Implementation of Digital Synthesis in Functional Programming

Giorgi Botkoveli
Cenikj Nikola
Abdin Hossameldin
Beka Grdzelishvili
Evan Sitt
Sylaj Tringa
Su Xiaotian
Szumi Xie
Viktória Zsók
Tsinadze Zurab[1]

[1]Eötvös Loránd University, Faculty of InformaticsDepartment of Programming Languages and CompilersH-1117 Budapest, Pázmány Péter sétány 1/C., Hungary
zsv@inf.elte.hu, Sitt.Evan@gmail.com

SZÉCHENYI 2020
HUNGARIAN GOVERNMENT
European Union
European Social Fund
INVESTING IN YOUR FUTURE

OUTLINE

- Motivation
- Background
- Implementation
    - Wavetable Lookup Synthesis
    - Additive Synthesis
    - Envelopes
    - MIDI Input
    - PCM WAV Output
- Results
- Conclusion
- Future Work

MOTIVATION

- **To extend the possible applications that can be created with the pure lazy functional programming Clean.**

  The project aimed to tackle an application that is completely dominated by C++ and JAVA frameworks.

- **The language offers very good abstractions tools for sound generation implementation.**

  Higher order functions, lazy evaluation, and algebraic data types are especially useful for processing digital signals and the musical paradigm.

- **Typically accomplished within a C++framework**

  However, the methods can be non-intuitive and non-conducive to the musical paradigm.

- **Reinforce, develop, and expand functional programming skills of students.**

  Demonstrate the applicability of Functional Programming skills.

BACKGROUND

- **Digital Synthesis**

  First pioneered in 1957 with a punch card controlled analog synthesizer using 750 vacuum tubes.

  It is still continuously innovated by the music industry. Digital synthesizers use the power of microprocessors to replicate analog synthesis.

- **Cross Disciplinary**

  Extensively popular today across many disciplines, including music and telecommunications.

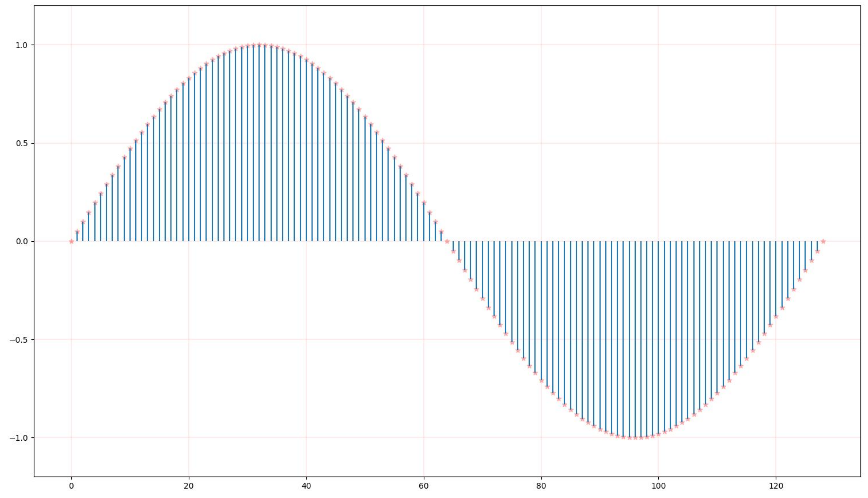
WAVETABLE LOOKUP SYNTHESIS

- **Storing a single period of a certain waveform.**

  Storing precomputed values of a single period of a waveform in an constant time access array to remove the cost of repeated calculations.

- **Table size based upon Sampling Rate and Frequency**

  To accomodate a sampling rate of 44,100hz and the lowest human audible frequency of 20hz the project uses a table of 2205 values.



WAVETABLE LOOKUP SYNTHESIS

- **The sound produced can be harmonically changed via access points.**
  Determined by frequency, it is simple to generate a sound by simply modifying which indices to access.

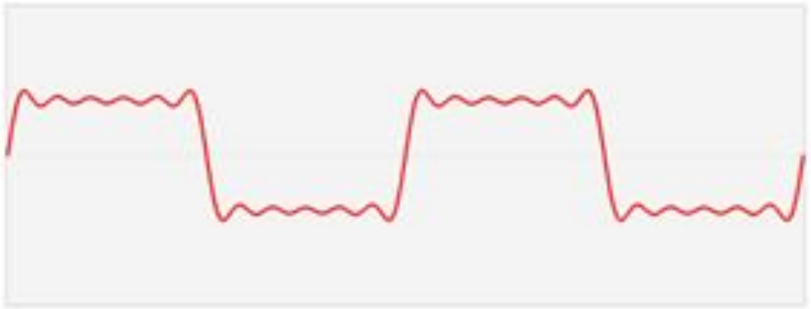
- **Wavetable lookup synthesis allows for quick prototyping of features.**
  This allows for quick waveform generation in conjunction with low frequency oscillators, envelopes, and other effects.

- **Works especially well with interpolation methods.**
  Values from indices that fall between two normal indices can be determined via interpolation methods.

ADDITIVE SYNTHESIS

- **Synthesizing a Fourier series by weighted summation of harmonics of the basic sine wave can generate all waves imaginable.**

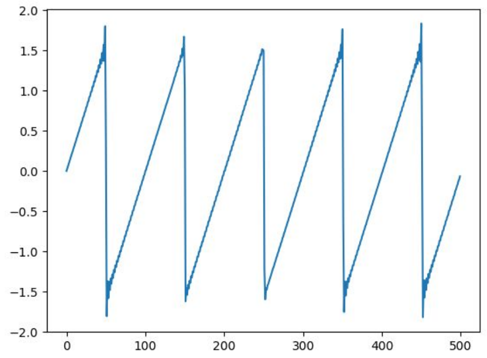
  Approximation serves the additional process of appropriately converting digital discrete signals to analog continuous signals



ADDITIVE SYNTHESIS

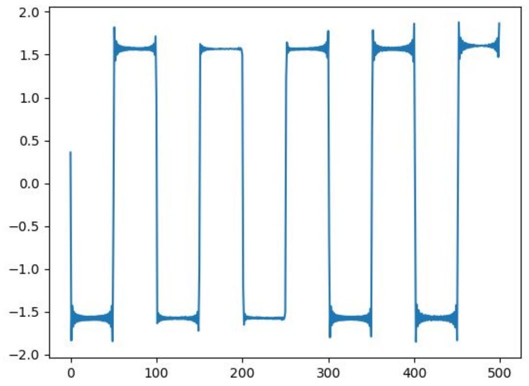
- **Summation of input wave signals to create a new output signal.**

  Additive and subtractive synthesis are used between waveforms to generate more complex and sophisticated waveforms.



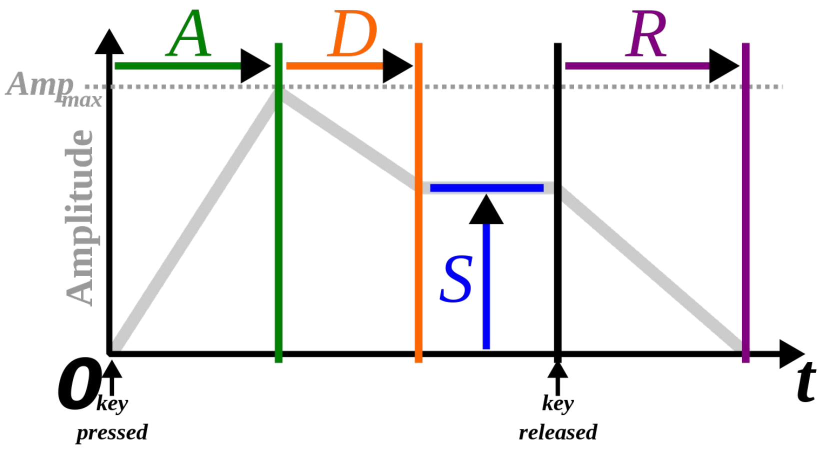
- **Resulting waveforms can be stored into waveform tables.**

  Recursively using additive/subtractive synthesis and storing the result into new wavetables turns sophisticated waveforms into a simple and easily optimizable sequence of binary operations.



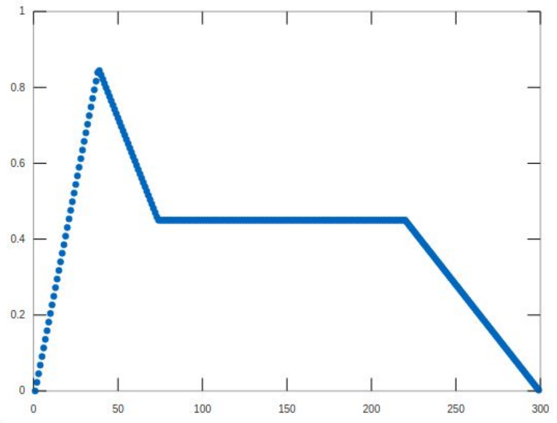
ENVELOPES

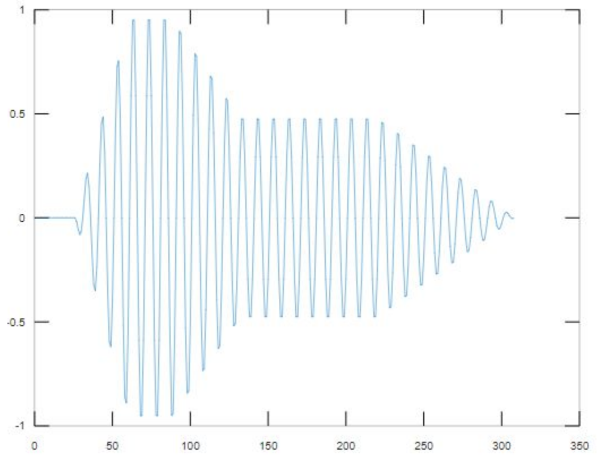
- **Applies dampening/expansion to different stages of a sound.**
  - Attack - Modifies sound from 0 to max amplitude from noteOn.
  - Decay - Modifies sound from max amplitude to Sustain level after Attack phase.
  - Sustain - Level of max amplitude to hold constant after Decay phase.
  - Release - Modifies sound from Sustain level to 0 after noteOff.



ENVELOPES

- **Drastically alter a sound’s character.**
  - The difference between a waveform becoming a “guitar” or a “cello”.
- **Start with the basic ADSR, extended to generalized multistep.**
  - The project initially implemented the basic ADSR envelope.
  - Extended to the DAHDSR and 8-Step Casio envelopes.
  - Final implementation of the Generalized Multistep envelope.





MIDI INPUT

- **Stands for "Musical Instrument Digital Interface"**

  "Technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing and recording music."

- **Contains detailed instructions that can be interpreted into musical notation.**

  MIDI files don't contain actual audio data and are therefore much smaller in size. Due to this, they are more compact but this makes it more difficult to parse it since there are a lot of information to extract and store.

  Can further contain other detailed information for synthesizer standards such as General MIDI (GM)

PCM WAV OUTPUT

- **An audio file format standard, developed by Microsoft and IBM, for storing an audio bitstream on PCs.**
  - Resource Interchange File Format (RIFF) bitstream format
  - Pulse Code Modulation (PCM) Encoding
  - Uncompressed, universally supported.

- **8, 16, and 32 bit WAV supported in any sampling rate.**
  - Conversions are handled independently of the digital synthesis and file writing.

RESULTS

- **Initial Testing was done with the first 16 bars of Beethoven's *Für Elise.***

  Für Elise was chosen for its simple notation involving only a single instrument and monophonic harmonic and melodic lines.

- **Test renders average around 3 seconds.**

  Initial renders averaged between 900 to 1000 seconds. After further iteration on the wavetable synthesis with implementation shifted from lists to arrays, the process was well optimized.

- **Render with basic reverb effect.**

  Successful application of a time-delay based reverb effect and rendered for Fur Elise.

CONCLUSION

- **The musical paradigm is an intuitively functional paradigm**

  Functional higher order functions and parallelism work well in tandem with the musical paradigm and workflow.

- **Quick prototyping and implementation of features and core elements of music software.**

  Prototyping and implementing new features such as envelopes and support for additional bitrates was painless and quick.

- **Optimised to become more competitive**

  Optimization to utilize direct access arrays and further optimization for unboxed arrays will bring performance even closer or better than current standards.

- **Components were developed by different team members and then integrated and extended quickly between different modules.**

  Work within an Agile methodology such as Scrum/Kanban flows well.

FUTURE WORK

- **Support for additional input and output file types**
  Import file types such as MusicXML and export file types such as .mp3, .flac, and .ogg to be competitive with existing frameworks.

- **Additional functionality via filters and effects**
  Frequency based filters such as passes, shelves, and EQ
  Amplitude based effects such as compression, gate, and distortion.
  Time based effects such as delay,reverb, and chorus.

- **Explore parallelism in further development**
  Integrating our current project into current industry standards such as VST3

THANK YOU FOR YOUR ATTENTION!

This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

SZÉCHENYI 2020

HUNGARIAN GOVERNMENT

European Union
European Social Fund

INVESTING IN YOUR FUTURE