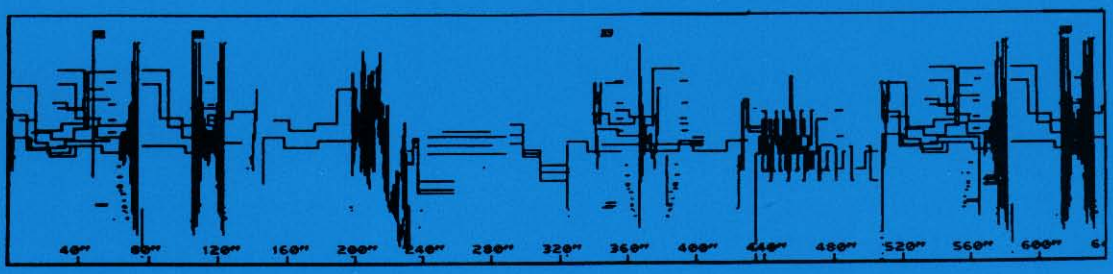
INSTITUT FÜR ELEKTRONISCHE MUSIK
AN DER HOCHSCHULE FÜR MUSIK UND DARSTELLENDE KUNST IN GRAZ

NORBERT SCHNELL

GRAINY - Granularsynthese in Echtzeit

Eine Objekt-Sammlung für MAX auf der ISPW



BEITRÄGE ZUR ELEKTRONISCHEN MUSIK 4

IMPRESSUM

Herausgeber:	Institut für Elektronische Musik (IEM) an der Hochschule für Musik und darstellende Kunst in Graz
Redaktion:	Robert Höldrich, Andreas Weixler
Druck:	Hochschule für Musik und darstellende Kunst in Graz © 1995
Erscheinungsort:	Graz, Österreich

Kontaktadresse:	Institut für Elektronische Musik Jakoministr. 3-5 A - 8010 Graz, Österreich Tel.: ++43/ +316/ 389 - 7010 (Sekretariat) Fax: ++43/ +316/ 389 - 7008 email: hoeldric@wm.mhsg.ac.at

Titelblatt:	Graphik aus der Komposition "Natté" von Helmut Dencker mit freundlicher Genehmigung des Komponisten.

Bisher erschienen folgende "Beiträge zur elektronischen Musik":

BEM **1**	HARALD FRIPERTINGER	**ENUMERATION IN MUSICAL THEORY**	1992
BEM **2**	GREGOR WIDHOLM	**HOLOGRAPHIE, CAD UND MODALANALYSE IM DIENSTE DER MUSIK**	1993
BEM **3**	HELWIG BRUNNER	**DER NACHTIGALLENGESANG IN DER EUROPÄISCHEN KUNSTMUSIK**	1994
BEM **4**	NORBERT SCHNELL	**GRAINY - GRANULARSYNTHESE IN ECHTZEIT**	1995

Die Reihe "Beiträge zur Elektronischen Musik" stellt Arbeiten des Instituts für Elektronische Musik und Akustik zu den Themenbereichen Akustik, Computermusik, Musikelektronik und Medienphilosophie vor. Dabei handelt es sich meist um Ergebnisse von Forschungsarbeiten am Institut oder um überarbeitete Vorträge von InstitutsmitarbeiterInnen.

Darüber hinaus soll hier eine Diskussionsplattform zu den genannten Themen entstehen.

Beiträge können auch eine Beschreibung von Projekten und Ideen sein, die sich noch in Entwicklung befinden und noch nicht fertiggestellt sind.

Beiträge bzw. Anfragen über bisher erschienen Artikel richten Sie bitte an untenstehende Adresse.

Die Redaktion behält sich das Recht vor, Beiträge abzulehnen. Mit der Einsendung erklärt sich der/die AutorIn mit dem Abdruck einverstanden.

Wir hoffen, dass die Schriftreihe "Beiträge zur Elektronische Musik" eine Anregung für Ihre wissenschaftliche und künstlerische Arbeit bietet.

Robert Höldrich
(Redaktion)

The series "Beiträge zur Elektronischen Musik" (contributions to electronic music) presents papers by the Institute of Electronic Music and Acoustics on various topics including acoustics, computer music, music electronics and media philosophy. The contributions present results of research performed at the Institute or edited lectures held by members of the Institute.

The series shall establish a discussion forum for the mentioned fields. Articles should be written in English or German. The contributions can also deal with the description of projects and ideas that are still in preparation and not yet completed.

Submissions and inquiries concerning already published articles should be sent to the adress below.

We hope that the series "Beiträge zur Elektronischen Musik" will provide thought-provoking ideas for your scientific and artistic work.

Robert Höldrich
(editor)

Vorwort des Herausgebers:

Band 4 der "Beiträge der Elektronischen Musik" beschreibt die Ergebnisse eines Projekts über Granularsynthese in Echtzeit. Angeregt durch die Ideen des Komponisten Klaus Hollinetz, der zwischen 1993 und 1995 mehrmals als Gastkünstler am IEM Graz arbeitete, wurde ein Spielinstrument für Echtzeit-Granularsynthese auf der Ircam Signal Processing Workstation unter MAX entwickelt. Die Komposition "Tide", die Hollinetz mit diesem System realisiert hat, wurde im Juni 1995 in Wien uraufgeführt. Ein Vortrag, in dem er seine Arbeitsweise in Graz erläutert hat, wird 1996 im Sammelband zur Ringvorlesung "Die Klangwelt am Rand der Datenautobahn" veröffentlicht. Norbert Schnell hat das Granularsynthese-Instrument in MAX entwickelt und zu einer Objektsammlung erweitert, die den Aufbau verschiedener Granularsynthese-Maschinen erlaubt. Dieser Beitrag erklärt die Architektur von "GRAINY" und verdeutlicht anhand der Beispielanwendung "GRAINY's First" die Arbeitsweise mit dem System.
Der Artikel richtet sich an jene Leser, die sowohl mit den Prinzipien der Granularsynthese als auch mit MAX auf der ISPW vertraut sind. Eine kleine Auswahl einführender Literatur in beide Gebiete ist im Literaturverzeichnis angegeben.

GRAINY ist zusammen mit diesem Artikel via anonymous FTP am IEM verfügbar:

iem.kug.ac.at/pub/ISPW/GRAINY2.0

(Robert Höldrich)

GRAINY – Granularsynthese in Echtzeit

Eine Objekt-Sammlung für MAX auf der ISPW

IEM Graz
Institut für Elektronische Musik
Leiter: O.HProf.Dipl.Ing.Heinz Hoenig

Autor:
Norbert Schnell

Abstract

Der vorliegende Artikel beschreibt die MAX-Objektsammlung *GRAINY*, die am IEM Graz im Rahmen eines Projektes über Granularsynthese entwickelt wurde. GRAINY ermöglicht die einfache Implementation diverser Echtzeitanwendungen von Granularsynthese für die ISPW.

Nach einer kurzen Einleitung stellt der erste Teil des Artikels die allgemeine Architektur einer GRAINY-Anwendung vor. Darauf folgen die detailierte Beschreibung der Beispielanwendung *GRAINY'sFirst* und einige praktische Hinweise zur Arbeit mit GRAINY. Im Anhang finden sich die *demo patches*, die die Verwendung der GRAINY-Objekte in einer Übersicht zeigen, sowie die GRAINY Objektreferenz.

This paper describes the MAX object library *GRAINY*, developped at the IEM Graz in the framework of a granular synthesis project. GRAINY provides easy implementation of various real-time applications of granular synthesis.

After a brief introduction the first section of the paper gives an overview of the generic architecture of a GRAINY application, followed by a detailed description of the example application *GRAINY'sFirst* and a few hints for working with GRAINY. The appendix includes *demo patches*, which show the usage of the GRAINY objects in MAX, as well as the GRAINY object reference.

Inhaltsverzeichnis

1 Einleitung **5**

2 Überblick über GRAINY`s Architektur **7**
2.1 Der Graingenerator . . . 8
2.1.1 Die Quellsignale . . . 9
2.1.2 Die Stimmen . . . 10
2.1.3 Die Ausgangskanäle . . . 12
2.2 Die Synthese der Grainparameter . . . 12
2.2.1 Die Syntheseparameter . . . 12
2.2.2 Zeitstruktur . . . 14
2.2.3 Verwendung der Quellparameter in der Synthese . . . 14
2.3 Die Steuerung . . . 15
2.3.1 Parameterräume . . . 15
2.3.2 Die X-Objekte und die Metaebene . . . 18
2.3.3 Externe Steuerung und das KeyboardDriver-Objekt . . . 18
2.4 Voreinstellungen und Initialisierung . . . 19

3 GRAINY'sFirst, eine Beispielanwendung **20**
3.1 Laden von GRAINY'sFirst . . . 21
3.2 Die Bedienungsoberfläche und die Initialisierung . . . 22
3.2.1 Der patcher interface . . . 22
3.2.2 Der *subpatch* Faders0-15 . . . 23
3.3 Steuerung und Grainparameter-Synthese . . . 24
3.3.1 Der patcher control . . . 24
3.3.2 Der patcher parameters und die PTables . . . 25
3.3.3 Der patcher synthesis . . . 27
3.4 Der Graingenerator . . . 29
3.4.1 Der patcher voices und der patcher sources . . . 29
3.4.2 Der patcher channels und der patcher outs . . . 30

4 Installation und Hinweise **31**
4.1 GRAINY *path* . . . 31
4.2 *help patches* und *demo patches* . . . 32
4.3 MAX table-Objekte . . . 32
4.4 Abtastrate . . . 32

A ANHANG: GRAINY *demonstration patches* **33**
A.1 Graingenerator 33
A.2 Parametereingabe/-anzeige 34
A.3 Steuerung 35

B ANHANG: GRAINY Objektreferenz **36**
B.1 Das SourceAdministrator-Objekt und das Source-Objekt 36
B.2 Das VoiceAllocator-Objekt, das Voice- und Channel-Objekt 38
B.3 Das Parameter-Objekt 41
B.4 Das PFader-Objekt und das PTable-Objekt 42
B.5 Das PSpaceGuide-Objekt und das PSpace-Objekt 43
B.6 Die X-Objekte 45
B.7 Das KeyboardDriver-Objekt 47
B.8 Die Faders-*patches*, das GRAINYinit- und das GRAINYkey-Objekt 48

1 Einleitung

GRAINY ist eine Sammlung von Objekte für MAX[1], mit denen auf einfache Weise Umgebungen für verschiedenste Anwendungen von Echtzeit-Granularsynthese erstellt werden können. Zu einer solchen Umgebung gehören außer den Objekten zur Klangerzeugung Werkzeuge zur Steuerung und Komposition von Klangverläufen sowie Eingabe- und Anzeigeelemente.

GRAINY wurde am IEM Graz entwickelt.

Granularsynthese wird im allgemeinen als eine Syntheseform verstanden, in der viele kurze Ausschnitte (Grains) aus vorgegebenem Klangmaterial zu einem neuen Klang überlagert werden. Es werden Graindichten von mehreren tausend Grains pro Sekunde erreicht. Typische Grainlängen liegen zwischen einer und etwa hundert Millisekunden und mehr. Die Qualität eines einzelnen Grains steht damit an der Grenze zwischem einem eigenständigen musikalischen Ereignis – gleich einer Note – und einer dem Gesamtklang untergeordneten quantitativen Einheit.

In GRAINY findet sich diese Ambivalenz in der Unterscheidung zwischen *Grainparametern* und *Syntheseparametern* wieder. Während die Grainparameter das einzelne Grain beschreiben, wird die Klangentwicklung durch den Verlauf der Syntheseparameter bestimmt. Der Begriff des Grains wird für GRAINY so weit gefaßt, daß es möglich ist, 'Grains' mit einer Länge im Sekundenbereich zu erzeugen. Das kürzeste Grain ist je nach Abtastfrequenz der ISPW zwischen zwei und vier Millisekunden lang.

Anwendungen von Granularsynthese sind meist nicht in Echtzeit implementiert, da die außerordentlich hohe Dichte von Ereignissen (Grains) zuweilen enorme Anforderungen an Rechenleistung und Speicherzugriff stellt. Für Granularsynthese wurde bisher im Gegensatz zu anderen Syntheseformen wie Frequenzmodulation und *physical modelling* kein spezialisierter Synthesizer entwickelt.

Mit der *Ircam Signal Processing Workstation* (ISPW) und der *MAX* Software[2] steht seit einiger Zeit ein System zur Verfügung, das seine Vielseitigkeit mit einer Rechenleistung verbindet, die für die Klangsynthese und -steuerung in Echtzeit neue Möglichkeiten eröffnet.

MAX wird als 'graphische Programmiersprache' bezeichnet. Im Unterschied zu konventionellen, auch objektorientierten Programmierspachen entfällt bei MAX jedoch die Trennung zwischen Programmierung und Anwendung fast gänzlich. Die beiden Domänen sind in MAX nicht durch den Vorgang des Kompilierens getrennt, sondern lediglich durch Umschalten des Editierungsmodus. Der MAX-*patch* ist gleichzeitig Programmcode und Bedienungsoberfläche des Programmes.[3]

[1] GRAINY besteht im wesentlichen aus MAX-*abstracts*, also MAX-*patches*, die wie die MAX-Objekte mit verschiedenen Argumenten instanziert werden können.

[2] ISPW, FTS und MAX ©1992 IRCAM.

[3] Eine Ausnahme bilden hier die *externs*, die es ermöglichen dem MAX System selbst eigene in *C* programmierte Objekte hinzuzufügen. Für GRAINY wurde lediglich ein Objekt als *extern* implementiert.

Um diesem Umstand Rechnung zu tragen, wurde die ursprüngliche Bestrebung, mit dem Projekt GRAINY *eine* bestimmte Maschine für Granularsynthese zu schaffen, dahingehend erweitert, Objekte zur Verfügung zu stellen, die die Implementation und Steuerung verschiedenster Granularsynthese-Algorithmen ermöglichen[4]. (Abb.1)

Die genaue Definition der Algorithmen bleibt so ein Teil der Anwendung und vor allem ein Teil der Komposition mit dem Werkzeug GRAINY. Die ursprünglich geplante Maschine wurde unter dem Namen GRAINY´sFirst als beispielhafte Anwendung dieser Objekte realisiert.

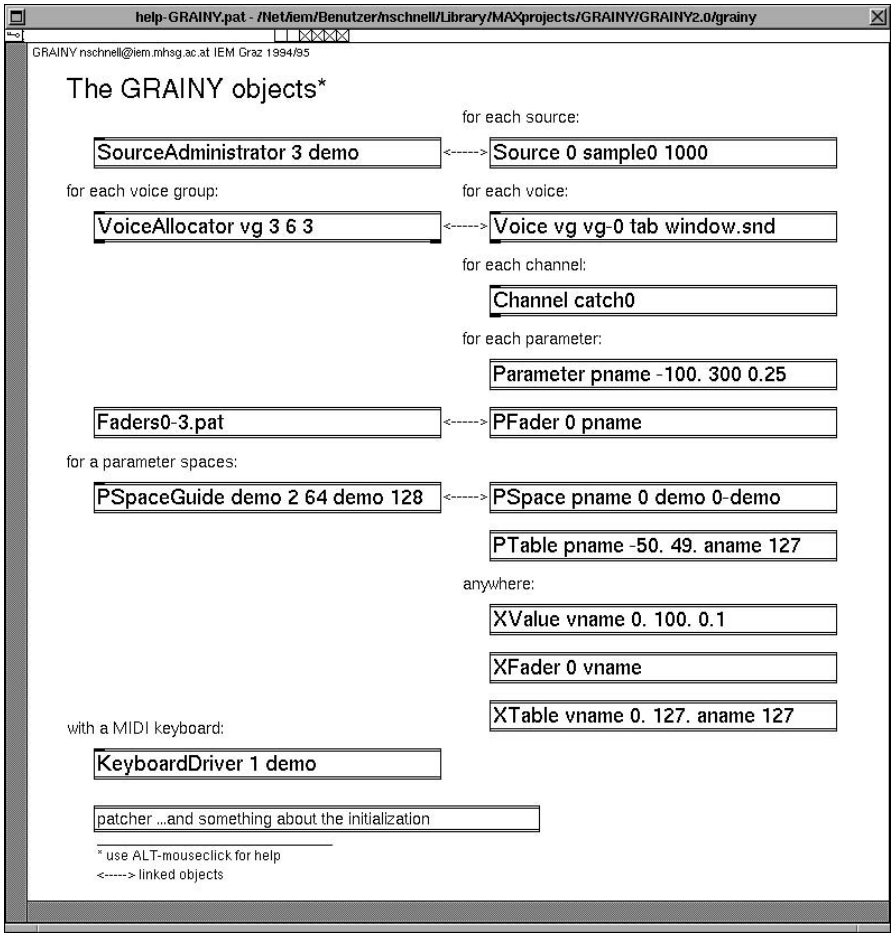


Abbildung 1: Die GRAINY Objekte

[4]Auch für GRAINY´s Objekte gibt es keine strikte Trennung von Systemkomponenten und Applikation. GRAINY versteht sich als Modellvorgabe, deren Erweiterung und Modifikation keine Grenzen gesetzt sind.

2 Überblick über GRAINY`s Architektur

GRAINY verfolgt konsequent den objektorientierten Ansatz der Programmierung in MAX. In den Objekten von GRAINY sind MAX-Objekte integriert, deren Methoden teilweise übernommen, meistens jedoch zu mächtigeren Funktionen zusammengefaßt wurden.

Die vom Anwender zusammengestellten Objekte bilden durch den Austausch von *messages* über vordefinierte MAX *send/receive*-Kanäle eine Struktur, deren Architektur allen GRAINY-Anwendungen gemeinsam ist[5].

GRAINY´s Architektur gliedert sich in drei Ebenen. (Abb.2)

- Die Bestimmung der musikalischen Struktur durch die Steuerung der Syntheseparameter.
- Die Synthese der Grainparameter aus den Syntheseparametern durch die vom Anwender erstellten Synthesealgorithmen.
- Die Generierung der Grains aus den Grainparametern durch den Graingenerator.

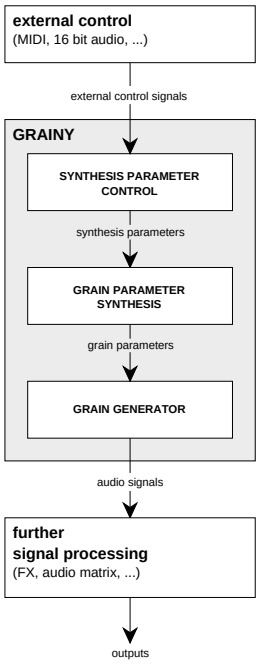


Abbildung 2: Architektur einer GRAINY-Anwendung

[5] Um Seiteneffekte und ungewollte Interaktionen mit anderen MAX *patches* zu vermeiden, ist allen in GRAINY verwendeten Namen das Postfix .grainy angehängt.

2.1 Der Graingenerator

Jedes Grain ist durch sechs Parameter, die Grainparameter, bestimmt.

source	Eine Nummer für das Quellsignal, aus dem es stammt.
position	Seine Anfangsposition relativ zum Anfang des Quellsignals.
length	Seine Länge.
pitch	Ein linearer Tonhöhen- oder auch Geschwindigkeitsfaktor.
gain	Ein linearer Amplitudenfaktor.
channel	Eine Nummer für den Signalkanal, auf dem es ausgegeben wird.

Der Graingenerator setzt einen Strom von Grainparameter-Vektoren (in Form von MAX *list messages*) in einen Strom von Grains (als *signals*) auf seinen Ausgangskanälen um. Die durch die Grainparameter *source*, *position* und *length* spezifizierten Ausschnitte (Grains) aus den Quellsignalen werden in Echtzeit um den Faktor *pitch* transponiert, mit dem *gain*-Faktor verstärkt und auf dem durch *channel* bestimmten Ausgangskanal ausgegeben.

Der Graingenerator gliedert sich in drei Teile, die Source-Objekte mit dem Source-Administrator-Objekt, die Voice-Objekte mit dem VoiceAllocator-Objekt und die Channel-Objekte. (Abb.3)

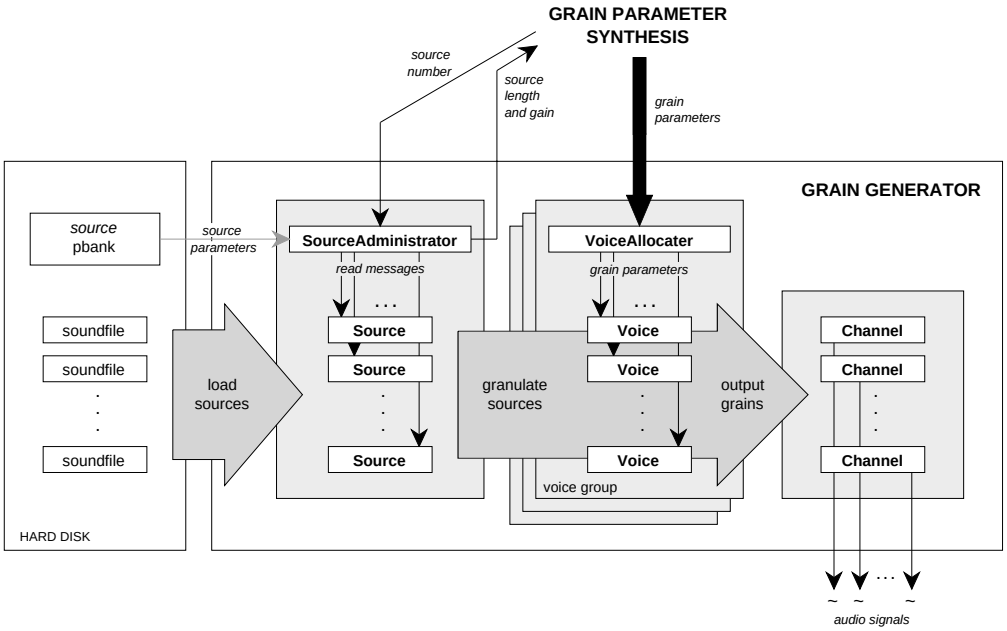


Abbildung 3: Der Graingenerator

2.1.1 Die Quellsignale

Die Quellsignale müssen vor dem Start der Synthese in den Speicher der ISPW geladen werden, da MAX bzw. das Betriebssystem FTS ein dynamisches Laden von Soundfiles während laufendem DSP-Rechenzyklus nicht unterstützt. Für jedes Quellsignal muß ein Source-Objekt zur Verfügung stehen, das einen genügend großen Speicherplatz auf der ISPW reserviert[6].

Jedes Quellsignal ist in einer Zeile der Source-pbank durch vier Parameter, die Quellparameter, spezifiziert.

filename	Dateiname und Pfad eines Soundfiles.
offset	Die Anfangsposition des Quellsignals relativ zum Anfang des Soundfiles (in ms).
srclength	Die genaue Länge des Quellsignales (in ms).
srcgain	Ein Amplitudenfaktor zum Angleichen des Pegels mehrerer Quellsignale.

Das SourceAdministrator-Objekt liest die Quellparameter aus der Source-pbank ein und sendet eine read-*message* mit dem Dateinamen des Soundfiles und der Anfangsposition des Quellsignales an jedes Source-Objekt(Abb.4). Die Source-Objekte laden daraufhin die Quellsignale in den von ihnen reservierten Speicher (Abb.5). Die beiden Quellparameter für Länge und Amplitude können in der Grainparameter-Synthese für jedes Quellsignal aus dem SourceAdministrator-Objekt abgerufen werden (siehe unten).

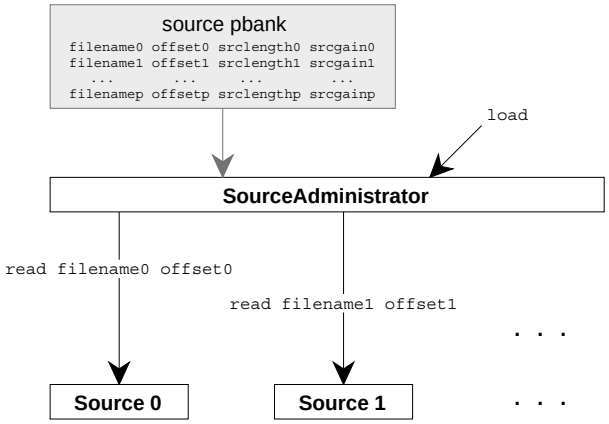


Abbildung 4: Laden der Source-pbank

[6] Das Source-Objekt besteht im wesentlichen aus einem MAX table~-Objekt. Der Speicherplatz wird stets in Millisekunden angegeben.

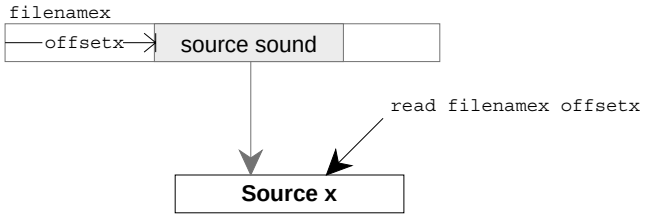


Abbildung 5: Laden eines Quellsignales aus einem Soundfile

Alle Stimmen teilen sich die gleichen Quellsignale, deren Anzahl prinzipiell nicht beschränkt ist. Die Gesamtlänge der Quellsignale muß in den Speicher der ISPW passen[7].

2.1.2 Die Stimmen

Den innersten Kern des Graingenerators bilden die Voice-Objekte, GRAINY´s Stimmen. Empfängt ein Voice-Objekt einen Vektor von Grainparametern (in Form einer MAX *list message*), spielt es daraufhin ein einzelnes, dadurch spezifiziertes Grain ab.

Mehrere Voice-Objekte können zu einer Stimmgruppe zusammengefaßt werden, die von einem VoiceAllocator-Objekt verwaltet wird. (Abb.6)

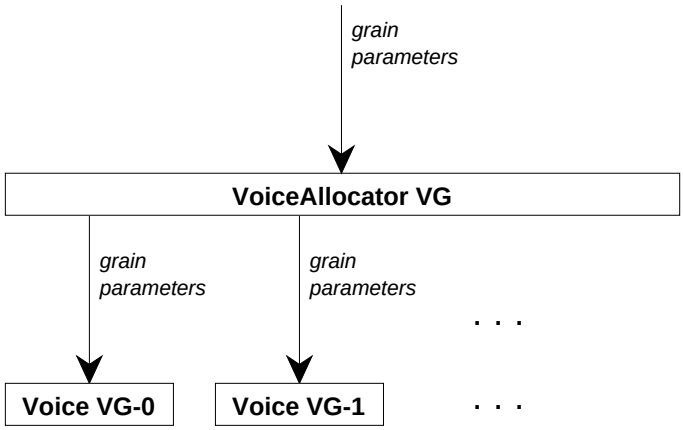


Abbildung 6: Stimmgruppe (VG) mit VoiceAllocator und Voice-Objekten

[7] Genauer ist es der freie Speicherplatz, der jener ISPW CPU zugeordnet ist, auf der auch die GRAINY Voice-Objekte laufen.

Das Voice-Objekt ist in mehreren Varianten implementiert. Die Implementationen unterscheiden sich durch die Anzahl und Reihenfolge der Grainparameter, die das Voice-Objekt als Vektor empfängt, und die Art der Einhüllenden, mit der jedes Grain versehen wird. Während die Voice-Objekte einer Stimmgruppe von der gleichen Art sein sollten, können sie sich von Gruppe zu Gruppe unterscheiden. Die Variante des Voice-Objektes wird bei seiner Instanzierung angegeben.[8]

Manche Granularsynthese-Anwendungen kommen über weite Strecken mit weniger als den sechs oben vorgestellten Grainparametern aus. Für diese Anwendungen bedeutet es einen unnötigen Mehraufwand, stets eine *list message* mit sechs Argumenten an den Graingenerator zu senden. Deshalb können einige Versionen des Voice-Objektes einen kürzeren Vektor von Grainparametern benutzen. Die Grainparameter, die nicht innerhalb des Vektors an das Voice-Objekt gesendet werden, können teilweise über einen eigenen *message*-Kanal verändert werden[9].

Die von einem Voice-Objekt erzeugte Einhüllende, das *grain window*, kann entweder relativ zur Länge des Grains oder mit einer festgelegten Anstiegs- und Abfallszeit definiert sein. Für Voice-Objekte mit einer absolut definierten Einhüllenden können Anstiegs- und Abfallzeit über *message*-Kanäle verändert werden. Es sollten in diesem Fall keine Grains erzeugt werden, die kürzer als die Summe dieser beiden Zeiten sind.

Für den Graingenerator stellt jedes Grain einen eigenen Prozeß dar, der – einmal durch die Grainparametern initiiert – unabhängig von allen anderen Grains und jeder weiteren äußeren Beeinflussung abläuft und für die Zeit seiner Länge eine Stimme belegt.

Das VoiceAllocator-Objekt, als erste Stufe des Graingenerators, empfängt einen mehr oder weniger kontinuierlichen Strom von Grainparameter-Vektoren und weist sie jeweils auf freie Voice-Objekte zu. Wenn alle Stimmen belegt sind, werden die ankommenden Vektoren ignoriert. Die Länge der Grainparameter-Vektoren ist ein Argument bei der Instanzierung des VoiceAllocator-Objektes. Empfängt das VoiceAllocator-Objekt einen kürzeren Vektor als hier angegeben, werden die nicht spezifizierten Werte der weiter hinten stehenden Grainparameter vom vorigen Grain übernommen.

Ein VoiceAllocator-Objekt verarbeitet nur ein Grain pro DSP-Rechenzyklus[10]. Damit ist die maximale zeitliche Graindichte einer Stimmgruppe begrenzt. Ferner ist mit dem Zyklus ein Raster vorgegeben, auf dem die Anfänge und Enden aller Grains liegen. Die Grainlängen und Triggerperioden werden dennoch genau in Millisekunden angegeben. Die tatsächlichen Grainlängen und Grainanfänge werden auf das nächstgelegene Vielfache der Zykluszeit gerundet, sodaß z.B. eine konstante Triggerperiode im Mittel erreicht wird. (siehe 4.4 S.32)

Prinzipiell können beliebig viele Voice-Objekte in einer GRAINY-Anwendung verwendet werden. Ihre maximale Anzahl ist jedoch in der Praxis durch die Rechenleistung der ISPW beschränkt.

[8] Die einzelnen Varianten des Voice-Objektes sind im MAX-*patch* help-Voice.pat erklärt.

[9] Prinzipiell sollten diese Grainparameter während der Synthese konstant bleiben, da ein neuer Wert gleichzeitig an alle Voice-Objekte gesendet wird, was einen hohen Rechenaufwand bedeutet. Der *message*-Kanal trägt den Namen der Stimmgruppe gefolgt von dem Namen des Grainparameters und dem Postfix VOX.grainy (z.B.: VGlengthGP.grainy).

[10] Dieser Zyklus hat die Länge von 64 Samples und variiert mit der Abtastrate der ISPW. Für eine Abtastrate von 44.1 kHz liegt die DSP-Rechenzykluszeit bei 1.45 ms, für 32 kHz bei genau 2 ms.

2.1.3 Die Ausgangskanäle

Die letzte Stufe des Graingenerators bilden die Ausgangskanäle. Jedes einzelne Grain kann über den *channel*-Parameter einem von beliebig vielen dieser *signal*-Kanäle zugemischt werden. (Abb.7)

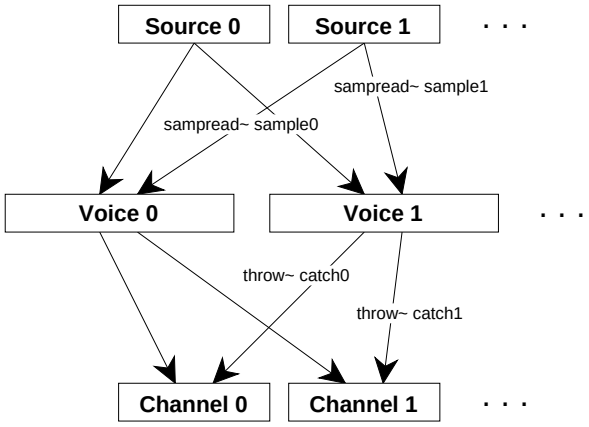


Abbildung 7: Signaldatenfluß im Graingenerator

Für jeden Ausgangskanal muß ein Channel-Objekt[11] instanziert werden, das den Strom der für diesen Kanal bestimmten Grains ausgibt. Natürlich muß ein GRAINY-Ausgangskanal nicht unbedingt auch einen Audioausgang der ISPW darstellen. Wenn es die Rechenleistung der ISPW erlaubt, kann der GRAINY-Anwendung auch eine weitere Signalverarbeitung in MAX folgen.

2.2 Die Synthese der Grainparameter

Die wichtigste Ebene einer GRAINY-Anwendung ist die Synthese der Grainparameter. Hier wird der Strom von Grainparameter-Vektoren erzeugt, den der Graingenerator in die Ströme von Grainsignalen auf den Ausgangskanälen umsetzt.

2.2.1 Die Syntheseparameter

Der Anwender definiert zunächst die Syntheseparameter, über die die Granularsynthese gesteuert werden soll. Dann erstellt er jene 'Matrix' von Algorithmen, die in Echtzeit aus den Syntheseparametern die Werte und die zeitliche Struktur der Grainparameter erzeugt.

Für jeden Syntheseparameter wird ein Parameter-Objekt verwendet, das den Wert des Parameters hält und eine bidirektionale Verbindung zwischen den Synthesealgorithmen und der Steuerumgebung herstellt. (Abb.8)

[11] Das Channel-Objekt besteht lediglich aus einem MAX catch~-Objekt.

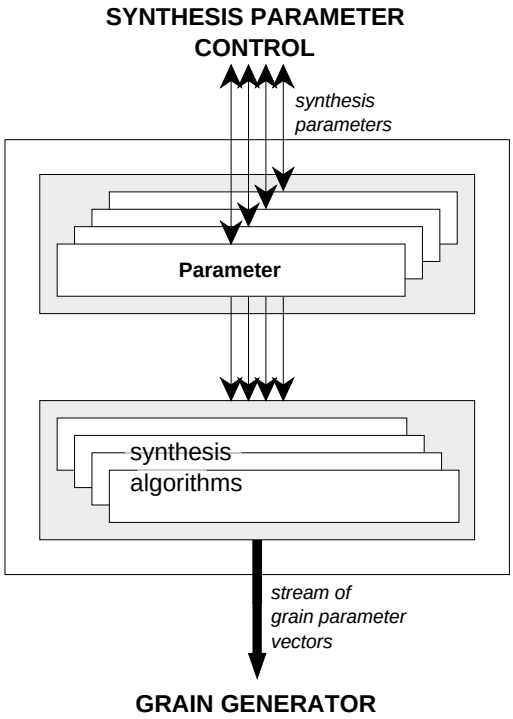


Abbildung 8: Parameter-Objekt und Grainparameter-Synthesealgorithmen

Im einfachsten Fall sind die Syntheseparameter mit den Grainparametern identisch[12]. In der Beispielanwendung GRAINY'sFIRST sind geradeaus vierzehn Syntheseparameter definiert, die im wesentlichen jeweils Mittelwert und Abweichung für die Werte der sechs Grainparameter sowie der Periodendauer eines Triggergenerators angeben. Defininiert man für eine GRAINY-Anwendung komplexere Syntheseparameter[13], ist zu erwarten, daß man mit weniger Parametern auskommt, jedoch die Synthese der Grainparameter kompliziertere Algorithmen erfordert.

Da die Algorithmen zur Grainparameter-Synthese auf der Basis von MAX *messages* implementiert werden, stehen dem Anwender von GRAINY hier eine Vielzahl von *built-in* MAX-Objekten zur Verfügung.

Die Ausgänge der Parameter-Objekte an die Steuerumgebung können durch Senden von 0 und 1 auf dem *message*-Kanal switchPOUTS.grainy deaktiviert und aktiviert werden. Für die Arbeit mit den einzelnen Sytheseparametern sollte die Verbindung zur Steuerumgebung aktiviert sein, damit hier immer die aktuellen Werte vorliegen und angezeigt werden können. In Anwendungen, die kritisch bezüglich der Rechenleistung sind,

[12] Die Zeitstruktur der Grains bleibt hier allerdings unberücksichtigt.

[13] Man denke an Begriffe wie *Dichte*, *Rauigkeit* oder *Aggressivität* und an Syntheseparameter, die sich bereits auf ein konkretes Quellsignal beziehen.

ist es jedoch sinnvoll, diese Verbindungen zu deaktivieren. Bei weitgehend selbstständig ablaufenden komplexeren Steuerungen der Syntheseparameter von außen oder durch die Steuerumgebung kann so der Aufwand für die Kommunikation der GRAINY-Objekte untereinander und die graphische Darstellung minimiert werden. Die Verbindungen zur Grainparametersynthese sind hiervon unabhängig und stehts aktiv.

2.2.2 Zeitstruktur

Während die Syntheseparameter in der Regel unabhängig voneinander und asynchron gesteuert werden können, müssen die Grainparameter für jedes Grain synchron an den Graingenerator übergeben werden. Die zeitliche Struktur dieser Vektoren entspricht – sofern genügend Stimmen zur Verfügung stehen – genau der zeitlichen Struktur der Grains auf den Ausgangskanälen des Graingenerators.

Die Grainparameter werden zu bestimmten (Trigger-)Zeitpunkten entweder aus den momentanen Werten der Syntheseparameter quasi synchron erzeugt oder aus den Werteverläufen unabhängiger Prozesse abgetastet. In jedem Fall muß ein die Zeitstruktur der Grains bestimmendes Trigger-Signal generiert werden.

2.2.3 Verwendung der Quellparameter in der Synthese

Die zusammen in einer GRAINY-Anwendung verwendeten Quellsignale können verschieden lang und verschieden ausgesteuert sein. Um mit einer solchen inhomogenen Zusammenstellung von Quellsignalen umzugehen, ist es unter Umständen sinnvoll die Grainparameter *position* und *gain* zunächst für alle Quellsignale als relative Werte (in %) zu erzeugen. Mit Hilfe der aus dem SourceAdministrator-Objekt abrufbaren Quellparameter *srclength* und *srcgain* können diese relativen Werte für ein bestimmtes Quellsignal in die absoluten Werte der Grainparameter umgerechnet werden, bevor sie an den Graingenerator übergeben werden. (Abb.9)

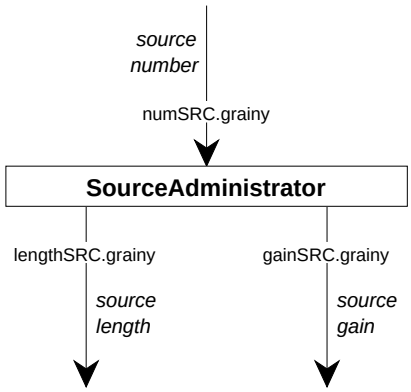


Abbildung 9: Abrufen der Quellparameter aus dem SourceAdministrator-Objekt

2.3 Die Steuerung

Die oberste Ebene der GRAINY-Architektur bilden Objekte, die die Steuerung der Syntheseparameter unterstützen (Abb.10). Zunächst wird jeder Syntheseparameter über ein PFader-Objekt einer Anzeige/Eingabe-Einheit zugewiesen, wo sein aktueller Wert angezeigt und mit Hilfe der Maus verändert werden kann.

2.3.1 Parameterräume

Die n Syntheseparameter spannen einen n-dimensionalen Raum auf, wobei jeder Punkt des Raumes eine Konstellation der Parameterwerte und somit einen mehr oder weniger stationären Klang repräsentiert. Ein bestimmter Weg durch den Parameterraum bedeutet somit eine bestimmte Klangentwicklung.

Jeder Syntheseparameter kann einer Koordinate eines (Unter-)Raumes zugeordnet werden. In GRAINY gibt es zwei Möglichkeiten, solche Parameterräume zu definieren und Klangentwicklungen zu editieren bzw. abzurufen.

- PTables
- PSpaces

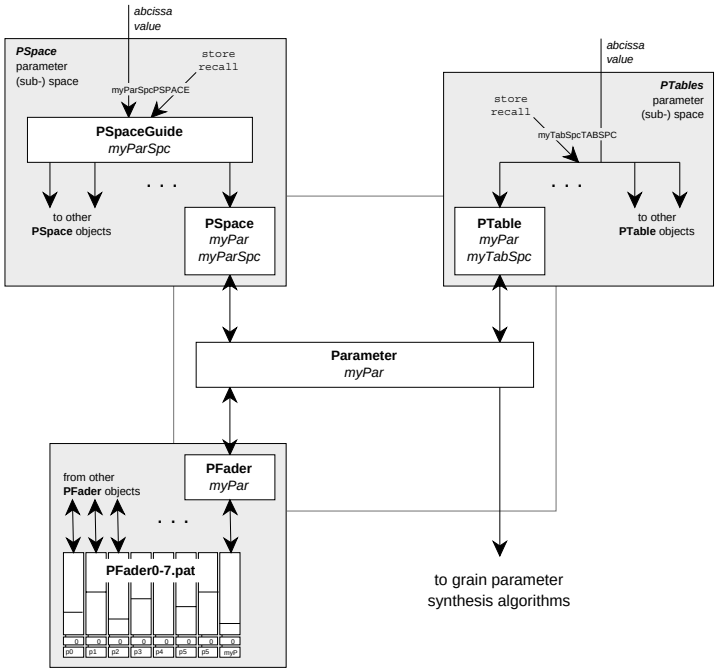


Abbildung 10: Steuerung eines Syntheseparameters über ein PFader-, PSpace- und PTable-Objekt

Die PTables (Abb.11) basieren im wesentlichen auf MAX table-Objekten und ihren Möglichkeiten der graphischen Editierung von Werteverläufen. Für jeden Parameter wird ein PTable-Objekt mit dem Namen des Unterraumes erzeugt. Der Verlauf des Parameterwertes kann in den Table des Objektes gezeichnet werden. Ebenso können die momentanen Werte aller Parameter eines Unterraumes an einer bestimmten Stelle ihrer jeweiligen Tables gespeichet werden, um so z.B. Markierungen für einen dann zu zeichnenden Verlauf zu setzen. Der Weg durch den Parameterraum wird mit einem gemeinsamen Wert für die Abszisse aller PTable-Objekte eines Unterraumens durchfahren. Zwischen zwei Punkten in einem Table wird linear interpoliert.

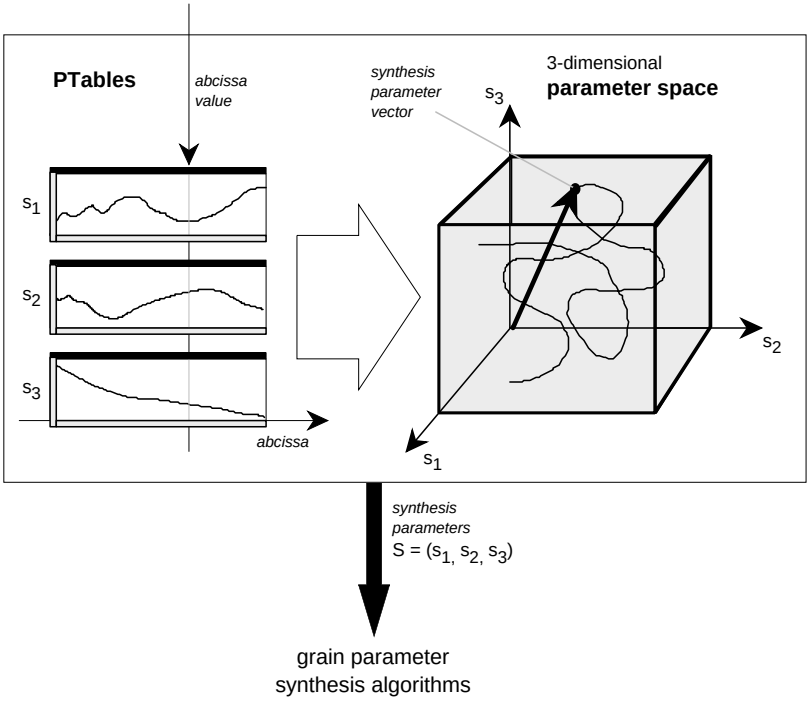


Abbildung 11: 3-dimensionaler PTable-Parameterraum

Während die Methode der PTables eher die Steuerung des Verlaufes einzelner Klangeigenschaften unterstützt, steht bei den PSpaces (Abb.12) der Übergang zwischen Klangzuständen im Vordergrund. Hierzu werden Vektoren von Syntheseparametern eines Parameterraumes in einer MAX pbank festgehalten. Jede Zeile dieser PSpace-pbank repräsentiert so einen Klangzustand, der mit der Zeilennummer assoziert ist. Zumeist wird ein Klang händisch eingestellt und dann sein Parametervektor mit der *message* store abgespeichert. In einer weiteren (einzeiligen) pbank, der Index-pbank, können die in der PSpace-pbank festgelegten Klänge als Folge von Zeilennummern in ihrer Aufeinanderfolge und Wiederkehr arrangiert werden.

Für einen PSpace-Parameterraum muß ein PSpaceGuide-Objekt instanziert werden, das seine PSpace- und Index-pbank enthält. Jeder Syntheseparameter wird durch ein PSpace-Objekt einer Koordinate des Unterraumes (einer Spalte der PSpace-pbank) zugeordnet.

Empfängt der PSpaceGuide eine fraktionale Zahl, weist deren ganzzahliger Teil auf einen Eintrag in der Index-pbank. Der fraktionale Teil bestimmt den Übergang von diesem Klang zum nachfolgenden durch lineare Interpolation.

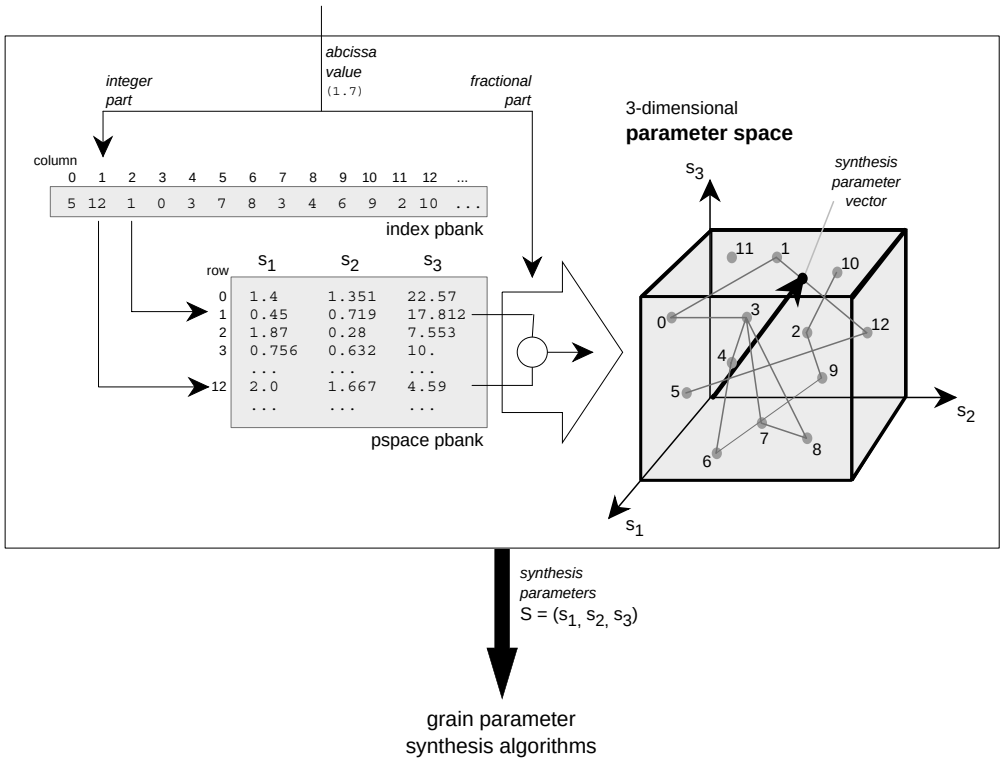


Abbildung 12: 3-dimensionaler PSpace-Parameterraum

2.3.2 Die X-Objekte und die Metaebene

Analog den *P*-Objekten (Parameter-, PFader- und PTable-Objekt) gibt es in GRAINY die *X*-Objekte.

- XValue
- XFader
- XTable

Die X-Objekte funktionieren im wesentlichen wie ihre P-Vorbilder. Während die P-Objekte jedoch fest mit den Syntheseparametern verbunden sind, dienen die X-Objekte zur Speicherung und Steuerung beliebiger Parameter einer GRAINY-Anwendung.

So ist es beispielsweise möglich, auf höherer Ebene mit einem einzigen X-Parameter den Ablauf des Weges durch einen Parameter(unter)raum über der stetig ablaufenden Zeit graphisch zu editieren.

2.3.3 Externe Steuerung und das KeyboardDriver-Objekt

Die externe Steuerung der GRAINY-Anwendung ist am einfachsten über MIDI möglich. MAX stellt eine Reihe von Objekten zur Verfügung, die den Empfang und die Verwaltung von MIDI-Messages unterstützt. Die meisten Quellen von MIDI-Steuerdaten arbeiten allerdings mit einer Wortbreite von 7 Bit, was für viele Syntheseparameter nicht ausreicht.

Um eine universelle Quelle von Steuersignalen zu schaffen, wurde in GRAINY das KeyboardDriver-Objekt integriert. Dieses Objekt erlaubt die Eingabe von Werten in einem beliebigen Bereich und mit beliebiger Auflösung mit Hilfe einer gewöhnlichen MIDI-Tastatur auf der Basis von Inkrementieren und Dekrementieren. Der KeyboardDriver kommuniziert direkt mit Parameter- und XValue-Objekten. Für jede MIDI-Note kann ein solches Zielobjekt definiert werden, dessen Wert verändert wird. Wird eine Taste der MIDI-Tastatur gedrückt, erscheint in einem Fenster am Bildschirm ein Kommentar zu dieser Taste (z.B. der Name des Syntheseparameters). Der Wert kann nun mit dem *modulation wheel* über seinen ganzen Bereich grob eingestellt und mit dem *pitch bender* in verschiedenen Stufen inkrementiert und dekrementiert werden. Mit dem *after touch* kann die Geschwindigkeit des In- und Dekrementierens erhöht werden.

Die Einstellungen für den KeyboardDriver werden in einer pbank mit dem Postfix .keyboard gespeichert. Die pbank enthält für jede MIDI-Note eine Zeile mit drei Spalten.

1. Die MIDI-Notennummer (gleichzeitig die Zeilennummer: 0...127), die lediglich der Orientierung in der pbank dient, falls man sie mit einem Texteditor editiert.
2. Der Name des *message*-Kanals auf dem die Daten gesendet werden sollen (z.B. mynamePIN für ein Parameter-Objekt oder mynameIN für ein XValue-Objekt).
3. Der Kommentar, der für die entsprechende Taste erscheinen soll. (Dieser Kommentar muß zu einem MAX *symbol* zusammengefaßt sein. Mehrere Worte müssen deshalb z.B. durch einen Punkt statt durch ein Leerzeichen getrennt werden, da sonst jedes Wort ein eigenes *symbol* darstellt.)

Eine MIDI-Note, der kein Zielobjekt zugeordnet ist, bekommt den *message*-Kanal NULL.

2.4 Voreinstellungen und Initialisierung

Beim Öffnen einer GRAINY-Applikation lesen der SourceAdministrator, die PSpaceGuide-Objekte und der KeyboardDriver ihre bei der Instanzierung voreingestellten pbank-Daten ein.[14] Existiert kein entsprechendes File, gibt es eine Fehlermeldung des jeweiligen pbank-Objektes.

Durch Senden einer bang *message* auf dem Kanal INIT.grainy können einige GRAINY Objekte initialisiert werden.

Der Anwender kann mit dem GRAINYinit-Objekt an jeder Stelle seiner Anwendung weitere Initialisierungswerte anbringen. Das Objekt wird mit einer *message* (*symbol* Zahl) als Argument instanziert, die es bei der Initialisierung mit INIT.grainy ausgibt.

Folgende Intitialisierungen werden über INIT.grainy vorgenommen.

- Der SourceAdministrator liest die aktuelle Source-pbank ein, und die Source-Objekte laden die darin spezifizierten Quellsignale. Voreinstellung ist die bei der Instanzierung des SourceAdministrator angegebene pbank.
- Der KeyboardDriver liest die zuletzt gelesene bzw. die voreingestellte pbank mit der MIDI-Tastaturbelegung ein.
- Die GRAINYinit-Objekte senden ihren Initialisierungswert aus.

[14] Die Quellsignale werden nicht in die Source-Objekte geladen, bevor über INIT.grainy eine Initialisierung der Anwendung stattgefunden hat oder eine entsprechende *message* an den SourceAdministrator gesendet wurde.

3 GRAINY'sFirst, eine Beispielanwendung

GRAINY'sFirst ist eine grundlegende Anwendung von Granularsynthese mit GRAINY. Die Syntheseparameter in GRAINY'sFirst sind in im wesentlichen mit den Grainparametern identisch. Die verwendeten Synthesealgorithmen basieren auf Pseudo-Zufallsprozessen. Die zeitliche Struktur wird durch einen Triggergenerator bestimmt.

Folgende vierzehn Syntheseparameter sind definiert:

Trigger	Die mittlere Periode des Triggergenerators (2...10000 ms).
TriggerVar	Die Variation der Periode des Triggergenerators (± 0...100 %).
Source	Der virtuelle Mittelpunkt der Verteilung der Grainsabstammung aus den Quellsignalen. (siehe 3.3.3 S.28)
SourceDiff	Die Unschärfe der Abstammung aus den Quellsignalen.
Position	Die mittlere Position des Grains im Quellsignal (0...100 % der Länge des Quellsignales).
PositionVar	Die Variation der Position (± 0...10000 ms).
Length	Die mittlere Grainlänge (2...10000 ms).
LengthVar	Die Variation der Grainlänge (± 0...100%).
Pitch	Die mittlere Tonhöhe (−7200...2400 cent Abweichung vom Orginalsignal).
PitchVar	Die Variation der Tonhöhe (± 0...2400 cent).
Gain	Der mittlere Pegel (Abweichung vom Quellsignal −80...+ 40 dB).
GainVar	Die Variation des Pegels (± 0...60 dB).
Channel	Der virtuelle Mittelpunkt der Grainverteilung auf die Ausgangskanäle. (siehe 3.3.3 S.28)
ChannelDiff	Die Unschärfe der Verteilung auf die Ausgangskanäle.

Jeder Syntheseparameter ist gleichzeitig einem PSpace- und einem PTable-Parameterraum zugeordnet. Die Steuerwerte für die Parameterräume sind durch XValue-Objekte repräsentiert.

PTablePointer	Zeiger (*float*) auf die Abszisse, um die Werteverläufe der Syntheseparameter aus den PTables abzurufen.
PSpacePointer	Zeiger (*float*) auf die Index-pbank, um die Werteverläufe der Syntheseparameter aus der PSpace-pbank abzurufen.

Der Graingenerator von GRAINY'sFirst arbeitet in der vorgegebenen Zusammenstellung mit drei Quellsignalen, einer Stimmgruppe mit sechzehn Stimmen und vier Ausgangskanälen, die direkt mit den Audioausgängen der ISPW verbunden sind.

3.1 Laden von GRAINY'sFirst

Um GRAINY'sFirst aufzurufen, wird der MAX *patch* GRAINY'sFirst.pat geöffnet. Daraufhin öffnen sich zwei Fenster, GRAINY'sFirst.pat selbst und der patcher interface mit einigen Bedienungselementen.

Der *patch* GRAINY'sFirst.pat enthält lediglich die patcher-Objekte mit den zusammengefaßten Objektgruppen und ein MAX declare-Objekt. (Abb.13)

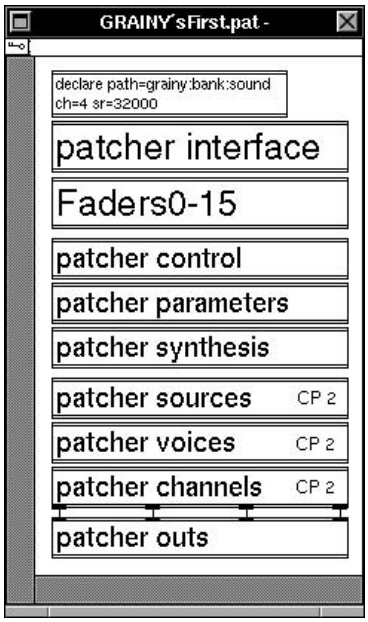


Abbildung 13: Das oberste Fenter von GRAINY'sFirst.

Jeder MAX patcher kann einer ISPW CPU zugeordnet werden, auf welche beim Öffnen der Anwendung alle enthaltenen Objekte geladen werden. Die Objekte von GRAINY'sFirst verteilen sich auf beide Prozessoren der ISPW. Für die Signalverarbeitung müssen die Voice-Objekte und die zugehörgen Source- und Channel-Objekte auf dem gleichen Prozessor liegen (hier CP2). Um diesen Prozessor optimal auszunutzen, d.h. möglichst viele Voice-Objekte unterzubringen, liegt der VoiceAllocator zusammen mit der Steuerung und der Grainparameter-Synthese auf dem anderen Prozessor (CP1). Die dac~-Objekte für die Signalausgänge müssen stets auf dem ersten Prozessor (CP1) laufen, da nur dieser direkt mit den Ein- und Ausgängen der ISPW verbunden ist. Zwischen den beiden Prozessoren der ISPW werden also in die eine Richtung die Grainparameter-Vektoren an die Stimmen, in die andere die Ausgangssignale an die Audioausgänge gesendet.

Das declare-Objekt setzt die Abtastrate der ISPW auf 32 kHz und stellt vier Audioausgänge ein, sofern die ISPW mit der *SoundSmith*-Erweiterung betrieben wird. Ferner werden die relativen Pfade grainy, bank und sound gesetzt, in denen MAX nach den GRAINY Objekten, den pbank-Dateien und Soundfiles sucht.

3.2 Die Bedienungsoberfläche und die Initialisierung

Der patcher interface und der *subpatch* Faders0-15 bilden die Bedienungsoberfläche von GRAINY'sFirst.

3.2.1 Der patcher interface

Der patcher interface erscheint beim Öffnen von GRAINY'sFirst.pat automatisch am Bildschirm. Hier sind alle Bedienungselemente außer den Anzeige-/Eingabeelementen für die Syntheseparameter zusammengefaßt. (Abb.14)

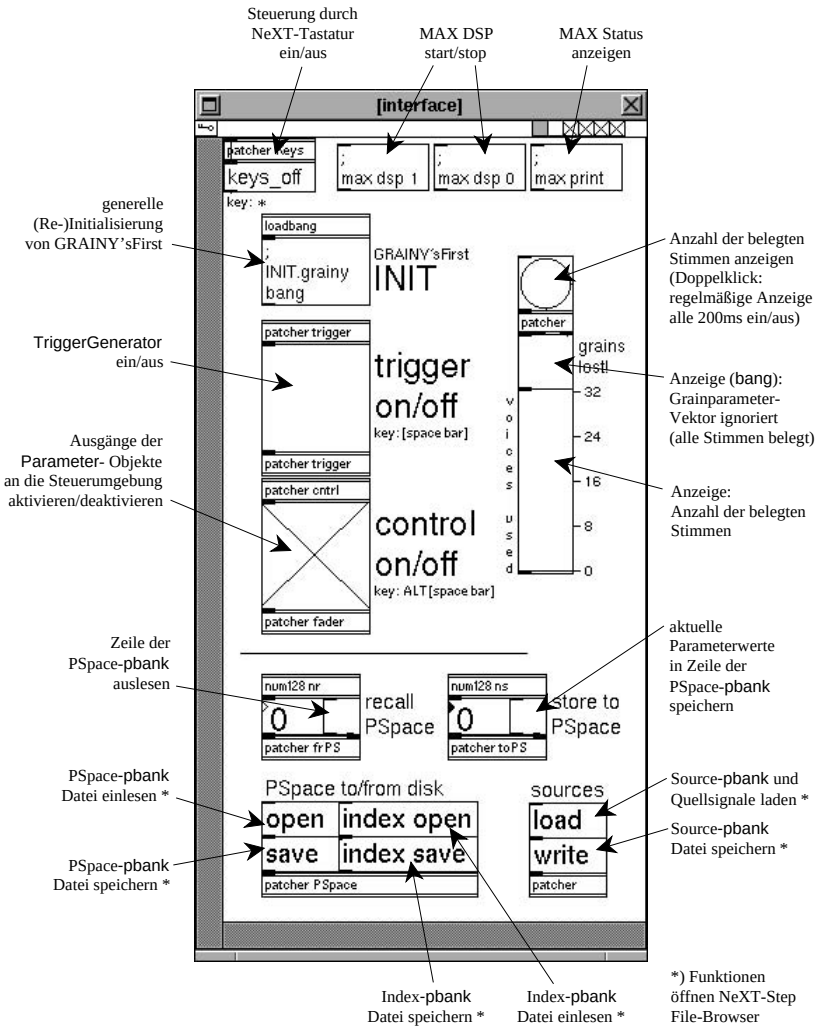


Abbildung 14: Der patcher interface

Am oberen Rand im patcher interface befinden sich die *messages* an MAX, um die Signalverarbeitung an- und auszuschalten und den aktuellen MAX Status anzuzeigen (dsp 1, dsp 0 und print). Mit dem keys on/key off Schalter wird die Steuerung einiger Funktionen durch Tasten der NeXT-Tastatur an- und ausgeschaltet.

Die bang *message* auf dem Kanal INIT.grainy initialisiert die GRAINY-Objekte und die Bedienungsoberfläche von GRAINY'sFirst:

- Der KeybordDriver liest die Datei der aktuellen pbank mit der MIDI-Tastaturbelegung ein.
- Der SourceAdministrator liest die Datei der aktuellen Source-pbank ein. Die Source-Objekte laden die darin spezifizierten Quellsignale.
- Der trigger on/off Schalter wird auf off gesetzt und der Triggergenerator damit ausgeschaltet.
- Der control on/off Schalter wird auf on gesetzt, sodaß die Ausgänge der Parameter-Objekte mit der Steuerumgebung verbunden sind und die Werte der Syntheseparameter an die PFader-, PTable- und PSpace-Objekte gesendet werden.
- Es wird die Anzeige aktiviert, die alle 200 ms die Anzahl der durch den VoiceAllocator belegten Stimmen ausgibt. Durch Doppelklicken kann die Anzeige aus- und eingeschaltet werden. Ein einfacher Mausklick fragt einmalig die Stimmenbelegung ab.
- Die Werte der Syntheseparameter in Zeile 0 der aktuellen PSpace-pbank werden abgerufen.
- Die PFader- und XFader-Objekte senden ihren Namen an die ihnen zugeordnete Anzeige-/Eingabeeinheit in Faders0-15. Der Name erscheint dort als Beschriftung.

3.2.2 Der *subpatch* Faders0-15

Der *subpatch* Faders0-15 enthält die sechzehn Anzeige-/Eingabeeinheiten, die den vierzehn Syntheseparametern, dem *PSpacePointer* und dem *PTablePointer* zugeordnet sind. Die Werte können hier entweder mit dem fader-Objekt verändert oder direkt numerisch eingegeben werden. Das fader-Objekt und die *number box* sind über das Parameter-Objekt miteinander verbunden. Sofern die Steuerausgänge der Parameter-Objekte aktiviert sind, wird eine numerische Eingabe durch den fader angezeigt und umgekehrt. Sind die Ausgänge der Parameter-Objekte deaktiviert, haben Veränderungen der Elemente immer noch Einfluß auf den Wert des jeweiligen Sytheseparameters. Der richtige Wert wird allerdings nur von dem Element richtig angezeigt, mit dem er zuletzt verändert wurde.

Faders0-15 ist kein eingebettetes patcher-Objekt sondern ein MAX *subpatch*, der automatisch in GRAINY'sFirst geladen wird. (Abb.15)

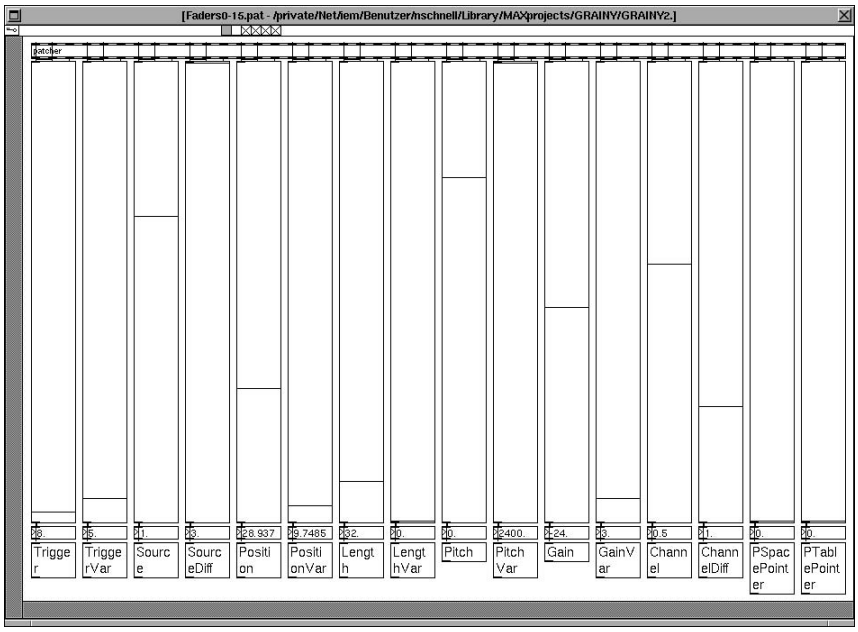


Abbildung 15: Der *subpatch* Faders0-15 mit den Ein-/Ausgabeelementen für die Syntheseparameter, den *PTablePointer* und den *PSpacePointer*

3.3 Steuerung und Grainparameter-Synthese

3.3.1 Der patcher control

Im patcher control sind alle Steuerungsobjekte untergebracht, die nicht einem einzelnen Syntheseparameter zugeordnet sind. (Abb.16)

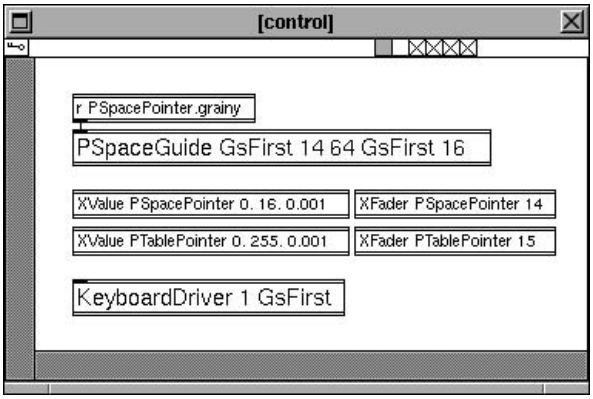


Abbildung 16: Der patcher control

Das PSpaceGuide-Objekt verwaltet von hier aus die PSpace-Objekte im patcher parameters. Beim Laden von GRAINY'sFirst liest das PSpaceGuide-Objekt zunächst die pbank-Dateien GsFirst.pspace und GsFirst.index, wo in der Voreinstellung bereits einige Klangzustände gespeichert sind. Weitere pbank-Dateien können von der Bedienungsoberfläche aus geschrieben und eingelesen werden. Ebenfalls von der Bedienungsoberfläche aus werden die aktuellen Werte der Syntheseparameter in die PSpace-pbank gespeichert und von dort wieder abgerufen. Die Index-pbank läßt sich am einfachsten in einem Texteditor erstellen.

Die beiden XValue-Objekte für den *PSpacePointer* und den *PTablePointer* sind über XFader-Objekte mit je einen Anzeige-/Eingabeelement in Faders0-15 verbunden.

Der KeyboardDriver empfängt ankommende Daten von einer MIDI-Tastatur und sendet *messages* an alle Parameter-Objekte und die beiden XValue-Objekte. Die Tastenbelegung für das KeyboardDriver-Objekt ist in der pbank-Datei GsFirst.keyboard gespeichert und wird beim Laden von GRAINY'sFirst automatisch eingelesen.

3.3.2 Der patcher parameters und die PTables

Für jeden der vierzehn Syntheseparameter gibt es in GRAINY'sFirst vier Objekte.

1. Ein Parameter-Objekt, welches den Syntheseparameter selbst repräsentiert.
2. Ein PFader-Objekt, das es einem Eingabe-/Anzeigeelement zuordnet.
3. Ein PTable-Objekt für die Zuordnung zum PTable-Parameterraum.
4. Ein PSpace-Objekt für die Zuordnung zum PSpace-Parameterraum.

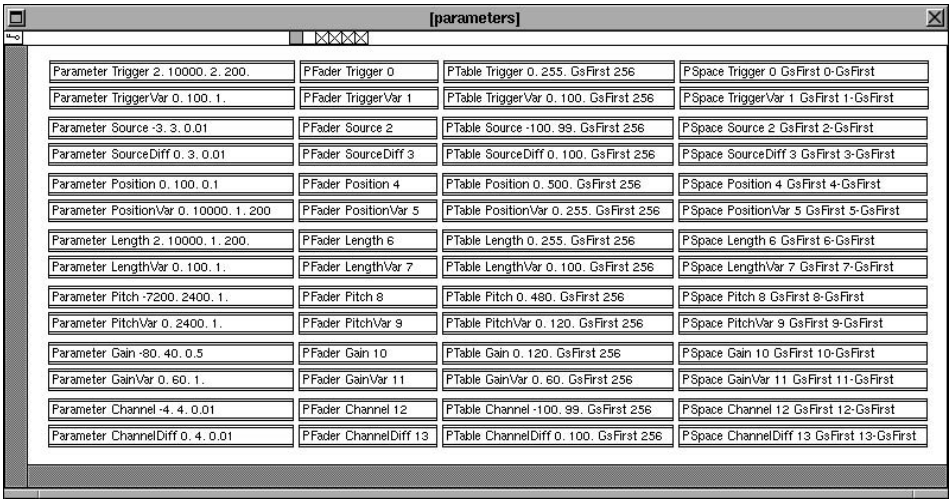


Abbildung 17: Der patcher parameters

Alle Einstellungen, die zu einem bestimmten Syntheseparameter gehören, sind damit im patcher parameters zusammengefaßt. (Abb.17)

Für jeden Syntheseparameter wird sein Wertebereich und eine Quantisierung festgelegt. Die Parameter-Objekte der Syntheseparameter *Trigger*, *PositionVar* und *Length* haben zusätzlich einen Wert für die exponentielle Skalierung angegben. Dadurch werden die von dem jeweiligen PFader- und PTable-Objekt eingehenden Werte exponentiell skaliert. Die in die gegengesetzte Richtung aus dem Parameter-Objekt ausgegebenen Werte werden entsprechend logaritmiert.

Die zu den PTable-Objekten gehörigen MAX table-Objekte sind in einem eigenen MAX *patch* gespeichert, der jeweils eigens geladen werden muß (siehe 4.3 S.32). Nur so können in der vorliegenden Version von MAX verschiedene Werteverläufe für einen PTable-Parameterraum gespeichert und wieder eingelesen werden. Ein Prototyp hierfür ist der *patch* GsFirst.tables.pat. (Abb.18)

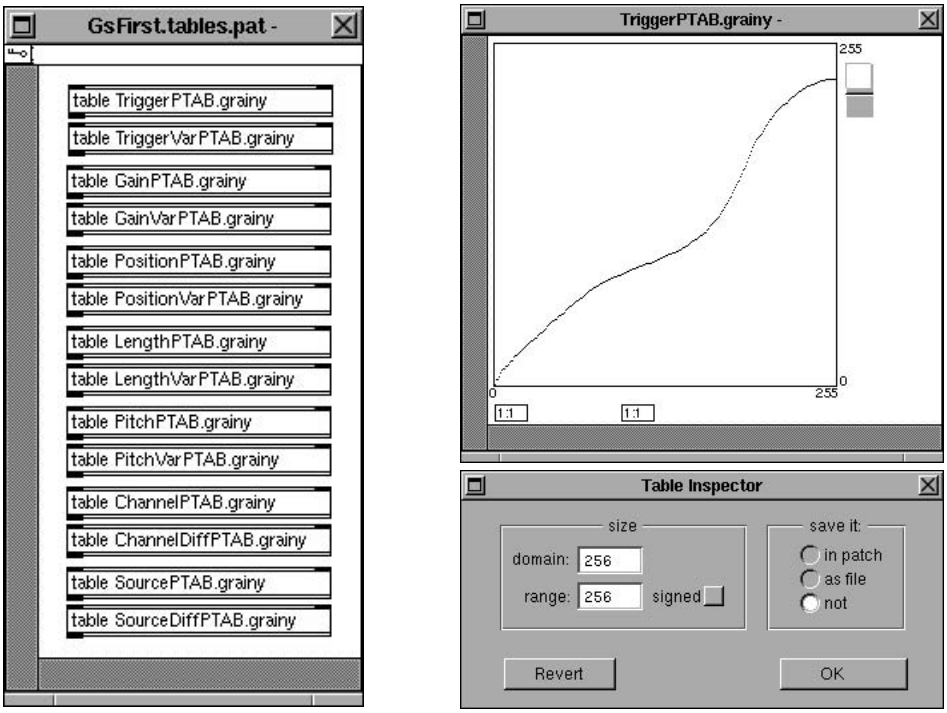


Abbildung 18: Der *patch* GsFirst.tables.pat und ein Table mit den Einstellungen im MAX *Table Inspector*

Für jeden Table ist seine Größe und sein Wertebereich im MAX *Table Inspector* entsprechend den Argumenten des PTable-Objektes eingestellt[15] (siehe Anhang B.4 S.42).

[15] Als Einstellung für *save it* muß *in patch* gewählt sein, damit der Inhalt der Tables mit GsFirst.tables.pat abgespeichert wird.

3.3.3 Der patcher synthesis

Der patcher synthesis enthält für jeden Grainparameter und den Triggergenerator ein eigenes patcher-Objekt, in dem ein Paar von Syntheseparametern empfangen wird (z.B. *Trigger* und *TriggerVar*). Der patcher TriggerSYNTH gibt für jedes Grain eine bang *message* aus. Die anderen patcher erzeugen je einen Wert für einen Grainparameter. Die Werte werden durch das pack-Objekt zu einer *list message* zusammengefaßt und mit dem bang des Triggers als Grainparameter-Vektor an den VoiceAllocator ausgegeben (Abb.19). Das VoiceAllocator-Objekt ist ein MAX *extern*, das die Grainparameter-Vektoren direkt an die Voice-Objekte sendet. Die *send/receive*-Kanäle des VoiceAllocator-Objektes dienen lediglich der Kommunikation mit der Bedienungsoberfläche.

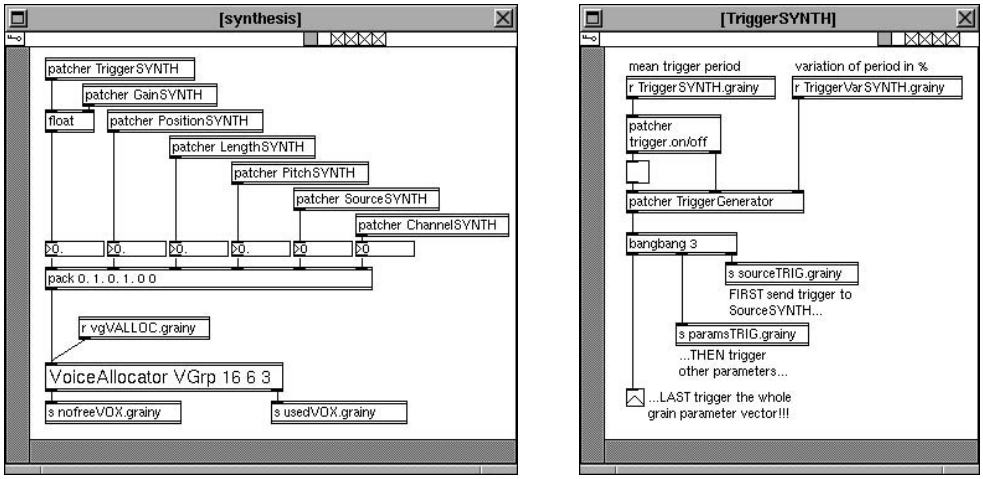


Abbildung 19: Der patcher synthesis und der patcher TriggerSYNTH

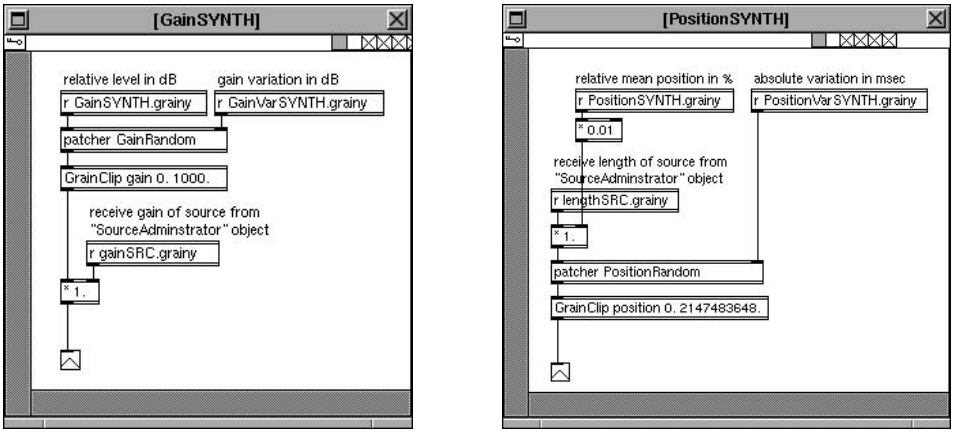


Abbildung 20: Die patcher GainSYNTH und PositionSYNTH

Der bang des Triggergenerators für ein Grain wird zuerst an den patcher SourceSYNTH gesendet. So wird vor allem anderen das Quellsignal bestimmt, aus dem das Grain stammt. Die Nummer des Quellsignales wird daraufhin an den SourceAdministrator übergeben, der die Quellparameter *srcgain* und *srclength* an die patcher GainSYNTH und PositionSYNTH ausgibt. (Abb.20)

Als nächstes wird der bang des Triggergenerators über paramsTRIG.grainy an die Zufallsgeneratoren in GainSYNTH, PositionSYNTH, LengthSYNTH, PitchSYNTH und ChannelSYNTH gesendet. Dort wird für den jeweiligen Grainparameter ein gleichverteilter Zufallswert im Rahmen der jeweiligen Variation erzeugt und zu dem Mittelwert addiert. Für die Grainparameter *gain* und *position* wird der relative Wert mit dem bereits vorliegenden Quellparameter multipliziert, um die absolute Größe zu erhalten.

Der Syntheseparameter *Channel* gibt den Mittelwert der Grainverteilung auf die vier Ausgangskanäle 0 bis 3 an. Durch eine Modulo-Arithmetik liegt der Wert auf einem Kreis, der in vier gleich große Segmente eingeteilt ist. Jedes der Segmente repräsentiert einen Ausgangskanal. (Abb.21)

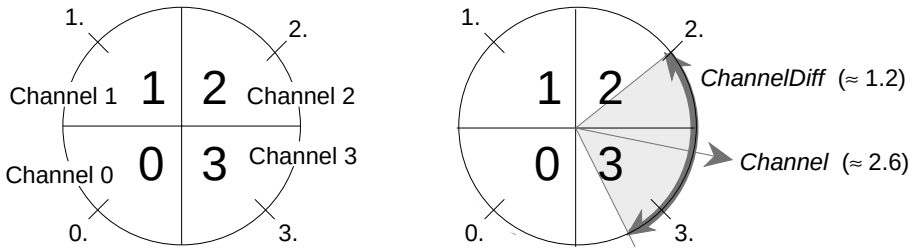


Abbildung 21: Die Verteilung der Grains auf die vier Ausgangskanäle

Um den Mittelwert *Channel* werden im Bereich von *ChannelDiff* gleichverteilte Zufallswerte erzeugt. Ist beispielsweise *ChannelDiff* mit 1.0 angegeben, ist die Variation so breit wie das einem Ausgangskanal zugeordnete Kreissegment. Wenn der *Channel*-Parameter dazu 2.0 beträgt, werden die Grains nur auf Kanal 2 ausgegeben. Für einen *Channel*-Wert von 2.2 liegen 80% der Grains auf Kanal 2 und 20% auf Kanal 3. Steigt der Mittelwert weiter an, wandert die Verteilung der Grains allmählich auf Kanal 3. Für einen größeren Wert von *ChannelDiff* verteilen sich die Grains auf mehrere Ausgangskanäle. Ist *ChannelDiff* 4.0, sind die Grains gleichmäßig auf alle vier Ausgangskanäle verteilt, unabhängig von dem Wert von *Channel*.

Die Bestimmung des Quellsignales aus den Syntheseparametern *Source* und *SourceDiff* geschieht analog zur Bestimmung des Ausgangskanales. Da es in GRAINY'sFirst nur drei Quellsignale gibt, ist der Kreis hier lediglich in drei Segmente aufgeteilt.

3.4 Der Graingenerator

Die drei Quellsignale des Graingenerator von GRAINY'sFirst können jeweils maximal zehn Sekunden lang sein. In der Stimmgruppe VgGrp sind sechzehn Stimmen der Variante tab zusammengefaßt.

3.4.1 Der patcher voices und der patcher sources

Der patcher voices (auf CP2) enthält für jede der sechzehn Stimmen ein Voice-Objekt. Die Objekte müssen zum *order forcing* miteinander verbunden sein. (Abb.22)

Im patcher sources (CP 2) befinden sich der SourceAdministrator und die drei Source-Objekte für die Quellsignale (Abb.22). Bei der ersten Initialisierung liest der SourceAdministrator die pbank-Datei GsFirst.sources ein. Die Source-Objekte laden daraufhin die dort angegebenen Quellsignale. In der Voreinstellung sind es die Soundfiles demo.snd, demo1.snd und demo2.snd. Dateien für die Source-pbank lassen sich von der Bedienungsoberfläche aus laden. Die Source-pbank wird am einfachsten in einem Texteditor erstellt.

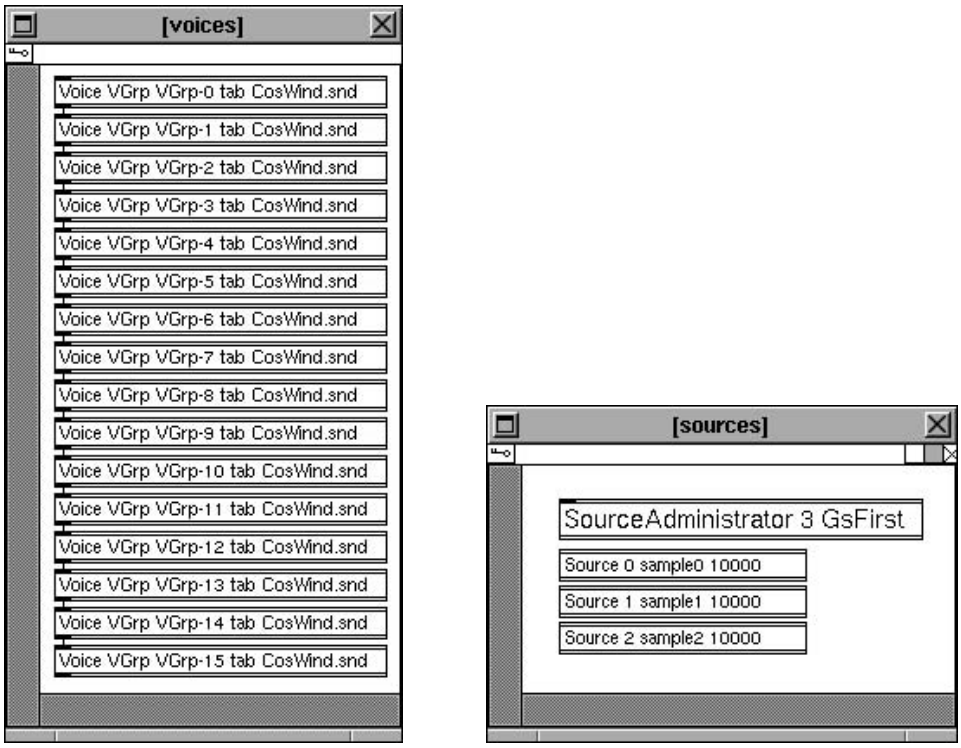


Abbildung 22: Der patcher voices und der patcher sources

3.4.2 Der patcher channels und der patcher outs

Im patcher channels (CP 2) befinden sich die vier Channel-Objekte für die vier Ausgangskanäle von GRAINY'sFirst. Die Objekte sind zwecks *order forcing* kaskadiert und über *outlets* mit dem patcher outs auf dem ersten Prozessor (CP 1) verbunden.

Der patcher outs enthält lediglich die vier dac~-Objekte für die ISPW Ausgänge.

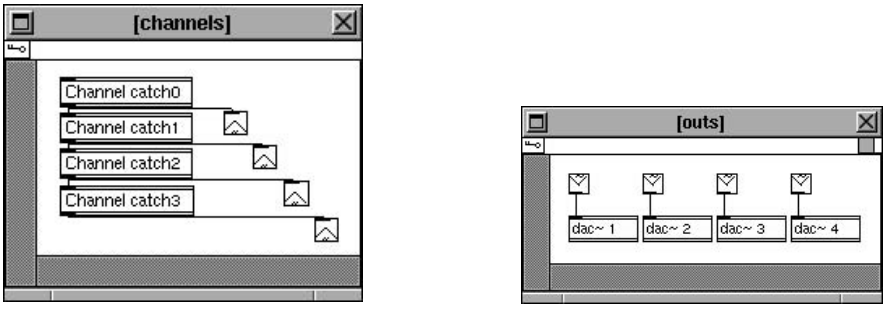


Abbildung 23: Der patcher channels und der patcher outs

4 Installation und Hinweise

4.1 GRAINY *path*

Alle zu GRAINY gehörenden MAX *patches* befinden sich in einem Verzeichnis mit dem Namen grainy. (Abb.24)

- Die *abstractions* und *externs* der GRAINY Objekte.
- Die *help patches* für die GRAINY Objekte.
- Die *demo patches*.
- Die pbank-Dateien und Soundfiles zu den *help patches* und *demo patches*.
- Einige für viele GRAINY Anwendungen hilfreiche *abstractions*, die auch in GRAINY'sFirst verwendet werden.

GRAINY-Objekte:

SourceAdministrator.abs
Source.abs
VoiceAllocator/
Voice.abs
Channel.abs
Parameter.abs
PFader.abs
PTable.abs
PSpaceGuide.abs
PSpace.abs
XValue.abs
XFader.abs
XTable.abs
KeyboardDriver.abs

Faders0-15.pat
Faders0-3.pat
Faders0-7.pat

Varianten des Voice-Objektes:

noVOX.abs
rampVOX.abs
tabVOX.abs

externs für GRAINY Objekte:

exscale/
logscale/

help und *demo patches:*

help-GRAINY.pat
help-SourceAdministrator.pat
help-Source.pat
help-VoiceAllocator.pat
help-Voice.pat
help-Channel.pat
help-Parameter.pat
help-PFader.pat
help-PTable.pat
help-PSpaceGuide.pat
help-PSpace.pat
help-XValue.pat
help-XFader.pat
help-XTable.pat
help-KeyboardDriver.pat

demo-GrainGenerator.pat
demo-PSpace.pat
demo-Parameter.pat

demo.index
demo.keyboard
demo.pspace
demo.source

demo.snd
demo1.snd
demo2.snd

CosWind.snd

weitere GRAINY *abstractions:*

GRAINYinit.abs
GRAINYkey.abs
GrainClip.abs
num128.abs

Abbildung 24: Die Dateien im Verzeichnis grainy

Für eine GRAINY Anwendung muß sichergestellt sein, daß das Verzeichnis dieser Dateien als einer der Pfade angegeben ist, in denen MAX nach Objekten sucht.

4.2 *help patches* und *demo patches*

Die *help patches* können von MAX aus durch Halten der Taste *alternate* und Mausklick auf ein GRAINY-Objekt geöffnet werden[16]. Im *patch* help-GRAINY.pat sind alle GRAINY Objekte zusammengefaßt.

Die *demo patches* zeigen in Kurzform die Anwendung und Funktionsweise aller Objekte. (siehe Anhang A S.33ff)

4.3 MAX table-Objekte

Wegen der Fehlerhaftigkeit der table-Objekte in der vorliegenden MAX Version (0.25) empfiehlt es sich alle table-Objekte einer GRAINY Anwendung in einem eigenen *patch* zusammenzufassen. In diesem PTable-*patch* müssen die entsprechenden Einstellungen für *domain* und *size* für jeden PTable im *Table Inspector* gesetzt werden. Alle PTables können von hier aus editiert werden. Wenn im *Table Inspector* bei allen Objekte für die Speicherung *in patch* gewählt ist (Voreinstellung), wird mit der Speicherung dieses *patches* der Inhalt aller PTables gesichert.

Da sich in den PTable-Objekten jeweils ein MAX table-Objekt mit gleichem Namen befindet, das allerdings von sich aus keine sinnvollen Daten und Einstellungen enthält, muß ein PTable-*patch* stets v o r der GRAINY-Anwendung geladen werden. Nur so sind die Namen der GRAINY PTable-Objekte mit den MAX table-Objekten assoziiert, die die richtigen Einstellungen für *domain* und *size* haben. Weitere Datensätze für die PTables können dann in jeweils eigenen PTable-*patches* erstellt werden.

4.4 Abtastrate

Die Implementation der MAX Objekte, die *messages* verarbeiten, beruht auf einem Takt von einer Millisekunde. Alle Verzögerungen (z.B. für delay) und Perioden (z.B. für metro) lassen sich daher nur in ganzen Millisekunden einstellen. Der tatsächliche Rechenzyklus der ISPW beträgt allerdings das 64fache der eingestellten ISPW Abtastperiode.

- 1.33 ms für 48 kHz.
- 1.45 ms für 44.1 kHz.
- 2 ms für 32 kHz.

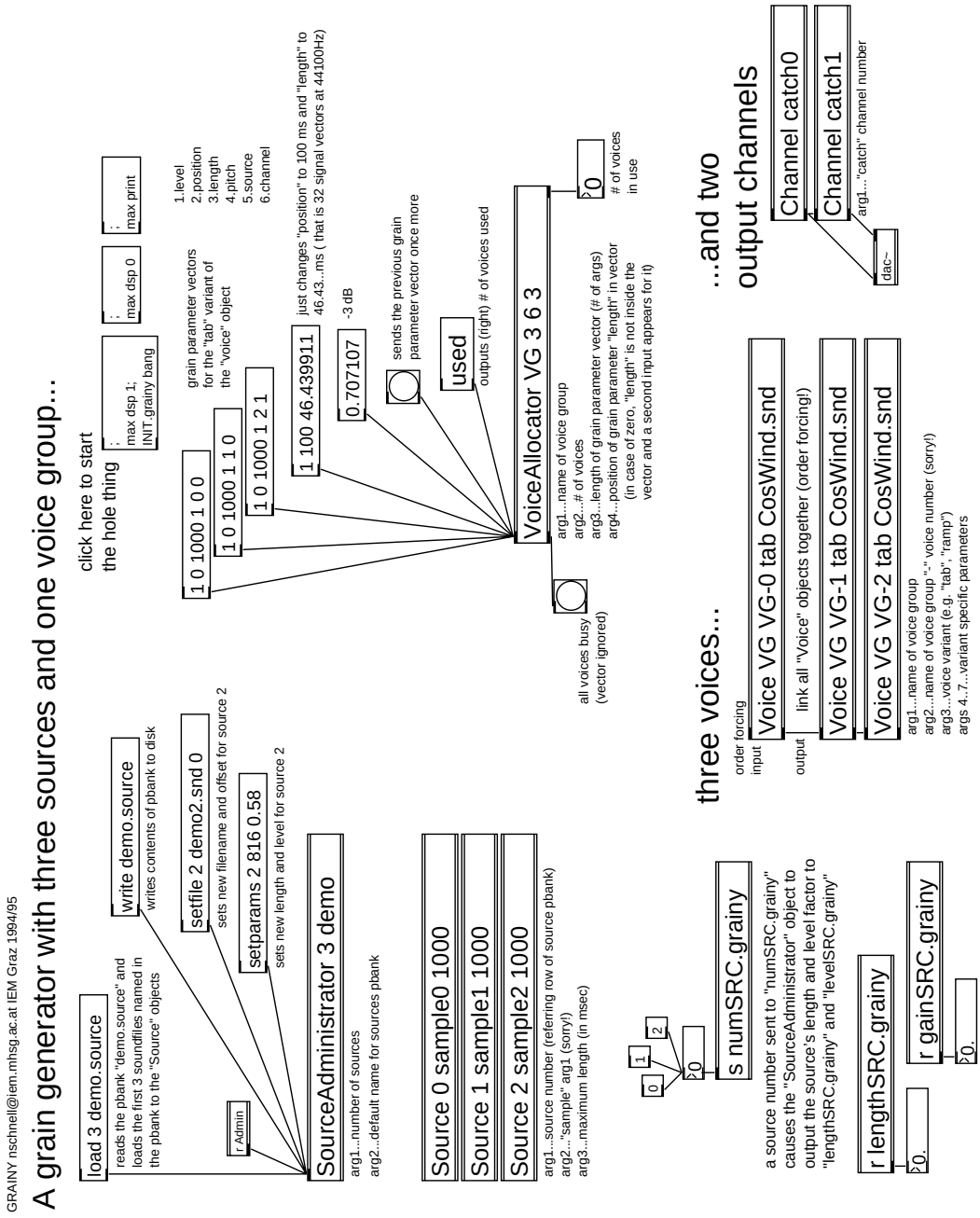
Die dadurch entstehenden Unregelmäßigkeiten für *messages* mit einer Periode, die kein Vielfaches des Rechenzyklus ist, haben meist keinen hörbaren Effekt. In einigen Anwendungen von GRAINY mit kurzen Perioden und gleichmäßiger Verteilung der Grains über der Zeit macht sich jedoch eine starke Rauhigkeit bemerkbar. Für eine Abtastrate von 32 kHz lassen sich Effekte solcher Art vermeiden, indem nur geradzahlige Perioden und Grainlängen verwendet werden (2, 4, 6, ... ms). Diese Abtastrate ist auch wegen der hohen Auslastung der ISPW zu empfehlen. Mit 32 kHz ist es möglich sechzehn Voice-Objekte der Variante tab (sicherlich die aufwendigste Variante) auf einem Prozessor der ISPW unterzubringen.

[16] Wegen der *send/receive*-Kanäle für den Datenaustausch zwischen den GRAINY Objekten, kann es zu Wechselwirkungen zwischen bereits geladenen Objekten und denen in einem *help patch* kommen.

A ANHANG: GRAINY *demonstration patches*

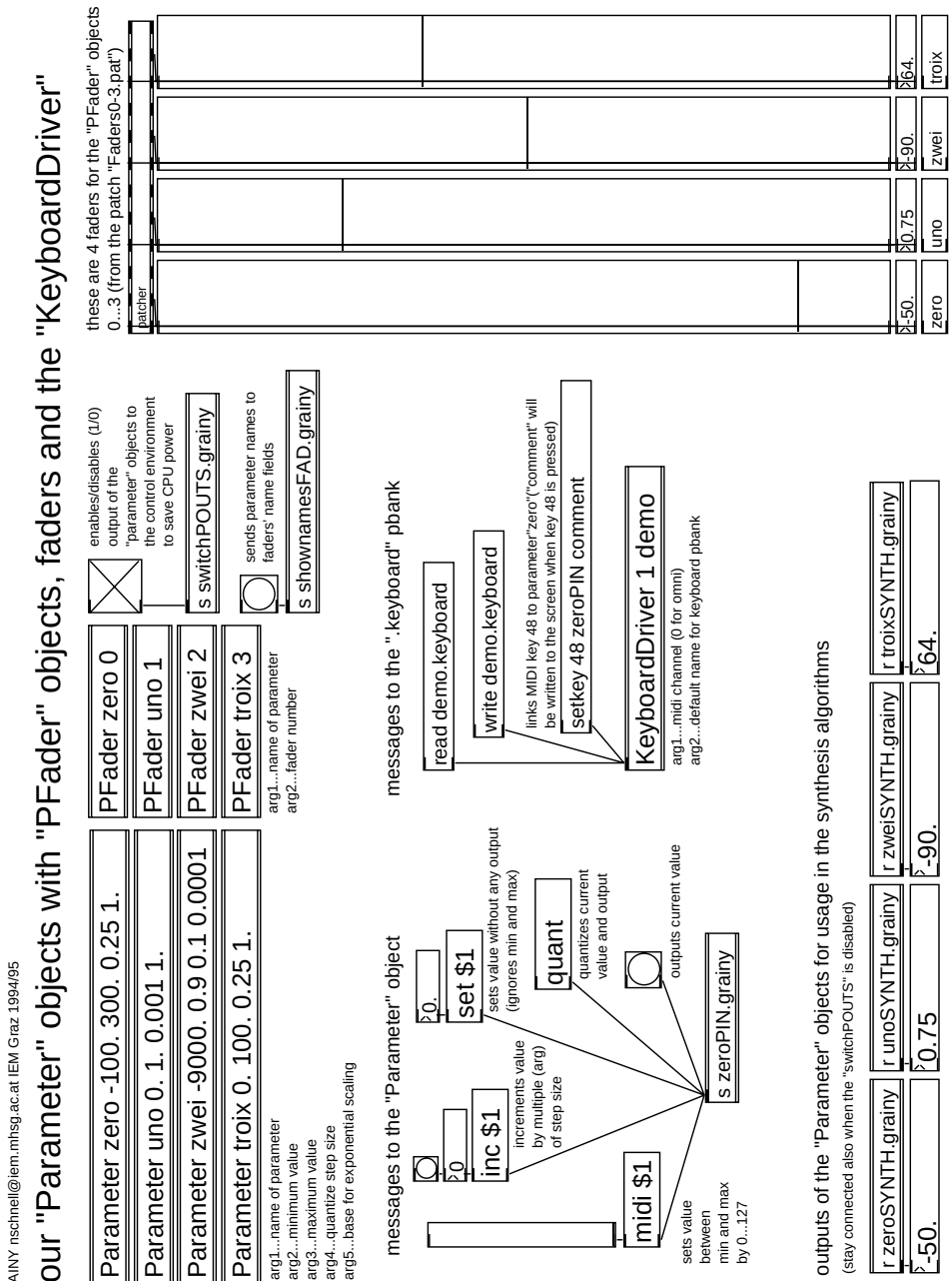
A.1 Graingenerator

Graingenerator mit drei Quellsignalen und einer Stimmgruppe.



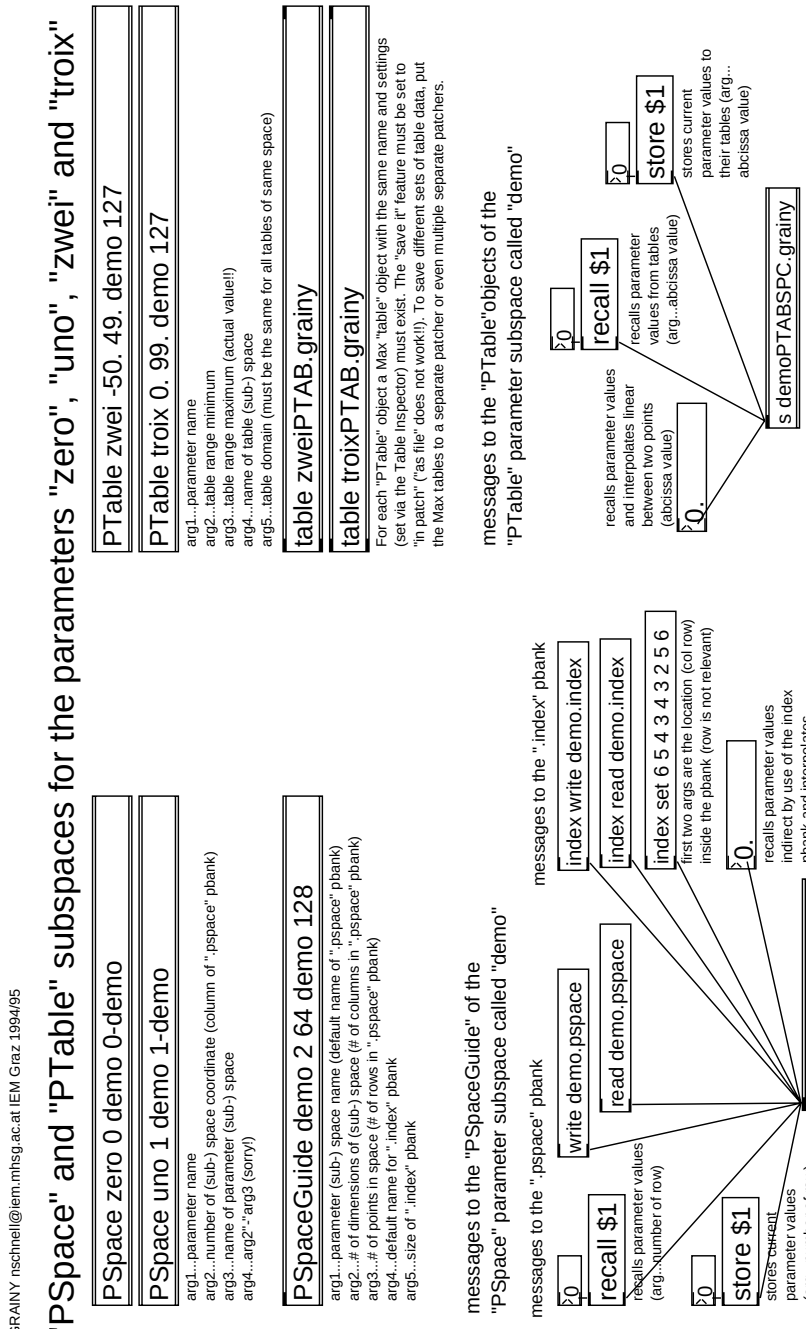
A.2 Parametereingabe/-anzeige

Vier Parameter-Objekte mit PFader-Objekten, Eingabe-/Anzeigeelementen und Keyboard-Driver.



A.3 Steuerung

Ein *PTable*-Parameterraum und ein *PSpace*-Parameterraum mit jeweils zwei Parametern.



B ANHANG: GRAINY Objektreferenz

B.1 Das SourceAdministrator-Objekt und das Source-Objekt

SourceAdministrator int symbol

Verwaltet die Quellsignale (Source-Objekte und Source-pbank).

Argumente:

int Anzahl der zu verwaltenden Source-Objekte.

symbol Default Name für Source-pbank.
(wird mit dem Postfix .source ergänzt).

inlet/messages:

load int symbol
Liest Source-pbank Datei und veranlaßt die Source-Objekte dazu , die Quellsignale zu laden.

int Anzahl der Quellsignale.

symbol Pfad und Dateiname einer Source-pbank Datei.
(.source Postfix wird nicht automatisch ergänzt!).

setfile int symbol int
Setzt Werte für die Quellparameter *filename* und *offset* in Source-pbank.

int Nummer des Quellsignals (Zeile in pbank).

symbol Pfad und Dateiname für *filename.*

int Wert für *offset* (in ms).

setparams int int float
Setzt Werte für die Quellparameter *srclength* und *srcgain* in Source-pbank.

int Nummer des Quellsignals (pbank Zeile).

int Wert für *srclength* (in ms).

float Wert für *srcgain.*

write symbol
Speichert Source-pbank in Datei.

symbol Pfad und Dateiname für Source-pbank Datei.
(.source Postfix wird nicht automatisch ergänzt!).

receive:

r SRC.grainy *message*
Empfängt alle *messages* wie *inlet*.

r numSRC.grainy int
Empfängt Nummer eines Quellsignals.
(Das SourceAdministrator Objekt sendet daraufhin Quellparameter *srclength* und *srcgain*)

int Nummer eines Quellsignals.

send:

s lengthSRC.grainy int
Sendet Quellparameter src*length* (siehe oben)

int src*length*, Länge des Quellsignals aus Source-pbank (in ms).

s gainSRC.grainy int
Sendet Quellparameter src*gain* (siehe oben)

float src*gain*, Amplitudenfaktor aus Source-pbank.

Format der Source-pbank:

Je eine Zeile für jedes Quellsignal.
(Zeilennummer referenziert auf Source-Objekt mit gleicher Nummer.)
Zeilenformat: symbol int int float

symbol	Pfad und Dateiname für *filename.*
int	Wert für *offset* (in ms).
int	Wert für *srclength* (in ms).
float	Wert für *srcgain.*

Source int symbol int

Repräsentiert Quellsignal (reserviert Speicher und lädt Quellsignal).

Argumente:

int	Nummer des Quellsignals (*Nummer*).
symbol	"sample" und Nummer des Quellsignals sein (z.B.: sample0).
int	Reservierte Länge für Quellsignal in ms (= maximale Länge für Quellsignal).

receive:

r *Nummer***SRC.grainy**
Empfängt read-*message* vom SourceAdministrator-Objekt.

B.2 Das VoiceAllocator-Objekt, das Voice- und Channel-Objekt

VoiceAllocator symbol int int int (MAX *extern*)

Verwaltet eine Stimmgruppe. Leitet eingehende Grainparameter-Vektoren an jeweils freie Stimmen der Stimmgruppe weiter.

Argumente:

symbol Name der Stimmgruppe.

int Anzahl der zu verwaltenden Voice-Objekte.

int Länge des Grainparameter-Vektors der in der Stimmgruppe verwendeten Variante des Voice-Objektes.

int Position des Grainparameters *length* im Grainparameter-Vektor. (für 0 hat VoiceAllocator-Objekt einen zweiten *inlet*)

inlet links*:*

list *Grainparameter*
Grainparameter-Vektor wird an freies Voice-Objekt gesendet. Wenn kein freies Voice-Objekt verfügbar ist, wird der Vektor ignoriert und ein bang am linken Ausgang ausgegeben .

Grainparameter hängen von der verwendeten Variante des Voice-Objektes ab.

bang
Wiederholt vorigen Grainparameter-Vektor.

int
Wie list mit einem Argument.

used
Sendet Anzahl der momentan benutzten Stimmen aus rechtem Ausgang.

inlet rechts (nur falls 4. Argument nicht angegeben oder 0)*:*

float
Grainlänge (in ms).

outlet links*:*

bang
bang, wenn kein freies Voice-Objekt verfügbar ist.

outlet rechts*:*

int
Anzahl der momentan benutzten Stimmen.

send:

Kanäle für Grainparameter-Vektoren an die einzelnen Voice-Objekte.

Voice symbol symbol symbol [. . .]

Repräsentiert eine Stimme. Empfängt Grainparameter-Vektoren vom VoiceAllocator-Objekt und gibt die Grains auf den Ausgangskanälen aus.

Argumente:

symbol Name der Stimmgruppe (*Group*).

symbol Name der Stimmgruppe, "-" und Nummer der Stimme (*Voice*).
(z.B.: VG-0 für Stimme 0 in Stimmgruppe "VG")

symbol Variante des Voice-Objektes.

. . . Von der Variante abhängige Argumente.

inlet/outlet:

signal für *order forcing*

receive:

r *Voice***VOX.grainy** list
Empfängt Grainparameter-Vektoren von VoiceAllocator-Objekt.
list Grainparameter-Vektor.

Bisher implementierte Varianten des Voice-Objektes:

Voice symbol symbol **no**
Voice symbol symbol **tab** symbol
Voice symbol symbol **pospitch** symbol int
Voice symbol symbol **ramp** float float

---

no

Leere Variante, die lediglich die *signal message* zum *order forcing* weitergibt.

---

tab symbol

Variante mit Einhüllender durch MAX osc1~ und sechs Grainparametern.

Argumente:

symbol Pfad und Name von Soundfile für Einhüllende.

Grainparameter-Vector: float float float float int int

float	*gain,*	linearer Amplitudenfaktor.
float	*position,*	Anfangsposition relativ zum Anfang des Quellsignals (in ms).
float	*length,*	Grainlänge (in ms).
float	*pitch,*	linearer Tonhöhenfaktor.
int	*source,*	Nummer des Quellsignals.
int	*channel,*	Nummer des Ausgangskanals.

---

pospitch symbol int

Variante mit Einhüllender durch MAX `osc1~` und zwei Grainparametern.

Argumente:

symbol Pfad und Name von Soundfile für Einhüllende.

int Grainlänge (in ms).

receive:

r *Group***lengthVOX.grainy** float
Grainlänge (in ms).

r *Group***sourceVOX.grainy** int
Nummer für Quellsignal.

r *Group***channelVOX.grainy** int
Nummer für Ausgangskanal.

Grainparameter-Vektor: float float

float	*position,*	Anfangsposition relativ zum Anfang des Quellsignals (in ms).
float	*pitch,*	linearer Tonhöhenfaktor.

---

ramp float float

Variante mit trapezförmiger Einhüllender und sechs Grainparametern.

Argumente:

float Anstiegszeit der Einhüllenden (in ms).

float Abfallszeit der Einhüllenden (in ms).

receive:

r *Group***attackVOX.grainy** float
Anstiegszeit der Einhüllenden (in ms).

r *Group***decayVOX.grainy** float
Abfallszeit der Einhüllenden (in ms).

Grainparameter-Vektor: float float float float int int

Wie tab-Variante.

Channel symbol

Repräsentiert einen Ausgangskanal. Gibt Signal der Grains aus.

argument:

symbol "catch" und die Nummer des Ausgangskanals (z.B.: catch0)

inlet:

signal für *order forcing*

outlet:

signal Signal des Ausgangskanals.

B.3 Das Parameter-Objekt

Parameter symbol float float float float

Repräsentiert einen Syntheseparameter.
Verbindet die Grainparameter-Synthese mit Steuerumgebung den Syntheseparameter .

Argumente:

symbol Name des Syntheseparameters (*Name*).
float Minimum.
float Maximum.
float Quantisierung.
float Basis zur exponentiellen Skalierung.

messages:

float und **int**
Setzt neuen Wert.

midi float
Setzt den Wert innerhalb des angegebenen Wertebereiches des Syntheseparameters.

float 0. bis 127. wird auf Wertebereich skaliert.

inc int
Inkrementiert Wert um Vielfaches (int) von Quantisierung.

set float
Setzt Wert (float) ohne Ausgabe (ignoriert Minimum und Maximum).

bang
Gibt aktuellen Wert aus.

quant
Quantisiert aktuellen Wert.

receive:

r *Name***PIN.grainy** *message*
Empfängt alle *messages* wie *inlet*.

r *Name***PIN%.grainy** float
Skaliert Eingangswerte (float) zwischen 0. und 127. auf Wertebereich des Syntheseparameters.

send:

s *Name***POUT.grainy** float
Gibt aktuellen Wert des Syntheseparameters aus.

s *Name***POUT%.grainy** float
Gibt aktuellen Wert des Syntheseparameters auf Wertebereich zwischen 0. bis 127. skaliert aus.

s *Name***SYNTH.grainy** float
Gibt aktuellen Wert aus (für Verbindung zur Grainparameter-Synthese).

s switchPOUTS.grainy int
Aktiviert (int != 0) und deaktiviert (int == 0) alle Ausgänge (*sends*) bis auf ...SYNTH.grainy

B.4 Das PFader-Objekt und das PTable-Objekt

PFader symbol int

Verbindet Parameter-Objekt mit Anzeige-/Eingabeeinheit.

Argumente:

symbol Name des Syntheseparameters.

int Nummer des Eingabe-/Anzeigeobjektes.

receive:

r shownamesFAD.grainy bang
Setzt Beschriftung am Eingabe-/Anzeigeelement.

Weitere *send/receive*-Kanäle zum Datenaustausch mit dem Eingabe-/Anzeigeelement und dem zugehörigen Parameter-Objekt.

PTable symbol float float symbol int

Ordnet Syntheseparameter einem PTabble-Parameterraum zu.
Verbindet Parameter-Objekt mit PTable.

Argumente:

symbol Name des Syntheseparameters (*Name*).

float Niedrigster Wert des zugehörigen MAX table-Objektes *Name*PTAB.grainy.

float Höchster Wert des zugehörigen MAX table-Objektes.

symbol Name des PTable-Parameterraumes (*Space*).

int Größe (Abszisse) des table-Objektes
(muß für alle tables des PTable-Parameterraumes gleich sein).

messages:

float und **int**
Wert für Abzisse. Ruft die am gegebenen Index (int) im PTable gespeicherten Wert ab. Zwischen den Werten an den benachbarten Indizes (float) wird linear interpoliert .

int
Ruft die am gegebenen Index (int) im PTable gespeicherten Wert ab.

store int
Schreibt den aktuellen Wert des Syntheseparameters am gegebenen Index (int) in PTable.

recall int
Ruft die am gegebenen Index (int) im PTable gespeicherten Wert ab (wie int).

receive:

r *Space***PTABSPC.grainy** *message*
Empfängt alle *messages* (siehe oben).

Weitere *send/receive*-Kanäle zum Datenaustausch mit Parameter-Objekt.

B.5 Das PSpaceGuide-Objekt und das PSpace-Objekt

PSpaceGuide symbol int int symbol int int

Verwaltet einen PSpace-Parameterraum (seine PSpace-Objekte sowie PSpace- und Index-pbank).

Argumente:

- **symbol** Name des PSpace-Parameterraumes (*Name*).
  (auch Default Name für PSpace-pbank, wird mit .pspace ergänzt).
- **int** Anzahl der PSpace-Objekte (Anzahl der Spalten der PSpace-pbank).
- **int** Maximale Anzahl der Punkte im Parameterraum (Zeilen der PSpace-pbank).
- **symbol** Default Name für Index-pbank (wird mit .index ergänzt).
- **int** Größe der Index-pbank (Anzahl der Spalten).

inlet/messages:

float und **int**
Ruft Werte des PSpace-Parameterraumes über Index-pbank ab (fraktional adressiert).

store int
Speichert aktuelle Werte der zum PSpace-Parameterraum gehörenden Syntheseparameter in einer Zeile der PSpace-pbank ab.

recall int
Ruft Werte für die zum PSpace-Parameterraum gehörenden Syntheseparameter aus einer Zeile der PSpace-pbank ab.

write symbol
Schreibt Inhalt der PSpace-pbank in Datei.

- symbol Pfad und Dateiname für Source-pbank Datei.
  (.pspace Postfix wird nicht automatisch ergänzt!).

read symbol
Liest Inhalt der PSpace-pbank aus Datei.

- symbol Pfad und Dateiname für Source-pbank Datei.
  (.pspace Postfix wird nicht automatisch ergänzt!).

index write symbol
Schreibt Inhalt der Index-pbank in Datei .

- symbol Pfad und Dateiname für Index-pbank Datei.
  (.index Postfix wird nicht automatisch ergänzt!).

index read symbol
Liest Inhalt der Index-pbank aus Datei.

- symbol Pfad und Dateiname für Index-pbank Datei.
  (.index Postfix wird nicht automatisch ergänzt!).

index set int int list
Setzt ab der gegebenen Spalte neue Werte für die Index-pbank.

- int Spalte.
- int Zeile (irrelevant, da Index-pbank nur eine Zeile hat).
- list Indizes (Zeilennummern der PSpace-pbank).

receive:

r *Name***PSPACE.grainy** *message*
Empfängt alle *messages* wie *inlet*.

Weitere *send/receive*-Kanäle zum Datenaustausch mit PSpace-Objekten.

Format der PSpace-pbank:

Je eine Zeile für jeden Punkt im PSpace-Parameterraum (Klangzustand).
Je eine Spalte für jeden Syntheseparameter des PSpace-Parameterraumes
(Spaltennummer referenziert auf PSpace-Objekt mit gleicher Nummer).

Format der Index-pbank:

Lediglich eine Zeile mit den aufeinanderfolgenden Indizes der Punkte im PSpace-Parameterraum
(Zeilennummern in der PSpace-pbank).

PSpace symbol int symbol symbol

Ordnet Syntheseparameter einem PSpace-Parameterraum zu.

Argumente:

symbol Name des Syntheseparameters.

int Nummer für Koordinate (Spalte in PSpace-pbank).

symbol Name des PSpace-Parameterraumes.

symbol Nummer für Koordinate, "-" und Name des PSpace-Parameterraumes (z.B.: 0-demo).

send/receive:

Kanäle zum Datenaustausch mit PSpaceGuide- und Parameter-Objekt.

B.6 Die X-Objekte

XValue symbol float float float float

Verwaltet einen beliebigen Wert und verbindet ihn mit einer Steuerumgebung (ähnlich wie Parameter-Objekt).

Argumente:

- **symbol** Name des X-Parameters (= Name eines receive-Objektes) (*Name*).
- **float** Minimum.
- **float** Maximum.
- **float** Quantisierung.
- **float** Basis zur exponentiellen Skalierung.

inlet/messages:

float und **int**
Setzt neuen Wert.

midi float
Setzt den Wert innerhalb des angegebenen Wertebereiches des X-Parameters.

float 0. bis 127. wird auf Wertebereich skaliert.

inc int
Inkrementiert Wert um Vielfaches (int) von Quantisierung.

set float
Setzt Wert ohne Ausgabe (ignoriert Minimum und Maximum).

bang
Gibt aktuellen Wert aus.

quant
Quantisiert aktuellen Wert.

receive:

r *Name***IN.grainy** *message*
Empfängt alle *messages* wie *inlet*.

r *Name***IN%.grainy** float
Skaliert Eingangswerte (float) zwischen 0. und 127. auf Wertebereich des X-Parameters.

send:

s *Name***OUT.grainy** float
Gibt aktuellen Wert aus.

s *Name***OUT%.grainy** float
Gibt aktuellen Wert des X-Parameters auf Wertebereich zwischen 0. bis 127. skaliert aus.

s *Name***.grainy** int
Sendet an Zielobjekt.

s switchXOUTS.grainy int
Aktiviert (int != 0) und deaktiviert (int == 0) alle Ausgänge (*sends*) bis auf den an das Zielobjekt.

XFader symbol int

Verbindet ein XValue-Objekt mit einer Anzeige-/Eingabeeinheit (ähnlich wie PFader-Objekt).

Argumente:

symbol Name des X-Parameters.

int Nummer des Eingabe-/Anzeigeobjektes.

receive:

r shownamesFAD.grainy bang
Setzt Beschriftung am Eingabe-/Anzeigeelement.

Weitere *send/receive*-Kanäle zum Datenaustausch mit dem Eingabe-/Anzeigeelement und dem zugehörigen XValue-Objekt.

XTable symbol float float symbol int

Verbindet ein XValue-Objekt mit einem XTable (ähnlich wie PTable-Objekt).

Argumente:

symbol Name des X-Parameters (*Name*).

float Niedrigster Wert des zugehörigen MAX table-Objektes *Name*XTAB.grainy.

float Höchster Wert des zugehörigen MAX table-Objektes.

symbol Name des PTable-Parameterraumes (*Space*).

int Größe (Abszisse) des table-Objektes
(muß für alle tables des PTable-Parameterraumes gleich sein).

messages:

float und **int**
Wert für Abzisse. Ruft die am gegebenen Index (int) im XTable gespeicherten Wert ab. Zwischen den Werten an den benachbarten Indizes (float) wird linear interpoliert .

store int
Schreibt den aktuellen Wert des X-Parameters am gegebenen Index (int) in XTable.

recall int
Ruft die am gegebenen Index (int) im XTable gespeicherten Wert ab (wie int).

receive:

r *Space***PTABSPC.grainy** *message*
Empfängt alle *messages* (siehe oben).

Weitere *send/receive*-Kanäle zum Datenaustausch mit Parameter-Objekt.

B.7 Das KeyboardDriver-Objekt

KeyboardDriver int symbol

Steuert die Werte von Parameter- und XValue-Objekten über eine MIDI-Tatstatur.

Argumente:

int MIDI-Empfangskanal (0 = omni)

symbol Default Name für Keyboard-pbank
(wird mit dem Postfix .keyboard ergänzt).

inlet/messages:

read symbol
Liest Keyboard-pbank Datei ein.

symbol Pfad und Dateiname einer Keyboard-pbank Datei.
(.keyboard Postfix wird nicht automatisch ergänzt!).

write symbol
Speichert Keyboard-pbank in Datei.

symbol Pfad und Dateiname für Source-pbank Datei.
(. keyboard Postfix wird nicht automatisch ergänzt!).

setkey int symbol symbol
Setzt Information für eine MIDI-Taste (Zeile in Keyboard-pbank).

int Nummer der MIDI-Taste.

symbol Zielobjekt (wird mit Postfix .grainy ergänzt).
NULL, falls nicht verwendet.

symbol Kommentar, der beim Drücken der Taste auf dem Bildschirm erscheint.

send:

Sendet folgende *messages* an Zielobjekt der gedrückten Taste..

bang
für *note on*

midi 0...127
Wert für *modulation wheel.*

inc -10...10
Alle 100...5 ms für Stellung von *pitch bend wheel.*
(Rate kann mit *after touch* geregelt werden).

Format der Keyboard-pbank:

Je eine Zeile für jede MIDI-Note (0...127).
Zeilenformat: int symbol symbol

int Zeilennummer, MIDI-Note.

symbol Zielobjekt (wird mit Postfix .grainy ergänzt).

symbol Kommentar, der beim Drücken der Taste auf dem Bildschirm erscheint.

B.8 Die Faders-*patches*, das GRAINYinit- und das GRAINYkey-Objekt

---

Faders0-3, Faders0-7 und Faders0-15

Anzeige-/Eingabeelemente für Parameter- und XValue-Objekte.

send/receive:

s/r *Nummer***FADOUTfad.grainy**
s/r *Nummer***FADOUTbox.grainy**
s/r *Nummer***FADOUTname.grainy**
Kanäle zum Datenaustausch mit den PFader- und XFader-Objekten.

GRAINYinit

Gibt bei Initialisierung beliebige int, float oder symbol *message* aus.

Argumente:

int, **float** oder **symbol** Initialisierungswert.

outlet:

int, **float** oder **symbol**
Gibt bei Empfang von bang *message* über INIT.grainy den Initialisierungswert aus.

receive:

r INIT.grainy
Empfängt bang *message* zur Initialisierung.

GRAINYkey

Sendet bang *message* für Taste der NeXT-Tastatur, wenn aktiv.

Argumente:

int ASCII-Kode einer Taste der NeXT-Tastatur.

outlet:

bang
Bei Niederdrücken der zugeordneten Taste der NeXT-Tastatur.

receive:

r switchKEY.grainy int
Aktiviert (int != 0) und deaktiviert (int == 0) NeXT-Tastatur für alle GRAINYkey-Objekte.

Literatur

G. De Poli, A.Piccialli
Pitch-Synchronous Granular Synthesis.
In *Representations of Musical Signals.*
Cambridge, Massachusetts: The MIT Press, pp. 187-219

Miller Puckette.
FTS: A Real-Time Monitor for Multiprocessor Music.
Computer Music Journal, Vol.15, Nr.3.

Miller Puckette.
CPOS: A Real-Time Operating System for IRCAM Musical Workstation.
Computer Music Journal, Vol.15, Nr.3.

Miller Puckette.
Combining Event and Signal Processing in the
MAX Graphical Programming Environment.
Computer Music Journal, Vol.15, Nr.3.

Curtis Roads.
Introduction to Granular Synthesis.
Computer Music Journal, Vol.12, Nr.2.

Curtis Roads.
Asynchronous Granular Synthesis.
In *Representations of Musical Signals.*
Cambridge, Massachusetts: The MIT Press, pp. 143-186

Curtis Roads (Editor).
IRCAM MAX Manual (Beta Version).
IRCAM, 1993.

Barry Truax.
Real-Time Granular Synthesis With a Digital Signal Processor.
Computer Music Journal, Vol.12, Nr.2.

Eric Viara.
The Architecture of the IRCAM Musical Workstation.
Computer Music Journal, Vol.15, Nr.3.

David Zicarelli.
Writing External Objects for Max.
Opcode Systems, Inc., 1991.