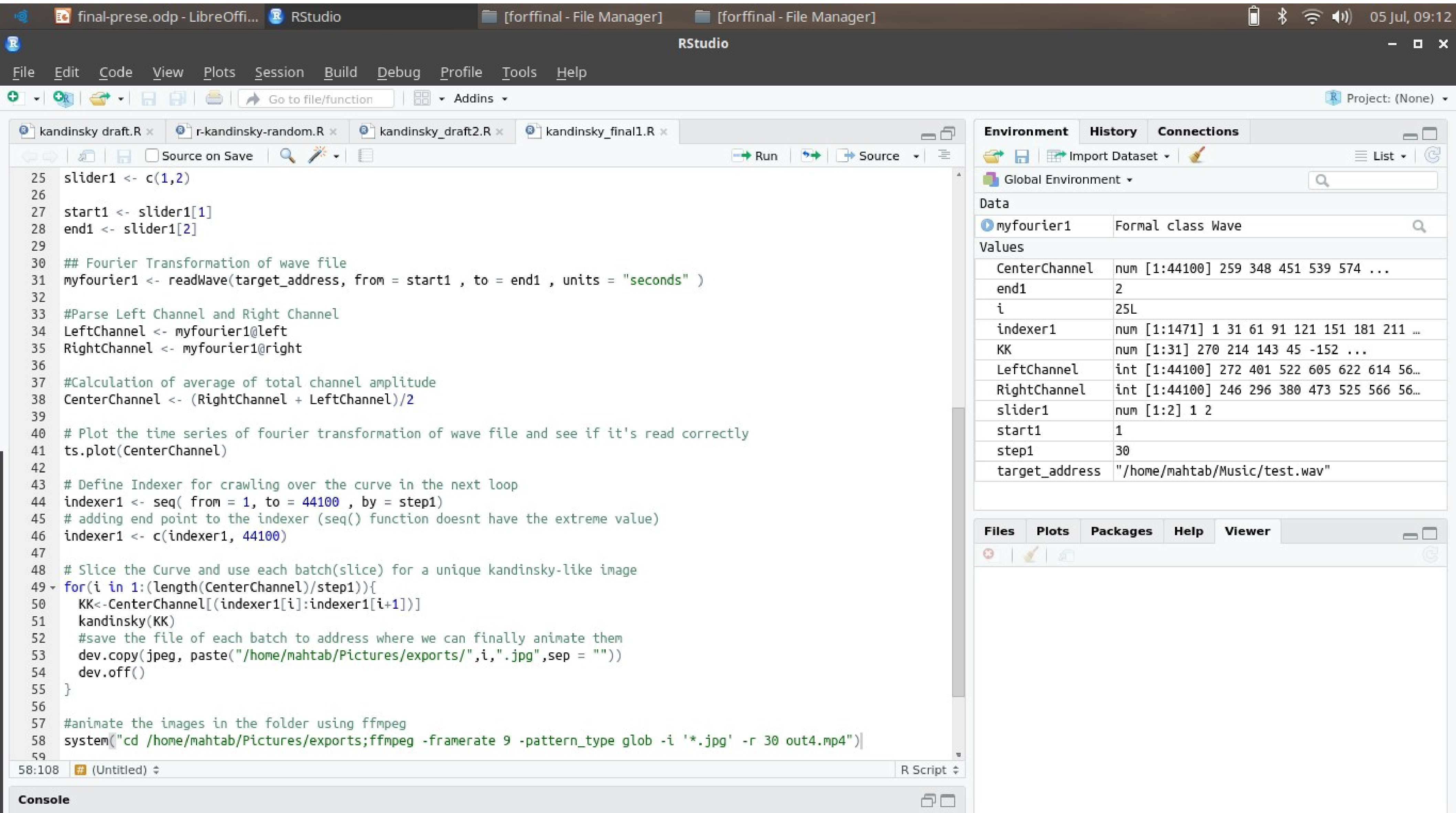
How it is done:

See the demo

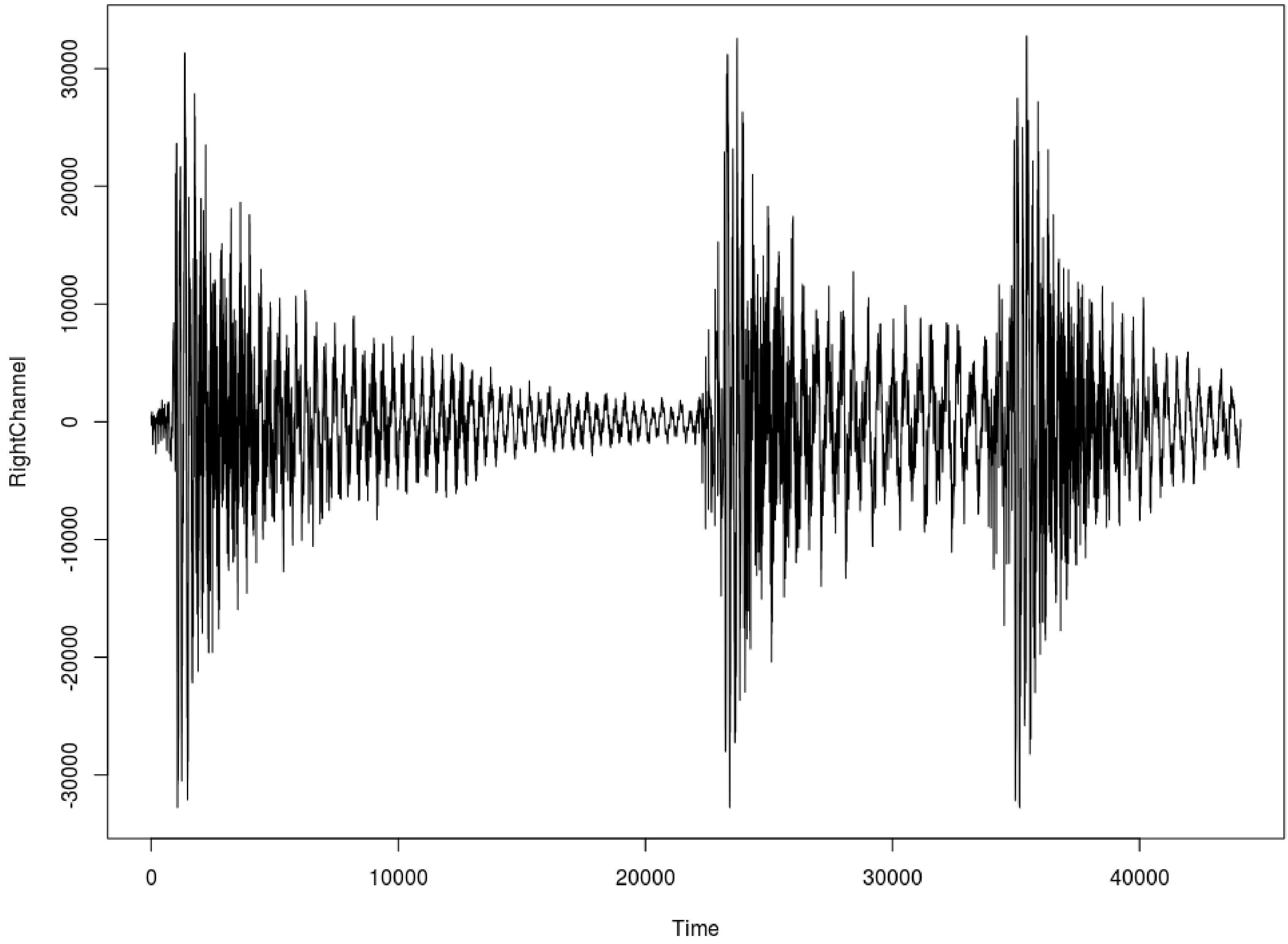
devtools::install_github("nautilus69/kandinsky")

Sound file visualization with Fourier transformation in R

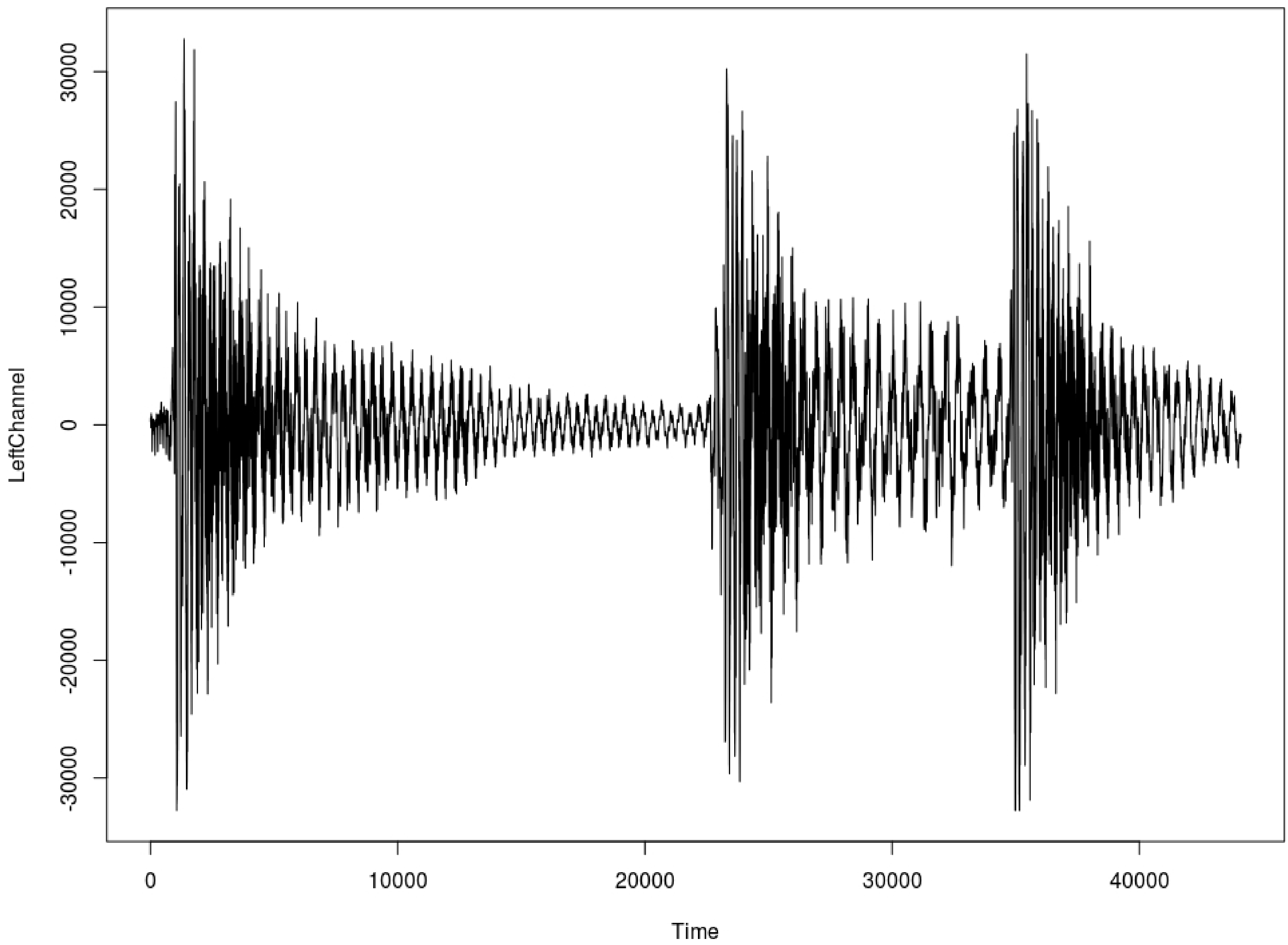
Descriptions: This Program Read a Wave music file from Local Hard Drive and Visualize the Fourier transformation processed music with Kandinsky-like output and animate them.
(I try to make a Web app with ShinyR so the user can interact with the parameters)



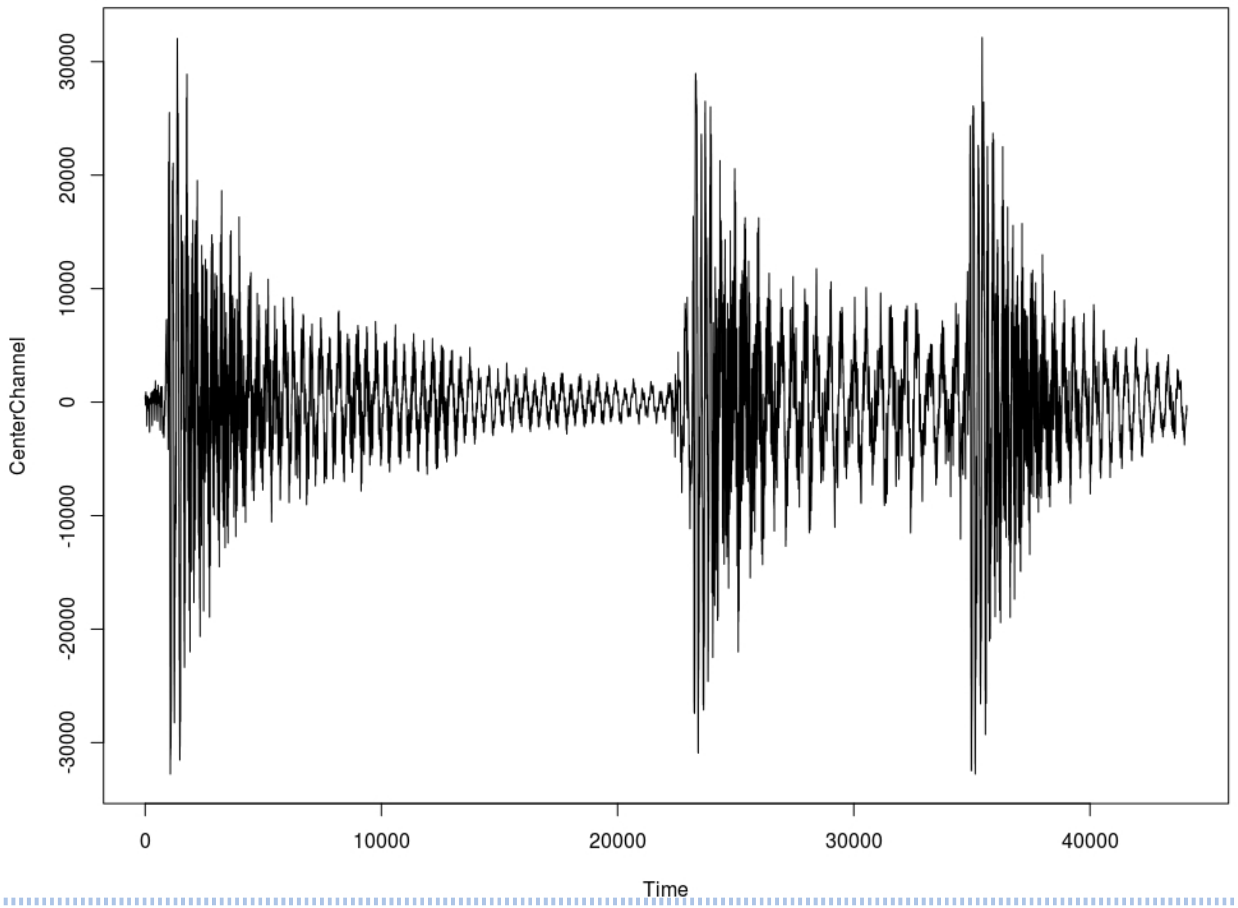
Sound file visualization with Fourier transformation in R



Sound file visualization with Fourier transformation in R



Sound file visualization with Fourier transformation (Amplitude per time)



Calculation of average of total channel amplitude

CenterChannel <- (RightChannel + LeftChannel)/2

Plot the time series of fourier transformation of wave file and see if it's read correctly

ts.plot(CenterChannel)

Sound file visualization with Fourier transformation (Amplitude per time)

DEMO

Thank You