Joachim Heintz

Mitentwicklung freier Audio-Software durch Studierende und ihre Institutionen

Für eine offene Plattform aller Studios zum Austausch von Beispielen

Freie Software für Audio hat sich in den letzten Jahren so entwickelt, dass die wesentlichen Arbeitsbereiche in einem elektronischen Studio mit ihr geleistet werden können. Ich möchte einige Vorteile der Arbeit mit freier Software für Studierende und ihre Studios diskutieren, und einen Nachteil, dem aber durch eine Zusammenarbeit aller Studios in diesem Bereich abgeholfen werden könnte.

Zunächst also zum Begriff freier Software und der Vorteile ihrer Benutzung. Freie oder „open source“ Software ist durch folgende Eigenschaften charakterisiert:

- Sie kann beliebig ausgeführt werden.
- Sie kann beliebig weiterverbreitet werden.
- Sie kann beliebig verändert werden; dazu muss der Quelltext zugänglich sein. Die Veränderungen können dann wiederum weiterverbreitet werden.

Was sind nun die Vorteile der Benutzung freier Software, sowohl für die Studierenden als auch die Studios? Ich sehe Vorteile auf verschiedenen Ebenen.

1. Da ist natürlich zuerst die Kostenfrage. Obwohl es nicht die wesentliche Qualität freier Software ist, umsonst zu sein, ist es natürlich ein wichtiger Faktor, dass jeder Student und jede Studentin sich das Programm legal und kostenlos auf den eigenen Rechner laden kann. Kein Gewürge also mit Demoversionen oder geknackten Zugangscodes. Kein Problem, dass der Freund in Chile oder Korea sich den eigenen Patch auf seinem Rechner anschaut und für seine Zwecke umbaut. Freie Software lässt das nicht nur zu, sie ist genau dazu da.

   Und für die Studios? Auch wenn hier eine Lizenz von einigen Hundert Euro nicht so ins Gewicht fällt, wie für einen einzelnen Studenten: im Geld schwimmen wir wohl alle nicht, und wenn ein Studio einen vierstelligen Betrag im Jahr an Lizenzgebühren sparen kann, kann es mit diesem Geld andere Dinge tun - zum Beispiel die Entwicklung einer freien Software durch eine Spende unterstützen.
2. Freie Software hat naturgemäß offene Protokolle und Schnittstellen. Die Interaktion mit anderen Programmen ist erwünscht; es gibt nirgends die Notwendigkeit, eine Schnittstelle vielleicht aus Gründen des Geschäftsgeheimnisses oder der Konkurrenz zu einem anderen Programm nicht zu unterstützen. Vor allem aber sind diese Protokolle dauerhaft offen. Bei kommerzieller Software weiß man prinzipiell nicht, ob es vielleicht nächstes Jahr eine strategische Entscheidung gibt, mit diesem oder jenem Programm (nicht mehr) zusammen zu arbeiten oder ein bestimmtes Betriebssystem (nicht mehr) zu unterstützen.[1] Nun ist es zwar auch bei freier Software möglich, dass sich grundlegende Strukturen ändern, aber dies geschieht nicht aufgrund von Aufkäufen oder Geschäftsinteressen, und prinzipiell kann jeder hin-gehen und eine eigene Variante schaffen für das, was von dem Hauptzweig der Entwicklung nicht mehr unterstützt wird.
3. Die Entscheidungsstrukturen bei Open-Source-Projekten sind transparent. Sie lassen sich über die Diskussionen in den Developer Mailing Listen nachvollziehen. Auch wenn man mit einer Entscheidung der Entwickler einmal nicht zufrieden ist, liegt die Entscheidungsfindung offen. Das ist ein grundlegendes demokratisches Prinzip, das

uns als demokratischen öffentlichen Institutionen nahe sein sollte.

4. Bei freier Software gilt: Wer mehr macht, hat auch mehr Einfluss. Wenn sich eine Institution entscheidet, sich an einer freien Software maßgeblich zu beteiligen, dann wird sie über verschiedene Kanäle auch deren Ausrichtung mitbestimmen und von der Nutzung der Software selbst profitieren. So hat sich beispielsweise das IEM Graz durch die Förderung von *PD* einen Namen gemacht und mit Johannes Zmölnigs *PD-extended* eine eigene Version entwickelt.
5. Und schließlich der Punkt, um den es mir in diesem Zusammenhang am meisten geht: Jeder Benutzer ist eigentlich Mitentwickler der Software. Der grundlegende Unterschied in der Haltung des Benutzens ist: Bei der kommerziellen Software ist man Kunde und Konsument; bei der freien Software ist jeder aufgefordert mitzuhelfen, dass die Software besser wird, und jeder hat auch die Möglichkeit dazu. Sicher ist es bequemer, Geld auf den Tisch zu legen und dafür ein funktionierendes Produkt zu erwarten, aber besser für die Selbständigkeit und die Entwicklung der Studierenden ist es doch allemal, wenn sie sich einbringen und Verantwortung übernehmen. Das ist bei freier Software eigentlich nötig (sonst macht man sich über sie nur als eine nicht so schicke Billigversion kommerzieller Software lustig), aber es ist auch möglich, und zwar in ganz individueller Weise, je nach Kenntnis und Interesse der Studierenden.

Ich denke also, dass sowohl die Studierenden als auch die Hochschulstudios ein besonderes Interesse an der Nutzung freier Audio-Software haben sollten:

- Freie Software spart Studierenden wie Hochschulen viel Geld und ermöglicht eine freie Weiterverbreitung.
- Sie macht unabhängig von Marktentscheidungen und ermöglicht langfristige Verlässlichkeit in der Nutzbarkeit.
- Sie hat transparente und demokratische Entscheidungsstrukturen.
- Sie ermöglicht den Institutionen eine Mitbestimmung bei der Entwicklung und lässt sie von der Öffentlichkeit einer Community profitieren.
- Sie ermöglicht den Studierenden, aus der Rolle des Kunden zu treten und selbst Verantwortlichkeit zu übernehmen.

Wenn dem nun so ist, dass wir alle – Studierende wie Lehrende – eigentlich ein großes Interesse an einer stärkeren Nutzung von freier Software haben sollten, frage ich mich, warum wir sie nicht mehr nutzen. Ich denke, das hat verschiedene Gründe. Einen davon möchte ich hier diskutieren und einen Vorschlag zu einer möglichen Abhilfe machen. Es geht um das wohl bekannte Problem der Dokumentation, das ich hier an einem konkreten Beispiel schildern möchte:
Letzten Herbst fing einer meiner Studenten an, an einem Stück für Flöte und Live-Elektronik zu arbeiten. Er kam aus Taiwan, hatte schon einen Master in Komposition, aber keinerlei Erfahrung mit Elektronik. Es wurde schnell klar, dass er für sein Stück an verschiedenen Punkten Live-Input aufnehmen und in verschiedenen Transpositionen wiedergeben musste; das vertraute „Record and Play Buffers“ also. Aus meiner Sicht kam für ihn als Umgebung für diesen Anfang nur entweder *Max* oder *PD* infrage. Und das waren unsere Erfahrungen an diesem Punkt:

- Bei *Max* gibt es das alte *MSP* Tutorial Nr. 13: Es versorgt den Studierenden mit einer funktionierenden Grundkonstellation, die er Schritt für Schritt ausprobieren kann und an der er die verschiedenen Objekte und Zusammenhänge kennenlernen kann. Zu dem Patch gibt es die Beschreibung im Tutorial mit vielen Querverweisen. So kann man sofort beginnen und kommt in gutem Sinne vom Hölzchen aufs Stöckchen.

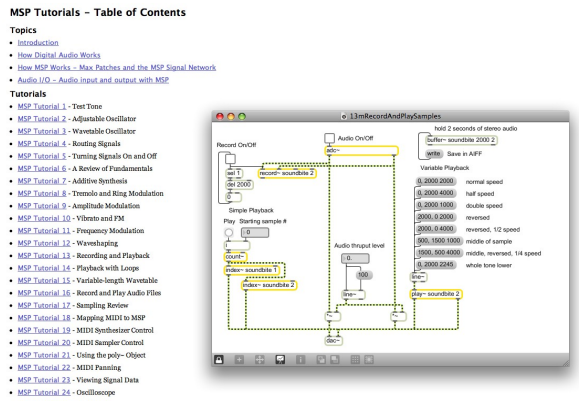

Abb.1: Das Beispiel „Record and Play“ Samples in *MaxMsp*

- Bei *PD* finde ich in der gesamten Standardhilfe kein einziges Beispiel dafür, wie man in einen Buffer aufnimmt. Der Abschnitt B behandelt nur das Abspielen von Soundfiles aus Buffern.

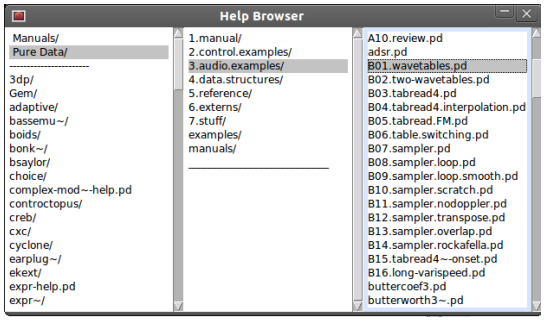

Abb.2: Ausschnitt aus der Hilfe in *PD*

- Das *FLOSS*-Manual zu *PD*[2] hat ebenfalls kein Beispiel zum Aufnehmen in einen Buffer.

Abb.3: FLOSS Manual zu PD

- In Johannes Kreidlers *PD*-Tutorial[3] gibt es zwar ein kurzes Beispiel dafür, wie man grundsätzlich in einen Buffer schreibt, aber insgesamt geht es auch hier mehr um das Thema „Sampler“, und es gibt kein Beispiel, das ähnlich einfach und weiterführend ist wie das *MSP* Tutorial.

Nun sage ich das alles ganz gewiss nicht, um der *PD* Community oder einem der Genannten ans Schienenbein zu treten. Es ist ja ganz logisch, dass kommerzielle Software ein viel größeres Interesse an einer guten Dokumentation und Hilfe hat, weil sie sich sonst nicht verkaufen würde. Ich bezweifele auch nicht, dass es irgendwo einen Patch gibt, der für meinen Studenten gepasst hätte. Der Punkt ist nur, dass diese Beispiele gerade für einen Anfänger zu schwer zugänglich und zu ungleichmäßig dokumentiert sind.

Hier können wir selbst etwas tun. Ich bin sicher, dass es haufenweise gute Beispiele für typische Situationen aus Lehre und Lernen gibt. Was wir brauchen, ist eine Plattform, wo wir diese Beispiele austauschen und gemeinsam verbessern können. Ich denke an folgendes:

- Eine Gliederung typischer Situationen und Tools auf der obersten Ebene, jederzeit erweiterbar, durch die man möglichst schnell findet, wonach man sucht.
- Innerhalb dieser Gliederung dann Beispiele in den verschiedenen Umgebungen, also C*sound*, *PD*, *SuperCollider* und andere. So könnte man auch vergleichen, wie man eine bestimmte Aufgabe in der einen oder anderen Sprache löst.
- Zur Ausführung der Beispiele sollte ggf. eine Beschreibung treten.

Ein erster Vorschlag für die Gliederung[4]:

Example Collection

- Record and Playback
    - Soundfiles
    - Buffers
    - Sampler
- External Control
    - MIDI
    - OSC
    - Serial Interfaces
- Sound Synthesis
    - Additive Synthesis
    - Subtractive Synthesis
    - Amplitude and Ring Modulation
    - Frequency and Phase Modulation
    - Waveshaping
    - Wavetables
    - Granular Synthesis
    - Physical Modelling
- Sound Modification
    - Envelopes
    - Dynamic Processing
    - Filters
    - Reverberation
    - Delay and Feedback
    - AM / RM / Distortion
    - Granular Synthesis
    - Convolution
    - FFT / Spectral Processing
- Spatialization
    - Routing
    - Panning
    - VBAP
    - Ambisonics
- Sequencing
    - Timelines
    - Control structures
- Noise, Random, AC
    - Noise Generators
    - Random Choices
    - Algorithms
- Miscellaneous

Vieles ist hier im Einzelnen zu diskutieren: Soll solch eine Sammlung auch auf Deutsch sein? Oder nur auf Deutsch? Wer hat Zugang? Wie geht man mit Eigentümlichkeiten der jeweiligen Programmiersprachen um und vermeidet eine Verdoppelung mit einem Tutorial?

Ich denke, solch eine Plattform könnte sowohl Studierenden als auch Lehrenden zum Austausch dienen. Im Idealfall hätten Anfänger einen guten Einstieg, fortgeschrittene Studierende könnten ihr Wissen teilen, und die Lehrenden hätten eine zentrale Beispielsammlung zu immer wiederkommenden Problemstellungen.

(Vortrag, 17.06.2011 im Rahmen des Festivals n*ext_generation 4.0 KOMMUNIKATION,* ZKM | Institut für Musik und Akustik, Karlsruhe*.)*

Endnoten

1. Um ein paar Beispiele zu geben: Der Kauf von *Emagic* durch Apple führte 2002 zum Einstellen der Windows-Version von *Logic*. Cycling entschied sich vor kurzem, die *Pluggo* Schnittstelle nicht mehr zu unterstützen, sondern nur noch auf *Max4Live* zu setzen. Und „natürlich“ (?!?) gibt es *Max* nicht für Linux ...
2. www.flossmanuals.net/puredata
3. das ansonsten oft eine große Hilfe beim Lehren und Lernen von *PD* ist, und ein gutes Beispiel, wie eine Institution helfen kann Dokumentationen zu verbessern (es wurde ermöglicht durch ein Stipendium der Musikhochschule Freiburg)
4. Dank an Kilian Schwoon und André Bartetzki für Diskussionen zu diesem Vorschlag.

Zusammenfassung

Joachim Heintz referiert über Vorzüge und Nachteile bei der Verwendung freier Software bei der Arbeit mit Studierenden. Die so genannten *Open Source* Programme sind für den Nutzer kostenfrei erhältlich und bieten ihm die Möglichkeit, sich an deren Weiterentwicklung zu beteiligen.

Autor

Joachim Heintz studierte Komposition in Bremen bei Younghi Pagh-Paan und Günter Steinke und leitet das elektronische Studio Incontri an der Hochschule für Musik, Theater und Medien in Hannover. An der Hochschule für Künste in Bremen ist er tätig als Dozent für Audio-Programmierung.

Titel

Joachim Heintz: *Mitentwicklung Freier Audio-Software durch Studierende und ihre Institutionen. Für eine offene Plattform aller Studios zum Austausch von Beispielen*, in: kunsttexte.de/auditive_perspektiven
Nr. 4, 2011 (5 Seiten), www.kunsttexte.de.