České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Pavel Lieberzeit**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Granularní zvukový syntezátor**

Pokyny pro vypracování:

Prozkoumejte oblast granulární syntézy, tj. generování zvukových signálů "rozbíjením" vstupních vzorků na krátké úseky (grains) a jejich spojováním v jiném pořadí, počtu či časování. Prozkoumejte existující platformy pro realizaci virtuálních syntezátorů v prostředích DAW (digital audio workstations), např. VST.

Navrhněte vlastní techniku syntézy zvuku založenou na metodách granulární syntézy, tuto techniku implementujte coby virtuální syntézátor na zvolené počítačové platformě a realizovaný syntezátor podrobte uživatelskému testu.

Design, implementaci, i její rozsah konzultujte s vedoucím práce.

Seznam odborné literatury:

Rick Snoman (2008) Dance Music Manual: Tools, Toys, and Techniques Paperback. Focal Press.
Curtis Roads (1996) The Computer Music Tutorial. MIT Press

Vedoucí: Ing. Adam Sporka, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

[redacted]

prof. Ing. Jiří Zára, CSc.
vedoucí katedry

L.S.

[redacted]

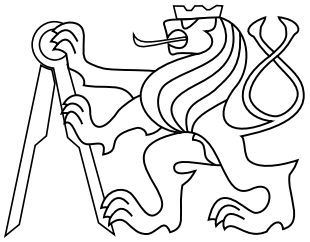
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 3. 2015

Diplomová práce

Granulární zvukový syntezátor

Pavel Lieberzeit



Květen 2015

Vedoucí práce: Ing. Adam Sporka, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra počítačové grafiky a interakce

Poděkování

Rád bych poděkoval zejména Ing. Adamovi Sporkovi, Ph.D. za jeho neutuchající nadšení, podporu a nezměrné množství rad. Dále chci poděkovat také své rodině, blízkým kamarádům a kolegům, kteří měli se mnou pochopení a trpělivost během psaní této práce. V neposlední řadě bych také rád poděkoval nezměrnému množství zkonzumovaného kofeinu, bez kterého by tato práce nikdy nemohla vzniknout.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona 4. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 11. května

Abstrakt

Cílem této práce je navržení vlastní techniky granulární syntézy a její implementace jako plugin použitelný v Digital Audio Workstation prostředí. Navržená technika umožňuje granulární syntézu zvuku za využití semi-náhodně vybíraných vzorků z dodaného vstupního zvuku. Plugin byl implementován ve formátu VSTi 2 a úspěšně používá navrženou techniku. Výsledný plugin lze využívat pro skládání hudby, kde je vyhledáván jistý prvek náhody při výběru zvuku.

Klíčová slova

granulární syntéza; VST; Virtual Studio Technology; DAW; Digital Audio Workstation; syntéza zvuku; JUCE

Abstract

The goal of this thesis was design of own technique of granular synthesis and its implementation as a plugin deployable into the Digital Audio Workstation environment. The designed technique allows the user to perform granular sound synthesis using semi-randomly chosen samples from a supplied input sound. The plugin was implemented in the VSTi 2 format and it successfully uses the designed technique. The result plugin can be used for composing of music where a certain degree of randomness in the selection of sound is desired.

Keywords

granular synthesis; VST; Virtual Studio Technology; DAW; Digital Audio Workstation; sound synthesis; JUCE

Obsah

1 Úvod **1**

2 Vybrané metody syntézy zvuku **3**
2.1 Aditivní syntéza . . . 3
2.2 Subtraktivní syntéza . . . 3
2.3 Amplitudová modulace . . . 4
2.4 Frekvenční modulace . . . 4
2.5 Ring modulace . . . 4
2.6 Wavetable syntéza . . . 4
2.7 Syntéza pomocí fyzikálního modelování . . . 5
2.8 Granulární syntéza . . . 5

3 Granulární syntéza **7**
3.1 Obsah zrnka . . . 7
3.2 Obálka zrnka . . . 8
3.2.1 Lichoběžníkovité . . . 9
3.2.2 Zakřivený tvar . . . 9
3.2.3 Komplexní obálka . . . 9

4 Metody granulární syntézy **11**
4.1 Třídění podle tvorby zrnka . . . 11
4.1.1 Sinusoida . . . 11
4.1.2 Impulzní odezva . . . 11
4.1.3 Vzorky zvuků . . . 11
4.2 Třídění podle organizace zrnek . . . 11
4.2.1 Synchronní metody . . . 12
4.2.2 Asynchronní metody . . . 14
4.3 Mřížky . . . 14
4.4 FOF syntéza . . . 17
4.5 VOSIM . . . 18
4.6 Kvazi-synchronní syntéza . . . 20
4.7 Výškově synchronní syntéza . . . 20
4.8 Pulsarová syntéza . . . 21

5 Digital Audio Workstation a formáty audio pluginů **27**
5.1 Digital Audio Workstation . . . 27
5.2 Formáty audio pluginů . . . 28
5.2.1 Virtual Studio Technology . . . 30
5.2.2 Audio Units . . . 30
5.2.3 Real Time AudioSuite a Avid Audio eXtension . . . 30

6 Design audio pluginu využívající techniky granulární syntézy **33**
6.1 Motivace . . . 33
6.2 Popis techniky granulární syntézy . . . 33
6.3 Vstupní parametry . . . 36
6.4 Návrh ovládání . . . 37
6.5 High-level design pluginu . . . 38

7 Implementace audio pluginu využívající techniky granulární syntézy 45
7.1 Volba formátu . . . 45
7.2 Detaily implementace . . . 46
7.2.1 Správce hlasů . . . 47
7.2.2 Zrnko . . . 47
7.2.3 Generátor zrnek . . . 48
7.2.4 Hlas . . . 49
7.2.5 Editor . . . 49
7.2.6 Utils . . . 50
7.2.7 Hlavní modul . . . 50
7.2.8 Krátká ukázka . . . 50

8 Testování implementace audio pluginu 53
8.1 Konfigurace testování . . . 53
8.2 Plán testování . . . 53
8.2.1 Popis a ovládání . . . 53
8.2.2 Seznam otázek . . . 54
8.3 Průběh a výsledky testování . . . 54
8.4 Analýza výsledků testování . . . 55

9 Revize implementace a designu po testování 57
9.1 Úprava designu . . . 57
9.2 Úprava implementace . . . 58

10 Závěr 61
10.1 Budoucí práce . . . 62

Přílohy

A Uživatelský manuál k syntezátoru 63

B Obsah přiloženého DVD 65

Literatura 67

Seznam obrázků

1 Ukázka aditivní syntézy [2]. 3
2 Ukázka amplitudové a frekvenční modulace [3]. 4
3 Rozdělení vlny u wavetable syntézy [2]. 5

4 Ukázka obálky zrnka [14]. 8
5 Ukázky obálek lichoběžníkových tvaru. 9
6 Ukázky obálek zakřiveného tvaru. 10
7 Ukázka komplexní obálky [14]. 10

8 Ilustrace dvou proudů v jedné textuře [14]. 12
9 Ukázka několika oblaků v čase [16]. 12
10 Typy oblaků a) cumulus b) glissandi c) stratus [11]. 13
11 Regiony frekvence [9]. 15
12 Regiony intenzity [9]. 15
13 Regiony hustoty [9]. 16
14 Příklad mřížky [9]. 16
15 Způsob navázání proměnných na MTP [9]. 16
16 Matice podmíněných přechodů mezi mřížkami A až H [9]. 17
17 Trubka hrající dvě různé noty, čistou kvartu od sebe. Formanty (pevně dané resonance) ale zůstávají na stejných místech [18]. 18
18 VOSIM funkce. N=8, b=0.85, T=10 milisekund [19]. 19
19 Ukázka kvazi-synchronní syntézy. Je možné pozorovat mírně proměnnou prodlevu mezi zrnky [11]. 20
20 Proud překrývajících se zrnek ve výškově synchronní syntéze. *Hop size* je vzdálenost mezi následujícími zrnky [11]. 21
21 Pulsarová syntéza [23]. 22
22 Typické pulsaretové vlny. a) Sinusoida b) Vícecyklová sinusoida c) Band-limited puls d) Tlumená multicyklová sinusoida e) Vlna kosmického pulsaru neutronové hvězdy Vela X-1 [22]. 22
23 Ukázka vlivů různých obálek na spektrum pulsarů. Výše - frekvence-versus-čas sonogramy jednotlivých pulsarů. Fundamentální frekvence = 12 Hz, formantová frekvence = 500 Hz. Sonogramy jsou založené na fast fourier transformation s 1024 body a užitím Hannových oken. Níže – sonogramy vyrobené a) obdélníkovou obálkou b) exponenciálním útlumem c) Gaussovou křivkou. Spektrum na dB škále. [22]. 23
24 Efekt konvoluce na pulsarový sled. a) infrasonický pulsarový sled s proměnnou fundamentální a formantovou frekvencí b) Navzorkovaný zvuk: Italské „qui“ c) konvoluce a) a b) [22]. 25

25 Ukázka DAW REAPER. 29

26 Znázornění zobrazení MIDI klávesy do zvukové stopy společně s pravděpodobností výběru segmentu jako začátku zrnka při zmáčknutí klávesy . 34
27 Hannova obálka [31] 35

28 Znázornění konstantní úrovně hlasitosti při 50% překryvu u Hannovy obálky [32] 35
29 Znázornění procesu tvorby zrnka s potřebnými parametry uvedenými u jednotlivých procesů 40
30 Znázornění procesu spuštění přehrávání noty hlasem 41
31 Znázornění procesu přehrávání noty hlasem 42
32 Graf vztahů modulů v pluginu 44

33 Normované normální rozdělení [37] 49
34 Ukázka GUI pluginu po implementaci 51

35 Ukázka GUI pluginu po revizi implementace 60

1 Úvod

Vymýšlení způsobů, jakými tvořit nové zvuky, byla pro lidi vždy zajímavá otázka. Většinu své historie byli vázaní na fyzické nástroje, ale počátkem dvacátého století se jim otevřely dveře elektronické syntézy zvuku a tím možnost upravovat existující zvuky a vytvářet nové. Jak byly vymýšleny nové a nové možnosti upravování zvuku, vyskytla se i myšlenka neupravovat zvuk po dlouhých spojitých kusech, ale zkusit s ním zacházet po malých, pár desítek milisekund dlouhých, úsecích. A tak se zrodila granulární syntéza.

Granulární syntéza dovoluje upravovat všechny možné aspekty zvuku, například rychlost, výšku, barvu a mnoho jiných. Také dovoluje jednotlivá „zrnka“ zvuku různě přehazovat a tím vytvářet chaotičtěji znějící zvuky. Pokud však nechceme úplně chaotickou změť zvuků, ale nemáme zájem ani o běžné, uspořádané zvuky, musíme navrhnout vlastní techniku granulární syntézy tak, aby byla kontrolovaná i náhodná zároveň.

Tím se dostáváme k samotnému tématu a cílům práce. Konečným výsledkem celé práce by měl syntezátor implementující vlastní techniku granulární syntézy, která poskytne určitou náhodnost, a zároveň dovolí kontrolovat výsledný zvuk. Technika bude využívat uživatelem určený vstupní zvuk. Vhodný formát pro implementaci syntezátoru je plugin do softwarových prostředí pro digitální práci se zvukem, takzvaných Digital Audio Workstation (DAW).

Abychom tento úkol mohli splnit, musíme nejprve v průzkumné části práce prozkoumat existující druhy syntézy a zejména techniky granulární syntézy. Také musíme prozkoumat specifika DAW a zvyklosti jejich uživatelů, aby náš syntezátor nebyl náročný na použití. V neposlední řadě je nutný průzkum používaných formátů pluginů.

Po dokončení průzkumu využijeme nabyté informace při designu naší techniky granulární syntézy a syntezátoru tuto techniku využívající. Následně nadesignovaný syntezátor naimplementujeme. Poté ho podrobíme uživatelskému testu. Pokud to budou výsledky testu žádat, provedeme revizi designu a implementace.

Výsledný syntezátor by měl být dostatečně jednoduchý, aby ho mohl používat i začátečník, a zároveň poskytovat dostatečnou míru volnosti, aby se profesionál při práci s ním necítil omezený.

2 Vybrané metody syntézy zvuku

Lidstvo bylo a je během celé své existence provázeno vytvářením zvuků. S postupem času se také v tomto aktu zlepšovalo: od obyčejných skřeků a bouchání předměty přes mluvu a zpěv až po složité nástroje, například housle, varhany a celé orchestry. Všechny tyto prostředky nicméně trpí vázaností na fyzikální vlastnosti daného předmětu – mohou vyluzovat pouze určitou škálu zvuků danou materiálem, způsobem vyluzování zvuku a podobě.

K začátku dvacátého století však došlo k průlomu. Tím bylo objevení možnosti elektronické syntézy zvuku, Telharmonium Thaddeuse Cahilla [1], a zpřístupnění potenciálu vytvoření jakéhokoli představitelného zvuku. V to je zahrnuta i možnost úpravy již existujícího zvuku. Celý tento nový svět hudby se začal rozvíjet, a společně s potřebami vyrábět složitější, věrohodnější[1] a zajímavější zvuky byly vytvářeny i nové metody syntézy. Následuje výčet a popis vybraných z nich:

2.1 Aditivní syntéza

Velice účinná metoda syntézy a zároveň jednoduchá na pochopení. Funguje na bázi sdružování dvou nebo více signálů do jednoho složitějšího signálu (Obr. 1).

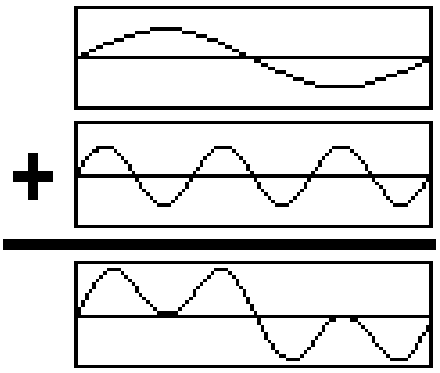
Její implementace obvody je obecně pokládána za složitou, a je proto v dnešní době přenechávána digitálním syntezátorům.

Aditivní syntéza by teoreticky měla být schopná vytvořit jakýkoliv zvuk. Bohužel, přírodní zvuky jsou velmi složité a vyžadují mnoho skládání, abychom je cíleně reprodukovali.

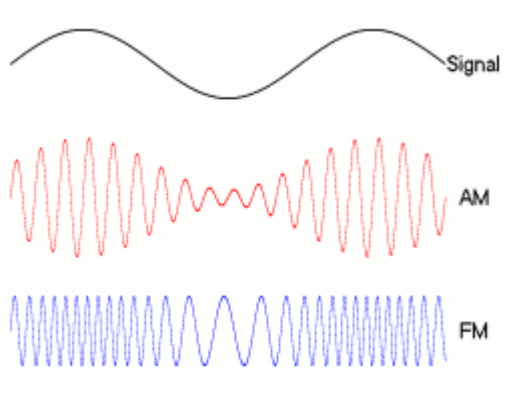
2.2 Subtraktivní syntéza

Sesterská metoda metody aditivní – pouze místo přičítání signály odčítáme. V praxi tedy nejde o nic jiného než aplikaci různých filtrů na zvukovou stopu, což se dá poměrně

[1] Věrohodnější ve smyslu přiblížení se zvukům vyskytující se v přírodě – mimo elektronické zvuky.



Obrázek 1 Ukázka aditivní syntézy [2].



Obrázek 2 Ukázka amplitudové a frekvenční modulace [3].

snadno implementovat analogově. Mimo jiné se tato metoda dá použít na relativně věrnou aproximaci skutečných nástrojů, především strunných.

2.3 Amplitudová modulace

Amplitudová modulace (AM) je důvěrně známa z jejího využití v radiokomunikaci. Princip, na kterém pracuje, používá dvě vlny: Jednu nosnou a jednu modulační. Modulační vlna, která u AM je unipolární (má pouze nezáporné hodnoty), je použita jako funkce, podle které je měněna amplituda nosné vlny. Výsledná vlna má tedy proměnlivou hlasitost (tremolo). Příklad je možné vidět na Obr. 2 (nosná vlna je předpokládaná sinusová o vyšší frekvenci než modulační).

2.4 Frekvenční modulace

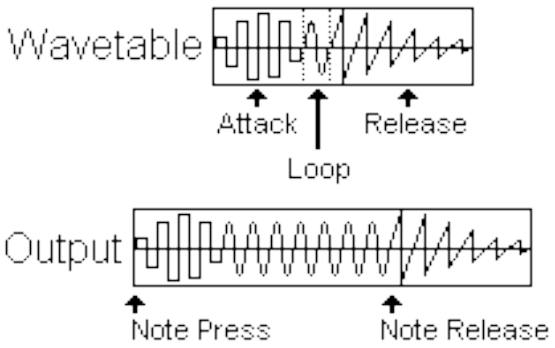
Frekvenční modulace (FM) je v principu podobná amplitudové modulaci, pouze místo amplitudy se mění frekvence. Výsledná vlna má tedy proměnlivou frekvenci (vibrato). Příklad je možné vidět na Obr. 2 (nosná vlna je předpokládaná sinusová o vyšší frekvenci než modulační).

2.5 Ring modulace

Ring modulace (RM), stejně jako předešlé dva druhy modulace, pracuje se dvěma vlnami. Jedná se o druh amplitudové modulace, kde modulační vlna je bipolární (obsahuje jak kladné, tak záporné hodnoty). Výsledná vlna obsahuje součet a rozdíl nosné a modulační vlny, bez přítomnosti původní nosné. V digitálním světě je to ještě jednodušší – výsledek se rovná součinu obou původních vln. Tato metoda většinou produkuje zvonivé a nepřirozené zvuky. Jeden z velmi známých příkladů použití je hlas Daleků z britského sci-fi seriálu Doctor Who [4].

2.6 Wavetable syntéza

Tato metoda funguje na bázi vyhledávání v tabulkách. Syntezátor má v sobě uloženo mnoho různých vln, od jednoduchých (sinusoidy, trojúhelníkové), až po složité, například skutečných nástrojů. Každou vlnu také může mít rozdělenou na několik segmentů:



Obrázek 3 Rozdělení vlny u wavetable syntézy [2].

Attack, Loop a Release. Attack část se zahraje při nástupu vlny; Loop je opakován, dokud má vlna hrát; Release představuje dozvuk.

Kromě seznamu vln také wavetable syntéza disponuje metodou, jak mezi jednotlivými vlnami přecházet. Může jít o jednoduché lineární prolínání, nebo i o složitější metody navrhované například Hornerem [5]. Dále může wavetable syntéza využívat i další metody syntézy, například amplitudovou nebo frekvenční, aby zmírnila repetitivnost výstupní vlny.

Mezi její výhody patří nízká náročnost na paměť a výpočetní sílu a zároveň rozumná kvalita.

2.7 Syntéza pomocí fyzikálního modelování

Matematicky velmi náročná metoda syntézy zvuku. Ve své podstatě jde o modelování fyzikálních vlastností skutečných nástrojů pomocí matematických rovnic. Kupříkladu pro model bubnu je potřeba popsat vlastnosti blány (tuhost, hustotu), uchycení k tělu bubnu, samotné tělo a další možné parametry. Z toho všeho po dosazení do modelu a výpočtu vyjde generovaná zvuková vlna. Parametry se liší nástroj od nástroje (kytara se chová jinak než buben. Pokud na ni není bubnováno). Jak je vidět, skutečně se jedná o velmi složitou, ale na druhou stranu věrohodnou metodu. Její použití v reálném čase však není snadné bez určité relaxace přesnosti modelu a výpočtu.

Tuto metodu využívá například formantová (FOF) syntéza, která je používána kupříkladu v syntéze řeči.

2.8 Granulární syntéza

Touto metodou se bude podrobně zabývat následující kapitola.

3 Granulární syntéza

Granulární syntéza je v základu podobná wavetable syntéze, ale v mnohém se liší. Ve velmi stručné verzi by vysvětlení jádra granulární syntézy mohlo vypadat zhruba následovně: Vezmou se krátké granulky zvuku (o délce zhruba 10 milisekund až 50 milisekund) a určitým způsobem se poskládají dohromady jak časově za sebe, tak ve vrstvách. Tímto způsobem je možno vytvářet nové zvuky i barvy zvuků, měnit výšku a rychlost audio vzorků nezávisle na sobě, to vše pod plnou kontrolou skladatele (a v reálném čase, při přístupu k dostatečně výkonnému počítači). Ten může ovlivňovat skladbu jak na mikro úrovni změnami jednotlivých granulek, tak na makro úrovni volbou, jak granulky skládat dohromady. Bohužel, už z tohoto popisu je patrné, že je potřeba dávat pozor na extrémní množství parametrů, a tak až do rozsáhlejšího použití počítačů nebyla tato metoda rozšířena.

Granulární syntéza jako idea existuje už poměrně dlouhou dobu. První známky nápadu se objevují na přelomu 17. století v poznámkách Isaaca Beeckmana [6], kde popisuje myšlenku, že zvuk je složen z malých částeček – globulí. Sám byl zastáncem vznikající myšlenky atomismu.

Než však byla tato idea rozvinuta, uběhlo mnoho let. Ve čtyřicátých letech dvacátého století Dennis Gabor navrhl základní matematické modely a teorie [7][8], které byly dále rozvíjeny dalšími autory. Také ze svých výzkumů lidského sluchu zjistil, že aby člověk rozeznal tón od šumu, musí tón trvat minimálně 10 milisekund. Dalšími poznatky jsou minimální doby na rozpoznání změny amplitudy (alespoň 21 milisekund) a změny frekvence (alespoň dvakrát tak dlouho co bylo potřeba na rozpoznání předešlé frekvence), s tím, že tato doba je závislá na frekvenci – čím nižší frekvence, tím rychlejší rozpoznávání [7]. Všechny tyto poznatky jsou v granulární syntéze využívány pro stanovení přibližné délky zrnka.

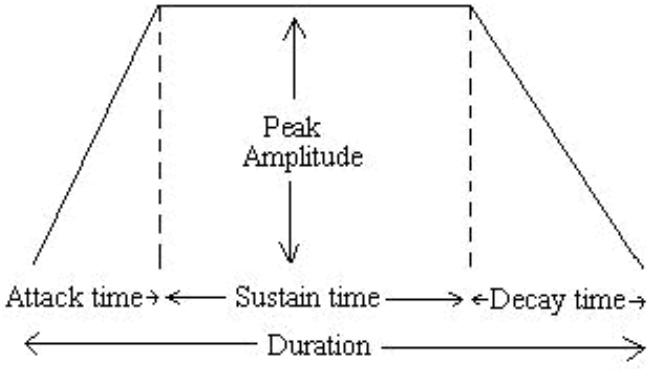
První praktickou ukázkou je skladba Analogique A-B od Iannise Xenakise z roku 1959 [9]. Tato skladba je pro smyčcový orchestr a magnetofonovou pásku. Granulární syntéza je zde prováděna čistě ručně nastříháním a poslepováním pásky. Xenakis také dále rozvíjel teorii granulární syntézy.

Následně na něj navázali další autoři, například Curtis Roads ([10], [11] a mnoho dalších) nebo Barry Truax ([12] a mnoho dalších). Truax také vyrobil první program pro granulární syntézu v reálném čase a založil na ní svoji skladbu Riverrun [13].

Vraťme se zpět k technickým detailům. Bylo zmíněno, že v této metodě syntézy je celá skladba složená z malých zrnek zvuku. Každé zrnko se skládá ze svého obsahu a obálky.

3.1 Obsah zrnka

Obsahem zrnka není nic jiného než zvuková vlna. Může jí být naprosto cokoliv. Může to být jednoduchá vygenerovaná sinusoida, může to být zachycený zvukový vzorek. Co se týče zvukového vzorek, jedná se o poměrně častou techniku, kde se vezme existující skladba a z ní (celé nebo jejích částí) se vytvoří jednotlivá zrnka.



Obrázek 4 Ukázka obálky zrnka [14].

3.2 Obálka zrnka

Důležitou částí zrnka je jeho obálka. Obálka u zrnka určuje jeho délku a amplitudu. Jedná se tedy o využití amplitudové modulace, kde jako nosná vlna je používán obsah zrnka a jako modulační vlna obsah obálky. Obálka typicky vypadá jako na Obr. 4. Z tohoto obrázku můžeme odvodit hlavní složky, které definují obálku [14]. Jak vidíme, tato se liší od klasické ADSR obálky[1].

Attack time (doba nástupu)
Doba než obálka dosáhne maximální amplitudy. Postupné zvyšování amplitudy zrnka je důležitá část v procesu skládání zrnek. Pokud bychom najednou přešli z ticha do plné amplitudy, ostrým přechodem by byly vyvolány signály na okolních frekvencí (sidebands) a ozval by se hlasitý praskavý zvuk. Pokud tedy tento praskavý efekt není žádán, je na místě nastavit příhodnou dobu nástupu.

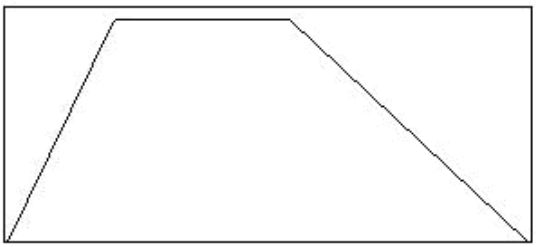
Sustain time (doba držení)
Doba, po kterou je držena maximální amplituda. Tento údaj z velké míry určuje, jak bude výsledný zvuk znít. Krátkou dobou držení je vyvoláván pocit bicích nástrojů, delší časy zase dodávají výsledné textuře více melodický nádech.

Decay time (doba ústupu)
Analogická k době nástupu. Jedná se o dobu, než se amplituda dostane z maxima zpět na nulu. Není nutné, aby byla stejně dlouhá jako doba nástupu.

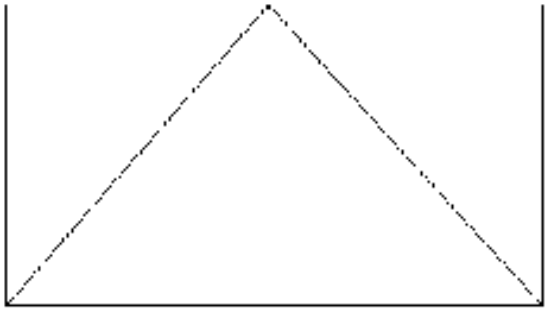
Peak amplitude (maximální amplituda)
Určuje, jaké maximální amplitudy obsah zrnka může dosáhnout. Pohybuje se mezi hodnotami 0% až 100% (respektive 0 až 1). Jinými slovy, na kolik je obsahu zrnka dovoleno dosáhnout své maximální amplitudy.

Duration (délka)
Délka celého zrnka. Výzkumy ukázaly, že je dobré se pohybovat v rozmezí zhruba 10 milisekund až 50 milisekund [7]. Dá se říci, že čím kratší délka, tím více se zrnko a následná textura z nich složená blíží šumu. I toto může být žádané chování, například pokud chceme do výsledného zvuku přivést více frekvencí. Na druhou stranu delší zrnka v sobě obsahují více informací o zvuku, ze kterého je zrnko vytvořeno, a tak tato zrnka dodávají větší pocit melodie. Pokud bychom ale délku prodloužili až příliš, vzniknou slyšitelně slepované časové úseky.

[1] Attack, Decay, Sustain, Release obálka typicky používaná pro celé noty v syntetizérech. V Attack fázi stoupá hlasitost k maximu, následně v Decay klesá k Sustain úrovni, na té se drží, dokud je nota přehrávána, a v následné Release fázi klesá na nulu.



a) Ukázka lichoběžníkového tvaru obálky [14].



b) Ukázka lichoběžníkového, konkrétně trojúhelníkového tvaru obálky [14].

Obrázek 5 Ukázky obálek lichoběžníkových tvaru.

Těchto pět údajů lze také hromadně popsat funkcí popisující tvar obálky. Tento tvar může být libovolný a dá se rozdělit několika kategorií.

3.2.1 Lichoběžníkovité

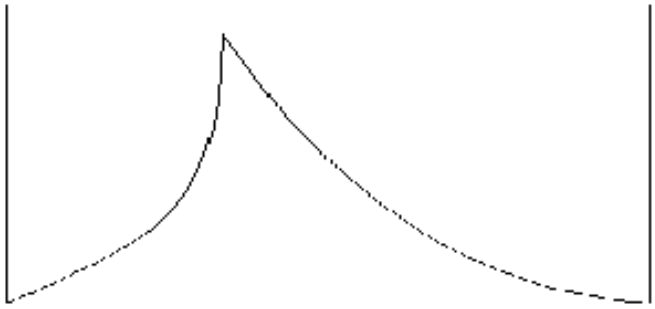
Tyto tvary jsou jednoduché na výrobu. Vyznačují se však ostrými rohy v místech, kdy se přechází z jednoho segmentu do druhého (například z nástupu do držení). To může vyvolávat pocity ostrosti ve výsledném zvuku. Trojúhelníkový tvar je speciální případ lichoběžníku.

3.2.2 Zakřivený tvar

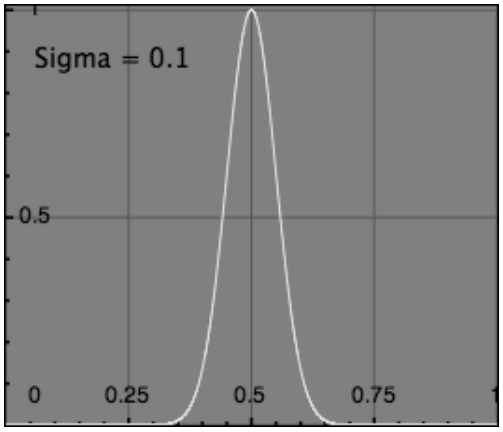
Tvar, který nejde vyjádřit čistě rovnými segmenty. Jejich tvar může vyplývat z různých statistických křivek, například Gaussovy, nebo různých exponenciál a goniometrických funkcí.

3.2.3 Komplexní obálka

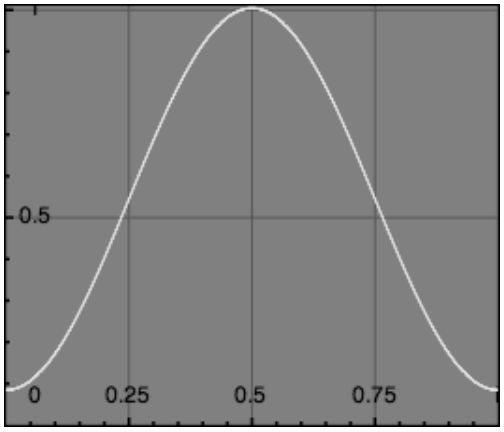
Obálky, které se nedají popsat nijak jednoduše. Mohou být složité na vytvoření, ale ve výsledku mohou produkovat nezvyklé a zajímavé zvuky.



a) Obálka složená ze dvou exponenciálních křivek [14].

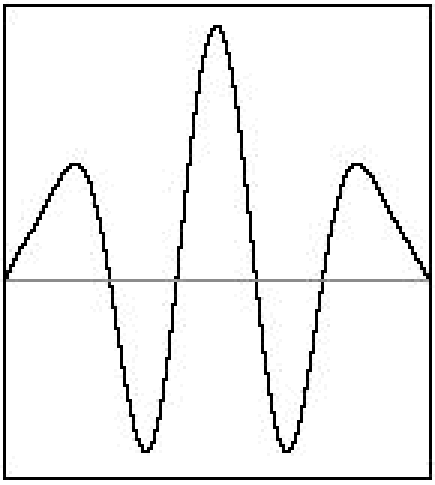


b) Gaussova křivka jako obálka [15].



c) Hammingova křivka jako obálka [15].

Obrázek 6 Ukázky obálek zakřiveného tvaru.



Obrázek 7 Ukázka komplexní obálky [14].

4 Metody granulární syntézy

Tato kapitola se zabývá jednotlivými metodami granulární syntézy. Dají se rozlišit dva základní druhy jejich klasifikace: Podle způsobu, jakým jsou vytvářeny jednotlivá zrnka, nebo podle způsobu, jakým jsou zrnka organizována. Nejprve uvedeme způsoby klasifikace a které metody kam spadají. Následně popíšeme jednotlivé metody.

4.1 Třídění podle tvorby zrnka

Proces tvorby zrnek má velký vliv na výsledný zvuk. Způsob, jakým je vytvořen obsah zrnka, rozhoduje především o spektru frekvencí ve výsledném zvuku, zatímco způsob tvorby obálky ovlivňuje z velké části výslednou barvu zvuku.

Následující vybrané metody pro tvorbu zrnka jsou zaměřené především na tvorbu obsahu zrnka.

4.1.1 Sinusoida

V základu metody spadající do této kategorie používají základní sinusovou vlnu. Následně ji mohou poupravit, jako je to například u **formantové (FOF) syntézy**.

4.1.2 Impulzní odezva

Na dodaném vstupním zvuku je provedena analýza výšky tónu a každá perioda výšky je brána jako samostatné zrnko. Následně se ze spektrální analýzy jednotlivých zrnek zjistí impulzní odezva, která poslouží pro nastavení FIR (Finite impulse response) filtrů pro následnou resyntézu.

Tento proces používá například **výškově synchronní (pitch-synchronous) syntéza**.

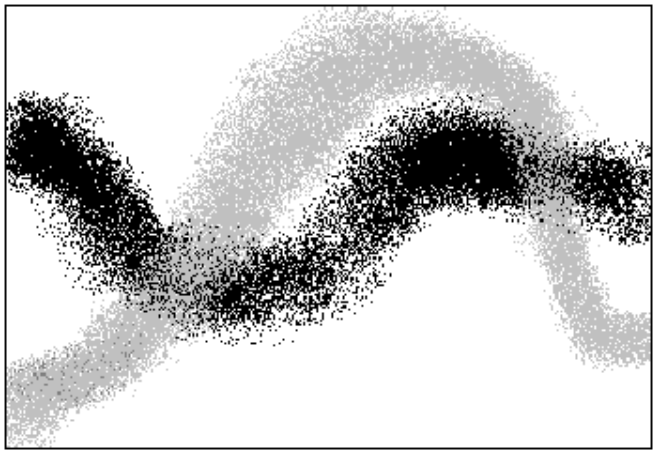
4.1.3 Vzorky zvuků

Tyto metody svoje zrnka vytvářejí z dodaného zvuku. Jednotlivá zrnka jsou tedy vzorky vstupního zvuku a berou si z něj jeho charakteristickou barvu. Zde se hodí používat především zvuky, u kterých nebude rozbita žádná komplexní charakteristika, například melodie, jejich rozkouskováním. Příklady budiž použití kapek vody pro generování zvuků proudů vody, padání střepů pro zvuky rozbíjeného skla a podobně. Je samozřejmě možné použít jakýkoli zvuk, ale ne každý zní ve výsledku lákavě.

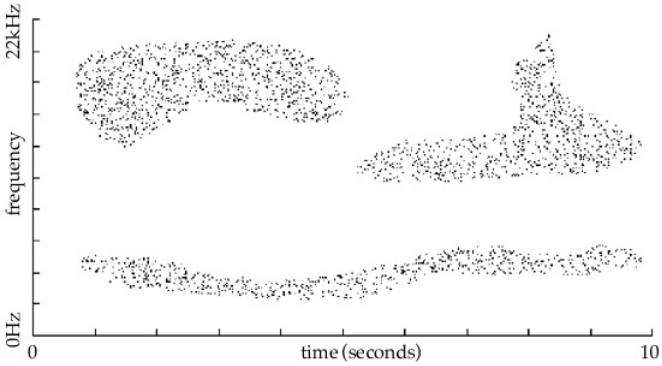
Metody z této kategorie jsou například **FOG syntéza** nebo **pulsarová syntéza**.

4.2 Třídění podle organizace zrnek

Jak bude výsledný zvuk znít do značné míry ovlivňuje způsob, jakým zrnka skládáme za sebe. Pokud bychom například z dostupných zrnek použili jedno, mnohokrát nakopírované, a pokládali ho v čase za sebe s konstantními prodlevami, dostali bychom stále



Obrázek 8 Ilustrace dvou proudů v jedné textuře [14].



Obrázek 9 Ukázka několika oblaků v čase [16].

stejně znějící zvuk. Jeho „sekanost“ by záležela na velikosti prodlev. Ve většině případů však používáme více různých zrnek, která se mohou i různě překrývat (k získání výsledného zvuku z překrytých segmentů se nejčastěji používá aditivní syntéza).

Kromě organizace zrnek do jedné vrstvy v čase se ještě používá technika překrývajících se proudů (streams). Jedná se v podstatě o organizaci zrnek do skupin podle požadovaného chování. Je možné si jednotlivé proudy ve zvukové textuře představit jako harmonie plynoucí texturou, podobně jako skladby mohou mít různé nástroje, každý hrající jinou melodii. Obr. 8 ilustruje tuto ideu.

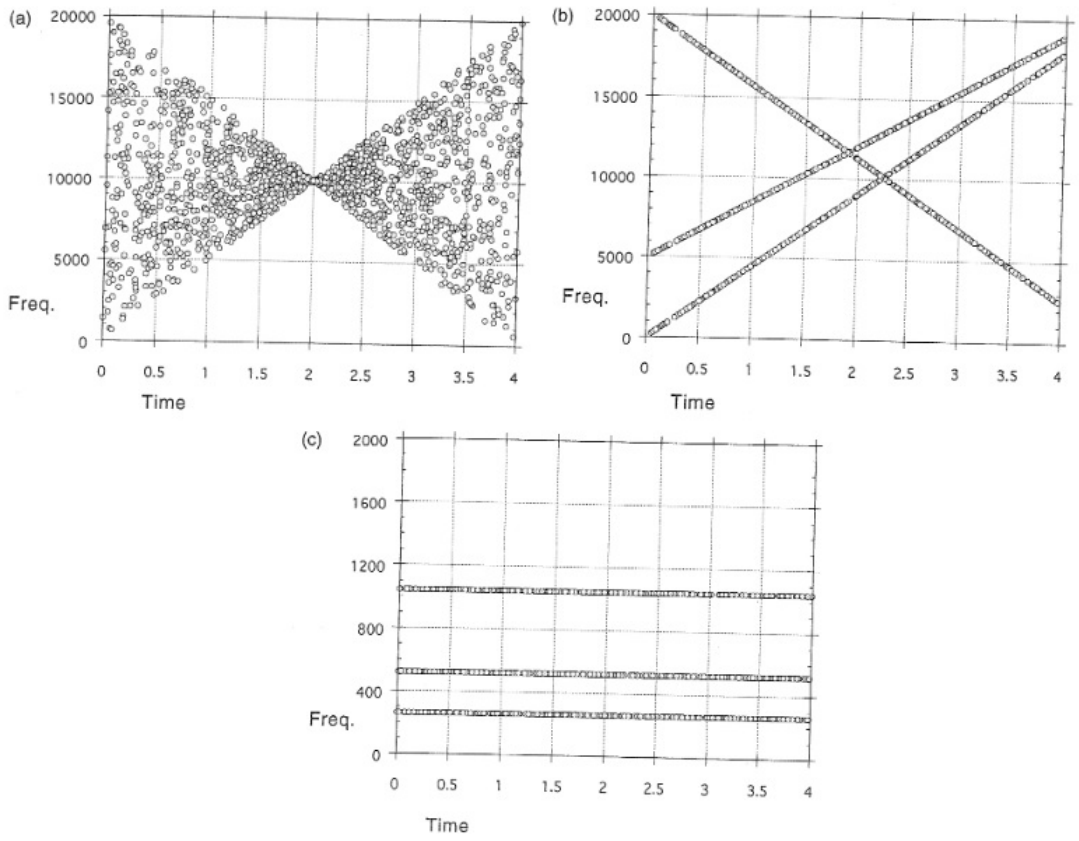
Používá se i technika oblaků (clouds). Oblaky jsou v jistém ohledu podobné proudům, ale vyskytují se každý v určitém čase. Obr. 9 ukazuje, jak by mohlo vypadat rozmístění oblaků v čase a frekvenci. Obr. 10 ukazuje různé typy oblaků podle jejich přibližného uspořádání frekvence v čase.

Ve výsledku se všechny proudy nebo oblaky složí do výsledného zvuku. Opět nejčastěji aditivní syntézou.

Co se týče kategorizace metod granulární syntézy podle organizace zrnek, nejčastěji se uvádí dvě kategorie: Synchronní a asynchronní. Jedná se o popsání rozdílů ve způsobu nastavování prodlevy mezi jednotlivými zrnky v jednom proudu.

4.2.1 Synchronní metody

U synchronních metod jsou všechny zrnka oddělena stejným množstvím času, případně je tento čas dán lineárním vztahem. V každém případě je prodleva mezi zrnky deter-



Obrázek 10 Typy oblaků a) cumulus b) glissandi c) stratus [11].

ministicky odvoditelná [11].

Tyto metody používají několik parametrů, zejména [14]:

Amplituda
Je možné kontrolovat amplitudu pro celou zvukovou texturu, podobně jako lze ovládat amplitudu v jednotlivých zrnkách.

Hustota zrnek
Jeden z nejdůležitějších faktorů ovlivňujících konečný výsledek. Základní efekt je jak hustě jsou zrnka v textuře rozmístěna, ale z toho vyplývá několik vedlejších efektů. Za prvé, vyšší hustoty způsobují vyšší komplexitu barvy zvuku – v textuře se vyskytne více vedlejších frekvencí a formantů obohacující zvuk. Za druhé, vyšší hustota zvyšuje vnímanou výšku zvuku. Také je zvýšena amplituda. A za třetí: nízká hustota u synchronních metod vzbuzuje dojem rytmického zvuku. Čím více se hustota zvyšuje, tím více se ztrácí rytmus a naopak nastupuje větší míra vnímání výšek daných zrnky. Taktéž je vhodné poznamenat, že je přímý vztah mezi mírou překryvu, hustotou a délkou zrnek. Vyšší hustota společně s dlouhou délkou způsobí více a delší překryvy [14].

Výška (pitch)
Je možné ovlivňovat výšku tónů ovládáním vzorkovací rychlosti přehrávání zrnek. Častější je nastavovat výšku přímo v zrnkách, ale je možné to dělat i na úrovni textury. A jak bylo zmíněno, výšku lze ovládat i hustotou zrnek.

Inter-onset time, čas mezi zrnky
Jedná se o prodlevu mezi jednotlivými zrnky. Je přímo určena synchronním charak-

terem metod a hustotou zrnek, proto se u synchronních metod používá spíše pro snadnější sledování.

Mezi synchronní metody spadá **FOF syntéza**, **VOSIM**, **kvazi-synchronní granulární syntéza**, nebo **výškově synchronní (pitch-synchronous) syntéza**.

4.2.2 Asynchronní metody

U asynchronních metod není prodleva mezi zrnky pevně daná. Velmi často se do vztahu určujícího prodlevy přidává náhodný prvek nebo nějaký nepřímý vztah. Ve výsledku tedy tyto metody mohou zrnka rozmisťovat do textury naprosto nahodile.

Podobně jako u synchronních metod, i asynchronní jsou ovlivňovány podobnými parametry [14]:

Amplituda
Funguje přesně jako u synchronních metod.

Hustota zrnek
U asynchronních metod určuje relativní hustotu zrnek v čase. Pokud by byla stanovena jako 200 zrnek za sekundu, v dané sekundě bude náhodným nebo statistickým způsobem rozmístěno 200 zrnek. Z tohoto důvodu vyšší hustota, podobně jako u synchronních metod, způsobí spojitější zvuk, ale už nemusí způsobit změnu výšky. Bary Truax pro svoji skladbu Riverrun použil hustoty 1 až 2300 zrnek za sekundu [17].

Výška (pitch)
Tento parametr hraje podobnou roli jako u synchronních metod, pouze není tolik ovlivňován hustotou zrnek.

Inter-onset time, čas mezi zrnky
Jedná se o prodlevu mezi jednotlivými zrnky. Asynchronicity lze dosáhnout nastavením toho parametru pro každé zrnko náhodně, případně dle vybrané statistické metody.

Mezi asynchronní metody spadá **metoda mřížek (screens)**, nebo **pulsarová syntéza**.

Dále se tato kapitola bude věnovat přiblížení zmíněných metod granulární syntézy.

4.3 Mřížky

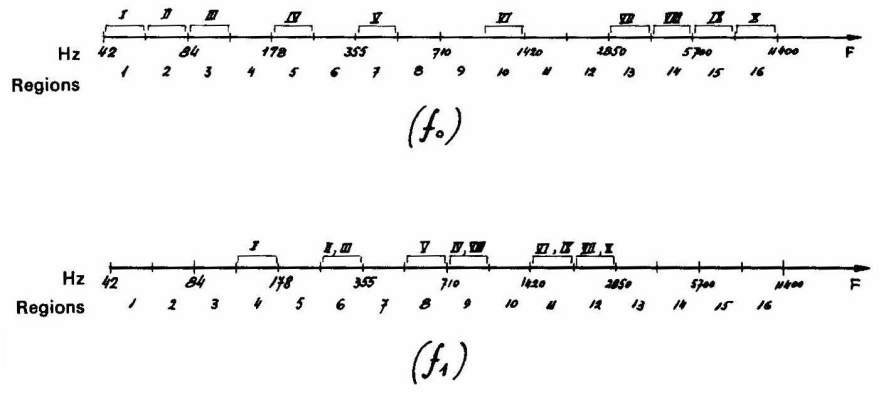
Mřížky (screens) jsou nejstarší známá metoda granulární syntézy použitá v praxi. V letech 1958 – 1959 ji využil Iannis Xenakis pro složení své skladby Analogique B [9].

Metoda mřížek je založena na umístění zrnek do několika mřížek, jejichž osy znázorňují frekvenci, intenzitu a hustotu, a následné přehrání mřížek v pořadí. Všechny operace – umísťování do mřížek, vybírání, která mřížka se bude následně přehrávat – jsou stochasticky určené, konkrétně pomocí Markovových řetězců [9].

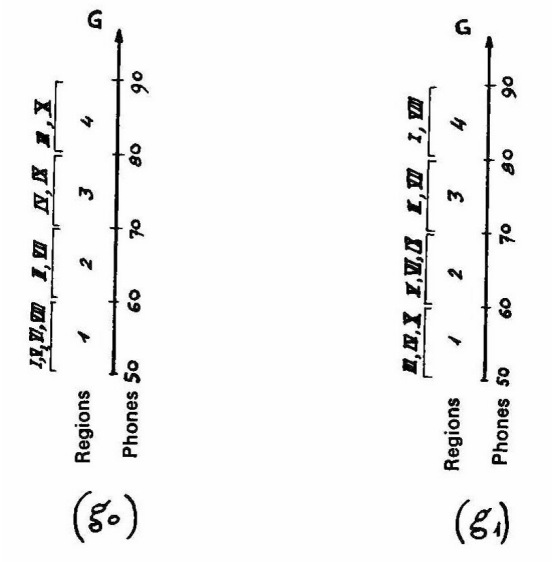
Každá mřížka je určená použitými regiony frekvence (f), intenzity (g) a hustoty (d). Na Obr. 11, Obr. 12, Obr. 13 je možno vidět od každé veličiny dvě různé konfigurace. Rozmístění zrnek v regionech frekvence a intenzity jsou dány Gaussovou distribucí, určené střední hodnotou frekvence a intenzity, podle toho, který region zaplňujeme. Hustota je dána Poissonovou distribucí určenou regionem hustoty děleným časem pro každou mřížku. Čas přehrávání každé mřížky je asi 0,5 sekundy [9].

Každé zrnko je sinusoida konstantní délky (doby) asi 0,04 sekundy.

Mřížek může ve výsledku být libovolný počet, nicméně v uvedeném Xenakisově příkladě, kde máme od frekvence, intenzity a hustoty (tří veličin, kterými je mřížka určená)



Obrázek 11 Regiony frekvence [9].



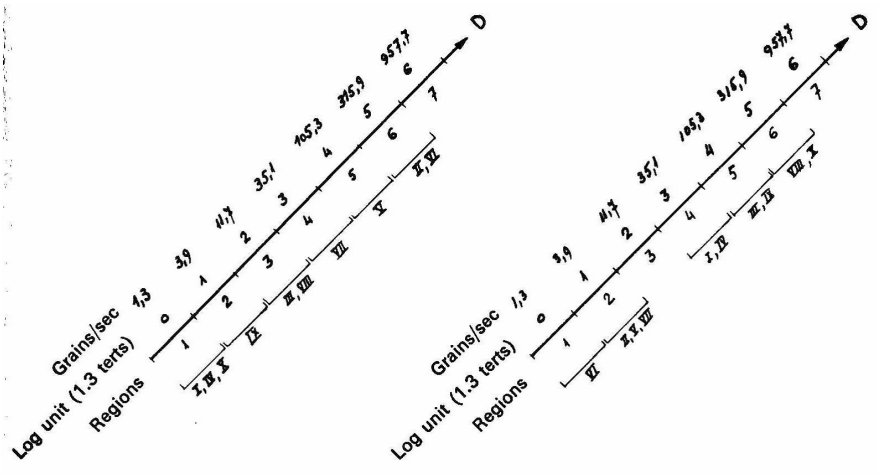
Obrázek 12 Regiony intenzity [9].

pokaždé dvě verze (označované 0 a 1, tedy f_0 a f_1, g_0 a g_1, d_0 a d_1), můžeme vytvořit celkem 8 mřížek, označených A až H. Mřížku A můžeme vidět na Obr. 14.

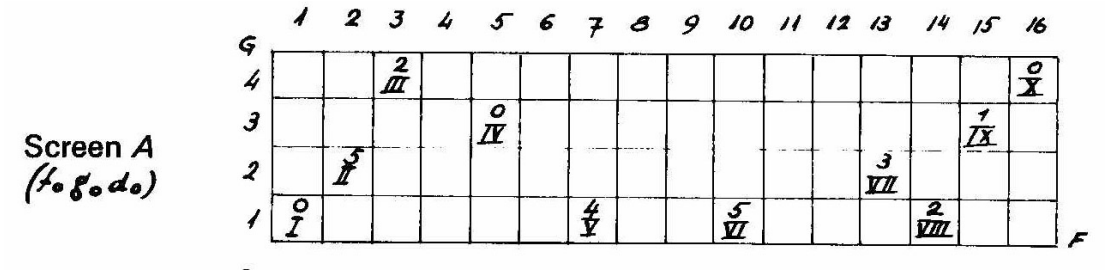
Takto máme definované mřížky, zbývá definovat přechody mezi nimi. K tomu Xenakis používal matice podmíněných přechodů (Matrix of Transitional Probability, MTP). Pro každou veličinu *f*, *g*, *d* definoval dvě různé matice, které vyjadřovaly s jakou pravděpodobností veličina přejde z jednoho svého stavu do druhého (například $f_0 \longrightarrow f_1$ nebo $f_0 \longrightarrow f_0$). Každá tato matice může a nemusí mít různé pravděpodobnosti, nicméně musí být stochastické ve smyslu Markovových řetězců. Pro frekvenci Xenakis stanovuje matice MTPF α a β, pro intenzitu MTPG γ a ε a pro hustotu λ a μ [9].

Každý stav proměnné má k sobě navázanou matici, ze které by se měla použít pravděpodobnost. Způsob navázání uváděný Xenakisem můžeme vidět na Obr. 15.

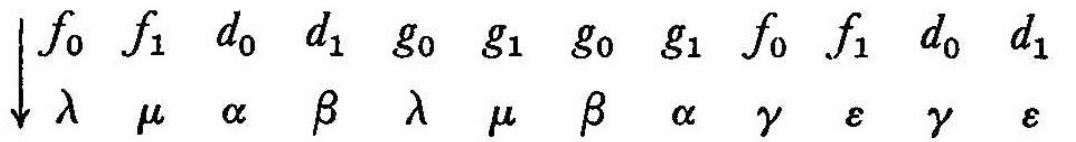
Pokud se nacházíme v mřížce A(f_0, g_0, d_0), matice pro spočtení pravděpodobnosti získáme následujícím způsobem. Zvolíme jeden parametr (f_0), a poté vezmeme zbývající dva parametry (g_0, d_0) a v Obr. 15 najdeme, jestli ukazují na α nebo β. g_0 ukazuje na β a d_0 na α, obě matice tedy mají stejnou pravděpodobnost, že budou vybrány. Pro zjištění konečné pravděpodobnosti přechodu zvoleného parametru do dalších stavů tedy budeme muset sčítat pravděpodobnosti z obou matic. Pokud chceme spočítat pravděpodobnost



Obrázek 13 Regiony hustoty [9].



Obrázek 14 Příklad mřížky [9].



Obrázek 15 Způsob navázání proměnných na MTP [9].

pro $f_0 \longrightarrow f_0$, a v α je tato pravděpodobnost $0,20$ a v β $0,85$, výsledná pravděpodobnost přechodu je

$$f_0^{'} = 0,5(0,2) + 0,5(0,85) = 0,525$$

Podobným způsobem by se postupovalo i pro zbylé přechody a proměnné. Výsledná matice přechodů z jedné mřížky do druhé (MTPZ) by vypadala jako na Obr. 16.

Z této velké matice by vycházely Markovovy řetězce, ze kterých by se nakonec zjistilo pořadí mřížek ve výsledné skladbě.

Kromě čistých Markovových řetězců Xenakis používal ještě pertubace, neboli umělá narušení přirozeného chodu. Ty sloužily jednak k ozvláštnění skladby, jednak pro snížení výpočetní náročnosti. Pertubace spočívá ve své nejjednodušší verzi ve vynuceném přechodu do jedné mřížky odkudkoliv. Při častém používání pertubací tedy není potřeba počítat MPTZ do žádné velké hloubky kroků a místo toho je možné pro kroky bezprostředně po pertubaci počítat s maticí s pouze jedním sloupcem (víme přesně, z jaké mřížky vycházíme, proto pouze jeden sloupec) [9].

MTPZ

↓	A ($f_0g_0d_0$)	B ($f_0g_0d_1$)	C ($f_0g_1d_0$)	D ($f_0g_1d_1$)	E ($f_1g_0d_0$)	F ($f_1g_0d_1$)	G ($f_1g_1d_0$)	H ($f_1g_1d_1$)
A($f_0g_0d_0$)	0.021	0.357	0.084	0.189	0.165	0.204	0.408	0.096
B($f_0g_0d_1$)	0.084	0.089	0.076	0.126	0.150	0.136	0.072	0.144
C($f_0g_1d_0$)	0.084	0.323	0.021	0.126	0.150	0.036	0.272	0.144
D($f_0g_1d_1$)	0.336	0.081	0.019	0.084	0.135	0.024	0.048	0.216
E($f_1g_0d_0$)	0.019	0.063	0.336	0.171	0.110	0.306	0.102	0.064
F($f_1g_0d_1$)	0.076	0.016	0.304	0.114	0.100	0.204	0.018	0.096
G($f_1g_1d_0$)	0.076	0.057	0.084	0.114	0.100	0.054	0.068	0.096
H($f_1g_1d_1$)	0.304	0.014	0.076	0.076	0.090	0.036	0.012	0.144

Obrázek 16 Matice podmíněných přechodů mezi mřížkami A až H [9].

4.4 FOF syntéza

Formantová syntéza (Formant-wave synthesis, fonction d'onde formantique, FOF) vytváří proud zrnek, každé oddělené kvantem času, odpovídající periodě fundamentální frekvence. Jedna nota vyrobená touto metodou tedy obsahuje stovky FOF zrnek. Používaná FOF zrnka obsahují sinusoidu s buďto strmým nebo hladkým nástupem a kvazi-exponenciálním útlumem [18]. Jak bylo řečeno dříve, FOF syntéza spadá také pod syntézy využívající fyzikální modelování.

Formanty jsou špičky energie ve spektru, které obsahuje jak harmonické, tak neharmonické tóny a šum. Formantové špičky jsou charakteristikou mluvených samohlásek a barvy mnoha hudebních nástrojů. Formanty hlasu a hudebních nástrojů nicméně nejsou pevně dané, ale pohybují se podle fundamentální frekvence. Navíc jsou pouze jedním z mnoha znaků, podle kterých lidské ucho určuje zdroj zvuku [18].

FOF používá především dvojí obálky: lokální pro jednotlivá FOF zrnka, a globální pro celou notu. Díky lokální obálce trvá FOF zrnko pár milisekund a má charakter krátkého tlumeného sinusoidního shluku. Konvoluce krátké obálky a zmíněné sinusoidy přispívají ke slyšitelným vedlejším frekvencím kolem sinusoidy a vytváří formantové spektrum [18].

Každý FOF generátor je ovládán mnoha parametry. Mezi nimi jsou, mimo jiné:

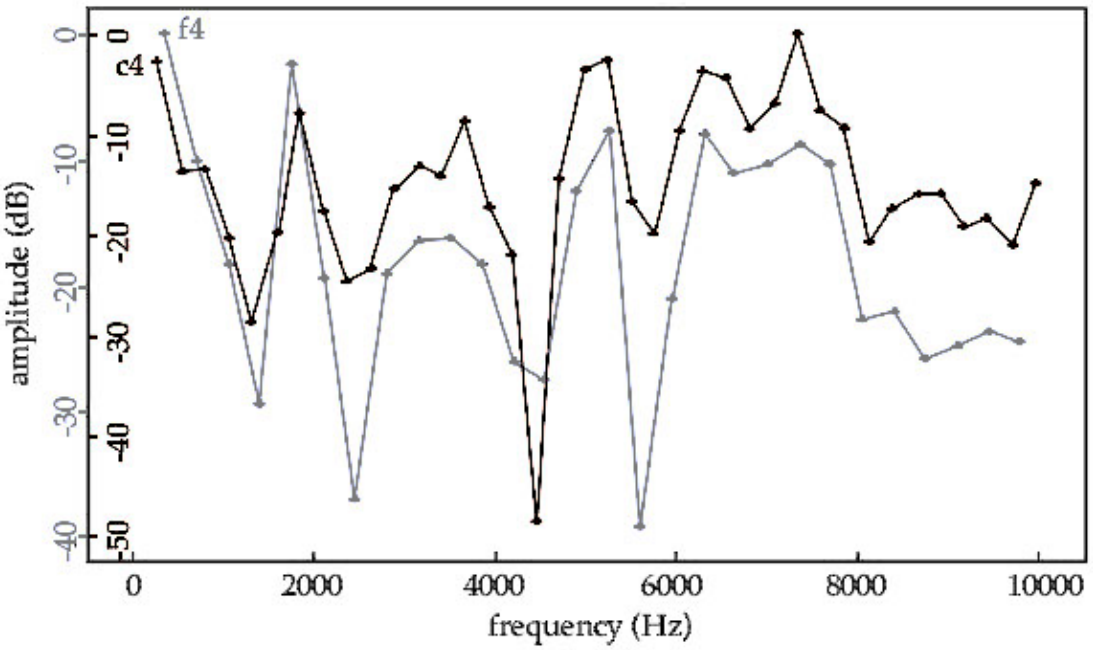
p1 Centrální frekvence formantu.

p2 Šířka pásma formantu, která je definována jako vzdálenost mezi body které jsou -6dB od špičky formantu.

p3 Nejvyšší amplituda formantu.

p4 Šířka formantové sukně (formant skirt). Formantová sukně je dolní část formantové špičky, asi 40 dB pod vrcholem, podobně jako pata hory. Parametr sukně je nezávislý na parametru šířky pásma, který spíše definuje šířku vrcholu hory.

Propojení operací časové a frekvenční domény je patrné ze specifikace FOF parametrů. Dva hlavní parametry jsou specifikovány v časové doméně jako parametry obálky FOF zrnka. Délka FOF nástupu ovládá parametr *p4*. Jak se doba nástupu prodlužuje, sukně se zužuje. Délka FOF útlumu zase ovlivňuje *p2*, šířku pásma formantu. Dlouhý útlum způsobuje ostrý resonanční vrchol, zatímco krátký útlum rozšíří šířku pásma



Obrázek 17 Trubka hrající dvě různé noty, čistou kvartu od sebe. Formanty (pevně dané resonance) ale zůstávají na stejných místech [18].

signálu [18].

Základní použití FOF syntézy leží v syntéze hlasu. Nicméně šikovnou manipulací s parametry lze dosáhnout i jiných syntetických zvuků a emulace hudebních nástrojů.

Typicky se při používání konfiguruje několik FOF generátorů najednou. Některé implementace jsou velmi složité, obsahující více jak 60 parametrů na každou zvukovou událost. Program CHANT, vyrobený v osmdesátých letech dvacátého století, byl vytvořen pro vyřešení této komplexity. Poskytuje sbírku pravidel pro zacházení s více FOF proudů najednou [18].

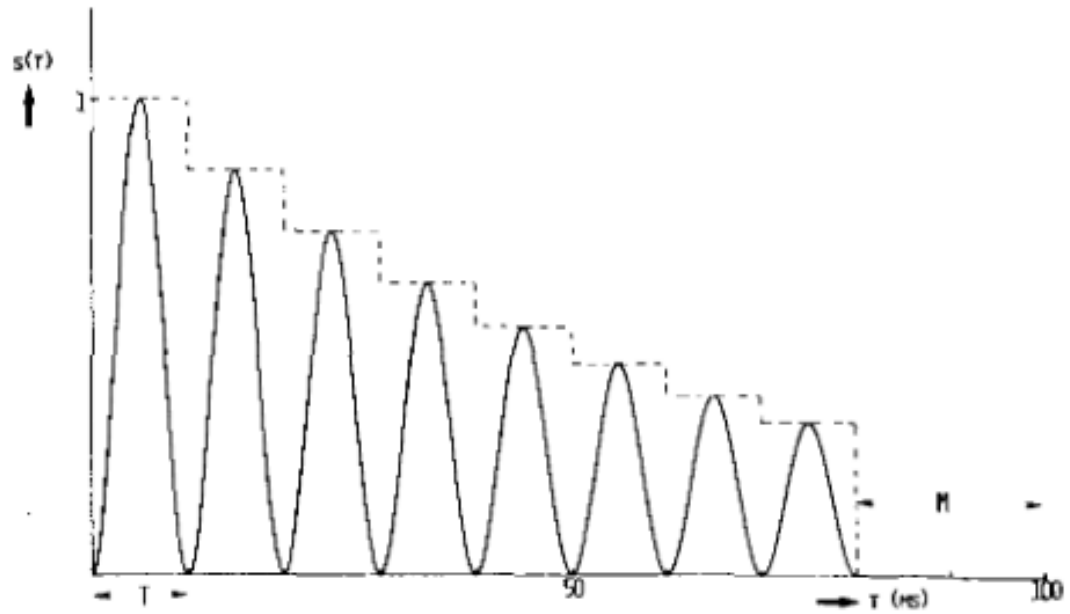
4.5 VOSIM

VOSIM (VOice SIMulation) je metoda syntézy zvuku založená na idee, že používáním opakujících se shluků tónů proměnné délky a prodlevy je možné dosáhnout zvukového výstupu s vysokou lingvistickou a hudební vyjadřovací silou. Byla vytvořena Wernerem Kaegim a Stanem Tempelaarsem [19].

První, na co byl VOSIM během vývoje orientován, byla syntéza řeči. Argumenty pro tento krok byly dva. Jedním byla snazší a více prozkoumaná práce zabývající se vlastnostmi lidské řeči (alespoň u indoevropských jazyků) než u charakteristických znaků vyskytujících se v hudbě. Druhým byla myšlenka, že metoda syntézy schopná produkovat přirozeně znějící řeč by měla být dostatečně flexibilní pro tvorbu hudby.

Každý tón VOSIM modelu se dá pochopit jako zrnko, jehož základem je sin^2 vlna. Tato vlna je rozdělena do N pulsů, každý délky T, s klesající amplitudou (amplitudou prvního pulsu budiž A), a celá je následována prodlevou M. Klesání amplitudy není exponenciální jako v přírodě, ale probíhá v krocích v závislosti na konstantním faktoru b vyjadřujícího výšku amplitudy v závislosti na předešlém pulsu. Příklad takovéto VOSIM funkce můžeme vidět na Obr. 18.

Kromě zmíněných proměnných se pro zvýšení flexibility zavádějí další proměnné. Aby



Obrázek 18 VOSIM funkce. N=8, b=0.85, T=10 milisekund [19].

bylo možné dostat vibrato, modulaci frekvence a šum, M musí být modulovatelné, podle sinusoidy nebo náhodně. Proto byly zavedeny proměnné S (výběr typu modulace – dle náhody nebo sinu), D (maximální odchylka od M), NM (míra modulace vyjádřená jako počet period, za kterou celý modulační cyklus musí být hotov). Nakonec byly zavedeny proměnné pro popis přechodových zvuků: NP (počet period) a DT, DA, DM (kladné nebo záporné přírůstky T, A a M v daném počtu period NP pomocí lineární interpolace). Celkově tedy máme následujících 12 proměnných [19]:

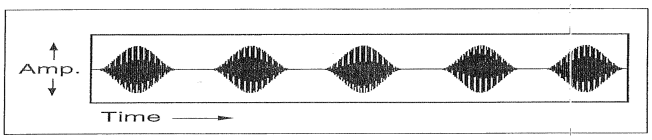
T (μs) – délka pulsu
DT (μs) – přírůstek k T
M (μs) – prodleva
DM (μs) – přírůstek k M
D (μs) – maximální odchylka od M
A – amplituda
DA – přírůstek k A
b (%) – útlumová konstanta
N – počet pulsů za periodu
S – typ modulace (1 = sin, 0 = náhodná)
NM – míra modulace
NP – počet period

Tyto proměnné lze zapsat do VOSIM vektorů jako

$$V = (T, DT, M, DM, D, A, DA, b, N, S, NM, NP)$$

Je vhodné poznamenat, že výsledné spektrum vykazuje jednak silnou složku korespondující frekvenci opakování f_0 signálu [$f_0 = 1/(NT + M)$], jednak vrcholy v místech korespondující frekvenci sin^2 pulsů. Tím pádem existuje „formantová“ oblast a metoda je použitelná pro simulaci řeči a hudebních signálů [19].

Zkušenosti ukázaly, že stačí pouze dvě synchronizované VOSIM funkce, namixované dohromady, pro syntézu řečových zvuků snad všech indoevropských jazyků. Na tomto



Obrázek 19 Ukázka kvazi-synchronní syntézy. Je možné pozorovat mírně proměnnou prodlevu mezi zrnky [11].

základě lze syntetizovat bohatší hudební vzorky – s pouhými třemi synchronizovanými VOSIM funkcemi se podařilo syntetizovat zvuk houslí [19].

Syntéza hudby pomocí VOSIM funkcí může být lépe ovládaná pomocí jazyku MIDIM (MInimum DescrIption of Music) [20].

4.6 Kvazi-synchronní syntéza

Ve svém základu není kvazi-synchronní syntéza (quasi-synchronous granular synthesis, QSGS) nikterak těžká na pochopení. Čistá synchronní syntéza klade v jednom proudu zrnka za sebe v naprosto pravidelných intervalech – prodleva mezi zrnkem a jeho následovníkem je vždy stejná. Asynchronní syntéza pro změnu může používat libovolnou prodlevu. Kvazi-synchronní syntéza pokládá zrnka v intervalech, které jsou v jádru pravidelné, ale mají náhodnou odchylku v určitém rozmezí. Neboli jsou kladeny téměř pravidelně [11]. Na Obr. 19 je vidět ukázka.

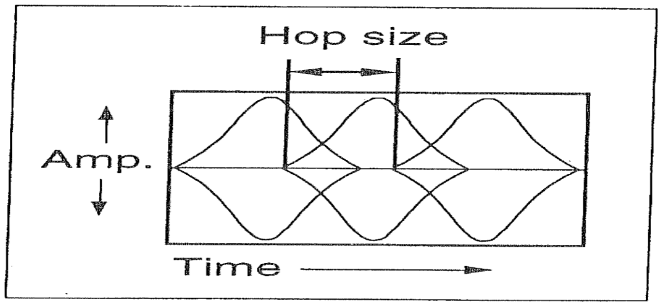
Na synchronní a kvazi-synchronní granulární syntézu se dá pohlédnout jiným způsobem. Ve své podstatě se chovají stejně jako amplitudová modulace, kde nosná vlna je složená z obsahu zrnek, a modulačním signálem by byla obálka celého proudu. Amplitudová modulace způsobuje růst síly frekvencí ve vedlejších pásmech (sidebands). Výsledkem je tvorba formantových regionů kolem frekvence nosné vlny. Nicméně pokud by, jak tomu je u synchronní syntézy, zůstala obálka čistě periodická, šlo by pouze o jeden stále se opakující zvuk (když bychom ještě předpokládali konstantní obsah zrnek). U kvazi-synchronní syntézy můžeme tedy změnami periody obálky tento zvuk ozvláštnit a tvořit tóny s drobnými odchylkami a tím je přiblížit reálným zvukům (například u dechových nástrojů málokdo udrží během jednoho tónu zcela konstantní dech a občas ujede nějaký záchvěv). Když následně sloučíme několik paralelních proudů, můžeme věrněji simulovat zpěv a různé nástroje [11].

4.7 Výškově synchronní syntéza

Výškově synchronní granulární syntéza (pitch-synchronous granular synthesis, PSGS) je metoda granulární syntézy snažící se o jemné a navazující přechody výšky tónu mezi po sobě jdoucími zrnky.

Jako taková je PSGS složitý mnohakrokový proces, začínající analýzou vstupu. Nejprve se na vstupu pomocí analýzy výšky tónů detekují jednotlivé periody výšky. K nim se ke každému přistupuje jako k samostatnému zrnku. Následuje analýza spektra každého zrnka. Je odvozena impulzní odezva spektra a podle ní nastaveny parametry filtrů pro resyntézu [11].

Během resyntézy je využívaná sada FIR (finite impulse-response, filtr s konečnou odezvou) filtrů, které na základě odezvy pulsového signálu vytváří výsledný signál. V každém časovém okamžiku systém vytvoří zrnko, které se překrývá s ostatními a společně s nimi vytváří dojem plynule se měnícího zvuku.



Obrázek 20 Proud překrývajících se zrnek ve výškově synchronní syntéze. *Hop size* je vzdálenost mezi následujícími zrnky [11].

Implementace od De Poliho a Piccialliho [21] ukazují několik transformací, které vytvářejí různé variace původního zvuku. Stejně tak popisují tuto metodu do mnohem větších detailů, než je uvedeno zde.

4.8 Pulsarová syntéza

Pulsarová syntéza je relativně nová metoda stavějící na základech daných granulární syntézou a starších analogových syntézách [22]. Ve své základní formě produkuje zvuky podobné elektronickým analogovým nástrojům, ve své pokročilejší formě zase mnoho rytmicky strukturovaných vzorkovaných zvuků.

Samotný jeden pulsar, malá částice zvuku, se skládá ze zvolené pulsaretové vlny w o délce d (doba sepnutí), následované periodou ticha s (Obr. 21(a)). Celková délka pulsaru je tedy $p = d + s$. Repetice pulsarů se nazývá pulsarový sled (pulsar train). Odvozujeme frekvenci repetice $f_p = 1/p$ a frekvenci korespondující době sepnutí $f_d = 1/d$. Typický rozsah f_p je 1 Hz až 5 kHz a typický rozsah f_d 80 Hz až 10 kHz [23].

V pulsarové syntéze jsou f_p a f_d průběžně proměnné hodnoty. Obě jsou ovládané různými obálkovými křivkami táhnoucími se přes celý sled pulsarů. Tento sled je jednotkou hudební organizace časově na úrovni not a frází. Může mít délku kdekoli mezi pár set milisekundami a více jak minutou [23].

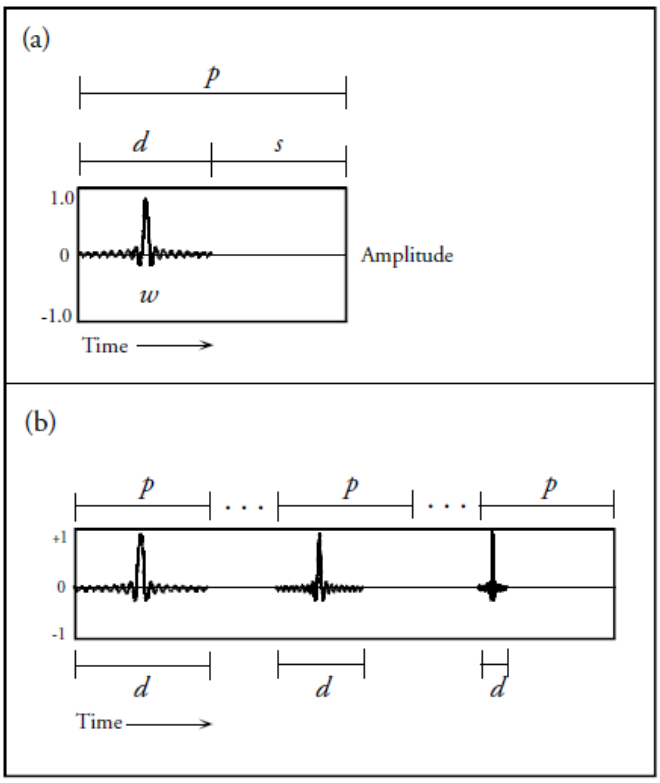
Na Obr. 21(b) si můžeme všimnout změny poměru $d : s$ za zachování délky p. Je to ukázka toho, že můžeme měnit jak fundamentální frekvenci (míra emise pulsarů), tak takzvanou formantovou frekvenci (která odpovídá době sepnutí), každou různými obálkami, jak bylo zmíněno výše.

Zatím je tento popis podobný standardním impulsovým generátorům. Nicméně pulsarová syntéza tento koncept v několika směrech zobecňuje.

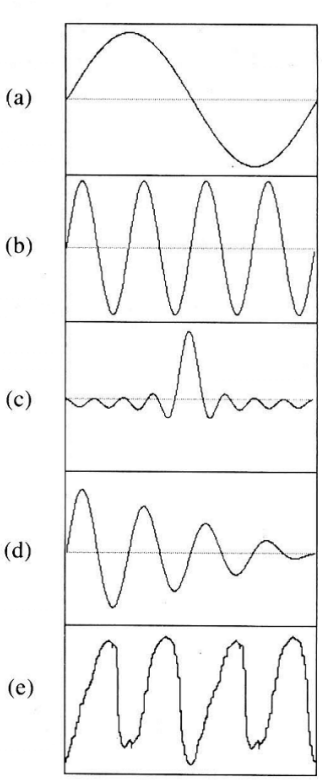
Za prvé, v pulsaru se může vyskytovat jakákoli vlna jako pulsaretová vlna w. Na Obr. 22 je možno vidět mnoho různých příkladů.

Každý pulsaret má svoji obálku v ovlivňující jeho výsledný tvar, podobně jako bylo popsáno v předchozích kapitolách. Co je důležité zmínit, tato obálka může mít jakýkoli tvar.

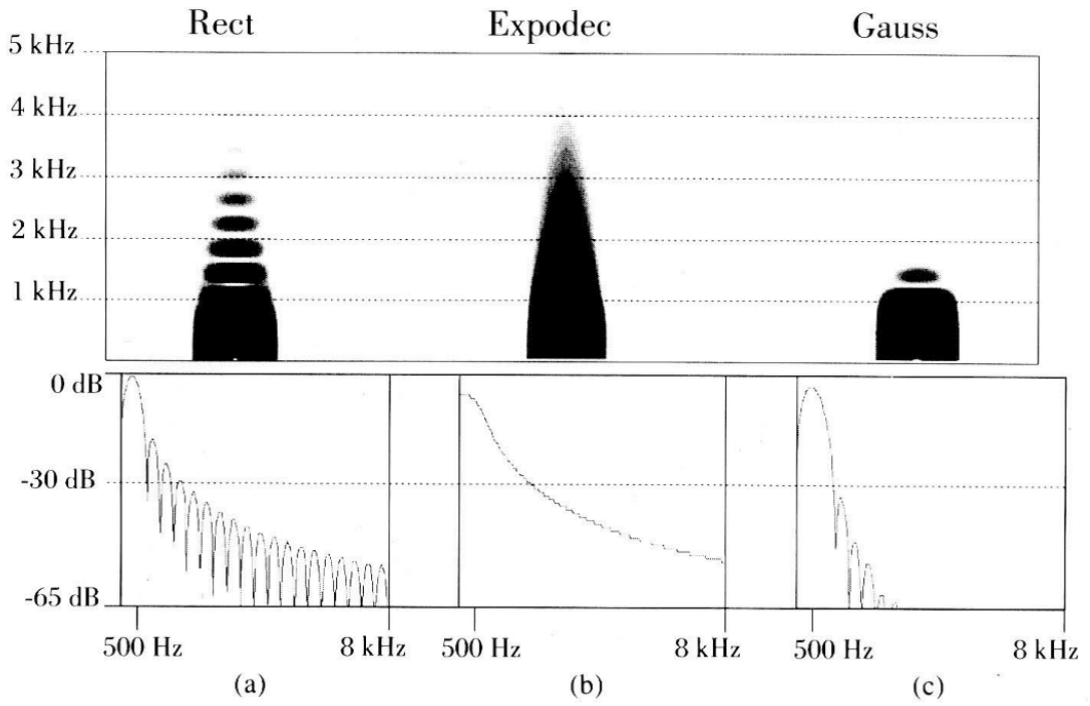
Pro syntézu konečné podoby pulsarového sledu používá pulsarová syntéza techniku podobnou modulaci šířky pulsu (Pulse-width modulation, PWM), a to modulaci šířky pulsaretu (Pulsaret-width modulation, PulWM), případně i v překrývané podobě (Overlapped pulsaret-width modulation, OPulWM). Jedná se o rozšíření PWM. Jedním z nich je, že vlna v pulsaretu může být jakákoli zvolená. Druhým je, že frekvence doby sepnutí může procházet skrz i být nižší než fundamentální frekvence. Překrývaná a nepřekrývaná verze se liší v chování, pokud je doba sepnutí d delší než délka pulsaretu



Obrázek 21 Pulsarová syntéza [23].



Obrázek 22 Typické pulsaretové vlny. a) Sinusoida b) Vícecyklová sinusoida c) Band-limited puls d) Tlumená multicyklová sinusoida e) Vlna kosmického pulsaru neutronové hvězdy Vela X-1 [22].



Obrázek 23 Ukázka vlivů různých obálek na spektrum pulsarů. Výše - frekvence-versus-čas sonogramy jednotlivých pulsarů. Fundamentální frekvence = 12 Hz, formantová frekvence = 500 Hz. Sonogramy jsou založené na fast fourier transformation s 1024 body a užitím Hannových oken. Níže – sonogramy vyrobené a) obdélníkovou obálkou b) exponenciálním útlumem c) Gaussovou křivkou. Spektrum na dB škále. [22].

p [22].

Pro p mezi zhruba 25 ms a 200 μs je sled pulsarů vnímán jako souvislý tón. Při zpomalování výroby nových pulsarů mizí pocit souvislosti a začíná být možné vnímat pulsary samostatně. Manipulováním fundamentální frekvence tedy můžeme vytvářet rytmus.

Vliv obálky na výsledné spektrum je značný. Spektrum vzniká konvolucí vlny pulsaretu w a obálky pulsaretu v. Obr. 23 ukazuje vliv různých tvarů obálky na pulsar, kde w je pevná sinusoida.

Pokročilá pulsarová syntéza k dosud probraným konceptům přidává další. Je založena především na třech principech [22]:

1. Více pulsarových generátorů se stejnou fundamentální frekvencí, ale různými individuálními formanty a prostorovými trajektoriemi.
2. Maskování pulsů pro tvarování rytmu pulsarového sledu.
3. Konvoluce pulsarových sledů s navzorkovanými zvuky.

Pulsarové generátory mají následující parametry:

1. Délka pulsarového sledu.
2. Obálka fundamentální frekvence pulsarového sledu f_p.
3. Obálka formantová frekvence pulsaretu f_d.
4. Pulsaretová vlna w.
5. Pulsaretová obálka v.
6. Amplitudová obálka pulsarového sledu a.
7. Prostorová cesta (spatial path) pulsarového sledu s.

S těmito parametry lze vytvořit více různých generátorů najednou a tím skládat složitější tóny.

Maskování pulsů probíhá jako vymazávání jednotlivých pulsů na určitých místech a tím přidávání nebo odebírání rytmu. Může probíhat ve třech formách:

Shlukové

Dávky pulsarů jsou mazány v pravidelných intervalech.

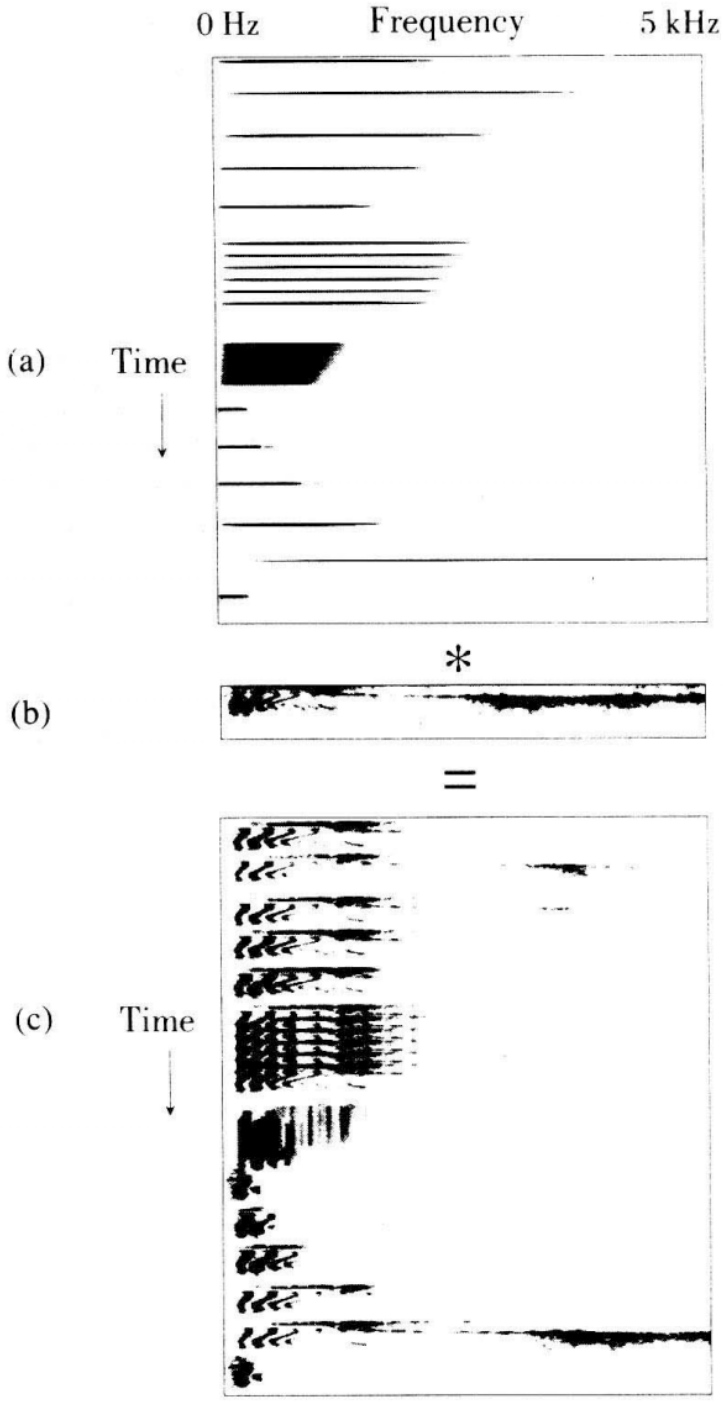
Kanálové

Vymazává pulsary střídavě z různých kanálů zvuku.

Stochastické

Maže pulsary v závislosti na stochastické funkci, implementované jako obálka.

Konvoluce ve své podstatě mísí dva signály a vytváří nový signál, který má v sobě zakomponovány spektra a časové struktury obou signálů. Pokud se zkříží pulsarový sled a krátký zvuk, je možné tento zvuk zobrazit do sledu. Tím je možné vytvářet zajímavé zvukové efekty a snadné dosahování barev přirozených zvuků [22]. Obr. 24 přibližuje tento efekt.



Obrázek 24 Efekt konvoluce na pulsarový sled. a) infrasonický pulsarový sled s proměnnou fundamentální a formantovou frekvencí b) Navzorkovaný zvuk: Italské „qui“ c) konvoluce a) a b) [22].

5 Digital Audio Workstation a formáty audio pluginů

V této kapitole se budeme zabývat obecným prostředím pro skládání a úpravu hudby a zvuků, jak do něj zapadají audio pluginy a prozkoumáme vybrané formáty audio pluginů.

5.1 Digital Audio Workstation

V dřívějších dobách, před zhruba osmdesátými lety dvacátého století, byly pro práci se zvukem používány výhradně analogové přístroje, které však byly drahé, náročné na údržbu a převoz a zacházení se zvukem bylo limitováno technologickými možnostmi. Stále však byly levnější a rychlejší než digitální technologie. Zejména rychlost a velikost tehdejších disků byla silně nepříznivá. Digitální technologie však s postupem času zlevňovaly a zvyšoval se jejich výkon, až nakonec v roce 1978 Thomas Stockham a jeho firma Soundstream představili první veřejnosti dostupný stroj pro digitální nahrávání a úpravu zvuku, The Digital editing system [24]. DES mimo mnoho jiných zajímavých funkcí dovoloval upravovat audio nahrané na hard discích a pracovat s různými efekty, například prolínáním. Ve své podstatě se jednalo o první Digital Audio Workstation (DAW).

Zhruba od poloviny osmdesátých let už počítače dostupné pro osobní užití začínaly mít dostatek výkonu, aby na nich bylo možné pouštět software pro editaci a nahrávání zvuku. Ze začátku dovedly pracovat pouze v jedné nebo dvou audio stopách a jednoduchými efekty. S dalším zvyšováním výkonu se ale možnosti rozšiřovaly, a mnoho nahrávacích společností začalo přecházet z čistě analogových stanic na digitální. Tyto digitální stanice sahaly od jednoduchých, čistě softwarových, až po složitější s integrovaným hardwarem (například klávesy nebo dedikované obvody) pro kvalitnější a rychlejší práci.

Výhody DAW oproti analogovým stanicím jsou v dnešní době několikeré. Stejně jako analogové verze, DAW dovolují práci se zvukem v několika různých stopách, nezávisle na sobě je upravovat, tlumit a mnoho dalších operací. DAW však nepotřebují žádný složitý hardware a pro většinu funkcí jim stačí obyčejný počítač (výkon počítače však ovlivňuje výsledné možnosti). Už to znamená nižší vstupní investici. Pokud bychom ale chtěli více funkcí a možností, vždy je možné prostředí rozšířit přídavným hardwarem. Dále, díky výkonům dnešních počítačů, dovedeme napodobit valnou většinu možností analogových stanic, a navíc je už kvalita digitálně zpracovaného zvuku velmi blízká, ba téměř stejná jako ten analogově zpracovávaný. Za další, na rozdíl od analogu, v DAW je možné používání klasických funkcí známých z běžné práce s počítačem jako je vrácení posledního kroku (undo), kopírování (cut, copy, paste) nebo ukládání rozpracovaného projektu, které výrazně usnadňují práci. Také je možné rozšířit možnosti zpracování zvuku pomocí různých zásuvných modulů (pluginů), ať už komerčních nebo volně dostupných. Většina DAW také poskytuje jistou automatizaci, neboli namapování různých akcí (změna parametrů, zahrání určitých MIDI not apod.) na body v čase a tím zajištění přesného opakování sledů akcí v časovém úseku, zatímco se snažíme vyladit jiné

části skladby.

V dnešní době existuje mnoho různých DAW, pro ukázku uveďme například Ableton Live [25], Steinberg Cubase [26] nebo REAPER [27], jehož ukázku můžeme vidět na Obr. 25.

Obecně se při vytváření skladeb využívá několika stop. Každá stopa představuje určitý nástroj nebo zvuk. Jako velmi jednoduchý příklad je možné si představit jednotlivé stopy jako třeba bicí, kytaru, basovou kytaru, druhé housle a podobně. Každou z těchto stop můžeme upravovat nezávisle na jiných, a to například mazáním částí zvuku, jejich tlumením, překrýváním, nebo například použitím různých pluginů. Existuje mnoho pluginů všemožných druhů - některé fungují jako filtry měnící vlastnosti zvuku jako například hlasitost, ořezávání výšek nebo hloubek, různé modulace, ozvěny a mnoho dalších, jiné zase fungují jako syntezátory, pomocí kterých je možné vytvářet zvuky z MIDI událostí. Tyto pluginy vesměs mají nastavitelné parametry ovlivňující jejich práci, například od které frekvence zvuku mají ořezávat výšky a podobně. Příklad rozpracovaného projektu můžeme vidět na Obr. 25, s otevřeným nastavením jednoho pluginu v pravé části obrázku. Pluginů může na jednu stopu být připojeno více a takto propojené tvoří řetěz zpracování, kde je zvuk postupně prohnán sledem připojených pluginů v daném pořadí, tedy každý následující plugin zpracovává výstup předchozích. Tímto způsobem je možné upravit prakticky jakýkoli aspekt zpracovávaného zvuku.

Těchto pluginů existuje vícero formátů. V další sekci rámcově prozkoumáme ty nejpoužívanější.

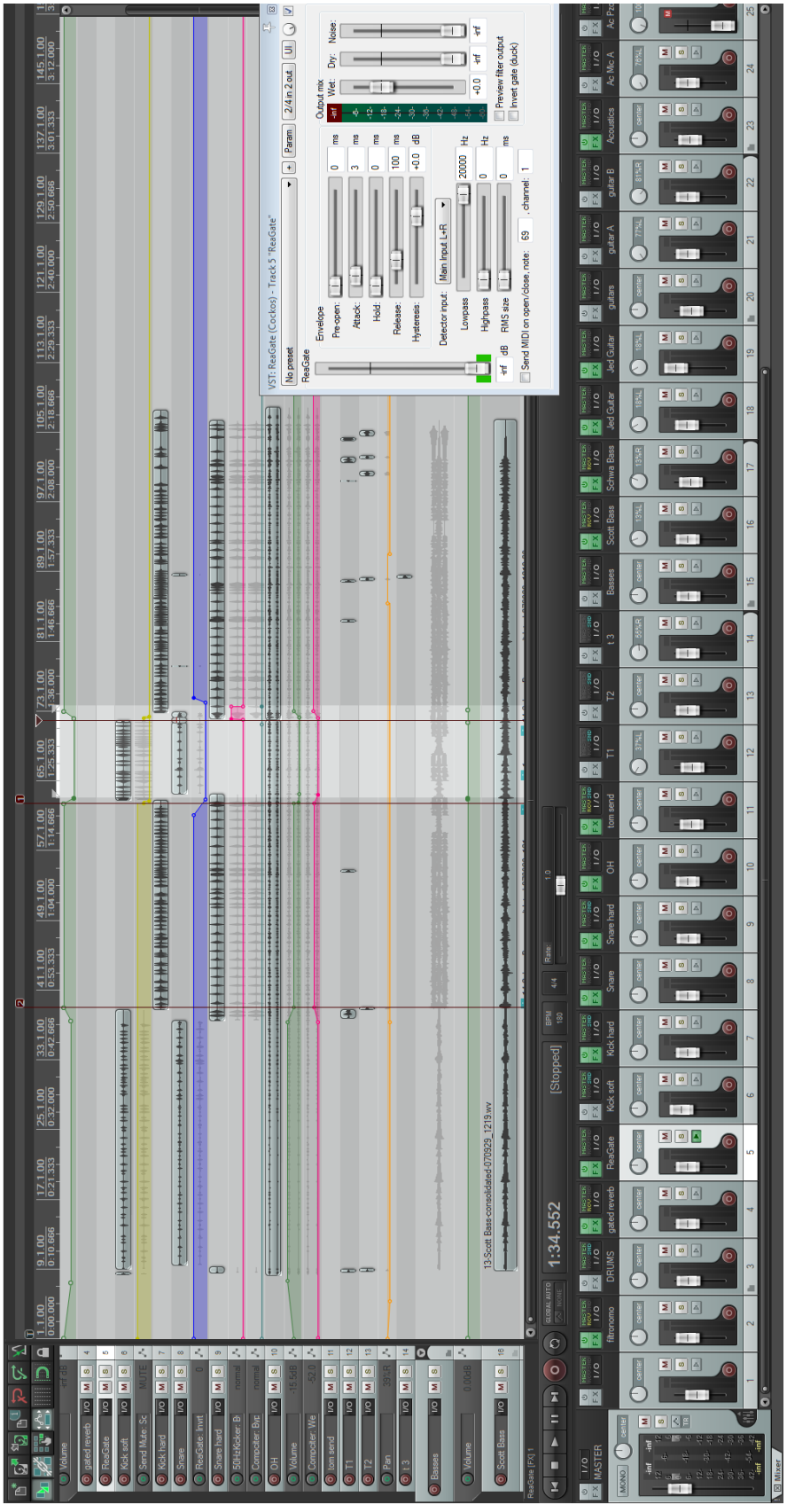
5.2 Formáty audio pluginů

Nejčastěji používanější formáty pluginů jsou v dnešní době Virtual Studio Technology (VST), Audio Units (AU), Real Time AudioSuite a Avid Audio eXtension (AAX), nejsou však zdaleka jediné. Vytvořené pluginy jsou používané v DAW (v tomto případě též nazývané plugin host), které dovede s daným formátem pracovat. Každý formát funguje jinak, minimálně z hlediska interfaců), proto ne každý plugin host podporuje všechny formáty.

Přestože se jednotlivé formáty v určitých záležitostech liší, v jádru pracují všechny víceméně stejně. Plugin host dá pluginu blok audia dané stopy na zpracování (délka bloku záleží na více faktorech a v této chvíli je irelevantní). Pokud je plugin vyžaduje, je k bloku přidán i seznam MIDI událostí pro daný blok. Plugin tento blok audia zpracuje/naplní (podle své funkce) a vrátí ho plugin hostu. Pokud plugin nebyl poslední v řetězu, je zpracovaný blok poslán dalšímu pluginu v pořadí, a takto až do zpracování celého řetězu pluginů.

Pluginy taktéž mohou mít nastavitelné parametry ovlivňující jejich chování. Změny těchto parametrů mohou pocházet buďto přímo z pluginu (konkrétně od uživatele měnícího hodnoty v GUI), nebo z plugin hostu, a to buďto z automatizovaných příkazů, nebo ze seznamu parametrů dostupných hostu ovládaného uživatelem.

Formáty se pak liší zejména v implementačních detailech. Jejich podrobné rozebírání by však nepřineslo velký užitek a bylo by příliš rozsáhlé. Prakticky platí, že co jde udělat v jednom formátu, jde velmi podobně udělat v jiném. Pro účely této práce postačí rámcové srovnání týkající se hlavně rozšířenosti formátu, na kterých operačních systémech je možné je používat, a případné zvláštnosti. Jak uvidíme později v implementaci, toto srovnání je skutečně postačující.



Obrázek 25 Ukázka DAW REAPER.

5.2.1 Virtual Studio Technology

Zkráceně VST. Jedná se o formát vyvíjený firmou Steinberg. Jedná se o jeden z prvních vyvinutých plugin formátů, původně vydaný v roce 1996 [28]. Díky své dlouhé historii a možnosti vyrábět pluginy kýmkoli s přístupem k SDK je formát podporovaný velkým množstvím plugin hostů na různých operačních systémech, například prostředími Ableton Live, Acid PRO, Cubase, Nuendo nebo REAPER. Jeho momentální verze (v květnu 2015) je 3.6. Obecně verze 3 přinesla velké změny [29], mimo jiné kompletní přepis jádra formátu, zvýšení výkonnosti a mnoho dalšího, ale také zpětnou nekompatibilitu s předešlými verzemi (souhrnně označovanými jako VST 2.x, zatímco verze 3 je známá jako VST3). K podpoře VST3 formátu se však zatím ještě nedostaly všechny plugin hosty, které podporovaly VST 2.x. SDK pro verzi 2.x již není distribuováno, nicméně novější verze VST3 SDK obsahují i wrappery pro kompilaci do VST 2.x formátu pro využití i v hostech, které ještě neprodělaly přechod na nový formát. Pro přístup k SDK je potřeba se přihlásit do vývojářského programu Steinbergu. Toto přihlášení je zpracováno nejvýše do několika hodin, poté je možné SDK stáhnout.

Je také možné narazit na formát VSTi, neboli VST instrument. Tento formát je v podstatě rozlišení mezi pluginy pouze upravující zvuk (VST) a pluginy syntetizující nový zvuk (neboli nástroje, VSTi). Z hlediska implementace mezi těmito dvěma není skoro žádný rozdíl.

5.2.2 Audio Units

Audio Units, též AU, je systémový formát operačního systému Mac OS X od firmy Apple. Důsledkem toho je, že AU je možné používat pouze na počítačích se systémem Mac OS X a plugin hostech na něm běžících. AU využívají sadu API poskytnutých operačním systémem pro manipulaci s proudy zvuku v téměř reálném čase s minimální latencí. Jako takové se mohou, díky podpoře přímo od systému, chlubit větší rychlostí a stabilitou než jiné formáty běžící na stejném počítači. Díky jeho podobnosti s VST existují také wrappery mezi těmito dvěma formáty, například Symbiosis nebo FXpansion VST-AU Adapter. SDK je volně dostupné ke stažení společně s vývojovým prostředím Xcode v Mac App Store. Mezi plugin hosty schopné hostovat AU patří například Ableton Live, Logic Pro, Cubase a další.

5.2.3 Real Time AudioSuite a Avid Audio eXtension

Zkráceně RTAS a AAX, oboje jsou formáty od firmy Avid. RTAS je starší z těchto dvou, zatímco AAX je jeho nástupce. RTAS v budoucnu nebude podporován, ale zatím, po přechodnou dobu, stále je. Oba formáty jsou podporovány pouze v produktech firmy Avid, zejména prostředí Pro Tools, RTAS ve verzích Pro Tools 10 a níže, AAX 11 a výše. Hlavní důvod pro vznik AAX, a tím i hlavní rozdíl mezi RTAS a AAX, byl přechod z 32-bitového formátu zpracování na 64-bitový. S těmito informacemi by tedy nové pluginy měly být vyvíjeny ve formátu AAX. Kromě čistě softwarového zpracování zvuku (verze formátu označovaná jako AAX Native) je AAX schopné i efektivně využívat hardware pro digitální zpracovávání signálů (Digital Signal Processing) schopné pracovat s Pro Tools, a to ve verzi formátu označované jako AAX DSP. Mírný problém AAX je nutnost opatřit pluginu digitální podpis firmou PACE [30], který je, dle podmínek vývojářského programu, placený [30]. Tyto podmínky obecně působí oproti podmínkám předchozích formátu restriktivně. Mimo jiné se přihlášením do vývojářského programu zavazujete k neprozrazování detailů SDK a k uvádění pluginů na marketplace Avidu. Pro získání

SDK je potřeba se přihlásit do zmíněného vývojářského programu. Doba pro vyřízení přihlášky je uváděna do deseti dnů.

6 Design audio pluginu využívající techniky granulární syntézy

V této kapitole je rozebírán design audio pluginu založeném na metodách granulární syntézy.

Postupně bude definována stručná motivace, použitá technika granulární syntézy, vstupní parametry a jejich vliv, způsob ovládání a high-level design celého pluginu.

6.1 Motivace

Motivací a záměrem této práce je vytvoření audio pluginu použitelného v DAW pro syntézu hudby a zvuků obecně. Tento plugin by měl umožňovat nahrát zvukový soubor a pomocí MIDI kláves z něj přehrávat krátké (délky do zhruba 200 milisekund) útržky vybírané částečně náhodným, částečně kontrolovaným způsobem. Mělo by též být možné ovlivňovat jednotlivé parametry útržků, například jejich délku.

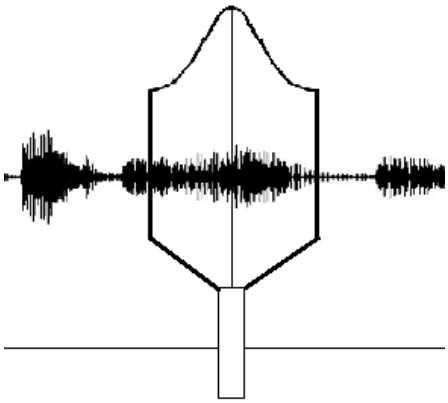
Z tohoto popisu je zřejmá velká příbuznost s granulární syntézou - jednotlivé útržky zvuku jsou totéž jako zrnka a chovají se téměř stejně.

6.2 Popis techniky granulární syntézy

Využití granulární syntézy je v tomto pluginu poměrně očividné. Jak bylo poznamenáno výše, jednotlivé útržky zvuku si lze představit jako zrnka používaná v granulární syntéze. Je však potřeba blíže rozebrat způsob, jakým jsou jednotlivá zrnka vytvářena.

Na vstupu máme MIDI události z kláves. Tyto události jsou v podstatě zprávy, které klávesy jsou právě zmáčknuté, které byly puštěny a podobně. Z těchto zpráv musíme nějakým způsobem vytvořit požadovaná zrnka. Způsob, který bude využit, si lze představit následovně: Za předpokladu, že známe rozsah vstupních MIDI kláves (první klávesu a jejich počet), můžeme na ně načtený soubor zobrazit. Snad nejjednodušší zobrazení, které se nabízí, je prosté časové - první klávesa představuje začátek zvuku, poslední klávesa konec a zbytek kláves je rovnoměrně rozmístěn ve zbytku zvuku.

Další, podobný, způsob, který se nabízí, je před zobrazením kláves tóny v souboru seřadit podle výšky. Tento způsob byl však shledán nepraktickým z několika důvodů. Za prvé už řazení tónů je problematické. U jednoduchých (například čistá generovaná sinusoida) lze samotné tóny řadit relativně snadno, ale pro složitější tóny (například různé nástroje, hlasy a podobně) už je řazení těžší. Každý nástroj vydává jiný charakteristický zvuk, což se ve spektrální analýze projevuje jinou směsí frekvencí pro daný tón. Není divu, že vytvoření univerzálního algoritmu pro řazení podle výšky, který by spolehlivě fungoval na všechny druhy zvuků, je obecně znám jako velmi složitý problém. Pokud ještě přibereme možnost souzvuku několika tónů, ať už od stejných nebo různých nástrojů, dostáváme až příliš složitý problém na záběr této práce. Dalším argumentem, proč tento způsob nevyužít ani pro jednoduché zvuky, je fakt, že uživatel pluginu má často možnost si sám vygenerovat seřazenou posloupnost zvuků (například



Obrázek 26 Znázornění zobrazení MIDI klávesy do zvukové stopy společně s pravděpodobností výběru segmentu jako začátku zrnka při zmáčknutí klávesy

obyčejný lineární pitch sweep[1]) a tu použít pro první popsaný způsob, bez nutnosti dalšího řazení.

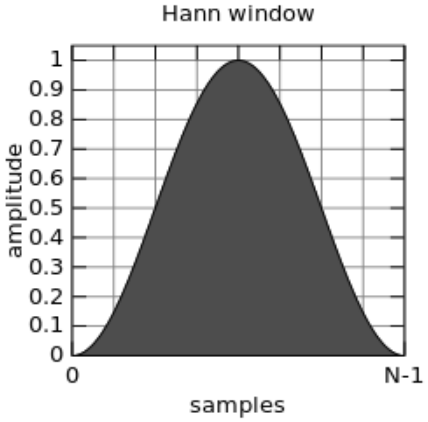
Nyní tedy máme jednotlivé klávesy namapovány na samostatné body v načteném zvuku. Tyto body zhruba označují místa, odkud se mají brát zrnka pro dané klávesy. Nicméně pokud bychom pokaždé brali zrnko z naprosto stejného místa, dostali bychom stejné zrnko, zatímco zbytek zvuku, na který nejsou namapované žádné klávesy, by zůstal ignorován. Zároveň by takovéto chování nesplňovalo požadavek na částečně náhodný výběr zvuku. Proto je ke každé klávese připojeno i okolí bodu, na který je klávesa namapována. Nic nebrání tomu, aby se sousední oblasti překrývaly. Zrnka, respektive vzorek, kterým začínají, jsou poté vybírána z dané oblasti náhodným způsobem. Silným kandidátem pro použití je normální rozdělení pro její tíhnutí ke středu (bodu dané klávesy) jako na Obr. 26. Použitím normálního, na rozdíl například od rovnoměrného rozdělení, je možné zvýšit míru kontroly nad výsledným zvukem.

Nyní tedy víme, kterým zvukovým vzorkem nové zrnko začínat. Dále potřebujeme vědět, kolik následujících vzorků použít. K tomu musíme znát délku zrnka ve vzorcích. U té by bylo záhodno, aby bylo možné ji měnit během používání pluginu. Necháme ji tedy jako vstupní parametr. Stejně tak velikost oblasti výběru zrnka by měla být variabilní - v některých případech můžeme při vytváření skladby ocenit širší oblast a tím i větší náhodnost, jindy naopak můžeme požadovat menší rozsah.

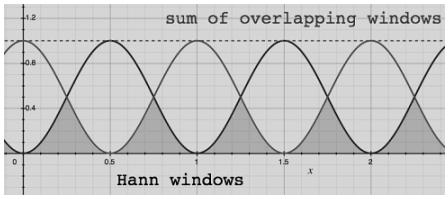
Z těchto informací teď dovedeme sestavit celé zrnko. Nicméně, pokud bychom ho používali v nezměněné formě, narazili bychom na nežádoucí jevy. Jmenovitě příliš prudké přechody hlasitosti mezi jednotlivými zrnky nebo zrnkem a tichem, a tím vznikající praskání v reproduktorech. Tomuto se dá zabránit aplikací vhodné amplitudové obálky. Hlavním požadavkem na obálku je, aby její začátek i konec byl v nule a tím při začátku ani konci zrnka nedocházelo ke skokům v hlasitosti. Vhodným kandidátem je Hanningovo okno (též známé jako Hannova obálka, Obr. 27), zejména kvůli jedné její vlastnosti. Pokud se dva stejně dlouhé objekty, na které je aplikována Hannova obálka, překrývají z 50%, výsledný součet faktorů hlasitosti je konstantně 1 (Obr. 28). Tedy při tomto překryvu dostaneme konstantní úroveň hlasitosti, od které jakékoli odchylky jsou způsobeny pouze odchylkami hlasitosti ve zdrojovém zvuku zrnka. Toto použití taktéž diktuje prodlevu mezi začátkem přehrávání dvou zrnek - přesně polovina zrnka.

Pro doplnění definice vytvářené techniky granulární syntézy nám zbývá pouze způsob, jakým jsou vytvářeny sekvence zrnek a jak jsou skládány. Máme vstup z MIDI kláves.

[1] Plynulý přechod od určeného nejnižšího tónu po určený nejvyšší, to celé za určený čas.



Obrázek 27 Hannova obálka [31]



Obrázek 28 Znázornění konstantní úrovně hlasitosti při 50% překryvu u Hannovy obálky [32]

Zmáčknutím jedné klávesy se vydá signál, aby se vytvořilo a přehrálo zrnko z oblasti přiřazené k dané klávese. Jedno zrnko se však přehraje velice rychle (do 200 milisekund), proto je potřeba při držení klávesy vytvářet další zrnka. Moment, kdy se má začít přehrávat další zrnko, určuje prodleva mezi zrnky, kterou jsme výše určili na polovinu délky zrnka pro vytvoření konstantní úrovně hlasitosti.

Je otázka, jestli by každé nové zrnko při držení klávesy mělo být vytvářeno výše popsaným náhodným procesem, nebo jestli by se po dobu držení mělo přehrávat stále stejné zrnko. Praktičtější a více v duchu granulární syntézy je spíše první možnost, tedy každé nové zrnko je opravdu nové. Dalším argumentem pro tuto volbu je náhodný výběr prvotního zrnka - vzhledem k většímu rozsahu výběru by bylo obtížné dosáhnout přesného požadovaného zvuku a být schopný ho následně reprodukovat při opětovném zmáčknutí klávesy. Nicméně, obě metody se sejdou ve speciálním případě, kdy je velikost oblasti výběru nastavena na nulu. Tento stav dá pokaždé stejné zrnko, protože není z čeho vybírat.

Nakonec tu je otázka vytvoření obálky pro celou notu (stisknutou klávesu). Tato obálka může vyvolat lepší pocit hry na nástroj a zároveň má možnost zcela eliminovat praskání v reproduktorech při začátku a konci noty. Tvar obálky by též měl být nastavitelný, aby se dal přizpůsobit nahranému zvuku a zamýšlenému způsobu hraní. Pro většinu účelů tedy bohatě postačí nastavitelná ADSR obálka (Attack, Decay, Sustain, Release). Ta má výhody jednak vysoké nastavitelnosti, jednak je velmi rozšířená a dá se předpokládat, že uživatel s ní bude umět zacházet bez nutnosti se učit, jak funguje.

Čtenář si jistě všiml, že zde uvedený popis se zaměřuje na úkony kolem stisku jedné klávesy. Při stisku více kláves pro každou probíhá stejný proces, a jejich skládání, tedy otázku polyfonie, probíráme níže v návrhu ovládání.

Uveďme ještě pro lepší představu příklad, co tato metoda dovede. Pokud jako vstupní soubor použijeme pitch sweep, jednotlivé klávesy potom představují tóny jakési stupnice. Vzhledem k náhodnému výběru zrnka v okolí klávesy a faktu, že ono okolí obsahuje

výškově blízké tóny, tak můžeme například vyvolat dojem kolísavého tónu.

6.3 Vstupní parametry

Vstupní parametry byly zmíněné v motivaci a popisu techniky. Zde jsou shrnuty a stručně popsán jejich vliv.

Vstupní soubor
Zvukový soubor, ze kterého jsou brána data pro zrnka. Ovlivňuje, jaký druh (barva, výška, tóny a tak dále) zvuků bude přehráván.

Vzorkovací frekvence (sample rate)
Počet vzorků za vteřinu výstupního audia. Tento údaj je potřeba pro případ, že by vzorkovací frekvence vstupního souboru byla jiná než výstupní a bylo potřeba ho převzorkovat. Zároveň najde využití, pokud by časové vstupní údaje (například délka zrnka) byly udávány v čase a ne v počtu vzorků. Vzhledem k nepraktičnosti zadávání těchto údajů v počtu vzorků by toto využití bylo velmi vhodné.

Délka zrnka
Délka jednoho přehrávaného zrnka. Očekávaný rozsah je zhruba mezi 10 a 200 milisekundami (založené na výzkumech z kapitoly 3). Může ovlivnit, jak dlouhé úseky souvisejícího zvuku se vyskytnou v konečném zvuku.

Velikost oblasti výběru
Velikost oblasti kolem klávesy, kde se bude ve vstupním souboru provádět náhodný výběr začátku nového zrnka. Stejně jako délka zrnka bude lepší z hlediska uživatelské přívětivosti, pokud bude nastavitelná v milisekundách než jako počet vzorků. Ze stejného důvodu se bude hodit, když nastavovaná velikost bude velikost celé oblasti, tedy oblast bude od začátku do konce dlouhá nastavenou hodnotu a bod klávesy bude uprostřed této délky. Na rozdíl od možnosti, kdy se oblast rozpíná o nastavenou hodnotu dopředu i zpět od bodu klávesy, nepotřebuje tento způsob žádný další krok v mysli uživatele pro výpočet velikosti oblasti.

První MIDI klávesa
Označuje první MIDI klávesu, na kterou se provádí zobrazení zvuku. Nastavená hodnota musí být platný index MIDI klávesy, tedy celé číslo v rozsahu 0 až 127.

Počet MIDI kláves
Počet MIDI kláves, na které je prováděno zobrazení zvuku. Tento počet zahrnuje i první MIDI klávesu, tedy bude v rozsahu 1 až 128. Pokud by rozsah namapovaných kláves měl zahrnovat i indexy, které nejsou platné (vyšší než 127), bude rozsah sahat pouze po maximální MIDI klávesu, tedy 127.

Attack obálky noty
Doba, než nota po stisku klávesy dosáhne maximální hlasitosti. Vlastnost amplitudové obálky noty.

Decay obálky noty
Doba, než nota po dokončení attack fáze dosáhne sustain hlasitosti. Vlastnost amplitudové obálky noty.

Sustain obálky noty
Úroveň hlasitosti noty udržovaná během držení klávesy. Vlastnost amplitudové obálky noty.

Release obálky noty
Doba, než nota po zvednutí klávesy dosáhne nulové hlasitosti. Vlastnost amplitudové obálky noty.

Vstupní MIDI události

Zprávy z MIDI kláves o stisknutých a puštěných klávesách. Tyto zprávy jsou zpracovávány a rozhoduje se podle nich, jaká zrnka generovat.

6.4 Návrh ovládání

Dalším krokem designu pluginu je návrh ovládání. Při něm je potřeba respektovat zvyklosti uživatelů. Jedná se sice o virtuální pluginy, ale i tak je zvykem navrhovat ovládání co nejvíce blížící se skutečnému, fyzickému modulu. Často se také při vytváření skladeb spoléhají na externí zařízení připojená k počítači, například už zmíněné MIDI klávesy, a skrz něj ovládají co největší část prostředí. Proto by bylo vhodné, aby byl minimalizován počet částí vyvíjeného pluginu, které vyžadují ovládání myší nebo klávesnicí a nelze je ovládat jinak, na minimum. Naštěstí jediný vstupní parametr, který je takto potřeba ovládat, je zadávání vstupního souboru. Ten je vhodné řešit klasickým systémovým dialogem pro výběr souboru. Tento dialog bude nejlepší ještě upravit, aby přijímal pouze soubory podporovaných formátů. Konečná množina podporovaných formátů samozřejmě závisí na konečně implementaci, ale pro lepší představu jako příklad uveďme, že by plugin byl schopný zpracovat pouze soubory formátu WAV nebo AIFF. V tom případě by dialog pro výběr vstupního souboru měl přijímat pouze tyto dva formáty.

Další zajímavý případ je vzorkovací frekvence. Tato hodnota by měla být nastavená na vzorkovací frekvenci zvukového výstupu (ať už zvukové karty nebo souboru). Tedy jedná se o vstupní parametr, avšak nenastavovaný uživatelem, ale prostředím, ve kterém plugin běží.

Ovládání MIDI událostí je poměrně očividné. Ty by měly být vkládané plugin hostem, ve kterém náš plugin běží, který je bude získávat z přednastavené automatizace nebo uživatelských příkazů. Případně lze do grafického rozhraní přidat komponentu kláves, kterou by taktéž šlo vytvářet MIDI události. Nicméně nejedná se o důležitou součást, tedy by mu měla být přiřazena nižší priorita. Lze ji však ještě vylepšit, pokud by její vzhled byl synchronizován s MIDI událostmi od plugin hosta - jednotlivé klávesy by vypadaly stisklé, pokud by byly stisknuté uživatelem nebo by vstupní MIDI události obsahovaly zprávu, že tato klávesa je stisknutá.

U MIDI událostí musíme vyřešit i otázku polyfonie. Nejjednodušší by byla implementace monofonní verze, kde od stisku jedné klávesy do jejího puštění nemá stisk jakékoli jiné klávesy žádný efekt. Přívětivější a schopnější možnost je však implementace vícehlasé polyfonie. To znamená, že každý stisk klávesy by spustil přehrávání zvuku v jednom z volných hlasů a všechny hrající hlasy by se složily do výsledného zvuku. Skládání probíhá prostým sčítáním. Počet hlasů je v tomto případě omezený především výpočetním výkonem. Po puštění klávesy by hlas přestal hrát a hlas se zařadil zpět mezi volné. Stále tu je ale otázka, jak se bude polyfonie chovat po vyčerpání volných hlasů. Jedna možnost je takzvané „kradení not“ - po vyčerpání hlasů by další stisklá klávesa přerušila jeden z hrajících hlasů, „ukradne“ ho a začne přehrávat svůj zvuk. Tato verze je však nešikovná, protože může dojít k přerušení uprostřed zrnka a bez sestupu hlasitosti na nulu (protože k „ukradení“ dojde okamžitě) a tím vznikne praskání v reproduktorech. Jiná možnost je stejná jako u výše zmíněné monofonie - po vyčerpání by další klávesy neměly žádný efekt. I zde jsou dvě verze. První reaguje pouze na moment stisku klávesy a následně nijak nereaguje až do jejího puštění, takže po uvolnění nějakého hlasu je potřeba stisklou nehrající klávesu stisknout znovu, aby začala přehrávat. Druhá verze zvuk stisklé nehrající klávesy začne přehrávat ihned po uvolnění hlasu. Intuitivnější by byla tato druhá verze, nicméně její přidaná hodnota k

první, jednodušší, verzi není tak velká, aby si zasloužila vysokou prioritu a zůstává tak pouze jako „nice-to-have.“

Zbylé parametry jsou už zamýšlené jako nastavitelné uživatelem. Nejprve proberme dvojici parametrů zabývající se používaným rozsahem MIDI kláves, první MIDI klávesa a počet MIDI kláves. Toto jsou dva celočíselné parametry, ale s mnoha možnými hodnotami (128). Jak bylo zmíněno na začátku oddílu, ovládání by se ideálně mělo blížit fyzickým modulům. Fyzický ovládací prvek hodící se pro takto diskrétní hodnoty je přepínač, například otočný. Nicméně pro 128 hodnot by byl takovýto přepínač příliš složitý a špatně ovladatelný. Proto budeme muset upustit od myšlenky ovládacího prvku s diskrétními hodnotami, použít spojitý a mezihodnoty správně zpracovávat, ať už ořezáváním nebo zaokrouhlováním. Dobrý prvek by byl obyčejný slider pro jeho rozšířenost, ovladatelnost a popisnost. Pokud použijeme horizontální, dostaneme i hezkou analogii ke klávesám, minimálně pro parametr první MIDI klávesy.

Poté zbývají pouze délka zrnka, velikost oblasti výběru a parametry ADSR obálky noty. U těchto taktéž využijeme slider, ale nejsme vázaní vizuálními analogiemi. Proto můžeme zvolit jakýkoliv druh a orientaci slideru, která se bude nejvíce hodit. Z hlediska šetření místem by však nejvhodnější byla otočná verze. Všechny tyto parametry, až na sustain, jsou časové. Pro větší přehlednost by všechny měly používat stejné jednotky. Délka zrnka samotná už ve své podstatě diktuje používání milisekund, takže tím je tento problém vyřešen. Sustain, úroveň hlasitosti, je zvyklostí uvádět jako zlomek maximální možné hlasitosti. Jestli jako desetinné číslo mezi 0 a 1 nebo procenta, na tom už ve výsledku nezáleží, nicméně častější je verze s desetinným číslem.

Při rozmisťování jednotlivých ovládacích prvků je také třeba dbát na blízkost souvisejících prvků. Máme v podstatě tři skupiny obsahující více prvků: První MIDI klávesu a počet MIDI kláves; délka zrnka a velikost oblasti výběru; parametry ADSR obálky noty. Ovládací prvky v každé skupině by vždy měly být blízko sebe. Důvodem pro toto rozhodnutí je zvýšení uživatelského komfortu - prvky ovládají významově blízké hodnoty, tedy uživatel je bude očekávat blízko sebe.

Pokud to bude možné, každý ovládací prvek s číselnou hodnotou by měl zobrazovat svoji momentální hodnotu. Tímto krokem bude uživatel mít větší přehled o stavu, ve kterém se nastavení pluginu nachází. Jako další úroveň přívětivosti by bylo dobré, aby navíc bylo možné tuto hodnotu přímo napsat a uživatel nemusel hledat přesnou hodnotu otáčením slideru.

6.5 High-level design pluginu

Nyní máme definovanou techniku, na které je plugin založen, a jeho vstupní parametry. Zbývá definovat, jaké softwarové komponenty by plugin měl obsahovat a jak spolu budou interagovat.

V jádru má každý formát pluginu (VST, AU, AAX...) hlavní modul, který komunikuje s plugin hostem. Před přehráváním host pluginu předá vzorkovací frekvenci výstupu. Následovně host volá metodu pluginu, ve které předává buffer se vstupním zvukem, buffer pro zápis výstupního zvuku a informaci o počtu vzorků ke zpracování (případně vygenerování). Taktéž mu, ve stejné metodě nebo na vyžádání, předává MIDI události pro daný časový úsek, respektive počet vzorků. Tato metoda na vygenerování zvuku je volaná pokaždé, když je potřeba vygenerovat další blok zvuku. Ve své podstatě plugin zajímá pouze fakt, že tato metoda bude volána a dostane správné informace. S touto součástí také host komunikuje v případě, kdy je z jeho strany změněn některý ze vstupních parametrů. Vzhledem k tomu, že právě tomuto modulu je vydán pokyn

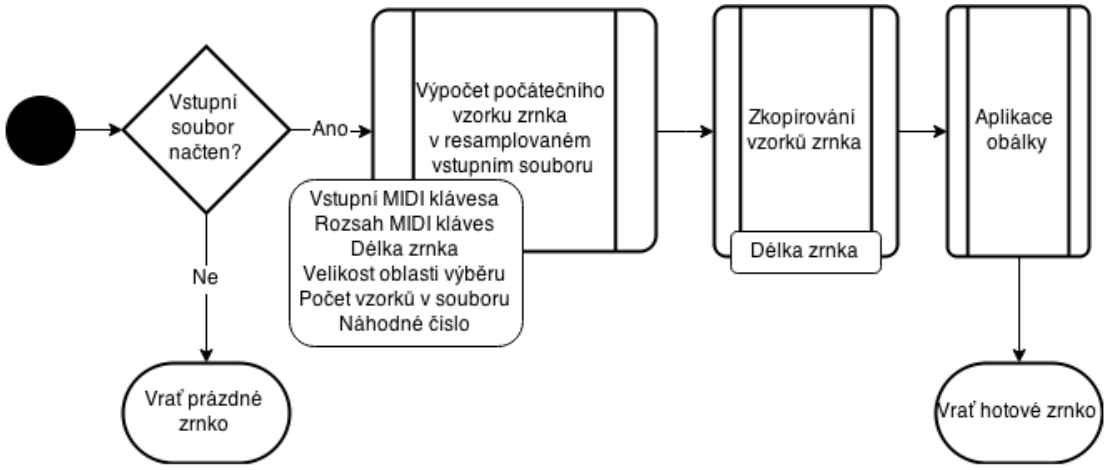
vygenerovat další blok zvuku, je logicky zřejmé, že veškerá funkcionalita, co se týče generování zvuku, bude volána právě odsud. Taktéž v tomto modulu budou uloženy veškeré nastavitelné parametry, protože se jedná o modul komunikující s hostem a host také může tyto hodnoty zobrazovat a nastavovat. Jakékoli změny parametrů tedy budou procházet skrz hlavní modul.

Centrální z hlediska granulární syntézy bude modul starající se o samotné generování zrnek. K tomu, abychom mohli zrnka generovat, musíme mít přístup k několika parametrům. Jmenovitě k délce zrnka, velikosti oblasti výběru, vzorkovací frekvenci, první MIDI klávese a jejich počtu, vstupnímu souboru a MIDI klávese, pro kterou zrnko generovat. Všechny parametry kromě právě stisklé MIDI klávesy by mělo být možné měnit za běhu a bude se tak dít mimo cyklus přehrávání zvuku z hlavního modulu. Pro snadnější používání je dobré parametry (s výjimkou vstupního souboru, vzorkovací frekvence a stisklé MIDI klávesy) tohoto modulu po jeho vytvoření inicializovat na počáteční hodnotu dle uvážení.

Změna vzorkovací frekvence a vstupního souboru má však jisté zádrhele. Když je modulu na generování zrnek dodán nový vstupní soubor, modul si uloží jeho původní verzi, a, pokud zná výstupní vzorkovací frekvenci, vyrobí verzi zvuku resamplovanou na dodanou frekvenci. Je potřeba podotknout, že dokud není nahraný žádný zvuk, nemohou vznikat žádná zrnka. Tento stav je potřeba v implementaci ošetřit, například vrácením prázdného zrnka. Při dodání nové vzorkovací frekvence by se nejprve mělo otestovat, jestli není stejná jako momentálně používaná. Momentálně používaná může být poslední dodaná, případně vzorkovací frekvence vstupního souboru, pokud žádná frekvence dodána nebyla. Pokud se právě dodávaná liší od používané, je nutné soubor resamplovat, za předpokladu, že nějaký je nahraný. Ale ať už nahraný je nebo ne, dodaná frekvence se uloží a využije se při budoucím nahrání souboru.

Proces změny parametrů zrnka by měl být jednoduchý - jde čistě o sdělení nové hodnoty, která se projeví při generování následujícího zrnka. Tyto hodnoty jsou interně uchovávány v počtu vzorků místo v milisekundách, je tedy potřeba je přepočítat za pomocí momentální vzorkovací frekvence. Vedlejší efekt je, že pokud se změní vzorkovací frekvence, je potřeba přepočítat i tyto hodnoty. Jednoduše na tom je také změna rozsahu MIDI kláves, akorát bez dalšího přepočítávání - čistě se nastaví požadované hodnoty. Pokud se pluginu změní rozsah kláves takříkajíc „pod rukou“, projeví se tato změna až s generováním nového zrnka, a to konkrétně změnou bodu, kolem kterého se nachází oblast výběru.

Samotné vytvoření zrnka je za této situace snadné. Metoda, kterou se žádá o zrnko, musí dostat číslo MIDI klávesy, pro kterou se má zrnko vytvořit. Následně se projde procesem uvedeným v popisu techniky syntézy. Pomocí právě stisklé klávesy, rozsahu MIDI kláves a velikosti oblasti výběru se vybere počáteční vzorek zrnka v nahraném zvuku (respektive jeho index) a následně se do zrnka překopíruje tolik vzorků, aby naplnily délku zrnka (zde se využijí právě parametry uložené v počtu vzorků namísto milisekund). Index počátečního vzorku musí samozřejmě ležet mezi 0 a *počet vzorků resamplovaného vstupního zvuku - 1*, hodnoty ležící vně tohoto intervalu je potřeba nastavit na nejbližší povolenou hodnotu. Tento případ bude nastávat především u první a poslední klávesy, protože polovina jejich oblasti výběru leží mimo nahraný zvuk. Vzhledem k tomu, že zrnka kratší než požadovaná délky budou narušovat výsledný zvuk, je potřeba maximální hodnotu rozmezí, kde může index počátečního vzorku ležet, ještě snížit. Maximální hodnota intervalu tedy budiž *počet vzorků resamplovaného vstupního zvuku - 1 - délka zrnka ve vzorcích*. Jediný speciální případ pak nastává, pokud je celý resamplovaný zvuk kratší než zrnko. V tom případě bude index počátečního vzorku konstantně 0. Po zkopírování (či pokud to půjde a bude to rychlejší rovnou během

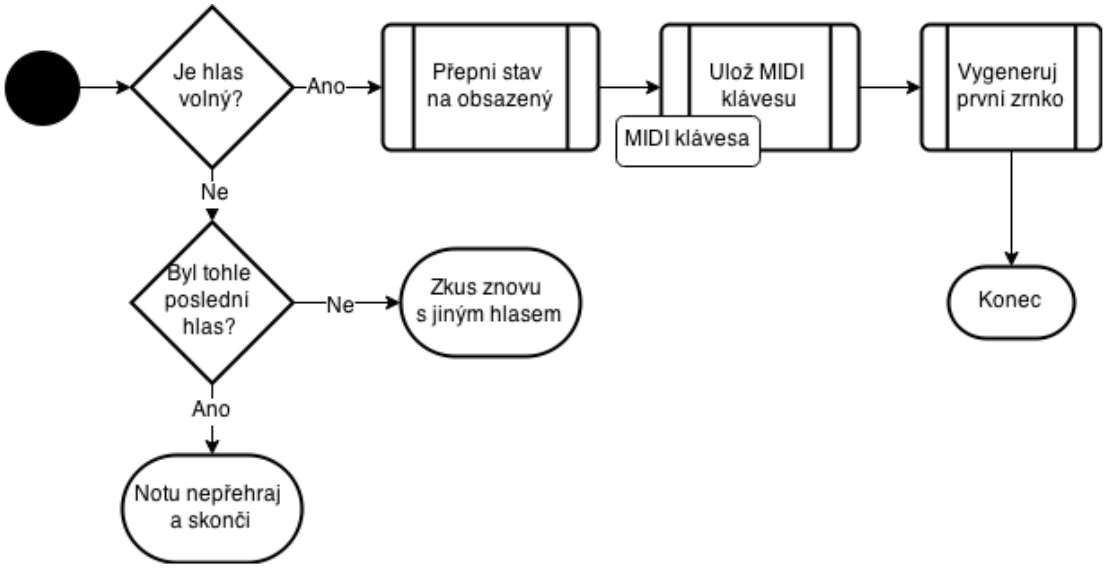


Obrázek 29 Znázornění procesu tvorby zrnka s potřebnými parametry uvedenými u jednotlivých procesů

kopírování) vzorků je na zrnko aplikována obálka, dle návrhu techniky Hannova. Tím je zrnko hotové a je možné ho vrátit volajícímu kódu. Znázornění procesu lze vidět na Obr. 29.

Jakmile máme jednotlivá zrnka, musíme je sdružit do not. Pro připomenutí, jedna nota v případě tohoto pluginu trvá od stisku klávesy do dohrání zvuku po jejím puštění. Modul, který se stará o přehrávání noty pojmenujme hlas. Každý hlas bude designován, aby přehrával najednou právě jednu notu. Pokud tedy budeme mít zájem o polyfonní přehrávání (a ano, budeme), bude nutné navrhnout ještě správce hlasů, který se bude starat jaký hlas bude přehrávat jakou notu. Nyní zpět k samotnému hlasu. Jak bylo zmíněno, ten se stará o přehrávání jedné noty se vším všudy. To zahrnuje žádání o nová zrnka, vždy když je třeba, a aplikaci ADSR obálky noty a následné kopírování zvuku do výstupního bufferu. Vyžadované parametry tedy jsou parametry ADSR obálky noty, vzorkovací frekvenci pro jejich přepočet na počet vzorků (stejně jako u délkových parametrů zrnka u předchozího modulu), právě stisklá MIDI klávesa (pro žádost o zrnko), výstupní buffer, od jakého vzorku do něj zapisovat a kolik vzorků do něj zapsat. Potřebujeme také upozornění, kdy dojde ke stisku klávesy (v tu chvíli bychom mohli dostat stisklou klávesu) pro začátek přehrávání, a kdy dojde k jejímu puštění, abychom mohli začít přehrávat release část obálky.

Stejně jako u modulu na výrobu zrnek by nastavování většiny parametrů, zde konkrétně ADSR parametrů, mělo být možné kdykoli, bez ohledu na stav, ve kterém se zrovna modul nachází. Upozornění na stav stisku klávesy, samotná klávesa, buffer a počet vzorků budou dodávány v průběhu přehrávání. Abychom byli konkrétní, nejprve hlavní modul získá seznam MIDI událostí pro momentálně zpracovávaný blok. Tento seznam je zpracován správcem hlasů a pokud se v seznamu nachází stisky kláves, je pro každý nalezen volný hlas a vydán pokyn k započetí přehrávání dané noty od času události. Pokud není dostupný žádný volný hlas ve chvíli, kdy zpracováváme nový stisk klávesy, záleží další akce na typu polyfonie, kterou implementujeme. Pokud pouze jednoduchou verzi, kdy je takový stisk klávesy ignorován, uděláme přesně to - zahodíme ho. Pokud používáme složitější verzi, kdy daná nota začne hrát po uvolnění hlasu, musíme událost uložit do fronty čekajících kláves. O tuto frontu se dále musíme složitě starat, mimo jiné hlídat, jestli čekající klávesa nebyla puštěna a podobně, proto pro jednoduchost budeme dále pracovat s jednodušší verzí polyfonie. Nicméně zpět ke zpracování události. Pokyn ke přehrávání noty by měl obsahovat, pro kterou MIDI klávesu se má

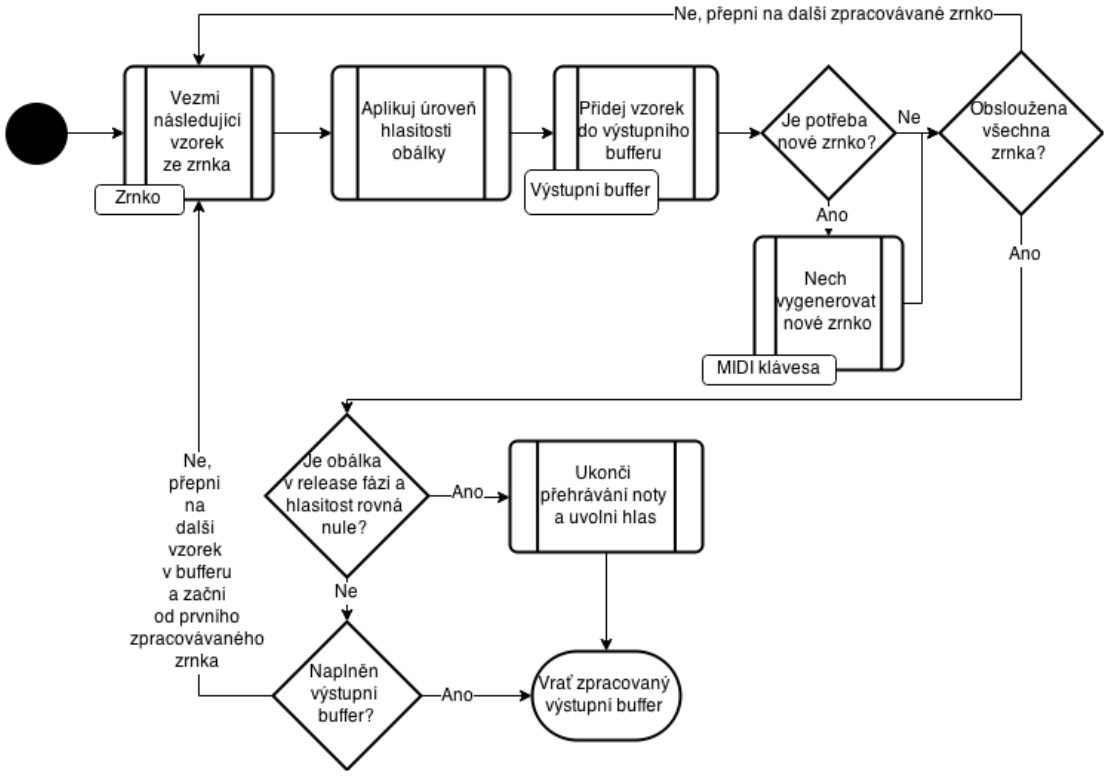


Obrázek 30 Znázornění procesu spuštění přehrávání noty hlasem

zvuk generovat. Hlas přepne svůj stav na obsazený, uloží si MIDI klávesu a vygeneruje první zrnko noty. Proces obsazení a spuštění přehrávání je možné vidět na Obr. 30. Od této chvíle až do uvolnění hlasu bude volána metoda pro generování zvuku.

Generování zvuku hlasem probíhá následovně. Daná metoda dostane buffer, kam bude přidávat svůj zvuk, index vzorku, od kterého má do bufferu zapisovat, a počet vzorků, které má zaplnit. Dále musí mít přístup k modulu pro generování zrnek. Pokaždé, když bude nutné vygenerovat nové zrnko, požádáme tento modul o zrnko a začneme ho přehrávat. Moment, kdy k této žádosti dojde, je popsaný výše v technice syntézy - nové zrnko je generováno vždy ve chvíli dosáhnutí poloviny délky předešlého zrnka. Vždy tedy přehráváme dvě zrnka najednou, s výjimkou první poloviny zrnka na úplném začátku noty. Je potřeba podotknout, že délka výstupního bufferu a délka zrnka může být rozdílná, takže můžeme naplnit celý výstup a skončit s nedohranými zrnky. Zbytek zrnek se tedy bude muset dohrát v následujícím bloku. Tím pádem je nutné ukládat si informaci o již přehrané části zrnek mimo tuto metodu. Přehrávání probíhá přičítáním vzorků zrnek do určených pozic ve výstupním bufferu, začíná se od indexu vzorku bufferu sděleného při volání metody. Hlasitost zpracovávaných vzorků je při kopírování upravována ADSR obálkou. Pro každou zpracovávanou pozici ve výstupním bufferu je potřeba spočítat úroveň hlasitosti obálky a případně upravit její stav. To neznamená nic jiného, než že při dosažení konce attack fáze se přejde do decay fáze, po dokončení decay se přejde na sustain. V sustainu se zůstává až do signálu k ukončení noty vzniklým puštěním klávesy. V ten moment je ukončena sustain fáze a přejde se do release fáze. V této fázi stále dochází ke generování nových zrnek až do úplného konce noty. Jakmile hlasitost obálky klesne na nulu, končí přehrávání a hlas je uvolněn pro využití novými notami. Je nutné podotknout, že release může začít teprve po skončení attack a decay. Pokud je signál k ukončení vydán před jejich skončením, obě fáze proběhnou až do svého konce, následně je přeskočen sustain a přejde se rovnou k release fázi. Tento už složitější proces je znázorněn na Obr. 31.

Už několikrát zmíněný modul je správce hlasů. Ten se stará o spouštění a ukončování jednotlivých not. Aby toho byl schopný, musí od hlavního modulu dostávat seznam MIDI událostí pro právě zpracovávaný blok. Tento seznam je následně prozkoumán pro stisky a puštění kláves. Jakmile narazí na stisk klávesy, nejprve ověří u generátoru zrnek,



Obrázek 31 Znázornění procesu přehrávání noty hlasem

jestli dovede s touto klávesou pracovat, neboli jestli se jedná o klávesu z nastaveného rozsahu. Jestliže ano, správce projde seznam hlasů, které má přidělené a pokud najde volný, klávesu mu přiřadí a nařídí přehrávání od daného časového momentu v bloku. Jestliže ne, klávesy si nevšímá. Podobný proces probíhá u puštění klávesy - nejprve se nalezne hlas, který danou klávesu přehrává. Jestliže existuje, je vydán pokyn k ukončení noty. Jestli neexistuje, není co řešit. Přehrávání zvuku je vyvoláváno hlavním modulem. Správce dostane výstupní buffer a jeho délku, a tyto pošle dál jednotlivým hrajícím hlasům společně s informací, od kterého vzorku má který hlas začít (pokud začnou přehrávat někdy během bloku). Tyto hlasy buffer naplní, tedy každý do něj na svá určená místa přičte svoje vzorky zvuku, a buffer je následně vrácen hlavnímu modulu.

Z tohoto procesu vyplývá, že správce musí mít přístup ke modulu pro generování zrnek a všem hlasům, o které se má starat. Obecně bude z hlediska architektury přehlednější tyto entity přiřadit přímo pod správce hlasů místo jejich „skladování“ přímo v hlavním modulu. Ten sice potřebuje s nimi komunikovat v případě změny parametrů, ale to je pouze proces vedlejší k samotnému generování zvuku. Toto spojení lze vyřešit buďto vytvořením proxy metody u správce, která pouze požadavek předá dál, nebo vyžádáním si odkazu na dané entity od správce hlasů, případně uložením si odkazu na ně pro rychlejší práci s nimi. I když si ale uloží odkazy, entity zůstávají pod hlavní správou správce.

Je tu samozřejmě také možnost hlasy vyrábět dynamicky při začátku každé noty a s koncem noty je opět zničit. To by (za předpokladu, že by nebyl stanoven maximální počet hlasů) vyřešilo problém s „kradením not“ a usnadnilo hledání volného hlasu. Nicméně, vzhledem k tomu, že se jedná o zpracovávání v reálném čase, dynamická alokace u většiny jazyků není levnou záležitostí a po jeho vytvoření bychom museli hlasu správně nastavit parametry, nemusí tato možnost být optimální. Taktéž je možné,

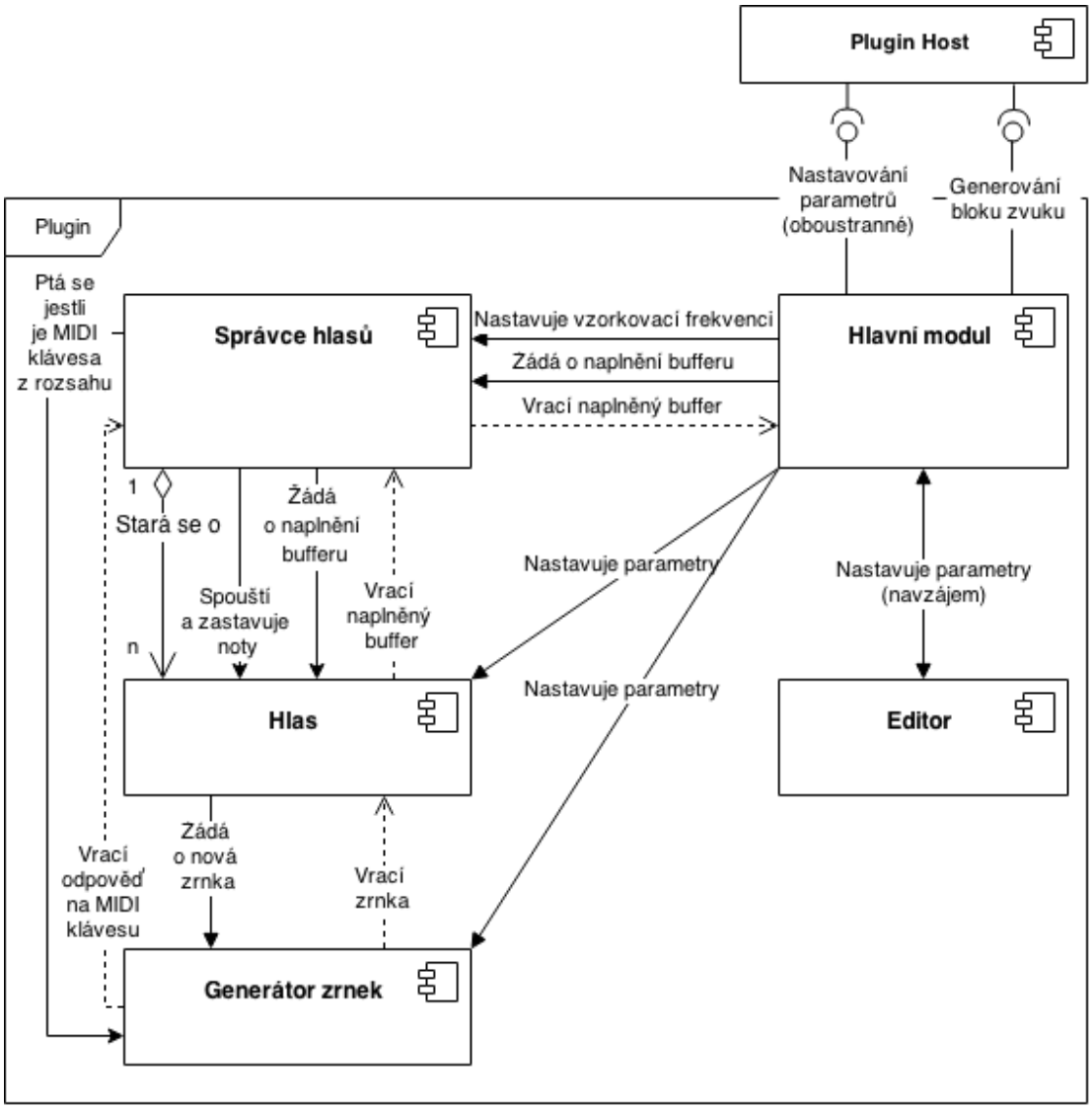
že nebude podporována zvoleným formátem pluginu.

Nyní se vrátíme zpět k hlavnímu modulu. Z většiny byla jeho funkcionalita popsaná. V jádru má metodu, kterou plugin host volá pokaždé, když je potřeba vygenerovat nový blok zvuku. V ní dostane výstupní buffer, který plugin naplní zvukem a následně ho odešle zpět hostu, počet vzorků v bufferu a seznam MIDI událostí pro blok. Nyní už víme, že aby nám celý navržený plugin správně fungoval, musíme z této metody správci hlasů poslat seznam MIDI událostí, výstupní buffer a jeho délku, které zpracuje a pošle dál. Po celém procesu nám nakonec správce vrátí naplněný buffer, který vrátíme hostovi. Také předtím, než můžeme začít přehrávat, musíme od hosta dostat vzorkovací frekvenci výstupního zvuku. Tu předáme správci hlasů, který ji opět předá dál.

Hlavní modul se také stará o parametry. Většina hostů s nimi pracuje v normalizovaných hodnotách, tedy 0 až 1. Host si je může kdykoli vyžádat a změnit je skrz standardizované funkce (každý formát má nějakou svoji obdobu). Bude tedy praktičtější si je taktéž ukládat v normalizované formě, avšak jiným modulům pluginu, aby se zmenšilo jejich navázání s venkovním prostředím, bude lepší je dodávat v denormalizované formě, tedy původních hodnotách. Při každé změně parametru si tedy uložíme novou normalizovanou hodnotu a modulům, které s daným parametrem pracují, pošleme denormalizovanou hodnotu.

Tento modul se také musí starat o ukládání a načítání nastavení při přerušení práce s projektem. Pro přiblížení, o co jde, si představme, že v plugin hostu pracujeme s naším pluginem. Máme všechny parametry nastavené dle našich přání, v hostu jsme naplánovali sled MIDI událostí. Nyní se rozhodneme, že je čas jít spát, projekt uložíme, další den ho zase načteme pro další práci. Pokud bychom v tento moment museli všechny parametry nastavovat znova a pokoušet se trefit do toho správného nastavení, byli bychom velmi rozladění. Proto musí plugin být schopný nastavení parametrů ukládat a následně i načítat. Místo pro uložení nastavení je pluginu poskytnuto hostem. Ukládání číselných parametrů, což jsou skoro všechny, je jednoduché. Pokud známe pořadí, či v případě ukládání jako slovník jméno parametru, můžeme je snad uložit a načíst jako hodnotu. S tímto přístupem ale očividně narazíme u načteného zvukového souboru. Nebude stačit, abychom ho uložili pouze jako cestu k souboru, protože soubor může mezitím být přesunut nebo projekt můžeme načítat na jiném počítači. Nezbývá nám tedy, než soubor uložit do poskytnutého bloku paměti skutečně celý. Při implementaci se můžeme podle možností rozhodnout, zda v nějaké zakódované formě (například jako base64 string), nebo přímo jako binární data.

Poslední částí celého pluginu je grafické rozhraní (GUI, editor). Pokud bychom měli plugin, u kterého by všechny parametry byly nastavitelné skrz hosta, bez GUI bychom se obešli. Naneštěstí musíme nastavovat i vstupní soubor, který takto nastavit nejde. GUI není potřeba zdlouhavě popisovat, většina byla řečená v sekci 6.4. Zde je třeba implementovat jednotlivé slidery, tlačítko a na něj navázaný dialog pro výběr souboru, a ideálně MIDI klávesy. Dále musíme ovládací prvky napojit na parametry hlavního modulu. Při tom je třeba dbát, aby na každou změnu parametru byl upozorňován i host - informace o změnách parametrů používají při nahrávání automatizace pluginu. Naopak jakákoli změna parametrů ze strany hosta se musí projevit i v editoru.



Obrázek 32 Graf vztahů modulů v pluginu

7 Implementace audio pluginu využívající techniky granulární syntézy

V minulé kapitole jsme rozebírali design pluginu. V této kapitole přistoupíme k jeho implementaci, neboli budeme se na design často odkazovat. Nejprve musíme zvolit formát, ve kterém budeme plugin implementovat, design mu přizpůsobit a nakonec ho implementovat.

7.1 Volba formátu

Volba formátu byla nakonec snadnější, než se zdálo. Shrňme faktory, které hrály roli ve výběru pro jednotlivé formáty. Podrobnější popis formátů nalezneme v kapitole 5.

VST a VST3
: Velmi rozšířený formát firmy Steinberg, ve verzi VST 2.x implementovaný mnoha plugin hosty ns různých operačních systémech. S verzí VST3 přišlo kompletní přepsání (a v podstatě vznik nového) formátu, zpětně nekompatibilního s verzemi VST 2.x. VST3 není dosud implementováno takovým počtem plugin hostů jako předešlá verze, ale v jeho pozdějších verzích byly přidány i wrappery pro kompilaci do VST 2.x formátu. Dostupné SDK je nyní pouze pro formát VST3, pro jeho poskytnutí je potřeba se přihlásit do vývojářského programu Steinbergu. Toto přihlášení je zpracováno nejvýše do několika hodin.

Audio Units
: Systémový formát operačního systému Mac OS X od firmy Apple. Jako takový tedy funguje pouze v operačních systémech Mac OS X a plugin hostech na něm běžících. Zakomponování do systému mu však poskytuje výhodu lepší a rychlejší odezvy a výkonu. SDK je veřejně dostupné.

RTAS a AAX
: Oboje jsou formáty firmy Avid, a jako takové fungují pouze v produktech Avidu, především prostředí Pro Tools. AAX je nástupce RTAS, které je již vyřazováno z oběhu. Přístup k SDK je podmíněný přihlášením se do vývojářského programu a splněním jistých podmínek, mimo jiné neprozrazování detailů SDK a uvádění pluginů na marketplace Avidu. Doba pro vyřízení přihlášky je uváděna do deseti dnů. Aby AAX plugin mohl být používán v Pro Tools, je potřeba mu opatřit digitální podpis firmou PACE [30], který je placený a proslýchá se, že poplatek se pohybuje v řádu 500 USD za rok [33].

Okamžitě dostupné prostředí pro vývoj pluginu je počítač s operačním systémem Windows 7. Dále bychom rádi minimalizovali finanční a časové náklady pro další zařizování vývojového a testovacího prostředí, a také co nejméně omezovali dostupnost pluginu. Tímto nám odpadnou Audio Units, pro ty bychom museli pořizovat nový systém (pravděpodobně i s celým novým počítačem). Můžeme také vyřadit RTAS a AAX, jednak z důvodu finančních nákladů na poplatky a pořízení Pro Tools na testování, jednak kvůli omezení pouze na jeden plugin host (Avid Pro Tools). Jediný formát, který nám tedy zbývá, je VST. Z důvodu rozšířenosti zvolíme spíše formát VST (verze

2.x) než VST3, kompilace bude využívat wrappery pro VST z SDK VST3. Abychom byli přesnější, bude se jednat o formát VSTi, protože implementujeme syntetizér (viz rozdělení v kapitole 5.2.1).

Po chvíli pokusů s čistým SDK jsme zjistili, že jeho použití v čisté formě je velmi neohrabané. Naštěstí komunita kolem audio pluginů není nijak mladá a má vyvinutá svá řešení. Nejčastěji se používá jeden ze dvou frameworků, JUCE [34] nebo WDL-OL [35]. Oba mají za cíl usnadnit vývoj audio pluginů a poskytnout možnost napsat kód pouze jednou a následně ho zkompilovat do různých formátů pluginu. Stačí pouze zvolit formát, dodat jeho SDK a je vystaráno. Jsou si v lecčems podobné, včetně použitého jazyka (C++). Stručné porovnání lze sestavit zhruba takto, za využití existujícího srovnání [36]:

JUCE

- Dobrá dokumentace.
- Konzistentní a vysoce kvalitní kód.
- Široká uživatelská základna.
- Velmi dobré grafické možnosti, včetně vektorové grafiky atd.
- Hezký generátor projektu (Introjucer).
- Dá se využít i na jiné věci než pluginy.
- Nevynucuje žádné určité techniky pro thread safety - je potřeba použít vlastní.
- Zdarma dostupný pod GPL, je možné zaplatit za non-GPL verzi.

WDL-OL/IPlug

- O něco jednodušší pro začátečníky v C++.
- Stručná implementace pluginů.
- Velké množství pomocných skriptů.
- Nastavení IDE projektů pro debugování.
- Možnost si ho z githubu „forknout“ a vyrobit vlastní verzi.
- Zdarma.

Pro nás asi nejdůležitější otázky jsou, jestli nám vadí GPL a jestli chceme lepší grafiku. GPL nám ve své podstatě nevadí, nemáme s tímto pluginem žádné plány na výdělky. Lepší grafika se nám bude hodit pro návrh GUI, jak bylo popsáno v předešlé kapitole. Z těchto dvou frameworků tedy využijeme JUCE.

7.2 Detaily implementace

Nyní máme vybranou platformu i framework, můžeme se tedy pustit do práce.

Nejprve jsme pomocí Introjuceru (aplikace JUCE frameworku sloužící k vytváření projektů) několika snadnými kroky, během kterých jsme nastavili takové věci jako jméno, požadované komponenty a jejich nastavení, umístnění VST SDK, vygenerovali projekt, včetně potřebných includů, a solution pro Visual Studio. Rozchodit samotné Visual Studio (použita byla Community edice) nebyl žádný problém, a vygenerovaný solution byl bezchybný.

Pro nás důležité jsou v projektu především třídy PluginEditor a PluginProcessor včetně jejich headerů. PluginEditor se stará o GUI pluginu, PluginProcessor zase plní roli hlavního modulu z designu - komunikuje s hostem. Prvotní stav ihned po vygenerování je „hluchý“ Hello World plugin, který pouze nechává skrz sebe proudit zvuk, bez jakékoliv úpravy.

Tento vzorový plugin lze bez problémů zkompilovat a použít v libovolném plugin hostu implementujícím VST rozhraní. Instalace a deployment se neliší u vzorového pluginu, konečného pluginu a ani jakéhokoli jiného - plugin hosty mají každý svůj adresář,

ve kterém hledají pluginy, které by mohly použít. Do tohoto adresáře zkopírujeme náš zkompilovaný plugin (pro VST formát *.dll), necháme plugin host adresář prozkoumat a následně můžeme náš plugin do hostu vložit a používat. Toto je pouze velmi obecný postup a přesné instrukce se mohou lišit host od hostu.

K testování během vývoje jsme používali vzorový plugin host JUCE frameworku a REAPER, v těchto dvou je tedy zaručená funkčnost. Toto testování probíhalo průběžně ručním testováním změn a občasným ověřením celkové funkčnosti. Vzhledem k relativně malé velikosti pluginu je tento postup použitelný a nezabralo výrazně více času než odhadujeme, že by zabral vývoj automatických testů.

Dále implementace postupovala víceméně podle designu, zde v textu budou zdůrazněny především různé zajímavosti a zvláštnosti implementace.

7.2.1 Správce hlasů

Tento modul je z valné většiny implementován třídou JUCE frameworku jménem Synthesiser. Po jeho vytvoření do něj musíme dodat seznam hlasů a zvuků, které má spravovat. Zmíněnému zvuku se v našem případě rovná náš generátor zrnek (musí dědit ze třídy SynthesiserSound), kterého se syntetizér při zkoumání MIDI událostí ptá, jestli zkoumanou MIDI notu dovede přehrávat. Pokud zvuk odpoví kladně, je předán některému z hlasů, které sdělí, že ho dovedou přehrát. Tyto hlasy pro změnu musí dědit ze třídy SynthesiserVoice. Otázka, jestli hlas dovede přehrát daný zvuk, se řeší většinou dynamickým přetypováním dodaného SynthesiserSound zvuku na třídu zvuku, pro kterou byl hlas stavěn, v našem případě DynamicGranulizedSampleSound. Jestli je přetypování úspěšné, znamená to, že zvuk má potřebný interface a je možné ho naším hlasem používat. Následně syntetizér hlasu předá zvuk i s pokynem, ať začne hrát, a dále už jen opětovně volá metodu hlasu renderNextBlock, ve které je plněn předávaný audio buffer tak, jak je popsáno v designu.

Tento syntetizér bez problémů zvládá polyfonii, a to až do počtu hlasů rovnající se počtu dodaných hlasů. Typ polyfonie je jednodušší z popsaných, tedy pokud při stisku klávesy není volný žádný hlas, je klávesa ignorována. Implicitně je ale zapnuté „kradení not“ zmíněné v kapitole 6.4. Po vytvoření syntetizéru tedy bylo nutné ho vypnout.

Před započetím přehrávání vlastního pluginu je potřeba syntetizéru dodat vzorkovací frekvenci. To se děje v metodě prepareToPlay uvnitř hlavního modulu. Tato frekvence je poskytnuta jednotlivým hlasům. Zvukům ji však musíme dodat sami.

Můžeme pozorovat jistou odchylku od designu, že generátor zrnek je přímo vlastněn správcem hlasů, místo aby existoval volně. Jedná se však pouze o miniaturní změnu.

Vzhledem k obecné implementaci syntetizéru však není možné měnit parametry jednotlivých hlasů a zvuků přímo skrz syntetizér. Místo toho si musíme pokaždé vyžádat seznam hlasů a zvuků, přetypovat je a teprve poté volat metody pro změnu jejich parametrů. Pro rychlejší přístup tedy odkazy na hlasy a zvuky v už přetypované podobě ukládáme do hlavního modulu. Samotné odkazy nemusíme s destrukcí pluginu ručně dealokovat. O to se stará syntetizér, kterým jsou hlasy a zvuky spravovány.

7.2.2 Zrnko

Samotné zrnko je implementováno strukturou Grain. Je velice jednoduchá, obsahuje pouze samotný audio obsah zrnka (automaticky dealokovaný při zničení zrnka), počet vzorků v zrnku a index následující vzorku zrnka k přehrání.

7.2.3 Generátor zrnek

Tento modul je implementován třídou DynamicGranulizedSampleSound. Jako taková má tedy především poskytovat na požádání zrnka z dodaného zvuku.

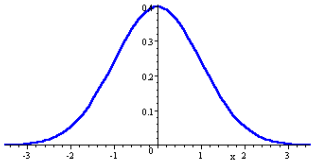
Jak bylo řečeno dříve, dědí ze třídy SynthesiserSound, aby bylo možné ji použít v syntetizéru. Přímo v konstruktoru je také vynucené dodání základních hodnot parametrů, abychom nemuseli řešit jejich dodatečné nastavování a případné opomenutí nastavení. Proč je to řešeno parametry konstruktoru a ne nastavením pomocí konstanty? Jak uvidíme později u souboru Utils, základní hodnoty parametrů jsou uchovávány v normalizované formě. Tato hodnota je nejprve nastavena parametru v hlavním modulu a následně, aby byla i při případných změnách implementace jistota, že jde o stejné hodnoty, je tento nastavený parametr denormalizován a předán konstruktoru generátoru.

Při nastavování parametrů je použitá filozofie, že hodnoty dodávané do modulů, kde mají být využity, by měly být denormalizované a v jednotkách, které jsou pro parametr určené. Tedy pokud je parametr v milisekundách, dostane hodnotu v milisekundách. Jisté uvolnění je v případě, že proměnná v generátoru je celočíselná, ale z venku přichází desetinná. V takovém případě je zaokrouhlovaná. Většina parametrů audio pluginů je typu float, a poskytnutí setterů s float parametrem a následné zpracování uvnitř setteru usnadňuje použití. Po nastavení parametrů je spočítána také jejich verze v počtu vzorků místo milisekund.

Nahrávání souboru je jednoduché - generátor dostane přímo audio buffer se vzorky zvuku a původní vzorkovací frekvenci. Pokud už známe výstupní vzorkovací frekvenci (a prakticky vždy ji nyní známe), vyrobíme i resamplovanou kopii původního zvuku. K resamplování je využita třída LagrangeInterpolator přímo z JUCE knihoven. Jedná se o 4-bodovou lagrangeovskou interpolaci. Stejný resamplovací postup se používá v případě, kdy dostaneme novou (různou od stávající) vzorkovací frekvenci. Je třeba mít na paměti, že dokud není známa vzorkovací frekvence a není nahrán zvuk, generátor produkuje pouze zcela prázdná zrnka. S každou změnou vzorkovací frekvence jsou také znovu přepočítané parametry s hodnotami v počtu vzorků.

Nejzajímavější z celého generátoru je samozřejmě samotné generování zrnek. V jeho jádru leží využití náhodného výběru, konkrétně pomocí normovaného normálního rozdělení (střední hodnota rovná nule, rozptyl jedna). To má ovšem problém, že možné hodnoty jím generované jdou teoreticky do nekonečna. Jak ale vidíme na Obr. 33, drtivá většina hodnot se nachází mezi -3 a 3, proto po vygenerování hodnotu omezíme na tento interval. Ve výsledku chceme dostat rozsah pro celou oblast výběru zrnka (rozsah vzorků, kterými může začínat zrnko), proto nejprve vydělíme náhodnou hodnotu třemi (dostane se do rozsahu -1 až 1) a následně vynásobíme polovinou velikosti oblasti (nakonec tedy leží v intervalu *-polovina velikosti oblasti* až *polovina velikosti oblasti*). Rozsah intervalu je sice o 1 větší než velikost oblasti (o nulu ve středu intervalu), ale vzhledem k tomu, že počet vzorků se pohybuje většinou ve stech až tisících, na takto malé odchylce nesejde. Poté spočítáme, na který vzorek ve vstupním zvuku je namapovaná stisknutá MIDI klávesa a toto číslo přičteme ke zpracované náhodné hodnotě. Tato hodnota tedy nakonec leží v rozsahu velikosti oblasti výběru se středem na vzorku stisknuté klávesy. Jak bylo zmíněno v designu, je nutné také tento rozsah omezit. Omezení zespodu je logicky 0, neboli první vzorek. Omezení seshora je *počet vzorků zvuku - počet vzorků zrnka*, abychom pokaždé dostali celé zrnko. Jediná výjimka je, pokud by celý zvuk byl kratší než zrnko - to bude první vzorek zrnka vždy nula.

Poté už jen zbývá zkopírovat vzorky do bufferu zrnka a aplikovat obálku. Oba tyto kroky se provádějí současně, aby se ušetřilo jedno procházení bufferu.



Obrázek 33 Normované normální rozdělení [37]

7.2.4 Hlas

Hlasy jsou implementovány třídou DynamicGranulizedSampleVoice. Pro připomenutí, stará se o přehrávání celé noty skládající se ze zrnek a obálku noty.

Obálka je řešená krokovou změnou úrovně hlasitosti, kde provedení kroku je zpracování jednoho vzorku výstupního bufferu, a jeho velikost je předpočítaná pomocí délky dané fáze (Attack, Decay, Release) v milisekundách, vzorkovací frekvence a v případě Decay a Release i Sustain úrovně. Popsáno jinak, o kolik se každý vzorek musí změnit hlasitost, abychom se z počáteční hodnoty (pro Attack 0, Decay 1 a Release Sustain úroveň) na konečnou (Attack 1, Decay Sustain a Release 0) za daný počet vzorků. Jakmile hlasitost dosáhne konečné hodnoty dané fáze, přepne se na další fázi. Do Release fáze se přepne po přijetí signálu o puštění klávesy, případně konci Decay fáze. Záleží, která situace nastane později. Spočítanou úrovní hlasitosti je následně vynásoben vzorek zrnka kopírovaný do výstupního bufferu.

ADSR parametry je možno kdykoli měnit. Z tohoto důvodu je nutné velikosti kroků dynamicky přepočítávat s každou změnou. Narazíme ale na problém, pokud změníme Sustain, pokud je nota v Release fázi. Pokud v Release změníme Sustain na 0, při počítání velikosti kroku zjistíme, že se snažíme dostat z hlasitosti 0 na hlasitost 0, takže velikost kroku by byla taktéž 0 a skončili bychom s nekonečným zvukem (až do další změny Sustainu). Proto, pokud měníme Sustain v Release fázi, musíme si nový Sustain uložit a změnit ho až po dokončení Release fáze, a do té doby pracovat se starým Sustainem.

Kromě této odlišnosti změna parametrů probíhá obdobně jako u generátoru zrnek - dodávané hodnoty jsou denormalizované.

Samotné plnění bufferu se příliš neliší od popisu v designu. Do výstupního bufferu se kopírují vzorky obou zrnek (případně jednoho, pokud ještě nedohrála polovina prvního zrnka) upravené o ADSR hlasitost. Informace, kolik vzorků bylo ze zrnka přehráno je ukládána do zrnka. Jakmile první zrnko dospěje do svojí poloviny, je vygenerováno druhé zrnko a dále se kopíruje z obou. Jakmile druhé zrnko dospěje do poloviny, znamená to, že první dohrálo a je odstraněno. Druhé zrnko nahradí první a místo druhého se vygeneruje nové. Kromě začátku tedy přehráváme vždy právě dvě zrnka. Takto se postupuje až do konce přehrávání, které nastane dokončením Release fáze.

7.2.5 Editor

Editor se stará o GUI pluginu a je implementován třídou PluginEditor.

Implementace se skoro neliší od jeho designu. Ovládání parametrů je provedeno pomoc sliderů, jak otočných, tak horizontálních, načítání souboru je skrz tlačítko a dialog. JUCE samotný následně poskytuje možnosti práce s velkým množstvím formátů souborů, mimo jiné wav, mp3, ogg, asf, takže se čtením souboru nebyl žádný problém. Nakonec se povedlo implementovat i zamýšlené MIDI klávesy.

Parametry pluginu jsou z editoru nastavovány včetně upozorňování plugin hostu.

Přenastavování hodnot v editoru při změně ze strany hostu řešíme pomocí periodického nastavování na hodnoty nacházející se v hlavním modulu.

Zobrazování hodnot parametrů už byla jiná otázka. Parametry jsou v uživateli nic neříkajících normalizovaných hodnotách. Vyvstává tedy otázka, kdy je převést do čitelné podoby? Za tímto účel jsme vytvořili NormalizedValueSlider, potomka třídy Slider, který vnitřně uchovává normalizovanou hodnotu, ale pro zobrazování ji denormalizuje. K tomu potřebuje znát minimální a maximální hodnoty jednotlivých parametrů. Ty jsou jako konstanty uvedeny v souboru Utils.h.

7.2.6 Utils

V souborech Utils.cpp a Utils.h se nachází všemožné pomocné funkce, například pro (de)normalizaci hodnot. Hlavní součástí je ale definice konstant, které uvádí minimální, maximální a základní hodnoty parametrů. Minima a maxima jsou uvedeny v denormalizovaných hodnotách a jsou použity pro normalizaci a denormalizaci hodnot jednak ve sliderech, jednak při nastavování parametrů generátoru a hlasů. Základní hodnoty, které jsou použity na přímé nastavování parametrů v hlavním modulu, jsou uložené v normalizovaném tvaru.

7.2.7 Hlavní modul

Hlavní modul je implementován třídou PluginProcessor. Ten má na starost především komunikaci s plugin hostem.

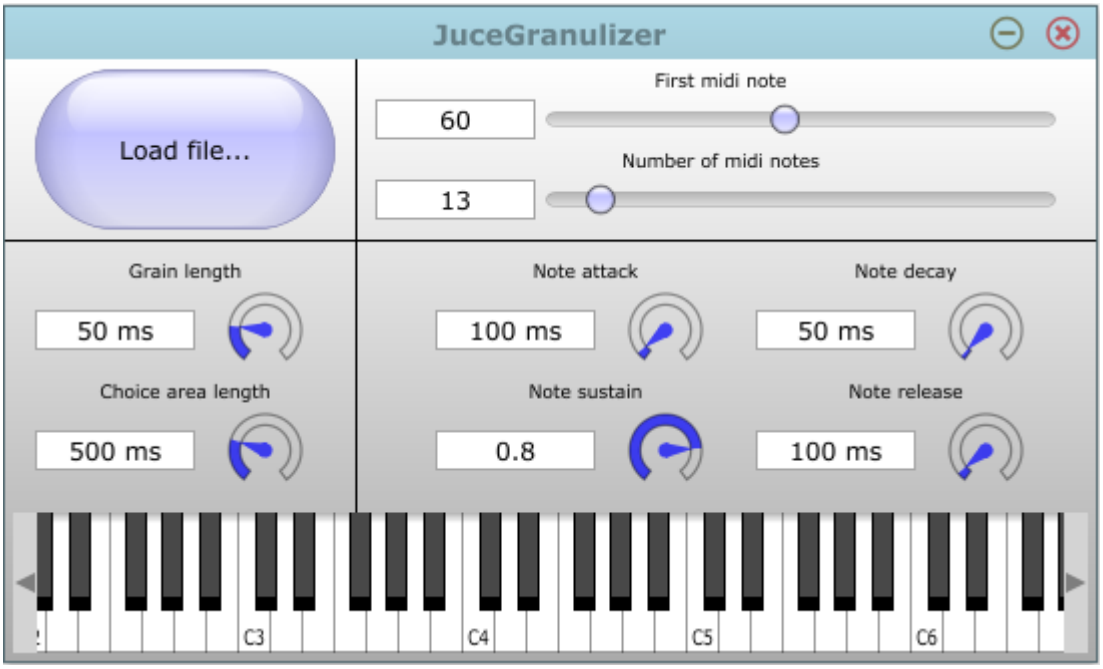
Opět se příliš neliší design a implementace. Jak už bylo zmíněno, parametry jsou v této třídě uloženy v normalizovaných hodnotách. Plugin host a editor k nim přistupují za pomocí indexace, o kterou se stará enumerace Parameters - jedná se zejména o zvýšení čitelnosti kódu. Pokud by se používala čistě čísla, velmi rychle bychom se ztratili.

Jak jsme podotkli u správce hlasů, nastavování parametrů ostatních částí pluginu provádíme pomocí odkazů na tyto moduly uložených jako proměnné v této třídě. Nejprv však musíme parametry denormalizovat.

Co se týče ukládání stavu, ten je ukládán v XML formátu, který je po dopsání serializován do paměťového bloku. Každá proměnná má svůj klíč, pod kterým je uložena. Samotný zvukový soubor nakonec ukládáme jako binární data zakódovaná do base64 stringu. Abychom ale ho vůbec mohli zakódovat, museli jsme ho při načítání načíst v podstatě dvakrát - jednou jako audio buffer se vzorky zvuku pro generátor zrnek, podruhé jako binární data pro účel uložení stavu. Při načítání stavu jsou jednotlivé parametry načteny a uložený soubor dekódován. Nesmíme však zapomenout tyto parametry poslat ostatním modulům, aby se skutečně dostali do požadovaného stavu.

7.2.8 Krátká ukázka

K práci je přiložená krátká ukázka v souboru `opilavcela.mp3`. Tato nahrávka ukazuje, že plugin je funkční a chová se dle očekávání. Plugin byl pouštěn pod vzorovým plugin hostem JUCE.



Obrázek 34 Ukázka GUI pluginu po implementaci

8 Testování implementace audio pluginu

Nyní máme hotový design a implementaci našeho pluginu. Dále na něm provedeme uživatelské testování, abychom se ujistili, že náš plugin je uživatelsky pochopitelný a splňuje očekávání. Jako u každého testování je taktéž možné, že budou nalezeny chyby, které nebyly odhaleny průběžným testováním během implementace.

8.1 Konfigurace testování

Testování bude nutné udělat mimo laboratoř, přímo u participantů. Naprostá většina vývojářů a skladatelů totiž má svoje vlastní prostředí, které mají nastavené podle svých vlastních potřeb a zvyklostí. Toto nastavení může zahrnovat i speciální hardware, nejtypičtěji klávesy, ale může se jednat i o velké mixovací stanice. Přechod do jiného by tedy vyvolal pouze zmatení a mohl by produkovat zavádějící výsledky testů. Zároveň je možné, že se nepovede nalézt lidi v dostupné vzdálenosti a bude nutné testování provést jako vzdálené.

Nemůžeme tedy zaručit jednotnost jednotlivých testovacích prostředí a měli bychom ji zohlednit při zpracování výsledků. Jediná jistota, kterou máme je, že plugin poběží v plugin hostu implementující VST rozhraní a všichni participanti dostanou stejné materiály.

8.2 Plán testování

Každý participant dostane k dispozici kopii pluginu, popis jeho funkcionality a ovládání, a krátký seznam otázek ohledně spokojenosti a návrhů na zlepšení. Distribuované materiály jsou psané v angličtině, zde uvádíme pouze jejich překlad.

8.2.1 Popis a ovládání

Juce Granulizer je VST plugin založení na technikách granulární syntézy. Dovoluje vám přehrávat zrnka zvuku proměnlivé délky generované z nějakého vstupního souboru kontrolovaným způsobem za pomoci MIDI kláves.

Jak to celé funguje: Vy (uživatel) nahrajete zvukový soubor (wav, mp3, ogg, asf nebo některý z mnoha jiných podporovaných formátů) do pluginu pomocí velkého tlačítka Load file.

Následně vyberete rozsah kláves na MIDI klávesách, které chcete používat. Rozsah je zvolen výběrem první klávesy rozsahu (kde 60 je C4, pokud první oktáva začíná tónem C-1) a následně počtem kláves.

Nahraný soubor je následně časově namapován na klávesy, kde začátek souboru je namapován na první klávesu a konec na poslední. Tedy pokud používáte 5 kláves, postupně by ukazovaly na body v 0/4, 1/4, 2/4, 3/4 a 4/4 souboru.

Poté už můžete bez obav začít hrát!

Můžete nastavit délku jednotlivých zrnek, měřeno v milisekundách. Prodleva mezi zrnky je konstantní, nové zrnko se přidá vždy v polovině předchozího zrnka. Společně

s použitím Hannovy obálky by takto, při stejně hlasitých zrnkách, měla být zachována konstantní hlasitost proudu zrnek.

Důležité nastavení je velikost oblasti výběru. Tento slider vám dovoluje vybrat velikost oblasti, ze které se budou zrnka náhodně vybírat. Každá MIDI klávesa má daný bod v nahraném souboru, a velikost oblasti definuje oblast kolem těchto bodů, ze kterých je možno zrnka brát. Pokud by velikost byla 0 ms, vždy byste dostali stejné zrnko při stisku stejné klávesy. Pokud ji nastavíte na 2000 ms, zrnko je náhodně vybráno (s použitím normálního rozdělení) z oblasti <-1000 ms; 1000 ms> kolem daného bodu.

Berte na vědomí, že pokaždé když je generováno zrnko (například když je stále držena klávesa), pokaždé se vybírá náhodně z výše definované oblasti a není žádná záruka, že bude stejné jako předchozí (pokud velikost oblasti výběru není 0, samozřejmě).

Nakonec můžete měnit ADSR obálku noty (stisku klávesy), ADR jsou měřené v milisekundách a S je podíl maximální úrovně hlasitosti.

Juce Granulizer byl vytvořen pomocí JUCE knihoven a je licencovaný pod General Public License. Kopie licence by měla být přiložená. Copyright 2015 Pavel Lieberzeit.

8.2.2 Seznam otázek

Otázky a odpovědi jsou čistě anonymní. Zároveň prosíme o informaci, v kterém prostředí plugin testujete.

1. Do jaké míry Juce Granulizer splnil vaše očekávání? (na stupnici 1 až 5, kde 1 je naprosto nespokojen, 3 naplnil očekávání, 5 velmi předčil očekávání)
2. Narazili jste na nějaké nesnáze nebo problémy během testování?
3. Co byste na pluginu změnili, případně přidali nebo odstranili?

8.3 Průběh a výsledky testování

Po sepsání materiálů se přistoupilo ke shánění participantů a testování. Výsledný počet participantů bohužel dopadl poněkud tragicky - byl nalezen pouze jeden participant. Možným důvodem může být mimo jiné relativní vzácnost audio programátorů. Testování se tedy nedá považovat za významné a důkladné a nad návrhy je potřeba se zamyslet, zda by byly užitečné obecně, či pouze pro daného člověka. I tak je ale možné, že dojdeme k zajímavým závěrům.

Zpracované odpovědi našeho participanta na položené otázky jsou následující:

1. 5 (s komentářem „THAT'S AWESOME“).
2. Participant neměl problémy s pochopením a orientováním se v pluginu. Při přehrávání konstantních zvuků (konkrétně komorní A) však při některých délkách zrnka zvuk zněl s jakoby mírně kolísavou hlasitostí namísto konstantní.
3. Participant navrhuje přidání tří funkcí. První je možnost volby prodlevy mezi jednotlivými zrnky, aby bylo možné dělat mezery nebo překryvy. Druhá je možnost nastavování délky zrnka v závislosti na tempu, konkrétně ve zlomcích beatu. Příklad: Pokud by tempo výstupu bylo 120 BPM[1], jeden beat by trval půl sekundy. Od této doby by se tedy počítaly další zlomky - 1/2, 1/3, 1/4, 1/6 a tak dále. Na tyto zlomky by se nastavovala délka zrnka. Třetí návrh se opět týká prodlevy a jde o přidání nebo odebrání náhodné doby k prodlevě mezi zrnky. Tuto dobu by mělo být možné nastavit.

[1] Beats Per Minute

8.4 Analýza výsledků testování

Jak je vidno, hodnocení je povzbudivé, přestože se jednalo pouze o jednoho člověka. Návrhy taktéž nevypadají k zahození a stojí za to zvážit, zda je přidat do pluginu. Nejprve ale analyzujme nalezený problém.

Při pokusech o reprodukování problému jsme zjistili, že skutečně při určitých délkách zrnka zní výsledek mírně „nasekaně“. Použitým zvukem byl tón A4 jako hladká sinusoida. Efekt se projevoval častěji, pokud velikost oblasti výběru byla nenulová, přestože se v takovém případě pochopitelně vyskytoval náhodněji. Došli jsme k závěru, že se nejpravděpodobněji jedná o špatnou návaznost zvukových vln jednotlivých zrnek a tím vzniklých oblastech většího ticha na koncích, případně i začátcích zrnek. V momentální implementaci se s tímto nedá nic dělat, některé z návrhů by však mohly pomoci.

Jako první návrh máme volitelnou prodlevu. Uznáváme, že tato možnost volby by dovolila různé překryvy a mezery a tím dala větší volnost. Původní konstantní prodleva byla použita kvůli zjednodušení ovládání a především stálé úrovni hlasitosti. Jak jsme ale měli možnost pozorovat na nalezeném problému, ne vždy je pak výsledek podle plánu. Navíc, i s volitelnou prodlevou se dá nastavit prodleva poloviny zrnka jako je v momentální implementaci, tedy nic neztratíme.

Druhý návrh se zabývá přidáním možnosti nastavovat délku zrnek podle tempa. Ve svém důsledku se jedná pouze o volitelnou změnu stylu vstupu a na samotné funkcionalitě syntézy by se neměnilo nic. S touto změnou by pak pomocí pluginu bylo možné vyrábět i zvuky sedící do rytmu skladby a tím zvýšit jejich líbivost. Jednoznačně doporučujeme, pokud to bude možné.

Poslední návrh je přidání náhodného prvku do prodlev mezi zrnky. Tato změna by ve své podstatě byla v duchu chaotičnosti granulární syntézy, ale nedá se použít vždy. Například s předchozí navrhovanou změnou by si příliš nerozuměla. V případě nastavování parametrů na zlomky beatů by se však tato náhodnost dala vypnout. I toto se dá tedy schválit, i když vzhledem k výhradám s nižší prioritou.

9 Revize implementace a designu po testování

V minulé kapitole jsme otestovali dosavadní implementaci. Z testování nám však vyvstalo několik možných (a dobrých) návrhů, jak plugin vylepšit. V této kapitole tyto změny zakomponujeme do designu a následně i implementujeme.

9.1 Úprava designu

Začneme s prvním navrhovanou úpravou, to jest volitelnou prodlevou. Tato změna se nejvíce dotkne samotných hlasů (právě ty se starají o přehrávání proudů zrnek), následně i hlavního modulu a editoru. Nejprve ale musíme promyslet, v jakém tvaru bude tato hodnota nastavována. Máme dvě možnosti - buďto brát hodnotu prodlevy relativně vůči konci posledního zrnka, nebo vůči jeho začátku. Sami se přikláníme spíše k druhé možnosti. S první bychom se totiž v GUI dostali k záporným číslům, pokud bychom chtěli nastavit překryv, druhak bychom museli spodní hranici rozsahu dynamicky přenastavovat podle momentální délky zrnka - jinak by mohla nastat situace, že by začátek následujícího zrnka nastal před začátkem předchozího zrnka, což by způsobilo značné problémy[1].

Mějme tedy prodlevu od začátku předchozího zrnka, v intervalu od 0 do nějaké maximální hodnoty. Toto maximum by mělo být vyšší, než maximální délka zrnka, aby se vždy dala udělat mezera mezi dvěma zrnky. Stejně jako pro jiné hodnoty, i pro prodlevu je ideální nastavování pomocí slideru. Následně v hlavním modulu budeme muset vyrobit nový parametr pro prodlevu a propojit nastavování parametru z plugin hostu a editoru s jeho nastavováním v samotném hlasu. Tam nyní máme verzi pro dvě zrnka. Toto musíme rozšířit pro proměnný počet zrnek. Určíme také maximální počet zrnek, které mohou být v jeden moment v jedné notě přehrávána. Toto omezení je především kvůli minimalistickým prodlevám - pokud by neexistovalo, během pár milisekund bychom se dostali do úděsných kakofonií a pomalému zpracovávání zvuku. Tato hodnota bude stanovená empiricky s příměsí zdravé úvahy, předběžně budiž 8. Hlas během přehrávání zrnek hlídá dobu od začátku posledního zrnka, a jakmile dosáhne doby prodlevy (nebo přesáhne, při náhlém přenastavení), nechá vygenerovat další zrnko a resetuje čítač. Pokud již hraje maximální počet zrnek, pouze resetuje čítač.

Druhá změna se týká nastavování délky zrnky podle tempa. Aby se dostavil požadovaný efekt synchronizace, měla by se tato změna dotknout i prodlevy mezi zrnky. Jedná se o změnu ovládání a zobrazování, dotkne se tedy pouze editoru a hlavního modulu. Požadované zlomky beatů jsou typicky používané, tedy 1/1, 1/2, 1/3, 1/4, 1/6, 1/8, 1/12, 1/16, 1/24, 1/32 a 0. Jak je vidno, nejde o nějaký lineární přechod. Namapování z normalizovaného parametru na tyto hodnoty tedy budeme muset provést složitěji a skokově. Následně narazíme na jiný problém - VST může vyžádat tempo (BPM) skladby pouze během samotného přehrávání. Nastavit hodnotu v generátoru zrnek a v hlasech tedy můžeme teprve až začneme přehrávat. Také se může stát, že host BPM nepodpo-

[1] Problémy, které mohou zahrnovat trhliny v časoprostoru, cestování časem a podobné věci.

ruje a plugin nedostane žádnou hodnotu. V tom případě budeme muset použít nějakou základní hodnotu. Po ruce je standardních 120 BPM. Zároveň musíme vyrobit i parametr sloužící jako přepínač, zda se zrovna používají beaty nebo milisekundy. I pro délku a prodlevu v beatech budou potřeba nové parametry - milisekundové parametry by si měly uchovat svoji hodnotu nezávisle na nastavení v beatech, jednak kvůli skokovému přepínání beatových hodnot, jednak kvůli možnosti rovnou přejít na původní nastavení před beaty. Také musíme při změnách parametrů dávat pozor, zda se zrovna používají beaty nebo milisekundy, a generátoru zrnek a hlasům předávat pouze hodnoty právě zapnuté možnosti.

Co se týče zobrazování, pro parametry v beatech nám opět poslouží slidery. Pro přepínání mezi beaty a milisekundami můžeme použít nějaký přepínač nebo checkbox. Zobrazování hodnot nebude tak jednoduché jako u milisekundových parametrů. Aby bylo srozumitelné, hodnoty u sliderů by měly být zobrazované jako zlomky. K tomu můžeme využít podobný proces jako u mapování normalizovaného parametru na zlomek beatu, akorát tuto hodnotu ještě převedeme do podoby zlomku. Také bychom měli vhodně zvýraznit, který mód je zrovna používaný. Samotný stav spínače nebo checkboxu může být nevýrazný. Vhodným způsobem je typické „zašednutí“ a zneaktivnění ovládacího prvku. V případě, že hodnota zneaktivněného prvku je změněna ze strany plugin hostu, měli bychom ji na ovládacím prvku také změnit. Co se týče rozmístění v GUI, prvky s milisekundovými hodnotami by měly být poblíž sebe, stejně jako prvky s beatovými hodnotami by měly být blízko sebe. Jejich relativní rozmístění uvnitř skupiny by mezi těmito dvěma skupinami také mělo být stejné. To znamená, že pokud by milisekundová délka zrnka byla nalevo od milisekundové prodlevy, tak by také beatová délka zrnka měla být nalevo od beatové prodlevy. Přepínač módu by se měl nacházet mezi těmito dvěma skupinami.

Třetí úprava, náhodný prvek u prodlevy mezi zrnky, nebude tak náročný na implementaci. Ve své podstatě musíme přidat další parametr a slider pro něj do GUI. Je to čistě milisekundová hodnota, takže její vytvoření nebude složité. Slider a předávání hodnoty dál do pluginu bychom však měli vypnout, pokud používáme mód parametrů ve zlomcích beatu - v tomto případě chceme hlavně synchronizaci s tempem a tyto jemné odchylky by působily rušivě. Hodnota parametru nakonec bude směřovat do hlasů, kde určí rozsah náhodných hodnot, ze kterého bude vygenerována hodnota, která bude přičtena k nastavené prodlevě. Prodleva se tím prodlouží, či v případě záporného čísla zkrátí. Rozsah náhodných hodnot bude určen podobně, jako u velikosti oblasti výběru. Parametr se bude pohybovat v hodnotách 0 a výše a bude označovat velikost celého rozsahu. Rozsah potom bude sahat od *-polovina hodnoty parametru* do *polovina hodnoty parametru*, odkud se přímo vygeneruje náhodné číslo upravující prodlevu. Použité rozdělení tentokrát nerozhoduje, můžeme použít klasické rovnoměrné.

9.2 Úprava implementace

V této sekci naimplementujeme změny nadesignované výše.

Většina první změny, proměnlivé prodlevy mezi zrnky, zejména co se týče GUI a nastavování parametru, je prakticky stejná jako u předchozích parametrů a není potřeba ji rozvádět. Změny v samotném hlasu už jsou obsáhlejší. Nejprve jsme místo původních pevných dvou zrnek vytvořili pole o délce maximálního počtu současně hrajících zrnek v jednom hlasu (momentálně 8). Do tohoto je vždy po uplynutí prodlevy vloženo nové zrnko na první volnou pozici a prodleva je resetována. Pokud není volné žádné místo, je pouze resetován čítač. Renderování zvukového bloku funguje víceméně stejně jako

předtím - postupně projdeme všechna zrnka a každé do výstupního bufferu přidá svoje vzorky. Jediný rozdíl je v počtu zrnek, která mohou v jednom cyklu najednou přidávat. Jakmile z nějakého zrnka přehrajeme všechny vzorky, odstraníme ho a následující zrnka v poli posuneme, abychom zaplnili mezeru. Teoreticky by mohl pro tento účel být praktičtější spojový seznam, ale u pole můžeme využívat přímou indexaci, nezdržujeme se dynamickou alokací a destrukcí kontejnerů prvků spojového seznamu a nakonec, s počtem prvků, se kterými pracujeme, nám posouvání prvků v poli příliš času nezabere. JUCE navíc má šikovné a optimalizované třídy pro takováto pole, takže není závažný důvod je nepoužít.

Druhá změna, volba délky zrnka a prodlevy mezi zrnky v závislosti na tempu skladby, už je složitější. Nejprve musíme vymyslet zobrazení mezi normalizovaným parametrem a jednotlivými možnými zlomky (1/1, 1/2, 1/3, 1/4, 1/6, 1/8, 1/12, 1/16, 1/24, 1/32 a 0). Za tímto účelem jsme vytvořili enumeraci s prvky reprezentujícími jednotlivé zlomky. Vzhledem k tomu, že se jedná o enumeraci, mají tyto prvky hodnoty od nuly do délky enumerace mínus jedna. Následně jsme stanovili minimální a maximální zlomek, kterých můžou oba parametry dosáhnout - od 1/2 do 1/32 pro délku zrnka a od 1/1 do 0 pro prodlevu. Prodleva má větší rozsah - na rozdíl od délky zrnka může být i nulová, a horní hranice by měla vždy být vyšší než pro délku zrnka, aby vždy šla vyrobit mezera mezi zrnky. Ve své podstatě se, vzhledem k tomu, že stále jde o prvky enumerace, jedná o interval přirozených čísel mezi dvěma hodnotami. Do tohoto intervalu převedeme normalizovaný parametr a následně zaokrouhlením na celá čísla zjistíme, který prvek enumerace, a tím i který zlomek hodnota představuje. K následnému spočtení délky (případně prodlevy) v milisekundách, abychom mohli hodnotu předat generátoru zrnek (nebo hlasu), potřebujeme už jen tempo (BPM). Z něj spočteme, jak dlouhý je jeden beat a následný převod zlomku jednoho beatu na milisekundu je triviální.

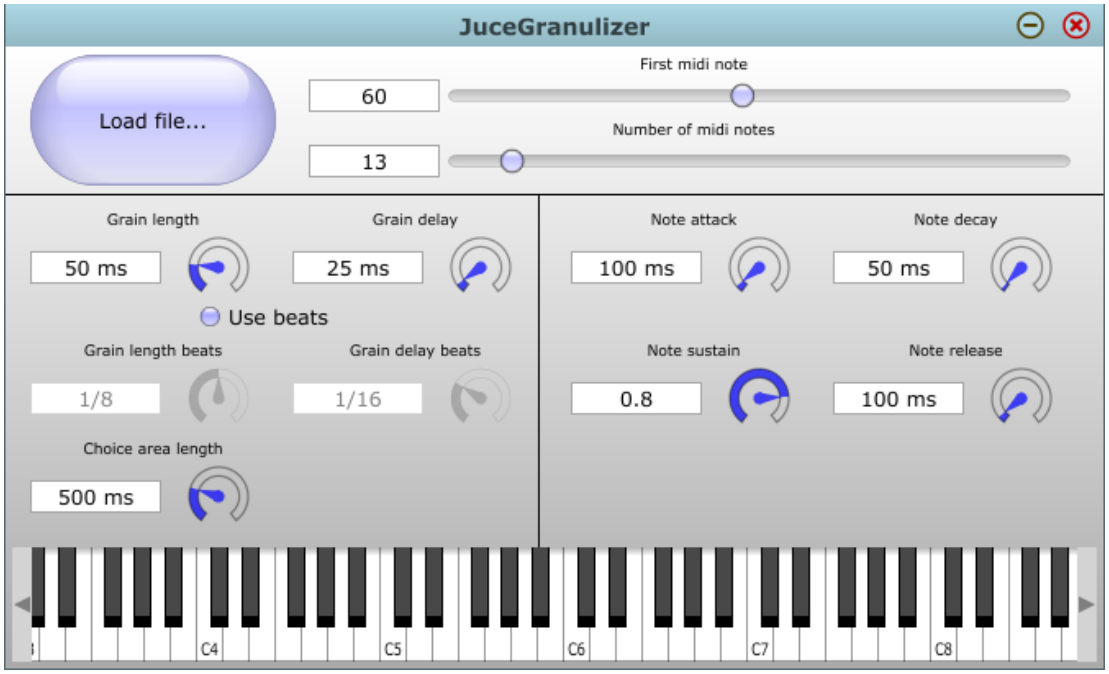
BPM dostaneme od plugin hosta během přehrávání, konkrétně v metodě žádající naplnění výstupního bufferu v hlavním modulu, a nepředpokládáme, že se tato hodnota bude během přehrávání měnit. Můžeme si ji tedy uložit a při přenastavení parametrů nemusíme čekat na další příležitost ji zjistit. Jak jsme ale podotkli v designu, může se stát, že host BPM poskytovat nebude. V takovém případě se budeme muset spokojit s pevně nastaveným standardním 120 BPM.

Pro zobrazování zlomku beatu u slideru jsme vytvořili novou třídu, BeatsNormalizedValueSlider, dědící z NormalizedValueSlider, kterou jsme popsali v předchozích kapitolách. Chová se téměř stejně jako mateřská třída. Má však přetížené metody na převádění své vnitřní hodnoty na text a naopak. Pro směr z hodnoty na text je využitý postup denormalizace pomocí enumerace popsaný v předminulém odstavci. Tím zjistíme, o který zlomek se jedná a není problém jednotlivé hodnoty přiřadit k textovým reprezentacím. V opačném směru je situace složitější. Hodnoty do vstupního pole je nutné psát ve tvaru *1/y*. U této hodnoty nejprve zkontrolujeme, zda je v tomto tvaru, a následně jestli čitatel zlomku je jedna. Poté zjistíme, jestli jmenovatel odpovídá nějakému ze známých jmenovatelů. Jestliže ano, nastavíme vnitřní hodnotu slideru na odpovídající hodnotu. V případě špatného vstupu nebo neznámého zlomku je slider automaticky nastaven na nejnižší hodnotu.

Samotné přepínání módů jsme implementovali tak, jak bylo popsáno. Využili jsme k tomuto účelu checkbox. Mapování z normalizovaného parametru je jednoduché - jedná se čistě o zaokrouhlení na celé číslo, a výsledek 0 znamená hodnotu false (nejsou zapnuté beaty) a 1 zase true (jsou zapnuté beaty).

Ovládací panel pluginu jsme museli kvůli novým prvkům zvětšit a přeskládat. Výsledek můžeme vidět na Obr. 35.

Na implementaci poslední změny bohužel nezbyl dostatek času. Stejně tak bychom



Obrázek 35 Ukázka GUI pluginu po revizi implementace

měli tuto revizi pluginu znovu otestovat, ale časové podmínky tomu bohužel opravdu nejsou nakloněné a bude nutné testování provést v budoucích pracích.

10 Závěr

Zde se dostáváme k závěru práce, je tedy vhodné zrekapitulovat naše cíle a do jaké míry jsme jich dosáhli.

Nejprve jsme se měli seznámit s různými metodami syntézy zvuku a zejména s granulární syntézou. Výsledky tohoto obsáhlého výzkumu je možné nalézt v kapitolách 2 až 4. Postupně jsme prošli od úplných základů až po jednotlivé složité metody granulární syntézy.

Další část výzkumu bylo seznámení se s Digital Audio Workstation (DAW) a formáty pluginů, které se v nich dají používat. Tuto část nalezneme v kapitole 5. Byly v ní popsány výhody DAW oproti analogovému zpracovávání zvuku a mírně rozveden typický postup při úpravě zvuku a skládání hudby. Průzkum formátů pluginů byl omezen na nejpoužívanější a rozdíly mezi nimi byly popsány rámcově, avšak v postačujícím rozsahu pro účely této práce.

Poté se přistoupilo k samotnému vývoji plánovaného syntezátoru založeném na vlastní technice granulární syntézy. Daná technika byla popsána v kapitole 6, a využívá vstupní zvuk dodaný uživatelem, který je následně namapován podle nastavených parametrů na MIDI klávesy. Jednotlivé klávesy potom přehrávají náhodná zrnka z namapovaných oblastí vstupního zvuku. Ve stejné kapitole též nalezneme design našeho syntezátoru včetně návrhu ovládání a jednotlivých potřebných komponent.

Následná implementace (kapitola 7) probíhala hladce. Po porovnání dostupných plugin formátů byl pro syntezátor vybrán formát VSTi verze 2, kompilovaný pomocí SDK VST3 s wrappery pro kompilaci do VST 2. Jediný větší problém, na který se narazilo, byl na samém počátku při používání VST SDK, kde se zjistilo, že samotné SDK je nešikovné na počáteční nastavení a používání. Tento problém byl vyřešen použitím frameworku JUCE. Zbylé problémy byly pouze menšího charakteru a byly vyřešeny a syntezátor byl úspěšně realizován jako VSTi plugin.

Uživatelské testování, k nalezení v kapitole 8, neproběhlo zcela podle plánu vzhledem k nízkému počtu participantů. Nízká účast byla zohledněna a testování i tak poskytlo cenné podklady pro další vývoj.

Po testování následovala v kapitole 9 revize designu a implementace, kde jsme využili návrhy pro opravu a rozšíření našeho syntezátoru. Design byl úspěšně upraven. Implementace mírně škobrtla při zjišťování tempa od plugin hostu, ale tento problém byl ošetřen.

Syntezátor je možné použít s jakýmkoli vstupním zvukem v jednom z podporovaných formátů, avšak při použití příliš rychle se měnících zvuků (příkladem budiž i obyčejné písničky) ztrácí uživatel efektivní kontrolu nad zvuky, které mohou jednotlivé klávesy generovat. Může to být samozřejmě záměr, ale z našeho pohledu jde spíše o méně využitelné užití.

Pokud uživatelův plugin host nepodporuje předávání tempa skladby pluginům, při udávání délky zrnka a prodlevy mezi zrnky ve zlomcích beatu bude použito pevné tempo 120 beats per minute.

Syntezátor je distribuován pod General Public License. Kopii licence je možné nalézt přiloženou ke zdrojovým souborům.

10.1 Budoucí práce

Může se zdát, že syntezátor je hotov a není už na něm co dělat. Naopak, vždy je možné vymyslet další úpravy. U každé úpravy je však potřeba nejprve zvážit, zda by plugin příliš nezesložitila a nebyla tím na škodu. Následuje několik návrhů.

Výběr typu obálek zrnka a noty
V momentální verzi je obálka zrnka pevně daná (je použita Hannova obálka) a nota využívá nastavitelnou ADSR obálku. Pokud by bylo možné pro obě obálky nabídnout několik druhů obálek, například trojúhelníková, s exponenciálním útlumem a další, uživatel by byl schopný vytvářet větší škálu zvuků.

Výběr typu náhodného rozdělení pro výběr zrnka
Při výběru zrnka z oblasti kolem stisklé klávesy je nyní používané normální rozdělení. Výběrem typu tohoto rozdělení, například místo normálního použít rovnoměrné, by se ovlivnil styl náhodného výběru zrnek a tím náhodnost výsledného zvuku.

Nastavování rozsahu MIDI kláves
Nastavování rozsahu MIDI kláves lze usnadnit automatickým zjišťováním rozsahu podle dvou stisknutých kláves. Toto zjišťování by mohlo být nastartováno stisknutím příslušného ovládacího prvku (například tlačítka) a následným stiskem dvou MIDI kláves, které by určily hranice rozsahu.

Příloha A

Uživatelský manuál k syntezátoru

Juce Granulizer je VST plugin založení na technikách granulární syntézy. Dovoluje vám přehrávat zrnka zvuku proměnlivé délky generované z nějakého vstupního souboru kontrolovaným způsobem za pomoci MIDI kláves.

Jak to celé funguje: Vy (uživatel) nahrajete zvukový soubor (wav, mp3, ogg, asf nebo některý z mnoha jiných podporovaných formátů) do pluginu pomocí velkého tlačítka Load file.

Následně vyberete rozsah kláves na MIDI klávesách, které chcete používat. Rozsah je zvolen výběrem první klávesy rozsahu (kde 60 je C4, pokud první oktáva začíná tónem C-1) a následně počtem kláves.

Nahraný soubor je následně časově namapován na klávesy, kde začátek souboru je namapován na první klávesu a konec na poslední. Tedy pokud používáte 5 kláves, postupně by ukazovaly na body v 0/4, 1/4, 2/4, 3/4 a 4/4 souboru.

Poté už můžete bez obav začít hrát!

Můžete nastavit délku jednotlivých zrnek a prodlevu mezi nimi, měřeno v milisekundách. Prodleva je měřená od počátku zrnka do počátku následujícího. Také je možné využít nastavení těchto dvou parametrů podle tempa skladby. Zaškrtnutím checkboxu Use beats tento mód aktivujete a následně můžete dvěma právě aktivovanými slidery nastavovat délku a prodlevu ve zlomcích beatu. Pokud však váš plugin host nepodporuje nastavování tempa skladby, bude v tomto módu použita základní hodnota 120 BPM.

Důležité nastavení je velikost oblasti výběru. Tento slider vám dovoluje vybrat velikost oblasti, ze které se budou zrnka náhodně vybírat. Každá MIDI klávesa má daný bod v nahraném souboru, a velikost oblasti definuje oblast kolem těchto bodů, ze kterých je možno zrnka brát. Pokud by velikost byla 0 ms, vždy byste dostali stejné zrnko při stisku stejné klávesy. Pokud ji nastavíte na 2000 ms, zrnko je náhodně vybráno (s použitím normálního rozdělení) z oblasti <-1000 ms; 1000 ms> kolem daného bodu.

Berte na vědomí, že pokaždé když je generováno zrnko (například když je stále držena klávesa), pokaždé se vybírá náhodně z výše definované oblasti a není žádná záruka, že bude stejné jako předchozí (pokud velikost oblasti výběru není 0, samozřejmě).

Nakonec můžete měnit ADSR obálku noty (stisku klávesy), ADR jsou měřené v milisekundách a S je podíl maximální úrovně hlasitosti.

Juce Granulizer byl vytvořen pomocí JUCE knihoven a je licencovaný pod General Public License. Kopie licence by měla být přiložená.

Příloha B

Obsah přiloženého DVD

Cesta	**Popis**
JuceGranulizer	Adresář syntezátoru
JuceGranulizer/Builds /VisualStudio2013	Adresář obsahující solution pro Visual Studio 2013
JuceGranulizer/JuceLibraryCode	Soubory JUCE knihovny potřebné pro projekt
JuceGranulizer/Source	Zdrojové kódy syntezátoru
JuceGranulizer/LICENSE.txt	GPL licence přiložená k projektu
JuceGranulizer/README.txt	Instrukce pro používání syntezátoru v EN a CZ
JuceGranulizer.dll	Zkompilovaný VSTi 2.x plugin
opilavcela.mp3	Ukázka užití první implementace syntezátoru
Granularní_zvukový _syntezátor_Lieberzeit.pdf	PDF verze této práce
zdroj_textu	Adresář se zdrojovými soubory tohoto textu

Literatura

[1] CAHILL, Thaddeus. “ART OF AND APPARATUS FOR GENERATING AND DISTRIBUTING MUSIC ELECTRICALLY”. US 580035 A. 1897.

[2] *Types of Synthesis*. URL: http://www.sonicspot.com/guide/synthesistypes.html (cit. 17. 01. 2015).

[3] BERSERKERUS. *File:Amfm3-en-de.gif*. URL: http://commons.wikimedia.org/wiki/File:Amfm3-en-de.gif (cit. 18. 01. 2015).

[4] BENTHAM, Jeremy. *Doctor Who: The Early Years*. 1986. ISBN: ISBN 0-491-03612-4.

[5] HORNER, Andrew, BEAUCHAMP, James a HAKEN, Lippold. “Methods for Multiple Wavetable Synthesis of Musical Instrument Tones”. In: *J. Audio Eng. Soc.* 41 (5 1993).

[6] COHEN, Hendrik Floris. *Quantifying Music*. 1984. ISBN: ISBN 978-94-015-7686-4.

[7] GABOR, Dennis. “Theory of Communication”. In: *The Journal of the Institution of Electrical Engineers* 93 (3 1946).

[8] GABOR, Dennis. “Acoustical quanta and the theory of hearing”. In: *Nature* 159 (4044 1947).

[9] XENAKIS, Iannis. *Formalized Music: Thought and Mathematics in Composition, revised edition*. Pendragon Press, 1992.

[10] ROADS, Curtis. “Introduction to Granular Synthesis”. In: *Computer Music Journal* 12 (2 1988).

[11] ROADS, Curtis a další. *The Computer Music Tutorial*. The MIT Press, 1996.

[12] TRUAX, Barry. “Real-time granular synthesis with a digital signal processor”. In: *Computer Music Journal* 12 (2 1988).

[13] TRUAX, Barry. *Riverrun*. Digital Soundscapes. Wergo.

[14] OPIE, Timothy. “Sound in a nutshell: Granular Synthesis”. BA Honours. La Trobe University, Melbourne, 1999.

[15] KRZYZANIAK, Mike. *Audio Synthesis*. URL: http://michaelkrzyzaniak.com/AudioSynthesis/2_Audio_Synthesis/11_Granular_Synthesis/1_Window_Functions/ (cit. 19. 01. 2015).

[16] BURK et al. *Music and Computers. The Department of Music at Columbia University*. Columbia University. URL: http://music.columbia.edu/cmc/musicandcomputers/chapter4/04_08.php (cit. 22. 01. 2015).

[17] TRUAX, Barry. “Composing with Real-time Granular Sound”. In: *Perspectives of New Music* 28 (2 1990).

[18] ROADS, Curtis. *Microsound*. The MIT Press, 2004. ISBN: ISBN 9780262681544.

[19] KAEGI, Werner a TEMPELAARS, Stan. “VOSIM - A New Sound System”. In: *Journal of the Audio Engineering Society* 26 (6 1978).

[20] KAEGI, Werner. "The MIDIM system". In: *ICMC 1983*. ICMC, 1983.

[21] DE POLLI, Giovanni a PICCIALLI, Aldo. "Pitch-synchronous granular synthesis". In: *Representations of musical signals*. MIT Press, 1991.

[22] ROADS, Curtis. "Sound composition with pulsars". In: *Journal of the Audio Engineering Society* 26 (6).

[23] ROADS, Curtis. "The evolution of granular synthesis: an overview of current research". In: *International Symposium on The Creative and Scientific Legacies of Iannis Xenakis*. 2006.

[24] FINE, Thomas. "The Dawn of Commercial Digital Recording". In: *ARSC Journal* 39 (1 2008).

[25] *What is Live? Learn more about Ableton's music making software*. Ableton. URL: `https://www.ableton.com/en/live/` (cit. 10. 05. 2015).

[26] *Start:* Steinberg. URL: `http://www.steinberg.net/en/products/cubase/start.html` (cit. 10. 05. 2015).

[27] *REAPER | Audio Production Without Limits*. Cockos. URL: `http://www.reaper.fm/` (cit. 10. 05. 2015).

[28] JOHNSON, Derek a POYSER, Debbie. "Steinberg Cubase VST". In: *Sound on Sound* (1996).

[29] *KVR: Steinberg releases VST 3 SDK*. URL: `http://www.kvraudio.com/news/steinberg_releases_vst_3_sdk_8522` (cit. 10. 05. 2015).

[30] *Avid | Audio Plug-In Developer Program*. URL: `http://www.avid.com/US/partners/audio-plugin-dev-program` (cit. 05. 05. 2015).

[31] NIEMITALO, Olli. *File:Window function and frequency response - Hann.svg*. URL: `http://commons.wikimedia.org/wiki/File:Window_function_and_frequency_response_-_Hann.svg` (cit. 28. 04. 2015).

[32] katja. *pitch shifting*. URL: `http://www.katjaas.nl/pitchshift/pitchshift.html` (cit. 28. 04. 2015).

[33] *Avid charging plugin manufacturers annual fees for AAX format?* URL: `https://www.gearslutz.com/board/music-computers/889061-avid-charging-plugin-manufacturers-annual-fees-aax-format.html` (cit. 05. 05. 2015).

[34] *JUCE Cross-Platform C++ Library*. URL: `http://www.juce.com/` (cit. 05. 05. 2015).

[35] *olilarkin/wdl-ol - GitHub*. URL: `https://github.com/olilarkin/wdl-ol` (cit. 05. 05. 2015).

[36] *JUCE vs. WDL-OL vs. ?* URL: `http://www.kvraudio.com/forum/viewtopic.php?p=5755304` (cit. 05. 05. 2015).

[37] *14. Normal Probability Distributions*. URL: `http://www.intmath.com/counting-probability/14-normal-probability-distribution.php` (cit. 05. 05. 2015).