An exploration of digital synthesis

March 3, 2023

Angelo Indre

Table Of Contents

* A short digital music history (With a strong focus on computer science topics)
* Digital Audio Basics
* MIDI
  + Speed required and how its done.
* VSTs
* A section all about the JUCE framework.
  + Subsection about what I was able to create with JUCE. Things it makes easy, challenges I still have,
* Results / Outcomes / Takeaways
* Glossary of technical terms

# Abstract

An Exploration of Digital Synthesis" is a comprehensive investigation into the world of digital audio and music production. The paper explores the fundamental concepts of sound synthesis, including MIDI, virtual instruments (VSTs), and the JUCE framework. The central focus of the paper is the implementation of a custom synthesizer, which serves as a case study for the practical application of digital synthesis. The paper addresses the key question of how to create a functioning synthesizer from scratch, providing detailed insights into the programming and design process. Overall, the paper represents a significant contribution to the field of digital audio and music technology, offering a valuable resource for both musicians and software developers alike.

# History of Digital Music

# Basics of Digital Audio

# MIDI

# VST’s

# The JUCE Framework

## JUCE Plugin Class Structure

## JUCE Class Inheritance

# MySynth

## The Audio Programmer Synthesizer

# Conclusion

## Overall

## Future Plans

# Resources

## Information sources and articles

Sampling Rate and Bit Depth - <https://www.adobe.com/uk/creativecloud/video/discover/audio-sampling.html>

The JUCE Framework - <https://juce.com/>

The Audio Programmer Synthesizer - <https://github.com/TheAudioProgrammer/tapSynth>

MySynth - <https://github.com/aerdni99/MySynth>

## Image Sources

**BASICS OF DIGITAL AUDIO**

Digital audio is what we do now for just about all studio processes. What did we do in the past? Uhh, tape recording? But how does digital audio recording work? Well, that’s something I remember from when I had lots of enthusiasm for this project. I read an article all about it. I’ll add it to the sources page now.

So, how do sound waves in the air get transformed into data? Well, lets look at a microphone. The microphone records air pressure at a point in time. This measure is called a sample. The continuous changes in air pressure over time are synonymous with sound. So if we could record and reproduce the exact air pressure over time for a period, then we can replicate recorded sounds. Now, to introduce a key term, **sampling rate** is as it seems, the rate at which samples of air pressure are collected. If we collected and reproduced a low number of samples per second, then our replica would be choppy and inauthentic, but with a higher frequency of samples, 44100Hz. To be specific, then we could create replicas that are indistinguishable from the original sound. 44100Hz. is the golden number where any frequency below that is noticeably different than the original. The highest detectable sound to human ears is around 20kHz. In order to capture any sound we must record at least 2 samples per the wave’s period.

44100 datapoints per second is a huge number. It raises the question of how much data we have to store in order to authentically reproduce audio. How many bits does it take to record a sample? Well, the answer is at least 16-bits per sample. With 16 bits, we can store 2^16th unique wave amplitudes which is the minimum number required to reproduce sounds that our ears are incapable of distinguishing from original source.

Say that for example, we are recording audio with a sampling rate of 44.1kHz, 16-bit depth, and 2 channels for stereo recording. That means that every second of recording requires 176.4kB.

So now that we know what digital audio is, how do we make it from scratch? It’s actually quite simple in theory. The most basic sound wave form is a sine wave. Knowing that frequency determines pitch and amplitude determines volume, you can predict and create the exact amplitude for a sine sound wave without playing back a recorded source. This is a very basic form of algorithmically generated sound. Now, if you assign specific frequencies to note names, and specific amplitudes to key press velocities, you can make a sine wave synthesizer operated by a midi controller. But I’m getting ahead of myself here. It’s a bit outside the scope of my research to trace how my computer hardware is accessed to produce sound But I’ll look into it because a good understanding of how that process is carried out would be helpful. How does this collection of samples get fed to my sound card? How does it translate that into instructions for my speakers to vibrate at the correct intensity and frequency?

Talk about DAWS\*\*\*

**What is MIDI and How Does It Get Applied to Digital Music?**

MIDI stands for Musical Instrument Digital Interface. It is, by design, a standard instrument language that can be understood across manufacturers. MIDI was created in the 1980’s when Ikutaro Kakehashi, the founder of Roland (one of the leading synthesizer manufacturers ) reached out to other popular synth manufacturers and addressed the new issue that with an exponential increase in digital music hardware production, there needed to be a standard way for these digital technologies to communicate. The answer to this issue was MIDI. But this was a standard introduced in 1982. It heavily influenced the way all digital music hardware was made for the next 4 decades.

One of the questions that I will address later in this paper is whether or not it is a good thing that this technology is so portable. What is the cost of this portability in terms of design choices?

**How Does MIDI Work?**

So in my project right now, I am trying to build an arpeggiator that accepts midi data as input. We know that a midi message contains information like touch velocity and note number, but what’s the standard way this information is communicated so that all different controllers can be understood the same way by software?

A MIDI message is a tuple of 3 bytes (represented by integers in juce) and 1 double variable representing a time stamp on the message. When I use my MidiControlCenter which is software provided by Arturia that interfaces with the keyboards that I own, I can open the MIDI console and see that this data is processed live. The three bytes and timestamp are logged to the MIDI console. MIDI is of course digital signal processing. A program that uses MIDI is always awaiting messages and handles them in-place. We’ve already addressed a need for low latency in MIDI programs. Samples of MIDI information are listened for at a rate of 44.1kHz.

What do the three bytes represent?

Well, there must be more than just the three bytes and timestamp because these three bytes have different meaning depending on the TYPE of message of which there are 7.

* Note On
* Note Off
* Monophonic (Channel) Aftertouch
* Polyphonic (Key) Aftertouch
* Pitch Bend
* Program Change
* Control Change

Of this list, the most important to me in this project are Note On and Note Off, but I will also be concerned with Pitch Bend and Control Change for features I want to add down the line. But Note On and Note Off are key to making music (pun intended). For this type of message, byte 1 represents whether the message is note on or note off (seems like a waste of a byte), byte 2 is note number ranging from 0-127. 60 represents middle C on a normal piano and incrementing by halfsteps, the range covers a whole 88-key grand piano plus almost 2 octaves on both sides. The third byte represents the velocity with which the note was hit. This one also ranges from 0-127. Classically, lower velocities mean quieter sounds and higher velocities are louder ones.

The Control Change (CC) parameters are particularly interesting. They (Along with Pitch Bend and the Aftertouches) are meant to relay continuous change in a parameter.

**How Fast Does All This Need To Happen in Order to Have an Instrument that passes as acoustic?**

Pretty darn fast. I ought to find a metric similar to 44100Hz and 20kHz being the max hearing range, but in terms of how much delay from key press to sound is noticeable to myself and an audience.

**What Makes The Speed from the previous section achievable?**

**What is a VST?**

VST stands for Virtual Studio Technology. It is a plugin that can be used when recording digital audio

**The JUCE Framework**

So, I’ve spent a lot of time so far reading about and working with the JUCE framework. It serves as a platform for audio plugin development that takes care of low level details like porting to other operating systems. It also templates useful functions for creating your applications. The main goal of this project for me was to create a basic VSTi with MIDI and synthesis capabilities.