Bachelorarbeit

Im Studiengang Audiovisuelle Medien

Entwicklung eines Loop-basierten Musik-Systems in Max/MSP

Anhand gängiger Klangerzeugungs- und veränderungsverfahren

vorgelegt von

Jonas Mayr

an der Hochschule der Medien am 26. September 2025

zur Erlangung des akademischen Grades eines **Bachelor of Engineering**

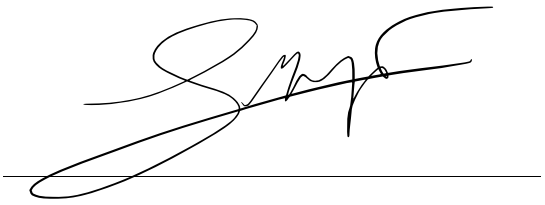
Erstprüfer*in: **Prof. Oliver Curdt**

Zweitprüfer*in: **Julien Herion**

Ehrenwörtliche Erklärung

Hiermit versichere ich, Jonas Mayr, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Entwicklung eines Loop-basierten Musik-Systems in Max/MSP – Anhand gängiger Klangerzeugungs- und veränderungsverfahren“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO, § 23 Abs. 2 Master-SPO (Vollzeit)) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.



Stuttgart, 25.09.2025 Jonas Mayr

Kurzfassung

In der vorliegenden Arbeit sollen die Möglichkeiten der visuellen Programmiersprache Max/MSP zur Realisierung grundlegender Verfahren der Klangsynthese sowie -verarbeitung untersucht werden. Nach einer Einführung in digitale Signalverarbeitung und klassische Synthesemethoden wurden verschiedene Module entwickelt, die in einem modularen Max/MSP-Patch zu einem Loop-basierten System zusammengeführt wurden. Dabei kamen unter anderem additive und subtraktive Synthese, Modulationsverfahren sowie Hüllkurven und LFOs zum Einsatz. Die Umsetzung verdeutlichte sowohl die Stärken von Max/MSP – insbesondere intuitive Visualisierung, Modularität und Erweiterbarkeit – als auch die Herausforderungen durch vielfältige Lösungswege. Das Ergebnis ist ein funktionales Werkzeug zur Erzeugung elektronischer Kompositionen, das theoretische Konzepte praxisnah demonstriert und eine Basis für weitere Entwicklungen bietet.

Abstract

This thesis examines the use of Max/MSP as a visual programming environment for implementing fundamental methods of sound synthesis and processing. After an introduction into concepts of digital audio and classical synthesis methods, multiple modules were developed and combined into a modular Max/MSP patch as loop-based system. The project implements additive and subtractive synthesis, modulation techniques, envelopes, and LFOs. The implementation demonstrates the strengths of Max/MSP, such as intuitive visualization and modular expandability, while also highlighting challenges caused by multiple possible approaches. The resulting system provides a functional tool for generating electronic compositions and serves as a foundation for future extensions.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation und persönliche Zielsetzung

Die Welt der elektronischen Musik ist vielfältig und geht einher mit der rapiden technologischen Entwicklung der letzten Jahrzehnte. Heutzutage sind die meisten Menschen in Deutschland im Besitz eines Computers mit einer Rechenleistung, welche die Ausführung aller gängiger digitaler Klangerzeugungsverfahren und weit darüber hinaus zuhause möglich macht. Mit Programmen wie Max/MSP wird zusätzlich der Einstieg in die Entwicklung solcher Programme zugänglicher gemacht.

Das Ziel dieser Arbeit ist es, ein Programm mit Hilfe der Sprache Max/MSP zu entwerfen, welches auf Basis von Loops die Generierung mehrerer unterschiedlicher musikalischer Teilelemente ermöglicht. Diese Teilelemente sollen hierbei modular gestaltet und anpassbar bzw. veränderbar sein und den Prinzipien klassischer Synthese- und Klangveränderungsverfahren folgen. Die Arbeit soll sich vorwiegend der Frage stellen, inwiefern das Entwickeln eines solchen Programms durch eine Umgebung wie Max/MSP ermöglicht wird.

1.2. Aufbau der Arbeit

Zuerst werden grundlegende theoretische Aspekte der digitalen Tontechnik geklärt, um ein Verständnis der Operationsweise von Max/MSP und dessen Objekte zu erlangen. Hier wird der digitale Teil eines Audiosignalverlaufs von Eingang bis Ausgang durchleuchtet und die wichtigsten Parameter und Einflussfaktoren eines solchen Systems besprochen. Danach wird die verwendete Entwicklungsumgebung Max/MSP vorgestellt. Hier wird vom Ursprung der Sprache und deren Konzept berichtet und die Struktur und der Signalfluss näher beleuchtet. Der nächste Punkt sieht die Einführung klassischer Klangsynthese- und veränderungsverfahren, von additiver Synthese und Frequenzmodulation bis zu Hüllkurven, sowie deren Implementierung in Max/MSP vor. Es werden die Grundbausteine für die spätere folgende praktische Ausarbeitung und deren Verständnis angefertigt. Die Regelung der zeitlichen Achse und die Synchronisierung von Abläufen

wird nachfolgend behandelt. Als nächstes wird die praktische Ausarbeitung beschrieben und die wichtigsten Bauelemente genauer betrachtet. Das Ende dieser Arbeit besteht aus einer Diskussionsrunde zu Max/MSP, dem Fazit der Ausarbeitung und einem Ausblick zu Erweiterungen des Programms.

2. Theoretische Grundlagen

2.1. Samplerate

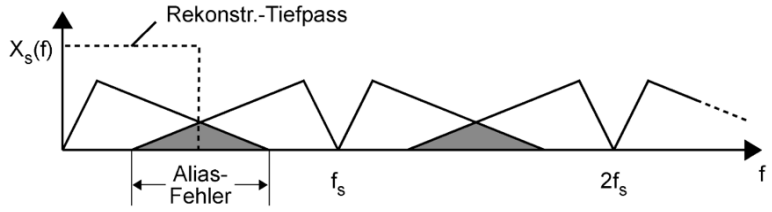
Ereignisse in der realen Welt erfolgen in der Regel nach analogem Prinzip, d.h. sie können durch ein kontinuierliches Signal beschrieben werden. Ein Computer kann nicht mit solchen Signalen umgehen und arbeitet daher samplebasiert. Das heißt, ein analoges Signal aus der realen Welt wird in bestimmten Zeitabständen abgetastet und die Werte innerhalb eines vorgegebenen Wertebereichs quantisiert. Dafür ist ein Analog-Digital-Wandler (A/D-Wandler) / Analog-to-Digital-Converter (ADC) notwendig. Dieser hat eine vorgegebene Samplerate (f_s), welche die Abstände der Abtastungen der Werte des eingehenden analogen Signals bestimmt. Bei einer in der Audioverarbeitung üblichen Abtastrate von 48k Hertz (Hz) werden demnach 48.000 Samples in der Sekunde dem eingehenden Signal entnommen [Analog Devices, Inc., 2004].

Um Fehler in der A/D-Wandlung zu vermeiden, muss bei der Abtastung das Nyquist-Theorem, auch bekannt als Abtasttheorem, eingehalten werden. Dieses besagt, dass die Abtastrate eines A/D-Wandlers mehr als dem doppelten der höchsten Frequenz des abzutastenden analogen Signals entsprechen muss, um es in der digitalen Welt akkurat darstellen zu können. Die für das menschliche Ohr hörbaren Frequenzen befinden sich in der Regel zwischen 16 und 20.000 Hz, daher werden in der Audioaufnahme und -verarbeitung Abtastraten jenseits der 40.000 Hz gewählt [Dickreiter et al., 2023]. Mit der CD hat sich erst 44,1 kHz etabliert, mittlerweile sind allerdings auch 48 kHz häufig vertreten. Mit letzterer können nach Nyquist-Theorem Frequenzen von bis zu 24 kHz korrekt wiedergegeben werden.

Wird die Grenze des Theorems überschritten, können irreversible Fehler in der Übertragung entstehen. Diese werden als Aliasing oder auch Aliasfehler bezeichnet. Treten Frequenzen oberhalb der Hälfte der Samplerate auf, werden die nach unten weg gespiegelt repräsentiert. Es wird beispielsweise von einer Samplerate von 48 kHz ausgegangen (ergo: 24 kHz = höchstmögliche Frequenz, welche fehlerfrei abgetastet werden kann). Enthält nun das abzutastende analoge Signal Energie an der Frequenz 28 kHz, tritt diese nach der A/D-Wandlung bei 20 kHz auf (Formel 2.1) (Abbildung 2.1 [Dickreiter et al., 2023, S. 793]). Es können also Frequenzen im gewandelten Signal vorkommen, welche im ursprünglichen Signal gar nicht vorhanden waren. In der Regel wird daher vor dem Abtasten des analogen Signals ein Tiefpassfilter eingesetzt, der das Signal auf die Hälfte der Abtastfrequenz begrenzt. Entspricht die Abtastfrequenz dem Abtasttheorem wird sie auch als Nyquist-Frequenz bezeichnet [Dickreiter et al., 2023].

$$f_{AF} = f_{Ny} - \left(f_a - f_{Ny}\right); wenn\ f_a > f_{Ny}$$

Abbildung 2.1
Aliasfehler durch Unterabtastunug



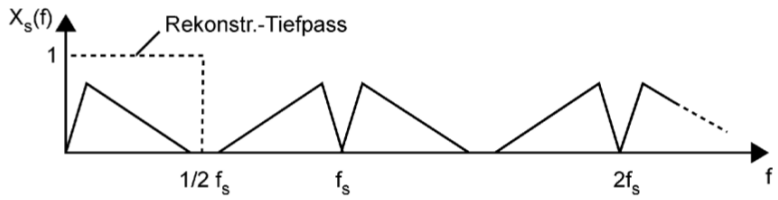
Am Ende der Signalkette steht in der Regel ein Digital-Analog-Wandler (D/A-Wandler) / Digital-to-Analog-Converter (DAC) der das Signal wiederum in eine analoges und kontinuierliches Signal wandelt. Diese Wandlung ist allerdings nur notwendig, wenn eine Ausgabe des Signals stattfinden soll. In dem Fall von Ton ist es üblicherweise die Ausgabe über Lautsprecher.

Auch in diese Richtung, der D/A-Wandlung, muss das Abtasttheorem beachtet werden. Für die korrekte Rekonstruierung des Eingangssignals, wird das digitale Signal vor der Wandlung ebenfalls mit einem Tiefpassfilter begrenzt. Die Grenzfrequenz wird auch hier mit der Hälfte der Abtastfrequenz bestimmt.

Bei einer Unterabtastung ($f_{Ny} \leq f_{a_{max}}$) des Eingangssignals oder einem Verzicht auf einen passenden Tiefpassfilter bei der A/D-Wandlung werden diese schon eingebrannten Aliasfehler bei der Rückwandlung in ein kontinuierliches Signal einfach übertragen.

Außerdem entsteht bei der Rückwandlung ein Spiegelbild des Signals bei f_{Ny} / $\frac{1}{2}f_s$ und das gesamte Signalabbild bis zur Samplerate wiederholt sich bei jedem ganzzahligem Vielfachen der Samplerate (Abbildung 2.2 [Dickreiter et al., 2023, S. 792]). Zur korrekten Wiedergabe des diskreten als kontinuierliches Signal wird auch hier ein Tiefpassfilter mit Grenzfrequenz f_{Ny} eingesetzt. Dieser Filter wird Rekonstruktionsfilter genannt [Dickreiter et al., 2023, S. 792].

Abbildung 2.2
Rekonstruktion des Basisbands durch Tiefpassfilterung



2.2. Bittiefe

Neben der Frequenz des Eingangssignals, müssen auch die kontinuierlichen Werte der Amplitude des analogen Signals in diskrete Werte quantisiert werden. Diese Quantisierung findet in einem vorgegebenen Bitbereich statt, auch Bittiefe (engl.: bit depth) genannt. Eine übliche Auflösungsgröße dafür sind 16 Bit in Binärcode. In das Dezimalsystem übersetzt sind das 2^{16}, also 65.536 Stellen, die angenommen werden können. Das heißt bei einer 16-Bit-Auflösung können Werte im Bereich von circa -32.000 bis +32.000 dargestellt werden (um die volle Auflösungsgröße ausnutzen zu können müssen in bipolaren Darstellungen die Werte asymmetrisch verteilt werden, da der Wert 0 auch angenommen werden können muss; das heißt z.B. -32.768 bis +32.767) [Dickreiter et al., 2023].

Bei der Übersetzung entstehen automatisch Quantisierungsfehler. Ein analoges Signal ist nicht nur zeitlich kontinuierlich, sondern auch in dessen Amplitude. Nimmt das Signal

Werte z.B. im von -1 und 1 an, dann muss es im Verlauf des Signals automatisch auch unendlich viele Werte dazwischen annehmen. Da das im Digitalen dann auf die angesprochene Bittiefe quantisiert wird, müssen die abgetasteten Werte gegebenenfalls gerundet werden. Wenn von einer Spannweite von -1 bis +1 V (Volt) bei dem analogen Signal und einer Bittiefe von 4 Bit in der Abtastung ausgegangen wird, entspricht jedes Sample einem Wertebereiche mit Breite von 0,125 V ($\frac{Amplitudenbereich}{2^{Bittiefe}}$). Alle Werte innerhalb eines solchen Bereichs werden dann demselben Wert zugewiesen. Dadurch können bei gegebenem Fall Rundungsfehler mit einer Größe von 0,125 V beziehungsweise 1/16 des gesamten Amplitudenbereichs entstehen. Das Ausmaß dieser als Quantisierungsfehler bezeichneten Abweichungen kann durch Verwendung einer größeren Bittiefe stark verringert werden. Bei 8 Bit ist die maximale Abweichung nur noch 1/256 des gesamten Amplitudenbereichs, und bei den heutzutage üblichen 16 (Compact Disc Digital Audio, CD-DA) und 24 Bit ist es 1/65.536 bzw. 1/16.777.216 des gesamten Amplitudenbereichs (oder 30,52 µV bzw. 0,12 µV) [Dickreiter et al., 2023].

Diese Quantisierungsfehler können im Endprodukt meist als weißes Rauschen (da die Abweichungen der Werte dem Menschen in der Regel zufällig erscheinen) wahrgenommen werden. Daher ist dieser Effekt als Quantisierungsrauschen bekannt. Wie stark das Rauschen im Verhältnis zum eigentlichen Signal steht, kann durch die Signal-to-Quantization-Noise-Ratio (SQNR; dt.: Signal-zu-Quantisierungsrausch-Verhältnis) beschrieben werden. Die SQNR wird in dB angegeben und ist die Differenz zwischen der maximal möglichen Amplitude des Signals und dem durchschnittlichen Pegel des Quantisierungsrauschens [Cycling ’74, o.J.c]. Wenn von einer Full-Scale-Sägezahnwelle als Eingangssignal ausgegangen wird lautet die Formel zur Berechnung des SQNR:

$$SQNR_{saw} = 6{,}02 * Bittiefe$$

Entsprechende SQNR-Werte:

Tabelle 2.1
Ausgewählte SQNR-Werte zu entsprechender Bittiefe

8 Bit (z.B. ISDN [Fine, 1995])	$SQNR_{saw} = 48{,}16\ dB$
16 Bit (z.B. CD-DA [Thomas, 2024])	$SQNR_{saw} = 96{,}32\ dB$
24 Bit (z.B. ALAC [Apple Inc., o.J.])	$SQNR_{saw} = 144{,}48\ dB$
32 Bit	$SQNR_{saw} = 192{,}64\ dB$

Wird von einer Full-Scale-Sinuswelle als Eingangssignal ausgegangen lautet die Formel zur Berechnung des SQNR:

$$SQNR_{sin} = 6{,}02 * Bittiefe + 1{,}76\ dB$$

Entsprechend liegen die Werte der $SQNR_{sin}$ bei gleicher Bittiefe des Systems immer 1,76 dB über dem der $SQNR_{saw}$. Das liegt an dem um 1,76 dB höher liegendem Effektivwert eine Full-Scale-Sinuswelle gegenüber einer Full-Scale-Sägezahnwelle [Kester, 2009]. Wichtig ist es auch anzumerken, dass die Werte den Spielraum bei Ausnutzung maximaler Amplitude (Full-Scale) des Eingangssignals beschreiben.

Als unterste Hörschwelle des Menschen wird $2 * 10^{-5}\ Pa$ (Pascal) Schalldruck bei 1 kHz definiert. Die Schmerzgrenze bei einem Schalldruck von $1{,}5 * 10^{2}\ Pa$. Der Schalldruckpegel (SPL) verwendet $2 * 10^{-5}\ Pa$ als Referenzwert, daher ist $2 * 10^{-5}\ Pa = 0\ dB_{SPL}$ [Dickreiter et al., 2023]. Zur Bestimmung des dB_{SPL}-Wertes der Schmerzgrenze gilt folgende Gleichung:

$$20 * \log_{10}\left(\frac{1{,}5 * 10^{2}}{2 * 10^{-5}}\right) \approx 138\ dB_{SPL}$$

Das bedeutet die Spanne zwischen dem niedrigsten für den Menschen hörbaren Schalldruck und dem größten, der Schmerzgrenze, beträgt circa 138 dB. Das lässt darauf schließen, dass eine Bittiefe von 24 ausreicht, um den gesamten für den Menschen relevanten Dynamikumfang darstellen zu können. Ein Grundsatz besagt, dass pro zusätzlichem Bit in der Auflösung etwa 6 dB auf die SQNR addiert werden können [Chaudhari, o.J.].

2.3. Digitale Signalverarbeitung

Zwischen dem ADC und dem DAC befindet sich die digitale Signalverarbeitung (DSP, für Digital Sound Processing oder Digital Sound Processor) (Abbildung 2.3 [Dickreiter et al., 2023, S. 786]). Hier wir das Signal gespeichert, übertragen (z.B. in der Rundfunktechnik vom Studio in die Regie) und / oder verarbeitet (z.B. durch Verstärkung, Filter, Kompressoren, etc.). In der Verarbeitung werden in der Regel unterschiedliche mathematische Operationen verwendet, um zu dem gewünschten Ergebnis zu kommen, z.B. Multiplikationen zur Pegelanhebung, Additionen zur Summierung mehrerer Signale oder die Multiplikation des Spektrums eines Eingangssignals mit der Übertragungsfunktion eines Filters zur Filterung eines Signals. In einem analogen Signalverarbeitungsstrang werden gewöhnlich vorgefertigte elektronische Schaltungen verwendet. Darin liegt ein großer Vorteil des DSP. Durch die Programmierbarkeit der Hardware-Elemente, können Verarbeitungsschritte ausgetauscht und abgeändert werden, ohne etwas an der Hardware abändern zu müssen. Dies kann teilweise sogar während eines laufenden Verarbeitungsprozesses geschehen.

Abbildung 2.3
Struktur eines digitalen Audiosystems

Wenn der eingehende Datenfluss nach einer A/D-Wandlung direkt in das DSP übergeht, demnach der Weg von der Verarbeitung zum Ausspielen des Audiosignals nur wenige Millisekunden beansprucht, kann von Realtime-Synthese (Echtzeit-Synthese)

gesprochen werden. Wird der Verarbeitungsprozess zuerst komplett berechnet, als eine vorliegende vollständige Audiodatei bereitgestellt und darauf als fertiges Produkt ausgespielt, kann von Non-Realtime-Synthese oder auch Offline-Synthese gesprochen werden [Cipriani & Giri, 2019].

Cipriani und Giri [2019] sehen drei Möglichkeiten der Signalverarbeitung:

- Präexistente Audiodatei, welche offline verarbeitet und als separate Audiodatei ausgespielt wird
- Präexistente Audiodatei, welche in Echtzeit verarbeitet und ausgespielt wird
- Echtzeit Audioeingabe, welche in Echtzeit verarbeitet und ausgespielt wird

Alle drei Prozesswege haben Vor- und Nachteile. Max/MSP bietet alle an, wobei hauptsächlich Echtzeitverarbeitungen Verwendung finden. Für offline Berechnungen muss in den Einstellungen ein einer anderer Audio-Treiber ausgewählt werden. Das heißt, dass die erste und zweite Art der Signalverarbeitung gleichzeitig innerhalb eines Programms stattfinden können, während die erste nur eigenständig verwendet werden kann.

In dieser Arbeit wird vor allem der Zweig der digitalen Signalverarbeitung eines Audiosystems in Bezug auf die Entwicklungsumgebung Max/MSP näher betrachtet und unterschiedliche Verarbeitungsprozesse durch diese vorgestellt.

3. Einführung in Max/MSP

3.1. Geschichte und Entwicklung von Max/MSP

Max ist eine grafische Entwicklungsumgebung zur Erstellungen visueller Programme des Herstellers Cycling ’74. Die Erweiterungen MSP und Jitter behandeln dabei einerseits die Audio- und Signalverarbeitung (MSP) und andererseits die Video-, Grafik- und Matrixdatenverarbeitung (Jitter). Max findet daher vor allem Verwendung in Musik und

Multimedia. Jitter ist für diese Arbeit irrelevant, daher wird die Behandlung dieser Erweiterung außen vor gelassen.

In den 1980ern entstand Max an dem Institut de recherche et coordination acoustique / musique (IRCAM) in Paris durch Miller Puckette. Zu diesem Zeitpunkt bediente Max nur externe digitale Synthesizer und keine eigenen Signalverarbeitungsfähigkeiten [Puckette, 1988]. Für eine Computer-Workstation (Computer mit hohen Rechenleistungsanforderungen [Lenovo Deutschland, o.J.]) des Herstellers NeXT Inc. arbeitete das IRCAM 1989 an einer Soundkarte für Echtzeit-Synthesizer namens IRCAM Signal Processing Workstation (ISPW). Puckette wandelte für diese Max um und fügte Objekte zur Audioverarbeitung hinzu. Max in Kombination mit der Audiofähigkeit war als Max/FTS bekannt. Diese Version wurde nicht nur am IRCAM, sondern auch etwa 30 weiteren Zentren und Studios verwendet. Ab 1990 wurde Max durch Opcode Systems, Inc. erstmals kommerziell vertrieben. 1994 wechselte Puckette zur Musik-Fakultät der University of California San Diego. Dort begann er die Entwicklung von Pure Data (Pd), welches als Ziel hatte, auf den damalig neuartigen Mikroprozessoren zu laufen. Die Anwendungsumstände von Max und Pd waren zwar unterschiedlich, aber beide Umgebungen konnten Echtzeit-Signalverarbeitung betreiben und wurden durch das grafische Verbinden von Objekten bedient. Nach dem Start der Entwicklung von Pd erweiterte David Zicarelli, welcher seit den späten 1980ern ein Entwickler der Max-Umgebung war, diese weiter [Cycling ’74, o.J.a]. Er ermöglichte Computern mit PowerPC-Mikroprozessor-Architektur die Fähigkeit zur Audiosignalverarbeitung mit Max. Daraus resultierte Max/MSP, welches die Signalverarbeitungsstruktur von Pd anwendete. Zicarelli transferierte außerdem Funktionen von Max/FTS, welche in Pd noch nicht implementiert wurden und entwickelte das User-Interface weiter.

Im Jahr 1997 gründete Zicarelli das Unternehmen Cycling ’74, welches erst nur die Erweiterung MSP anbot, jedoch bald die Entwicklung und den Vertrieb von Max selbst übernahm und bis heute weiterführt [Cycling ’74, o.J.a].

3.2. Grundkonzepte der visuellen Programmierung

Computerprogramme werden in der Regel mit textuellen Programmiersprachen umgesetzt. Beispielhaft hierfür sind Sprachen wie Python, Java oder C. Das Paradigma der visuellen Programmiersprache (Visual Programming Language; VPL) unterscheidet sich zu dem der textuellen Sprache in der Nutzung grafischer Elemente (Bilder, Formen und / oder andere Illustrationen) anstelle von Text als Syntax sowie in deren Semantik. Weitere Aspekte sind visuelle Informationen wie Farbe, Räumlichkeit, Tiefe oder Position.

VPLs werden in der Regel innerhalb einer visuellen Umgebung genutzt, in welcher die spezifisch verfügbaren Werkzeuge und Techniken zur Veränderung und Bearbeitung des Programms grafisch dargeboten oder mithilfe einer Benutzeroberfläche Verfügbar gemacht werden.

Den VPLs werden in der Regel vier Eigenschaften zugewiesen:

- Nur die für die Anwendung direkt relevante Logik wird dargestellt.
- Zum entstehenden Programm kann Feedback gegeben oder Spezifikationen bereitgestellt werden.
- Beziehungen zwischen Objekten oder Modulen werden auf eine sehr konkrete Art und Weise (z.B. durch Verbindungen oder Diagramme) dargestellt.
- Effekte der Veränderungen der Nutzer*innen werden automatisch und unmittelbar dargestellt.

VPLs können auch in rein visuelle Sprachen, Hybride aus Text und Visuellem sowie an Bedingungen orientierte Systeme bzw. auf Formen basierende Systeme eingeteilt werden.

Die Semantik rein visueller Sprachen wurde vollständig oder überwiegend aus grafischen Regeln abgeleitet. Diese Sprachen zeichnen sich in der Regel durch ein starke Abhängigkeit von visuellen Techniken während des gesamten Programmierprozesses aus. Das in einer rein visuellen Sprache erstellte Programm wird direkt aus seiner visuellen Darstellung kompiliert und nicht in eine textbasierte Sprache übersetzt [Ahmad, 1999].

3.3. Interface und Datenstruktur von Max/MSP

Die Entwicklungsumgebung Max ist Blockschaltbildern (bzw. Signalflussplänen) nachempfunden und zählt zu den objektbasierten und datenstromorientierten Programmiersprachen. Die einzelnen in Max entwickelten Programme werden Patches genannt und die Umgebung selbst (das Programmfenster) in der ein Programm erstellt wird ist ein Patcher. Innerhalb dieses Patchers können vorgefertigte Bauteile (Objects / Objekte) eingefügt und grafisch (als Kabel dargestellte Verbindungen) miteinander verknüpft werden. Die meisten Objekte bestehen aus Blöcken, die den Namen des Objekts und optional Argumente und Attribute tragen. Manche Objekte generieren ein eigenes grafisches Interface, dass von der eigentlichen Blockdarstellung abweicht, wie z.B. *toggle* (Ein/Aus-Schalter), *matrixctrl* (Schalter in einer Matrixanordnung), *itable* (Tabelle, z.B. als Wavetable siehe 4.1.1.) oder *umenu* (Dropdown-Liste). Manche dieser grafischen Objekte folgen dem Design Ableton Live‘s (z.B. *live.gain~* oder *live.dial*). Außerdem gibt es Message-Bauteile, welche eine Nachricht empfangen bzw. senden und mit spezifizierten Argumenten umgehen können und Comment-Bauteile zur Erstellung von Texten, welche der Erklärung oder Bezeichnung dienen. Bauteile können keine bis mehrere Eingänge sowie Ausgänge besitzen. Mehrere Patches können ineinander verschachtelt werden (ein Patch kann mehrere Subpatches beinhalten).

Nachrichten in Max können vier verschiedene Atomic-Data-Types (Atome; nicht weiter unterteilbare Datentypen) annehmen: Integer, Float, Symbol oder Bang. Ein Integer entspricht einem ganzzahligen Wert und Float einer Gleitkommazahl. Symbole sind aneinandergereihte Zeichen (*bang* ist eine Ausnahme und zählt als eigener Datentyp). Wenn die Zeichen nur aus Zahlen bestehen wird es als Integer bzw. Float interpretiert. Soll ein Zahlenwert als Symbol gewertet werden, muss dieser in Anführungszeichen gesetzt werden. Gleiches gilt, wenn Leerzeichen Teil eines Symbols sein sollen. Eine Liste besteht aus einer Folge dieser Atome. Einige komplexere Datentypen, wie XML oder JSON sind ebenfalls möglich [Cycling ’74, o.J.e].

3.4. Signalfluss in Max/MSP

Objekte in Max kommunizieren über als Kabel visualisierte Verbindungen, sogenannte Patch Cords. Die Kommunikation findet entweder durch Aktivität der Nutzer*in statt oder automatisch nach vorher bestimmten Bedingungen. Das heißt Max Objekte senden immer Nachrichten zu bestimmten Zeitpunkten. MSP Verbindungen sind dagegen in dauerhafter Kommunikation. Zwischen zwei MSP Objekten herrscht eine konstante Abhängigkeit. Durch diese Abhängigkeit werden Audioinformationen zu jeder Instanz berechnet. Diese Verbindung wird als „Signal-Network“ bezeichnet. Zu unterscheiden sind diese an ihrer Gestaltung. Max-Verbindungen sind grau, währen die Verbindungen in MSP hellgrün-dunkelgrün gestreift sind. Außerdem sind Objekte, welche mit ~ gekennzeichnet sind, fähig MSP-Verbindungen einzugehen.

MSP-Signale können der digitalen Technik bedingt auch keine kontinuierlichen Signale senden, daher lohnt es sich auch MSP-Verbindungen wie Max-Verbindungen zu sehen, allerdings mit einer deutlich höheren Sende- und Empfangsfrequenz. In MSP werden Nachrichten in Audiorate gesendet (z.B. 48 kHz, ergo 48.000 mal die Sekunde), während Max in Millisekunden arbeitet, also maximal 1.000 Nachrichten in der Sekunden versenden kann.

Es gibt Objekte, für die in Verbindungsetzung der Control-Rate (Max) und der Samplerate. Wenn Informationen von MSP in Max übersetzt werden sollen, geht das allerdings auch nur in der Control-Rate von maximal 1.000 Hz (*snapshot~*). Bestimmte Objekte können auch Informationen aus Sampling- sowie Control-Rate am selben Eingang annehmen [Cycling ’74, o.J.d].

Signale in Control-Rate müssen nicht immer Zahlen sein. Es können auch Strings (Symbole inkl. Buchstaben, Sonderzeichen, Zahlen, etc.) weitergegeben werden. Bestimmte Strings können je nach Objekt auch eine spezifische Funktion haben bzw. eine spezifische Operation oder Modifikation durchführen. Eine wichtige Nachricht (Message) in dem Control-Rate-System ist „bang”, sie aktiviert viele Objekte. Das bedeutet, wenn ein Objekt (welches eine bang-Nachricht auch empfangen kann) diese

Nachricht empfängt, führt es ihre Funktion aus, was zumeist für eine Absendung einer Nachricht des aktivierten Objekts sorgt [Cycling ’74, o.J.e].

4. Klangsynthese und -verarbeitung in Max/MSP

Die Entstehung elektronischer Musikinstrumente ging einher mit der Entwicklung der digitalen Signalverarbeitung, den DSPs. Um ein besseres Verständnis für elektronische Instrumente und elektronische Klangerzeugung zu bekommen, kann man in Klangsynthese und Klangveränderung bzw. -manipulation unterscheiden. Unter Syntheseverfahren versteht man die elektronische Erzeugung von Klängen. Gesteuert werden die Töne meist durch eine Klaviatur oder einen Controller, der dem MIDI-Standard (Musical Instrument Digital Interface) folgt. Das MIDI-Format orientiert sich hierbei ebenfalls an einer klassischen Klaviatur. Jede Note entspricht einer Zahl, die mit steigender Tonhöhe ebenfalls selbst ansteigt. Zusätzlich wird der Status einer Note, ob die Note angespielt wird oder nicht, beachtet und mit welcher Anschlagstärke die Note betätigt wird.

4.1. Objekte zur Klangerzeugung in Max/MSP

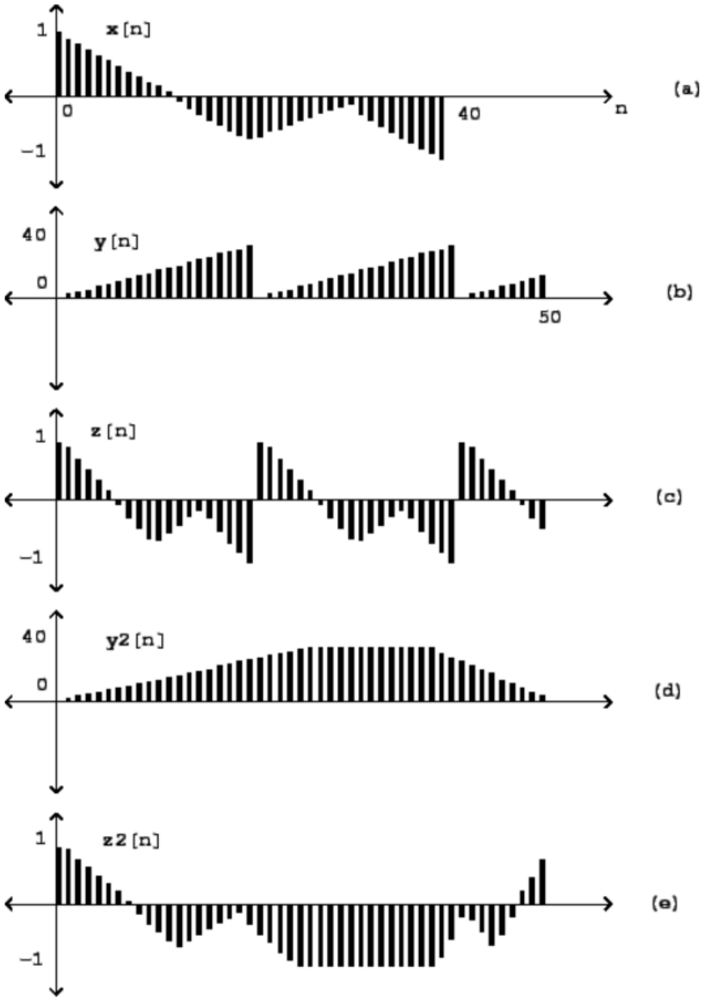
4.1.1. Wavetable-Lookup

Zur Erzeugung einer Wellenform können unterschiedlich Methoden vorkommen. Es können durch einen Eingang eingespielte Werte abgetastet und weitergeleitet werden oder durch eine vorher bestimmte Funktion in Echtzeit die passenden und gewünschten Werte berechnet werden. Es kann aber auch eine bestimmte Wellenform vorher berechnet und bestimmt werden und die dazu gehörigen Werte in Samples auf dem Speicher abgelegt werden. Diese könnten dann jederzeit nach Bedarf aufgerufen werden. Diese Vorgehensweise nennt man einen Wavetable-Lookup. Bei dieser Methode können die Werte aus einer solchen Tabelle dann auch auf jegliche Art und Weise und in einer beliebigen Reihenfolge ausgelesen werden [Dickreiter et al. 2023].

Die Abbildung 4.1 [Puckette, o.J.] zeigt ein Wavetable (a) und wie sich das resultierende Signal in Abhängigkeit der Lookup-Eingangssignale (anhand welcher die Wavetable ausgelesen wird) unterscheiden kann. Die auszulesende Wavetable hat hier ein Länge von 40 Samples (mit den Indizes 0 – 39; auf der x-Achse). Die Eingangssignale können Werte in ihrer Amplitude von 0 bis 39 annehmen (auf der y-Achse). Die Amplitudenwerte dieser Signal werden demnach den Indexwerten der Wavetable zugeordnet.

Das Eingangssignal (b) stellt hier einen Sägezahn, ergo eine Rampe, mit Werten von 0 bis 39 und 2,5-facher Periode dar. Das resultierende Signal ist (c), was (a) mit 2,5 Zyklen darstellt. Das Eingangssignal (d) ist ein Rampe mit Plateau bei 39 und schließlich einem Abfall bis auf 0. Daraus entsteht eine einfacher Zyklus der Wavetable mit halten des letzten Wertes und dann einem schnellerem rückwärts Auslesen der Werte der Tabelle.

Abbildung 4.1
Wavetable-Lookup. (a): Ein Wavetable; (b) und (d): Eingangssignale zur Auslesung der Wavetable; (c) und (e): Entsprechende Ausgänge



4.1.2. Wellenform- und Rauschgeneratoren in Max/MSP

Max/MSP bietet unterschiedliche Methoden zur Erstellung von Oszillatoren. Für klassische Wellenformen (Sinus, Sägezahn, Rechteck und Dreieck) sind die einfachsten und für diese Arbeit relevanten die Objekte *cycle~*, *saw~*, *rect~* und *tri~*. Zur Erzeugung eines weißen oder pinken Rauschen die Objekte *noise~* und *pink~*.

cycle~

Dieses Objekt wird in der Regel zur Erzeugung einer sinusoidalen Funktion bzw. Wellenform verwendet. Diese bildet die fundamentalste Form einer Welle in der Welt des Tons, da sie idealerweise Energie nur an einer Frequenz vorweist [Dobrian, o.J.]. In Max/MSP wird hierbei standardmäßig eine Periode einer Kosinuswelle generiert. Dabei werden die entsprechenden Werte aus einer Wavetable über Wavetable-Lookup periodisch ausgelesen. Mit dem *Set*-Attribut und einem weiteren Objekt (*buffer~*) kann hier jedoch auch eine beliebige Wavetable integriert werden [Cycling ’74, o.J.b].

saw~

Zur Erzeugung eines Sägezahns wird das Objekt *saw~* (für sawtooth) verwendet. Eine Sägezahnwelle weißt neben der Grundschwingung (Fundamentalen) auch an allen harmonischen Obertönen Energie vor. Die Amplitude der Obertöne lässt sich hierbei proportional zur Fundamentalen mit 1 geteilt durch die Ordnungszahl der harmonischen Frequenz beschreiben (ergo 1/1, 1/2, 1/3, 1/4, usw.). Das Objekt *saw~* basiert auf einem weiteren Objekt namens *phasor~*, welches eine Rampe von 0 bis (fast) 1 generiert [Dobrian, o.J.], allerdings hauptsächlich Anwendung in zeitbasierten Steuerungen oder anderen Steuersignalen findet und selten als Klangsignal verwendet wird [Cycling ’74, o.J.g]. Bei einem klassischen Sägezahn wird außerdem von einem bipolaren Signal ausgegangen, demnach muss das *phasor~*-Signal entsprechend skaliert werden. Da ein idealer Sägezahn ein obertöniges Signal mit unendlich vielen Summanden (Vielfachen des Grundtons) ist, wird zusätzlich eine Bandbegrenzung zur Vermeidung von Aliasfehlern benötigt [Cycling ’74, o.J.j].

rect~

Das Objekt *rect~* (für rectangle) erzeugt eine rechteckige Welle. Eine solche besteht neben der Fundamentale auch aus den ungerade harmonischen Obertönen. Die Energie dieser Harmonischen wird, wie bei dem Sägezahn, mit 1 geteilt durch die Ordnungszahl des jeweiligen Obertons beschrieben (ergo 1/1, 1/3, 1/5, 1/7, usw.) [Dobrian, o.J.]. Es gilt dasselbe wie bei *saw~*, da auch das Signal einer idealen Rechteckwelle ins unendliche obertönig wäre und muss deshalb ebenfalls bandbegrenzt werden muss, um keine Aliasfehler aufzuweisen [Cycling ’74, o.J.i]. Zusätzlich kann bei diesem Objekt der sogenannte Duty-Cycle mit einem Wert von 0 bis 1 bestimmt werden. Wenn von einem bipolaren Signal ausgegangen wird, sind bei einer regulären Rechteckwelle die positiven und negative Abschnitte von selber Länge. Das Verhältnis dieser Abschnitte wird durch den Duty-Cycle beschrieben. In diesem Fall beträgt der Wert des Duty-Cycles üblicherweise 0,5. Wird dieser Wert modifiziert verändert sich das Verhältnis der bipolaren Abschnitte zueinander. Liegt der Wert bei 0,25, ist die postive Phase 1/4 des Zykluses lang, die negative 3/4. Bei einem Wert von 0,75 verhält sich die Welle genau umgekehrt.

tri~

Für die Generierung einer Dreieckwelle wird das Objekt *tri~* (für *triangle*) verwendet. Eine solche Welle besteht, wie auch eine rechteckige, aus der Fundamentalen sowie ihren ungeraden Harmonischen. Im Vergleich zur Rechteckwelle verfügen diese Obertöne allerdings von deutliche geringerer Energie. Die Amplituden dieser Frequenzen werde mit 1 geteilt durch die Ordnungszahl der Oberschwingung im Quadrat berechnet (ergo 1/1, 1/9, 1/25, 1/49, usw.) [Dobrian, o.J.]. Auch hier gilt, dass eine ideale Form dieser Welle endlos viele Obertöne vorweisen würde, welche dann ebenfalls zu Aliasfehlern führen würden. Daher wird hier gleichermaßen eine Bandbegrenzung eingesetzt [Cycling ’74, o.J.l]. Zu erwähnen ist, dass es ein weiteres Objekt namens *triangle~* gibt, welches ebenfalls zur Erzeugung einer Dreieck- sowie aber auch einer Sägezahnwelle verwendet werden kann. Dieses formt lediglich ein Signal des Objekts *phasor~* in ein bipolares Sägezahnsignal um, woraus wiederum ein Dreieck geformt werden kann (Duty-Cycle) [Cycling ’74, o.J.k]. Das entstehende Signal ist allerdings nicht bandbegrenzt und findet zumeist in der Kontrollparametersteuerung Gebrauch. Wie bei *rect~* gibt es auch bei *tri~* einen Eingang zur Bestimmung des Duty-Cycles. Hier wird der Zeitpunkt des

Scheitelpunktes (lokales Maximum) innerhalb der Phase zeitlich nach vorne oder hinten geschoben. Auch hier bedeutet ein Wert von 0,5 eine reguläre Dreieckwelle. Bei einem Wert von 1 wird das Dreieck zu einem Sägezahn (eine aufsteigende Rampe), bei dem Wert 0 wäre es das Inverse (eine abfallende Rampe). In Max/MSP kann nur eine Annäherung an einen Sägezahn (oder davon Inverse) erreicht werden.

noise~ und pink~

Max/MSP bietet nur Generatoren zur Erzeugung von zwei Rauschfarben an. Das Objekt *noise~* produziert ein weißes, also ein pseudozufälliges (erscheint zufällig, kann aber reproduziert werden) Rauschen mit konstanter Energie pro Frequenzbereich, während *pink~* ein pinkes, also pseudozufälliges Rauschen mit konstanter Energie pro Oktave generiert. Bei beiden Signalen bewegen sich die Bewerte in eine Bereich zwischen -1 und 1 [Cycling ’74, o.J.f] [Cycling ’74, o.J.h].

4.2. Verfahren der Klangsynthese und -veränderung

Die Klangsynthese und -veränderung bietet unterschiedliche Ansätze zur Erzeugung neuer Geräusche. In dieser Arbeit werden ausschließlich die grundsätzlichsten und am häufigsten verwendeten Verfahren beleuchtet. Diese sind nicht nur der Komplexität dieser Arbeit entsprechend, sondern auch zumeist Fundament für viele weitere komplexere Syntheseverfahren.

4.2.1. Additive Synthese

Bei diesem Verfahren wird der Klang durch die Addition mehrerer Sinuswellen geformt. Der Sinuswelle des Grundtons werden Obertöne hinzugefügt. Die additive Synthese basiert auf den Erkenntnissen Fouriers, dass jede beliebige periodische Schwingung durch eine Summe von Sinusschwingungen beschrieben werden kann. Die Sinuswellen können sich dabei in Frequenz, Amplitude und Phase unterscheiden. Bei der additiven Synthese wird der Klang also durch das explizite Setzen der Frequenz, Amplitude sowie Phase von einzelnen Sinusschwingungen erzeugt. Die Frequenzen sind in der Regel ganzzahlige Vielfache der Grundfrequenz (Harmonische). Teilweise können den einzelnen Teiltönen eigene Hüllkurven zugewiesen werden [Dickreiter et al., 2023].

Es können einem Grundton durch ganzzahlige Division (1/2, 1/3, 1/4, etc. der Grundfrequenz) auch subharmonische Strukturen hinzugefügt werden. Diese kommen in akustischen Geräuschen der realen Welt nicht vor, sind also unnatürlich. Im Trautonium (Friedrich Trautwein und Oskar Sala) wurde dieses Prinzip das erste Mal in einem musikalischen Instrument umgesetzt.

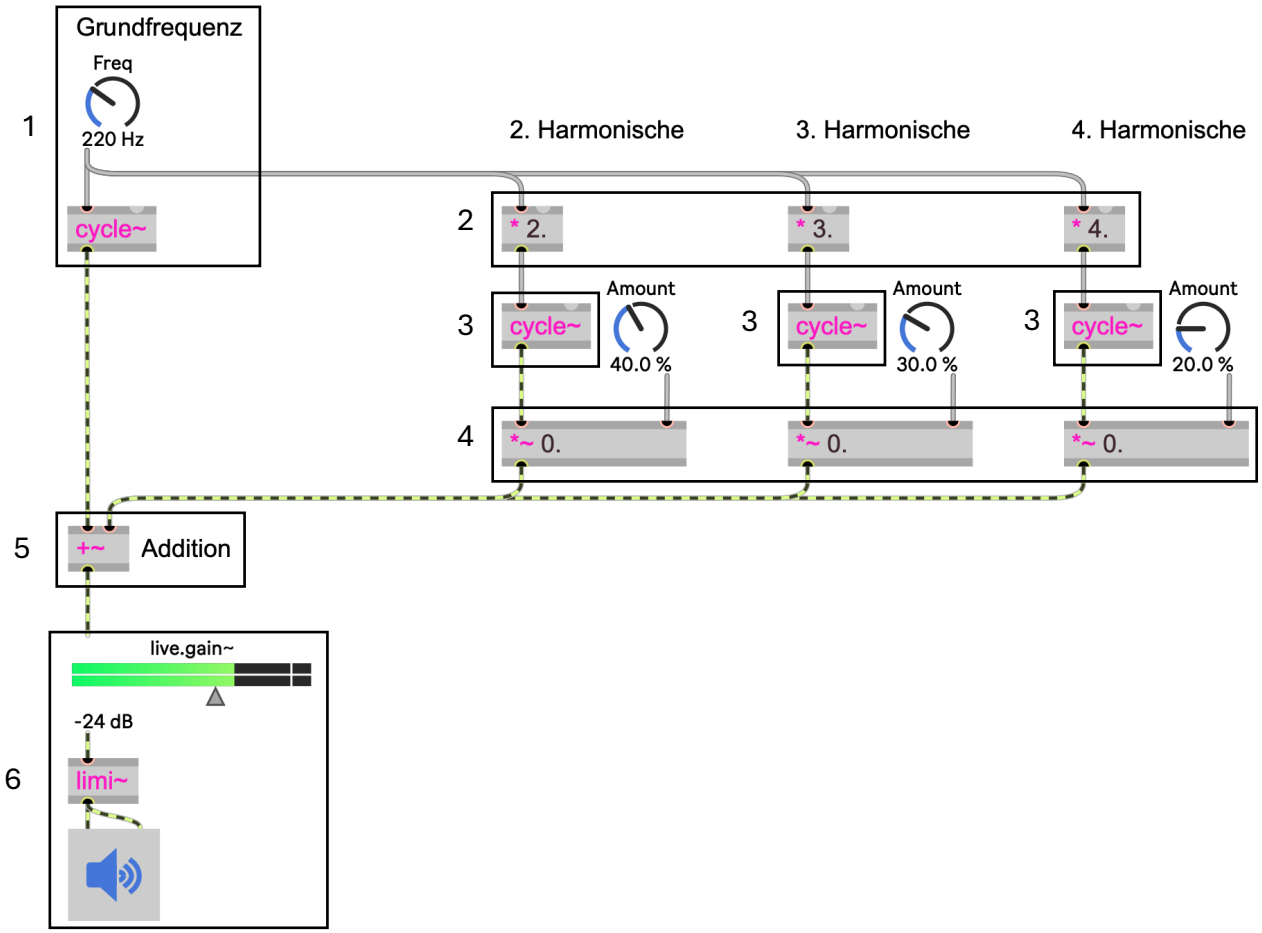
Das erste elektromechanische Musikinstrument, das Telharmonium (Thaddeus Cahill), arbeitete mit additiver Synthese. Zum Start des digitalen Zeitalters erschienen schließlich Instrumente wie das Synclavier (New England Digital Corporation) oder das Fairlight CMI (Fairlight). Die Audio-Plug-Ins (VSTs) des heutigen Musikmarktes bieten zumeist gleichzeitig unterschiedliche Arten der Klangsynthese und -veränderung. Die Prinzipien der additiven Synthese können unter anderem in Serum (Xfer), Operator (Ableton) oder Pigments (Arturia) gefunden werden.

4.2.2. Additive Synthese in Max/MSP

Erläuterungen relevanter Funktionen der angesprochenen Objekte sind unter **9. Legende relevanter Max/MSP-Objekte** *zu finden.*

Eine Möglichkeit der Umsetzung *additiver* Synthese in Max/MSP sieht folgendermaßen aus (Abbildung 4.2):

Abbildung 4.2
Beispielhafter Aufbau eines additiver Syntheseverfahrens in Max/MSP



Für Abbildung 4.2:

1. Erzeugung einer Sinusschwingung mit ausgewählter Frequenz
2. Bestimmung der harmonischen Frequenzen (hier: die ersten 3 ganzzahligen Vielfachen der Grundfrequenz)
3. Erzeugung der harmonischen Sinusschwingungen (Obertöne)
4. Bestimmung der Amplitude der Harmonischen
5. Addition aller Signale
6. Stereo-Signal-Ausgang mit Gain und Limitierung

Die Menge der Harmonischen sowie die Faktoren dieser können beliebig gewählt werden.

4.2.3. Subtraktive Synthese

Die subtraktive Synthese funktioniert quasi genau gegenteilig zum Prinzip der additiven Synthese. Hier wird von einer sehr obertonreichen Quelle ausgegangen, von welcher man gewünschte Frequenzteile abschwächt, verstärkt, entfernt, verändert. Trotz des Namens, ist hier nicht immer die Reduzierung des Signals das Ziel, da bestimmte Frequenzbereiche in der Verarbeitung auch nur angehoben werden können. Ursprungston ist häufig eine Sägezahn-, Dreieck-, Rechteck- oder Pulswelle, es kann aber auch ein Rauschen sein. Danach durchläuft das Signal einer einfachen oder mehrfachen Filterung. Häufig gibt es mehrere Signalquellen (Oszillatoren) die alle eine eigene Hüllkurve durchlaufen. Die Filterung ist üblicherweise ebenfalls zeitlich steuerbar und kann von Modulatoren sowie eigenen Hüllkurven abhängig sein [Dickreiter et al., 2023].

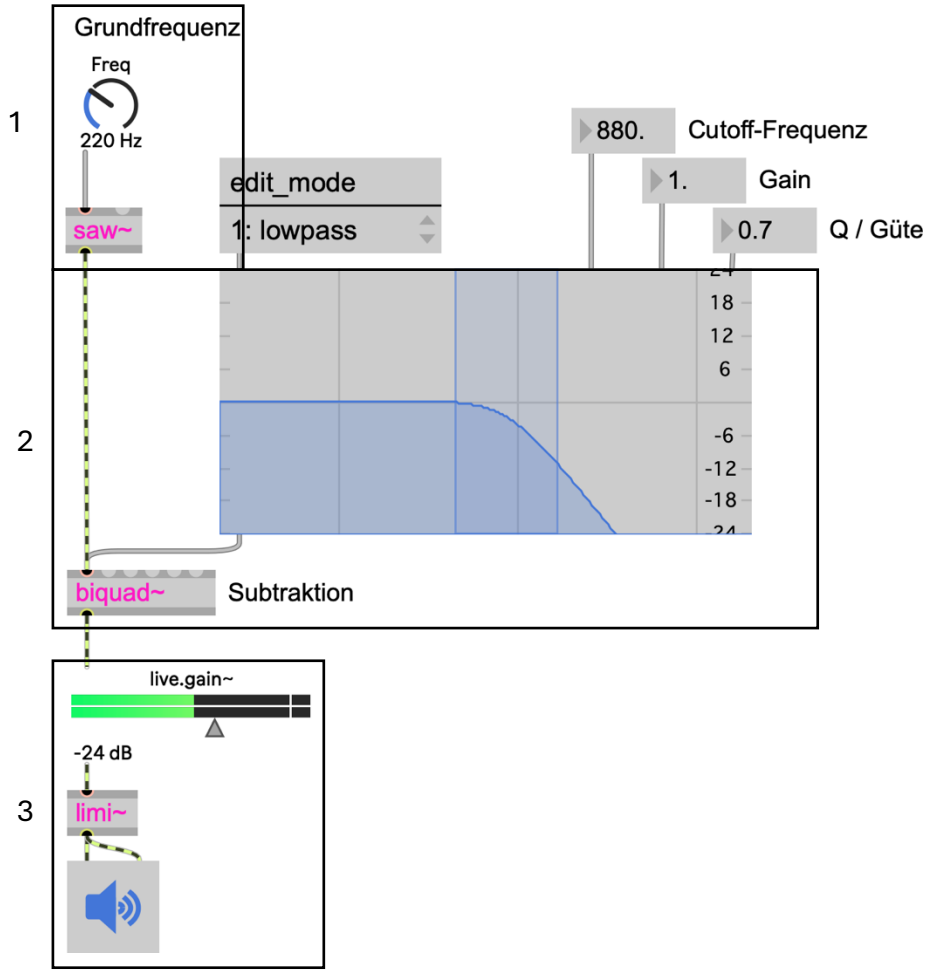
Als einer der ersten prominenten Vertreter mit subtraktiver Synthese zählt das Novachord (Hammond), welches vor allem in der Filmmusik der 1940er Nutzen fand. In der zweiten Hälfte des 20. Jahrhunderts erschienen dann Instrumente wie der Moog Synthesizer und der Minimoog (beide Moog Music) oder der MS-20 (Korg). Der OSCar (Oxford Synthesiser Company) bot in den 1980ern neben subtraktiver gleichzeitig auch additive Synthese an. Bei modernen VSTs zählt auch, dass die meisten Produkte mehrere Klangsyntheseverfahren und -veränderungsverfahren anbieten. Serum (Xfer), Operator (Ableton) und Pigments (Arturia) können alle ebenfalls subtraktive verwendet werden.

4.2.4. Subtraktive Synthese in Max/MSP

Erläuterungen relevanter Funktionen der angesprochenen Objekte sind unter **9. Legende relevanter Max/MSP-Objekte** *zu finden.*

Eine Möglichkeit der Umsetzung subtraktiver Synthese in Max/MSP sieht folgendermaßen aus (Abbildung 4.3):

Abbildung 4.3
Beispielhafter Aufbau eines subtraktiven Syntheseverfahrens in Max/MSP



Für Abbildung 4.3:

1. Erzeugung einer Sägezahnschwingung mit ausgewählter Frequenz
2. Filterung des frequenzreichen Signals (hier: Tiefpass; *biquad~* ist ein Filter 2ter-Ordnung, ergo Flankensteilheit von 40 dB/Dekade)
3. Stereo-Signal-Ausgang mit Gain und Limitierung

Die Menge und der Typ der Signalquellen (frequenzreiche Typen wie ein Rechteck oder Rauschen) sowie die Menge und die Typen der Filter können beliebig gewählt werden.

4.2.5. Amplituden- und Ringmodulation

Amplituden- (AM) und Ringmodulation (RM) folgen demselben Konzept. Das Eingangssignal (Träger) wird mit einem eingespeisten zweiten Signal (Modulator) multipliziert. Die zwei Verfahren müssen in der Polarität der Modulatoren unterschieden

werden. Während beide Signale der RM bipolar (über Zeit die Polarität wechselnd) sind, ist der Modulator bei der AM unipolar (über Zeit eine Polarität haltend). Dadurch können unterschiedliche Ergebnisse erzielt werden, welche sich in der resultierenden Wellenform sowie im Frequenzspektrum erkennen lassen. Zur einfacheren Verständlichkeit wird von Sinuswellen bei Träger sowie Modulator ausgegangen. Bei der RM nimmt die daraus entstehende Wellenform pro Periode des Modulators zweimal den Wert null an, während bei der AM das Ergebnis nie den Wert null annimmt. Bei beiden Modulationen entstehen im Frequenzspektrum zwei Seitenbänder, welche aus der Summe und Differenz der Frequenzen des Trägers und des Modulators herzuleiten sind. Bei der AM bleibt zusätzlich die Frequenz des Trägers erhalten. Bei Frequenzen bis etwa 20 Hz des Modulators sind die Seitenbänder nicht hörbar und es entsteht ein Tremolo-Effekt. Werden diese etwa 20 Hz allerdings überschritten wird auch die Veränderung des Klang in der Frequenzdomäne wahrgenommen [Dickreiter et al., 2023].

4.2.6. Amplituden- und Ringmodulation in Max/MSP

Erläuterungen relevanter Funktionen der angesprochenen Objekte sind unter **9. Legende relevanter Max/MSP-Objekte** *zu finden.*

Bei der Ringmodulation werden zwei bipolare Signale multipliziert. Bei der Amplitudenmodulation ist der Modulator ein unipolares Signal, weshalb dieser vor der Multiplikation durch zwei arithmetische Operationen umgeformt werden muss. In diesen Fällen beträgt die Frequenz der Träger jeweils 220 Hz und die der Modulatoren jeweils 2 Hz. Durch die geringe Frequenz (< 10 Hz) der Modulatoren entsteht ein Tremolo-Effekt (periodische Lautstärkenschwankung). Bei gleicher Frequenz der Modulatoren der RM und AM ist das hörbare Tremolo bei der AM nur halb so schnell. In den resultierenden Wellenformen kann außerdem erkannt werden, dass diese bei der RM (Abbildung 4.4) durch 0 geht, während bei der AM (Abbildung 4.5) diese nie 0 wird.

Abbildung 4.4
Resultierende Wellenform einer Ringmodulation

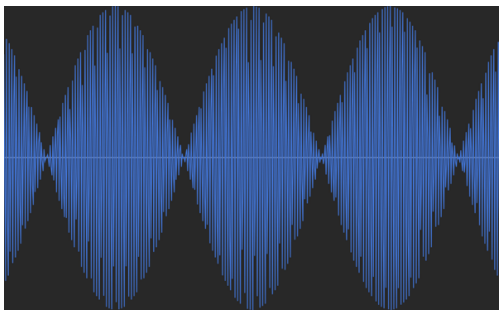
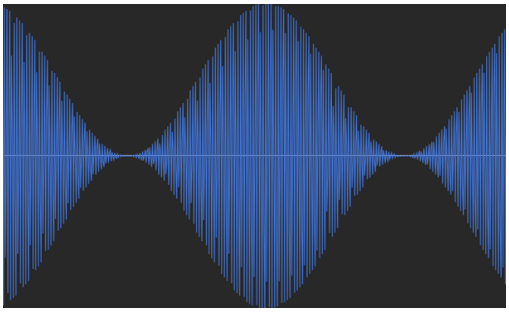
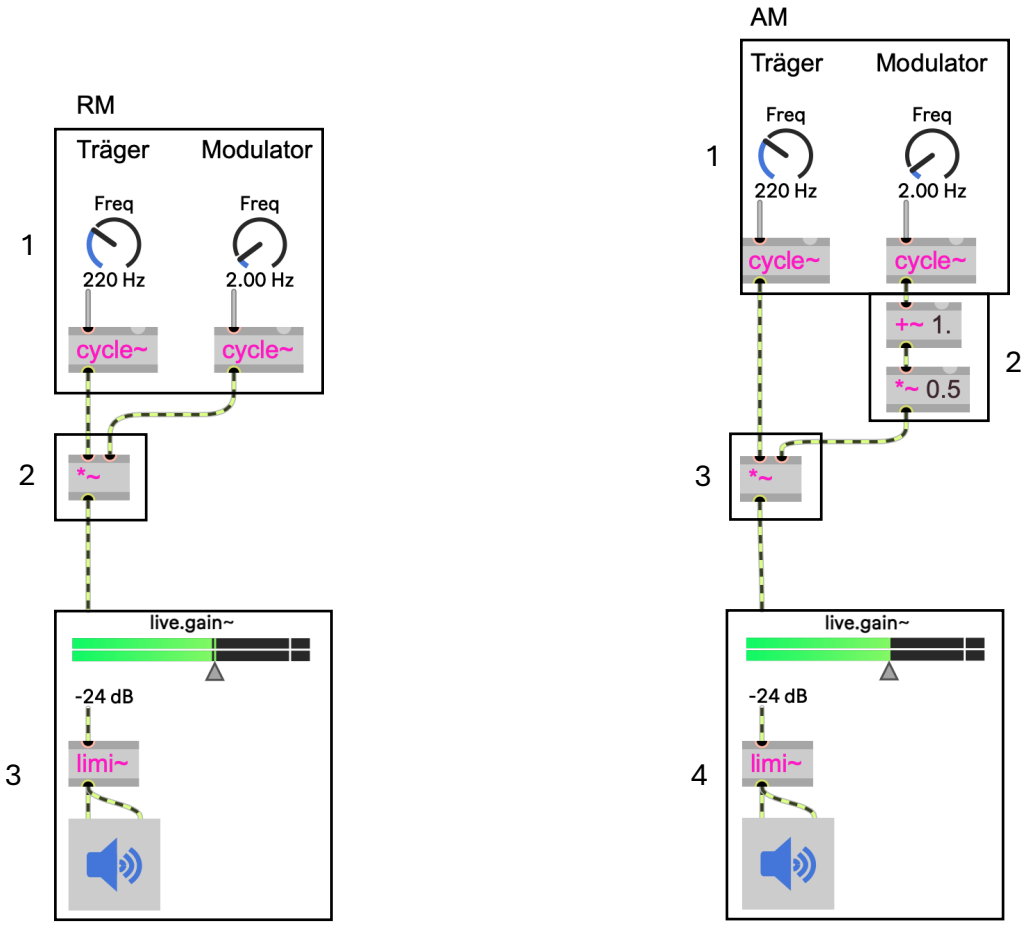


Abbildung 4.5
Resultierende Wellenform einer Amplitudenmodulation



Mögliche Arten der Umsetzung (Abbildung 4.6):

Abbildung 4.6
Beispielhafter Aufbau einer Ring- (links) sowie Amplitudenmodulation (rechts) in Max/MSP



Für Abbildung 4.6 (links):

1. Erzeugung des Träger- und Modulatorsignals (hier sinusoidale) mit ausgewählter Frequenz
2. Multiplikation beider Signale
3. Stereo-Signal-Ausgang mit Gain und Limitierung

Für Abbildung 4.6 (rechts):

1. Erzeugung des Träger- und Modulatorsignals (hier sinusoidale) mit ausgewählter Frequenz
2. Umformung des bipolaren (-1 bis 1) Modulatorsignals in ein unipolares (0 bis 1)
3. Multiplikation beider Signale

4. Stereo-Signal-Ausgang mit Gain und Limitierung

Die Träger- und Modulatorensignale können jegliche Wellenform annehmen. In dieser Arbeit werden zur einfacheren Anschaulichkeit der Resultate sinusoidale Wellen verwendet, da diese Energie an nur einer Frequenz tragen.

Die Abbildungen 4.7 und 4.8 zeigen das Frequenzspektrum (lineare Darstellung) der Trägersignale (grau) und der Ausgangssignale (blau) der RM und AM aus den Abbildungen 4.4 und 4.5 (mit jeweils 10 kHz f_T und 1 kHz f_M). In den blauen Spektren können die entstandenen Seitenbänder erkannt werden. Das Ausgangssignal der AM behält zusätzlich noch das Trägersignal.

Abbildung 4.7
Resultierendes Frequensspektrum einer Ringmodulation

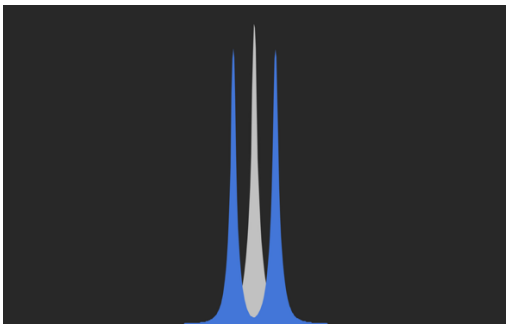
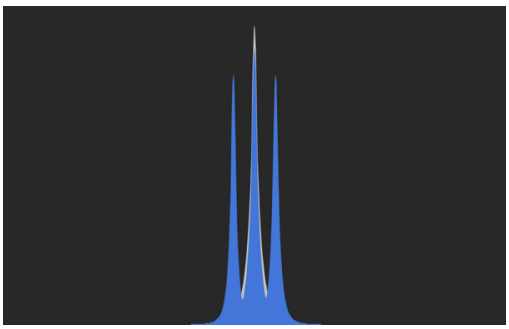


Abbildung 4.8
Resultierendes Frequensspektrum einer Amplitudenmodulation



4.2.7. Frequenz- und Phasenmodulation

Bei der Frequenz- (FM) und Phasenmodulation (PM) gibt es ebenfalls zwei Signale (Träger und Modulator), wobei diese Signale hier nicht einfach miteinander multipliziert werden. Während bei der FM die Frequenz des Trägers von der Schwingung des Modulators

beeinflusst wird, verschiebt bei der PM der Modulator die Phase des Trägers. Beide Modulationen sorgen für ähnliche klangliche Resultate und vor allem die Bezeichnung FM wird oft für beide Verfahren angewendet, daher werden sie hier zusammen aufgeführt. In der analogen Welt gilt FM als leicht umsetzbar, während PM nur schwer zu realisieren ist. Als digitales Verfahren ist dies jedoch genau umgekehrt. FM ist im Vergleich zu PM äußert rechenintensiv. In den ersten kommerziell verfügbaren digitalen Synthesizern (z.B. Yamaha DX-7) wurde deshalb häufig PM verwendet. Dies ist bis heute der Fall, möglicherweise auch, da der leicht abweichende Klang und das etwas andere Verhalten der PM erwartet wird [Dickreiter et al., 2023] [cooksonia, 2024].

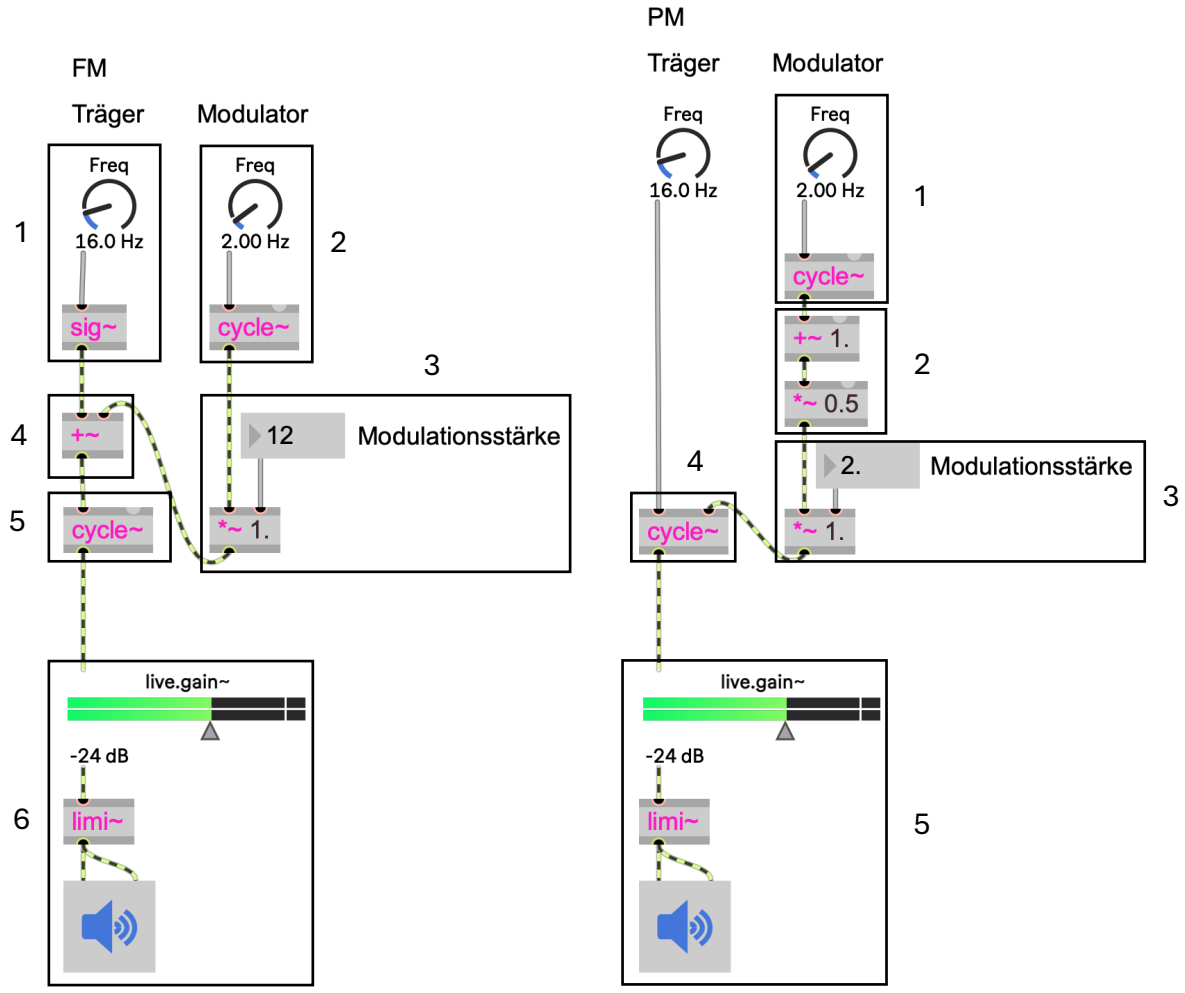
Für FM sowie PM gilt, dass niedrige Frequenzen des Modulators (etwa < 20 Hz) für einen Vibrato-Effekt sorgen. Wird dieser Schwellenwert jedoch überschritten entstehen zusätzliche hörbare komplexe Schwingungen (Seitenbänder: f_{SB}; theoretisch unendlich viele). Diese bilden sich bei den ganzzahligen Vielfachen der Differenz sowie Summe der Träger- (f_T) und Modulatorfrequenzen (f_M): $f_{SB_n} = f_T \pm nf_M$. Die Stärke der Amplituden lassen sich mit Hilfe von Bessel-Funktionen erster Ordnung. Da der Modulationsindex bei FM und PM unterschiedliche bestimmt wird, unterscheiden sich die Amplitudenwerte der Seitenbänder. Bei bestimmten Verhältnissen von f_T zu f_M können einzelne Seitenbänder auch die Amplitude 0 annehmen [Dickreiter et al., 2023].

4.2.8. Frequenz- und Phasenmodulation in Max/MSP

Erläuterungen relevanter Funktionen der angesprochenen Objekte sind unter **9. Legende relevanter Max/MSP-Objekte** *zu finden.*

Die Umsetzung von FM und PM kann in Max/MSP folgendermaßen aussehen (Abbildung 4.9):

Abbildung 4.9
Beispielhafter Aufbau einer Frequenz- (links) sowie Phasenmodulation (rechts) in Max/MSP



Für Abbildung 4.9 (links)**:**

1. Umwandlung des Frequenz-Wertes des Trägers in ein MSP-Signal
2. Erzeugung des Modulatorsignals mit ausgewählter Frequenz
3. Bestimmung der Amplitude des Modulatorsignals (Modulationsstärke)
4. Addition des Frequenz-Signals des Trägers mit dem Modulatorsignal

5. Erzeugung des resultierenden Signals mit Frequenz durch Summe (4) bestimmt
6. Stereo-Signal-Ausgang mit Gain und Limitierung

Für Abbildung 4.9 (rechts)**:**

1. Erzeugung des Modulatorsignals mit ausgewählter Frequenz
2. Umformung des bipolaren (-1 bis 1) Modulatorsignals in ein unipolares (0 bis 1)
3. Bestimmung der Amplitude des Modulatorsignals (Modulationsstärke)
4. Erzeugung des resultierenden Signals mit Phase durch Modulatorsignal bestimmt
5. Stereo-Signal-Ausgang mit Gain und Limitierung

Bei Anwendung gegebener Parameter (Abbildung 4.9) können durch FM und PM fast identische Wellenformen erreicht werden (Abbildung 4.10 und 4.11). Einzig der Wert der Modulationsstärke weicht ab, da dieser bei der FM und PM unterschiedliche Faktoren beeinflusst.

Abbildung 4.10
Resultierende Wellenform einer Frequenzmodulation

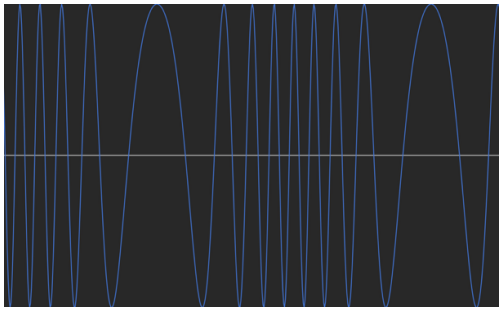
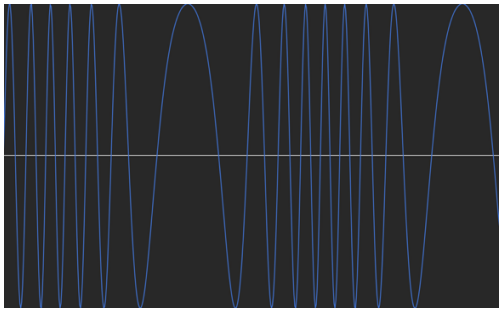


Abbildung 4.11
Resultierende Wellenform einer Phasenmodulation



Die Frequenzspektren (Abbildung 4.12 und 4.13) beider Modulationsarten mit denselben Signalfrequenzen ($f_T = 10\,kHz,\ f_M = 1\,kHz$)[1] und Modulationsstärken von 1000 (FM) und 0,25 (PM) unterscheiden sich nur in den Amplituden der Seitenbänder (hinterlegtes graues Spektrum = Trägersignals).

Abbildung 4.12
Resultierendes Frequenzspektrum einer Frequenzmodulation

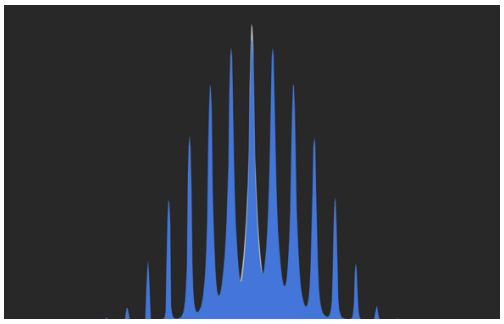
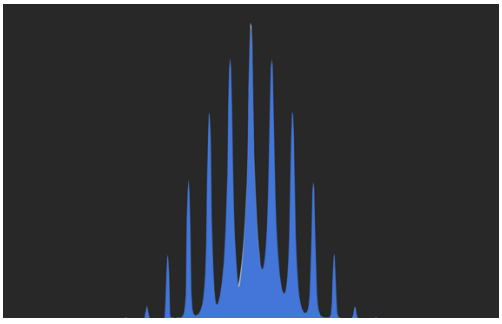


Abbildung 4.13
Resultierendes Frequenzspektrum einer Phasenmodulation

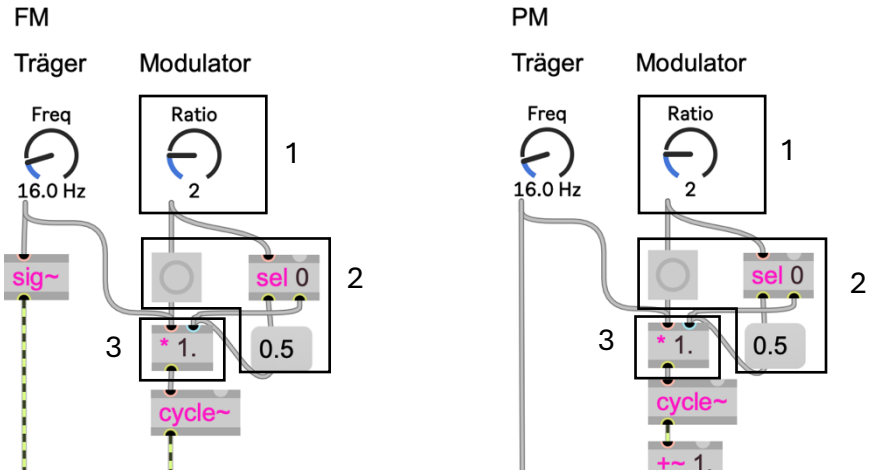


Häufig wird die Frequenz des Modulators durch ein Verhältnis (Ratio) zur Frequenz des Trägers bestimmt. In der Regel können ganzzahlige Vielfache der Trägerfrequenz für die Modulatorfrequenz gewählt werden.

[1] f_T: Trägerfrequenz; f_M: Modulatorfrequenz

Folgend ist eine mögliche Umsetzung in Max/MSP (Abbildung 4.14):

Abbildung 4.14
Umsetzung eines Ratio-Parameters für FM und PM Module



Für Abbildung 4.14 (links und rechts identisch):

1. Bestimmung der Modulatorfrequenz durch Verhältnisse von {0,5}U[1, 10] (der Ratio-Wähler gibt 0 anstatt 0,5 aus, daher Schritt 2)
2. Wandlung des Wertes 0 in 0,5 und Ausgabe einer bang-Nachricht bei Änderung der Ratio
3. Bestimmung der Modulatorfrequenz durch Multiplikation der Trägerfrequenz mit gewählter Ratio

4.3. Hüllkurven und LFOs

4.3.1. Hüllkurven

Klänge in der Synthese werden häufig zusätzlich auf Makroebene geformt. Klassischerweise geschieht das einerseits durch sogenannte Hüllkurven (engl. Envelopes). Andererseits durch LFOs (siehe 4.3.3.). Diese manipulieren in der Regel die Lautstärke und Tonhöhe des Klangs. Bei Betrachten des Zyklus einer Wellenform im Bereich einzelner Samples können ebenfalls schon Veränderungen in Amplitude sowie Frequenz erkannt werden. Dagegen formt eine Hüllkurve den Klang zumeist in deutlich größeren Zeitspannen, welche sich im Bereich weniger Millisekunden bis hin zu mehrerer

Sekunden befinden können. Das Formen durch Hüllkurven ist den Klangbildern akustischer Instrumente nachempfunden [Dickreiter et al., 2023].

Häufig werden die Steuerelemente zur Bedienung einer Hüllkurve mit ADSR beschriftet. Das Akronym steht für die vier Bestandteile einer Hüllkurve:

- **Attack** (Anstieg): die Zeit, in der die Amplitude von 0 auf einen Wert ansteigt
- **Decay** (Abklingen): die Zeit, in der die Amplitude von dem Attack-Endwert auf einen nächsten Wert weiter aufsteigt oder abfällt
- **Sustain** (Halten): die Zeit, in der die Amplitude auf dem Decay-Endwert gehalten wird
- **Release** (Ausklingen): die Zeit, in der die Amplitude von dem Sustain-Endwert auf 0 fällt

Diese vier Bestandteile treten nacheinander in aufgelisteter Reihenfolge auf. Klassischerweise ist der Endwert des Attacks auch die Maximalstelle einer gesamten Hüllkurve und die Decay-Phase sorgt für keine weiteren Anstieg. Dies ist allerdings vor allem bei modernen Software-Synthesizern mehr notwendigerweise der Fall.

Bei Hardware-Synthesizern kann normalerweise mindestens eine Lautstärken-Hüllkurve gefunden werden. Häufig wird allerdings auch eine Hüllkurve mit denselben vier Faktoren zur Formung eines Filters angeboten. Beispielswiese kann die Grenzfrequenz eines Tiefpassfilters dadurch über Zeit ansteigen und wieder abfallen. Gegebenenfalls gibt es auch einen Schalter, um die Polarität der Hüllkurve zu wechseln, wodurch z.B. der Attack-Wert einen Abfall beschreiben kann.

Viele Software-Synthesizer (ausgenommen Emulatoren von Hardware-Geräten) sind heutzutage oft modular gestaltet. Dadurch können Hüllkurven beliebige Parameter steuern, beispielswiese auch die Werte (A, D, S oder R) anderer Hüllkurven. Es entsteht ein deutlich größerer Pool an Klanggestaltungsmöglichkeiten.

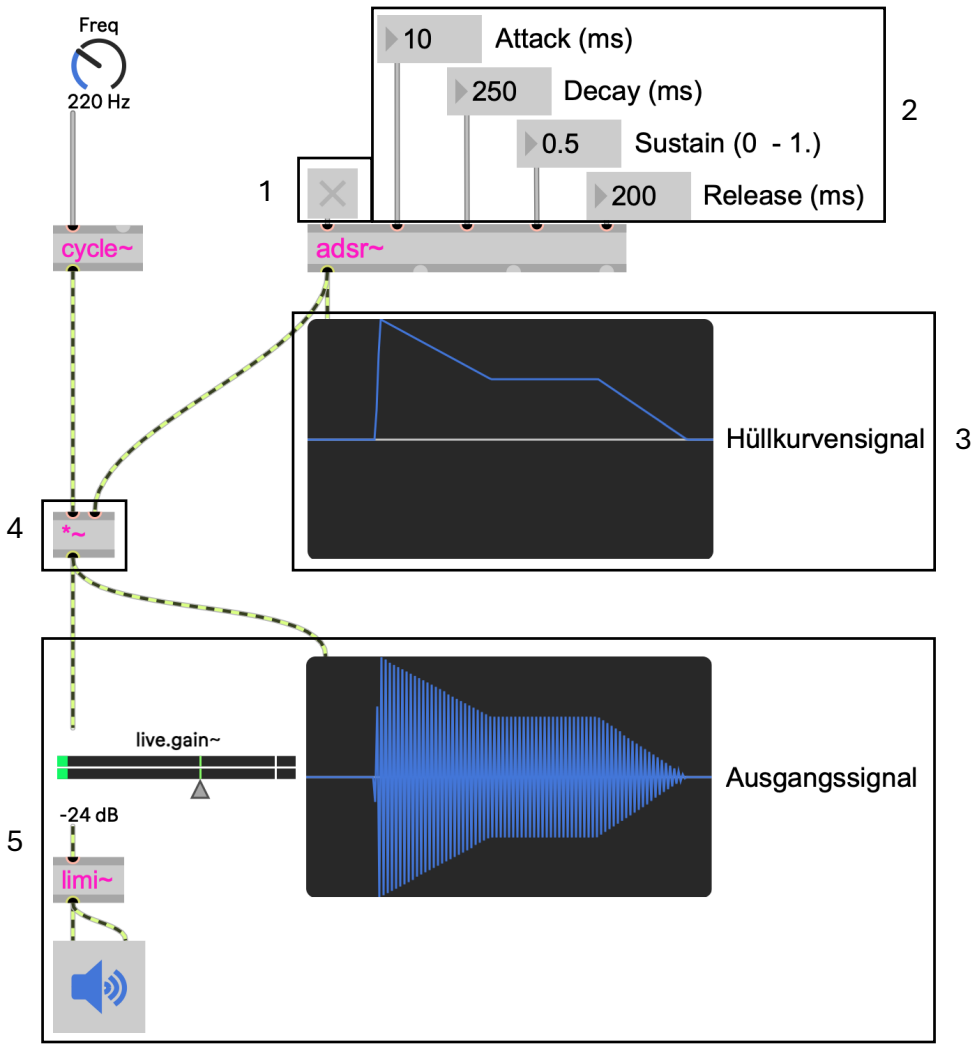
Ganz generell formuliert ist eine Hüllkurve eine Linie, welche die einzelnen lokalen Maxima einer Welle beschreibt. Diesem Prinzip nachgehend können auch Hüllkurven für akustische Instrumente erstellt werden [Cipriani & Giri, 2019].

4.3.2. Hüllkurven in Max/MSP

Erläuterungen relevanter Funktionen der angesprochenen Objekte sind unter **9. Legende relevanter Max/MSP-Objekte** *zu finden.*

Zur Erzeugung einer Hüllkurve in Max/MSP kann das Objekt *adsr~* verwendet werden. Über die Eingänge 2 bis 5 können die Parameter Attack (Eingang 2), Decay (Eingang 3), Sustain (Eingang 4) und Release (Eingang 5) bestimmt werden. Durch Eingabe eines Wertes ungleich 0 an Eingang 1 wird das Objekt aktiviert und es wird ein MSP-Signal erzeugt, welches den angegebenen Parametern folgt und auf dem Sustain-Wert gehalten wird. Erst bei Eingabe des Wertes 0 an Eingang 1 wird die Release-Phase generiert. Dadurch kann das Objekt mit MIDI-Daten gesteuert werden, da Note-On (bei MIDI-Keyboard: Halten einer Taste) eine Zahl ungleich 0 und Note-Off (bei MIDI-Keyboard: Loslassen / Nicht-drücken einer Taste) 0 ist.

Abbildung 4.15
Implementierung eines MSP-Objekts zur Erzeugung einer Hüllkurve



Für Abbildung 4.15**:**

1. Aktivierung / Deaktivierung bzw. Start / Stopp von *adsr~*
2. Einstellung der Hüllkurven-Parameter
3. Resultierendes Hüllkurvensignal
4. Multiplikation des Eingangssignals mit dem Hüllkurvensignal
5. Stereo-Signal-Ausgang mit Gain, Limitierung und Darstellung des Ausgangsignals

Mit dem Objekt *line~* in Kombination mit *function* oder dem Objekt *live.adsr~* in Kombination mit *live.adsrui* kann ein Hüllkurve auch „gezeichnet“ werden. Resultierend sind hier ebenfalls MSP-Signale, die mit den zu formenden Signalen multipliziert werden.

4.3.3. LFOs

Ein LFO (Low Frequency Oscillator) generiert im Prinzip eine Welle wie jeder andere Oszillator, jedoch befindet sich die Frequenz in der Regel in Bereichen unter der Hörschwelle des Menschen (< 20 Hz) und ist dadurch nicht hörbar. Dieses Signal findet ähnliche Anwendungsbereiche wie auch eine Hüllkurve, z.B. zur Modulation der Lautstärke oder der Frequenz, kann aber auf jeden Parameter angewendet werden. Bei Anwendung auf die Lautstärke wird hierbei von einem Tremolo und bei Anwendung auf die Frequenz von einem Vibrato gesprochen. Tremolo ist als niederfrequente Ring- und Vibrato als niederfrequente Frequenzmodulation einzuordnen [Dickreiter et al., 2023].

Unter den Wellenformen eines LFOs können alle klassischen gefunden werden. Am üblichsten sind allerdings sinusoidale Wellen, aber auch Rechteck- sowie Dreieckwellen finden häufig Anwendung.

Bei (vor allem älteren) Hardware-Synthesizern kann es auch sein, dass kein expliziter LFO verfügbar ist. Hier kann dieser auch unter anderen Namen auftauchen wie z.B. als Sub Oscillator (Yamaha CS-80) oder er ist z.B. mit der Lautstärke festverdrahtet und versteckt sich hinter der Bezeichnung Tremolo. Ähnlich wie bei Hüllkurven sind heutzutage LFOs bei Software-Synthesizern häufig modular gestallt und können beliebige Parameter modulieren.

5. Zeit in Max/MSP

Um mit der Zeitebene in Max/MSP umgehen zu können, werden primär zwei Objekte benötigt: *transport* und *phasor~*. Zusätzlich ist der Zeitangaben-Syntax wichtig zu verstehen, da neben *transport* und *phasor~* auch viele andere Objekte (z.B. *metro*: gibt eine bang-Nachricht in vorgegebenen Zeitintervallen aus) Zeitangaben in dieser Form annehmen können.

5.1. Time-Value-Syntax in Max/MSP

In Max gibt es gibt es zwei Kategorien von Zeitangaben. Zum einen gibt es fixe, zum anderen Tempo-abhängige Zeitangaben.

Fixe Werte werden in Millisekunden, in Stunden/Minuten/Sekunden-Formaten (00:00:00.000, die drei Ziffern am Ende für die Angabe von Millisekunden ist optional; 3-4 Zahlen gefolgt von der Formatangabe hh:mm:ss, die Millisekunden werden durch die vierte Zahl repräsentiert und sind auch hier optional), in Samples (dadurch abhängig von der Samplerate) oder in Hertz angegeben. Millisekunden ist in der Regel die Standardeinheit.

Tempo-abhängige Angaben basieren immer auf sogenannten Ticks. In einer Viertelnote sind 480 Ticks. Diese können in Form einer Zahl als Ticks, eines Notenwerts (ganze Zahl mit Exponent 2, d.h. 1, 2, 4, 8, ... bis 128 entsprechen ganzen Note, halben Note, Viertelnote, Achtelnote, ...) mit Angabe von Punktierung oder Triole (z.B. 4n = Viertelnote, 4nd = punktierte Viertelnote, 4nt = Vierteltriole; bei 64 gibt es nur n und nd, bei 128 nur n) oder Bars/Beats/Units (z.B. 6.3.120 oder 6 3 120 bbu; Units = Ticks) angegeben werden. Angaben in Beats/Bars/Unit-Format können auf zwei unterschiedliche Weise interpretiert werden. Entweder als Position oder als Intervall. Z.B. sendet das Objekt *timepoint* mit dem Argument 6 3 120 bbu eine bang-Nachricht bei Erreichen von 6 3 120 bbu. Das Objekt *metro* mit demselben Argument würde hingegen nach Ablauf eines Intervalls mit der Länge dieser Zeitangabe eine bang-Nachricht abschicken.

5.2. transport

Mit dem Objekt *transport* kann ein getaktetes (also Tempo-abhängiges) Signal generiert werden, an welche sich andere Objekte knüpfen können. Das erfolgt mit der Bestimmung des Taktes in Beats und Units (Units bestimmt den Notenwert) und des Tempos in BPM (Beats per Minute). Gleichzeitig läuft ein Zähler, der bei Aktivierung des Objekts die Bars, Beats und Ticks mitzählt. Das Objekt kann den globalen Transport von Max bestimmen (welcher durch ein Icon unten rechts auch gestartet werden kann), bei Angabe eines Namens allerdings auch als eigene Instanz separat laufen. Durch Doppelklicken des

Objekts öffnet sich ein eigenes Fenster mit den Angaben des Transport-Objekts. Das Fenster für den globalen Transport lässt sich auch über die Menüleiste von Max öffnen.

5.3. phasor~

Wie schon angesprochen produziert das Objekt *phasor~* eine lineare sich wiederholende Rampe von 0 bis (fast) 1, also einen unipolaren Sägezahn. Bei Erreichen des Maximums springt der Signal innerhalb eines Samples wieder auf 0. Die Länge einer Wiederholung kann in Hertz oder einem Tempo-abhängigem Zeitformat angegeben werden. Bei letzterem wird neben der Bestimmung des *transport*-Objekts, mit welchem der *phasor~* synchronisiert werden soll, auch ein Notenwert angegeben. Bei 1n braucht die Rampe so lange wie eine Bar des *transport*-Objekts, um von 0 auf 1 zu kommen. Bei 2n halbiert würde sich die Zeit halbieren, bei 4n vierteln. Durch diese Attribute wird *phasor~* häufig als Zeitquellen (vor allem in Abhängigkeit eines *transport*-Objekts) verwendet.

5.3.1. Vorteile gegenüber anderer Zeitquellen

In Max/MSP können neben *phasor~* auch andere Objekte als Zeitquelle genutzt werden. Ein übliches ist *metro* (s.o.: gibt eine bang-Nachricht in vorgegebenen Zeitintervalle aus), welches in Control-Rate läuft. Das heißt es kann nur Signale mit einer Genauigkeit von 1 ms erzeugen, im Vergleich zu Objekten des Signal-Networks, welche samplegenaue Signale erzeugen können. Durch *click~* kann eine *metro*-Uhr in Impulse (jede bang-Nachricht von *metro* löst einen Impuls durch *click~* aus) im Signal-Network gewandelt werden, behält dadurch allerdings die Ungenauigkeiten.

Bei modularen Synthesizern wird häufig ein Pulssignal als Taktgeber verwendet. Das ist auch in Max/MSP durch das Objekt *train~* möglich, welchem in Millisekunden die Intervalllänge mitgegeben werden kann. Dieses Objekt kann allerdings nicht *transport* synchronisiert werden (zumindest nicht ohne extra Objekte).

Die Form einer Rampe, bietet außerdem noch zusätzliche Vorteile:

- Besonders in Elektronischer Musik ist es häufig hilfreich zyklisch (in Loops) und nicht linear zu denken. *Phasor~* bietet das, da die Rampe zyklisch von 0 bis 1 sich

bewegt, wodurch außerdem die Position innerhalb dieses Zyklus festgestellt werden kann (bei Impulsen oder Pulssignal ist das nicht möglich).

- Max/MSP bietet die Möglichkeit mit jeweils einem weiteren Objekt das *phasor~*-Signal zu unterteilen (*subdiv~*) oder zu verlängern (*rate~*), wodurch ein Signal mit einem Vielfachen oder Teiler der Länge des *phasor~*-Signals erzeugt werden kann.

Mit *what~* kann ein *phasor~*-Signal in Impulse verwandelt werden (*what~* generiert ein Impuls bei Überschreiten eines spezifizierten Schwellenwertes; bei keiner Angabe ist das 0). Mit dem Objekt >~ kann ein aus einem *phasor~*-Signal ein Pulssignal generiert werden. Wird >~ der Wert 0.5 mitgegeben, handelt es sich um ein symmetrisches Pulssignal.

6. Praktische Umsetzung

6.1. Zielsetzung

Das Ziel des Projekts war die erfolgreiche Umsetzung einiger üblicher Klangsynthese und -veränderungsverfahren zur Erzeugung häufig genutzter Instrumente der elektronischen Musik und die Implementierung dieser in ein Werkzeug zur Erschaffung anpassbarer Loops. Zum einen sollte das Projekte innerhalb der Entwicklungsumgebung Max/MSP umgesetzt werden, da durch die visuelle Komponente der Programmiersprache die theoretischen Grundlagen einfach deutlich demonstriert werden können und zum anderen, weil Max/MSP das wahrscheinlich am weitesten verbreitetste Tool dieser Art ist.

Durch die breite Vielfalt an Optionen veränderte sich das Max/MSP-Patch während der Entwicklungsphase andauernd und viele bereits umgesetzten Implementationen mussten wieder entfernt werden.

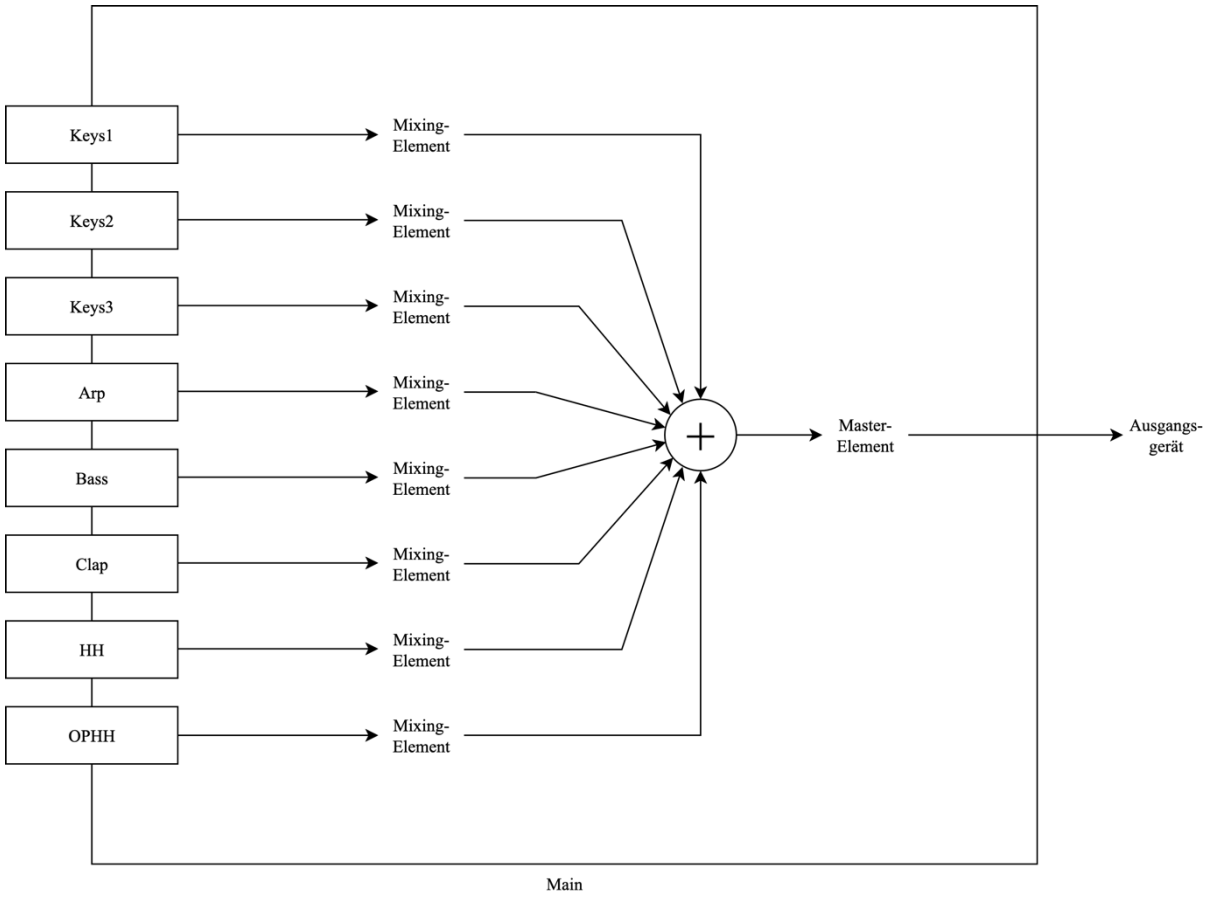
6.2. Aufbau des Patches

Das Projekt ist verschachtelt und besteht aus einem Hauptpatch (*BA_Main*) sowie weiteren Subpatches. In dem *BA_Main* befinden sich Elemente zur Bestimmung (BPM,

Looplänge) und Ausgabe (wo sich das Projekt innerhalb des Loops befindet) der Zeitdimension sowie bestimmter synchronisierte rhythmische Steuersignale (bang-Nachrichten in spezifischen Notenlängen).

Die Klangerzeugung erfolgt in Subpatches. Zu unterteilen sind diese in Keys (Abk.: k), Arp (Abk.: a), Bass (Abk.: b), Clap (Abk.: c), Hi-Hat (Abk.: hh) und Open-Hi-Hat (Abk.: ophh). Da innerhalb des Projekts viele Subpatches existieren werden diese Teilelemente fortan als Instrumente bezeichnet. Die Ausgänge der Instrumente führen in dem Hauptpatch in einzelne Mixing-Elemente, in welchen die Lautstärke und das Panning bestimmt wird und Effekte hinzugefügt werden können. Nach diesen Schritten führen alle Signale der Mixing-Elemente in ein Mastering-Element, welches wie alle Mixing-Elemente aufgebaut ist. Der Ausgang dieses Elements führt in den Max-Ausgang und wird ausgespielt.

Abbildung 6.1
Flußdiagramm zur Darstellung der Patchstruktur des Projekts



In dem Hauptpatch wird außerdem die Auslastung der CPU angezeigt und die Zwischenstände Parameter des Hauptpatches können als Presets gespeichert werden.

6.2.1. Keys

Die Keys-Instrumente dienen der Harmoniestruktur. Es gibt drei Instanzen die leicht unterschiedliche Noten spielen, basieren alle aber auf demselben Synthese-Verfahren und sind sehr ähnlich aufgebaut. Sie verwenden PM-Synthese mit zwei identischen Sinus-Oszillatoren, einem Träger und einem Modulator (Abbildung 6.2). Die Generierung der Oszillatoren-Signale erfolgt in einem Subpatch innerhalb des Instruments (*pm_engine*).

Die Oszillatoren haben jeweils einen Parameter zu Bestimmung der Ratio und einen zur Bestimmung der Amplitude. Die Ratio bestimmt das Verhältnis der Frequenz des Oszillators mit der eingehenden Frequenz. Bsp.: Bei einer eingehenden Frequenz von 220 Hz und einer Ratio von 2 wird dem Sinus-Generator (*cycle~*) die Frequenz 440 Hz (Verhältnis 2:1) mitgegeben. Die Amplitude wird in dBFS angegeben.

Abbildung 6.2
Erzeugung der Ratio- und Level-Parameter für die PM in Max/MSP

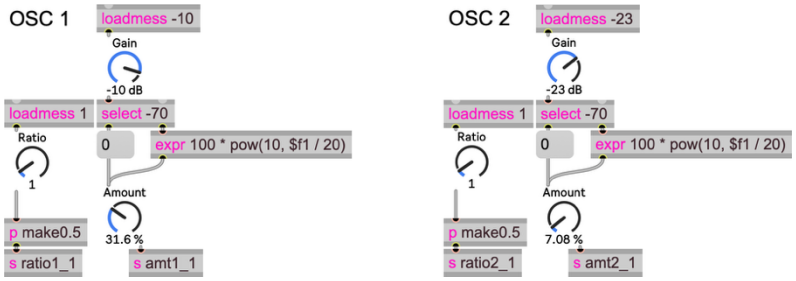
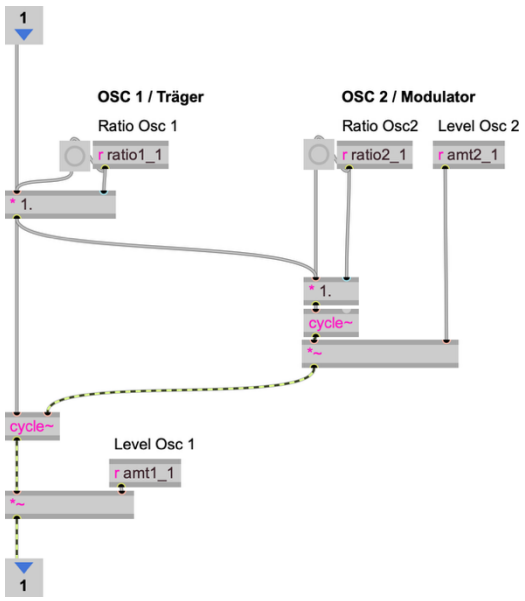
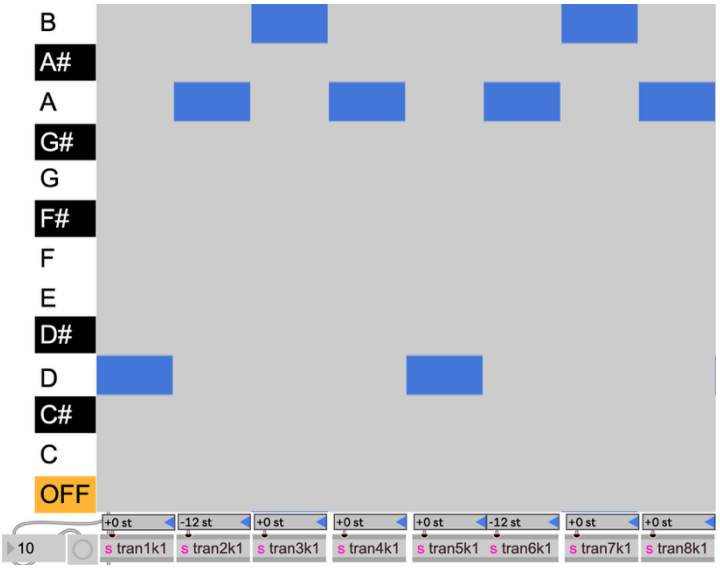


Abbildung 6.3
Umsetzung der PM-Struktur als modulares Subpatcht in Max/MSP



Die Bestimmung der eingehenden Frequenz (vor Ratio) erfolgt durch einen Sequenzer (Abbildung 6.4). Dieser gibt pro Bar (außer Bei keys3, hier pro 1/2 Bar) einen MIDI-Notenwert innerhalb einer Oktave aus. Der Sequenzer besteht aus 8 Schritten / Steps und für jeden Schritt (aber auch für die gesamte Folge) kann der MIDI-Notenwert im Nachhinein noch transponiert werden. Die Notenwerte werden dann in das Subpatch *pm_engine* gesendet, wo die Ratio- und Amplitudeneinstellungen angewendet werden (Abbildung 6.3). Hier erfolgt eine Phasenmodulation wie bei 4.2.8. beschrieben. Bei der PM sowie FM kann die Lautstärkenregelung des Trägers nicht vor der Modulation erfolgen, daher wird die Amplitude erst nach der Modulation mit dem Wert des ersten Oszillators verändert.

Abbildung 6.4
Der Sequencer des Keys-Instruments



Nach der Synthese wird die Lautstärke das Signals durch eine Hüllkurve geformt (hier *live.adsr~*). Mit jeder MIDI-Note wird auch ein Velocity-Wert generiert, durch welchen das Hüllkurven-Signal ausgelöst wird.

Anschließend läuft das Signal in zwei Send-Objekte (*s~*), eins für den Linken- und eins für den Rechten-Kanal. Diese werden an dem passenden Mixing-Element im Hauptpatch empfangen.

6.2.2. Arp

Das Arp-Instrument (kurz für Arpeggiator: Element zur Erzeugung einer kurzen Tonfolge) erzeugt eine Melodie durch additive Synthese.

Wie die Keys-Instrumente hat auch das Arp-Instrument einen Sequenzer (vergleiche Abbildung 6.4) zur Bestimmung der Notenwerte. Die Schritte sind hier allerdings in Achteltriolen eingeteilt (6 Schritte pro Bar).

Die Notenwerte werden an den Subpatch *addsynth_engine* gesendet, wo die Klangerzeugung stattfindet. Dieses Subpatch entspricht dem Aufbau von 4.2.2. und hat ein Preset-Element zur Speicherung der Amplituden der harmonischen Schwingungen. Das erzeugte Signal verlässt *addsynth_engine* und wird anschließend durch eine Hüllkurve geformt. Diese wird auch hier durch den Velocity-Wert der MIDI-Noten

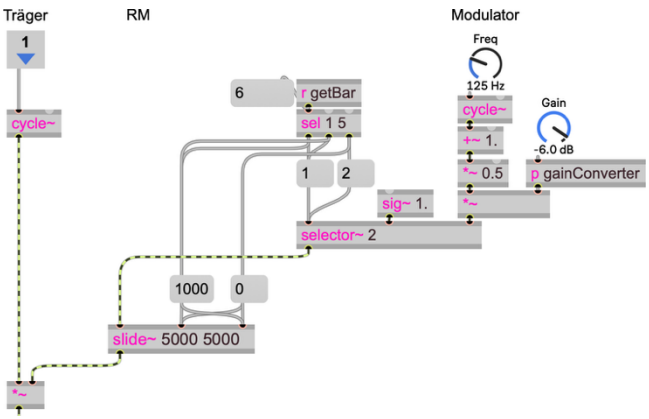
ausgelöst. Zuletzt wird das Signal an zwei Send-Objekte (Stereo) weitergegeben, um es in dem entsprechenden Mixing-Element im Hauptpatch empfangen zu können.

6.2.3. Bass

Das Bass-Instrument besteht ebenfalls aus einem Sequenzer zur Erzeugung der gewünschten Noten, welcher identisch zum dem der Keys1 und Keys2 Instrumente ist (Schritte in ganzen Noten).

Der Subpatch *bass_engine* erzeugt das Signal für den Bass mit Hilfe der Frequenzangaben des Sequenzers. Das Hauptsignal des Basses wird durch Ringmodulation (Abbildung 6.5) generiert (Ausnahmen sind Schritt 1 und 5 des Sequencers: hier wird das Signal des Trägers nicht moduliert). Die MIDI-Noten bestimmen die Frequenz des Trägers, während die Frequenz des Modulators separat eingestellt wird.

Abbildung 6.5
Ringmodulation innerhalb des Bass-Instruments



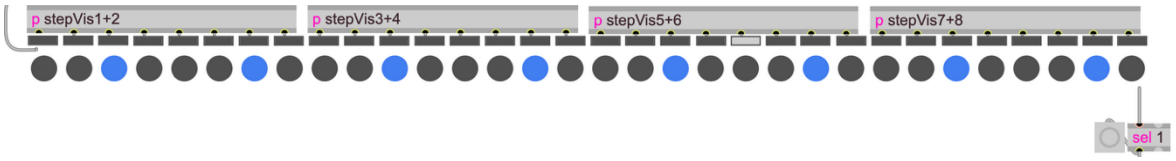
Das ringmodulierte Signal wird durch ein weißes Rauschsignal ergänzt, welches durch eine Hochpassfilter begrenzt wird (subtraktive Synthese, siehe 4.2.4). Das zusammengeführt Signal verlässt das Objekt *bass_engine*, wird nicht weiter durch eine Hüllkurve geformt und an zwei Send-Objekte übergeben (Stereo).

6.2.4. Clap

Das Clap-Instrument besteht ebenfalls aus einem Sequenzer, allerdings ist dieser anders aufgebaut. Da eine Clap ein perkussives Element ist, wird die Tonhöhe (in der Regel) nicht

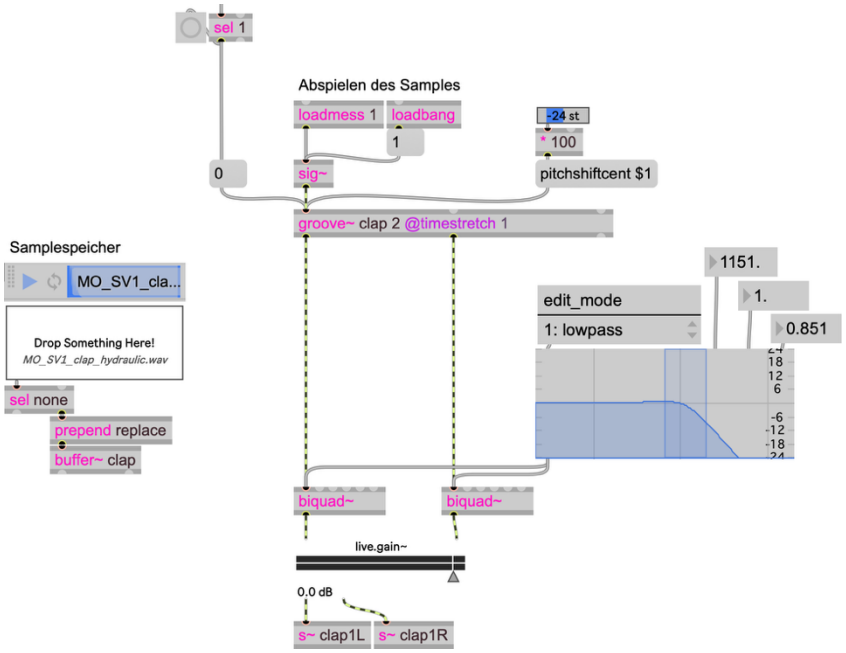
bei jedem Abspielen potenziell verändert. Demnach besteht dieser Sequenzer nur aus den einzelnen Schritten. Für jeden Beat des 8-Bar-Loops des Projekts (32 Schritte) kann durch Aktivierung einer dieser Schritte ein Clap-Sample abgespielt werden.

Abbildung 6.6
Sequencer des Clap-Instruments



Bei Erreichen eines aktivierten Schrittes wird das Objekt *groove~* angesprochen, welches in Kombination mit dem Objekt *buffer~* agiert. Letzteres dient als Speicher für Samples auf welches anschließend *groove~* (sowie weitere Objekte) durch Verwendung desselben Namens zugreifen und abspielen kann. In dem Clap-Instrument wurde in *buffer~* dementsprechend eine Audiodatei einer Clap geladen. Soll durch *groove~* das Sample in der angegebenen Samplerate in Max abgespielt werden, muss dem Objekt in den ersten Eingang ein MSP-Signal mit dem Wert 1 mitgegeben werden. Durch Mitgabe eines Max-Wertes in denselben Eingang kann der Startpunkt als Samplewert der Audiodatei bestimmt werden (die Audiodate wird dadurch auch abgespielt, sofern der Max-Wert geringer als die Länge der Datei in Samples ist und ≥ 0 ist). In dem Fall des Clap-Instruments wird bei Erreichen eines aktivierten Schrittes des Sequenzers dem Objekt *groove~* der Wert 0 überreicht, um das gespeicherte Sample von Beginn an abzuspielen. Durch Setzung des Attributes „timestretch“ auf 1 kann die Tonhöhe des Samples ohne Verlängerung oder Verkürzung der Abspielzeit verändert werden.

Abbildung 6.7
Implementierung der Objekte groove~ und buffer~ zur Ausgabe des Clap- Samples



Das Sample ist stereo, daher gibt *groove~* 2 Kanäle aus. Diese werden durch jeweils einen Tiefpassfilter geleitet und darauf an die zwei Send-Objekte zur Übergabe an das entsprechende Mixing-Element in dem Hauptpatch.

6.2.5. Closed- und Open-Hi-Hat

Das Closed-Hi-Hat-Instrument (im Projekt sowie dieser Arbeit wird die Closed-Hi-Hat nur als Hi-Hat bezeichnet) besitzt den gleichen Sequenzer wie das Clap-Instrument, jedoch können bei jedem aktivierten Schritt des Sequenzers vier Wiederholungen in zufälligen Abständen (zwischen 1/4 und 1/32 einer Barlänge) generiert werden. Ob diese stattfinden wird ebenfalls zufällig bestimmt.

Abbildung 6.8
Bestimmung der Parameter zur Wiederholung der HH-Ausgaben

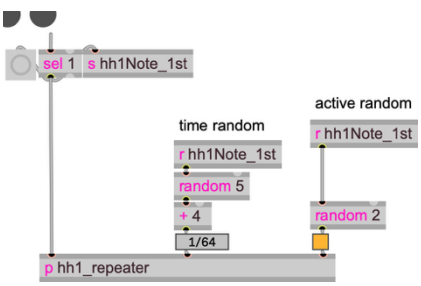
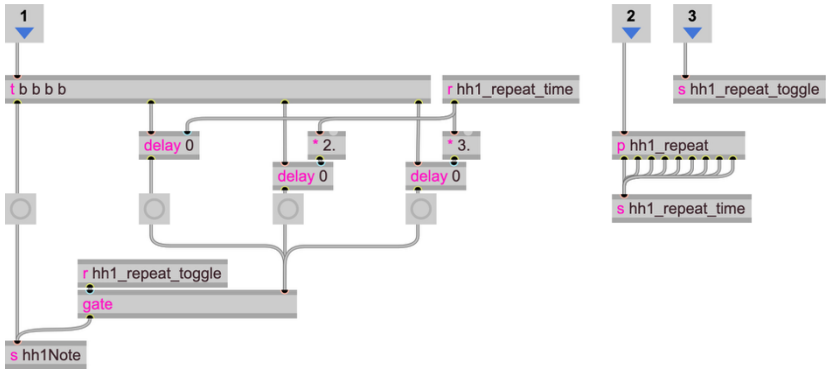


Abbildung 6.9
Umsetzung der Parameter zur Wiederholung der HH-Ausgaben



Mit jedem aktivierten Schritt des Sequenzers und dessen möglichen Wiederholungen wird eine Hüllkurve mit einer zufälligen Decay-Zeit von 10 bis (je nach Einstellung) 400 ms (Sustain auf 0 %) ausgelöst wodurch ein weißes Rauschen geformt wird. Darauf werden tiefe und mittlere Frequenzen durch einen Tiefpassfilter entfernt (subtraktive Synthese) und das Signal in zwei Send-Objekte für das Mixing-Element aufgeteilt.

Die Open-Hi-Hat besitzt einen ähnlichen Aufbau wie die Hi-Hat. Der Sequenzer agiert auf dieselbe Art und Weise und steuert eine weißes Rauschsignal. Wird ein aktivierter Schritt des Sequenzers erreicht, wird die Amplitude des Rauschens mit 1 multipliziert. Ist der aktuelle Schritt deaktiviert wird die Amplitude mit 0 multipliziert. Im Anschluss wird das Spektrum des Rauschens durch einen Bandpassfilter mit sehr hoher Cutoff-Frequenz (7227 Hz) begrenzt. Daraufhin wird das Signal an die zwei Send-Objekte übergeben und in das Hauptpatch geleitet.

6.2.6. Mixing- und Mastering-Elemente

Der Mixer im Hauptpatch ist dem Aufbau eines Mischpults oder einer DAW nachempfunden. Jedes Instrument hat einen eigenen zweispurigen Kanal in dem die Lautstärke (inklusive Mute-Funktion) und das Panning des Signals bestimmt werden kann. Die Mixing-Elemente empfangen an oberster Stelle durch Receive-Objekte (r~) die Signale der Send-Objekte der einzelnen Instrumente. Gedacht ist die weitere Bearbeitung dieser empfangenen Signale durch beliebige Effekte. In dem Projekt sind beispielsweise einige Effekte eingebunden (Hall, Saturator, Waveshaper, Limiter). Die

einzelnen Parameter der Mixing-Elemente können weiter bearbeitet werden. Zum Beispiel wird der Panning-Parameter des Arp-Instruments durch einen LFO moduliert.

Abbildung 6.10
Aufbau eines Mixing-Elements im Hauptpatch
(Arp 1 als Besipiel)

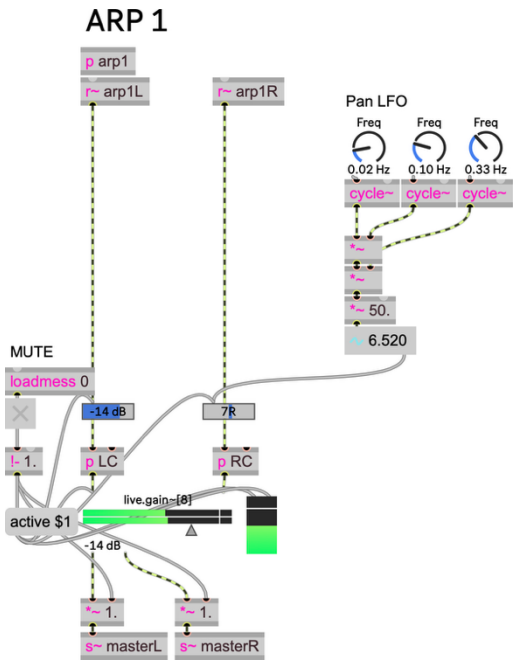
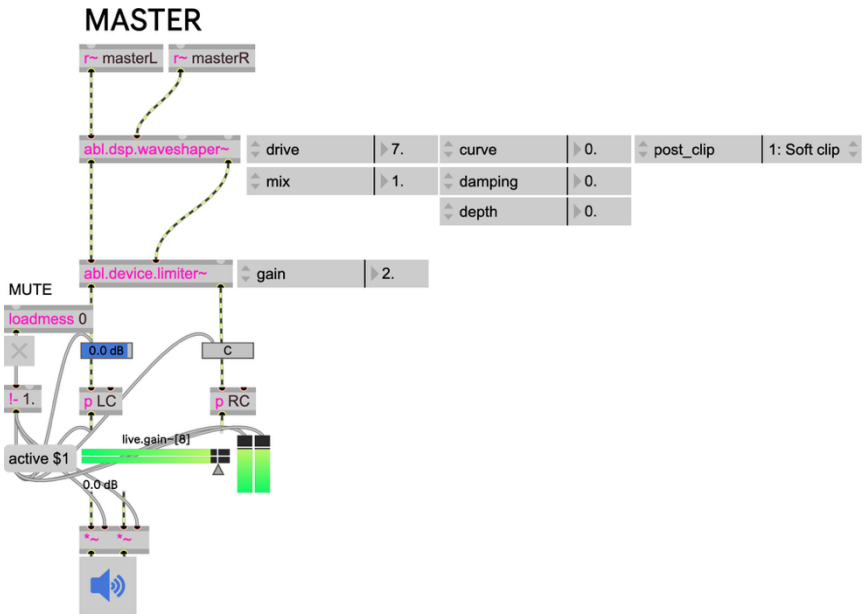


Abbildung 6.11
Aufbau des Mastering-Elements im Hauptpatch



7. Diskussion

7.1. Limitierungen von Max/MSP

Da Max ein digitales System ist herrschen auch hier die schon besprochen Limitierungen, welche noch einmal aufgegriffen werden. Zum einen gibt es die Mindestanforderungen an die Samplerate, falls mit analogen Eingangssignalen gearbeitet wird. Um nach oben hin alle für Menschen hörbaren Frequenzen abbilden zu können muss das Nyquist-Theorem beachtet werden. Zusätzlich können daraus irreversible Aliasfehler durch zu hohe Frequenzen im Eingangssignal entstehen, welche nach der A/D-Wandlung gespiegelt im Frequenzspektrum auftreten. Dadurch müssen Signale, am Eingang sowie Ausgang, einer Anti-Aliasing-Tiefpassfilterung unterzogen werden. Die Filterungs-Komponenten können dadurch eine große Rolle im ausgespielten Klang annehmen.

Bei der Zuweisung kontinuierlicher momentaner Amplitudenwerte eines analogen Signals auf diskrete digitale Werte, entstehen zusätzliche Störfrequenzen, welche durch die meist zufällig erscheinenden Quantisierungsfehler in der Regel als weißes Rauschen wahrgenommen werden (Quantisierungsrauschen). Die Stärke dieses Rauschens wird hauptsächlich durch die Bittiefe des Systems bestimmt, wird jedoch schon ab 16 Bit (CD-DA) und voller Ausnutzung der verfügbaren maximalen Amplitudenwerte kaum noch wahrgenommen (da SQNR bei 16 Bit schon fast dem maximalen Lautstärkenhörumfangs eines Menschen entspricht) [Dickreiter et al., 2023] [Cycling ’74, o.J.c].

Die Hauptlimitierungen die durch das Programm Max/MSP selbst entstehen sind weitestgehend von der Rechenleistung des verwendeten Computers geprägt. Um Echtzeit-Audio anbieten zu können, muss die CPU in kürzester Zeit sehr viele Berechnungen liefern können. Bei einer Samplerate von 48 kHz hieße das, für einen Kanal innerhalb von circa 0,02 ms alle für das nächste Sample benötigten Berechnungen zu tätigen. Dafür sind moderne CPUs in der Regel leistungsstark genug, jedoch ist der Komplexität eines Programs in Max/MSP (quasi) keine Grenze gesetzt, was die Anzahl an Berechnungen in die Höhe schnellen lassen kann. Dadurch können Verzerrungen höhrbar gemacht werden oder der Computer reagiert nicht mehr bzw. er oder das

Programm stürzen sogar ab. Zur Orientierung an die Möglichkeiten Max/MSPs nennt Cycling '74 vier Prinzipien [Cycling ’74, o.J.c]:

- Je schneller die CPU, desto besser läuft Max/MSP
- Schnellere Festplatte und Verbindung = schnelleres Ein-/Auslesen von Audiodateien
- Weniger Hintergrundtasks (z.B. Browser deaktivieren) = mehr Rechenleistung für Max/MSP
- Niedrigere Samplerate = weniger Berechnungen (aber: Verlust an Höhenwidergabe)

Außerdem sind Objekte unteschiedlich rechenintensiv. Nach außerdem gelten einfache arithmetische Operationen wie *+~, *~*, *phasor~* oder eine Floating-Point-Zahlenbox als rechenarm. Das Objekt */~* sei hingegen deutlich rechenintensiver. Objekte mit Lookup-Tabelle wie *cycle~* seien ebenfalls rechenarm. Besonders rechenintensiv (viele Berechnungen pro Sample) seien Filter-Objekte (z.B. *biquad~*), Spektral-Analysatoren (z.B. *fft~*, führt die Fast-Fourier-Transformation auf das Eingangssignal des Objekts durch) und Objekte dessen Parameter durch ein kontinuierliches Signal gesteuert wird (z.B. bei *groove~*) [Cycling ’74, o.J.c].

Eine weitere Limiterung gilt Nutzer*innen von ausschließlich Linux-Betriebssystemen, da das Programm nur für Windows und MacOS verfügbar ist.

7.2. Vorteile von Max/MSP

Cycling '74 listet folgende Möglichkeiten durch Max/MSP auf dessen Website auf [Cycling ’74, o.J.c]:

- Entwicklung von Tools, die es nicht in dieser Form gibt oder dir (im Moment) nicht zur Verfügung stehen
- Veränderungen während der Entwicklung in Echtzeit hören
- Verbindung beliebiger Parameter (z.B. mit MIDI-Daten)
- Ansteuerung automatisierter Abläufe (z.B. mit MIDI-Daten)

Zusätzlich bietet eine Entwicklungsumgebung dieser Art Vorteile wie keine extra Anschaffung einer expliziten Hardware, da die meisten Menschen (nicht nur die welche bereits vorher elektronische Musik kreierten) bereits einen Computer besitzen. Die analoge Umsetzung eines Patches wäre zwar teilweise möglich und in manchen simplen Fällen auch einfacher, allerdings sind viele Patches nur mit deutlich größerem Aufwand und größeren Kosten oder sogar gar nicht umsetzbar. Zusätzlich ist die Signal-to-Noise-Ratio bei digitalen Systemen deutlich geringer.

7.3. Zusätzliche Möglichkeiten und Erweiterungen von Max/MSP

Cycling '74 bietet außerdem einige kostenlose und selbstentwickelte sowie von Externen entwickelte Bibliotheken (Packages) an, wie z.B. Miraweb, jit.mo und ease (alle Cycling '74) oder CNMAT Externals (Center for New Music and Audio Technologies der UC Berkeley), AudioMix (Manuel Poletti) und Rhythm and Time Toolkit (Philip Meyer) online an. Packages können auch selbst programmiert und zur Verfügung gestellt werden.

Durch Jitter-Objekte wird zudem die Generierung von Grafik- und Videomaterial ermöglich und diese können mit Max- sowie MSP-Objekten kommunizieren und es können innerhalb der Max-Umgebung Patches mit der unabhängigen Programmiersprache Gen entwickelt werden (arbeitet eine Ebene tiefer als Max und Daten werden generell pro Sample verarbeitet). Ähnlich zu Gen ist RNBO, was den Export zur Nutzung in einem Raspberry Pi, als VST oder AU ermöglicht und den Export als Code in C++ oder JavaScript ermöglicht. Für Nutzer*innen von Ableton Live bietet Max eine einfache Implementierung der Patches in die DAW durch die Erweiterung Max4Live an.

7.4. Vergleich mit anderen Sprachen und Umgebungen

Neben Max gibt es auch weitere Programmiersprachen und Entwicklungsumgebungen mit welchen gleiche Ziel verfolgt und erzielt werden können. Als nächste Verwandte kann Pure Data (Pd) gesehen werden, welche wie Max bedient wird. Pd ist kostenlos erhältlich und besitzt dieselben grundlegen Möglichkeiten, ist nur in der weitergehenden Vielfalt

beschränkter und hat konstanten Support sowie weniger ausführliche Dokumentationen. Reaktor (Native Instruments) wird ähnlich bedient, ist allerdings keine eigene Sprache und muss innerhalb des dazugehörigen Plug-Ins bedient werden. Reaktor kann als modularer Software-Synthesizer verstanden werden.

SuperCollider und Csound ermöglichen es ebenfalls Audio-Instrumente und Effekte zu entwickeln, sind jedoch rein schriftliche Programmiersprachen sowie Tidal Cycles und Strudel (Tidal Cycles als webbasierte Applikation), welche vor allem für Live-Coding (Musizieren durch Umschreiben des Codes) gedacht sind.

8. Fazit und Ausblick

8.1. Zusammenfassung der Ergebnisse

Auch wenn das finale Max/MSP-Patch den Zielen grundlegend entspricht, war der Weg zur Umsetzung dieser häufig nicht klar ausgelegt. Durch die sehr offene Gestaltung der Entwicklungsumgebung (nicht nur visuell durch das Interface, sondern auch inhaltlich) standen zu jeder Zeit für die allermeisten Arbeitsschritte mehrere Möglichkeiten im Raum zwischen welchen letztlich entschieden werden muss. Gleichzeitig wurde die Auswahl eines Weges durch das Anfertigen dieser Arbeit limitiert bzw. musste diese mit in Betracht gezogen werden.

Das übersichtliche Interface, die extensiven Dokumentation, die Help-Patches der einzelnen Objekte und der intuitive Umgang mit diesen sowie deren Kommunikation untereinander ermöglichte das schnelle Finden einer Lösung für die meisten Hindernisse. So gut wie alle Konzepte, die aus der elektronischen Musik bekannt sind, können in Max/MSP umgesetzt und beliebig erweitert werden. Zusätzlich sorgt dies für ein klareres Verständnis einiger Vorgänge, welches auch außerhalb der Max/MSP-Umgebung angewendet werden kann.

Da fast alle von Max/MSP bereitgestellten Objekte entweder direkt oder über Umwege miteinander kommunizieren können und die Sprache selbst auf Modularität beruht, war

die Umsetzung dieser in einem eigenen Projekt gut möglich. Weitere Anpassungen und Erweiterungen an dem eigenen sind zusätzlich umsetzbar und auch bei externen Projekten möglich, da die Programmstruktur in der Regel schnell durch die visuelle Gestaltung der Kommunikationswege leicht nachvollziehbar ist.

Die Umsetzung ist also gelungen, jedoch kann die der Fragestellung dieser Arbeit nicht klar beantwortet werden. Da es heutzutage bereits eine riesige Bandbreite an digitalen Tools zur Klangsynthese und -verarbeitung gibt, wird vielen Nutzer*innen das Entwickeln eines eigenen Tools in der Erschaffung elektronischer Musik eher zusätzliche Zeit kosten und nicht in Frage kommen. Beruht die zu erschaffende Musik jedoch auf komplexen Modulationsstrukturen und Bearbeitungsschritten in kleinsten Zeitintervallen, kann Max/MSP große Vorteile und Möglichkeiten bieten.

8.2. Perspektive für Weiterentwicklung

Dem Programm stehen endlos viele Möglichkeiten zur Erweiterung offen. Die Instrumente zur Erzeugung der harmonischen Elemente der Komposition sind derzeitig noch monophon (können nicht mehrere Noten gleichzeitig spielen). Um dies Umzusetzen könnten die Subpatches zu Erzeugung der Klänge durch *poly~* erweitert werden, was das Instanziieren von Patches möglich macht, d.h. dasselbe Subpatch in voneinander unabhängigen mehreren Instanzen gleichzeitig laufen zu lassen. Ein Art der Polyphonie wäre auch durch Multichannel-Objekte umsetzbar, welche mehrere Signale über ein Kabel senden können.

Außerdem kann das Programm durch beliebige Instrumente anderer komplexerer Syntheseverfahren implementiert werden, wie z.B. einen Granulator.

Trotz bestehender Modularität könnte diese außerdem weitergeführt und vereinfacht werden. Bisher müssen viele Objekte umhergeschoben werden, um Platz für bestimmte neue Bauteile schaffen zu können. Vor allem in den Mixing- und Mastering-Elementen könnte das Einfügen von Effekten und neuen Bauteilen vereinfacht und übersichtlicher gestaltet werden.

9. Legende relevanter Max/MSP-Objekte

Es gilt für alle Objekte: Die Ein- und Ausgänge werden von links nach rechts nummeriert.

Tabelle 9.1
Legende für diese Arbeit relevanter Max/MSP-Objekte

	Eingang 1 wird mit Eingang 2 oder dem Wert im Objekt addiert (Eingang 2 überschreibt Wert im Objekt)
	Eingang 1 wird mit Eingang 2 oder dem Wert im Objekt multipliziert (Eingang 2 überschreibt Wert im Objekt)
	Ausgang 1 gibt durch Zeiger bestimmten Float-Wert aus (hier 220.)
	Ausgang 2 gibt durch Zeiger bestimmten linearen Float-Wert zwischen 0. und 1. aus
	Erzeugt eine Sinusschwingung mit durch Eingang 1 bestimmter Frequenz
	Erzeugt eine Sägezahnschwingung mit durch Eingang 1 bestimmter Frequenz
	Filtert an Eingang 1 eingehendes Signal anhand von Koeffizienten (Filter 2. Ordnung; Koeffizienten können als Liste an Eingang 1 oder einzeln an die anderen Eingänge übergeben werden)
	Gibt dem Graphen entsprechende Liste aus Koeffizienten für *biquad~* aus (Typ des Filters sowie dessen Cutoff-Frequenz, Verstärkung und Güte-Faktor kann an den Eingängen bestimmt werden)

Quellenverzeichnis

Ahmad, R. (1999). Visual Languages: A New Way Of Programming. *Malaysian Journal of Computer Science, 99* (12), 76-81.

Apple Inc. (o.J.). *About losless audio in Apple Music*. Abgerufen am 25. September 2025, von https://support.apple.com/en-us/118295

Chaudhari, Q. (o.J.). *On Analog-to-Digital Converter (ADC), 6 dB SNR Gain per Bit, Oversampling and Undersampling*. Abgerufen am 25. September 2025, von https://wirelesspi.com/on-analog-to-digital-converter-adc-6-db-snr-gain-per-bit-oversampling-and-undersampling/

Cipriani, A., & Giri, M. (2019). *Electronic Music and Sound Design: Theory and Practice with Max 8 – Volume 2*. Contemponet.

Cycling ’74. (o.J.a). *About Cycling ’74*. Abgerufen am 25. September 2025, von https://cycling74.com/company

Cycling ’74. (o.J.b). *cycle~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/cycle~/

Cycling ’74. (o.J.c). *How Digital Audio Works*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/learn/articles/02_mspdigitalaudio/

Cycling ’74. (o.J.d). *How MSP Works: Max Patches and the MSP Signal Network*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/learn/articles/03_msphowmspworks/

Cycling ’74. (o.J.e). *Message Types*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/userguide/message_types/

Cycling ’74. (o.J.f). *noise~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/noise~/

Cycling ’74. (o.J.g). *phasor~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/phasor~/

Cycling ’74. (o.J.h). *pink~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/pink~/

Cycling ’74. (o.J.i). *rect~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/rect~/

Cycling ’74. (o.J.j). *saw~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/saw~/

Cycling ’74. (o.J.k). *triangle~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/triangle~/

Cycling ’74. (o.J.l). *tri~*. Abgerufen am 25. September 2025, von https://docs.cycling74.com/reference/tri~/

Dickreiter, M., Dittel, V., Hoeg, W., & Wöhr, M. (Hrsg.). (2023). *Handbuch der Tonstudiotechnik*. Walter de Gruyter GmbH.

Dobrian, C. (o.J). *Generate classic waveforms*. Abgerufen am 25. September 2025, von https://music.arts.uci.edu/dobrian/maxcookbook/generate-classic-waveforms

Fine, T. A. (1995). *All about Narrowband ISDN*. Abgerufen am 25. September 2025, von https://hea-www.harvard.edu/~fine/ISDN/n-isdn.html

Kester, W. (Hrsg.). (2004). *Analg-Digital Conversion*. Analog Devices, Inc.

Kester, W. (2009). *Taking the Mystery out of the Infamous Formula, “SNR = 6.02 + 1.76dB,” and Why You Should Care*. Abgerufen am 25. September 2025, von https://www.analog.com/media/en/training-seminars/tutorials/MT-001.pdf

Lenovo Deutschland (o.J.). *Was ist eine Workstation?*. Abgerufen am 25. September 2025, von https://www.lenovo.com/de/de/glossary/workstation-vs-desktop-business/?orgRef=https%253A%252F%252Fwww.ecosia.org%252F

Puckette, M. (1988). The Patcher. *Proceedings of the 1986 International Computer Music Conference. San Francisco: International Computer Music Association*, 420-429.

Puckette, M. (o.J.). *Wavetables and samplers*. Abgerufen am 25. September 2025, von https://msp.ucsd.edu/techniques/v0.03/book-html/node22.html#fig02.01

Thomas, C. (2024). *High bitrate audio is overkill: CD quality is still great*. Aberufen am 25. September 2025, von https://www.soundguys.com/high-bitrate-audio-is-overkill-cd-quality-is-still-great-16518/

cooksonia (2024, 14. November). *The difference between FM and Phase Modulation – Examples in gen~* [Video]. YouTube. https://www.youtube.com/watch?v=HeTk6JDlkqY

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

a	Arp oder Arpeggiator (in Bezug auf das Max/MSP-Projekt)
ADC	Analog-to-Digital-Converter
ADSR	Attack, Decay, Sustain, Release
A/D-Wandler	Analog-Digital-Wandler
ALAC	Apple Losless Audio Codec
AM	Amplitudenmodulation
AU	Audio Unit
b	Bass (in Bezug auf das Max/MSP-Projekt)
BPM	Beats per Minute
c	Clap (in Bezug auf das Max/MSP-Projekt)
CD-DA	Compact Disc Digital Audio
CPU	Central Processing Unit
DAC	Digital-to-Analog-Converter
DAW	Digital Audio Workstation
dB	Dezibel
D/A-Wandler	Digital-Analog-Wandler
DSP	Digital Sound Processing oder Digital Sound Processor
f	Frequenz
f_a	Frequenz im analogen System
$f_{a_{max}}$	höchste vorkommende Frequenz eines analogen Systems
f_{AF}	Aliasfehler
f_M	Frequenz des Modulators einer RM, AM, FM oder PM
f_{Ny}	Nyquist-Frequenz
f_S	Samplerate
f_T	Frequenz des Trägers einer RM, AM, FM oder PM
f_{SB}	Frequenz eines Seitenbands einer Modulation
FM	Frequenzmodulation
hh	(Closed-)Hi-Hat (in Bezug auf das Max/MSP-Projekt)
Hz	Hertz
IRCAM	Institut de recherche et coordination acoustique / musique
ISDN	Integrated Services Digital Network
ISPW	IRCAM Signal Processing Workstation

k	Keys (in Bezug auf das Max/MSP-Projekt)
kHz	Kilohertz
LFO	Low Frequency Oscillator
MIDI	Musical Instrument Digital Interface
ms	Millisekunde
µV	Mikrovolt
ophh	Open-Hi-Hat (in Bezug auf das Max/MSP-Projekt)
Pa	Pascal
Pd	Pure Data
PM	Phasenmodulation
RM	Ringmodulation
SPL	Schalldruckpegel
SQNR	Signal-to-Quantization-Noise-Ratio
V	Volt
VPL	Visual Programming Language
VST	Virtual Studio Technology

Anhang

Anhang A

Max-Collective-Datei (.mxf)

BA_JonasMayr.mxf

Anhang B

Max-Patch-Datei (.maxpat)

BA_Main.maxpat