Hochschule der Medien

Ton-Seminar

Plugin-Programmierung

Clemens Hildebrand

14.12.2020

Inhaltsverzeichnis

1 Grundlagen

1.1 Plug-Ins

Plug-Ins sind Programmerweiterungen, die über Programmierschnittstellen (auch API - application programming interface) an Host-Programme angebunden werden. Diese Schnittstellen geben einen Kommunikationsstandard zwischen Host und Plug-In vor.

Bei Software zur Audiobearbeitung bzw. Musikproduktion sind Plug-Ins mit Effektgeräten vergleichbar. Kauft und integriert man im realen Studio eine neue Hardware, so installiert man im virtuellen Studio ein Plug-in.

Plug-Ins im Audio-Bereich sollen in den meisten Fällen in Echtzeit arbeiten - somit sind sie zeitkritisch. Bevorzugte Programmiersprache für Plug-Ins ist C++, weil sie sehr nah am System arbeitet und dadurch gut für zeitkritische Anwendungen geeignet ist.

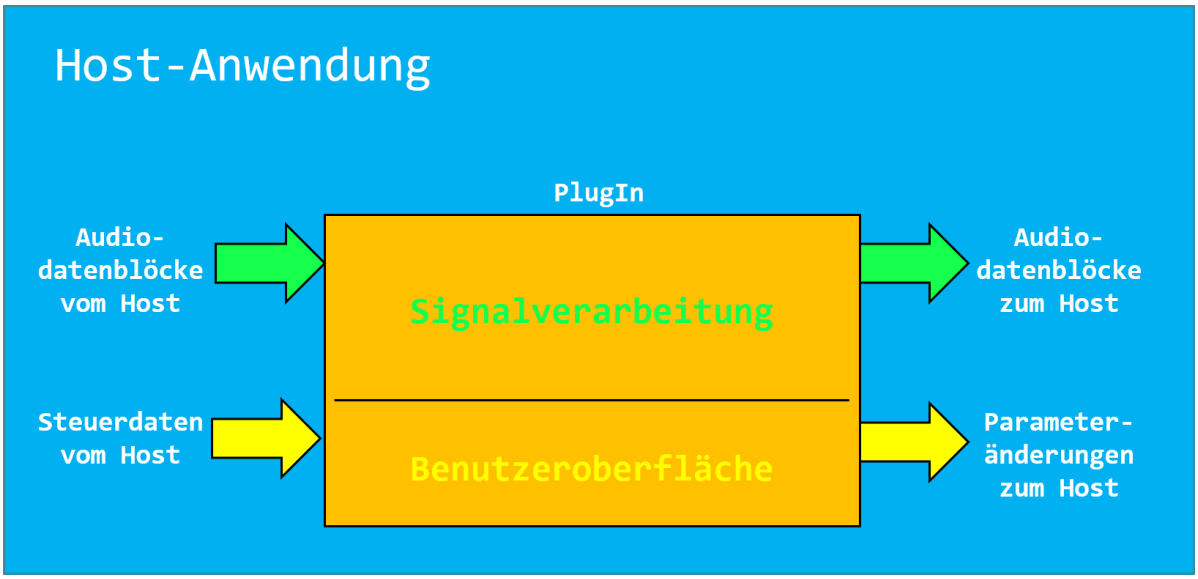

Abbildung 1: Integration von Plug-In in Hostanwendung

1.2 Plug-In Architekturen

Plug-In Architekturen sind Standards für die Kommunikation zwischen Plug-Ins und Hostanwendungen. Es gibt eine Vielzahl verschiedener Architekturen - nicht jeder Host unterstützt alle Architekturen. Einige proprietäre Architekturen sind lediglich für einzelne Programme lizenziert, wie zum Beispiel AAX - Avid Audio eXtension. Die wohl populärste Plug-In Architektur ist VST - Virtual Studio Technology von Steinberg.

Name	Developer	License	GUI support	Supported types	Supported platforms
Rack Extension	Reason Studios	BSD-style [8]	Yes	Transformation and synthesis	Mac OS X and Windows
Virtual Studio Technology	Steinberg	Proprietary or GPLv3[9]	Yes	Transformation and synthesis	Mac OS X, Windows and Linux[10]
Audio Units	Apple	Proprietary	Yes	Transformation and synthesis	Mac OS X, iOS and tvOS [11]
Real Time AudioSuite	Avid	Proprietary	Yes	Transformation and synthesis	Mac OS X and Windows
Avid Audio eXtension	Avid	Proprietary	Yes	Transformation and synthesis	Mac OS X and Windows
TDM	Avid	Proprietary	Yes	Transformation and synthesis	Mac OS X and Windows
LADSPA	ladspa.org	LGPL	No	Transformation	Mac OS X, Windows and Linux
DSSI	dssi.sourceforge.net	LGPL, BSD	Yes	Transformation and synthesis	Mac OS X, Windows and Linux
LV2	lv2plug.in	ISC License	Yes	Transformation and synthesis	Linux, OS X, Windows
DirectX plugin	Microsoft	Proprietary	Yes	Transformation and synthesis	Windows
VAMP	vamp-plugins.org	BSD-style	No	Analysis	Mac OS X, Windows and Linux

Abbildung 2: Die gängigsten Plug-In Architekturen für Audioanwendungen

Da Linux im Bereich der Audioproduktion eine eher untergeordnete Rolle spielt, sind die meisten Plug-In Architekturen nicht dafür verfügbar. Aus diesem Grund gibt es dafür vor allem nicht-proprietäre Architekturen.

1.3 VST Plug-Ins

Als am weitesten verbreitete Architektur wird hier tiefer auf VST - Virtual Studio Technology eingegangen. Da diese Architektur auch schon fast so alt wie die ersten DAWs (Digital Audio Workstations) ist, lässt sich daran sehr gut die Geschichte und Entwicklung von Plug-Ins zeigen.

Die Schnittstellenspezifikation VST wurde im Jahr 1996 von Steinberg zusammen mit der Version 3.02 der DAW Cubase veröffentlicht. Die ersten Plug-Ins waren Espacial (Reverb-Plug-In), Choirus, Stereo Echo und ein Auto-Panner. VST wurde im Jahr 1999 auf die Version 2.0 aktualisiert. Damit wurde die Weiterverarbeitung von MIDI-Daten (Musical Instrument Digital Interface) ermöglicht. Auf Basis dessen konnte eine neue Gattung der VST-PlugIns entstehen - die VSTi-PlugIns (Virtual Studio Technology Instrument). VST-Instrumente können als eigenständige Software-Synthesizer, Sampler oder Drumcomputer eingesetzt werden. (Wikipedia, 2020c)

Das erste verfügbare VST-Instrument 3 war ein 16-stimmiger virtueller analoger Synthesizer mit zwei Oszillatoren, ADSR Envelopes, einem LFO und einem Filter.

Abbildung 3: Das erste VST-Instrument „Neon“

Im Jahr 2006 wurde die VST-Schnittstellenspezifikation auf Version 2.4 aktualisiert. Zu den Änderungen gehörte die Möglichkeit, Audio mit 64-Bit Genauigkeit zu verarbeiten.

Mit Version 3.0 wurden im Jahr 2008 Audio-Inputs für VST-Instrumente eingeführt, sowie multiple MIDI Inputs/Outputs und optional die Steinberg Kernel Interface Einbindung (SKI), mit der Plugins von Steinberg auf proprietären DSPs ausgeführt werden können.

Anschließend wurde VST 2011 mit Version 3.5 ein Notenausdruck hinzugefügt, mit dem einzelnen Notenereignissen in einer polyphonen Anordnung Artikulationsinformationen zugeordnet werden können.

Im September 2013 wurde die Wartung des VST2 SDKs eingestellt und die Verbreitung im Dezember des selben Jahres gestoppt. Höhere Versionen werden fortgesetzt.

Seit 2017 gibt es VST auch für Linux-Systeme - mit Version 3.6.7 existiert dafür eine Dual-Lizenz im SDK: „Proprietary Steinberg VST3“ und „Open-source GPLv3“.

1.4 Grundbausteine DSP

Um ein Audio-Plug-In zu entwickeln, benötigt man die Grundbausteine des DSP (Digital Signal Processing). Diese sind Multiplizierer, Addierer und Verzögerer - bis auf den Verzögerer sind die Bausteine die einfachsten numerischen Operationen, die ein Prozessor beherrscht und sind nicht zeitaufwändig. Verzögerer benötigen immer kurzzeitig Speicherplatz und sorgen dafür, dass die Zeit zur Signalberechnung größer wird. Mithilfe dieser Grundbausteine lassen sich sämtliche Effekte realisieren. Bei der Plug-In Entwicklung wird aber in der Regel nicht auf dieser Ebene programmiert, sondern es gibt bereits fertige Funktionen in der Entwicklungsumgebung, auf die zurückgegriffen werden kann.

Allerdings gilt: Je näher an den Grundbausteinen programmiert wird, desto weniger Ressourcen benötigt am Ende das Processing, da nicht so viel unnötiger Code aus fertigen Funktionen nebenher mitläuft.

1.5 Arbeitsbereich DSP

Ein DSP-System kann gleichwertig auf zwei verschiedene Weisen beschrieben werden - im Frequenzbereich und im Zeitbereich. Jede dieser beiden Betrachtungsweisen hat bestimmte Vorteile und kann gewisse Informationen bereitstellen, die in der anderen Betrachtungsweise nicht ersichtlich würden. Mithilfe von Fourier-Transformation können die Arbeitsbereiche ineinander überführt werden.

Im Zeitbereich wirken sich Multiplikation und Addition auf die Amplitude aus, während sie sich im Frequenzbereich auf die Frequenz auswirken.

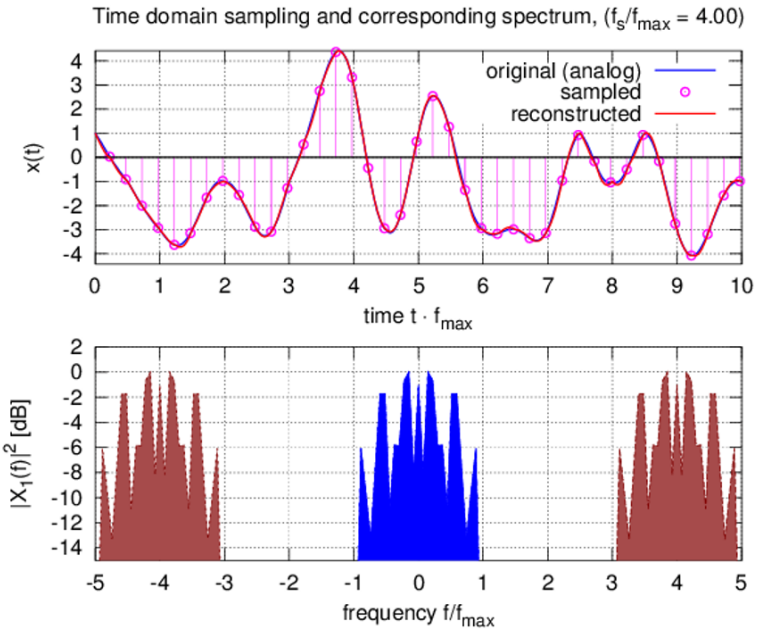

Abbildung 4: Darstellung der Arbeitsbereiche für ein Signal
(Knaub, 2020)

Eine Faltung im Zeitbereich entspricht einer Multiplikation im Frequenzbereich. Um z.B. ein Signal zu Falten, wird es nach einer FFT (Fast Fourier Transformation) im Frequenzbereich Multipliziert und anschließend mittels reverser FFT wieder zurückgewandelt. Auf diese Weise kann die relativ komplexe Faltungsoperation sehr effizient und mit einfachen Operationen durchgeführt werden.

Da im normalen Anwendungsfall meist mit fertigen Funktionen programmiert werden kann, gehe ich an dieser Stelle nicht weiter auf die Schaltungstechnischen Grundlagen ein.

2 Entwicklungsprozess

2.1 Allgemeines Vorgehen

Die Plug-In Entwicklung läuft im Allgemeinen in folgender Reihenfolge ab:

1. Definition des Algorithmus und der benötigten Parameter
2. Implementieren einer Testumgebung
3. Algorithmus testen durch Messung, Simulation, etc.
4. Portierung in ein Plug-In mit einfachen GUI Elementen
5. Testung und gegebenenfalls GUI Entwicklung
6. Betatest

Oftmals laufen Punkt 4 und 5 parallel ab - für die Testung benötigt man ein GUI (Graphical User Interface).

2.1.1 Signalfluss-Diagramm

Signalfluss-Diagramme stellen den Weg des Signals durch den Effekt oder das Plug-In dar und bieten einen guten Überblick, was mit dem Signal passieren soll. Für die Plug-In Entwicklung ist es meist sehr hilfreich, sich zu Anfang ein solches Diagramm zu erstellen.

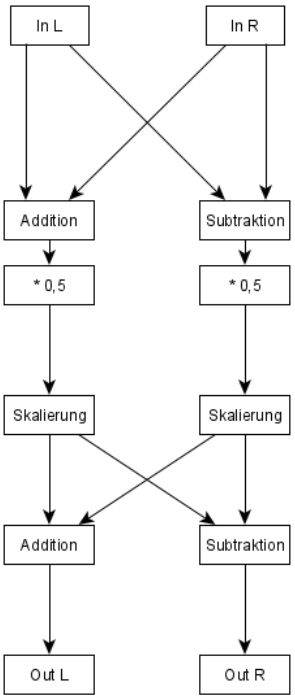

Abbildung 5: Signalfluss-Diagramm für Plug-In zur Regelung der Stereobreite

2.1.2 Parameter und Variablen festlegen

Im nächsten Schritt muss sich überlegt werden, welche Parameter das Verhalten des Plug-Ins beeinflussen können und welche davon verändert werden können sollen. Daraus leitet sich ab, welche Variablen und Konstanten es geben muss und entsprechend auch welche Bedienelemente es auf dem GUI geben wird.

2.1.3 Code schreiben

In diesem Abschnitt wird das Plug-In Konzept in einen Algorithmus überführt. Ein Algorithmus ist eine detaillierte und explizite Vorschrift zur schrittweisen Lösung eines Problems. Es gelten folgende Eigenschaften:

- Die Ausführung des Algorithmus erfolgt in einzelnen Schritten
- Jeder Schritt umfasst lediglich eine einfache Grundaktion
- Zu jedem Zeitpunkt muss klar sein, welcher Schritt als Nächstes auszuführen ist

Danach wird der Algorithmus in einer geeigneten Programmiersprache aufgeschrieben. In den meisten Fällen ist dies für Audio-Plug-Ins C++.

2.1.4 Testen

Um das Plug-In zu testen, muss dafür eine Testumgebung implementiert werden, die das Plug-In hosten kann und entsprechende Eingänge und Ausgänge bereitstellt.

Anschließend werden sinnvolle Wertebereiche für die Parameter festgelegt.

Beim Testen des Plug-Ins muss insbesondere auf die Folgenden Punkte Augenmerk gelegt werden:

- Erfüllt das Plug-In die gewünschten Funktionen?
- Läuft das Plug-In stabil?
- Sind die Laufzeiten bzw. durch das Plug-In verursachten Latenzen in einem vertretbaren Ausmaß?
- Gibt es unerwünschte Artefakte und welche Parameter-Kombinationen rufen diese hervor?
- Sind die Wertebereiche wirklich sinnvoll gewählt?

Nachdem diese Fragen beantwortet sind und alle Korrekturen gemacht, wird nochmals über diese Schritte iteriert, bis das Plug-In allen Anforderungen genügt.

2.1.5 GUI designen

In den meisten Fällen muss in diesem Schritt das Test-GUI weiter optimiert werden, wobei es vor allem zu einer intuitiven Bedienung beitragen soll, visuell ansprechend sein soll, sowie ggf. einem Corporate Design entsprechen muss.

3 Entwicklungsumgebungen

Für die Plug-In Entwicklung gibt es verschiedene Umgebungen - welche man nutzt hängt stark davon ab, für welche Host-Anwendungen man das Plug-In entwickelt.

3.1 MATLAB von Mathworks

MATLAB ist eine kommerzielle Software des US-amerikanischen Unternehmens MathWorks zur Lösung mathematischer Probleme. MATLAB ist vor allem für numerische Berechnungen mithilfe von Matrizen ausgelegt, woher sich auch der Name ableitet: *MATrix LABoratory*. Es handelt sich bei MATLAB sowohl um eine Bezeichnung für eine Entwicklungsumgebung, als auch für die darin verwendete Programmiersprache. Die Programmiersprache wurde Ende der 1970er Jahre an der Universität New Mexico entwickelt, um den Studenten Fortran-Bibliotheken für lineare Algebra von einer Kommandozeile aus ohne Programmierkenntnisse in Fortran zugänglich zu machen. Deshalb hat MATLAB eine stark mathematisch orientierte Syntax.

Innerhalb der Entwicklungsumgebung werden themenspezifische Toolboxen zur Verfügung gestellt. Unter Anderem gibt es die Audio-Toolbox mit Bibliotheken für die Plug-In Entwicklung. MATLAB kann selbst als Testumgebung für die Plug-In Entwicklung genutzt werden. Mit MATLAB können VST- und AU-Plug-Ins erstellt werden.

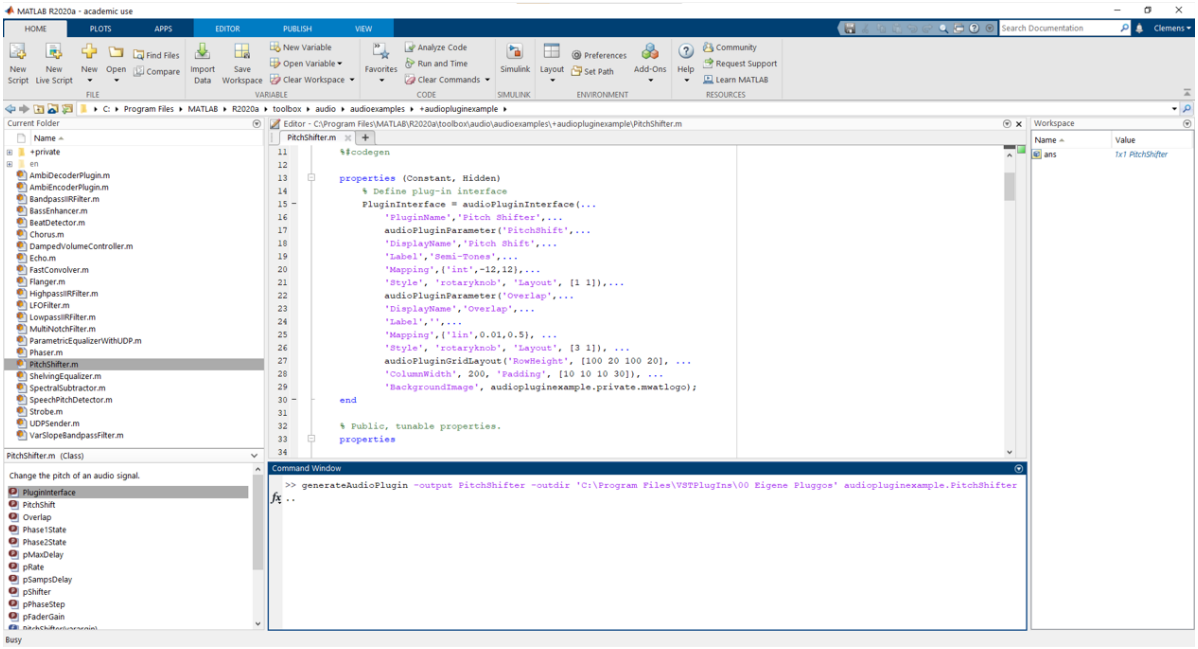

Abbildung 6: MATLAB Entwicklungsumgebung

In Abbildung 6 sieht man die Benutzeroberfläche von MATLAB. Der Plug-In Code in MATLAB zeichnet sich dadurch aus, dass er im Vergleich zu anderen Entwicklungsumgebungen relativ kurz ist. Beherrscht man die Sprache, ist hier also sehr schnell ein Prototyp erstellt.

MATLAB ermöglicht es, die Plug-In Prototypen für JUCE zu exportieren, um dort das GUI zu verändern - in MATLAB selbst bekommt man nur eine relativ Nüchterne Standard-Benutzeroberfläche. (Wikipedia, 2020b)

3.2 JUCE

JUCE (kurz für „Jules' Utility Class Extensions“) ist eine Entwicklungsumgebung für die Programmiersprache C++, die Open-Source entwickelt wird. Sie hat den Anspruch, Programme zu erzeugen, die für alle gängigen Betriebssysteme kompiliert werden können und sich überall gleich verhalten.

JUCE zeichnet sich insbesondere durch seine Großen GUI- und Plug-In-Bibliotheken aus, die eine Entwicklung von Audio-Plug-Ins stark erleichtern.

JUCE ist eigentlich ein Nebenprodukt der Entwicklung der DAW „Tracktion“ - dafür wurden die JUCE-Libraries 2002 ursprünglich angelegt und seit 2004 parallel dazu vertrieben.

JUCE ist durch den Anspruch an die universelle Funktionalität auf allen Plattformen wohl die Umgebung, die am besten für die Entwicklung von Plug-Ins geeignet ist.

Für das kompilieren der Plug-Ins wird Microsoft Visual Studio 2019 (Windows) oder XCode (macOS) benötigt. (Wikipedia, 2020a)

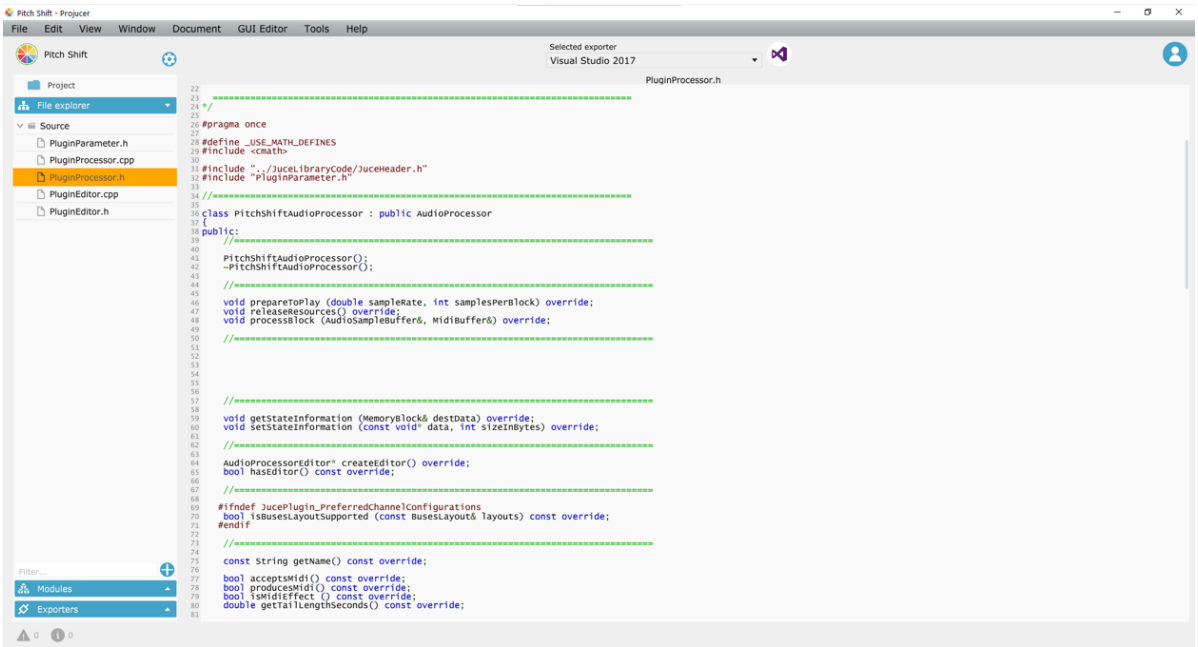

Abbildung 7: JUCE Entwicklungsumgebung

3.3 MaxMSP (Max for Live)

Eine Entwicklungsumgebung, die etwas unkonventionell arbeitet, ist MaxMSP von Cycling'74. Es handelt sich dabei um eine graphische, integrierte Entwicklungsumgebung für Musik und Multimedia, die für Echtzeitprozesse ausgelegt ist.

Die Entwicklungsumgebung ist stark modular - Programmroutinen existieren als „shared libraries“ (über eine API können von Dritten neue Routinen erstellt werden). Max wird durch eine große Community von unabhängigen Programmierern ständig erweitert. Max ist eine Art Universalsprache zur Entwicklung von Audio-Software und ist durch das visuelle programmieren sehr einfach für Menschen mit musikalischem bzw. elektrischem Hintergrundwissen zu bedienen, ohne eine Programmiersprache beherrschen zu müssen.

Max wurde in den 80er Jahren bei IRCAM (Institut de Recherche et Coordination Acoustique/Musique) in Paris entwickelt - unter dem Namen "The Patcher". Ende der 80er wurde die Software an Opcode Systems lizenziert, von dort stammt auch ein Derivat von Max, welches *Pure Data* heißt. Anfang der 90er wurde Max von Cycling'74 gekauft. 2017 wurde Cycling'74 von Ableton aufgekauft - Max wird weiterhin von Cycling'74 vertrieben, welche nun aber eine Tochterfirma von Ableton ist. Dadurch wurde die Anbindung von Max an Ableton Live mit Max4Live intensiviert.

Max bietet von Haus aus erstmal keine Möglichkeit mehr, Plug-Ins für andere DAWs als Ableton Live zu erstellen. Allerdings gibt es einen Max-Patch, mit dem Plug-Ins für JUCE exportiert werden können, dort ein neues UI bekommen müssen und dann entsprechend auf allen Plattformen vertrieben werden können.

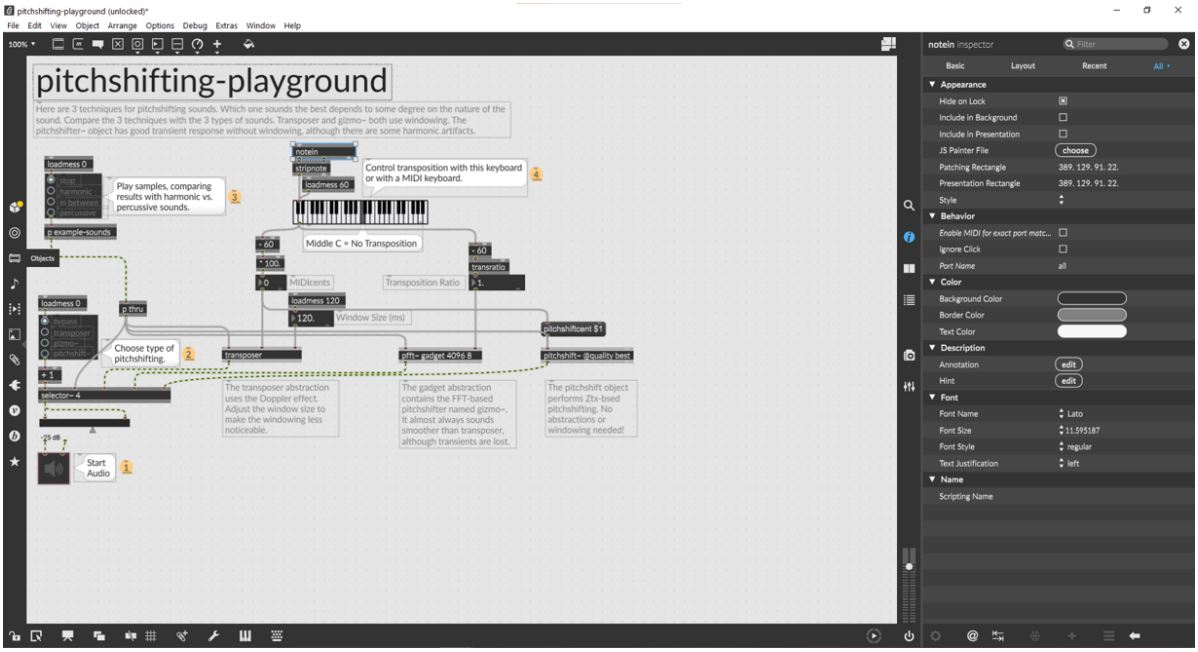

Abbildung 8: MaxMSP Entwicklungsumgebung

3.4 Vergleich der Entwicklungsumgebungen

Da MATLAB nicht primär für die Entwicklung von Audio-Plug-Ins gedacht ist, gibt es einige Hürden zu überwinden. Für Menschen mit wissenschaftlich-mathematischem Hintergrund ist MATLAB sicherlich das Mittel der Wahl. Außerdem gibt es eine gute Dokumentation, jedoch außerhalb dessen wenig Hilfe für den Start. MATLAB mit entsprechenden Toolboxen für die Audio-Entwicklung kostet rund 250€.

JUCE setzt gute Vorkenntnisse in C++ voraus. Es gibt dazu eine gute Dokumentation und eine große Community. JUCE ist genau wie MATLAB nicht für eine bestimmte DAW optimiert und die einzige der hier vorgestellten Entwicklungsumgebungen, die ein ausführlicheres GUI-Design ermöglicht. Außer für die kommerzielle Verwendung ist JUCE komplett kostenlos.

MaxMSP ist für nicht-Informatiker wohl die einfachste Lösung, da es im Prinzip keine Programmiervorkenntnisse erfordert. Die Dokumentation ist sehr ausführlich und enthält zudem viele Tutorials, außerdem gibt es eine große Community und man kann sämtlichen Patches sehr einfach „unter die Motorhaube schauen“ und sich Fragmente daraus kopieren. Max hat den großen Nachteil, dass es das Erstellen von VST-Plugins nicht wirklich unterstützt und somit sehr stark auf die Verwendung mit Ableton Live fokussiert ist. Max kostet in der Vollversion mit Educational Discount 250€.

Literatur

Knaub, A. (2020). *Sampling of analog signals—Webdemo*. http://webdemo.inue.uni-stuttgart.de/webdemos/02_lectures/uebertragungstechnik_1/sampling_theorem/index.php?id=4# Zugriff am: 13.12.2020

Wikipedia. (2020a). *JUCE - Wikipedia*. https://en.wikipedia.org/wiki/JUCE Zugriff am: 13.12.2020

Wikipedia. (2020b). *Matlab - Wikipedia*. https://de.wikipedia.org/wiki/Matlab Zugriff am: 13.12.2020

Wikipedia. (2020c). *Virtual Studio Technology - Wikipedia*. https://en.wikipedia.org/wiki/Virtual_Studio_Technology Zugriff am: 13.12.2020