21M.380 Music and Technology
Sound Design

Sound design exercise 2 (ex2)
Steam whistle

Due: Wednesday, March 30, 2016, 9:30am
Submit to: MIT Learning Modules ‣ Assignments
5% of total grade

1 Instructions

Synthesize the sound of a steam train's whistle in Pd vanilla from scratch, following the guidelines below. No recorded sounds may be used, except for analytic purposes. While this assignment focuses on the implementation stage of the sound design process (cf., figure 1), the following instructions will first discuss the preceding stages to provide context.

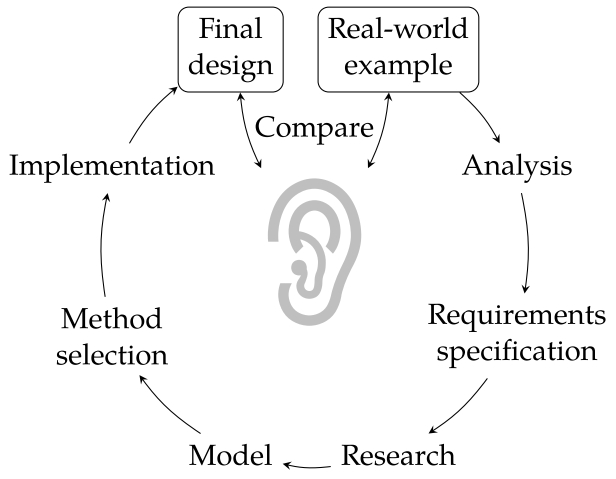

Figure 1. Stages of the sound design process (after Farnell 2010, figs. 16.7, 16.1)

2 Find a real-world example

Look for a representative real-world example of a steam whistle sound online, which will serve you as a reference throughout the design process. Identify a suitable recording according to the following two criteria.[1]

[1] You might have to trade off these two criteria against each other.

Cliche factor: Look for the sound that in your opinion best embodies the sonic characteristics and behavior of a steam whistle. In other words, look for the most cliche whistle that you can find.

Suitability for analysis: To facilitate later analysis, the recording should include as few sounds as possible from sources other than the whistle itself. Try to find a good-quality recording without excessive background noise.

3 Analysis

Once you have found a suitable recording, find a way to convert it to an uncompressed audio file (if needed), and analyze its spectrogram using the *Sonic Visualiser* software package. A visual and aural analysis will probably reveal the following:

- The sound of a steam whistle contains both harmonic (pitched) and inharmonic (noisy) components. It is 'airy yet pitched'.
- Pitch and loudness are somewhat correlated. As the whistle gets louder, its pitch also rises.
- On a more detailed level, harmonicity might also be correlated with loudness. Soft sounds tend to be 'airy', whereas pitch becomes more pronounced as the whistle gets louder.

Try to also read the following information from the spectrogram:

- How many harmonics does the whistle roughly generate? A single one? Dozens? Somewhere in between?
- Does the whistle create *all* harmonics, or is it limited to a certain set (e.g., only odd) harmonics? What can this tell us about the whistle's phsyical properties?[2] Are specific harmonics particularly prominent or weak?
- Are the harmonics perfectly in tune, or to their frequencies deviate from the perfect integer multiples of the fundamental?
- Do all harmonics behave similarly over time? Do some come in later and/or fade out earlier? Does this follow a pattern?

[2] Remember the different acoustics of open vs. closed pipes.

4 Requirements specification

To reduce complexity, it is important to define our design's requirements early in the process.

4.1 Sonic requirements

For now, we will deliberately limit ourselves to generating only the sound of the whistle itself, disregarding any environmental acoustic effects such as echoes, background noise, other train sounds, Doppler effect, etc.[3]

[3] Our goal is to create a dynamic sound object which can later be embedded into a more complex acoustic scene that includes such effects, but these are beyond the scope of the current assignment.

4.2 User interaction

Our sound object needs to provide only limited possibilities for user interaction. Specifically, such interaction should be limited to a single button (i.e., a `[bang(` in Pd), which allows the user to trigger the sound of the whistle. To make things more realistic, however, the sound of the whistle should change slightly each time we trigger it, which corresponds to our real-world experience of such a whistle (cf., section 6.3 for details).

5 Research

While research on how steam whistles are actually made can certainly strengthen our design, I suggest that for now you try how far you can take things without getting too concerned with such details. Let's try to create a model of a steam whistle that is primarily grounded in our earlier analysis of its sound.

6 Model

6.1 White noise through harmonically tuned bandpasses

The key to understanding the sound of a steam whistle is the realization that its pitched (harmonic) components cannot be separated but rather *emerge* from its noisy (inharmonic) components as harmonic oscillation builds up inside the whistle.[4] Rather than modeling the whistle through a bunch of harmonically tuned sine wave oscillators on the one hand, plus a white noise source on the other, it is therefore preferable to model it as a single noise source, which is filtered by a series of bandpass filters whose center frequencies are tuned harmonically. The resulting sound's degree of harmonicity can be controlled through the filters' *quality* (or *Q*) *factor*. A higher Q factor will result in a more sharply tuned bandpass filter, which will result in a more pronounced pitch.

[4] For acoustic purposes we can understand a whistle as a pipe.

6.2 Correlation between amplitude and frequency

In our earlier analysis, we discovered a correlation between a whistle's loudness and its pitch. For our model, this means that both, the whistle's overall amplitude and the center frequencies of all bandpass filters can be controlled by a single envelope. While the numeric output range of that envelope has to be scaled to suitable amplitude and frequency ranges, respectively, its temporal behavior is identical for either application.

6.3 Randomization

Our experience of a real-world steam whistle suggests that while some parameters of its sound production are relatively static (e.g., spectral composition, correlation between amplitude and pitch), other details remain more unpredictable, owed to the inconsistent behavior of a human operator. This includes the overall duration, the length of attack and decay phases, and the exact amplitude (and therefore also pitch) profile, all of which will be different each time the whistle is triggered. Although we do not need to model these inconsistencies in detail, one important aspect of our model will be to make the whistle sound dynamic. In other words, your patch should produces a slightly different sound each time it is triggered. This can be achieved by introducing an element of randomization that mimics the unpredictable behavior of a human operator.

6.4 Advanced details (for ambitious students)

The above aspects of our model should be sufficient to create a fairly convincing steam whistle sound, and your implementation should focus on these. If after that, you still have time and energy to spare, you could try to strengthen your design by implementing a more detailed model.[5]

[5] While this might make the resulting sound more realistic, it is important to evaluate whether the gain in realism actually justifies the increased complexity. In Einstein's words, you should attempt to make things as simple as possible, but not simpler.

- For example, you could attempt to mimic the correlation between harmonicity and loudness that we have observed in our analysis. We could do so by controlling the Q factors ouf our bandpass filter bank dynamically, using the same envelope that we are already using for amplitude and filter center frequencies.
- Another simplification that we have made is that all harmonics in our whistle behave identically over time. A closer analysis of a real-world whistle might, however, reveal that the number of harmonics correlates with amplitude, i.e., that more harmonics appear as more energy is invested. Modeling such behavior

would require to control the individual harmonics with separate envelopes.

7 Method selection

The method of shaping sound by filtering a broadband noise source to reveal the desired spectrum is generally known as *subtractive synthesis*, which we will employ for the implementation of our whistle sound.

8 Implementation

- Use Pd vanilla rather than Pd extended for your implementation. No objects from Pd extended may be used.
- You can use subpatches as needed to make your code more readable, and you may also use abstractions where they are beneficial to encapsulate repeating behavior and avoid duplicate code.
- The user interface should be limited to a single `[bang(` button that triggers the steam whistle sound.
- The main patch that needs to be started to test your whistle must be named `main.pd`.
- The only sound-generating[6] Pd object that you may use is `[noise~]`. Try and get by with a single instance of this object, using `[s~]` and `[r~]` to send and receive its output to other parts of your patch if needed.
- To randomize the sound of you whistle, you can use the `[random]` object. Should you need to generate random numbers at audio rather than message rate, you can use the above `[noise~]` object's output and scale it with `[*~]` and `[+~]` to fit your needs. The `[pack]` object will be useful to combine multiple random numbers into a list of parameters that you can send to `[vline~]`, and `[t b b]` might be useful to guarantee the order in which such parameters are processed by `[pack]`.
- You can use any sound processing object available in Pd vanilla, but the `[vcf~]` object, a bandpass filter with variable center frequency, will be particularly useful. Prefer `[vcf~]` over `[bp~]`, as the latter's center frequency can only be changed at message, not signal rate. As the Q increases, the overall output amplitude of

[6] In computer music, we distinguish between sound *generators*, which produce sounds 'from nothing' and ound *processors*, which alter an existing signal.

the filter will decrease (and quite drastically so), so you should expect to amplify the output of [vcf~] with [*~], which might require double-digit multiplication factors (be careful with your ears). It is probably sufficient to find a suitable Q factor by trial and error using a single filter band. Once you've found a number that you like, try hardcoding it as the first creation argument into every [vcf~] object of your filter bank. That should do the trick, and if it does, there shouldn't be any need to use [vcf~]'s third inlet (which changes Q dynamically).

- The envelope(s) controlling amplitude, filter center frequencies, and possibly Q factor, are best implemented through a [vline~] object. Rather than creating multiple instances of this object with nearly identical parameters, scale the output of a single [vline~] object to fit your needs.[7] Note that you can scale numeric ranges also in the signal, not just the message domain by using [*~] and [+~].[8] Do *not* use [snapshot~] to permanently convert signals to messages!
- You should extensively document your Pd patch(es) with comments. Keep in mind that this is our first Pd-related sound design assignment where different students might find very different solutions, so make sure I can retrace your design!

[7] Remember that the temporal development of all amplitudes, frequencies (and possibly Q factors) are identical, unless you deliberately decide to control the different harmonics separately, which I do not generally recommend for this assignment.

[8] To simplify debugging, you can first test your scaling operation in the message domain and then reimplement it with tilde objects.

9 Fine-tuning

Remember to leave enough time for comparing your implementation against the real-world recording. Fine-tuning will often produce better results than revising the underlying model. The Q factor of your bandpass filters is a good parameter to trade off 'pitchiness' vs. 'airiness' of your whistle sound. Another characteristic signature of any sound is its behavior over time. If you get the fade-in and fade-out times right, as well as the amplitude and frequency ranges of your envelopes, the finer details of spectral composition will become surprisingly unimportant for making your sound convincing.

10 Assessment criteria

Sonic realism and liveliness of your whistle sound

User interaction design How easy is it for an outside user to run your patch?

Documentation How well is your code documented? How easy is it for an outside developer to understand how your project works?

11 Submission format

Submit a single `.zip` archive that contains any `.pd` files required to run your assignment (and nothing else). Unzipping your archive, opening your `main.pd` patch in Pd vanilla, and hitting `[bang(` should result in a working example; don't expect the user to move files around before running your example. Make sure to test this on your own machine before submission.

References and useful resources

Farnell, Andy (2010). *Designing Sound*. Cambridge, MA and London: MIT Press. 688 pp. ISBN: 978-0-262-01441-0. MIT LIBRARY: 001782567. Hardcopy and electronic resource.