\chapter{Abstract} \label{ch:Abstract}

The aim of this work is to provide a comprehensive explanation of how to code an audio plugin in the audio application framework JUCE as well an introduction to the mathematics involved. In particular lays the groundwork to code a synthesizer that represents all "reasonable" sound waves on a line through an infinite dimensional space filling curve and an effect plugin that implements the Volterra series to unify different nonlinear time invariant effects.

\chapter\*{Preface}

\addcontentsline{toc}{chapter}{Preface}

I have always been interested in both music and mathematics. I have been playing the violin since I was five and the hammered dulcimer since I was eight. Back when I was eleven years old and in my last year of primary school the game "Geometry Dash" got me into the EDM genre, and I wanted to try my hand at creating my own electronic music. I was given the program FL Studio for my twelfth birthday and started playing around. But it soon became clear that creating sounds I was happy with was a difficult task. I couldn't get my synths to growl, whine or whatever else I was trying to do. It simply didn't sound like what the professionals produced. So, I stopped trying. But as I learned more mathematics, especially during COVID, where time was abundant, my fascination of audio processing reignited. I wanted to know the inner workings of the synthesizers and effects I had experimented with years earlier. Maybe now I could finally design the sounds I wanted to. Maybe now I could learn this skill that had eluded me. I had also started to develop my own ideas for audio plugins that I wanted to create. I did some research and put that idea to the side as a candidate for the matura-topic, and here we are.

I would like to take this opportunity to thank \*\*\* for checking parts of the text for comprehensibility, he has been an enormous help in improving the readability of the text. I would also like to thank \*\*\* and \*\*\* for the support I have received from them when I was learning C\texttt{++} and similarly my supervisor for helping me with \LaTeX and my work more generally though I regret not asking for help more often. I would like to thank \*\*\* especially for also creating the images \ref{fig:hilbert curve} and \ref{fig:Z order} when time was running short. I need to thank my dad for providing the tools and materials used to create this booklet. Finally, I would like to thank all my friends and both my parents for the emotional support they have provided throughout the years.

\chapter{Introduction}\label{ch:Introduction}

Nowadays most of the audio you hear online underwent some editing process on a computer, be it splicing together different recordings, the reduction of noise in a recording, the balancing of loudness throughout different times in the recording or a multitude of other possible tweaks. Music is especially digitized with each instrument often being recorded separately and undergoing a separate processing chain if it is being recorded at all. Many instruments are being replaced by software synthesizers that sound increasingly similar to their real-world counterparts. But synthesizers can also be used to create completely new sounds, opening the door to a world of possibilities we could never before imagine. Audio effects have similarly been used creatively. One prominent example is the controversial use of strong Autotune to create a robotic sounding voice. And all this stands on the shoulders of the software behind it all.

Most audio production takes place in a so called \gls{DAW} in which multiple recordings can be arranged, layered and edited and which typically include features to play back virtual instruments off of a virtual piano roll. \glspl{DAW} usually also allow the use of third party software called plugins to be used as both synthesizers and effects in addition to those included in the installation of the \gls{DAW} itself. This gave rise to a market of often pricey audio plugins which makes it all the more valuable to know how to program one yourself.

This work will go through the process of programming such an audio plugin.

\chapter{Mathematical Groundwork}\label{ch:Mathematical Groundwork}

%A reference can be included by using \verb|\cite{NameOfReference in refs.bib}|. For example \cite{AmannEscher1}.

\section{Prerequisites}

Below is a list of topics the reader should be familiar with before reading this work along with references to resources covering them: %References to resources where you can learn about these topics are provided in \ref{ch:Links}:

\begin{itemize}

\item Complex number basics

\item Linear algebra fundamentals \cite{3b1b\_Linear\_Algebra}

\item Taylor series \cite{3b1b\_Taylor\_series}

\end{itemize}

\section{Complex exponential}

The concept of a complex valued exponent will appear a lot later so I am including an explanation here though the definition may take time and practice to properly digest.

So far, we have extended the domain of the function %f(t)=

$f(t) = b^t$ (for positive real $b$) from the positive integers to the the real numbers using the property that $b^{s+t}=b^s\cdot b^t$ ($ \Rightarrow (b^t)^n=b^{nt}$ for integer $n$)

and the assertions that $b^t$ is a continuous function (and that it is real valued for real $t$).

These properties alone are not enough to uniquely define $b^t$ for complex $t$.

To see this, observe that if $b^{ih}$ is defined for some $h \in \R$ we can derive definitions for $b^{x+inh}$, $n \in \N$, $x \in \R$. Now define $b^{ih}$ to approach $b^0 = 1$ as $h \rightarrow 0$ from any direction in the complex plane and any velocity of your choosing. No matter which direction and velocity is chosen, this will lead to a continuous extension of $b^t$ to the complex numbers (see \ref{fig:wrong approaches}).

%To see this, define $b^{ih} = r(\cos(\phi)+i\sin(\phi))$ for some $h \in \R\backslash\{0\}$, $r \in [0, \infty[$, $\phi \in ]-\pi, \pi]$ of your choosing. From this, we can derive definitions for $b^{x+inh}$, $n \in \N$, $x \in \R$. Additionally, define $b^{\frac{ih}{m}} = r^\frac{1}{m}(\cos(\frac{\phi}{m})+i\sin(\frac{\phi}{m}))$, $m \in \N$ to be the principal $m$th root of our $b^{ih}$. We now have definitions for $b^{x+i\frac{n}{m}h}$, $n,m \in \N$, $x \in \R$ which we can extend to $b^{x+iy}$, $x, y \in \R$ by taking limits; $b^{x+iy} = \lim\_{\frac{n}{m}h \rightarrow y} b^{x+iqh}$. No matter how we choose to define $b^{ih}$ in the beginning, this will lead to a continuous extension.

%For example the definitions $b^{ix}=c^x(\cos(x)+i\sin(x))^d$ for $c,d \in \R$ are all consistent with these properties.%fact check

\begin{figure}[h]

\centering

\begin{subfigure}{0.3\textwidth}

\centering

\includegraphics[width=\textwidth]{images/MA\_exp\_wrong\_1.png}

\caption{}

\end{subfigure}

\hfill

\begin{subfigure}{0.3\textwidth}

\centering

\includegraphics[width=\textwidth]{images/MA\_exp\_wrong\_2.png}

\caption{}

\label{fig:right approach}

\end{subfigure}

\hfill

\begin{subfigure}{0.3\textwidth}

\centering

\includegraphics[width=\textwidth]{images/MA\_exp\_wrong\_3.png}

\caption{}

\end{subfigure}

%\caption{Showing $b^{it}$ for real t for different arbitrary definitions of $b^{ih}$}

\caption{Red: $b^{ih}$ and consequent definitions for $b^{inh}$. Dashed: The direction of approach.

%$b^0$ can be approached from different angles yielding multiple possible extensions of $b^t$ that fit the previous criteria.

}

\label{fig:wrong approaches}

\end{figure}

To remedy this we will add the further constraint that $b^t$ is differentiable (in fact we only need it to be differentiable at $0$).

Since the derivative of $b^t$ is $\ln{b}$ at $t=0$, $ \lim\_{\substack{h \rightarrow 0\\ h \in \R}}\frac{b^{0+ih}-b^0}{ih} = \ln(b) \implies b^{0+ih}-b^0 \rightarrow ih\ln(b)$ as $h \rightarrow 0$, $h \in \R$, i.e. $b^{ih}$ approaches $b^0 = 1$ from the purely imaginary direction with velocity $\ln(b)$ as $h \rightarrow 0$ as in figure \ref{fig:right approach}. %elaborate

This in conjunction with the other properties uniquely defines $b^{it}$ to be $(\cos{t}+i\sin{t})^{\ln{b}}$ for real $t$.

The graph of $b^{it}$ is a helix through the complex plane around the $t$-axis with angular frequency $\ln{b}$ (see \ref{fig:complex exponential}).

\begin{figure}[h]%{R}{0.4\textwidth}

\centering

\includegraphics[width = 0.4\textwidth]{images/ComplexExponentialProvisory.jpg}

\caption{The graph of $e^{it}$ is a helix. \cite{Corkscrew}}

\label{fig:complex exponential}

\end{figure}

Important special cases are $e^{it}$ which has \gls{angular frequency} $1$ and $e^{2 \pi i t}$ which has \gls{angular frequency} $2\pi$ i.e. it has frequency $1$.

This definition also retains the property that $\frac{d}{dt}e^{\omega t}=\omega e^{\omega t}$.

$c^t$ for complex $c = |c| e^{\varphi i}$ is usually defined as $|c|^t e^{\varphi i t}$ such that $-\pi<\varphi\leq\pi$.

\section{Linear operators and integral transforms}

Let $M\{x(\tau\_M)\}(t)$ be an \gls{operator} (e.g. an audio effect), $x(t)$, $y(t)$ be complex valued functions (e.g. sound waves) and $a \in \C$ a constant:\\

$M\{x(\tau\_M)\}(t)$ is said to be linear if

\begin{equation}

M\{x(\tau\_M)+y(\tau\_M)\}(t)=M\{x(\tau\_M)\}(t)+M\{y(\tau\_M)\}(t)

\end{equation}

and

\begin{equation}

M\{a\cdot x(\tau\_M)\}(t)=a\cdot M\{x(\tau\_M)\}(t)

\end{equation}

that is, applying $M$ after adding $x$ and $y$ has the same effect as applying $M$ to $x$ and $y$ individually, then adding $Mx+My$.

Given an operator $M$ and a function $x$ to apply $M$ to,

we can make use of the linearity of $M$ by writing $x$ as a sum of "spike functions" centered at different points in time and applying $M$ to each "spike" individually first before summing them up again.

Define $\delta[t]$ to be such a spike function that is $1$ at $t=0$ and $0$ everywhere else. Spikes centered at different times $\tau$ are generated by the expression $\delta[t-\tau]$.

\begin{equation}

x(t)=\sum\_{\tau} x(\tau)\cdot \delta[t-\tau]

\end{equation}

thus:

\begin{wrapfigure}{l}{0.3\textwidth}

\centering

\includegraphics[width = 0.3\textwidth]{images/MA\_spikes.png}

\caption{$x(t)$ can be written as a sum of "spike functions".}

\vspace{-120pt}

\label{fig:spikes}

\end{wrapfigure}

\begin{align} \label{eq:discrete linear transform}

M\{x(\tau\_M)\}(t) & = M\lrc{\sum\_{\tau} x(\tau)\cdot \delta[\tau\_M-\tau]}(t) \nonumber \\

& = \sum\_{\tau} M\{x(\tau)\cdot \delta[\tau\_M-\tau]\}(t) \nonumber \\

& = \sum\_{\tau} x(\tau)\cdot M\{\delta[\tau\_M-\tau]\}(t) \nonumber \\

\shortintertext{defining $M\{\delta[\tau\_M-\tau]\}(t):=m(\tau, t)$:}

& =\sum\_{\tau} x(\tau)\cdot m(\tau, t)

\end{align}

$m$ is called the kernel function of the transform. You may recognize this as the formula for matrix multiplication because that is exactly what it is.

This expression works fine for discrete functions but it has the downside that, if the sampling period changes by a factor of $T$, all else being equal, the resulting amplitude will be scaled by $1/T$.

To keep the amplitude constant we multiply the expression by $T$:

\begin{equation}

\sum\_{\tau} x(\tau)\cdot m(\tau, t)\cdot T

\end{equation}

And as $T$ tends to 0 the expression tends to the integral:

\begin{equation}

\int\_{\tau} x(\tau)\cdot m(\tau, t)d\tau

\end{equation}

This type of expression is termed an integral transform.

\subsection{Composition}

The composition $L=MN$ of two linear operators $M,N$ is itself a linear operator and its kernel function is given by:

\begin{equation} \label{eq:discrete composed kernel}

l(\tau, t) = \sum\_f m(f, t) \cdot n(\tau, f)

\end{equation}

Where $l, m, n$ are the kernel functions of $L, M, N$ respectively.

Or for the continuous case:

\begin{equation}

l(\tau, t) = \int\_f m(f, t) \cdot n(\tau, f) df

\end{equation}

Derivation for the discrete case:

\begin{equation}

\begin{split}

L\{x(\tau\_L)\}(t) &= \sum\_f m(f, t) \sum\_\tau x(\tau) \cdot n(\tau, f) \\

&= \sum\_\tau x(\tau) \underbrace{\sum\_f m(f, t) \cdot n(\tau, f)}\_{\text{kernel function}}

\end{split}

\end{equation}

The derivation for the continuous case is analogous.

\section{Linear time invariant systems and convolution}

Again, let $M\{x(\tau\_M)\}(t)$ be an \gls{operator}, $x(t)$ be a complex valued function and $\tau\_M, t, t\_0 \in \R$ (or more generally elements of some group under addition);\\

$M\{x(\tau\_M)\}(t)$ is said to be time invariant if

\begin{equation}

M\{x(\tau\_M+t\_0)\}(t)=M\{x(\tau\_M)\}(t+t\_0)

\end{equation}

that is, applying $M$ after shifting $x$ in time by some $t\_0$ has the same effect as applying $M$ first, then the shift in time.

Examples of operators that are both linear and time invariant (LTI) include scaling, time shifting itself, differentiation (and differencing),

and real world examples like \glspl{equalizer}, blur (weighted averaging over time) and \gls{reverb}.

When M is time invariant, aside from being linear, the kernel function can be expressed as a single variable function:

\begin{equation}

M\{\delta[\tau\_M-\tau]\}(t)=M\{\delta[\tau\_M]\}(t-\tau):=m(t-\tau)

\end{equation}

$m(t)$ is also called the system's impulse response because it represents the output, i.e. response, of the system given an impulse, i.e. spike function, centered at 0.

\begin{figure}[H]

\centering

\includegraphics[width=0.7\textwidth]{images/Impulse.png}

\caption{The Impulse response from a simple audio system. Showing, from top to bottom, the original impulse, the response after high frequency boosting, and the response after low frequency boosting.\cite{Impulse\_Response}}

\label{fig:impulse response}

\end{figure}

Replacing $m(\tau, t)$ with $m(t-\tau)$ in \ref{eq:discrete linear transform} we get:

\begin{equation}

M\{x(\tau\_M)\}(t) = \sum\_{\tau} x(\tau)\cdot m(t-\tau)

\end{equation} %=sum(for all tn, x(t-tn)\*m(t))

The last expression is called the convolution of $x$ with $m$ and is denoted $x\*m$. By substitution one can show that $x\*m=m\*x$ is commutative.

Analogously for continuous applications:

\begin{equation}

\int\_{\tau} x(\tau)\cdot m(t-\tau)d\tau

\end{equation}

\subsection{Eigenfunctions}

We might be interested in finding \glspl{eigenfunction} for LTI \glspl{operator}. For this, it suffices to find a set of eigenfunctions of all time shift operators. %rephrase?

Let $x(t)$ denote one such eigenfunction. Then:

\begin{equation} \label{intermediate1}

x(0+\tau)=\lambda\_{\tau}\cdot x(0)

\end{equation}

applying a time shift of $\tau$ $m$ times yields:

\begin{equation}

x(m\cdot \tau)=\lambda\_{\tau}^m\cdot x(0)

\end{equation}

Put in words, the eigenfunctions of the time shift operator are exponential functions and can be expressed as $Ae^{st}$ for $A,s \in \mathbb{C}$.

These functions are also eigenfunctions of all other LTI operators:

Let $m$ denote the impulse response of the LTI operator\cite{LTI\_Eigenfunctions}:

\begin{equation} \label{eq:convolution eigendecomposition}

\begin{split}

&\int\_{\tau} Ae^{s\tau}\cdot m(t-\tau) d\tau \\

=&\int\_{\tau} Ae^{s(t-\tau)}\cdot m(\tau) d\tau \\

=&\int\_{\tau} Ae^{st} \cdot e^{-s\tau} \cdot m(\tau) d\tau \\

=&\underbrace{Ae^{st}}\_{\text{Input}} \underbrace{\int\_{\tau} e^{-s\tau} \cdot m(\tau) d\tau}\_{\text{Scalar}}

\end{split}

\end{equation}

%Let $M$ denote an LTI operator and $x$ the function from before:

%\begin{equation} \label{intermediate2}

% M\{x(\tau\_M)\}(0+\tau)=M\{x(\tau\_M+\tau)\}(0)=M\{\lambda\_{\tau}\cdot x(\tau\_M)\}(0)=\lambda\_{\tau}\cdot M\{x(\tau\_M)\}(0)

%\end{equation}

%Meanwhile solving for $\lambda\_{\tau}$ in \ref{intermediate1} yields:

%\begin{equation}

% \lambda\_{\tau}=\frac{x(\tau)}{x(0)}

%\end{equation}

%Inserting into \ref{intermediate2}:

%\begin{equation}

% M\{x(\tau\_M)\}(\tau)=x(\tau)\cdot \frac{M\{x(\tau\_M)\}(0)}{x(0)}

%\end{equation}

%Which is what we set out to prove.

\section{Fourier transform}

Knowing that exponential functions are eigenfunctions of LTI \glspl{operator} we might be interested in decomposing functions into a sum of exponential functions.

This will make calculations easier as will become apparent in section \ref{subsec:Convolution Theorem}.

One such decomposition is the Fourier transform, which concerns itself with decomposing functions into periodic exponentials of frequency $f$ i.e. complex exponentials of the form $e^{2\pi ift}$.\\

The Fourier transform of a function $x(t)$ is defined as the function $\hat{x}(f)$ such that:

\begin{equation} \label{eq:inverse Fourier transform}

\int\_{-\infty}^\infty \hat{x}(f)\cdot e^{2\pi ift}df=x(t)

\end{equation}

That is, for each frequency $f$, $\hat{x}(f)$ returns the amplitude of the complex exponential of that frequency "contained" in $x$

such that summing over all complex exponentials with their respective amplitudes returns $x$. The process of summing over the complex exponentials as above is called the inverse Fourier transform.

If $x$ is periodic then it will only have frequency components at integer multiples of the \gls{fundamental frequency} $f\_0 = \frac{1}{T}$. In this case it makes sense to define $\hat{x}$ as the Fourier series coefficients:

\begin{equation} \label{eq:Fourier series}

\sum\_{n=-\infty}^\infty\hat{x}[n] \cdot e^{2\pi int/T} = x(t)

\end{equation}

(this definition is not equivalent to \ref{eq:inverse Fourier transform})

If we were to define a \gls{DFT} to act on such a periodic function which was sampled at N discrete points in \emph{time}, for the \gls{DFT} to be unique and invertible, the \gls{IDFT} must also act on N discrete points in \emph{frequency}. This follows from the properties of linear transformations. We thus define the the \gls{IDFT} $\hat{x}$ of $x$ via a truncated Fourier series with period N:

\begin{equation} \label{eq:IDFT}

\sum\_{f=1}^N \hat{x}[f] \cdot e^{2\pi i ft/N}=x[t]

\end{equation}

Sometimes the \gls{IDFT} is scaled by a factor of $\frac{1}{\sqrt{N}}$ so as to make it a \gls{unitary} operator, and we will use this definition moving forward:

\begin{equation} \label{eq:unitary IDFT}

\frac{1}{\sqrt{N}}\sum\_{f=1}^N \hat{x}[f]\cdot e^{2\pi i ft/N}=x[t]

\end{equation}

it is worth noting that the continuous inverse Fourier transform \ref{eq:inverse Fourier transform} is also unitary.

One could also choose to center the summation around 0 for odd N so as to make it possible for the imaginary components of the positive and negative frequency components to cancel:

\begin{equation} \label{eq:centered IDFT}

\frac{1}{\sqrt{N}}\sum\_{f=-(N-1)/2}^{(N-1)/2} \hat{x}[f]\cdot e^{2\pi i ft/N}=x[t]

\end{equation}

To derive a closed form expression for the Fourier transform recall that a matrix is unitary when its inverse equals its conjugate transpose. Analogously, a linear operator $M$ (on \gls{L2}) is unitary when its kernel function $m$ is related to the kernel function of its inverse $n$ by: $m(\tau, t)=\overline{n(t,\tau)}$, where the bar denotes complex conjugation. If the inverse Fourier transform is indeed unitary then \emph{its} inverse, i.e. the Fourier transform itself, will be given by:

\begin{equation} \label{eq:Fourier transform}

\hat{x}(f) = \int\_{-\infty}^\infty x(t)\cdot e^{-2\pi i ft} dt

\end{equation}

(Indeed, this is the eigenvalue of the convolution operator from \ref{eq:convolution eigendecomposition}.) Similarly, the discrete Fourier transform would be:

\begin{equation} \label{eq:discrete Fourier transform}

\hat{x}[f] = \frac{1}{\sqrt{N}}\sum\_{t=1}^N x[t]\cdot e^{-2\pi i ft/N}

\end{equation}

Let us verify that this is correct in the discrete case (with $N>1$) using the following identity on the way:\\

For $N>1$:

\begin{equation} \label{eq:sum of roots of 1}

\Sigma=\sum\_{f=1}^N e^{2 \pi i f/N}=0

\end{equation}

To prove this, notice that the set $S = \{ e^{2 \pi i n/N} | n \in \N \}$ of $N$th \glspl{root of unity} is invariant with respect to elementwise multiplication with one of its elements; multiplication by some $z \in S$ is one-to-one, because it is invertible, and onto, because the product of two $N$th roots of unity is another $N$th root of unity. This means that $\Sigma$ must satisfy: $z \Sigma = \Sigma$. Now, for $N>1$, there exists a $z \in S, z \neq 1$ and thus the equation $z \Sigma = \Sigma$ can only be satisfied if $\Sigma = 0$.\\

\begin{figure}[H]

\centering

\begin{subfigure}{0.4\textwidth}

\centering

\includegraphics[width=\textwidth]{images/MA\_Roots\_of\_Unity.png}

\end{subfigure}

\hfill

\begin{subfigure}{0.4\textwidth}

\centering

\includegraphics[width=\textwidth]{images/MA\_Roots\_of\_Unity\_Sum.png}

\end{subfigure}

\hfill

\caption{Visualization of \ref{eq:sum of roots of 1}. Left: the fifth roots of unity. Right: the sum of the fifth roots of unity.}

\label{fig:sum os roots of 1}

\end{figure}

Notice also that, for $t \in \N$, $S^t$ is also a set of roots of unity which, as long as $N$ does not divide $t$, has strictly more than one element, so \ref{eq:sum of roots of 1} applies to:

\begin{equation}

\sum\_{f=1}^N (e^{2 \pi i f/N})^t=\sum\_{f=1}^N e^{2 \pi i ft/N}=0

\end{equation}

When $t \nmid N$.

Now we need to show that the composition of the inverse Fourier transform with \ref{eq:discrete Fourier transform} is the identity operator: \\

By \ref{eq:discrete composed kernel}, the kernel function of the composition of \ref{eq:discrete Fourier transform} and the inverse Fourier transform \ref{eq:unitary IDFT} is given by:

\begin{equation}

\begin{split}

&\sum\_{f=1}^N \frac{1}{\sqrt{N}}e^{2\pi i ft/N} \cdot \frac{1}{\sqrt{N}}e^{-2\pi i f\tau/N} \\

=& \frac{1}{N}\sum\_{f=1}^N e^{2\pi i f(t-\tau)/N}\\

=& \begin{cases}1, & \tau=t\\ 0, & \tau \neq t\end{cases} \\

\end{split}

\end{equation}

Which is the kernel function of the identity operator.

\subsection{Convolution theorem} \label{subsec:Convolution Theorem}

Using the Fourier transform, convolution of functions $x$ and $m$ can be written as:

\begin{equation}

\begin{split}

x\*m &= \int\_\tau x(\tau) \cdot m(t-\tau)d\tau \\

&= \int\_f \underbrace{\hat{x}(f) \cdot e^{2\pi ift}}\_{\text{\gls{eigenfunction}}} \overbrace{\underbrace{\int\_{\tau} e^{-2\pi if\tau} \cdot m(\tau) d\tau}\_{\text{eigenvalue from \ref{eq:convolution eigendecomposition}}}}^{\hat{m}(f)} df \\

&= \int\_{f} \hat{x}(f)\cdot \hat{m}(f)\cdot e^{2\pi ift}df

\end{split}

\end{equation}

This result is known as the convolution theorem.

\subsection{Physical meaning}

If the function $x(t)$ to be Fourier transformed is purely real, then $\hat{x}(f)=\overline{\hat{x}(-f)}$ so that the imaginary parts of the positive and negative frequency components cancel, effectively reducing the Fourier transform to a decomposition into shifted cosine waves. %elaborate

This means, the Fourier transform can be interpreted as decomposing $x$ into a sum of so called harmonic oscillations.

A harmonic oscillator is a system that exerts a restoring force proportional to its displacement:

\begin{equation}

F=-kx

\end{equation}

From which the potential energy follows:

\begin{equation}

E\_{pot}=\int\_0^x k s^2 ds= \frac{1}{2}kx^2

\end{equation}

A spring, for instance, is modeled as a harmonic oscillator.

The mass at the end of a spring has the kinetic energy:

\begin{equation}

E\_{kin}=\frac{1}{2}mv^2

\end{equation}

Consider the coordinate space $(x, y)$ with $x = x$ and $y = \frac{\sqrt{m}}{\sqrt{k}}v$. The coordinates are chosen to be proportional to the square roots of the potential and kinetic energies respectively.

For a given point $(x, y)$ in that space, its derivative in $y$ direction is:

\begin{equation}

\begin{split}

\frac{dy}{dt} &= \frac{\sqrt{m}}{\sqrt{k}} \cdot \frac{dv}{dt} \\

&=\frac{\sqrt{m}}{\sqrt{k}} \cdot \frac{-kx}{m} \\

&=\frac{-\sqrt{k}}{\sqrt{m}}x

\end{split}

\end{equation}

and similarly its derivative in x direction:

\begin{equation}

\frac{dx}{dt}=v=\frac{\sqrt{k}}{\sqrt{m}}y

\end{equation}

That is, for a point $(x, y)$ in this space, its direction vector is given by the scaled $90°$ rotation $\frac{\sqrt{k}}{\sqrt{m}}(y, -x)$

and because of this the point moves along a trajectory of a circle with velocity $\frac{\sqrt{k}}{\sqrt{m}}r = \frac{\sqrt{k}}{\sqrt{m}}x\_{\text{max}}$ (where $r$ is the radius of the circle) or \gls{angular frequency} $\frac{\sqrt{k}}{\sqrt{m}}$ (independent of $r$ or the energy in the system!). When regarding only the displacement axis, the mass at the end of the spring moves in a shifted and scaled cosine wave.

\begin{figure}%{l}{0.4\textwidth}

\centering

\includegraphics[width = 0.4\textwidth]{images/MA\_phase\_space.png}

\caption{The point moves around a circular trajectory.}

%\vspace{-80pt}

%\label{fig:}

\end{figure}

\subsection{Fast Fourier transform}

Not only is the Fourier transform useful for us humans to understand what frequencies are present in a sound, it can also be computed quickly (in $O(n \log n)$ time) which, in conjunction with the convolution theorem, makes it useful to speed up calculations. (A naive implementation of convolution would take $O(n^2)$ time.) This, and other reasons, give the Fourier transform applications in theoretical computer science, image processing, statistics and more aside from the familiar applications in signal processing (audio, radio etc.) and physics.

The algorithm that performs the Fourier transform in $O(n \log n)$ is called the \gls{FFT}.

Here is a reference to a video explaining a simple form of the algorithm: \cite{FFT}.

\subsection{Aliasing} \label{subsec:Aliasing}

In a computer signals are stored as samples at discrete points in time. The density of these points in time are given by the sampling frequency (or sample rate) $f\_s$. As one would expect, some information is lost going from a continuous sound wave to a sampled one. For example the complex exponential $e^{2 \pi i t f\_s/2}$ of frequency $\frac{f\_s}{2}$ would be sampled the same as its complex conjugate $e^{-2 \pi i t f\_s/2}$. $e^{2 \pi i t f\_s2}$ is thus called an alias of $e^{-2 \pi i tf\_s/2}$. Higher frequencies $\frac{f\_s}{2}+\Delta f$ also have aliases: $e^{2\pi i (f\_s/2+\Delta f) t} = e^{2\pi i tf\_s/2} \cdot e^{2\pi i\Delta f t}$ is sampled the same as $e^{-2\pi i tf\_s/2} \cdot e^{2\pi i\Delta f t} = e^{2\pi i (-f\_s/2+\Delta f) t}$. We must thus make an assumption about which alias should be used when our sampled sound is played back.

\begin{figure}

\centering

\includegraphics[width=0.5\textwidth]{images/MA\_Aliasing.png}

\caption{The red and blue sine waves are aliases of one another; they get sampled the same}

\label{fig:Aliasing}

\end{figure}

In \ref{eq:centered IDFT} one such assumption is made, namely that the highest absolute frequency component present during sampling is $\frac{1}{N}\frac{N}{2}=\frac{1}{2}$ \emph{per sample}. To calculate the frequency in Hz we multiply by the sampling frequency $f\_s$ given in $\frac{\mathrm{samples}}{\mathrm{second}}$ to get $\frac{f\_s}{2}$.

If higher frequencies were present during sampling then they would be reconstructed as a low frequency alias in a phenomenon known as aliasing. It is therefore recommended to filter out the high frequencies when sampling when possible. This is either done by multiplying the frequency spectrum $\hat{x}$ by the \gls{indicator function} for the range $[-\frac{f\_s}{2}, \frac{f\_s}{2}]$ or equivalently, by the convolution theorem, by convolving with its Fourier transform; the function $\sinc{\pi f\_s t}=\frac{\sin(\pi f\_s t)}{\pi f\_s t}$.

%You may have noticed that the \gls{DFT} (\ref{eq:discrete Fourier transform}) is similar to a Fourier series and, like a Fourier series, returns a periodic function. Just like the fundamental frequency of a Fourier series is given by the difference of frequencies between the consecutive frequency components and the period being given by its inverse, the period of the \gls{DFT} is given by the inverse of the difference in time between consecutive samples i.e. the sampling period. That is, the period of the \gls{DFT} is given by the sampling frequency $f\_s$. It is therefore natural to choose the range of length $f\_s$ $[-\frac{f\_s}{2}, \frac{f\_s}{2}]$ of lowest frequency and discard all other frequencies when reconstructing the signal. This corresponds to the assumption that no high-frequency material was present during sampling. If high frequencies were present then they would be reconstructed as a low frequency alias in a phenomenon known as aliasing. It is therefore recommended to filter out the high frequencies when sampling when possible. This is either done by multiplying the frequency spectrum $\hat{x}$ by the \gls{indicator function} for the range $[-\frac{f\_s}{2}, \frac{f\_s}{2}]$ or equivalently, by the convolution theorem, by convolving with its Fourier transform; the function $\frac{\sin\left(\frac{\pi}{f\_s}t\right)}{\frac{\pi}{f\_s}t}$.

\section{Volterra series} \label{sec:Volterra series}

In this chapter we will focus on time invariant systems without requiring they be linear.

Given a function $x(t)$ with period $T$, a time invariant \gls{operator} is guaranteed to preserve that period since $x(t+T)=x(t)$ and therefore $M\{x(\tau)\}(t+T)=M\{x(\tau+T)\}(t)=M\{x(\tau)\}(t)$.

Say $x(t)=a\_1e^{f\_1t}+a\_2e^{f\_2t}+...$, then its period $T$ is the least common multiple of the individual components' periods $T=\lcm{T\_1, T\_2, ...}$ where $T\_n=1/f\_n$.

Phrased in terms of frequencies; the frequency $f$ of $x(t)$ is the greatest common divisor of the component frequencies $f=\gcd(f\_1, f\_2, ...)$. %elaborate

This means that $M$ can generate new frequency components of frequencies at integer multiples of $f$.

This phenomenon is called intermodulation distortion and the new frequency components are called intermodulation products.

To approximate or describe a %generally

nonlinear but time invariant operator $M$ one can employ a so called Volterra series.

The Volterra series is essentially a time invariant, infinite dimensional Taylor series around $x\_0 = 0$.

To understand this, we need to define derivatives for multidimensional functions.

Here I will introduce the Fréchet derivative to generalize the derivative of real valued functions of one real variable.

%alternative formulation: Given a function m: X -> Y the Fréchet derivative D of m, when it exists, is defined as Dm=A, where A is the bounded linear operator, such that:

The Fréchet derivative $D$ of a function (or operator) $m: \boldsymbol{X} \rightarrow \boldsymbol{Y}$ when it exists, is defined as the bounded linear operator $A$, such that:

\begin{equation}

\lim\_{||h|| \to 0}\frac{||m(x+h)-m(x)-Ah||\_{\bm{{Y}}}}{||h||\_{\bm{{X}}}} = 0

\end{equation}

This means that the Fréchet derivative $Dm$ of $m$ is a function from $X$ to the space of linear functions from $\boldsymbol{X}$ to $\boldsymbol{Y}$, denoted $\boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y})$: $Dm: \boldsymbol{X} \rightarrow \boldsymbol{L}(\boldsymbol{X}, \boldsymbol{Y})$.

So, $Dm(x\_0)(x\_1)$, which is linear in all arguments except $x\_0$ %even m#

.

%Note that the Fréchet derivative itself is a map from the space of functions from X to Y to the space of functions from X to L(X, Y): D: %%

Higher order derivatives are then of the form $D^n m(x\_0)(x\_1)...(x\_{n-1})(x\_n)$ and are linear in all arguments except $x\_0$ %and n#

Analogous to a Taylor series for one dimensional functions, we wish to approximate $m(x)$ at $x\_0 = 0$.

The zeroth order approximation, as before, is $m(x\_0)$, the first order approximation is $m(x\_0)+Dm(x\_0)(x-x\_0)$, the second $m(x\_0)+Dm(x\_0)(x-x\_0)+\frac{1}{2}D^2m(x\_0)(x-x\_0)(x-x\_0)$ and so on.

Each term being a \gls{multilinear} operator, and for sufficiently well behaved $m$, we can write the generalized Taylor series the following way:

\begin{equation}

m(x)(t)=m(x\_0)(t)+ \sum\_{n=1}^\infty \int\_{\tau\_1} \cdots \int\_{\tau\_n} \frac{1}{n!}d\_n(\tau\_1, \dots , \tau\_n, t) \prod\_{j=1}^n (x(\tau\_j)-x\_0(\tau\_j)) d\tau\_j

\end{equation}

Where $d\_n(\tau\_1,\dots,\tau\_n,t)=D^nm(x\_0)(\delta[\tau\_D-\tau\_1]\cdots\delta[\tau\_D-\tau\_n])(t)$ is the kernel function of $D^n m(x\_0)$.

Let $m$ be time invariant, $x\_0 = 0$, define $h\_n(\tau\_1,\dots,\tau\_n)=d\_n(-\tau\_1,\dots,-\tau\_n,0)$ and notice $m(0)(t)$ is independent of t because a time invariant operator of a constant function is a constant function. Then this can be simplified to obtain the Volterra series:

\begin{equation}

\begin{split}

m(x)(t) &= m(0)(t)+ \sum\_{n=1}^\infty \int\_{\tau\_1} \cdots \int\_{\tau\_n} \frac{1}{n!}d\_n(\tau\_1, \dots , \tau\_n, t) \prod\_{j=1}^n x(\tau\_j) d\tau\_j \\

&= m(0)+ \sum\_{n=1}^\infty \int\_{\tau\_1} \cdots \int\_{\tau\_n} \frac{1}{n!}h\_n(t-\tau\_1, \dots , t-\tau\_n) \prod\_{j=1}^n x(\tau\_j) d\tau\_j \\

&= m(0)+ \sum\_{n=1}^\infty \int\_{\tau\_1} \cdots \int\_{\tau\_n} \frac{1}{n!}h\_n(\tau\_1, \dots , \tau\_n) \prod\_{j=1}^n x(t-\tau\_j) d\tau\_j

\end{split}

\end{equation}%explain/verify

%Where the last step involved a substitution.

\section{Mapping space on a line} \label{sec:Mapping space on a line}

Here's a fun fact: you can map all real numbers onto an interval like $[-1, 1]$. \linebreak

For instance the map $x \rightarrow \frac{2}{\pi}\arctan(x)$ is one such map. %bijective

\begin{wrapfigure}{l}{0.4\textwidth}

\centering

\includegraphics[width = 0.4\textwidth]{images/MA\_arctan.png}

\caption{the function $\frac{2}{\pi}\arctan(x)$}

\vspace{-10pt}

%\label{fig:}

\end{wrapfigure}

It is, perhaps even more fascinatingly, possible to map 2D space onto 1D space.

For instance given the coordinates of a 2D point $(...x\_1x\_0.x\_{-1}x\_{-2}..., ...y\_1y\_0.y\_{-1}y\_{-2}...)$ we can interleave their digits to obtain a single number: $x\_1y\_1x\_0y\_0.x\_{-1}y\_{-1}x\_{-2}y\_{-2}...$. (see \ref{fig:Z order})

Indeed this works for any number of finite dimensions.

Moreover, there exist mappings that are continuous (see \ref{fig:hilbert curve}).%

We can, in fact, devise a similar mapping valid for an infinite amount of dimensions:

Given a sequence $x[n]$ of numbers in $[0,1]$ with the $m$th most significant bit of the $n$th term denoted $x[n]\_m$ (with the indexes starting at 1) we can map the sequence $x[n]$ to the real number $0.x[1]\_1 x[2]\_1 x[1]\_2 x[3]\_1 x[2]\_2 \dots$ (see \ref{fig:snake}).

These schemes are collectively termed space-filling curves (or, often, the term "curve" is reserved for continuous mappings).

\begin{figure}[!h]

\centering

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/hilbert1.png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/hilbert2.png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/hilbert4.png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/hilbert8.png}

\end{subfigure}

\caption{The Hilbert curve. Iterations 1, 2, 4, 8. A continuous map from $[0,1]$ to $[0,1] \times [0,1]$. \cite{Hilbert\_Curve}}

\label{fig:hilbert curve}

\end{figure}

\begin{figure}[!h]

\centering

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/Z (1).png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/Z (2).png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/Z (4).png}

\end{subfigure}

\begin{subfigure}{0.2\textwidth}

\centering

\includegraphics[width=\columnwidth]{images/Z (8).png}

\end{subfigure}

\caption{The Z order (or Morton curve). Iterations 1, 2, 4, 8. A discontinuous map from $[0,1]$ to $[0,1] \times [0,1]$. Achieved by interleaving the bits of the $x$ and $y$ coordinates. \cite{Z\_Order}}

\label{fig:Z order}

\end{figure}

%\begin{figure}[H]%{R}{0.4\textwidth}

% \centering

% \includegraphics[width = 0.4\textwidth]{images/moore\_curve.png}

% \caption{The moore curve. A continuous map from $[0,1]$ to $[0,1] \times [0,1]$}

% \label{fig:moore curve}

%\end{figure}

\begin{figure}

\centering

\begin{tikzpicture}

\draw[->] (0,0) -- (6,0);

\draw[->] (0,0) -- (0,-6);

\draw[color=red, very thick]

(0.5, -0.5) -- (0.5, -1.5) -- (1.5, -0.5) -- (2.5, -0.5) -- (1.5, -1.5) -- (0.5, -2.5) -- (0.5, -3.5) -- (1.5, -2.5) -- (2.5, -1.5) -- (3.5, -0.5) -- (4.5, -0.5) -- (3.5, -1.5) -- (2.5, -2.5) -- (1.5, -3.5);

\draw[color=red, very thick, dashed]

(1.5, -3.5) -- (0.5, -4.5);

\draw[]

(0.5, 0) node[anchor=south]{1}

(1.5, 0) node[anchor=south]{2}

(2.5, 0) node[anchor=south]{3}

(3.5, 0) node[anchor=south]{4}

(4.5, 0) node[anchor=south]{5}

(5.5, 0) node[anchor=south]{$\dots$};

\draw[]

(0, -0.5) node[anchor=east]{1}

(0, -1.5) node[anchor=east]{2}

(0, -2.5) node[anchor=east]{3}

(0, -3.5) node[anchor=east]{4}

(0, -4.5) node[anchor=east]{5}

(0, -5.5) node[anchor=east]{$\vdots$};

\draw[]

(0.5, -0.5) node[fill=white]{\small$x[1]\_1$}

(1.5, -0.5) node[fill=white]{\small$x[1]\_2$}

(2.5, -0.5) node[fill=white]{\small$x[1]\_3$}

(3.5, -0.5) node[fill=white]{\small$x[1]\_4$}

(4.5, -0.5) node[fill=white]{\small$x[1]\_5$}

(5.5, -0.5) node[fill=white]{\small$\dots$}

(0.5, -1.5) node[fill=white]{\small$x[2]\_1$}

(1.5, -1.5) node[fill=white]{\small$x[2]\_2$}

(2.5, -1.5) node[fill=white]{\small$x[2]\_3$}

(3.5, -1.5) node[fill=white]{\small$x[2]\_4$}

(4.5, -1.5) node[fill=white]{\small$x[2]\_5$}

(5.5, -1.5) node[fill=white]{\small$\dots$}

(0.5, -2.5) node[fill=white]{\small$x[3]\_1$}

(1.5, -2.5) node[fill=white]{\small$x[3]\_2$}

(2.5, -2.5) node[fill=white]{\small$x[3]\_3$}

(3.5, -2.5) node[fill=white]{\small$x[3]\_4$}

(4.5, -2.5) node[fill=white]{\small$x[3]\_5$}

(5.5, -2.5) node[fill=white]{\small$\dots$}

(0.5, -3.5) node[fill=white]{\small$x[4]\_1$}

(1.5, -3.5) node[fill=white]{\small$x[4]\_2$}

(2.5, -3.5) node[fill=white]{\small$x[4]\_3$}

(3.5, -3.5) node[fill=white]{\small$x[4]\_4$}

(4.5, -3.5) node[fill=white]{\small$x[4]\_5$}

(5.5, -3.5) node[fill=white]{\small$\dots$}

(0.5, -4.5) node[fill=white]{\small$x[5]\_1$}

(1.5, -4.5) node[fill=white]{\small$x[5]\_2$}

(2.5, -4.5) node[fill=white]{\small$x[5]\_3$}

(3.5, -4.5) node[fill=white]{\small$x[5]\_4$}

(4.5, -4.5) node[fill=white]{\small$x[5]\_5$}

(5.5, -4.5) node[fill=white]{\small$\dots$}

(0.5, -5.5) node[fill=white]{\small$\vdots$}

(1.5, -5.5) node[fill=white]{\small$\vdots$}

(2.5, -5.5) node[fill=white]{\small$\vdots$}

(3.5, -5.5) node[fill=white]{\small$\vdots$}

(4.5, -5.5) node[fill=white]{\small$\vdots$}

(5.5, -5.5) node[fill=white]{\small$\ddots$};

\end{tikzpicture}

\caption{mapping a sequence of values in $[0, 1]$ to $[0, 1]$}

\label{fig:snake}

\end{figure}

Inspired by this, we might be interested in the prospect of making a synthesizer that could generate any "reasonable" sound wave by varying a single parameter.

Or perhaps having one parameter for the pitch of the sound, another for the volume and yet another for the timbre.

We may formulate the first idea as trying to find a

%continuous

surjective mapping from the real numbers to the set of real functions.

Unfortunately, the set of real functions is larger in cardinality than the set of real numbers.

Let $\{-1, 1\}^\R$ denote the set of functions $x: \R \rightarrow \{-1, 1\}$.

Clearly $\{-1, 1\}^\R$ is a subset of $\R^\R$; the set of functions from real numbers to real numbers.

Now we can use Cantor's famous diagonal argument to prove that $|\{-1, 1\}^\R|>|\R|$:

Suppose there exists a function $L: \R^+ \rightarrow \{-1, 1\}^\R$ whose range is all of $\{-1, 1\}^\R$ i.e. for any $x \in \{-1, 1\}^\R$ there exists an $r \in \R^+$ such that $L(r)(t)=x(t)$.

Then we could construct a function $d(t)=-L(t)(t) \in \{-1, 1\}^\R$. Then for any $r$, $d(t) \neq L(r)(t)$. But this means $d$ is not in the range of $L$ which contradicts our assumption.

This is not much of an issue however, restricting ourselves to the set of continuous real functions for instance already, perhaps surprisingly, yields a cardinality of $|\R|$.

This is because one can map the set of continuous functions to the set of real sequences $\R^\N$, which we have already proven to have the same cardinality as $\R$, by mapping each continuous function to its values on all the rational points.

Since the rational points are dense in $\R$, this determines the function. \cite{271641}

%Continuity is more elusive however. That there is no uniformly continuous map from the real numbers to a sequence space of real numbers, with distance defined uniformly across numbers in the sequence, can be proven by contradiction:

%If there were a distance $d$ such that

If we restrict ourselves to periodic functions of a specific frequency and sound measure (e.g. loudness/energy) we can identify each such function with its Fourier series and constrain its sound measure to a certain value.

Choosing the sound measure $\mu\{x\}=\sum\_{f} |\hat{x}(f)|^2$ for $f$ being the multiples of the fundamental frequency, functions of equal measure conveniently lie on a sphere (by the Pythagorean theorem) on which each function can be given a spherical coordinate.

%(this will be explained in more depth in chapter \ref{ch:code})

The sequence of coordinate values can then be mapped onto the real numbers as mentioned before.

It would be convenient if we could devise such an infinite dimensional space filling scheme that was also continuous. Unfortunately this may very well be impossible. Consider the following impossibility result: There cannot be a \gls{uniformly continuous} map $m$ from $[0,1]$ to the subset of $\gls{l2}$ of sequences with elements in $[0,1]$, denoted here as $\N^{[0,1]}$, because if there were, then, for a distance $\varepsilon < 1$, there would be some cutoff bit after which changing bits would no longer result in a change greater than $\varepsilon$ which means that there are finitely many bits that effect $m$ by more than $\varepsilon$ and hence a finite amount $N$ of images of $m$ that are less than $\varepsilon$ away from any other element of $\N^{[0,1]}$. But as the number of dimensions $D \rightarrow \infty$ the combined measure of the regions no further than $\varepsilon$ from this finite set of images tends to $\lim\_{D \rightarrow \infty} N\varepsilon^D = 0$ while the measure of the entire space stays $1$ which concludes the contradiction.\\

This impossibility result can certainly be generalized but I won't do so here due to time constraints.

\chapter{Code}\label{ch:code}

\raggedright

\section{The JUCE framework}

There are multiple formats audio plugins come in. A major one is Steinberg's \gls{VST} format. Steinberg does provide a \gls{SDK}\cite{Steinberg\_SDK} but unfortunately it has quite poor documentation. That is why I will be using the JUCE framework \cite{JUCE\_Download}. Not only is it better documented, but it also compiles the code you write into multiple different formats. JUCE does require you to pay if you want to commercialize your plugin though.

Both the Steinberg \gls{VST} \gls{SDK} and JUCE are in C\texttt{++}.

When you create a new audio plugin project in JUCE (see \cite{Plugin\_Tutorial}) the framework provides you with four files meant to be edited:

\begin{itemize}

\item PluginEditor.cpp

\item PluginEditor.h

\item PluginProcessor.cpp

\item PluginProcessor.h

\end{itemize}

The editor files are meant for the user interface while the processor files are meant for the actual calculations.

Inside the PluginProcessor.cpp file there are three main functions you may want to edit:

\begin{itemize}

\item The constructor of the \cppinline{PluginNameAudioProcessor} class, which is where you might initialize your plugin

\item The \cppinline{prepareToPlay} function, which is called before playback and passes the sample rate (and the expected maximum size of the audio blocks)

\item The \cppinline{processBlock} method, which is where the actual processing happens

\end{itemize}

The \cppinline{processBlock} method passes a buffer, with variable name \cppinline{buffer}, that serves to store both the input and the output audio and a separate \gls{MIDI} buffer, with variable name \cppinline{midiMessages}, that serves to store both the input and output \gls{MIDI} messages.

Memory can be deallocated in the \cppinline{releaseResources} function called after playback ends or the destructor of the \cppinline{PluginNameAudioProcessor} class depending on the nature of the memory.

Variables used in PluginProcessor.cpp can be declared in PluginProcessor.h.

\section{First sounds}

Let us first get our feet wet by coding synthesizer that plays a sine wave at $\mathrm{A4}=440\mathrm{Hz}$. Later we will extend its functionality to a polyphonic \gls{MIDI} \gls{wavetable} synthesizer.

We need to do the following:

\begin{enumerate}

\item Initialize a lookup table for our sine wave inside the constructor

\item compute the conversion factor \cppinline{freqToSampleRatio} for converting frequencies in $\frac{\mathrm{cycles}}{\mathrm{second}}$ to frequencies in $\frac{\text{lookup table samples}}{\text{buffer samples}}$ in the \cppinline{prepareToPlay} function

\item write to the output buffer in the \cppinline{processBlock} method

\item declare all the variables left to declare in PluginProcessor.h

\end{enumerate}

Here's the code:\\

Step 1:

\begin{lstlisting}[language=c++]

SynthAudioProcessor::SynthAudioProcessor()

//...

{

for (int i = 0; i < lookupSamples; ++i)

{

lookupArray[i]

= sin(juce::MathConstants<float>::twoPi \* i / lookupSamples);

}

}

\end{lstlisting}

Step 2:

\begin{lstlisting}[language=c++]

void SynthAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)

{

freqToSampleRatio = lookupSamples / sampleRate;

}

\end{lstlisting}

Step 3:

\begin{lstlisting}[language=c++]

void SynthAudioProcessor::processBlock

(juce::AudioBuffer<float>& buffer,

juce::MidiBuffer& midiMessages)

{

//clear any residual garbage in the buffer

buffer.clear();

//we write to all channels in the buffer by writing

//to all channels in buffer.getArrayOfWritePointers()

auto\* buffers=buffer.getArrayOfWritePointers();

for (auto sample = 0;

sample < buffer.getNumSamples();

++sample)

{

for (auto channel = 0;

channel < buffer.getNumChannels();

++channel)

{

buffers[channel][sample] = lookupArray[(int)position];

}

//Increment position. When position reaches

//the end of the table it has to wrap around

if ((position += A4 \* freqToSampleRatio)

> (float)lookupSamples)

position -= (float)lookupSamples;

}

}

\end{lstlisting}

Step 4:

\begin{lstlisting}[language=c++]

class SynthPlaygroundAudioProcessor : public juce::AudioProcessor

//...

{

//...

private:

const float A4 = 440.f;

static const int lookupSamples = 1 << 10;

float lookupArray[lookupSamples] = {};

float freqToSampleRatio = 0;

float position = 0;

}

\end{lstlisting}

(To test out the plugin see \cite{Plugin\_Tutorial}.)

\section{Coding a MIDI synthesizer}

Implementing \gls{MIDI} functionality makes things more complicated. The \gls{MIDI} protocol communicates whether a note is playing by sending a NoteOn message when it starts and a NoteOff message when it ends. These messages are stored in the \gls{MIDI} buffer. We will implement \gls{MIDI}-handling by keeping a list of playing "voices", iterating through the MIDI messages in the buffer, for each message rendering the section between this message and the previous one and then updating the list of voices according to the \gls{MIDI} message supplied.

Let us define the list of voices to be of type \cppinline{std::map<int, VoiceData>} where VoiceData is a class that stores the variable \cppinline{VoiceData::position} for each voice (delete the old variable \cppinline{position}) and \cppinline{frequency \* freqToSampleRatio} as \cppinline{delta}:

\begin{lstlisting}[language=c++]

class SynthPlaygroundAudioProcessor : public juce::AudioProcessor

//...

{

//...

private:

//...

struct VoiceData

{

float position = 0;

float delta = 0;

VoiceData(float initPosition, float initDelta)

{

position = initPosition;

delta = initDelta;

}

};

std::map<int, VoiceData> voices;

}

\end{lstlisting}

Now in the processBlock method:

\begin{lstlisting}[language=c++]

void LineSynthAudioProcessor::processBlock

(juce::AudioBuffer<float>& buffer,

juce::MidiBuffer& midiMessages)

{

buffer.clear();

//Add empty buffer end message

//This is necessary to ensure iterating through

//midiMessages reaches the end of the buffer

midiMessages.addEvent(juce::MidiMessage(), buffer.getNumSamples());

auto\* buffers = buffer.getArrayOfWritePointers();

auto lastSample = 0;

for (auto midiMetadata : midiMessages)

{

for (auto sample = lastSample;

sample < midiMetadata.samplePosition;

++sample)

{

for (auto& voice : voices)

{

for (auto channel = 0;

channel < buffer.getNumChannels();

++channel)

{

buffers[channel][sample]

+= lookupArray[(int)voice.second.position];

}

if ((voice.second.position += voice.second.delta)

> (float)lookupSamples)

voice.second.position -= (float)lookupSamples;

}

}

auto midiMessage = midiMetadata.getMessage();

if (midiMessage.isNoteOn())

{

float frequency = A4 \* pow(2.f,

float(midiMessage.getNoteNumber() - 69) / 12.f);

voices.insert({ midiMessage.getNoteNumber(),

VoiceData(0.f, freqToSampleRatio \* frequency) });

}

else if (midiMessage.isNoteOff())

{

voices.erase(midiMessage.getNoteNumber());

}

lastSample = midiMetadata.samplePosition;

}

}

\end{lstlisting}

\section{Anti-aliasing}

If we had chosen not to initialize \cppinline{lookupArray} with a sine wave but instead something with a lot of high harmonics we may run into aliasing (see \ref{subsec:Aliasing}). To hear this, initialize \cppinline{lookupArray} with a \gls{square wave}.

\begin{lstlisting}[language=c++]

SynthAudioProcessor::SynthAudioProcessor()

//...

{

for (int i = 0; i < lookupSamples; ++i)

{

lookupArray[i] = (i < (lookupSamples / 2)) ? -0.5f : 0.5f;

}

}

\end{lstlisting}

You should be listening for frequencies that sound inharmonic, distorted or anything that makes the synth sound different in the higher registers.

So how do we avoid aliasing while preserving the harmonic richness when possible?

Well as per \ref{subsec:Aliasing} there are two simple ways to fix this:

\begin{itemize}

\item Store the Fourier transform of \cppinline{lookupArray} and perform an \gls{FFT} every time a voice is added with the aliasing frequencies removed

\item Convolve \cppinline{lookupArray} with an approximation of the $\text{sinc}$ function either when a voice gets added or when it is read.

\end{itemize}

Both methods incur a sizeable performance penalty.

Instead we will create an array of lookup tables that, for each power of two, stores the waveform with all higher frequencies removed. To determine which lookup table to use we calculate the highest non-aliasing harmonic $h$ from the \gls{fundamental frequency} $f$ and the sampling frequency $f\_s$:

\begin{equation}

f \* h < \frac{f\_s}{2}

\end{equation}

\begin{equation}

h < \frac{f\_s}{2f}

\end{equation}

Which gives the highest non aliasing power of $2$ to be $\floor{\log\_2(\frac{f\_s}{2f})}=\floor{\log\_2(\frac{f\_s}{2})-\log\_2(f)}$.

To implement this we have to create an instance of \cppinline{juce::dsp::FFT} of order \cppinline{fftOrder} (make sure you add the dsp module in the Projucer app [see \cite{Projucer\_Tutorial}]), make \cppinline{fftOrder} lookup tables and add a pointer to our VoiceData class that points to the lookup table to be used. I will also add a variable \cppinline{logNyquist} to store $\log\_2(\frac{f\_s}{2})$ and another array of floats \cppinline{fftArray}. We are going to use the functions \cppinline{juce::dsp::FFT::performRealOnlyForwardTransform()} and \cppinline{juce::dsp::FFT::performRealOnlyInverseTransform()}. \cppinline{juce::dsp::FFT::performRealOnlyForwardTransform()} will replace the input array with the real and complex components of each successive frequency component in alternating fashion. \cppinline{juce::dsp::FFT::performRealOnlyForwardTransform()} actually operates on an array of floats but because each frequency component is stored as \emph{two} floats we need to pass an array of twice the size of the amount of data stored within it. Similarly \cppinline{juce::dsp::FFT::performRealOnlyInverseTransform()} will operate on arrays with twice the data as the output. All this is to say that if \cppinline{fftOrder} is the order of the \gls{FFT} and \cppinline{lookupSamples = 1 << (fftOrder)} is the amount of samples-in-use in \cppinline{lookupArray} and \cppinline{fftArray} we need \cppinline{lookupArray} and \cppinline{fftArray} to have length \cppinline{lookupSamples \* 2} to be able to use the \gls{FFT} on them.

This is how the variable declarations in PluginProcessor.h should look:

\begin{lstlisting}[language=c++]

class SynthPlaygroundAudioProcessor : public juce::AudioProcessor

//...

{

//...

private:

const float A4 = 440.f;

static const int fftOrder = 10;

static const int lookupSamples = 1 << (fftOrder);

float fftArray[lookupSamples \* 2] = {};

float lookupArray[fftOrder][lookupSamples \* 2] = {};

float freqToSampleRatio = 0;

float logNyquist = 1;

juce::dsp::FFT fft{ fftOrder };

struct VoiceData

{

float position = 0;

float delta = 0;

float\* dealiasedLookupArray = nullptr;

VoiceData(float initPosition, float initDelta, float\* initDealiasedLookupArray)

{

position = initPosition;

delta = initDelta;

dealiasedLookupArray = initDealiasedLookupArray;

}

};

}

\end{lstlisting}

When we initialize our plugin with our square wave we want to generate the dealiased versions as well:

\begin{lstlisting}[language=c++]

SynthAudioProcessor::SynthAudioProcessor()

//...

{

//initialize fftArray with our square wave

for (int sample = 0; sample < lookupSamples; ++sample)

{

fftArray[sample] = (sample < (lookupSamples / 2)) ? -0.5f : 0.5f;

}

//The true flag indicates that fft should only calculate

//nonnegative frequencies.

fft.performRealOnlyForwardTransform(fftArray, true);

//create dealiased copies

for (int cutoffOrder = 0; cutoffOrder < fftOrder; ++cutoffOrder)

{

//Copy all frequency components up to cutoffOrder.

//Leave the rest as 0.

//Remember, the frequency component at cutoffOrder is

//stored as two floats at indexes

//2 \* cutoffOrder and (2 \* cutoffOrder) + 1

for (int i = 0; i < (1 << cutoffOrder) \* 2; ++i)

{

lookupArray[cutoffOrder][i] = fftArray[i];

}

fft.performRealOnlyInverseTransform(lookupArray[cutoffOrder]);

}

}

\end{lstlisting}

Calculate \cppinline{logNyquist} in the \cppinline{prepareToPlay} function:

\begin{lstlisting}[language=c++]

void SynthPlaygroundAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)

{

freqToSampleRatio = lookupSamples / sampleRate;

logNyquist = log2(sampleRate / 2);

}

\end{lstlisting}

In the innermost for loop in \cppinline{processBlock} replace \cppinline{buffers[channel][sample] += LookupArray[(int)voice.second.position];} with \cppinline{buffers[channel][sample] += voice.second.dealiasedLookupArray[(int)voice.second.position];} and change the handling of a NoteOn event to:

\begin{lstlisting}[language=c++]

if (midiMessage.isNoteOn())

{

float frequency = A4 \* pow(2.f,

float(midiMessage.getNoteNumber() - 69) / 12.f);

voices.insert({ midiMessage.getNoteNumber(),

VoiceData{ 0.f, freqToSampleRatio \* frequency,

lookupArray[int(logNyquist - log2(frequency))] } });

}

\end{lstlisting}

\justifying

\chapter{Discussion}\label{ch:Discussion}

As is apparent, I have made the effort to explain a lot of math that I didn't end up using. I did intend on using all of what was explained in \ref{ch:Mathematical Groundwork} but due to very poor time management I failed to finish on time. As I had mentioned in \ref{sec:Mapping space on a line} I was, and still am, going to make a synthesizer plug-in that encodes all "reasonable" sound waves on a line (which could hopefully be represented on a slider). Unfortunately I am currently stuck at the debugging stage. Similarly, I was, and still am, going to make an effect plugin which can approximate other effect plugins via the Volterra series. I have not gotten started with that yet. The repositories for both will be provided in the Appendix.

Here is a run-down of the Ideas that I had not gotten to implementing:\\

For the space-filling curve synthesizer I was going to do the mapping in two stages:

\begin{enumerate}

\item Map the 1D input to a sequence of real numbers

\item Interpret the sequence as some kind of spherical coordinate

\end{enumerate}

Step 2 is to ensure that all inputs gave the same output volume.

Here are a couple of mappings I wanted to implement for step 1:

\begin{itemize}

\item Splicing Hilbert curves of successive dimensions end to end such that the resulting mapping would be continuous. This would have the downside of being ridiculously redundant but the advantage of being more gradual than most alternatives. It could also not represent sounds with infinitely many overtones.

\item Going through the array in figure \ref{fig:snake} in a Z order. This is probably the easiest to implement and does not have the downsides of the previous option. It, however has the downside of being discontinuous even when low-pass filtered.

\item Mapping the elements of \ref{fig:snake} with indexes $n,m$, $n + m = N$ for some fixed N into a first iteration Hilbert curve and splicing together. Could possibly preserve the advantages of the previous option while being continuous when low-pass filtered. (I am not quite sure about this one yet)

\end{itemize}

Here are a couple of mappings I wanted to implement for step 2:

\begin{itemize}

\item Interpret the input sequence as a spherical coordinate in frequency space

\item Interpret the input sequence as a polyspherical coordinate in the time domain

\end{itemize}

Overall I am quite unhappy with how this project turned out. I can only hope that I've learned my lessons.

If I were to give my past self some advice it would be:

\begin{itemize}

\item Meet with the supervisor more often to discuss "checkpoints"

\item Test functions in separate programs to see if they work

\end{itemize}

and of course:

\begin{itemize}

\item Start earlier

\end{itemize}

\newglossaryentry{operator}

{

name=operator,

description={Another term for function. Used to emphasize that it maps functions to functions}

}

\newglossaryentry{equalizer}

{

name=equalizer,

description={An audio effect that amplifies certain frequencies and suppresses others}

}

\newglossaryentry{reverb}

{

name=reverb,

description={The persistence of sound after it is produced}

}

\newglossaryentry{eigenfunction}

{

name=eigenfunction,

description={Analogue of an eigenvector for linear operators. An eigenfunction $f$ of a linear operator $M$ has the property that applying $M$ is identical to scaling $f$ by a factor $\lambda$; $M\{f(\tau\_M)\}(t)=\lambda f(t)$}

}

%\newglossaryentry{unitary operator}

%{

% name=unitary operator,

% description={A unitary operator generalizes the notion of an orthonormal matrix; a real matrix whose columns are all orthogonal and of unit length (i.e. normalized). Just like orthonormal matrices can be determined by their property of preserving the dot product, unitary operators are defined to be operators that preserve the inner product of their space. Since \emph{orthonormal} matrices preserve the dot product we can recover the input vector by taking the dot product of the output vector with each transformed basis vector, that is, their inverse is given by the transpose. Similarly since \emph{unitary} operators preserve the inner product we can recover the input by taking the inner product of the output with each transformed element of the standard basis, that is, in the case of $L^2$, the kernel of their inverse is given by the conjugate transpose}

%}

\newglossaryentry{L2}

{

name={\ensuremath{L^2}},

sort=L2,

description={The space of square-integrable functions, that is, for our purposes, the set of functions $x: \R \rightarrow \C$ for which $\int\_\R |x(t)|^2 dt$ converges. It is equipped with the inner product $\langle x, y \rangle = \int\_\R x(t) \overline{y(t)} dt$}

}

\newglossaryentry{angular frequency}

{

name=angular frequency,

description={Angle traversed per unit time}

}

\newglossaryentry{discrete}

{

name=discrete,

description={}

}

\newglossaryentry{multilinear}

{

name=multilinear,

description={Linear in all its arguments}

}

\newglossaryentry{unitary}

{

name=unitary,

description={Used to describe operators that preserve the inner product of their space. In $L^2$ the kernel function of the inverse of a unitary operator $n$, similar to orthonormal matrices, is given by the the conjugate transpose of the kernel function of the operator $m$: $n(\tau, t) = \overline{m(t, \tau)}$},

see={orthonormal}

}

\newglossaryentry{orthonormal}

{

name=orthonormal,

description={Used to describe real matrices that preserve the dot product, i.e. if $M$ is a real matrix acting on vectors $\mathbf{u},\mathbf{v}$ then $\mathbf{u}^\mathsf{T}\mathbf{v} = (M\mathbf{u})^\mathsf{T}M\mathbf{v}$. This implies that the images of the standard basis vectors ($\mathbf{e}\_i, \mathbf{e}\_j$, $i \neq j$) remain orthogonal ($\mathbf{e}\_i^\mathsf{T}\mathbf{e}\_j = 0$) and are of unit length (i.e. normalized; hence ortho\emph{normal})($\mathbf{e}\_i^\mathsf{T} \mathbf{e}\_i = 1$). Since an orthonormal matrix preserves the dot product $I^\mathsf{T}\mathbf{v} = (MI)^\mathsf{T}M\mathbf{v}$ (where $I$ is the identity matrix) whence $M^{-1} = M^\mathsf{T}$},

nonumberlist

}

\newglossaryentry{surjective}

{

name=surjective,

description={}

}

\newglossaryentry{map}

{

name=map,

description=Another term for function.

}

\newglossaryentry{l2}

{

name={\ensuremath{\ell^2}},

sort=l2,

description={The space of square-summable sequences, that is, for our purposes, the set of sequences $x: \N \rightarrow \C$ for which $\sum\_\R |x(t)|^2$ converges. It is equipped with the inner product $\langle x, y \rangle = \sum\_{t \in \N} x(t) \overline{y(t)}$}

}

\newglossaryentry{uniformly continuous}

{

name=uniformly continuous,

description={Used to describe functions. A function $x$ is said to be uniformly continuous if for all $\epsilon > 0$ there exists a $\delta > 0$ such that for all $s,t$ a distance less that $\delta$ apart $x(s), x(t)$ are less than a distance $\epsilon$ apart.}

}

\newglossaryentry{fundamental frequency}

{

name=fundamental frequency,

description=Inverse of the period of a periodic function

}

\newglossaryentry{indicator function}

{

name=indicator function,

description=A function that returns 1 when the argument is in the set and 0 otherwise

}

\newglossaryentry{wavetable}

{

name=wavetable,

description=A lookup table for a sound wave

}

\newglossaryentry{square wave}

{

name=square wave,

description={\includegraphics[width=0.4\textwidth]{images/MA\_Square\_Wave.png}}

}

\newglossaryentry{root of unity}

{

name=root of unity,

plural=roots of unity,

description={A root of unity of order N is a number $\omega$ such that $\omega^N=1$}

}

\newacronym{DAW}{DAW}{digital audio workstation}

\newacronym{DFT}{DFT}{discrete Fourier transform}

\newacronym{IDFT}{IDFT}{inverse discrete Fourier transform}

\newacronym{FFT}{FFT}{fast Fourier transform}

\newacronym{VST}{VST}{virtual studio technology}

\newacronym{SDK}{SDK}{software developement kit}

\newacronym{MIDI}{MIDI}{musical instrument digital interface}

\glsadd{orthonormal}

\chapter{Appendix} \label{ch:Appendix}

A Github repo containing this document as a PDF and the plugins once I finish them:\\

\qrcode{https://github.com/finxy/Programming-Audio-Plugins}