INFO0062 - Object-Oriented Programming

Project: creating artificial reverberation with your library

Jean-François Grailet

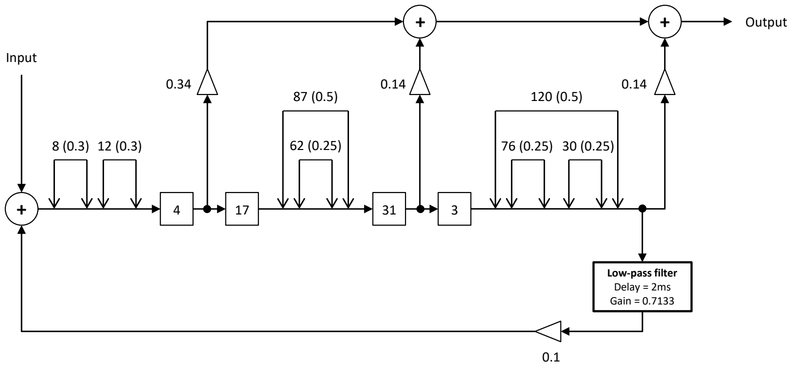
University of Liège

Faculty of Applied Sciences

Academic Year 2019 - 2020

Artificial reverberator

Artificial reverberator (large room)



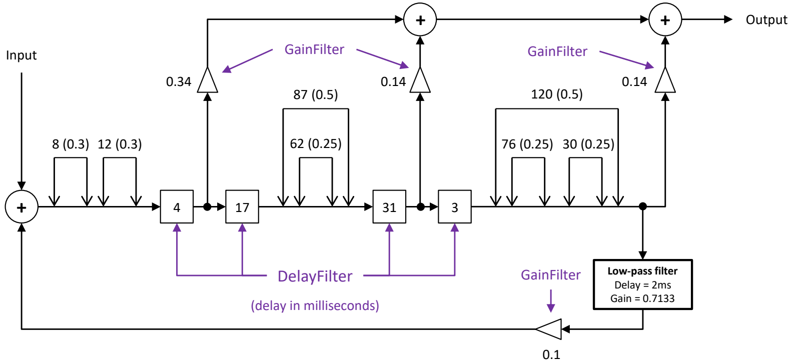
Artificial reverberator (large room) (II)

- This filter is designed to simulate the reverberation of a large room.
  - I.e., the original audio content should sound as if played in a concert hall.
  - Designed by William G. Gardner in 1992 for his master thesis. [1]
- If properly implemented, your filtering library **can** build this filter.
  - But first, make sure you can at least build a simple all-pass filter.
  - The diagram of an all-pass filter is showed on the project webpage. [2]
  - WAV files processed with an all-pass filter or the reverberator are provided too.
- This filter is complex but its *building blocks* are simple.
  - Mostly relies on combinations of all-pass filters and delays.
  - Also involves a single low-pass filter (cf. next slides).

[1] http://www.ee.columbia.edu/~dpwe/papers/Gardner92-virtroom.pdf
[2] http://www.run.montefiore.ulg.ac.be/~grailet/INFO0062_proj_19-20.php

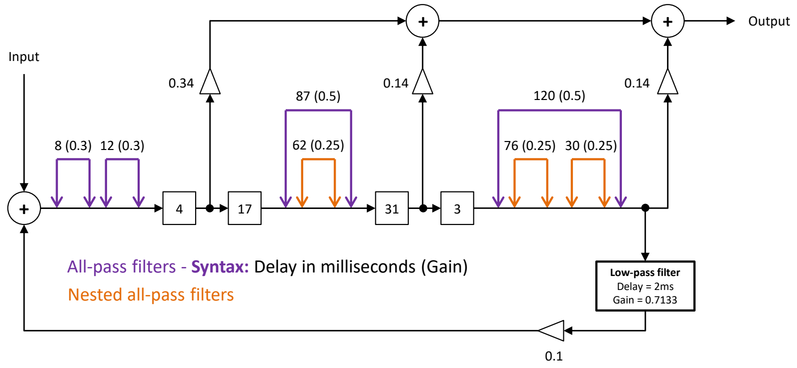
Components of the filter

- Basic filters are, of course, involved in this.
- The difficult part is to handle the all-pass filters.



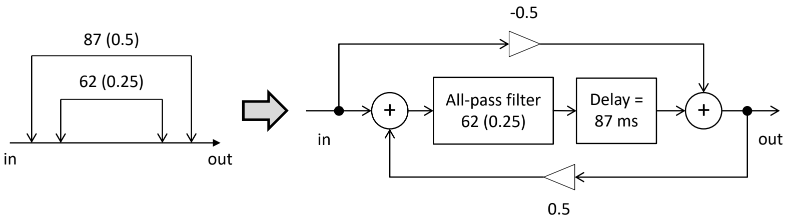
Components of the filter (II)

- In the diagram, all-pass filters are symbolized by double arrows.
- They are annotated with their delay (integer value) and gain (real value).



Components of the filter (III)

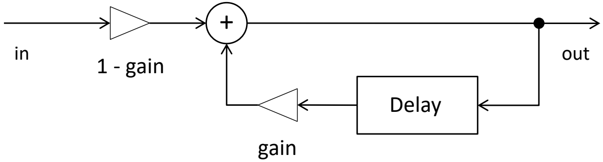
- As a consequence of the former notation, there are nested [3] filters.
- Next figure shows how two double arrows can be translated as a block diagram.
  - The nested filter is put before the delay block of the encompassing filter.



[3] FR: imbriqués

Components of the filter (IV)

- You will also need one *low-pass filter*. [4]
  - This filter attenuates higher frequencies.
- The next figure shows how you can build this filter with your library.



[4] `https://en.wikipedia.org/wiki/Low-pass_filter`

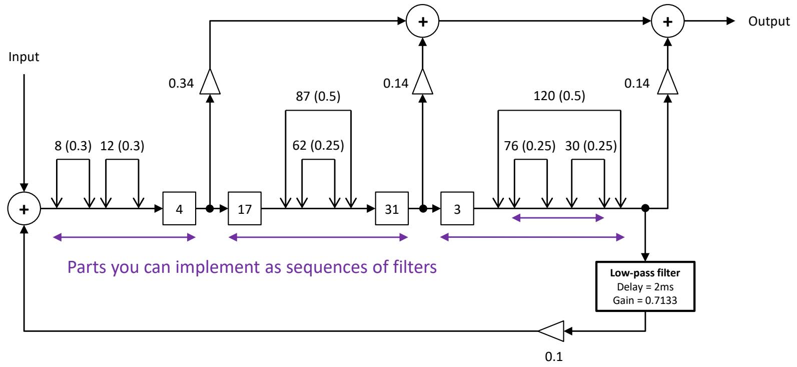
Tips

Tips to implement the reverberator

- More than ever, you need to **proceed step by step**.
  - Trying to build the filter right away will be tedious...
  - ... and not necessarily successful.
  - You will also have dozens of basic blocks to handle at once.
- In particular, see how the filter can be split in several parts.

Tips to implement the reverberator (II)

- You can first try to build each sub-sequence of filters.
- For instance, start with the two all-pass filters (+ delay) on the left.

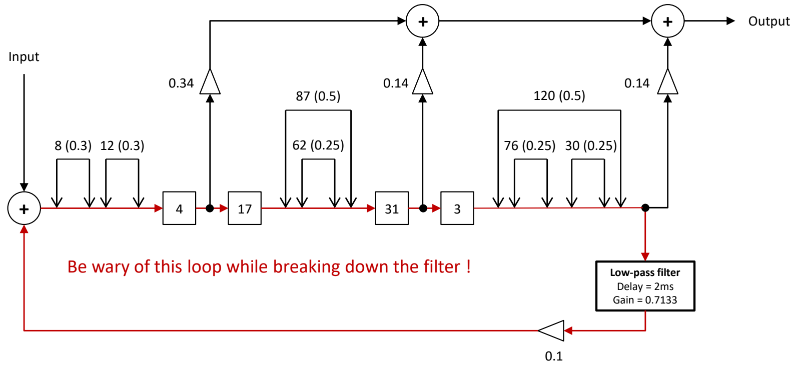


Tips to implement the reverberator (III)

- Consider creating auxiliary classes for this.
  - There are a total of 7 all-pass filters.
  - You might not want to build each one from scratch.
- This is also another opportunity to put some OO concepts to practice.
  - E.g., you could extend `CompositeFilter`.
  - Or use constructor polymorphism.
  - Or create classes designed to provide *templates* of filters.

Avoiding the loop pitfall

- Pay attention to the feedback loop !
- At least one `DelayFilter` should explicitely appear in the full filter.



Avoiding the loop pitfall (II)

- **Reminder:** it is assumed each loop contains at least one `DelayFilter`.
- We don't ask your library to check if a `CompositeFilter` behaves as one.
  - We assume each loop explicitely has a `DelayFilter` in it.
- If you are nesting parts in `CompositeFilter` objects, leave one delay out.
  - E.g., put the last `DelayFilter` only in the final montage.
- Otherwise, your library might consider the whole filter invalid.

About execution time

- Obviously, processing a file will take much more time with the reverberator.
- If creating reverberation takes several dozens of seconds: **don’t worry**.
  - The total execution time isn’t a huge concern here.
- For reference:
  - Suppose you can process a WAV file to add echoes in roughly 1 second.
  - Processing it with a reverberator might take a bit less than 1 minute.
  - More powerful computers can do the same job in around 20 seconds.

Regarding submission

What about submission ?

- Suppose you succeed in building this filter.
- First: congratulations !
- Second: you can add your additional work (and classes) in your final archive.
- However, if you do so, we will ask you two things.
  - You need to create a `README.md` file or expand your current one.
  - We will also ask you to slightly expand your `Demo` program.

What about submission ? (II)

- **About the `README.md` file**
  - Explicitely state you built the reverberator **at the first line**.
  - List all files that are *exclusively used* for your reverberator.
  - List them in a convenient format (e.g., one file per line).
  - This will help us to focus on the main part of the project while reviewing yours.

What about submission ? (III)

- **About the `Demo` program**
  - Add a `Reverb` mode.
  - This mode will be invoked with the following command:

```
java -cp bin:audio.jar Demo Reverb Source.wav Reverberated.wav
```

  - **The regular behaviour of your `Demo` program must still be implemented.**
  - I.e., we should still be able to run this command:

```
java -cp bin:audio.jar Demo Source.wav Echo.wav
```

  - Make good use of `args.length` to this end.