모듈 I
입체음향 구현

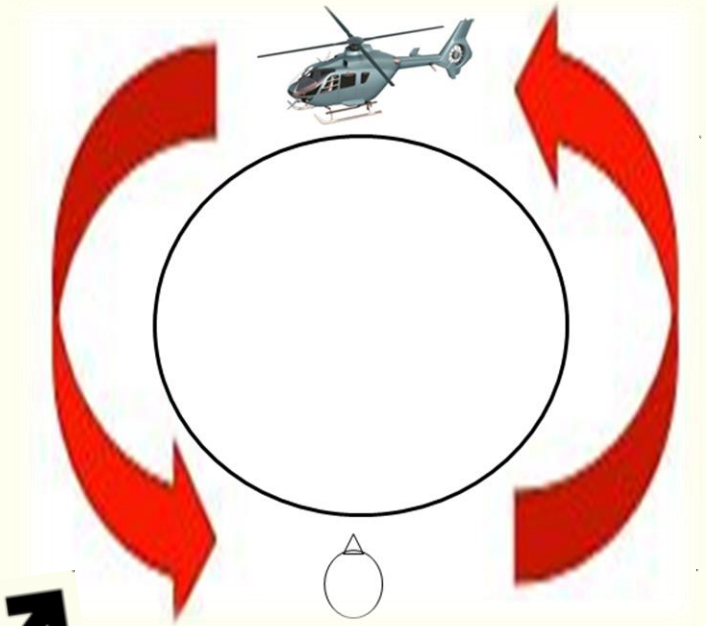
3조 디지털사운드101

15011136 신현욱
15011164 이수진
17011658 양재연
17011669 문성현
19040217 원이선

목차

입체음향 구현

컴퓨터와 이에 연결된 스테레오 스피커만으로 사용자의 머리 위로 소리를 발생시키는 물체가 원형으로 이동하는 음향효과를 구현

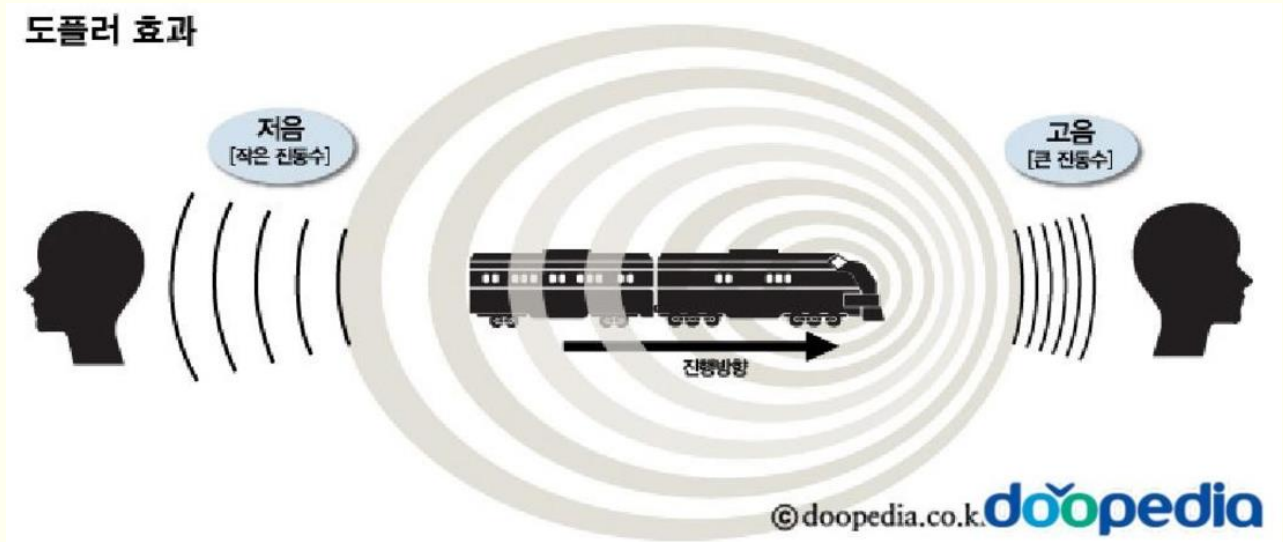


도플러 효과

소리의 세기를 이용한 양이효과

시간차를 이용한 양이효과

초기 접근 방법 : 도플러 효과

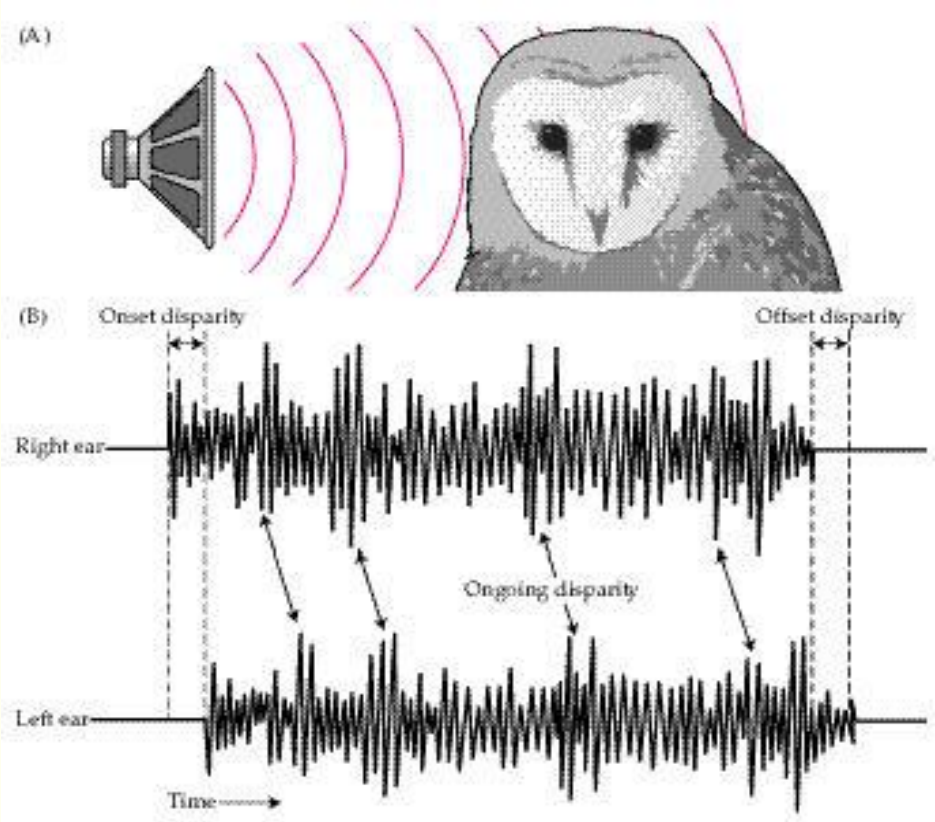


파원과 관측자 사이의 거리가 좁아질 때에는 파동의 주파수가 더 높게, 거리가 멀어질 때에는 파동의 주파수가 더 낮게 관측되는 현상

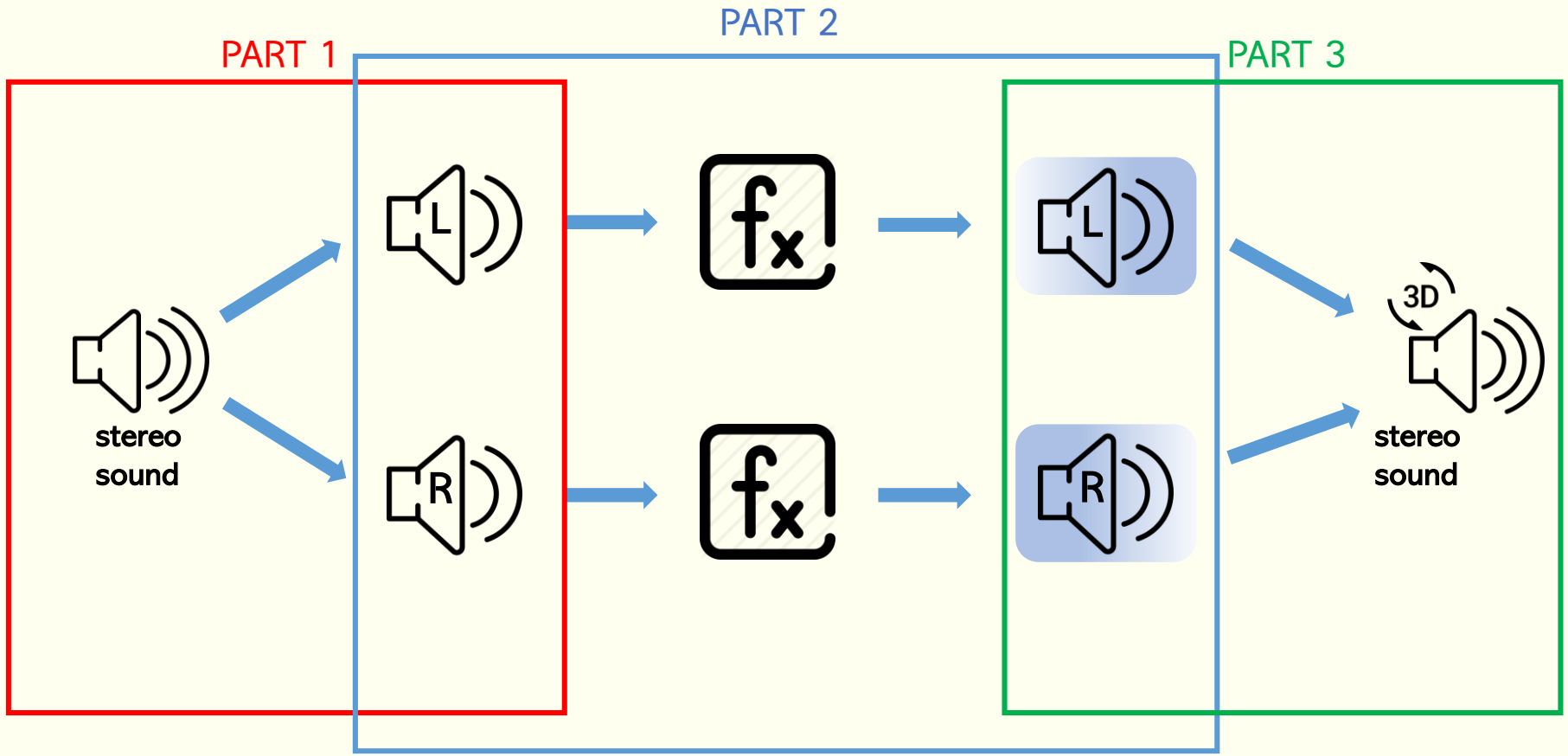
=> 디지털화 된 샘플링 데이터를 가지고 도플러 효과를 표현하는 데에는 어려움이 있음.

소리의 세기와 시간차를 이용한 양이 효과

2개의 소리가 들릴 때, 각 소리가 어느 쪽에서 온 건지
왼쪽, 오른쪽 소리의 도착시간을 통해 판단할 수 있다.
=> 소리가 연속적임으로 시간차를 두고 구현하기에는 우리 수준에서는 어렵다.
=> 좌우 소리 차를 구현하기 위한 소리의 강약 조절



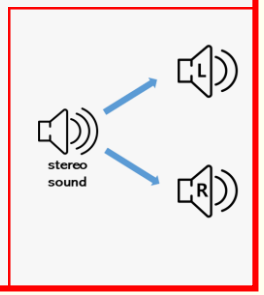
Python을 이용한 수행 방법



핵심 코드 분석

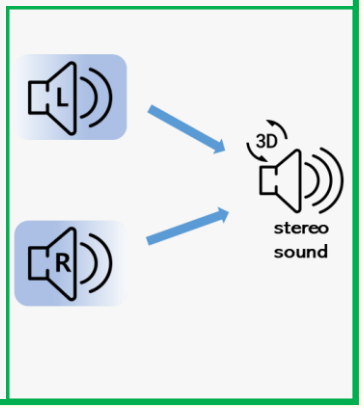
PART 1 - stereo sound를 좌우로 분리하기

```
# 파일 분리
def save_wav_channel(fn, wav, channel):
    # Read data
    nch   = wav.getnchannels() # getnchannels() : 오디오 채널 수 반환 (모노 : 1, 스테레오 : 2)
    depth = wav.getsampwidth() # getsampwidth() : 샘플 폭을 바이트 단위로 반환
    wav.setpos(0) # setpos(위치) : 파일 포인터를 지정된 위치로 설정
    sdata = wav.readframes(wav.getnframes()) # readframes(n) : 최대 n개의 오디오 프레임을 bytes 객체로 읽고 반환

    # Extract channel data (24-bit data not supported)
    typ = { 1: np.uint8, 2: np.uint16, 4: np.uint32 }.get(depth)
    if not typ:
        raise ValueError("sample width {} not supported".format(depth))
    if channel >= nch:
        raise ValueError("cannot extract channel {} out of {}".format(channel+1, nch))
    print ("Extracting channel {} out of {} channels, {}-bit depth".format(channel+1, nch, depth*8))
    data = np.frombuffer(sdata, dtype=typ)
    ch_data = data[channel::nch]

    # Save channel to a separate file
    outwav = wave.open(fn, 'w') # open(file, mode) : 파일 열기
    outwav.setparams(wav.getparams()) # setparams(tuple) : parameter 설정
    outwav.setnchannels(1) # setnchannles(n) : 채널 수 설정
    outwav.writeframes(ch_data.tostring()) # writeframes(data) : 오디오 프레임 쓰기
    outwav.close() # close() : 파일 닫기
```

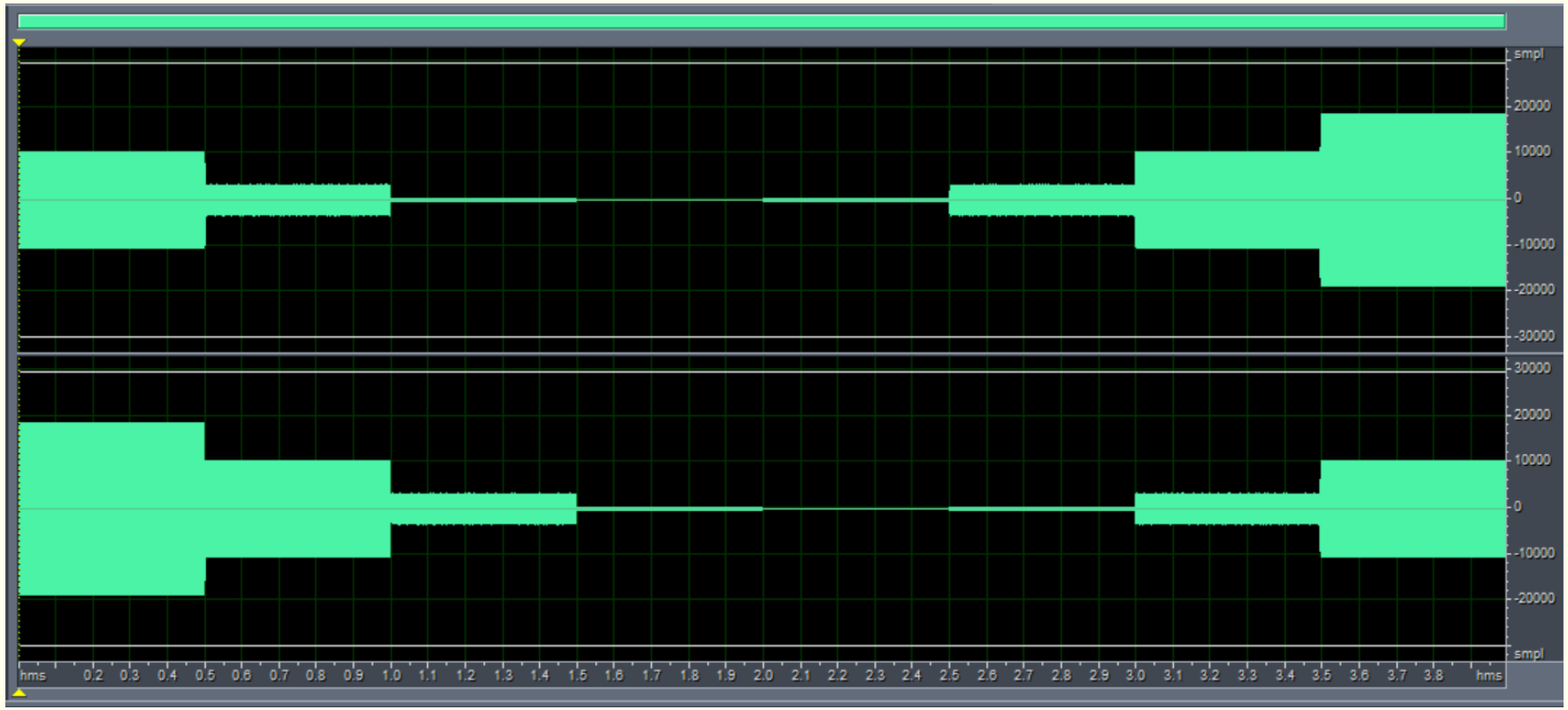


PART 3 - 좌/우 소리를 하나의 stereo sound로 합치기

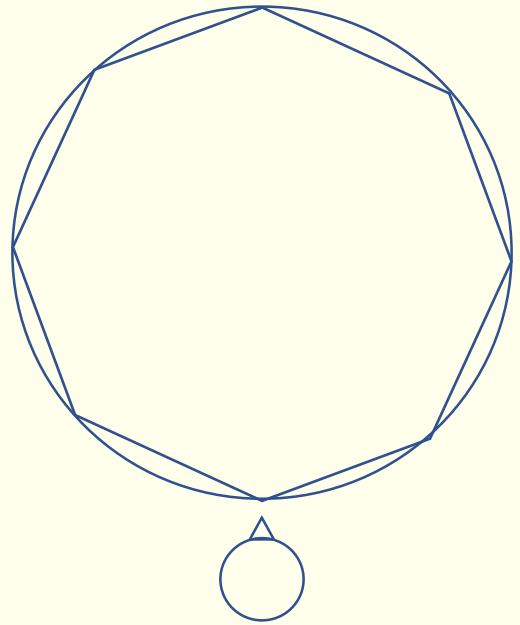
```
# 파일 결합
def make_stereo(input1, input2, output): # 함수 인자 : 왼쪽, 오른쪽, 합친 결과물

    wav1 = wave.open(input1) # open(file) : 파일 열기
    wav2 = wave.open(input2)
    (nchannels, sampwidth, framerate, nframes, comptype, compname) = wav1.getparams() # getparams() : namedtuple() 반환 (get*()의 반환값과

    assert comptype == 'NONE'  # Compressed not supported yet
    array_type = {1:'B', 2: 'h', 4: 'l'}[sampwidth]
    left_channel = array.array(array_type, wav1.readframes(nframes))[::nchannels]
    right_channel = array.array(array_type, wav2.readframes(nframes))[::nchannels]

    # 정보를 활용하는 부분

    print("nchannels", nchannels)
    print("sampwidth", sampwidth)
    print("framerate", framerate)
    print("nframes", nframes)
    print("comptype", comptype)
    print("compname", compname)
    print()

    perTime = nframes/framerate
    print(perTime,"초 짜리 음")

    # frameRate(고정),바꿀채널이름, arrayType(고정)
    right_channel = rightControl(framerate, right_channel, array_type)
    left_channel = leftControl(framerate, left_channel, array_type)

    printChannel(left_channel, 121)
    printChannel(right_channel, 122)

    wav1.close() # close() : 파일 닫기
    wav2.close() # close() : 파일 닫기

    stereo = 2 * left_channel
    stereo[0::2] = left_channel
    stereo[1::2] = right_channel

    ofile = wave.open(output, 'w') # open(file, mode) : 파일 열기
    ofile.setparams((2, sampwidth, framerate, nframes, comptype, compname))
    ofile.writeframes(stereo.tobytes())
    ofile.close() # close() : 파일 닫기
```



시행착오



=> 소리의 강약만으로 어느 정도의 구현이 된다는 사실을 알아냄

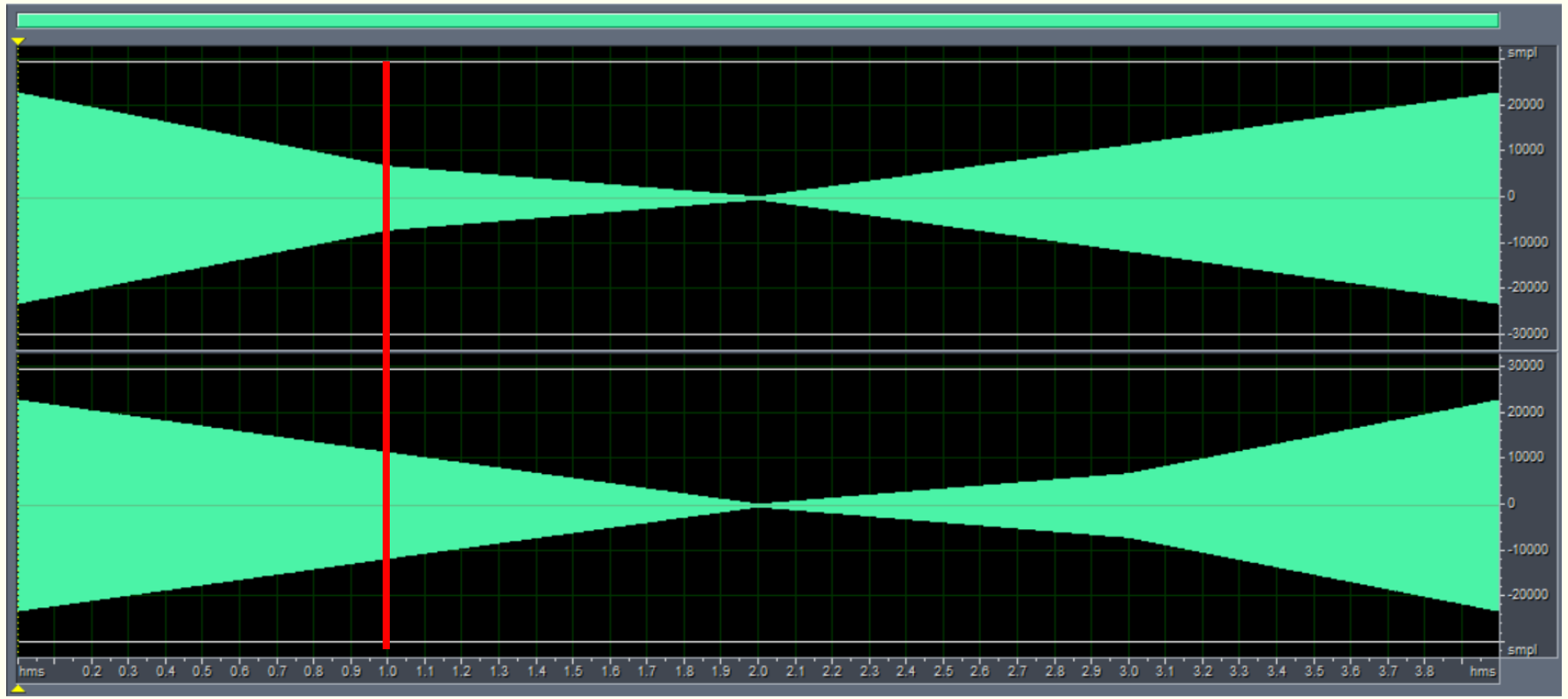


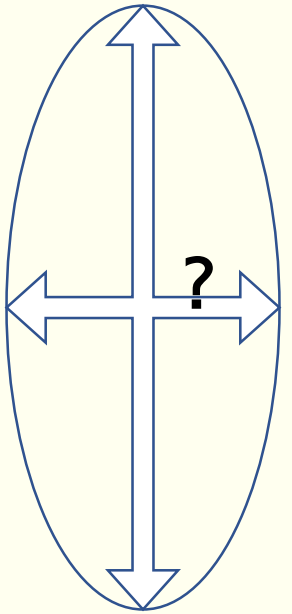
적당히 구간 8개를
나누어 표현

시행착오

L

R





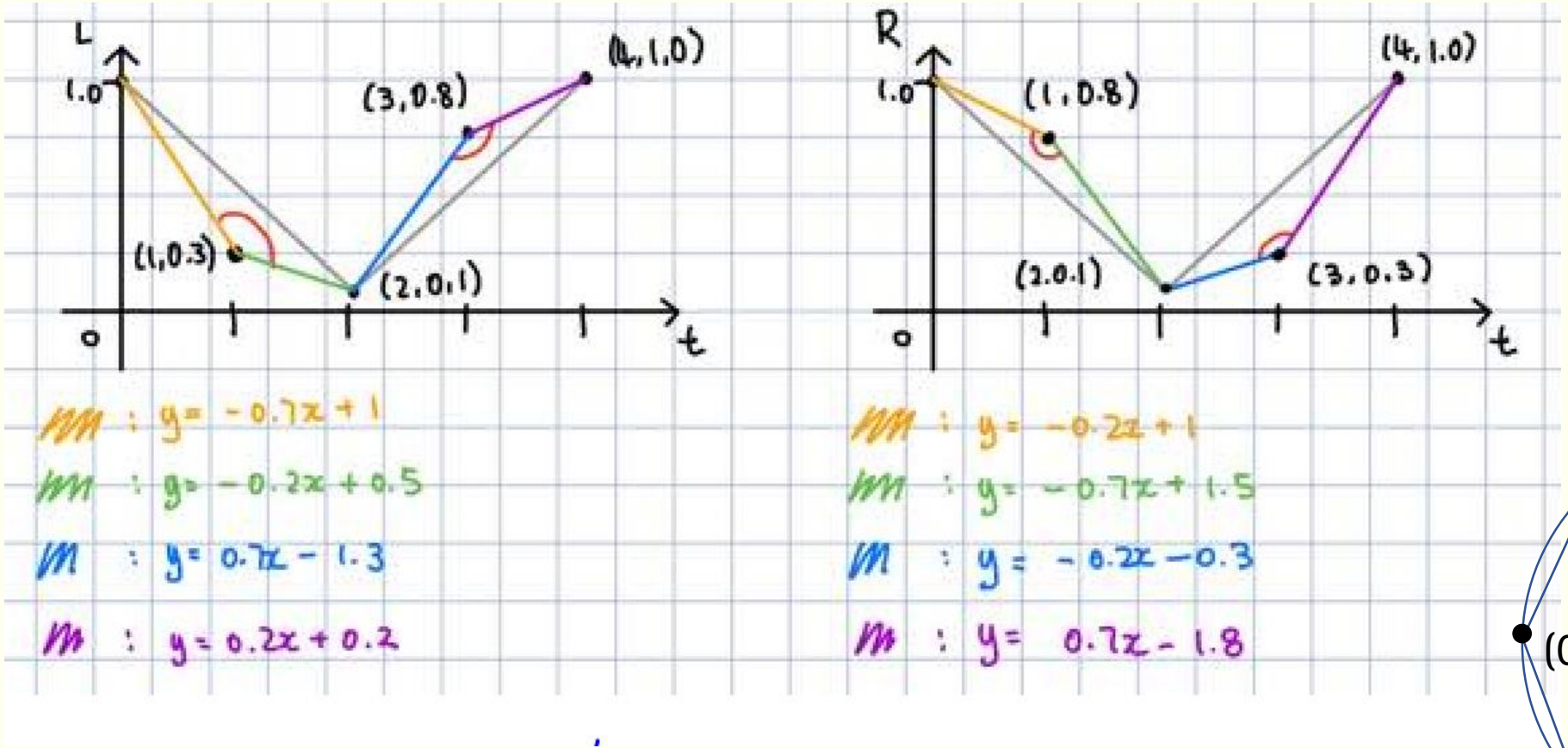
좌우 음량 차이가

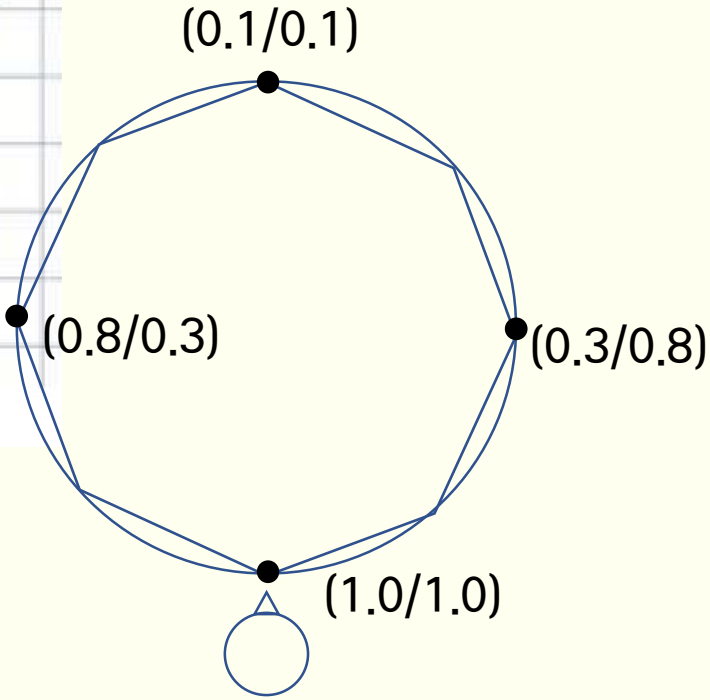
명백하지 않음

⇒ 왼쪽과 오른쪽의 차이가 명확하지 않다는 피드백

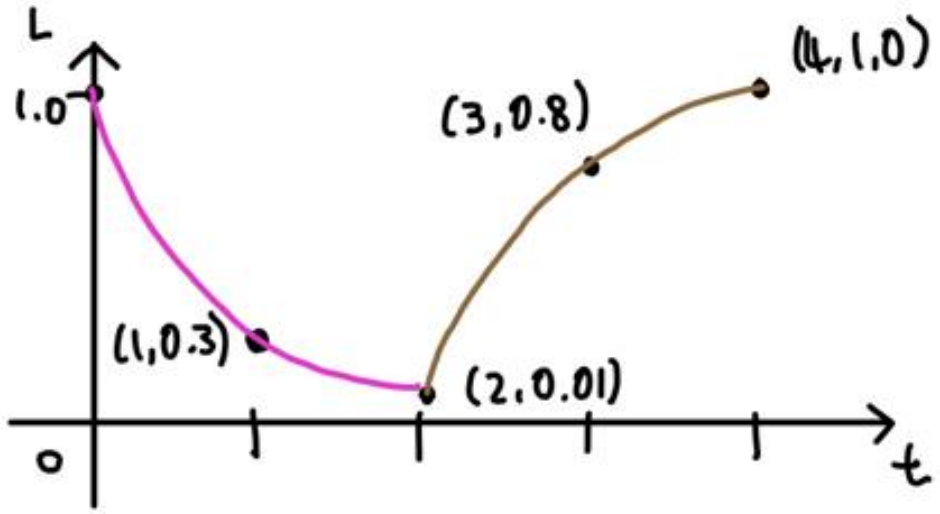
⇒ 좀 더 세분화하고 명백한 차이를 주기로 함

시행착오


L
1.0
(3,0.8)
(4,1.0)
(1,0.3)
(2,0.1)
0
t
R
1.0
(1,0.8)
(4,1.0)
(2,0.1)
(3,0.3)
0
t
y = -0.7x + 1
y = -0.2x + 0.5
y = 0.7x - 1.3
y = 0.2x + 0.2
y = -0.2x + 1
y = -0.7x + 1.5
y = -0.2x - 0.3
y = 0.7x - 1.8

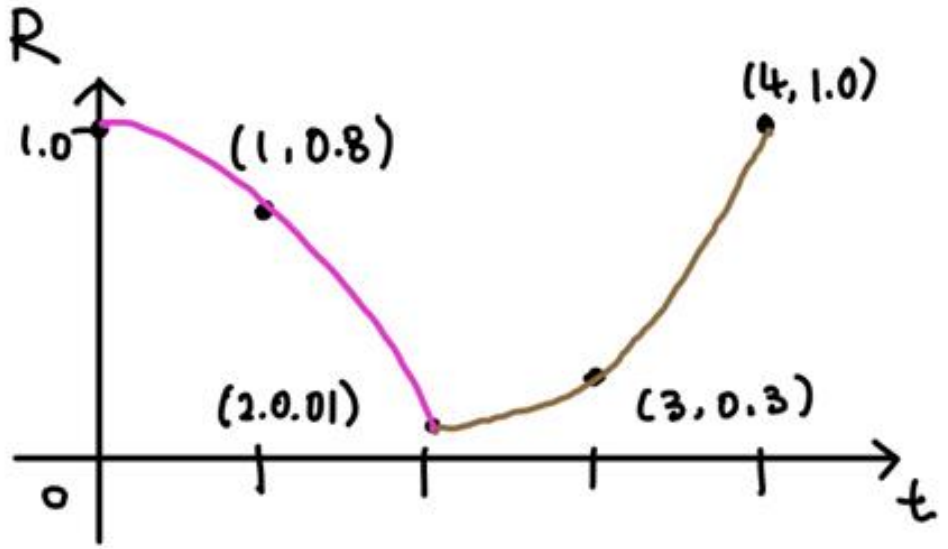

(0.1/0.1)
(0.8/0.3)
(0.3/0.8)
(1.0/1.0)

수행 방법



$m : \cos\left(\frac{\pi}{4}(x+2)\right) + 1$

$m : \cos\left(\frac{\pi}{4}(x-4)\right)$



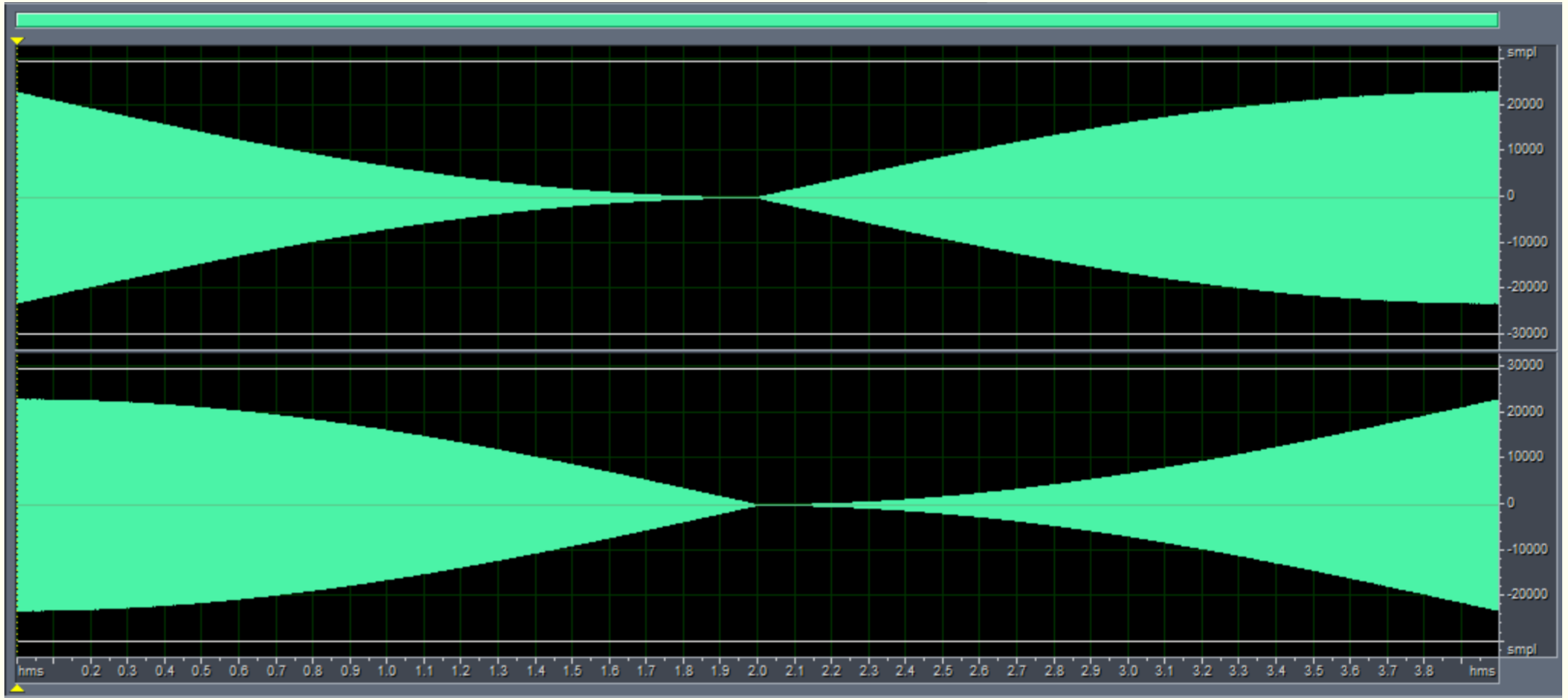
$m : \cos\frac{\pi}{4}x$

$m : \cos\left(\frac{\pi}{4}(x+2)\right) + 1$

수행 방법

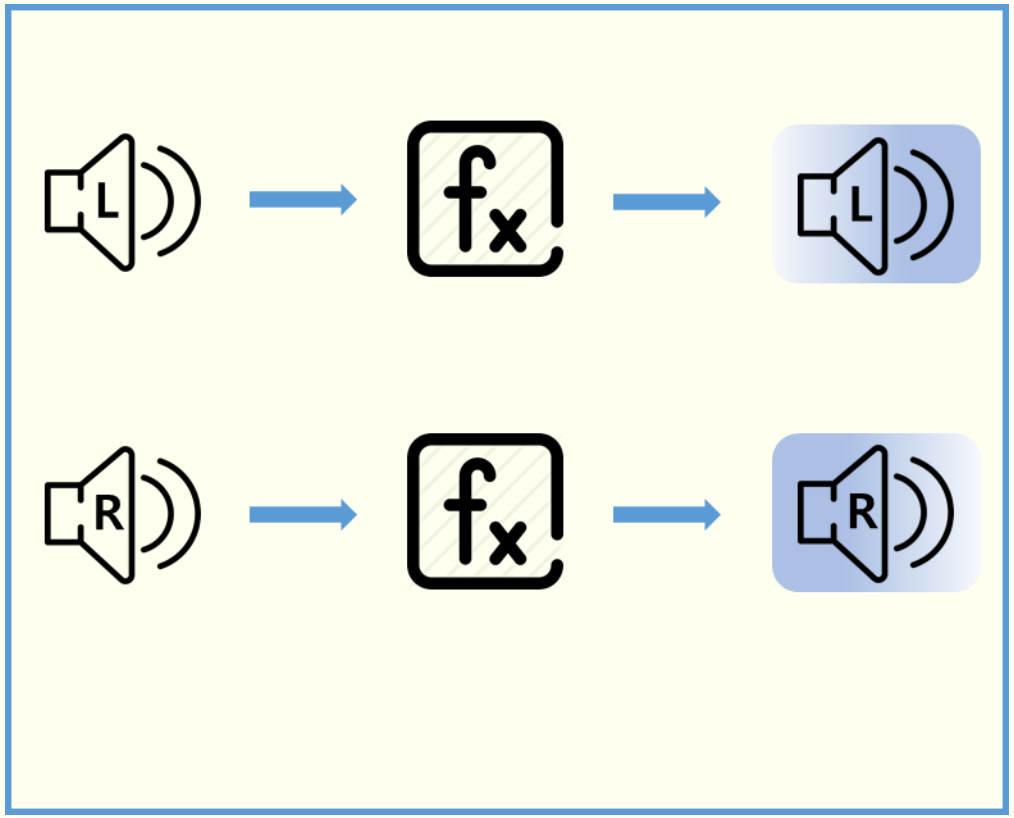
L

R

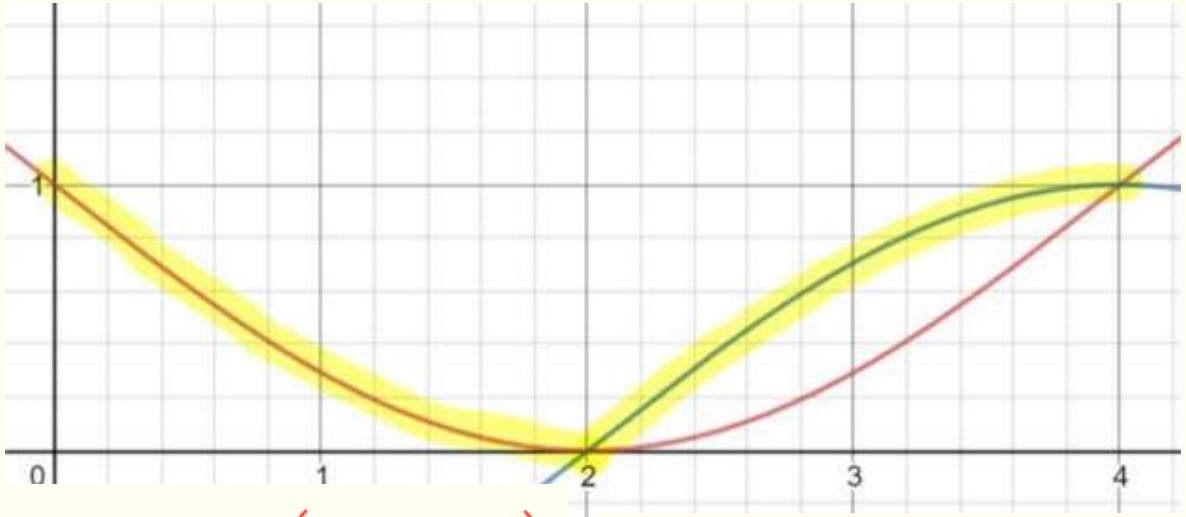

smpl
20000
10000
0
-10000
-20000
-30000
30000
20000
10000
0
-10000
-20000
smpl
hms
0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8
hms

핵심 코드 분석

PART 2

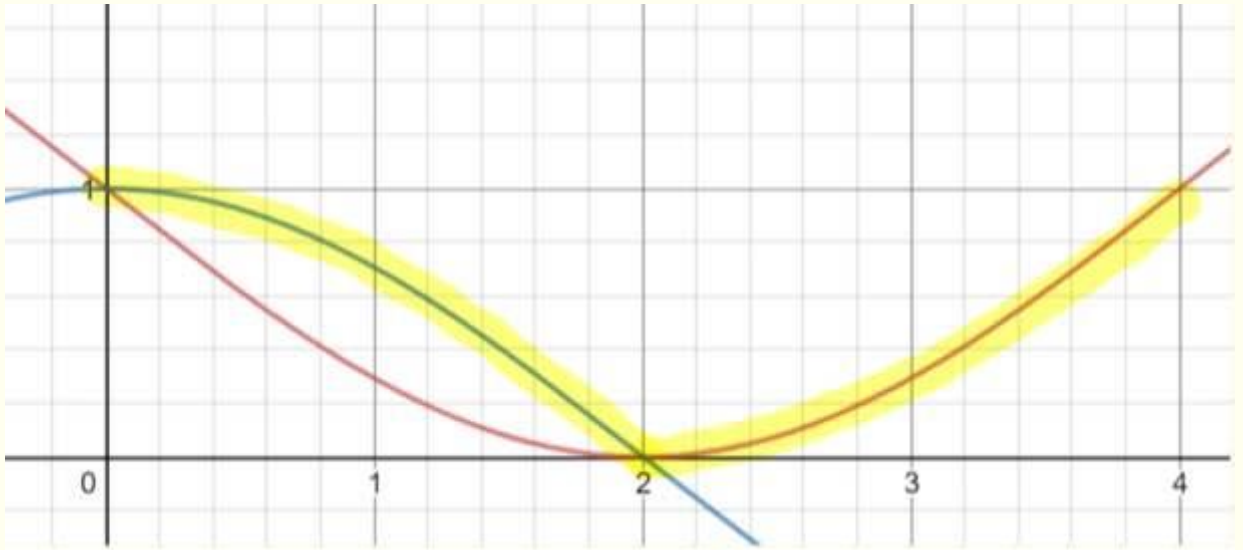


L



0~2초 전 : $\cos\left(\frac{\pi}{4}(x+2)\right)$ 2~4초 : $\cos(\frac{\pi}{4}(x-4))$

R



0~2초 전 : $\cos\left(\frac{\pi}{4}x\right)$ 2~4초 : $\cos(\frac{\pi}{4}(x+2))$

핵심 코드 분석

```
def leftControl (framerate, channel, array_type): # 왼쪽 파일 변환하는 함수
    startFrame = 0 # 파일의 처음부터
    endFrame = framerate*4 # 파일의 끝까지 (참고 : framerate = 16000)

    channelList = channel.tolist() # list로 변경해서 저장 (수정하기 위해)

    for i in range(startFrame, endFrame): # 파일의 처음부터 파일의 끝까지 돌면서
        pos = i/framerate # 현재 위치를 시간으로 나타냄 ex) 0초(=0/64000), 1초, 2초, 3초, 4초

        # 2초를 기준으로 적용하는 수식이 바뀜
        if pos < 2: # 2초 전이면
            channelList[i] = int(channelList[i]*(np.cos((pi/4)*(pos+2))+1))

        else: # 2초 후 라면
            channelList[i] = int(channelList[i]*(np.cos((pi/4)*(pos-4))))

    channel = array.array(array_type, channelList) # list로 변경하여 저장했던 것을 array로 변경해서 저장
    return channel # 변환된 array 반환
```

알게 된 점

- 처음에는 이론으로만 접근하려 해서 어려움을 겪었으나 수치를 대입해 소리를 직접 들어가면서 더 나은 결과를 찾을 수 있었다.
- 소리 세기 차이를 이용하기 위해 계속된 수치를 넣고 해보니 음량 차이만으로 입체 음향이 구현되는 사실을 알았다.

아쉬운 점

- 정확한 정답이 없어 귀로만 의존하게 되어 아쉬웠다.
- 최선의 방법으로 구현은 했지만 이 방법이 정확한 방법인지, 더 나은 방안은 없는지 고민하게 되었다.

감사합니다.