

Forum:[Hack Forums](#)

Thread: [Audio Compression Algorithm and In-depth explanation](#)

MiguelSanchez - February 13, 2022, 11:04 am

Compression today, I'll write up the decompression tomorrow, if I've explained this properly you shouldn't need it.

Extrema Compression V0.1

Introduction:

ExtremaCompression is a low-loss compression method for audio.

Unlike MP3, OGG and other common standards the results still vary widely proportional to frequency. Higher frequency oscillations will be encoded with minimal loss, but also minimal compression, while low frequencies, will compress very small but the quality may deviate from the original product.

How does it work:

Possibly the most intuitive form of compression, the algorithm simply the wave into segments from a high peak to a low peak or low peak to a high peak, finds the closest matching shape (out of 255 precomputed lines/curves) and replaces each segment with a 5 byte approximation segment.

The Filetype:

The intermediate file is simply an encoded bytearray to be read 5 bytes at a time.

Each 5 byte sample can represent two things, a curve, or silence.

If the first byte in the sample is a 0 it represents silence, any other value and it's a curve.

-Silence

Byte 1: Always 0

Byte 2 to 5: A 4-bit unsigned-long representing how many samples to remain silent.*

-Curve

Byte 1: Unsigned-char, ordinal number representing which shape is used, not zero-indexed (i.e. 1 is the first item, 2 is the second.)

Byte 2 & 3: Unsigned-Short representing the length of the curve in samples*

Byte 4 & 5: Signed-Short representing the position of the speaker at the end of the curve. The start of the curve will be the end of the last curve.

*Sample rate is 44100. That means half a second is 22050samples. 2 seconds 88200samples, and 10ms is 441 samples. So a 440Hz note should oscillate 440 times every 44100 frames (every 100.22 frames.)

Compression Step by step:

First we:

- initialise everything needed for the loop,
- and

- load the first frame to determine the start direction.

Code def RealtimeCompress(inputFile,outputFile):

WaveFile=openwave(inputFile)

```

samplewidth=WaveFile.getsampwidth()
cstruct="<l" if samplewidth==4 else "<h"
magnitude=2**(8*samplewidth-1)-1
channels=WaveFile.getnchannels()
print("sample width: %s"%samplewidth)
print("channels: %s"%channels)
print("magnitude: %s"%magnitude)

```

Open the file inputFile, define which cstruct to use depending on the sample width of the wave, unsigned-long for PCM32, unsigned-short for PCM16. (optional: Print the wave properties)

```

Code YThreshold=50/magnitude
OutFile=open(outputFile,"wb")
firstframe=WaveFile.readframes(1)[0:samplewidth]
first=unpack(cstruct,firstframe)[0]/magnitude
lastdirection=1 if first>YThreshold else -1 if first<-YThreshold else 0

```

#1 is up, -1 down, 0 still-set the Ythreshold (the minimum amount of change required to be considered a movement, until this amount is surpassed the wave is considered still)

- Initialise the output file,
- Read and unpack the first sample from the input file.
- if the first frame exceeds the threshold set the lastdirection to 1 or -1, else lastdirection=0 #1 is up, -1 is down, 0 is still

```

Code last=first
current=[first]
n=0

```

while 1:-Initialise the variable last (the last value) as the previous frames value (defined as first)

- Make a list containing only the first value, this will be used to store the current segment.
- Initialise a variable to keep count of how many frames have elapsed (I called it n) and start the loop.

The loop interior:

```

Code frame=WaveFile.readframes(1)[0:samplewidth]
if frame=="b":
    #END OF FILE
    breakLoad a single frame.

```

If no frame is available break the loop

```

Code else:
    currentval=unpack(cstruct,frame)[0]/magnitude
    direction=1 if currentval>last+YThreshold else -1 if currentval<last-YThreshold else 0
    #determine directionIf the frame isn't empty then unpack it into the variable currentval

```

divided by magnitude

if currentval exceeds last by at least YThreshold, set the direction to 1, otherwise if last exceeds currentval by more than YThreshold set direction to -1. Otherwise its 0.

```

Code if direction==lastdirection:
    current.append(currentval)
    last=currentval

```

now that the direction has been resolved, we compare it with the last direction,

If they're the same, append currentval to current (the current segment)

and set last to currentval and proceed to the next iteration.

Code else:

```
n+=1
##MAKE SECTION
if lastdirection==0:
    #ADD SILENCE BYTES
    silencebytes=pack("B",0)+pack("L",len(current)) #holobytes
    OutFile.write(silencebytes)
    del silencebytes
```

-In the event the direction has changed since the last sample (the else condition above) then

-Increase N by 1 (there's a new output)

-If the last direction was 0 (stillness) then make some bytes representing the just elapsed silence.

First the value 0 to signify it's silence and not a curve <1 byte unsigned-char> then len(current) <4 byte unsigned-long>

-write the silence bytes to OutFile

Code else:

```
#ADD SECTION
curvenum,score=Curves.closest(current)#compareCurve
section=(curvenum,len(current),current[-1])
#write-out section
curvenum=pack("B",section[0]+1)
xval=pack("H",int(section[1]))
yval=pack("h",int(section[2]*(2**15-1)))
curvebytes=curvenum+xval+yval
OutFile.write(curvebytes)
del curvebytes,xval,yval,curvenum,section,score
```

If it's not a silence it's a curve so run Curves.closest(current) to compare the contents of current with all 255 precomputed curves, to find the closest match. (This can be very slow, so we often just check 8 to 64 evenly spaced points along the curve, depending on the length)

Our whole section is as simple as (The closest shaped curve <number 1..255>, the length of the current segment, and where it ends)

These 3 values are packed to an unsigned char, an unsigned short, and a signed short respectively, and in that order, and written to outfile, after that the resources are all freed from python.

Code lastdirection=direction

current=[currentval]

last=currentval Before the next iteration replace lastdirection with direction, replace current with [currentval] and replace last with currentval.

Code if n==1024:

OutFile.flush()

n=0 Every 1024 frames flush the OutFile (everything we sent to file.write() will actually be visible in the directory now, if you want a peak before it's done)

Code OutFile.close()

WaveFile.close()

return 1 Once the loop is broken by the end of file, close both files, and return 1 to indicate a success.

I need a break. It's 5am, I'm off to bed. Obviously the next step is to import it somewhere faster. I'm getting great results though. I'll elaborate tomorrow. This took way too long to type. I hope someone reads it.

Much love to anyone making it this far <3

There's a lot more to this, this is part 1.
More to come for sure, not all in python but the prototypes will usually be.

MiguelSanchez - February 19, 2022, 4:02 am

Part 2:

Decompression:

Luckily for us the decompression phase is much simpler than the compression phase. It would have been simpler regardless but

We create a new wave file for the output, we simply need to have a while loop read through the file five bytes at a time, if the first of each 5 is a zero add a silence for as many frames as declared in the remaining 4 bytes.

If the first byte is non-zero 1 to 255, add curve 1-255 stretched to a length of bytes 2&3, and ending on the height encoded in bytes 4&5.

Then pack add write these bytes to the wave output.

Decompression Step by step:

Code def Decompress(FileIn,WaveOut=None):

if WaveOut==None:

WaveOut=FileIn+"Dcmp.wav"

wavobj = waveopen(WaveOut,'w')

wavobj.setnchannels(1) # mono

wavobj.setsampwidth(2)

wavobj.setframerate(samplerate)

magnitude=2**15-1 First we define the function, with two inputs, one of which is optional, and defaults to None.

If WaveOut has defaulted to none, then it's replaced with FileIn+"Dcmp.wav".

Open a new empty wave and set it to a single channel, 16bit samples, with a framerate of 44100, like the rest of the project.

Since the Y value, of each sample is only two bytes, the wave needs only be PCM16 and the magnitude $2(2^8-1)-1$.

Code InFile=open(FileIn,"rb")

last=(0,0)

while 1: Open the input file.

Declare the end of the last sample as (0,0), this is simply setting the start position of the wave, don't let the variable name "last" throw you off, it doesn't make sense here, but it will become apparent why I chose to name it such.

Start the loop:

Code head=InFile.read(1)

tail=InFile.read(4)

if head==b"":

break

curvenum=unpack("B",head)[0] Read in 1 byte from the file, as the sample's head, and the other four as its tail. If the file is empty, the head will be an empty byte string and we're done, so we can break.

But let's not get ahead of ourselves, unpack head as an unsigned-char named curvenum.

Code if curvenum==0:

```

x=unpack("L",tail)[0]
val=last[1]
data=pack("<h",int(val*magnitude))
for i in range(x):

```

wavobj.writeframesraw(data)As we know by now, if curve num is 0, it's not a curve at all, but silence, simply take the end height of the last curve (last[1]) multiplied by the wave's magnitude, pack it and then write it out to the wave for as many frames as declared in the tail, of course we must first unpack the tail into an unsigned-long (called x in the example)

Code else:

```

x,y=unpack("Hh",tail)
y/=magnitude
c=Curves.Curves[curvenum-1]
h=y-last[1]Otherwise, if the head is any number other than 0, it means a curve is to be
placed,
beginning on last, and (ending on last+x,y).

```

First unpack the tail, as a signed-short, followed by a an unsigned-short.

Assign these values to x and y respectively.

Divide y by the magnitude, to normalise the input so all values lie between -1 and 1.

Now load the curve. (We must deduct 1 from the curve num, because python lists are zero-indexed, whereas the curves in our file are 1-indexed (0 is silence instead)

Declare the height (named h) as the end position, minus the start position.

Code for i in range(x):

```

r=i/x
val=last[1]+c[int(r*1024)]*h
data=pack("<h",int(val*magnitude))
wavobj.writeframesraw(data)
last=(last[0]+x,y)
#resizecurve
return 1

```

loop i for the the length of the curve "x".

set r to i/x (so it's 0 at the beginning of the curve, and 1 at the end)

The current value is curve[int(r*1024)] multiplied by the height, meaning we stretch the 1024 frame curve into x frames, and multiply the value of each point by the height of the curve. Multiply the value by the desired output magnitude ($2^{(8*(bytes-1))}-1$ for signed) and write the packed data to the wave.

Once you've written in the entire curve, update last to (last[0]+x,y)

When done return 1 for success.

Sky - February 20, 2022, 6:49 am

Very cool thread. Thank you for taking the time to make this.