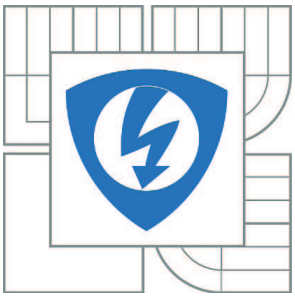
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

MODERNÍ ALGORITMY POSUNU VÝŠKY ZÁKLADNÍHO TÓNU A JEJICH VYUŽITÍ VE VIRTUÁLNÍCH HUDEBNÍCH NÁSTROJÍCH

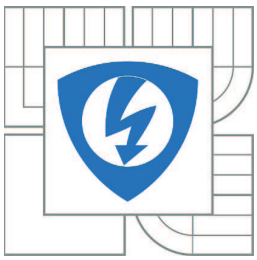
MODERN PITCH-SHIFTING ALGORITHMS AND ITS APLICATION IN VIRTUAL MUSICAL INSTRUMENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ALEŠ KŘUPKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL TRZOS
SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Aleš Křupka ***ID:*** 98175
Ročník: 2 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Moderní algoritmy posunu výšky základního tónu a jejich využití ve virtuálních hudebních nástrojích

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte moderní algoritmy číslicového zpracování signálů pro posun výšky základního tónu. Vybrané algoritmy implementujte v prostředí Matlab a porovnejte z hlediska věrnosti posunu výšky základního tónu, výskytu artefaktů a výpočetní náročnosti. Optimální algorimus použijte v jednoduchém virtuálním nástroji typu sampler. V navazující diplomové práci implementujte algoritmus virtuálního nástroje v jazyce C++ v podobě zásuvného modulu technologie VST. Virtuální nástroj doplňte o řízení pomocí MIDI protokolu. Při realizaci používejte nástroje pro verzování zdrojového kódu a nástroje pro automatické generování dokumentace ze zdrojového kódu.

DOPORUČENÁ LITERATURA:

[1] ZÖLZER, U. DAFX - Digital Audio Effects, 1st ed. New York: John Wiley & Sons, Ltd, 2002, 533 p. ISBN 0-471-49078-4.

Termín zadání: 7.2.2011 ***Termín odevzdání:*** 26.5.2011

Vedoucí práce: Ing. Michal Trzos

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Cílem diplomové práce je seznámení s metodami posunu základního tónu akustických signálů. Teoretická část práce obsahuje popis tří různých technik pro posun základního tónu, je to metoda s využitím modulace zpožďovací linky, metoda PICOLA a metoda s využitím fázového vokodéru. První dvě metody jsou zástupci zpracování v časové doméně, třetí metoda představuje zpracování v kmitočtové oblasti. V souvislosti s metodou PICOLA jsou v práci také zmíněny algoritmy pro odhad základní periody signálu. V závěru teoretické části jsou metody srovnány z různých hledisek.

Praktická část demonstruje využití těchto metod. Je zde popsán virtuální hudební nástroj sampler založený na přehrávání zvuků uložených v paměti. V této části jsou popsány jednotlivé funkční celky zajišťující požadovanou činnost sampleru. Generování zvuků virtuálního nástroje je řízeno pomocí MIDI protokolu. Pro generování různých tónů z jednoho uloženého zvuku je použita metoda PICOLA.

Abstract

This diploma thesis deals with pitch shifting methods of acoustical signals. The theoretic part of this thesis involves desription of three different pitch shifting techniques, these are the method using a modulated delay line, PICOLA method and method using a phase vocoder. The first two methods represent the processing in time domain, the third method represents the processing in frequency domain. In relation with the PICOLA method, the thesis also mentions algorithms for pitch estimation.

The practical part demonstrates the use of these methods. There is described a sampler virtual musical instrument based on the playback of the sounds stored in memory. In this part the particular units providing the required functionality are described. The generating of sounds is controlled by the MIDI protocol. In the sampler is implemented the PICOLA method.

Klíčová slova

Posun základního tónu, zpožďovací linka, fázový vokodér, sampler, VST.

Keywords

Pitch shifting, delay line, phase vocoder, sampler, VST.

KŘUPKA, A. *Moderní algoritmy posunu výšky základního tónu a jejich využití ve virtuálních hudebních nástrojích.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 48 s. Vedoucí diplomové práce Ing. Michal Trzos.

Prohlášení

Prohlašuji, že svoji diplomovou práci na téma „Moderní algoritmy posunu výšky základního tónu a jejich využití ve virtuálních hudebních nástrojích“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 26. 5. 2011 .....
podpis autora

Poděkování

Děkuji svému vedoucímu diplomové práce Ing. Michalovi Trzosovi za odborné vedení a pomoc poskytnutou při řešení mé diplomové práce.

Obsah

Seznam obrázků

1 Úvod

Tato diplomová práce na téma Moderní algoritmy posunu výšky základního tónu a jejich využití ve virtuálních hudebních nástrojích se zabývá metodami posunu základního tónu, tzv. pitch shifting, použitelných v různých hudebních aplikacích. Cílem je objasnit pojem výšky tónu a princip metod, které jsou schopny tuto výšku modifikovat za účelem dosažení požadované korekce tónu či hudebního efektu.

Nejprve je objasněn nejjednodušší princip změny výšky tónu, a to pomocí změny rychlosti přehrávání. Jelikož jeho výsledky nesplňují požadavky kladené na změnu výšky tónu, jsou uvedeny dvě další techniky pro změnu výšky tónu v časové doméně. První metoda je založena na modulaci zpožďovací linky. Druhá metoda z názvem PICOLA pro svoji realizaci potřebuje být vybavena postupem pro odhad základní periody signálu, proto je v dané katipole věnována pozornost i této problematice. Třetí metoda je typickým představitelem zpracování časových segmentů v kmitočtové oblasti, využívá principu fázového vokodéru. Po rozboru jednotlivých metod následuje srovnání všech metod.

Do řešení diplomové práce je také včleněna kapitola pojednávající o komunikačním rozhraní MIDI. Důvodem bylo nastudovat komunikační protokol využívaný audiovizuálními zařízeními, aby tohoto protokolu mohlo být následně využito pro návrh vlastního virtuálního hudebního nástroje. Práce se dále zmiňuje o problematice převzorkování číslicového signálu, tato technika je dále využita v praktické části.

Další kapitola pak obsahuje samotný návrh virtuálního hudebního nástroje typu sampler, jenž pro svoji činnost využívá techniky pro změnu výšky tónu. Pomocí diagramů tříd a vývojových diagramů je zde naznačena jeho implementace.

Poslední kapitola stručně popisuje verzovací systém Subversion a nástroj pro dokumentaci Doxygen, které jsou při řešení diplomové práce používány.

2 Výška tónu

Hudební tón se z fyzikálního hlediska od obecných zvuků liší především tím, že při analýze průběhu takového zvukového signálu lze vysledovat jistou periodicitu. V případě analýzy periodického signálu pomocí Fourierovy řady lze nalézt základní harmonickou složku, což je složka o nejnižším absolutním kmitočtu ze všech složek obsažených ve spektru sledovaného zvuku. Tato spektrální složka určuje tzv. výšku tónu. Jelikož je pro lidské vnímaní zvuku typické právě vnímání jeho spektrálního složení, je pojem výška tónu hlavním parametrem popisující tón. Kmitočet základních tónů používaných v hudbě se obvykle pohybuje od 16 do 4200 Hz.

Kromě základní harmonické složky tvoří spektrum hudebního tónu také vyšší harmonické složky, jejichž poloha ve spektru je určena násobky základního kmitočtu. Podle toho, které z vyšších harmonických složek jsou ve spektru zvuku obsaženy, je určen charakter hudebního tónu, který nazýváme barva tónu. Proto dva tóny zahrané na různé hudební nástroje zní odlišně, i když jejich základní harmonická složka nabývá stejné hodnoty. Rozdíl spočívá v různém rozložení vyšších harmonických složek, které je pro oba hudební nástroje odlišné. Čím více jsou vyšší harmonické složky ve spektru signálu zastoupeny a čím větší je jejich intenzita, tím zní hudební tón ostřeji. Naopak s klesajícím počtem vyšších harmonických složek získává hudební tón měkčí zabarvení [10].

2.1 Změna výšky tónu

Tato diplomová práce se zabývá metodami změny výšky tónu, tzv. Pitch shifting. Tento proces může být v hudebním odvětví použit k různým účelům, např. pro korekci hudebních nahrávek ve studiu, v hudebních nástrojích typu sampler, kde může být potřeba náhrávku (sample) představující určitý tón nástroje transponovat za účelem produkování tónů o různých výškách, či jako hudební efekt, kdy je k vstupnímu signálu z hudebního nástroje přičtena jeho transponovaná varianta a lze tak tón nástroje obohatit.

Aby bylo možné docílit této modifikace, je třeba hudební signál upravit takovým způsobem, kdy základní harmonická složka společně s vyššími harmonickými změní polohu ve výsledném spektru signálu. V případě translace každé spektrální složky o jistou pevně danou hodnotu f_p docílíme tzv. kmitočtového posunu (frequency shifting), jehož výsledek však neuspokojí naše požadavky, jelikož se zcela změní charakter zvukového signálu. O jednotlivých posunutých složkách spektra již nelze říct, že jsou vyššími harmonickými posunuté základní složky, protože hodnoty jejich kmitočtů nejsou celými násobky základní složky. Výsledný zvuk tak má disharmonický charakter [2].

Roztažením či stlačením spektra signálu ve směru kmitočtové osy podle předpisu

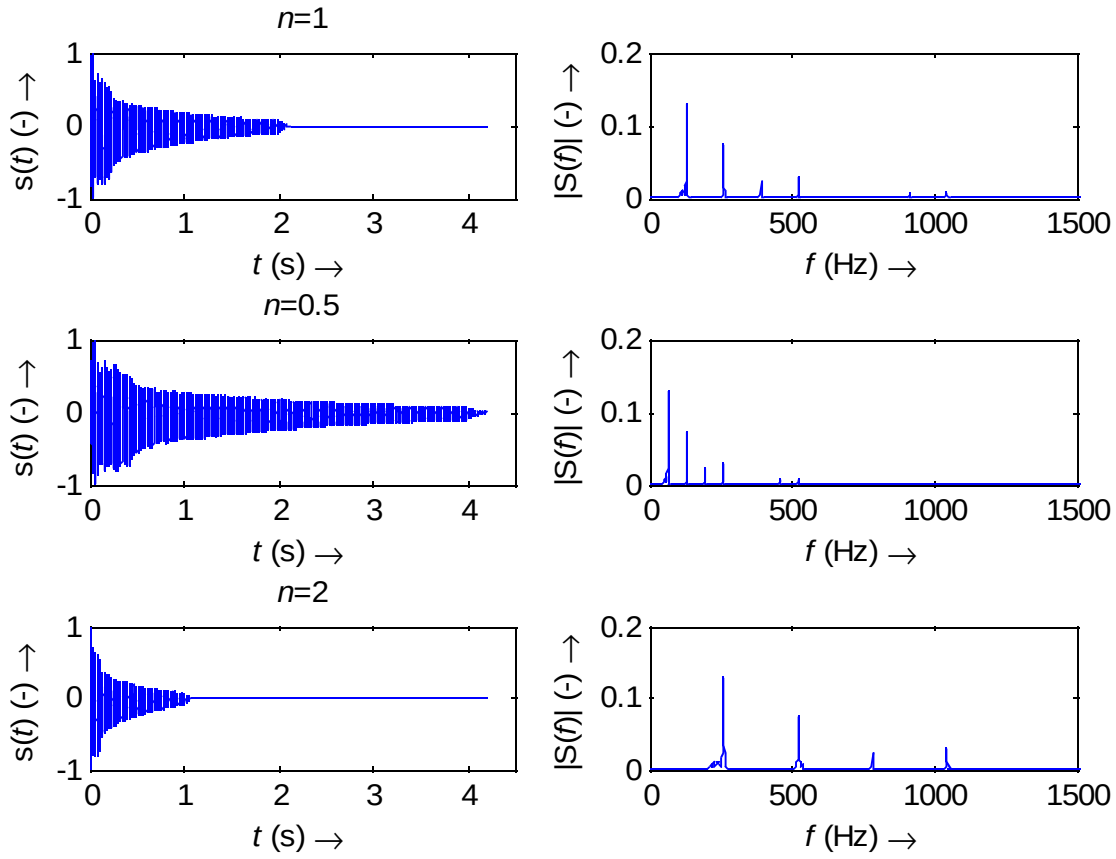
$$f_{pi} = \nu \cdot f_{oi} \quad , \tag{2.1}$$

kde f_{oi} představuje hodnotu kmitočtu i-té harmonické složky originálního signálu, f_{pi} hodnotu kmitočtu i-té harmonické složky transponovaného signálu a ν poměr změny výšky tónu (pro $\nu > 1$ se jedná o nadladění, pro $\nu < 1$ se jedná o podladění), je možné dosáhnout takové modifikace spektra signálu, kde je zachováno rozložení harmonických složek a tím i celková barva zvuku. Jednoduchým způsobem, jakým lze dosáhnout roztažení resp. stlačení spektra v případě číslicového zpracování je změna rychlosti přehrávání. Zvukový signál je tedy reprodukován s jiným vzorkovacím kmitočtem, než se kterým byl původně zaznamenán. Jako analogii tohoto procesu si lze představit klasickou magnetickou pásku se zvukovým záznamem. Různé rychlosti přehrávání záznamu mají vliv na výšku tónů obsažených v záznamu. Zpomalení přehrávání způsobí snížení tónů, zrychlení pak naopak zvýšení [16]. V případě, že by byl vzorkovací kmitočet pro přehrávání pevně daný a nebylo by možné jej měnit, šlo by dosáhnout podobného efektu pomocí převzorkovávání [12].

Na základě poměru změny výšky tónu v by byl podle vztahu

$$f_{vzy} = \frac{f_{vzx}}{v} \quad , \tag{2.2}$$

kde f_{vzx} je vzorkovací kmitočet zvuku, určen nový vzorkovací kmitočet f_{vzy}. Na tento nový kmitočet f_{vzy} by byl signál převzorkován, přehrán by byl ale s původním vzorkovacím kmitočtem f_{vzx}. Pro $v > 1$ by bylo $f_{vzy} < f_{vzx}$, došlo by tedy k decimaci signálu a při přehrání s původním vzorkovacím kmitočetem f_{vzx} by tak došlo ke zvýšení tónu. Pro $v < 1$ by došlo k interpolaci signálu a výsledkem by bylo snížení tónu. Tato situace je zobrazena na Obr. 2.1.



Obr. 2.1: Vliv proměnné rychlosti přehrávání na spektrum signálu.

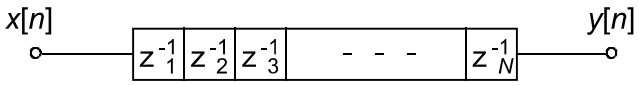
Nevýhodou tohoto zpracování je zřejmě fakt, že převzorkováním signálu při zachování konstantního vzorkovacího kmitočtu (tedy různou rychlostí přehrávání) je ovlivněna délka zvuku, což obvykle není žádoucí. Druhým problémem může být fakt, že uvedený přístup lze použít pouze tehdy, je-li zvuk k dispozici již celý dopředu, z čehož vyplývá, že uvedená metoda není vhodná pro zpracování v reálném čase, např. pro již uvedené efektování signálu přicházejícího z hudebního nástroje.

3 Metody pro posun výšky tónu se zpracováním v časové oblasti

Jak již bylo řečeno, dalším požadavkem při posunu výšky tónu je i zachování jeho časového průběhu. Za tímto účelem byly vyvíjeny různé pokročilejší metody, které se snaží různými typy přístupů o splnění tohoto požadavku s cílem minimalizovat výskyt nežádoucích artefaktů. V následujícím textu budou popsány dvě metody se zpracováním v časové oblasti.

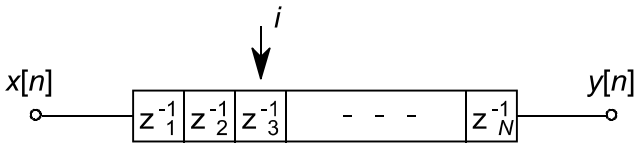
3.1 Posun výšky tónu využívající modulace zpožďovací linky

Hlavním prvkem využívaným při této metodě je zpožďovací linka délky N, viz Obr. 3.1. Výstupem této struktury je vzorek $y[n] = x[n - N]$, tedy vstupní vzorek zpožděný o N vzorkovacích period.



Obr. 3.1: Zpožďovací linka.

Pro aplikaci posunu výšky tónu použijeme modifikovanou zpožďovací linku, kde výstupní vzorek v čase n není vždy vzorek na N-té pozici zpožďovací linky, ale vzorek na i-té pozici zpoždovací linky. K určení aktuální pozice ve zpožďovací lince tady slouží ukazatel i. Situace je znázorněna na Obr. 3.2., viz také [5].



Obr. 3.2: Modulovaná zpožďovací linka.

Ukazatel i může nabývat hodnot v rozmezí <0; N>, výstup je dán $y[n] = x[n - i]$, v případě konstantní hodnoty ukazatele i je získána výše uvedená klasická zpoždovací linka o délce i. Za účelem dosažení určitých efektů lze hodnotu ukazatele modulovat průběhem různých funkcí.

Při požadavku dosažení efektu posunu výšky tónu, hodnota ukazatele i je modulována pilovou funkcí, předpis počtu vzorků jedné periody T této modulační funkce $r[n]$ je

$$r[n] = (1 - v)n, \quad \ldots\ldots\ldots\ldots \quad n = 0,1,2,\ldots,T, \tag{3.1}$$

kde v značí již zmiňovaný poměr změny výšky tónu. Pro $v > 1$ se její perioda určí jako

$$T = \frac{-N}{1 - v} \quad , \tag{3.2}$$

pro $0 < v < 1$ pak

$$T = \frac{N}{1 - v} \quad . \tag{3.3}$$

Hodnota ukazetele i pro daný časový index n pro $v > 1$ je určena podle vztahu

$$i[n] = N + r[n - O] \quad . \tag{3.4}$$

pro $0 < v < 1$ pak

$$i[n] = r[n - O] \quad . \tag{3.5}$$

kde hodnota O určuje časový posun modulační funkce $r[n]$, což ovlivní hodnotu ukazatele i v čase $n = 0$, tedy počáteční pozici ve zpožďovací lince.

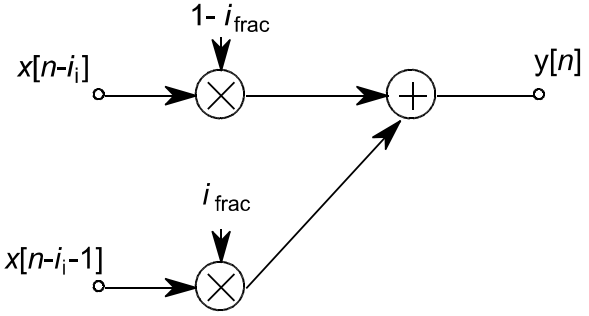
Jak si lze všimnout, hodnota ukazatele i může pro obecný poměr změny výšky tónu v nabývat reálných hodnot, ne tedy pouze celočíselných. Uvnitř zpožďovací linky jsou však k dispozici pouze vzorky $x[n]$ až $x[n-N]$ příslušející k okamžikům daným celočíselnými hodnotami n. Proto se hodnota ukazatele i rozloží na její celou část i_i a její zlomkovou část i_{frac}, takže

$$i = i_i + i_{frac} \quad . \tag{3.6}$$

3.1.1 Lineární interpolace

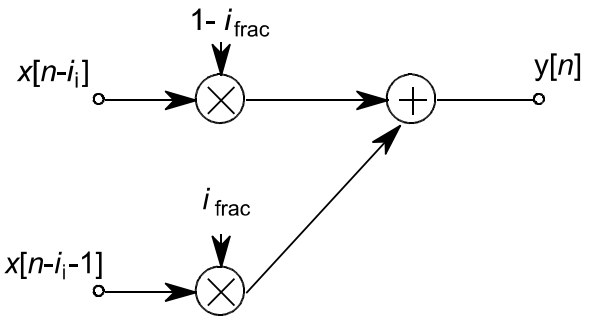
Nejjednodušší způsob, jak získat hodnotu vzorku $x[n - i]$ ze zpožďovací linky je použití lineární interpolace. Na Obr. 3.3 je zobrazeno, jak lze lineární interpolaci provést. Podle aktuální hodnoty ukazatele i jsou vybrány nejbližší uložené vzorky $x[n-i_i]$ a $x[n-i_i-1]$, tj. nejbližší zprava a zleva, v závislosti na zlomkové části i_{frac} ukazatele i jsou do výsledného vzorku váhovány podle předpisu

$$y[n] = (1 - i_{frac}) \cdot x[n - i_i] + i_{frac} \cdot x[n - i_i - 1] \quad . \tag{3.7}$$



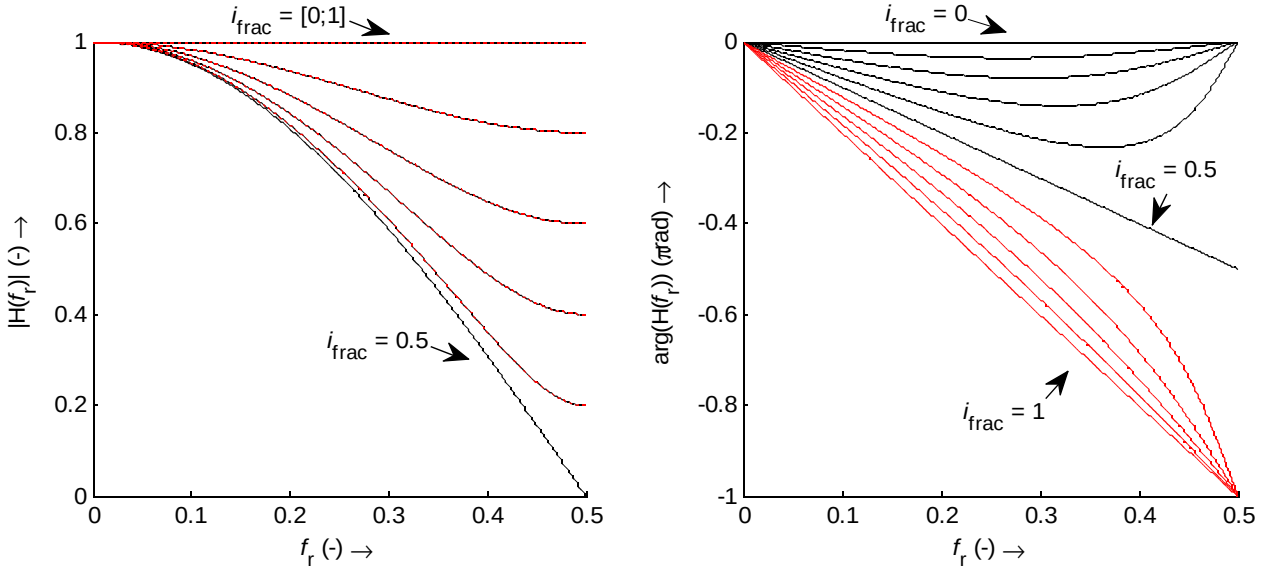
Obr. 3.3: Lineární interpolátor [5].

Pro analýzu vlivu lineární interpolace na vstupní signál si lze schéma na Obr. 3.3 aproximovat na situaci uvedenou na Obr 3.4. Nahrazení druhého vstupu zpožděným prvním vstupem si lze dovolit při předpokladu, že se hodnota i mění v čase pomalu [5].



Obr. 3.4: Zjednodušený lineární interpolátor [5].

Na Obr. 3.5 jsou vykresleny modulová kmitočtová charakteristika a fázová kmitočtová charakteristika zjednodušeného lineárního interpolátoru. Jejich jednotlivé průběhy se liší v závislosti na parametru i_{frac}, který se měnil v rozmezí od 0 do 1 s krokem 0,1. V grafu modulové kmitočtové charakteristiky platí, že průběhy pro hodnoty i_{frac} a 1-i_{frac} jsou vždy totožné, což je znázorněno překrývajícími se černými a červenými průběhy. Ze zobrazených průběhů lze nyní popsat vlastnosti lineární interpolace. Z modulové kmitočtové charakteristiky je vidět, že pro vzrůstající kmitočet dochází ke snižování přenosu. Nejvýraznější potlačení vyšších kmitočtů je při hodnotě i_{frac} = 0.5, kdy je snaha určit hodnotu vzorku nacházejícího se přesně v polovině mezi dvěma sousedními vzorky uloženými ve zpožďovací lince. Negativním rysem lineární interpolace je tedy narůstající potlačování kmitočtů směrem k polovině vzorkovacího kmitočtu [5].



Obr. 3.5: Modulová a fázová kmitočtová charakteristika zjednodušeného lineárního interpolátoru [5].

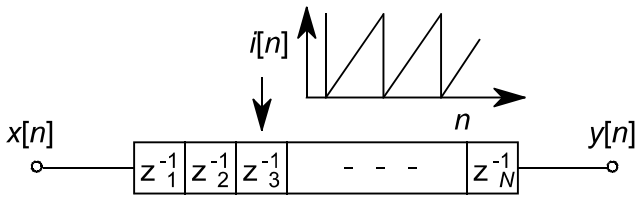
Ideální případ by byl, kdyby fázová kmitočtová charakteristika byla lineární, což znamená, že signály všech kmitočtů budou zpožděny vzhledem ke vstupnímu signálu o stejnou dobu, tedy i_{frac}. Z grafu fázové kmitočtové charakteristiky je však vidět, že toto je splněno pouze pro hodnotu i_{frac} = 0,5, při které je naopak bohužel dosaženo nejhoršího průběhu modulové kmitočtové charakteristiky. I když pro i_{frac} = 0 a 1 jsou fázové průběhy také lineární, je zbytečné bavit se o interpolaci, jelikož hodnoty vzorků jsou přímo k dispozici. Pro ostatní průběhy, kde již fázové kmitočtové charakteristiky nejsou lineární, nelze pro vyšší kmitočty docílit přesně definované hodnoty zpoždění i_{frac}. Výpočet interpolované hodnoty bude tedy kromě zkreslení amplitudy obsahovat i určité zkreslení fáze, tedy vypočítaný vzorek bude platný pro hodnotu mírně se odchylující od požadované hodnoty i_{frac}. Je však dobré si všimnout, že všechny fázové průběhy jsou pro nízké hodnoty kmitočtů téměř lineární. Fázové zkreslení se tedy stejně jako amplitudové zkreslení výrazně projeví až na vyšších kmitočtech.

Existují i další typy interpolací mající lepší vlastnosti oproti interpolaci lineární. Velkou výhodou lineární interpolace však zůstává především její jednoduchost.

3.1.2 Realizace posunu výšky tónu s využitím modulované zpožďovací linky

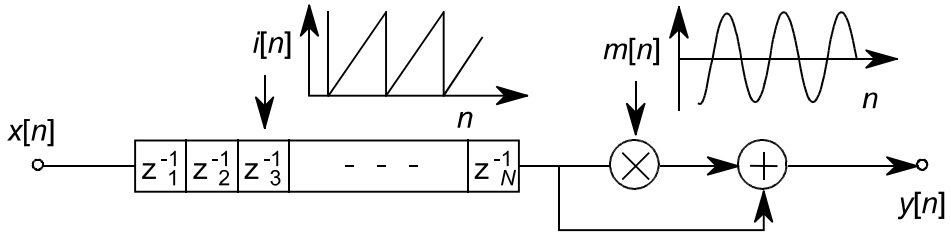
Jak bylo popsáno v prvním odstavci kapitoly 3.1, bude-li hodnota ukazatele *i* v čase konstantní, vstupní signál *x*[*n*] se bude pouze zpožďovat o pevně danou hodnotu *i*. Bude-li se však v čase lineárně měnit, je dosaženo vlastně podobného efektu, jako je proměnná rychlost přehrávání popsaná v kapitole 2.1, což má za následek požadované roztažení či stlačení spektra. Ukazatel *i* však může nabývat pouze omezených hodnot v rozmezí <0;*N*>, proto je modulován pilovou funkcí, která je po částech lineární podle předpisu (3.1). Hodnota *i* se pak v čase lineárně zmenšuje (posun výšky nahoru) či lineárně zvyšuje (posun výšky dolů). Po dosažení krajní hodnoty intervalu <0;*N*> se skokově změní na opačnou krajní hodnotu z intervalu <0;*N*>.

Základním blokem sktruktury je pak zpožďovací linka modulovaná pilovým signálem, což je zobrazeno na Obr. 3.6.



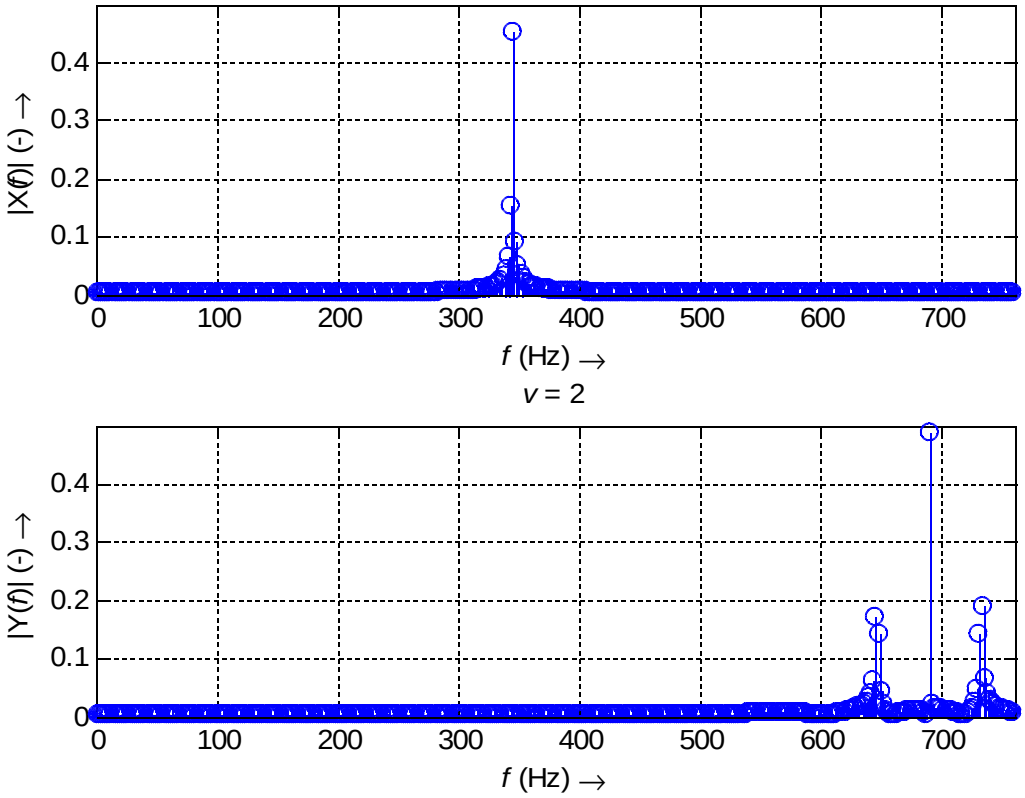
Obr. 3.6: Způsob modulace zpožďovací linky pro dosažení změny výšky tónu.

Výsledek tohoto zpracování není příliš uspokojivý, jelikož skoková změna ukazatele *i* ze začátku na konec zpožďovací linky (posun výšky nahoru) či z konce na začátek (posun výšky dolů) způsobuje ve výstupním signálu značné nespojitosti, jež v původním vstupním signálu nebyly a jež do výstupního signálu vnáší nepříjemné zkreslení. Tento nedostatek lze omezit váhováním výstupního signálu ze zpožďovací linky, situace je znázorněnana na Obr. 3.7. Výstupní signál je zde modulován harmonickým průběhem tak, aby v okamžiku nespojitosti byl výstup ze zpožďovací linky potlačen. Počet vzorků periody modulačního harmonického signálu $m[n]$ je stejný jako počet vzorků T periody modulačního signálu $i[n]$, která je určena podle vzahu (3.2) resp (3.3). Dojde tak ke znatelnému zlepšení, avšak v takto zpracovaném zvuku je výrazně slyšitelná amplitudová modulace. Tato skutečnost je zobrazena na Obr. 3.8, kde vstupní harmonický signál $x[n]$ o kmitočtu 345 Hz prošel zmíněným systémem z Obr. 3.7, pro který byl poměr změny výšky tónu $v = 2$ a délka zpožďovací linky $N = 1024$. Při porovnání spektra $X(f)$ vstupního signálu a spektra $Y(f)$ výstupního signálu je vidět, že se kmitočet harmonického signálu změnil na požadovanou hodnotu 690 Hz, navíc je spektrum výstupního signálu obohaceno o postranní složky způsobené amplitudovou modulací. Podle vztahu (3.2) určíme počet vzorků periody modulačního signálu $m[n]$: $T = -1024/(1-2) = 1024$. Kmitočet modulačního signálu při vzorkovacím kmitočtu 44100 Hz je pak $f_m = 44100/1024 = 43$ Hz. To odpovídá Obr. 3.8, kde je vidět, že postranní složky jsou od nosného signálu vzdáleny přibližně o hodnotu $f_m = 43$ Hz, což platí právě pro amplitudovou modulaci.

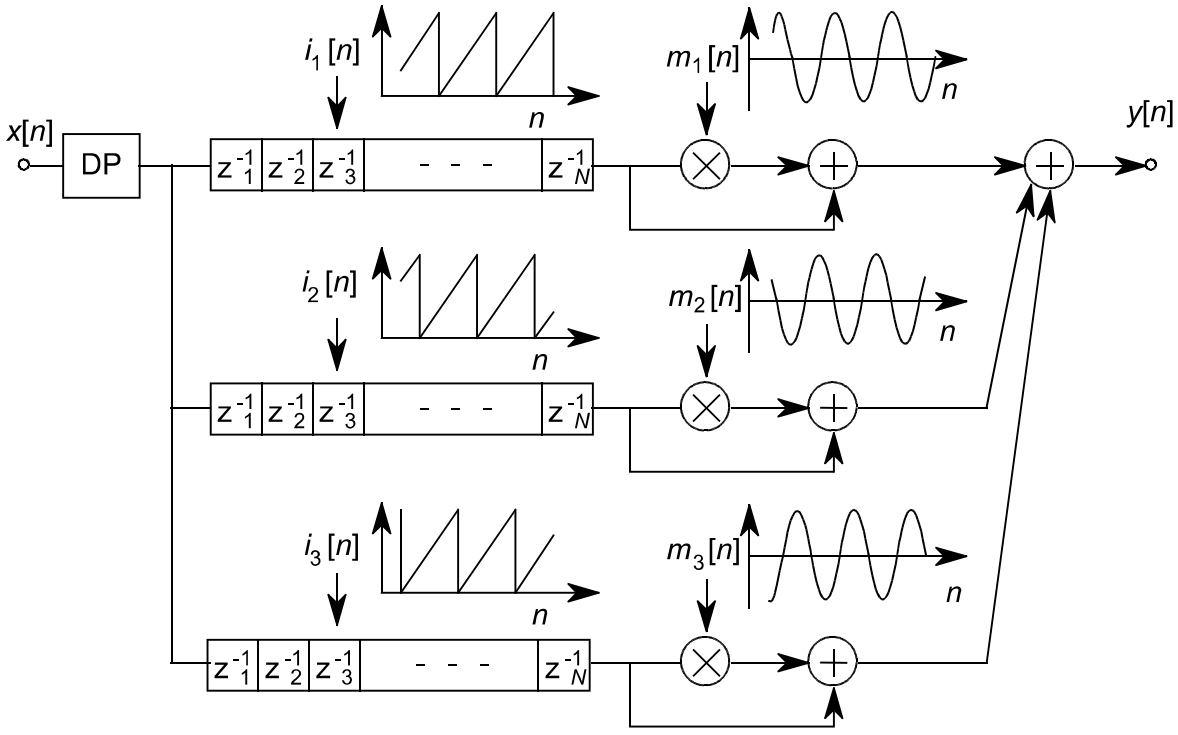


Obr. 3.7: Omezení nespojitostí pomocí váhování harmonickým signálem.

Vylepšení metody využívající modulace zpožďovací linky je zobrazeno na Obr 3.9, viz [6]. Vstupní signál $x[n]$ je prvně filtrován antialiasingovou dolní propustí, poté je rozdělen do tří větví, z nichž každá obsahuje jednu zpožďovací linku. Výsledný signál je pak tvořen součtem signálů z jednotlivých větví. Výstup ze zpožďovacích linek je opět modulován harmonickými signály. Zde je nutné si všimnout, že zpožďovací linky jsou modulovány pilovými signály fázově vzájemně posunutými o $2\pi/3$. Výstupní signál je tedy tvořen zpracovanými vstupními signály překrývajícími se se vzájemným posunutím o třetinu délky zpožďovací linky. Podle nastavení modulačních signálů zpožďovacích linek jsou nastaveny i váhovací signály výstupů, které jsou také vzájemně fázově posunuty o $2\pi/3$. Důležité je, že sčítání stejných harmonických signálů vzájemně posunutých o $2\pi/3$ dává jako výsledek konstantní hodnotu. Proto ubývající příspěvek do výstupního signálu z jedné větve způsobený modulačním signálem je kompenzován narůstajícím příspěvkem z další větvě. Tímto postupem je do značné míry eleminován parazitní efekt způsobený amplitudovou modulací.



Obr. 3.8: Spektrum před a po změně výšky tónu s využitím modulace zpožďovací linky s váhováním výstupu harmonickým signálem.



Obr. 3.9: Vylepšená metoda posunu výšky tónu modulací zpožďovací linky [6].

3.2 Posun výšky tónu metodou PICOLA

Tato metoda je prvotně určena pro změnu délky signálu při zachování jeho kmitočtových vlastností, tzv. time stretching. Postup pro posun výšky tónu je pak takový, že originální signál je s příslušným poměrem prodloužen, viz [9],[14]. Původní délky je pak dosaženo pomocí převzorkování s daným poměrem, což požadovaným způsobem ovlivní kmitočtové vlastnosti zvukového signálu.

Metoda PICOLA patří mezi metody, u kterých je pro docílení požadovaného efektu třeba odhadnout základní periodu p zpracovávaného signálu. Tato informace je pak využita při procesu prodlužování či zkracování signálu.

Následuje popis postupu metody, kdy dochází k posunu kmitočtu základního tónu směrem k vyšším hodnotám. Zároveň ji ilustruje Obr. 3.10.

- Pro vstupní signál $x[n]$ je určen odhad počtu vzorků p základní periody.
- Na základě hodnoty p a požadovaného poměru změny výšky v určíme délku bloku l podle vztahu

$$l = \frac{p}{v-1} \quad , \tag{3.8}$$

  při úpravě vztahu na

$$v = \frac{p+l}{l} \tag{3.9}$$

  lze vidět, že pro dosažení požadovaného poměru roztažení signálu je k původnímu signálu délky l nutné přidat p vzorků.
- Ze vstupního signálu $x[n]$ jsou vybrány dva po sobě jdoucí bloky o délce p, z nichž první končí s předchozím blokem o délce l a druhý začíná zároveň s následujícím blokem o délce l. Tyto dva bloky jsou následně vynásobeny váhovacími okny a vzájemně překryty a sečteny. Vznikne tak nový blok o délce p.
- Tento blok je do původního signálu vložen před následující blok o délce l, čímž dojde k prodloužení signálu o p vzorků. Tímto způsobem je tedy mezi každých l vzorků vloženo p vzorků, vznikne tak prodloužený signál $y[n]$.
- Prodloužený signál $y[n]$ je převzorkován s poměrem $1/v$, čímž je dosaženo jeho původní délky.

Podobným způsobem lze dosáhnout snížení základního tónu, následující postup znázorňuje Obr. 3.11.

- Pro vstupní signál $x[n]$ je určen odhad počtu vzorků p základní periody
- Na základě hodnoty p a požadovaného poměru změny výšky v je určena délka bloku l podle vztahu

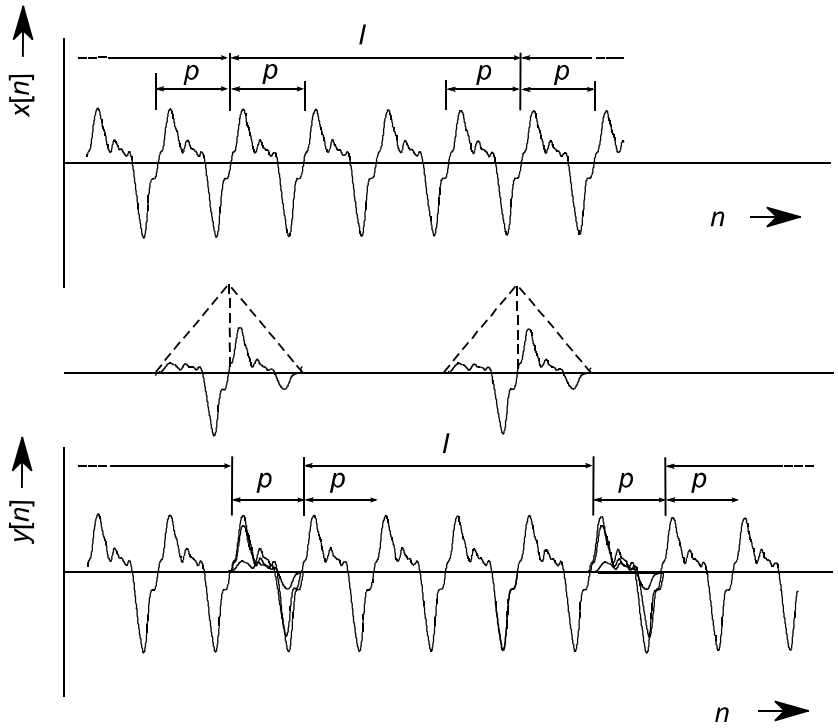
$$l = \frac{p}{1-v} \quad , \tag{3.10}$$

  po úpravě na

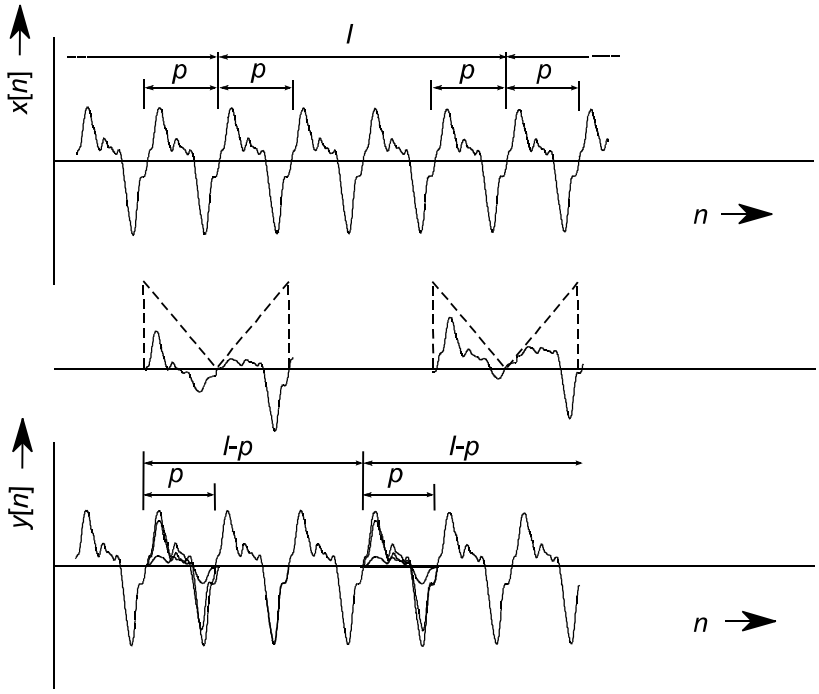
$$v = \frac{l-p}{l} \tag{3.11}$$

  je zřejmé, že pro zkrácení původního signálu $x[n]$ se zkrátí blok délky l o p vzorků.
- Ze vstupního signálu $x[n]$ jsou vybrány dva po sobě jdoucí bloky o délce p, z nichž první končí s předchozím blokem o délce l a druhý začíná zároveň s následujícím blokem o délce l. Tyto dva bloky jsou následně vynásobeny váhovacími okny a vzájemně překryty a sečteny. Vznikne tak nový blok o délce p.

- Tímto novým blokem je nahrazen první z váhovaných bloků, druhý váhovaný blok již není ve výsledném signálu zastoupen, tudíž je za první blok doplněno $l - p$ zbývajících vzorků. Tímto způsobem je každý blok o délce l zkrácen o p vzorků, vznikne tak signál $y[n]$.
- Zkrácený signál je převzokrován s poměrem $1/v$, čímž je dosaženo jeho původní délky.



Obr. 3.10: Prodloužení signálu metodou PICOLA.



Obr. 3.11: Zkrácení signálu metodou PICOLA.

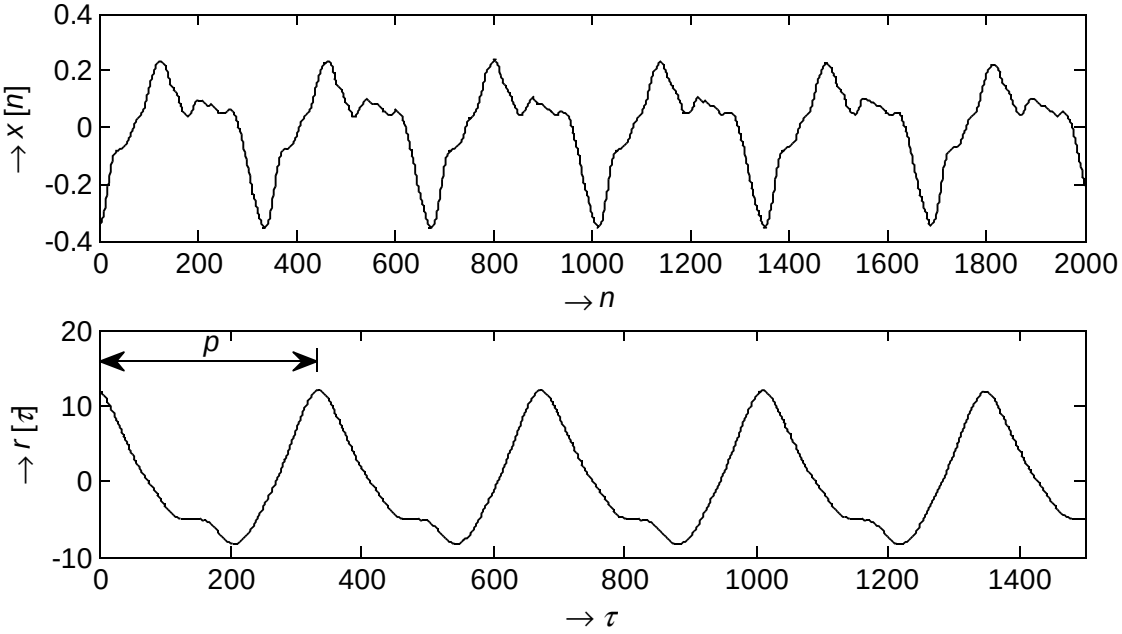
Výše uvedený postup tedy každých l vzorků vstupního signálu $x[n]$ prodlouží, resp. zkrátí vždy o p vzorků. Problémem zůstává určení odhadu základní periody p.

3.2.1 Odhad základní periody signálu

Nejznámější metodou pro detekci základní periody je autokorelační metoda [8], [1]. Autokorelační funkce je dána vztahem

$$r_t[\tau] = \sum_{j=t+1}^{t+W} x[j]\,x[j+\tau] \quad . \tag{3.12}$$

Udává míru podobnosti signálu $x[n]$ se sebou samým pro různé hodnoty posunu τ (tzv. lag). Parametr W určuje, pro jak dlouhý úsek signálu $x[n]$ bude autokrelační funkce počítána. Je-li signál periodický, pak lze v průběhu autokorelační funkce hledat maxima, pro která byla míra shody vzájemně posunutého signálu $x[n]$ největší, jejich hodnoty posunu τ pak odpovídají násobkům periody p. Obr 3.12 ukazuje průběh signálu a jeho autokorelační funkci určenou podle vztahu (3.12). Základní periodu p lze z grafu určit jako pozici τ prvního maxima autokorelační funkce, což je v kontrétním případě 334 vzorků.



Obr. 3.12: Autokorelační funkce.

Jako další metodu odstraňující některé nedostatky autokorelační metody je možné uvést metodu yin, podrobně viz [8]. Tato metoda sestává z několika kroků, s každým provedeným krokem je zvýšena přesnost metody.

Prvním krokem je výpočet rozdílové funkce podle vztahu

$$d_t[\tau] = \sum_{j=t+1}^{t+W} (x[j] - x[j+\tau])^2 \quad . \tag{3.13}$$

Od autokorelační funkce se liší tím, že mezi odpovídajícími si hodnotami se neprovádí součin, ale rozdíl, který je umocněn. Dojde-li k posunu posloupnosti o τ odpovídající periodě, odečítané hodnoty budou v případě přesně periodického signálu stejné a výsledná funkční hodnota bude nulová. Diferenční funkce tedy na rozdíl od autokorelační funkce pro hodnoty posunu τ odpovídajícím násobkům period nabývá minimálních hodnot, což lze vidět na Obr. 3.13, kde minima diferenční funkce odpovídají maximům autokorelační funkce na Obr. 3.12.

Pro eliminaci jevu, že jako minimum bude zvolen posun τ = 0, je vypočítáná diferenční funkce normalizovaná kumulativním průměrem podle předpisu

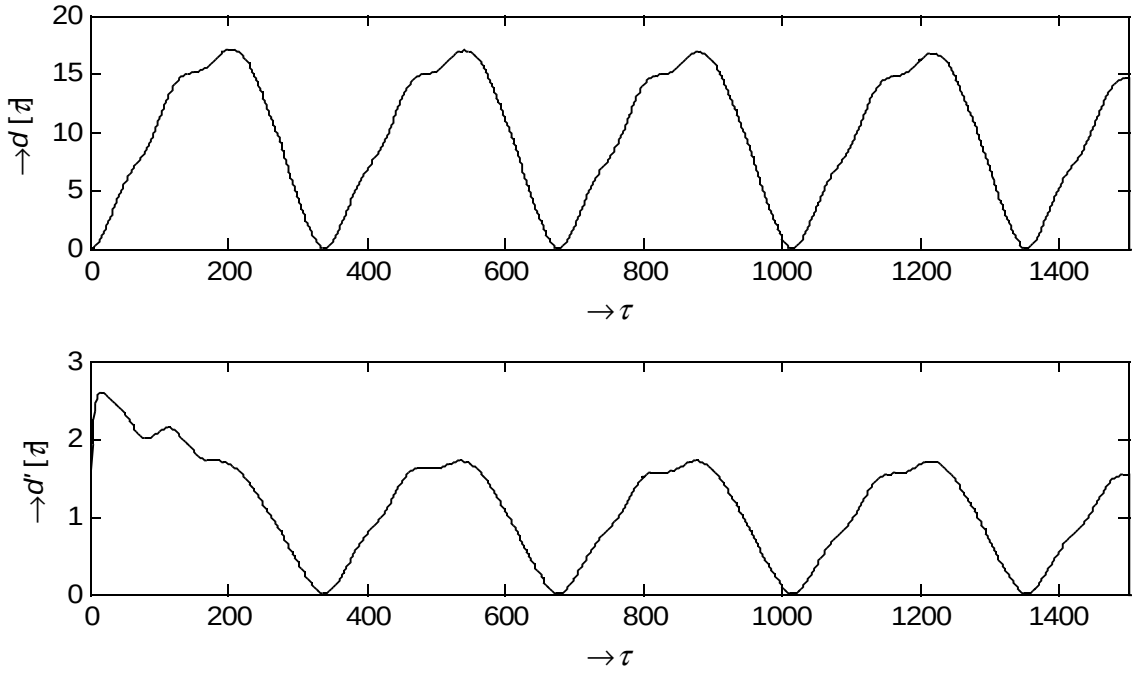
$$d'_t[\tau] = \begin{cases} 1, & \tau = 0 \\ d_t[\tau] / (1/\tau) \sum_{j=1}^{\tau} d_t[j], & \tau \neq 0 \end{cases}, \tag{3.14}$$

pro počáteční hodnoty τ tak funkce nebude nabývat minimálních hodnot, viz spodní graf na Obr. 3.13.

Dalším krokem je stanovení absolutního prahu. Posun τ prvního minima funkce $d'_t[\tau]$, které je menší než stanovený práh, je pak označen jako základní perioda. Práh se určuje empiricky.

Pokud základní perioda není přesným násobkem vzorkovací periody, lze použít kvadratickou interpolaci pro zpřesnění výsledku. V takovém případě se nalezené minimum a jeho dvě okolní hodnoty proloží parabolou a z minimální hodnoty paraboly se určí přesnější posun τ.

V rámci průbězného odhadu periody v čase může být použita mediánová filtrace pro odstranění skokových výkyvů v odhadech periody.



Obr. 3.13: Diferenční funkce a diferenční funkce normalizovaná kumulativním průměrem.

Kromě metody s autokorelační funkcí a metody yin existují i další metody pro odhad základní periody, např. spektrální metoda, kepstrální metoda [1] a další.

4 Metoda se zpracováním časových segmentů v kmitočtové oblasti

Další možnou metodou pro posun výšky tónu je metoda využívající tzv. fázového vokodéru. Tato metoda se vyznačuje zcela jiným principem zpracování, jedná se o tzv. časově-kmitočtové zpracování.

Hlavní myšlenka spočívá v transformaci určitého úseku časového průběhu vstupního signálu do průběhu vyjadřujícího jeho kmitočtové složení. V případě hudebního signálu toto vyjádření tedy přímo říká, které kmitočtové složky jsou v signálu obsaženy. Pro dosažení změny výšky tónu jsou v kmitočtovém vyjádření jednotlivé složky patřičným způsobem modifikovány a posléze je toto modifikované spetrum transformováno zpět do časového vyjádření, vznikne tak požadovaný výstupní signál.

4.1 Krátkodobá spektrální analýza

Pro vyjádření spektrálního složení určitého segmentu signálu se využívá tzv. krátkodobé spektrální analýzy, jež se od klasické spektrální analýzy liší především v tom, že není analyzován celý signál najednou, ale vždy po časových úsecích stanovené délky, tzv. segmentech. Výsledkem analýzy je pak informace nejen o samotném spektrálním složení signálu, ale i změnách tohoto spektrálního složení v čase. Představitelem krátkodobé spektrální analýzy pro diskrétní signál je tzv. diskrétní krátkodobá Fourierova transformace diskrétního signálu daná vztahem [16]

$$X(n,k)=\sum_{m=-\infty}^{\infty} x[m]h[n-m]\mathrm{e}^{-jk\frac{2\pi}{N}m}=|X(n,k)|\mathrm{e}^{j\varphi(n,k)} \quad , \tag{4.1}$$

kde $X(n,k)$ je časově proměnným spektrem signálu $x[m]$, které pro daný časový okamžik n udává amplitudy $|X(n,k)|$ a fáze $\varphi(n,k)$ jednotlivých složek k v rozmezí celých čísel <0, N-1>, je tedy vidět, že se jedná o transformaci definovanou v oboru komplexních čísel. Výraz $h[n-m]$ slouží jako posuvné okno délky N, které z teoreticky nekonečně dlouhého signálu $x[m]$ vybere segment délky N, který je podroben zpracování.

Pro představu diskrétní krátkodobé Fourierovy analýzy diskrétního signálu lze použít popisu využívajícího banky filtrů [16]. Jak je vidět ze vztahu (4.1), krátkodobé spektrum $X(n,k)$ je vlastně dáno konvolucí signálů. Lze si tedy představit, že vstupní signál $x[m]$ je filtrován k filtry, jejichž impulzové odezvy jsou

$$h_k[n]=h[n]\mathrm{e}^{jk\frac{2\pi}{N}n} \quad , \tag{4.2}$$

a kmitočtové charakteristiky jsou

$$H_k(\mathrm{e}^{j(\Omega)})=H(\mathrm{e}^{j(\Omega-\Omega_k)}),\Omega_k=k\frac{2\pi}{N} \quad . \tag{4.3}$$

Je tak získána banka N filtrů, viz Obr 4.1, jejichž kmitočtová charakteristika vznikne modulací impulzní charakteristiky $h[n]$ funkcí $\mathrm{e}^{jk\frac{2\pi}{N}n}$, což má za následek posun propustného pásma. Filtry s modulovanou impulzní chrakteristikou $h_k[n]$ představují pásmové propusti s jednostrannou kmitočtovou charakteristikou, výstupem k-tého filtru je komplexní hodnota y_k reprezentující spektrální složku o indexu k

$$y_k[n]=\tilde{X}(n,k)=|X(n,k)|\mathrm{e}^{j\tilde{\varphi}(n,k)} \quad , \tag{4.x}$$

filtrační proces lze popsat jako

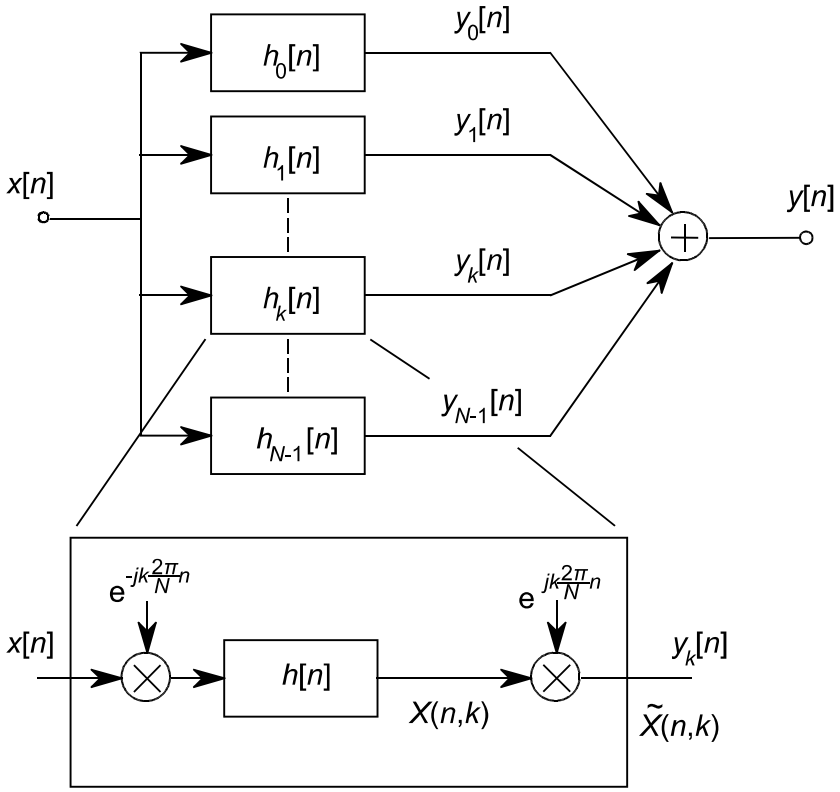
$$y_k[n]=\sum_{m=-\infty}^{\infty} x[m]h_k[n-m]=\sum_{m=-\infty}^{\infty} x[m]h[n-m]\mathrm{e}^{jk\frac{2\pi}{N}(n-m)}$$
$$=\mathrm{e}^{jk\frac{2\pi}{N}n}\sum_{m=-\infty}^{\infty} x[m]h[n-m]\mathrm{e}^{-jk\frac{2\pi}{N}m}=\mathrm{e}^{jk\frac{2\pi}{N}n}X(n,k). \quad (4.x)$$

Z předchozích vztahů plyne, že

$$\tilde{X}(n,k)=\mathrm{e}^{jk\frac{2\pi}{N}n}X(n,k)=\mathrm{e}^{jk\frac{2\pi}{N}n}|X(n,k)|\mathrm{e}^{j\tilde{\varphi}(n,k)} \quad (4.4)$$

a

$$\tilde{\varphi}(n,k)=k\frac{2\pi}{N}n+\varphi(n,k) \quad . \quad (4.5)$$



Obr. 4.1: Znázornění banky filtrů pro krátkodobou spetrální analýzu [16].

Obr. 4.1 zobrazuje výše popsaný proces. Vstupní signál $x[n]$ vstupuje do banky filtrů. Výstupní signál $y_k[n]$ k-tého filtru je roven komplexnímu pásmově omezenému signálu $\tilde{X}(n,k)$ se středním kmitočtem Ω_k.

V dolní části Obr. 4.1 je zobrazena vnitřní struktura k-tého filtru. Výstupem z bloku, kde dochází ke konvoluci signálu $x[n]$ násobeného funkcí $\mathrm{e}^{-jk\frac{2\pi}{N}n}$ a impulzové odezvy $h[n]$, je komplexní signál $X(n,k)$, který je nízkofrekvenčním ekvivalentem signálu $\tilde{X}(n,k)$, který je na výstupu získán vynásobením funkcí $\mathrm{e}^{jk\frac{2\pi}{N}n}$.

Výstupní signál $y[n]$ je dán součtem příspěvků ze všech větví, tedy

$$y[n]=\sum_{m=0}^{N-1} y_k[n]=\sum_{m=0}^{N-1} \tilde{X}(n,k)=\sum_{m=0}^{N-1} X(n,k)\mathrm{e}^{jk\frac{2\pi}{N}n} \quad . \tag{4.6}$$

V případě, že vstupní signál $x[n]$ je reálný, bude platit

$$y_k[n]=\tilde{X}(n,k)=\tilde{X}^*(n,N-k)=y^*_{N-k}[n] \quad , \tag{4.7}$$

takže odpovídající si dvojice výstupů podle předpisu jsou komplexně sdružené.

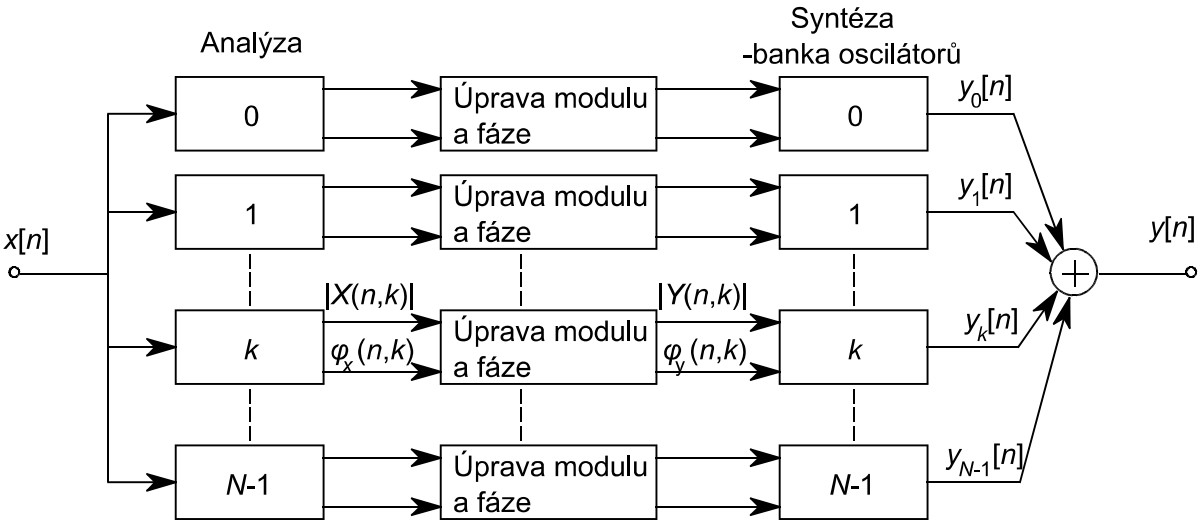
Také si lze všimnout, že výstupy filtrů banky $\tilde{X}(n,k)$ jsou vlastně vypočítanou diskrétní Fourierovou transformací, jež je dána vztahem

$$X(k)=\sum_{n=0}^{N-1} x[n]\mathrm{e}^{-jk\frac{2\pi}{N}n} \tag{4.8}$$

v případě, že posloupnost $h[n]$ představuje pravoúhlé okno [12].

4.2 Použití fázového vokodéru

Jak bylo uvedeno v úvodu kapitoly, převod signálu z časové do kmitočtové oblasti má tu výhodu, že existuje přístup k jednotlivým kmitočtovým složkám. Jejich manipulací tak lze dosáhnout požadovaného efektu. Převodem spektra do časového vyjádření je tak získán požadovaný signál. Na Obr. 4.2 je tato situace zobrazena. Analytická část převádí vstupní signál do kmitočtové oblasti, ve které je signál zpracován a pomocí syntetické části převeden zpět do časového průběhu.



Obr. 4.2: Realizace fázového vokodéru pomocí banky filtrů [16].

Na Obr. 4.3 je dále podrobněji rozkreslena struktura jedné větve banky filtrů z Obr.4.2. Analyzující část odpovídá části již uvedené na Obr. 4.1 s tím rozdílem, že na jejím výstupu není pásmově omezený signál $\tilde{X}(n,k)$, ale jeho nízkofrekvenční ekvivalent [16]

$$X(n,k)=(x[n]\mathrm{e}^{-jk\frac{2\pi}{N}n})*h[n]=(x[n]\mathrm{e}^{-j\Omega_k n})*h[n]=|X(n,k)|\mathrm{e}^{j\varphi(n,k)}, \tag{4.9}$$

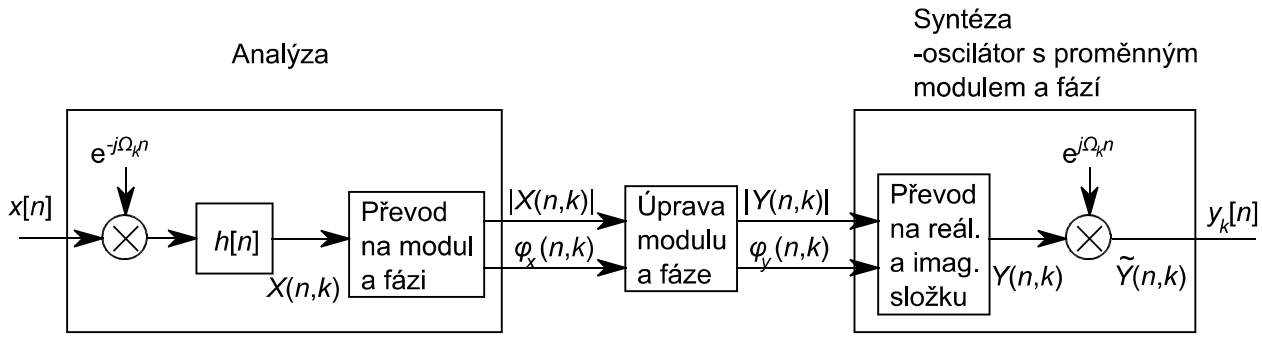
konkrétně jeho modul a fáze. Reprezentace nízkofrekvenčích ekvivalentnů $X(n,k)$ pásmových signálů $\tilde{X}(n,k)$ má tu výhodu, že bude-li mít vstupní signál $x[n]$ přesně úhlový kmitočet Ω_k, fáze $\varphi(n,k)$ bude v čase konstantní. Naopak její změna v čase značí odchylku od středního úhlového kmitočtu Ω_k. k-tého pásma. Okamžitý úhlový kmitočet Ω_i. se pak určí jako derivace fáze $\varphi(n,k)$ podle vztahu

$$\Omega_i(n,k)=\Omega_k(n,k)+\frac{d}{dn}\varphi(n,k)=\Omega_k(n,k)+\varphi(n,k)-\varphi(n-1,k). \tag{4.10}$$

Z tohoto vztahu pak lze odvodit výpočet okamžitého absolutního kmitočtu f_i jako

$$f_i(n,k)=\left(\frac{k}{N}+\frac{\varphi(n,k)-\varphi(n-1,k)}{2\pi}\right) f_{vz} \quad , \tag{4.11}$$

kde f_{vz} je vzorkovací kmitočet, který v předchozím vztahu nevystupoval, jelikož úhlové kmitočty Ω_k *a* Ω_i jsou relativními úhlovými kmitočty.



Obr. 4.3: Znázornění jedné větve fázového vokodéru [16].

Jakmile jsou k dispozici okamžité hodnoty kmitočtů, lze podle potřeby měnit amplitudy a kmitočty banky oscilátorů. Výstup $y[n]$ z banky filtrů je pak dán součtem signálů z jednotlivých oscilátorů

$$y[n]=\sum_{k=0}^{N-1}\tilde{Y}(n,k)=\sum_{k=0}^{N-1}|Y(n,k)|\,e^{j\varphi_y(n,k)}e^{j\Omega_k n}=\sum_{k=0}^{N-1}Y(n,k)e^{j\Omega_k n} \quad . \tag{4.12}$$

4.3 Posun výšky s využitím fázového vokodéru

Jak bylo popsáno v předchozím odstavci, změna fáze $\varphi(n,k)$ mezi dvěma časovými okamžiky určuje okamžitý kmitočet. Má-li být tedy měněna výška zpracovávaného zvuku, bude se modifikovat velikost změny fáze vynásobením faktorem změny výšky tónu [16].

Zpracování pak bude sestávat z těchto fází:

- Určí se změna fáze vzhledem k předchozímu okamžiku

$$\Delta\varphi(n,k)=\varphi(n,k)-\varphi(n-1,k) \quad . \tag{4.13}$$

- Tato změna se vynásobí faktorem změny výšky *v* a přičte se k aktuální hodnotě fáze výstupního oscilátoru

$$\tilde{\psi}(n+1,k)=\tilde{\psi}(n,k)+v\cdot\Delta\varphi(n,k) \quad . \tag{4.14}$$

- Sečtou se výstupní hodnoty z jednotlivých oscilátorů.

Tímto postupem může být dosaženo změny výšky tónu.

5 Shrnutí a srovnání metod pro posun základního tónu zvuku

Metody pro posun výšky základního tónu je třeba srovnávat především ze dvou základních hledisek. Prvním hlediskem je kvalita dané metody, tedy do jaké míry se výsledek liší od očekávaného výsledku. Druhým hlediskem je výpočetní náročnost dané metody.

5.1 Metoda s využitím modulace zpožďovací linky

Metoda s využitím modulace zpožďovací linky je relativně jednoduchá metoda využívající principu změny rychlosti přehrávání. Problematické jsou však nespojitosti vznikající při skokové změně ukazatele vyčítajícího vzorky ze zpožďovací linky v případě, kdy její délka neodpovídá násobkům délky základní periody zpracovávaného signálu. Omezování nespojitostí na výstupu zpožďovací linky pomocí váhování se ve výsledném signálu výrazně projevuje jako amplitudová modulace (tremolo efekt), při vyšších poměrech změny základního tónu jsou složky ve spektru vzniklé váhováním vnímány jako samostatné kmitočty mající disharmonický charakter vzhledem ke kmitočtovým složkách zpracovávaného signálu.

Výpočetní náročnost tohoto algoritmu je vzhledem k dalším dvěma uváděným metodám nízká, realizace obnáší pouze vyčítání vzorků ze zpožďovací linky, interpolaci mezi nimi a vhodné váhování výstupního signálu.

Simulace schématu z Obr. 3.9 v Matlabu, kde je využito sčítání signálů ze tří větví pro eliminaci modulačních efektů, také přináší určité artefakty. Ukazatele vyčítající ze zpožďovací linky jsou vzájemně posunuty o jednu třetinu její délky, při sčítání signálů z jednotlivých větví pak dochází ke sčítání různě zpožděného vstupního signálu. Pokud vzájemné zpoždění není přibližně rovno délce základní periody, může docházet k modifikaci spektrálního složení vlivem sčítání vzájemně časově posunutých signálů (lze připodobnit k funkci hřebenového filtru). Výsledek simulací, kdy byl metodě předán zvuk s požadovaným poměrem změny tónu, nebyl příliš kvalitní především v případech, kdy základní perioda signálu neodpovídala vzdálenosti ukazatelů v jednotlivých větvích. Často byly ve zpracovaných zvucích slyšet disharmonické tóny a výsledný slyšitelný poměr změny tónu neodpovídal poměru požadovanému. V simulacích byly prováděny pokusy o adaptivní přizpůsobování délky zpožďovací linky vzhledem k základní periodě signálu, což vedlo k mnohem lepším výsledkům, ale za cenu výrazného zkomplikování algoritmu a především nutnosti detekce základní periody, což v literárních pramenech u této metody není uváděno jako nutné. Po takovém rozšíření však byla metoda velice degradována z hlediska výpočetní náročnosti, výsledná kvalita upraveného zvuku navíc nedosáhla kvality následující metody.

5.2 Metoda PICOLA

Metoda PICOLA patří mezi metody, pro jejichž funkci je nutné správně odhadnout základní periodu zpracovávaného signálu. Za podmínky správného odhadu základní periody tato metoda vykazuje lepší výsledky v porovnání s předchozí metodou.

Při uvažování postupu, kdy je vstupní signál nejprve roztažen či zkrácen a poté převzorkován si lze uvědomit, že výpočetní náročnost tohoto algoritmu se bude zvyšovat, čím více se poměr změny výšky v bude lišit od hodnoty 1, jelikož pro větší změny klesá hodnota délky bloku l. Z toho vyplývá, že výpočet bloku o délce p vznikající ze dvou sousedních bloků o délce p jejich váhováním bude probíhat častěji, což výpočetní náročnost zvyšuje. Vhodnou implementací algoritmu však lze eliminovat popsanou závislost výpočetní náročnosti.

Výpočetní náročnost metody PICOLA zvyšuje ve srovnání s předchozí metodou nutnost odhadu základní periody. Je tedy nutné hledat kompromis mezi požadavkem nízké výpočetní náročnosti odhadu základní periody a požadavkem jejího dostatečně přesného určení. Jako příklad snížení výpočetní náročnosti u autokorelační metody lze uvést zvýšení kroku τ (lagu) při výpočtu autokorelační funkce [14].

Kromě metody PICOLA existují i jiné metody pro prodloužení a zkrácení signálu se zpracováním v časové oblasti, které jde na popsaném principu využít i pro změnu výšky základního tónu, např. metody SOLA a PSOLA, viz [16]. Obecně se však dá říct, že všechny tyto metody jsou založeny na principu správného odhadu základní periody, liší se jen ve způsobu zpracování. Fáze odhadu základní periody a fáze zpracování lze tak od sebe oddělit. Jednotlivé metody odhadu základní periody a metody zpracování je pak možné vzájemně podle konkrétních požadavků kombinovat.

5.3 Metoda s využitím fázového vokodéru

U metody využívající fázového vokodéru se lze setkat s artefakty připomínající určité rozmazávání zvuku, které je způsobeno principem neurčitosti, který je platný pro krátkodobou Fourierovu analýzu. Pro velké rozlišení v kmitočtu získáváme malé rozlišení v čase. Analýzou delšího segmentu signálu získáme více kmitočtových složek a tedy přesnější spektrum, avšak toto spektrum je platné pro dlouhý analyzovaný segment, chybí tak informace, jaké jsou časové poměry mezi jednotlivými spektrálními složkami v rámci analyzovaného segmentu [2].

Z hlediska výpočetní náročnosti patří tato metoda mezi nákladnější. Pro minimalizaci výpočetní náročnosti je vhodné v analyzující části banky filtrů rozdělit vstupní signál do subpásem, jejichž počet odpovídá hodnotě mocniny 2. V takovém případě pak lze pro analýzu použít rychlou Fourierovu transformaci (FFT), tedy algoritmus pro efektivní výpočet diskétní Fourierovy transformace (DFT). Pro další snížení výpočetní náročnosti není nutné počítat FFT ze segmentů s časovým krokem 1 vzorku, tak jak to vyplývá z výše uvedeného popisu, viz Obr. 4.3, ale s krokem představujícím kolem 25 % délky analyzovaného segmentu (délku analyzovaného segmentu představuje délka impulsní charakteristiky $h[n]$ filtrů v bance) . Uspokojivých výsledků bylo při testování metody v Matlabu v průměru dosaženo při délce analyzovaného segmentu 1024 s krokem 256 vzorků. Syntéza výstupního signálu pomocí oscilátorů, tak jak ji popisuje metoda popsaná v tomto článku, již nevyužívá FFT, což se na výpočetní náročnosti také negativně projevuje. Existují i jiné přístupy k implementaci fázového vokodéru, na něhož je nahlíženo jinak než na banku filtrů, [16].

Výhoda metod provádějící modifikaci signálu v kmitočtové oblasti je ta, že mohou být úspěšně využity na signál s vícehlasým obsahem, jelikož poloha každé kmitočtové složky vstupního signálu je s požadovaným poměrem změněna. To umožňuje aplikaci změny výšky tónu např. na celou skladbu, v níž může být zastoupeno několik hudebních nástrojů. Metody se zpracováním v časové oblasti v tomto ohledu nejsou vhodné, protože vícehlasý charakter vstupního signálu znemožňuje odhad jedné hodnoty základní periody, aplikace takových metod pak selhává [2].

6 Komunikační rozhraní MIDI

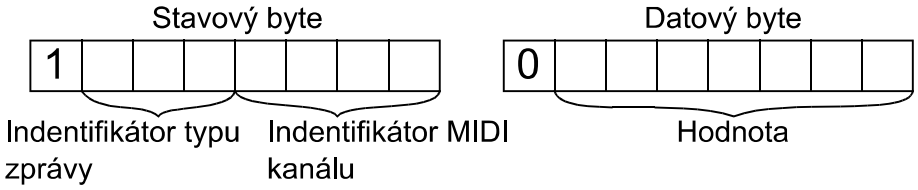
MIDI (Musical Instrument Digital Interface) je průmyslovým standardem vyvíjeným od počátku 80. let minulého století. Jeho cílem je umožnit komunikaci mezi hudebními nástroji, ovládání hudebních nástrojů, jejich automatizaci atd. Definované rozhraní tedy neslouží pro přenos samotného hudebního signálu, ale pro přenos řídících signálů mezi komunikujícími stranami. Příjemci (např. sampler či syntezátor) je doručena informace o tom, který tón má v daném čase generovat a jakou silou či jak mají být nastaveny parametry bloků ovlivňující zpracování signálu v přijímacím zařízení, tedy např. nastavení kmitočtových filtrů či parametry obálky časového průběhu signálu. Do řešení diplomové práce je popis MIDI protokolu zahrnut z důvodu jeho využití při realizaci virtuálního hudebního nástroje.

6.1 Struktura protokolu

Základním datovým blokem přenášeným přes MIDI rozhraní je tzv. MIDI zpráva (MIDI message). Ta se skládá z jednoho stavového a několika datových MIDI bytů nazývaných tzv. MIDI událost (MIDI event). MIDI událost je tedy reprezentována osmi bity, kde nejvýznamnější bit (MSB) určuje typ eventu. Pro MSB = 1 se jedná o stavový byte, pro MSB = 0 se jedná o datový byte [10].

MIDI zprávy se rozdělují na tzv. kanálová a systémová data. Jelikož lze přes MIDI rozhraní přenášet data odděleně až v 16 virtuálních kanálech, přenášená kanálová data jsou vždy platná pro specifikovaný kanál, jenž je určen dolními čtyřmi bity stavového bytu, viz Obr. 6.1. Naopak systémová data jsou platná pro všechny kanály, proto informaci o kanále nepřenáší.

Při přenosu MIDI zpráv je stavový byte následován několika datovými byty, jejich počet je závislý na typu zprávy.



Obr. 6.1: Struktura MIDI zprávy [15].

6.1.1 Kanálová data

Jak bylo zmíněno, kanálová data se vztahují pouze k jednomu virtuálnímu kanálu specifovaném ve stavovém bytu. Typ kanálových dat je určen třemi bity ve stavovém bytu, viz Obr. 6.1. V Tabulce 6.1 je sedm možných typů kanálových dat, poslední osmá nevyužitá možnost je vyhrazena pro systémová data.

Note On – Nota zapnuta

MIDI zpráva tohoto typu má identifikátor 0, přenáší informaci o stisknuté klávese. Za stavovým bytem této zprávy následují dva datové byty. První specifikuje číslo noty (celkem 128 různých not), druhý specifikuje dynamiku noty (většinou určena z rychlosti stisku klávesy, protože dynamika je jí přímo úměrná, taktéž 128 možných hodnot).

Note Off – Nota vypnuta

MIDI zpráva má identifikátor 1. Podává informaci o tom, která klávesa byla uvolněna a s jakou rychlostí, což je uvedeno ve dvou datových bytech následujících za stavovým bytem.

Tabulka 6.1: Typ zpráv kanálových dat.

MIDI zpráva	Význam	ID	Počet databytů
Note On	Nota zapnuta	0	2
Note Off	Nota vypnuta	1	2
Polyphonic Key Pressure	Individuální tlaková citlivost	2	2
Control Change	Změna kontroleru	3	2
Program Change	Volba programu	4	1
Channel Pressure	Společná tlaková citlivost	5	1
Pitch Bend Change	Ohýbání tónu	6	2

Polyphonic Key Pressure – Individuální tlaková citlivost

MIDI zpráva má identifikátor 2 a informuje o tlaku, jakým je působeno na klávesu po jejím stisku. V prvním datovém bytu je tedy přenášeno číslo noty určené stisknutou klávesou, v druhém bytu je přenášena hodnota tlaku působícího na tuto klávesu. Hodnota tlaku na klávesu může být využita pro ovládání parametrů znějícího tónu, často se však tato zpráva nepoužívá.

Control Change – Změna kontroleru

MIDI zpráva má identifikátor 3 a slouží pro nastavování parametrů ovlivňujících např. hlasitost nástroje, barvu zvuku či nastavení modulace. První datový byte přenáší číslo kontroleru, druhý byte pak jeho hodnotu. Datový byte je schopen přenést opět pouze 128 různých hodnot, což může být pro nastavování některých parametrů málo. Proto jsou kontrolery 0-31 sdruženy s kontrolery 32-63, kdy daná dvojice kontrolerů představuje nastavení jednoho parametru, přičemž kontroler z první skupiny přenáší nejvýznamnější byte a kontroler z druhé skupiny nejméně významný byte. Rozmezí hodnot, kterých může daný parametr nabývat, se tak zvýší ze 128 hodnot na 16384.

Program Change – Volba programu

MIDI zpráva má identifikátor 4. Přenáší pouze jeden datový byte, lze tak volit mezi 128 programy. Program může například představovat určité výchozí nastavení jednotlivých parametrů nástroje či skladbu zvuků atd.

Channel Pressure – Společná tlaková citlivost

MIDI zpráva má identifikátor 5 a přenáší pouze jeden datový byte. Informuje o tlaku, jakým je působeno na všechny stisknuté klávesy platné pro kanál určený stavovým bytem.

Pitch Bend Change – Ohýbání tónu

MIDI zpráva má identifikátor 6 a přenáší informaci o relativní změně výšky hraných tónů nahoru či dolů. Tato informace je přenášena ve dvou datových bytech, pro hodnoty 0 až 8191 se jedná o změnu výšky tónu směrem dolů, pro hodnoty 8193 až 16383 se jedná o změnu směrem nahoru.

6.1.2 Systémová data

Tato data neobsahují přímo informace hudebního charakteru, jako je tomu u kanálových dat, ale spíše provozní informace pro ovládaný nástroj, jako je např. synchronizace či reset systému. Tato data jsou ve stavovém bytu určeny zbývajícím identifikátorem 7. Informace o kanálu se již nepřenáší, proto jsou volné čtyři bity ve stavovém bytu využity přímo pro identifikaci typu systémových dat. Z hlediska potřeb diplomové práce je není třeba více rozebírat, pro bližší informace viz [10].

7 Změna vzorkovacího kmitočtu

V této kapitole je uvedena poznámka týkající se techniky změny vzorkovacího kmitočtu. U sampleru může nastat situace, kdy uživatel načte zvuk ze souboru, jehož vzorkovací kmitočet neodpovídá kmitočtu, se kterým pracuje používané výstupní zařízení, je tedy nutné zvukový signál na stejný vzorkovací kmitočet převzorkovat. Této operaci se říká převzorkování, viz [12]. Převzorkování signálu v poměru racionálního čísla se provádí tak, že se signál nejprve v poměru nějakého celého čísla nadvzorkuje a poté se v poměru jiného celého čísla podvzorkuje, viz Obr . 7.1.

Obr. 7.1: Převzorkování signálu v poměru racionálního čísla *L*/*M* [12].

Podvzorkováním signálu je chápán postup snižování jeho vzorkovacího kmitočtu, což je prováděno výběrem každého *M* - tého vzorku z původního signálu. U této operace musí být dodržen vzorkovací teorém, tzn. nejvyšší harmonická složka původního signálu nesmí být větší než polovina vzorkovacího kmitočtu, na kterou je signál podvzorkováván. Při nedodržení této podmínky se projeví aliasing, což má za následek degradaci signálu.

Nadvzorkováním signálu se rozumí postup zvyšování jeho vzorkovacího kmitočtu, což je prováděno vložením *L* - 1 nových aproximovaných hodnot vzorků mezi každé dva sousední vzorky původního signálu. Prakticky je to realizováno tak, že mezi každé dva sousední vzorky původního signálu je vloženo *L* - 1 nových nulových hodnot vzorků. Tyto nulové hodnoty vnesou do spektra vyšší harmonické složky, je tedy provedena filtrace pomocí dolní propusti, čímž dojde ke správné interpolaci vložených hodnot. Pro správné určení hodnot vložených vzorků musí být mezní kmitočet dolní propusti nastaven maximálně na polovinu původního vzorkovacío kmitočtu.

Bude-li třeba převzorkovat číslicový signál o vzorkovacím kmitočtu f_{vzx} na signál o vzorkovací kmitočtu f_{vzy}, je nejprve nalezen nejmenší společný násobek f_{vz} hodnot kmitočtů f_{vzx} a f_{vzy}. Na tento násobek f_{vz} vzorkovacích kmitočtů je nejprve signál nadvzorkován činitelem $L = f_{vz}/f_{vzx}$. Z tohoto vzorkovacího kmitočtu je pak signál podvzorkován činitelem $M= f_{vz}/f_{vzy}$ na signál o vzorkovacím kmitočtu f_{vzy}.

Aproximaci hodnot při nadvzorkování a antialiasingové omezení spektra při podvzorkování je možné při převzorkování číslicového signálu zajistit jedním číslicovým filtrem, viz Obr. 7.1. Pokud bude platit $f_{vzx} > f_{vzy}$, pak filtr musí mít mezní kmitočet nastaven maximálně na f_{vzy} /2 Hz. Pokud bude $f_{vzy} > f_{vzx}$, pak filtr musí mít mezní kmitočet nastaven maximálně na f_{vzx} /2 Hz.

Je tedy nutné mít k dispozici filtr, jehož proměnným parametrem bude hodnota mezního kmitočtu. Ta je zadávána na základě znalosti původního vzorkovacího kmitočtu f_{vzx} a cílového vzorkovacího kmitočtu f_{vzy} podle výše uvedených pravidel. Jelikož v případě jednoduchého sampleru nebude vadit fázové zkreslení signálu, lze použít filtr s nekonečnou impulzní odezvou (IIR), přenosová funkce takového filtru 4. řádu je

$$H(z)=\frac{b_4 z^4+b_3 z^3+b_2 z^2+b_1 z^1+b_0}{a_4 z^4+a_3 z^3+a_2 z^2+a_1 z^1+a_0} \quad , \tag{7.1}$$

kde b_4-b_0 a a_4-a_0 jsou koeficienty filtru.

Hodnoty koeficientů filtru b_4-b_0 a a_4-a_0 se určí pomocí využití bilineární transformace, kdy jsou vypočítány z předlohy analogového filtru, jehož koeficienty c_0-c_4 lze zjistit z tabulky, viz [13]. Koeficienty číslocového filtru pak jsou

$$
\begin{aligned}
b_4 &= \omega_p^4 T^4 c_0 \\
b_3 &= 4\omega_p^4 T^4 c_0 \\
b_2 &= 6\omega_p^4 T^4 c_0 \\
b_1 &= 4\omega_p^4 T^4 c_0 \\
b_0 &= \omega_p^4 T^4 c_0
\end{aligned}
$$

$$
\begin{aligned}
a_4 &= \omega_p^4 T^4 c_0 + 2\omega_p^3 T^3 c_1 + 4\omega_p^2 T^2 c_2 + 8\omega_p T c_3 + 16 c_4 \\
a_3 &= 4\omega_p^4 T^4 c_0 + 4\omega_p^3 T^3 c_1 - 16\omega_p T c_3 - 64 c_4 \\
a_2 &= 6\omega_p^4 T^4 c_0 - 8\omega_p^2 T^2 c_2 + 96 c_4 \\
a_1 &= 4\omega_p^4 T^4 c_0 - 4\omega_p^3 T^3 c_1 + 16\omega_p T c_3 - 64 c_4 \\
a_0 &= \omega_p^4 T^4 c_0 - 2\omega_p^3 T^3 c_1 + 4\omega_p^2 T^2 c_2 - 8\omega_p T c_3 + 16 c_4 ,
\end{aligned}
\qquad (7.2)
$$

kde T je vzorkovací perioda, c_0-c_4 jsou koeficienty předlohy analogového filtru požadované aproximace a ω_p je mezní úhlový kmitočet , který se z požadovaného mezního kmitočtu číslicového filtru ω_c určí podle vztahu

$$\omega_p = \frac{2}{T} tg \frac{\omega_c T}{2} \qquad (7.3)$$

označovaného jako předzkreslení kmitočtové osy, viz [13]. Podrobnější odvození koeficientů b_4-b_0 a a_4-a_0, viz [11].

Filtry vyšších řádů již není vhodné použít, protože např. pro převzorkování ze f_{vzx} =44,1 kHz na f_{vzy}=48 kHz je poměr nadvzorkování L = 160, vzorkovací kmitočet při návrhu dolní propusti tak bude $f_{vz} = Lf_{vzx}$, mezní kmitočet filtru f_c tak bude vzhledem k vzorkovacímu kmitočtu f_{vz} velmi malý a póly filtru se budou více přibližovat k jednotkové kružnici. Vlivem použité aritmetiky procesoru může dojít ke kvantizaci pólů tak, že se dostanou mimo jednotkovou kružnici a filtr již nebude stabilní. Pro lepší odfiltrování vyšších kmitočtových složek z nadvzorkovaného signálu filtrem 4. řádu lze provést filtraci vícekrát, avšak za cenu většího potlačení kmitočtových složek blížících se k meznímu kmitočtu filtru.

8 Využití techniky pro posun výšky tónu ve virtuálním nástroji typu sampler

Jedním z bodů zadání diplomové práce je navrhnout hudební nástroj, který bude pro svoji činnost využívat studované techniky posunu výšky tónu. Mezi nejznámnější nástroje, pro něž je tato technika velice podstatná, patří hudební nástroj nazývaný sampler. Principem tohoto nástroje, jak již název napovídá, je přehrávání krátkých zvukových nahrávek tzv. samplů. Tyto samply jsou tedy uloženy v paměti nástroje a při požadavku na vytvoření hudebního projevu jsou tyto zvuky přehrány. Aby byl nástroj použitelný v hudbě, zvuky ukládané do sampleru mívají často charakter hudebního tónu. V jejich spektru lze tedy vysledovat jistý základní kmitočet a vyšší harmonické složky. Podle hodnoty základního kmitočtu je pak možné zvuk přiřadit určité notě, bude-li tedy po sampleru požadováno generování právě této noty, stačí uložený vzorek přehrát. Z hlediska minimalizace paměťové náročnosti je snahou, aby pro každý tón nebylo nutné uložit do paměti nástroje zvuk splňující parametry požadovaného tónu, nýbrž výsledný zvuk odvodit z uloženého zvuku platného pro jiný tón. K tomuto odvození lze právě použít již zmiňované techniky posunu výšky tónu. Nelze samozřejmě předpokládat, že pro generování všech v hudbě používaných tónů vystačíme jen s jedním zvukem (např. klavír mívá rozsah 7 oktáv). Se zvětšující se mírou transpozice tónu se zřetelněji projevují artefakty používané metody pro změnu výšky tónu. Při překročení jisté míry již generované zvuky nepůsobí esteticky či se svým charakterem výrazně odlišují od požadovaného tónu. Uvedená míra je závislá na typu zvuku (typu nástroje). Pro některé nástroje je použití techniky posunu výšky vhodnější, pro některé naopak méně vhodné a je tedy nutné mít k dispozici více zvukových nahrávek, jejichž základní tóny jsou rovnoměrně rozmístěny přes používaný hudební rozsah [15].

Podobně může být pro jediný tón uloženo v paměti sampleru více zvuků, a to pro různé hodnoty zahrané dynamiky. Pro některé hudební nástroje nemusí platit, že pro různé intenzity jejich zvuku mají tóny stejné složení spektra a stačí tak daný zvuk zesílit či zeslabit. Přiřazením odlišných zvuků různým silám zahraného tónu tak lze simulovat proměnu spektra zvuku v závislosti na dynamice.

Pozn. Jako parametr určující, jak silně byl daný tón zahrán, bude v následujícím textu používán pojem rychlost (v korespondenci s MIDI protokolem).

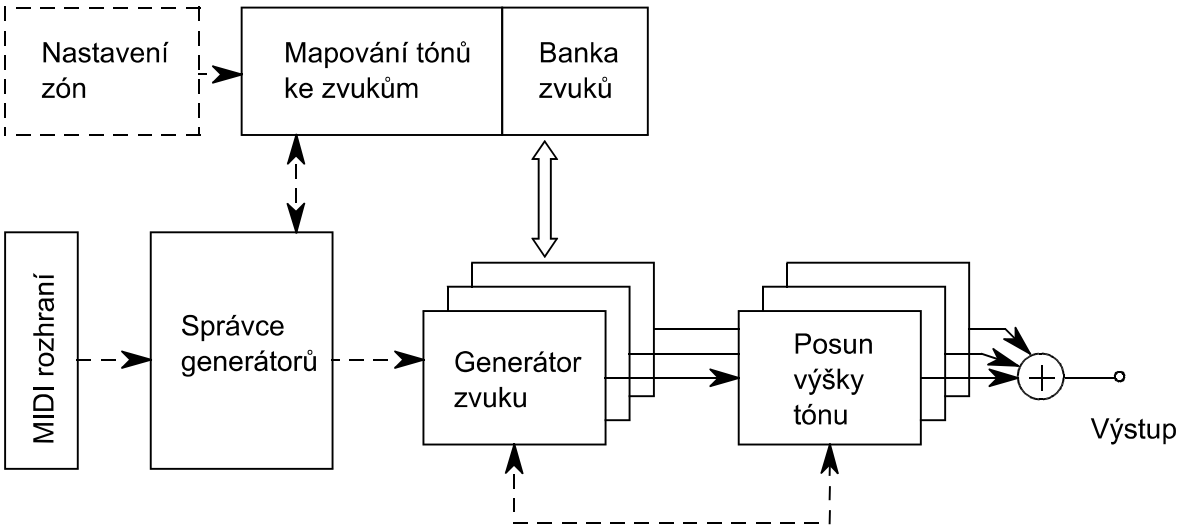
8.1 Popis virtuálního nástroje typu sampler

Na Obr. 8.1 je znázorněno blokové schéma znázorňující funkci sampleru. Spojité šipky ve schématu představují průchod zvukového signálu, čárkované šipky znázorňují řízení. Hlavním blokem ve schématu je správce generátorů, zajišťující požadovanou činnost sampleru v závislosti na příchozích požadavcích, jimiž se rozumí žádosti o generaci určitého tónu. Požadavky jsou realizovány pomocí MIDI událostí, které sampler přijímá pomocí bloku MIDI rozhraní. Generace zvuku o zvoleném tónu a rychlosti je spuštěna pomocí události Note On, generace zvuku je ukončena pomocí příchozí události Note Off.

Úkolem správce generátorů je pro jednotlivé příchozí požadavky na generaci tónu vyhradit generátor zvuku, při požadavku na ukončení generace tónu je generátor opět uvolněn k dispozici dalším požadavků. Toto schéma zajištuje polyfonnost sampleru, počet generátorů zvuku určuje, kolik tónů může zároveň znít v jednom okamžiku. Při implementaci ve formě softwaru je nutné tento počet stanovit vzhledem k výpočetní náročnosti především algoritmu pro posun výšky tónu a také efektivitě jeho implementace tak, aby sampler správně pracoval v reálném čase.

Při příchozí události Note On správce generátorů předá informaci o požadovaném tónu a rychlosti bloku mapování tónů, jenž zjistí, který ze zvuků uložený v bance zvuků odpovídá požadovanému tónu a rychlosti. Tuto informaci poté vrátí zpět správci generátorů zároveň s parametrem určujícím, jak je nutné upravit výšku tónu, aby odpovídala požadované notě. Blok mapování tónů ke zvukům má také uloženy pozice v jednotlivých zvucích, které mohou vytvářet

tzv. Sustaining Loop, což jsou smyčky, které vznikají při dlouho trvajících požadavcích na generaci tónu [15]. V případě krátkých zvuků tak lze vymezit úsek signálu, který bude cyklicky přehráván po dobu trvání požadavku na generaci tónu. Dále lze ve zvuku určit pozici, která společně s pozicí konce zvuku určí úsek, který ještě bude přehrán po příchodu události Note Off. Takto si lze pro každý uložený zvuk definovat dozvuk, který nastane po příchodu požadavku o zastavení generace tónu. Nastavení bloku mapování tónů ke zvukům probíhá při inicializaci sampleru, kdy je vytvořena databáze zón. Zóna obsahuje informaci o použitém zvuku a jeho výšce, tónový rozsah a rozsah rychlostí, pro který bude daný zvuk použit. Dále obsahuje pozice ve zvuku, které se použijí pro tvorbu sustaining loops a dozvuků.



Obr. 8.1: Funkční blokové schéma sampleru.

Poté, co správce generátorů zjistí zvuk odpovídající požadovanému tónu, požadovanou míru posunu a pozice pro tvorbu smyčky, vyhradí požadovanému tónu jeden generátor zvuku (je-li nějaký volný k dispozici) a předá mu tyto informace zároveň s povelem pro zahájení generace zvuku. Generovaný zvuk je pak v závislosti na požadovaném tónu kmitočtově upraven v bloku pro posun výšky tónu, každému generátoru zvuku musí náležet jeden blok pro posun výšky tónu, který je daným generátorem řízen.

Výstupní signály z jednotlivých bloků pro posun jsou pak sečteny a poslány na výstup.

8.2 Implementace sampleru

8.2.1 Využití technologie VST

VST technologie umožňuje realizaci softwarového digitálního hudebního nástroje či efektu ve formě tzv. plug-in modulu. Pod tímto pojmem si lze představit přídavný softwarový modul, který lze použít jako rozšíření hostující audio aplikace podporující technologii VST. Tímto lze dosáhnout například obohacení hostující aplikace o nový algoritmus zpracování digitálního zvukového signálu.

VST technologie definuje rozhraní nezbytné pro integraci VST plug-in modulu do řetězce digitálního zpracování audio signálu. Hostující aplikace zde plní funkci zprostředkovatele, kdy VST plug-in modulu předává vstupní data (audio signál kódovaný pomocí PCM, jehož vzorky jsou normalizovány v rozmezí hodnot < -1;1 > nebo MIDI data). Tato data mohou být například načítána ze souboru uloženém na pevném disku, nebo mohou být přímo odebírána ze vstupu zvukové karty. Tato vstupní data jsou algoritmem implementovaným ve VST plug-in modulu zpracována a již zpracovaná výstupní data jsou předána opět hostující aplikaci, která je může poslat na výstup zvukové karty. Hostující aplikace může také poskytovat jednoduché grafické uživatelské rozhraní (GUI – graphical user interface) sloužící pro editaci hodnot parametrů zpracování audio signálu,

nebo VST plug-in modul může obsahovat svoje vlastní GUI. Zdrojový kód VST plug-in modulu je nezávislý na platformě, ale forma, v jaké se vyskytuje, na platformě závisí. V operačním systému Microsoft Windows je realizován jako dynamická knihovna (DLL).

Pro vytvoření VST plug-in modulu se používá vývojová sada VST SDK ve formě zdrojových kódů tvořících VST API.

8.2.2 Využití frameworku JUCE

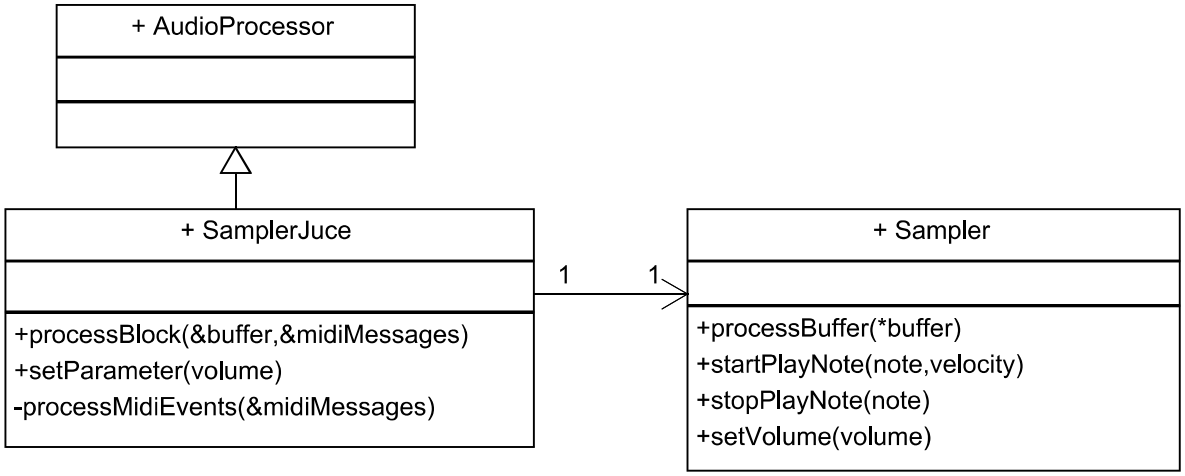
JUCE je multiplatformní knihovna pro jazyk C++, obsahuje velké množství tříd vytvářejících tzv. wrapper nad programovým rozhraním (API) cílové platformy. Knihovna disponuje širokými prostředky pro práci s grafikou a zvukem. Umožňuje snadné vytváření plug-in modulů pro zpracování digitálního zvukového signálu (VST, RTAS), pročež byla zvolena jako vhodný prostředek pro řešení této diplomové práce. Vývojová sada knihovny JUCE obsahuje velké množství ukázkových aplikací, které programátora názorně seznamují s různými oblastmi využití tohoto frameworku. Mezi těmito aplikacemi je zahrnuta i jednoduchá hostovací aplikace umožnující načítání vytvářených VST plug-in modulů, což usnadňuje jejich odlaďování.

Pro tvorbu VST plug-in modulu musí být ve zdrojovém kódu taktéž zahrnuty knihovny VST SDK, jádro VST plug-in modulu je v knihovně JUCE představováno třídou *JuceVSTWrapper* obsahující referenci na třídu typu *AudioProcessor*, ze které je pak odvozena třída virtuálního nástroje. S výhodou lze také využít možnost knihovny JUCE, kdy lze z kódu virtuálního nástroje typu *AudioProcessor* sestavit samostatnou aplikaci, pro otestování správné funkce již tedy není nutné mít k dispozici hostovací aplikace, což dále přináší větší komfort při vývoji.

8.2.3 Program sampleru

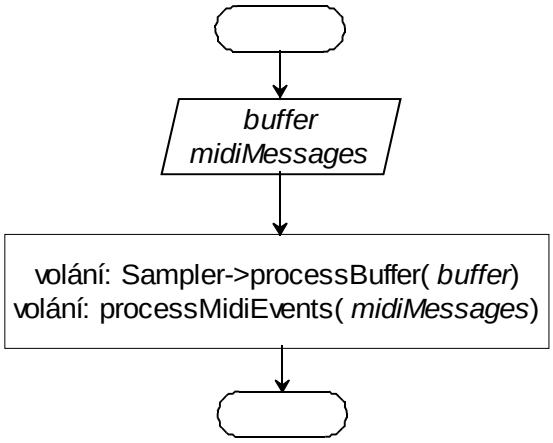
Snahou vývoje softwaru v objektově orientovaném jazyce je zapouzdření dílčích funkčních celků do samostatných tříd, což umožnuje jejich znovupoužití i v jiném kódu. Proto jsou veškeré funkce sampleru zapouzdřeny do třídy *Sampler*. V následujícím textu bude následovat popis tříd sampleru, v uvedených diagramech tříd budou pro přehlednost uvedeny pouze důležité metody.

Jak již bylo napsáno, při využití JUCE knihovny se moduly pro zpracování audia odvozují z virtuální třídy *AudioProcessor*, odvozenou třídu pak lze použít pro sestavení plug-inu nebo samostatné aplikace. Na Obr. 8.2 je vytvořena třída *SamplerJUCE* implementující čistě virtuální metody ze třídy *AudioProcessor*, dále obsahuje referenci na třídu *Sampler* představující zapouzdřený virtuální nástroj.



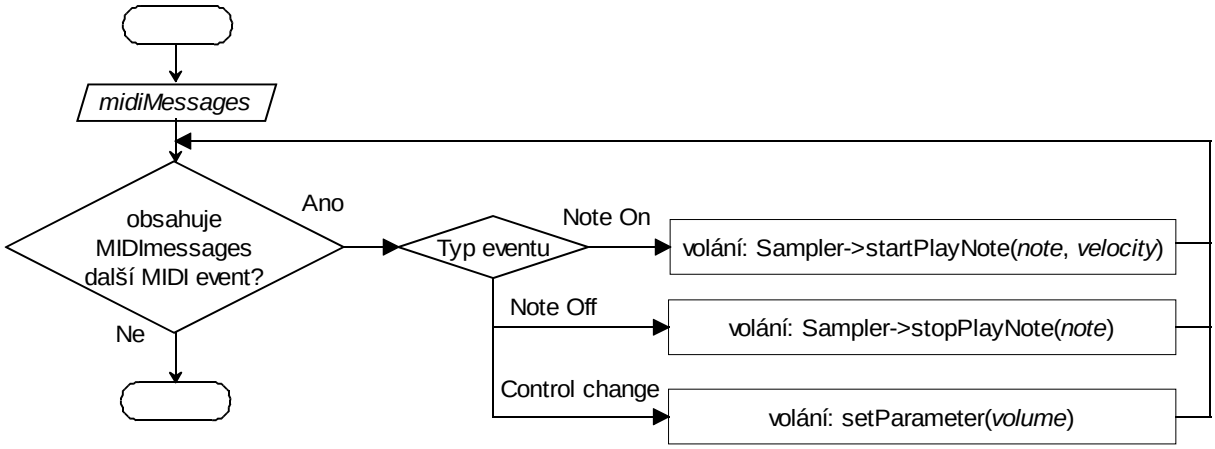
Obr. 8.2: Hlavni třídy v programu sampleru.

Obr. 8.3 znázorňuje metodu *processBlock* třídy *SamplerJuce*. Metodě je předán zásobník *buffer*, který je naplněn metodou *processBuffer* třídy *Sampler*. Pokud byly během doby odpovídající jednomu zvukovému zásobníku přijaty nějaké MIDI události, jsou uloženy v zásobníku *midiMessages*. Tento zásobník je zpracován pomocí metody *processMidiEvents*.



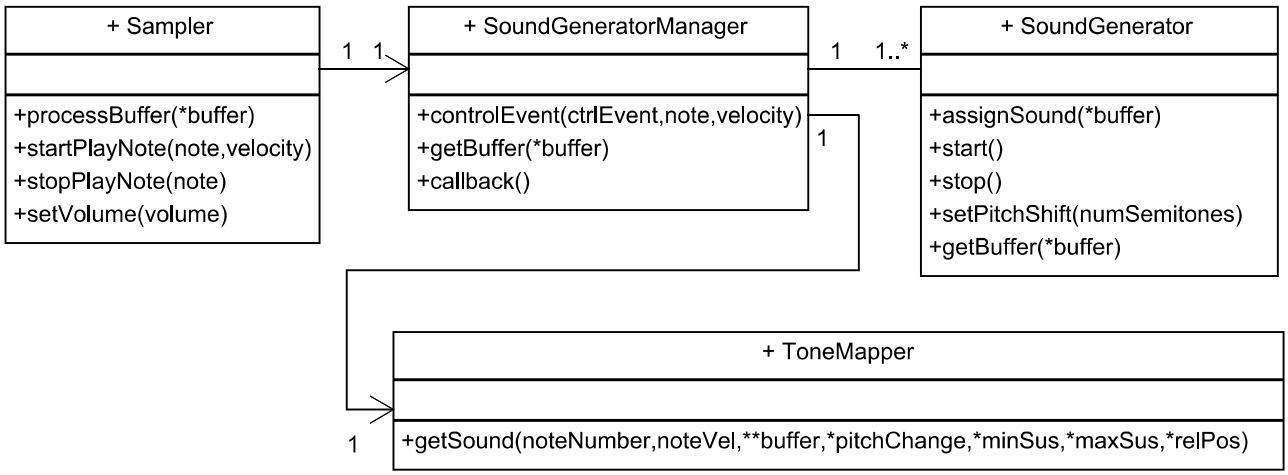
Obr. 8.3: Metoda *processBlock* třídy *SamplerJuce*.

Obr. 8.4 ukazuje činnost metody *processMidiEvents*. Ze zásobníku *midiMessages* jsou ve smyčce odebírány jednotlivé eventy a podle typu příchozí události se provede požadovaná akce, při příchodu událostí Note On nebo Note Off jsou zavolány metody pro započetí nebo skončení hraní, při příchodu události Control Change dojde k nastavení hlasitosti (pouze v případě, že číslo kontroleru odpovídá kontroleru hlasitosti, obvykle má číslo 0x07).



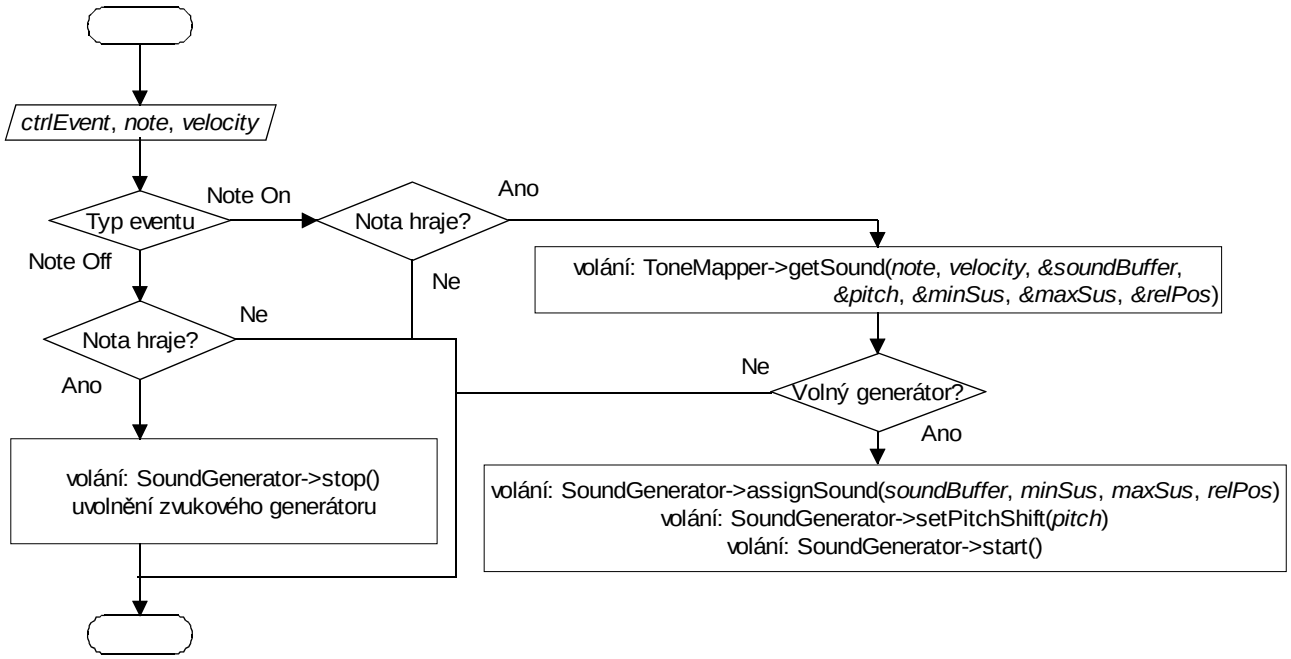
Obr. 8.4: Metoda *processMidiEvents* třídy *SamplerJuce*.

Na Obr. 8.5 je znázorněna skladba třídy *Sampler*, obsahuje referenci na třídu *SoundGeneratorManager*. Při zavolání metody *processBuffer* je zásobník *buffer* naplněn pomocí funkce *getBuffer* třídy *SoundGeneratorManager* a je vynásoben parametrem hlasitosti *volume*, který je nastavován pomocí metody *setVolume*. Metody *startPlayNote* a *stopPlayNote* jsou volány při příchodu specifických MIDI událostí, viz Obr. 8.4, a způsobí zavolání metody *controlEvent* třídy *SoundGeneratorManager*, jíž je předán typ události a číslo noty, případně rychlost.



Obr. 8.5: Vztah mezi třídami tvořící sampler.

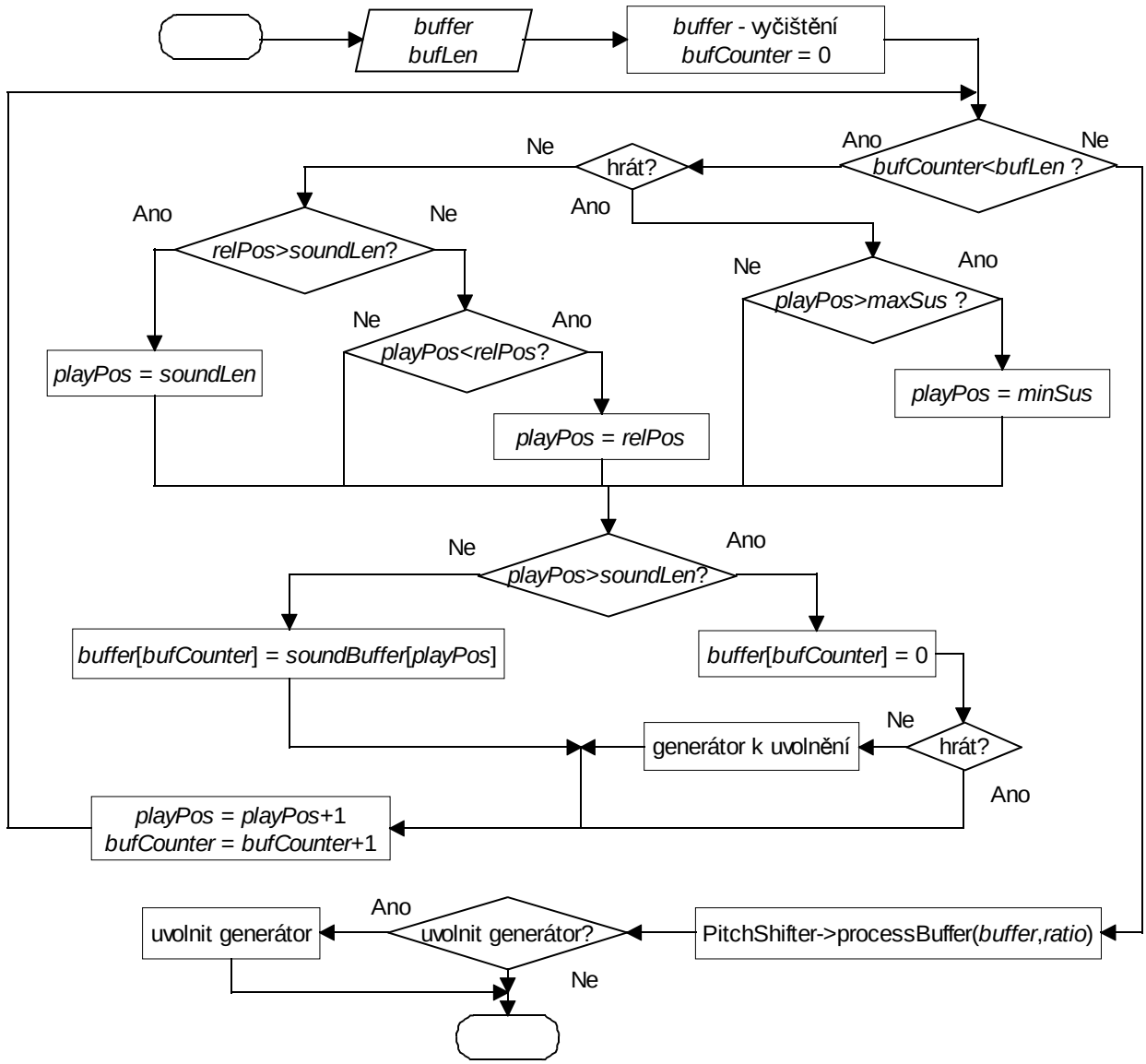
Třída *SoundGeneratorManager* slouží k obsloužení příchozích událostí a správě generátorů zvuku, obsahuje tedy reference na třídu *SoundGenerator* a *ToneMapper*. Obr. 8.6 představuje její metodu *controlEvent*. Byla-li metoda vyvolána událostí Note On, je zkontrolováno, zda-li již stejný tón není generován. Dále je zavolána metoda *getSound* třídy *ToneMapper*, která na základě požadovaného tónu a rychlosti zjistí odpovídající zvuk v bance, požadovanou míru posunu a také parametry pro tvorbu již výše zmiňované smyčky a dozvuku. Je-li k dispozici volný zvukový generátor, je vyhrazen danému tónu a poté je správci předána informace o zvuku, který bude generovat společně s parametry míry posunu tónu a parametry tvorby smyček a dozvuku. Nakonec je zvukový generátor spuštěn. Je-li zvukový generátor spuštěn, metoda *getBuffer* třídy *SoundGenerator* plní zásobník *buffer* vzorky generovaného zvuku. Metoda *getBuffer* třídy *SoundGeneratorManager* pak naplněné zásobníky od všech aktivních zvukových generátorů sečte do jednoho zásobníku, který je následně předán volající metodě *processBuffer* třídy *Sampler*.



Obr. 8.6: Metoda *controlEvent* třídy *SoundGeneratorManager*.

Obr. 8.7 popisuje algoritmus metody *getBuffer* třídy *SoundGenerator*, který zásobník *buffer* plní vzorky zvuku uloženého v zásobníku *soundBuffe*r. Po úvodní inicializaci algoritmus probíhá ve smyčcce, kdy je v každém průchodu do zásobníku *buffer* uložen jeden vzorek zvuku, první podmínka tedy kontroluje index *bufCounter*, zda-li je již zásobník naplněn.

V případě, že zásobník *buffer* naplněn není, je zjištěno, zda-li je ještě platný požadavek, že má být generován zvuk (tj. zda-li od posledního volání metody *getBuffer* nepřišla událost Note Off). Pokud trvá požadavek na generaci zvuku, je ověřeno, zda-li aktuální pozice *playPos* v zásobníku *soundBuffer* generovaného zvuku nepřesáhla oblast definující Sustaining Loop, tj. pozice je větší než *maxSus*. Pokud ano, je pozice *playPos* vrácena na začátek smyčky definované pozicí *minSus*. V případě, že si uživatel nepřeje využívat tuto smyčku, pozice *minSus* a *maxSus* jsou nastaveny na hodnotu větší než je délka *soundLen* generovaného zvuku, smyčka se pak v algoritmu neuplatní.



Obr. 8.7: Metoda *getBuffer* třídy *SoundGenerator.*

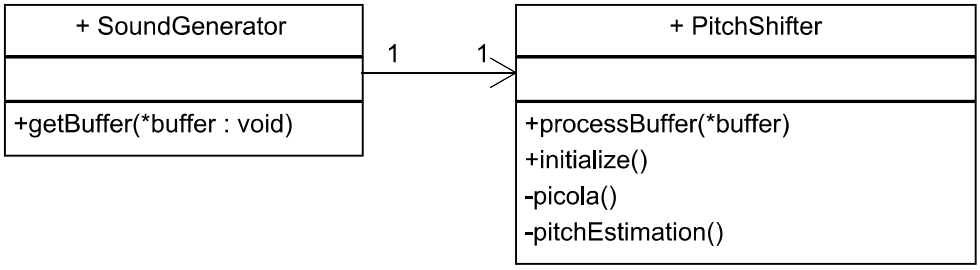
Pokud požadavek na generaci tónu již není aktivní, je testováno, zda-li se má uplatnit dozvuk, což je nastaveno pozicí *relPos*. Pokud dozvuk uplatněn není, musí být pozice *relPos* větší než délka *soundLen*, v tom případě je aktuální pozice *playPos* nastavena na hodnotu délky zvuku *soundLen* tak, aby zbytek zásobníku *buffer* byl naplněn pouze nulovými vzorky. V opačném případě se pozice *playPos* nastaví na pozici *relPos* tak, aby do zásobníku *buffer* byl ještě uložen úsek zvuku vymezený touto pozicí a koncem zvuku (což představuje požadovaný dozvuk).

Dále je v algoritmu na základě pozice *playPos* zjištěno, jestli byl již do zásobníku *buffer* uložen celý zvuk (včetně dozvuku). Pokud nebyl, jsou do něj dále ukládány vzorky ze zásobníku *SoundBuffer*. Pokud byl, jsou do něj již ukládány pouze nulové vzorky, skončil-li navíc požadavek na generaci tónu, je nastaven příznak pro uvolnění používaného zvukového generátoru.

Po ukončení smyčky plnění zásobníku *buffer* je na něj aplikován process posunu tónu (viz dále), v závěru je zkontrolován příznak, zda-li má být zvukový generátor uvolněn. Pro uvolnění je zavolána metoda *callback* třídy *SoundGeneratorManager* (viz diagram tříd na Obr. 8.5), která správce informuje o tom, že má daný generátor uvolnit a inicializovat pro další použití.

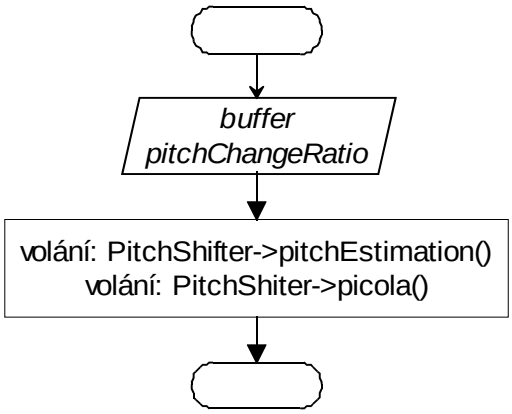
8.2.4 Proces posunu tónu

Na Obr 8.8 je znázorněn vztah tříd *SoundGenerator* a *PitchShifter*. Je vidět, že každému zvukovému generátoru odpovídá jeden blok pro posun tónu. Při požadavku na generaci tónu je volána metoda *initialize* třídy *PitchShifter* pro inicializaci. Jednotlivé zásobníky *buffer* naplněné zvukovým generátorem jsou předány metodě *processBuffer* třídy *PitchShifter*, která nad nimi provede zpracování.



Obr. 8.8: Vztah tříd *SoundGenerator* a *PitchShifter*.

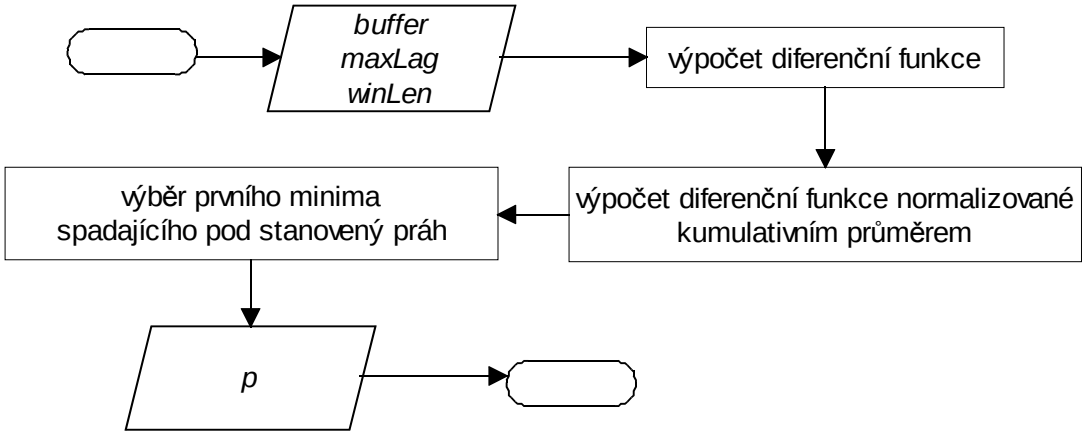
Na Obr 8.9 je metoda *processBuffer* vyobrazena. Z každého příchozího zásobníku je pomocí metody *pitchEstimation* odhadnuta základní perioda zpracovávaného zvuku a výsledek je uložen do členské proměnné *p* třídy *PitchShifter*. Metoda *picola* pak s využitím tohoto odhadu a parametrem požadované míry posunu provede zpracování zvuku uloženého v zásobníku *buffer*.



Obr. 8.9: Metoda *processBuffer* třídy *PitchShifter*.

Pro odhad základní periody jsou využity některé kroky metody yin, která byla rozebrána již v kapitole 3.2.1. Ve třídě *PitchShifter* je implementována ve formě metody *pitchEstimation*. Tuto metodu vyobrazuje Obr. 8.10. Třída *PitchShifter* stanoví na základě délky zásobníku *buffer* hodnotu maximálního posunu *maxLag* a délku integračního okna *winLen*, tyto parametry jsou stanoveny maximálně na polovinu délky zásobníku. Výsledná hodnota *maxLag* pak udává maximální periodu, kterou lze odhadnout. Ze zásobníku *buffer* je pak vypočítána diferenční funkce (3.13) a z ní pak

diferenční funkce normalizovaná kumulativním průměrem (3.14). Z této funkce je pak vybráno první minimum, spadající pod stanovenou hodnotu. Tato hodnota bývá stanovena na 0,1, viz [8]. Podle nalezeného minima je určen posun, který představuje počet vzorkovacích period v jedné periodě zvuku. Toto číslo je uloženo do členské proměnné *p*, která je dále využita metodou *picola*.

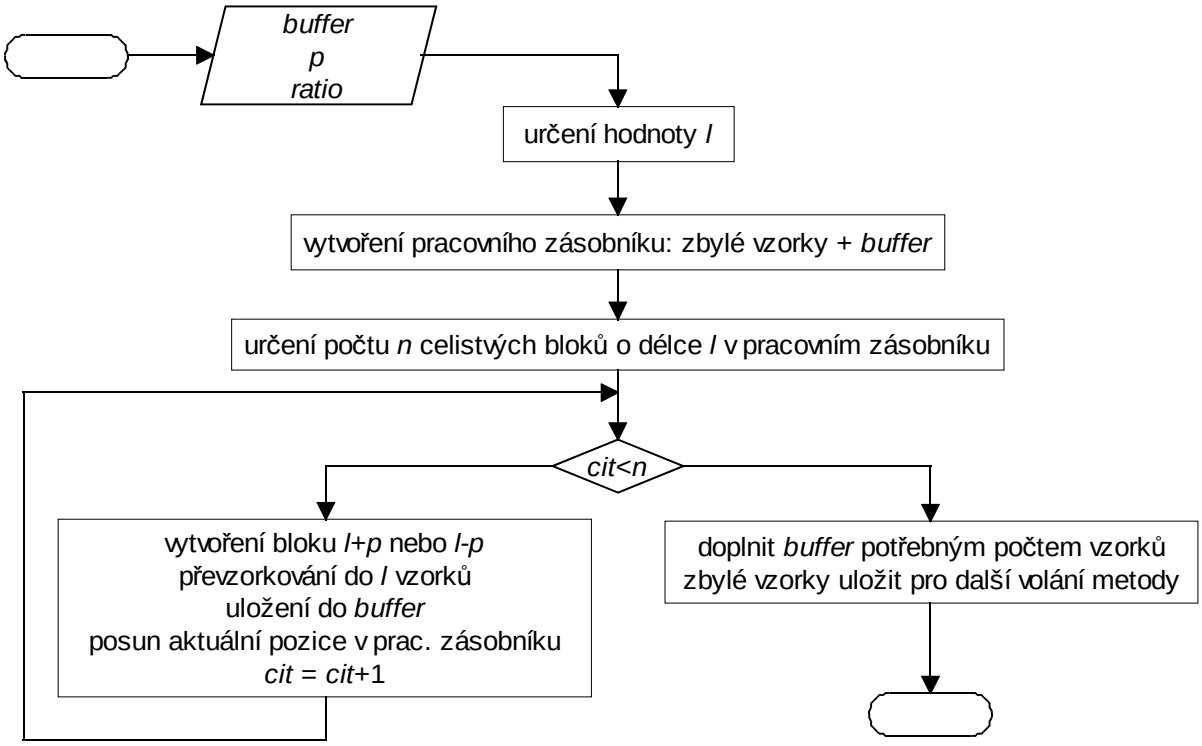


Obr. 8.10: Metoda *pitchEstimation* třídy *PitchShifter.*

Jak již napovídá název metody, v sampleru je pro posun výšky tónu použita metoda PICOLA, jejíž princip byl uveden již v kapitole 3.2. Jak bylo možné vidět, pro určení *l* vzorků zpracovaného signálu bylo nutné mít k dispozici *l+p* vzorků originálního signálu, které se s požadovaným poměrem podvzorkovaly (případ nadladění), nebo *l-p* vzorků, které se s požadovaným poměrem nadvzorkovaly (případ podladění). Daných *p* vzorků vznikne vždy překrytím a váhováním dvou sousedních bloků o délce *p*. Při zpracování je tedy nutné zásobník *buffer* rozdělit na celistvý násobek *n* bloků o délce *l*. Ke každým *l* vzorkům je následně třeba přidat nebo odebrat *p* vzorků vzniklých překrytím. Zásobník však nelze vždy přesně rozdělit na celistvý počet *n* bloků o délce *l*, po rozdělení tak vznikne i určitý zbytek vzorků ze zásobníku. Je-li však velikost zbytku menší než perioda *p*, nelze vždy z principu metody určit potřebný počet výsledných vzorků nutných pro úplné naplnění zásobníku *buffer.* Pro odstranění tohoto problému je tedy třeba ke každému zásobníku o dané délce mít navíc *p* vzorků. Toho lze dosáhnout zpožděním signálu o *p* vzorkovacích period. Před začátek každého zpracovávaného zásobníku *buffe*r tak bude vždy vloženo p_{max} vzorků z předchozího zásobníku, kde p_{max} je maximální možná hodnota odhadu základní periody. Z uvedeného řešení vyplývá, že daná implementace bude způsobovat zpoždění p_{max} vzorků, je tedy nutné najít kompromis mezi zpožděním a horní hranicí základní periody *p*, kterou bude možné algoritmem správně detekovat. Základní kmitočet zpracovávaných zvuků je tedy zdola omezen.

Dále může nastat situace, že ne všechny původní vzorky zásobníku *buffer* jsou v daném volání metody *processBuffer* využity. Tyto vzorky však nelze zahodit, protože by při dalších voláních metody *processBuffer* naopak chyběly. Proto je nutné nevyužité vzorky vždy uložit do paměti a přidat před zásobník *buffer* v dalším volání metody *picola*.

Algoritmus metody *picola* je zjednodušeně naznačen na Obr. 8.11. Podle odhadnuté hodnoty periody *p* a požadovaného poměru změny výšky tónu *ratio* dojde k určení délky bloku *l*. Dále k vytvoření pracovního zásobníku, do které jsou nejprve vloženy zbylé vzorky z předchozího volání metody *picola*, jak bylo uvedeno výše. Následně se určí počet *n* celistvých bloků o délce *l*. Následuje smyčka o *n* průchodech, kdy je v každém průchodu vytvořen blok o délce *l+p* nebo *l-p* (podle toho, jde-li o nadladění nebo podladění), tento postup je popsán v kapitole 3.2. Pro převzorkování je použita lineární interpolace, viz kapitola 3.1.1. Po *n* průchodech smyčkou dojde k zápisu zbývajících vzorků do zásobníku *buffer* tak, aby byl zaplněn. Poslední nevyužité vzorky v pracovním zásobníku jsou uloženy a využity při dalším volání metody *picola*.



Obr. 8.11: Metoda *picola* třídy *PitchShifter.*

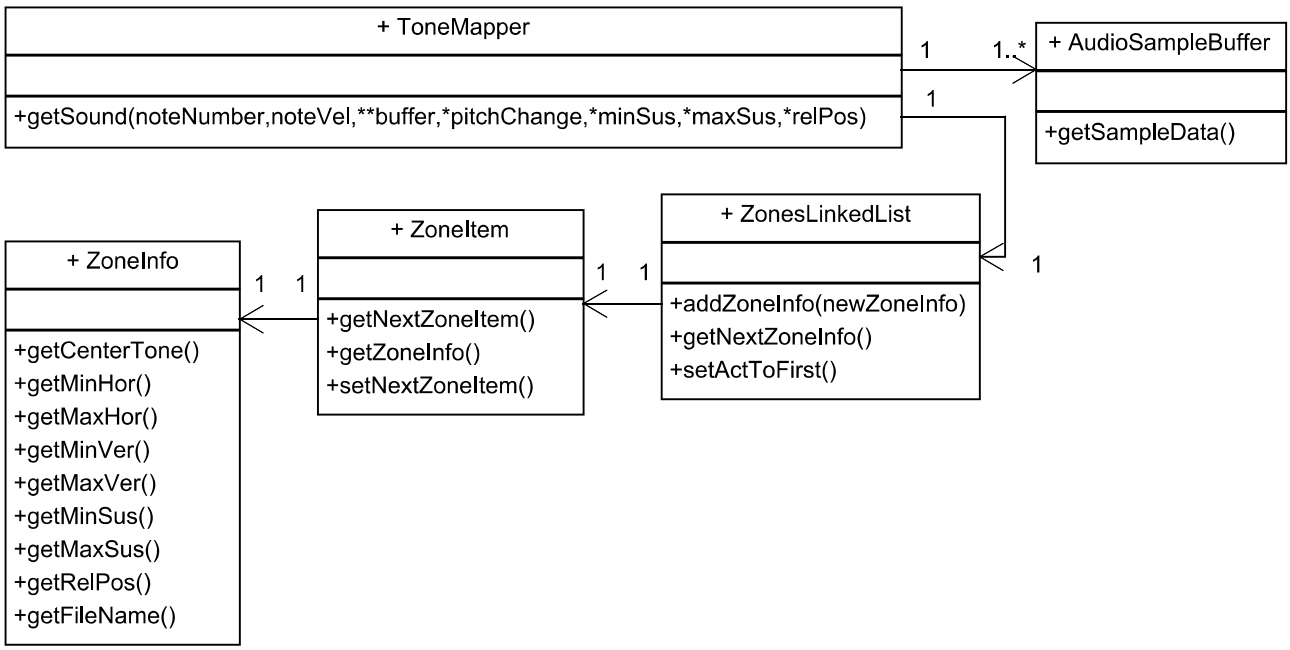
8.2.5 Nastavení bloku pro mapování zvuků k tónům

Na obrázku 8.12 je podrobněji rozepsán funkční celek bloku pro mapování zvuků k tónům. Tento blok je představován třídou *ToneMapper*, jejíž metoda *getSound* na základě požadovaného tónu a rychlosti vrací referenci na zásobník typu *AudioSampleBuffer* (třída frameworku JUCE), dále pak potřebnou míru posunu a parametry pro tvorbu smyčky a dozvuku (již bylo popsáno výše). Třída *ToneMapper* pro nalezení odpovídajícího zvuku udržuje referenci na třídu *ZonesLinkedList*, jež je implementací jednosměrně vázáného seznamu pro ukládání a vyhledávání dat. Takový seznam zahrnuje jednotlivé položky, přičemž udržuje referenci na první z nich. Každá položka seznamu pak obsahuje vždy jednu referenci na následující položku, viz [3]. Jednotlivé položky seznamu jsou představovány třídou *ZoneItem*, jež má informaci o dané zóně představovanou třídou *ZoneInfo*.

Banka zvuků je v programu implementována jako pole obsahující jednotlivé zvuky, zvuk je představován třídou *AudioSampleBuffer.*

Plnění jednosměrně vázaného seznamu položkami, jež obsahují potřebné údaje o jednotlivých zónách, probíhá při startu plug-inu či aplikace, tedy při spuštění konstruktoru třídy *ToneMapper*, tato procedura je vidět na Obr 8.13. Na začátku je vytvořena instance třídy *XmlSoundConfigReader*, která načte konfigurační XML soubor *zoneConfig.xml*. Tento soubor má následující strukturu:

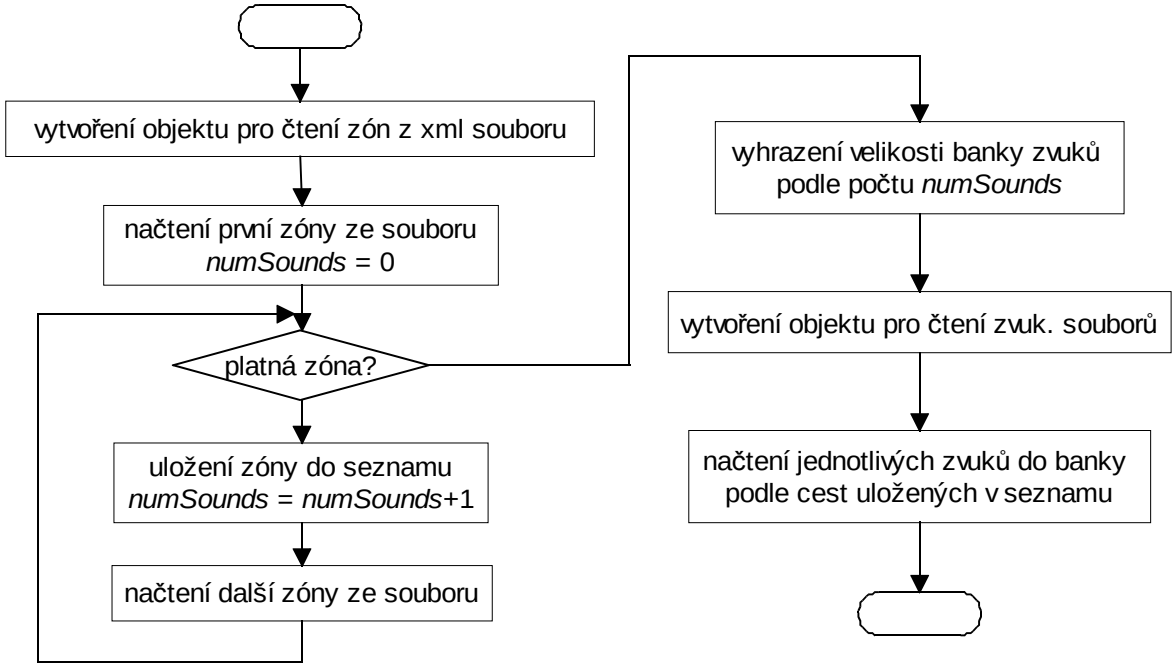
```
<Zones>
    <Note>
        <Tone Number = "45"/>
        <HorZoneRange Left = "5" Right = "6"/>
        <VerZoneRange Down = "0" Up = "127"/>
        <SustainLoop Start = "0.3" Stop = "0.5"/>
        <ReleasePos Start = "1.1" />
        <File Name = "D:\klavirA1.wav" />
    </Note>
</Zones>
```



Obr. 8.12 Třída *ToneMapper* sloužící k mapování zvuků k tónům.

Jedna zóna je vymezena pomocí tagů `<Note>` `</Note>`. Parametry zóny jsou uloženy v atributech jednotlivých elementů. Element `<Tone/>` specifikuje notu, které zvuk dané zóny odpovídá. Element `<HorZoneRange/>` určuje rozmezí tónů, pro které může být zvuk zóny použit, atributy `Left` a `Right` znamenají počet půltónů doleva a doprava (na klaviatuře) od tónu, kterému zvuk zóny odpovídá. Podobně element `<VerZoneRange/>` definuje rozmezí zahraných rychlostí, pro které může být zvuk zóny použit. Element `<SustainLoop/>` vymezuje smyčku ve zvuku zóny, která bude vytvořena při trvajícím požadavku na generaci tónu. Element `<ReleasePos/>` vymezuje část zvuku zóny, která bude přehrána jako dozvuk. Parametry smyčky a dozvuku se zadávají v rozmezí hodnot 0 až 1, pozice ve zvuku jsou tedy normovány vzhledem k počtu vzorků, jež zvuk zóny obsahuje. Pokud si uživatel nepřeje využít funkce smyčky a dozvuku, nastaví tyto parametry na hodnotu větší než 1.

Obr. 8.13 zobrazuje inicializaci bloku mapování. Ve smyčce jsou načítány jednotlivé zóny a jejich parametry ukládány do objektů typu *ZoneInfo*, které jsou ve formě položek typu *ZoneItem* ukládány do seznamu typu *ZonesLinkedList*, tento vztah je ukázán na obrázku 8.12. S každým průchodem smyčkou je inkrementován čítač *numSounds*, který značí počet načtených zón. Jakmile jsou všechny zóny ze souboru načteny, je na základě jejich počtu vyhrazeno pole představující banku zvuků. Dále jsou postupně z naplněného seznamu vyčteny cesty k souborům jednotlivých zón, které jsou předány objektu typu *WavReader*, který pro každou cestu načte daný soubor typu wav a zvukové vzorky uloží do zásobníku na příslušnou pozici v bance zvuků.



Obr. 8.13: Inicializace bloku pro mapování zvuků k tónům.

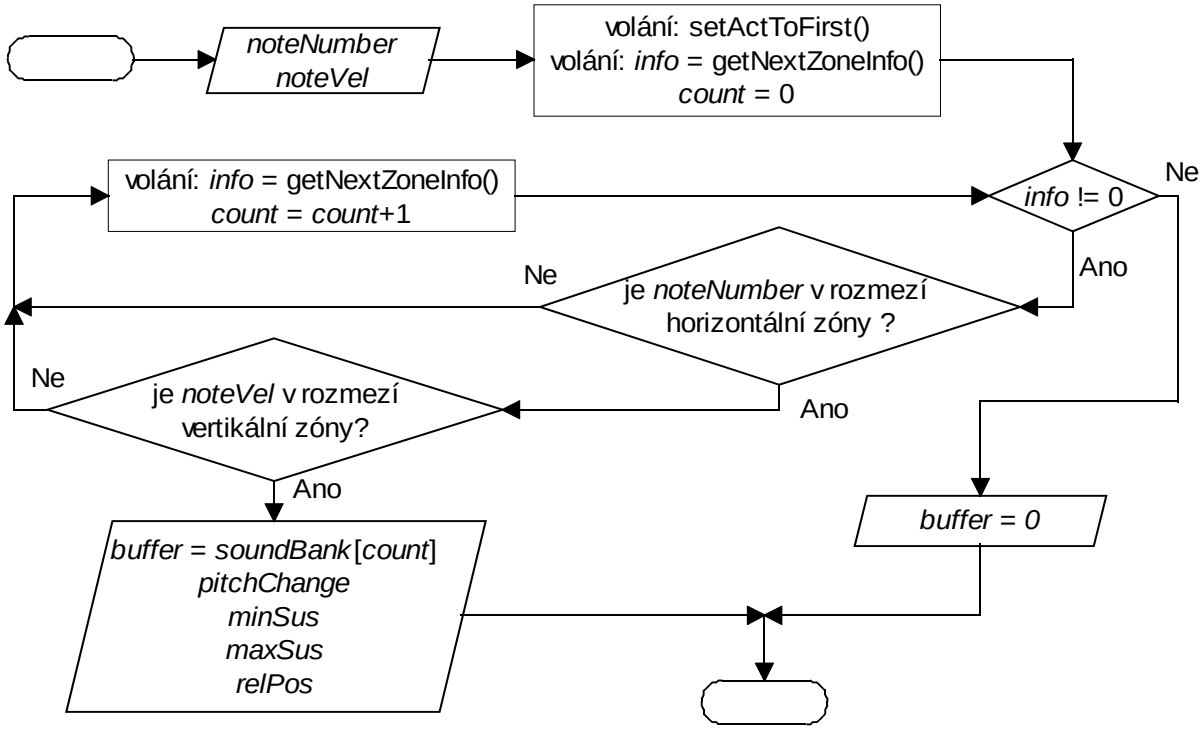
Metodu *getSound* třídy *ToneMapper* ilustruje Obr. 8.14. Vstupní hodnotou je číslo požadované noty *noteNumber* a rychlost *noteVel*. Třída *ZonesLinkedList* kromě reference na první položku seznamu obsahuje také referenci na aktuální porovnávánou položku. Tato reference je v úvodu nastavena také na první položku seznamu a pomocí volání metody *getNextZoneInfo* je vyčtena informace o zóně uložená v první položce. Ve smyčce je kontrolováno, zda-li aktuálně vyčtená zóna odpovídá vstupním požadavků, zároveň je s každým průchodem smyčkou inkrementován čítač *count*. V případě nalezení shody mezi aktuální vyčtenou zónou a požadovaným vstupem je smyčka opuštěna. Výstupem metody je reference *buffer* na odpovídající zvuk z banky zvuků, jež je určena pomocí výsledné hodnoty čítače *count*. Požadovaný kmitočtový posun *pitchChange* se určí podle rozdílu požadované noty a noty představované vyčteným zvukem. Výstupem jsou taktéž parametry pro tvorbu smyčky a dozvuku. V případě, že požadované notě neodpovídá žadná ze zón, reference na zvuk je naplněna nulovou hodnotou.

8.2.6 Převzorkování zvuků

V případě načítání zvuků, jejichž vzorkovací kmitočet neodpovídá vzorkovacímu kmitočtu, se kterým pracuje hostující aplikace, je potřeba zvukový signál převzorkovat. Princip převzorkování byl již popsán v kapitole 7.

Převzorkování v programu sampleru zajišťuje třída *Resampler* obsahující metodu *resample* znázorněnou na Obr. 8.15, které je v případě potřeby převzorkování předán zásobník *buffer* používaného zvuku, jeho vzorkovací kmitočet *sourceSR* a požadovaný vzorkovací kmitočet *destSR*. Nejprve je nalezen nejmenší společný násobek *SR* vzorkovacích kmitočtů *sourceSR* a *destSR*, kdy jsou hodnoty NSN_1 a NSN_2 nainicializovány na hodnoty *sourceSR* a *destSR*. Pokud si tyto hodnoty nejsou rovny, ve smyčce je vždy k menší z nich přičítána hodnota *sourceSR* nebo *destSR* tak dlouho, dokud se nerovnají. Dále je určen mezní kmitočet dolní propusti *cutoffFreq* a podle něj jsou určeny její koeficienty podle (7.2) pro 4. řád. Mezi každý vzorek původního signálu je vloženo *SR*/*sourceSR*-1 nulových vzorků, poté dojde k filtraci tohoto signálu tolikrát, aby zkreslení vzniklé nadvzorkováním a podvzorkováním bylo potlačeno do takové míry, kdy bude maskováno užitečným zvukovým signálem a pro lidský sluch tak bude neslyšitelné. Pro zvuky, které jsou převzorkovávány mezi kmitočty typické pro vzrokování audia (22,05 kHz, 44,1 kHz, 48 kHz, atd.)

postačí průchod tříkrát filtrem 4. řádu. Filtr je realizován v 1. kanonické struktuře, viz [13]. Po filtraci dojde k podvzorkování s činitelem *SR*/dest*SR* a výsledné vzorky jsou uloženy do zásobníku *buffer2*, který je odevzdán volající funkci. Metoda *resample* je volána buď při inicializaci objektu typu *ToneMapper*, kdy dochází k načítání zvuků do paměti, nebo v situaci, kdy hostující aplikace změní cílový vzorkovací kmitočet.



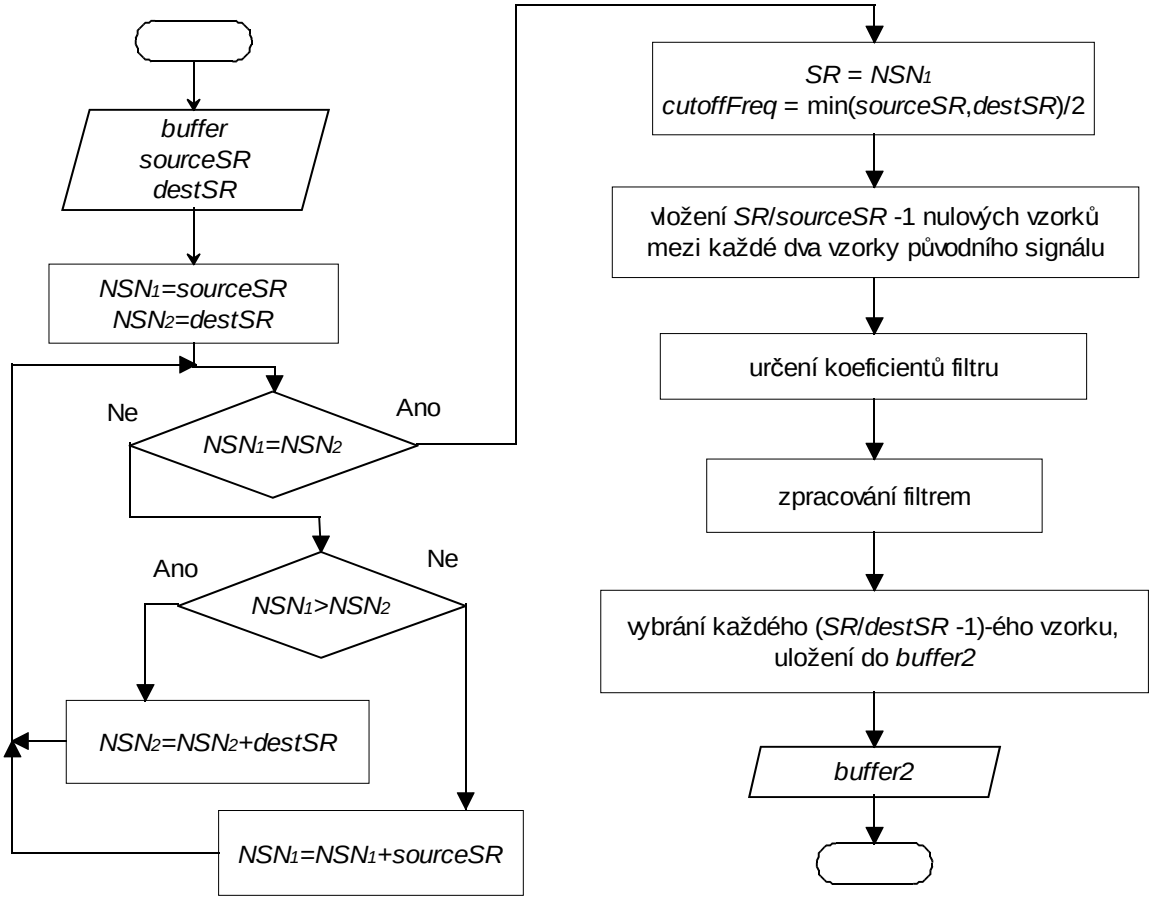
Obr. 8.14: Metoda *getSound* třídy *ToneMapper.*

8.2.7 Grafické uživatelské rozhraní sampleru

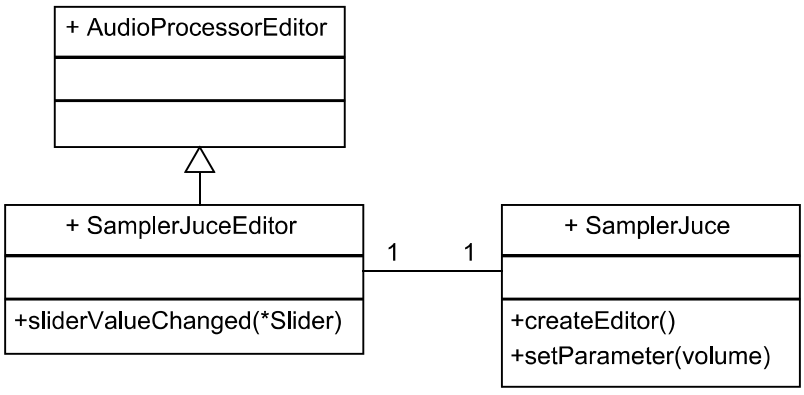
V případě VST plug-in modulu, některé hostovací aplikace poskytují svůj implicitní editor pro nastavování parametrů plug-inu. V případě hostovací aplikace bez této podpory je nutné vytvořit vlastní grafické uživatelské rozhraní (GUI) plug-inu, pomocí kterého budou ovládány jeho parametry.

Knihovna JUCE již obsahuje kódy, pomocí kterých lze snadno vytvořit samostané okno obsahující ovládací prvky, se kterými lze manipulovat. Tyto změny se přenesou na změny parametrů ve VST plug-in modulu. Grafický editor v knihovně JUCE je představován třídou *AudioProcessorEditor*, z níž je odvzozena třída editoru sampleru *SamplerJuceEditor*, což je vidět na Obr. 8.16. Při požadavku na zobrazení editoru je zavolána metoda *createEditor* třídy *SamplerJuce*, která vytvoří objekt typu *SamplerJuceEditor* a nastaví vzájemné reference. Dojde-li v grafickém editoru ke změně, je zavolána metoda *sliderValueChanged* třídy *SamplerJuceEditor*, díky referenci na objekt sampleru dojde k zavolání metody *setParameter* třídy *SamplerJuce*, čímž se změna parametru přenese do sampleru. Reference mezi třídami *SamplerJuceEditor* a *SamplerJuce* je oboustranná, dojde-li tedy ke změně parametru například na základě MIDI zprávy, změna parametru se projeví i v grafickém editoru.

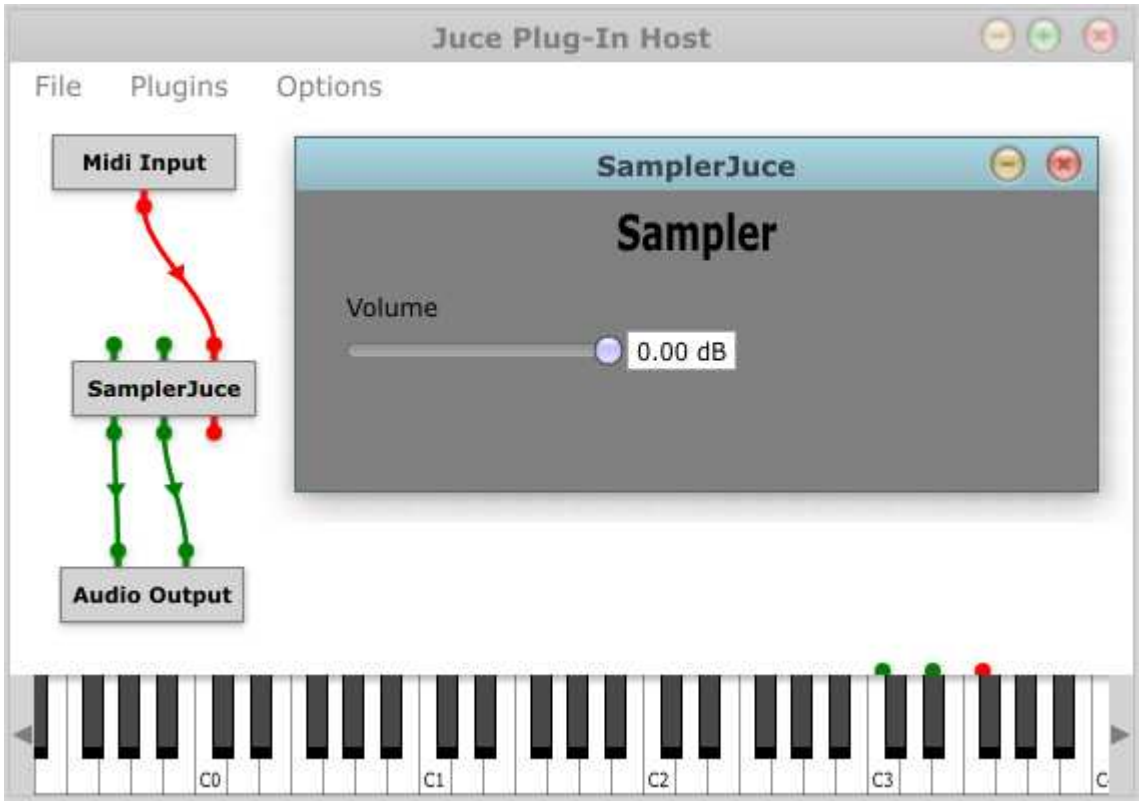
Na obrázku 8.17 je zobrazeno grafické uživatelské rozhraní sampleru ve formě VST plug-in modulu. Plug-in modul byl načten pomocí hostovací aplikace, která je součástí frameworku JUCE. Jak je vidět, editor sampleru má pouze jeden parametr, a tím je hlasitost. Generování tónů je již řízeno pomocí příchozích MIDI událostí, tyto události jsou řízeny pomocí klaviatury obsažené v hostovací aplikaci.



Obr. 8.15: Metoda *resample* třídy *Resampler.*



Obr. 8.16: Vztah tříd *SamplerJuceEditor* a *SamplerJuce.*



Obr. 8.17: Zobrazení plug-in modulu načteného do hostovací aplikace.

9 Využití nástroje pro verzování zdrojového kódu a automatické generování dokumentace

Součástí zadání diplomové práce bylo při jejím řešení používání nástroje pro verzování zdrojového kódu a nástroje pro automatické generování dokumentace ze zdrojového kódu. Následující část se o těchto nástrojích stručně zmiňuje a podává jejich popis.

9.1 Nástroj Subversion pro verzování zdrojového kódu

Verzovací systém je hojně využívaným prostředkem především v odvětví vývoje softwaru. Slouží ke správě adresářů a souborů, hlavním rysem je uchovávání informací o změnách v průběhu času, které byly nad adresáři a soubory provedeny. Pomocí této vlastnosti lze zkoumat historii vývoje a v případě potřeby se vracet i ke starším verzím uchovávaných dat.

Pro účely vývoje sampleru byl využit verzovací systém Subversion (SVN), podrobné informace viz [4]. Důvodem jeho volby byla jeho všeobecná oblíbenost a rozšířenost. Subversion umí pracovat po síti, proto lze využít pro práce více uživatelů na různých pracovním stanicích. V takové situaci pak v sítí existuje počítač (server), na kterém běží program s názvem SVN server, který spravuje hlavní uložiště dat nazývané repozitář. Repozitář udržuje data ve formě stromu souborového systému. K repozitáři pak mohou přistupovat uživatelé z různých stanic, které jsou vybaveny programem nazývaným SVN klient. Prostřednictvím SVN klienta mohou uživatelé z repozitáře soubory číst, nebo do nich zapisovat. Uvedený systém se od klasického souborového serveru liší v tom, že repozitář si uchovává informace o všech změnách, které v něm byly provedeny (např. vytvoření adresáře, pozměnění souboru, smazání souboru atd.). Klient pak po serveru může žádat kteroukoliv verzi, která byla v čase v repozitáři vytvořena. Při každém zápisu do repozitáře je tato změna uložena a je uložen jeho stav, který se nazývá revizí. Jednotlivé revize jsou číslovány vzestupně tak, jak postupně dochází k modifikacím repozitáře.

Jak už bylo uvedeno, uživatelé mohou z repozitáře číst a pořizovat si tak kopie celého nebo jen části stromu repozitáře o požadované revizi, a to prostřednictvím příkazu *checkout*. Taková kopie repozitáře, která se nachází na klientské stanici, se nazývá pracovní kopie. Jednotlivé soubory v kopii může uživatel podle potřeby upravovat, nebo do kopie nové soubory či adresáře přidávat. V případě přidávání však ještě soubory a adresáře nejsou pod kontrolou verzovacího systému, proto je třeba je do stromu repozitáře přidat pomocí příkazu *add*. Po dokončení potřebné práce může uživatel změny, které v pracovní kopii provedl zapsat do repozitáře uloženého na serveru pomocí příkazu *commit*. Při vykonávání tohoto příkazu dojde k odeslání změn v souborech do repozitáře na serveru, po zápisu těchto změn je vytvořena nová revize. Před samotným odesláním pomocí příkazu *commit* by si však měl uživatel A pomocí příkazu *update* ještě zkontrolovat, že před ním už jiný uživatel B do repozitáře nezapisoval. V takovém případě dojde ke spojení změn v repozitáři na serveru, které provedl uživatel B, a změn provedených uživatelem A do pracovní kopie. Tato aktualizovaná pracovní kopie je pak odeslána pomocí příkazu *commit* na server. V případě spojování změn může dojík k situaci, kdy verzovací systém není schopen tyto změny automaticky spojit, dojde k tzv. konfliktu. V tom případě je nutné daný konflikt manuálně vyřešit.

Program SVN serveru lze provozovat na vlastním počítači, kde běží ve formě služby (MS Windows) či démona (Unix) a naslouchá na příslušném portu požadavkům SVN klientů. V poslední době lze s výhodou využít služeb společností provozujících velká datová úložiště na vlastních serverech v Internetu, na kterých beží i program SVN serveru. Lze tedy mít data z repozitáře k dispozici z jakékoliv stanice s internetovým připojením. Tohoto přístupu bylo využito i při řešení této diplomové práce.

9.2 Nástroj Doxygen pro automatické generování dokumentace ze zdrojového kódu

Nástroj pro generování automatické dokumentace slouží k vytvoření přehledné dokumentace, která je získána ze struktur zdrojového kódu a jeho stanoveným způsobem formátovaných komentářů.

Při řešení této diplomové práce byl pro dokumentaci zdrojového kódu použit nástroj Doxygen. Výsledná dokumentace může být uložena v různých formách, např ve formátu HTML, výslednou dokumentaci pak lze pohodlně procházet v prohlížeči webových stránek využitím hypertextových odkazů. Nástroj lze použít nejen pro dokumentování vlastního kódu, ale i k extrakci struktury cizího kódu, který dokumentovaný není, což může být velmi užitečné pro zorientování se ve velmi rozsáhlém kódu. Dále lze vizualizovat vztahy mezi různými elementy kódu např. pomocí grafů zobrazujících dědičnost [7].

Program Doxygen dokáže ve zdrojovém kódu rozeznávat různé elementy (např. třídy, struktury atd.) Do dokumentace také umí zahrnout komentáře, které jsou ve zdrojovém kódu obsaženy. Tyto komentáře jsou pak v dokumentaci přiřazeny danému elementu, u kterého se ve zdrojovém kódu nacházejí. Aby však byly komentáře do dokumentace správně vyextrahovány ze zdrojového kódu, je třeba dodržet jejich požadované formátování definované programem Doxygen. Zde je uveden příklad jednoho z možných formátování komentářů ve zdrojovém kódě v jazyce C++:

```
//! Popis třídy TestDoc
/*!
     Bližší popis třídy TestDoc
*/

class TestDoc
{
public:
     //!Popis konstruktoru TestDoc()
     TestDoc();
};
```

Jak je vidět, uvedené komentáře jsou oproti klasickým v C++ rozšířeny o znak !. V dokumentaci pak budou patřit k elementům, které po nich v kódu následují.

Při generování dokumentace je nutné programu Doxygen předat jako parametr textový konfigurační soubor, který obsahuje různá nastavení (např., které elementy kódu mají být zahrnuty do kódu a které ne, typ výstupu atd.), dále pak obsahuje cestu k adresáři, který obsahuje zdrojové kódy, jež budou zpracovávány, a cestu k adresáři, kam bude výstup programu uložen. Kromě nastavení pomocí konfiguračního souboru lze použít i nastavení pomocí programu Doxywizard, kde jsou parametry generování nastaveny pomocí grafického uživatelského rozhraní. Výhoda konfigurace použitím textového souboru je však ta, že jde generování dokumentace snadno zautomatizovat pomocí skriptů.

10 Závěr

Hlavním úkolem této diplomové práce bylo studium metod pro posun výšky základního tónu hudebních zvukových signálů. V úvodu práce byly jednotlivé metody teoreticky rozebrány společně s ostatními technikami, které s jejich podstatou úzce souvisí (linární interpolace, odhad základní periody signálu, krátkodobá spektrální analýza). Metody zmiňované v této diplomové práci byly implementovány v prostředí Matlab, kde byly vždy jednorázově použity pro úpravu testovacích signálů. Zkoumány byly metody jak pro zpracování v časové doméně (využití modulace zpožďovací linky, PICOLA), tak i pro zpracování v doméně kmitočtové (fázový vokodér). Byly vyhodnocovány z hlediska jejich kvality a výpočetní náročnosti. Poté byly uvedené metody srovnány.

Praktickou částí diplomové práce byla realizace virtuálního hudebního nástroje typu sampler, který studované problematiky využívá pro svoji funkci. Zadání práce dále požadovalo, aby byl výsledný nástroj ovladatelný pomocí MIDI protokolu, proto je v práci zahrnut jeho popis. Sampler pro generování tónů využívá zvuků načtených do paměti ze souborů typu wav uložených na disku. V případě, že vzorkovací kmitočet načteného zvuku neodpovídá vzorkovacímu kmitočtu, se kterým výstupní zvukové zařízení aktuálně pracuje, musí dojít k převzorkování načtených zvuků. Pro tento účel je v práci rozebrán postup změny vzorkovacího kmitočtu signálu.

Softwarový sampler byl naprogramován v jazyce C++ s využitím knihovny JUCE, jež byla zvolena pro své užitečné funkce. Díky ní existuje výsledný produkt ve dvou formách. První typ je ve formě VST plug-in modulu, jako hostovací aplikace je použita aplikace dodávaná jako součást knihovny JUCE, která obsahuje vlastní MIDI klaviaturu využitelnou pro ovládání načteného plug-in modulu. Druhá forma je představována samostatnou aplikací, jejíž vlastnosti se v ničem neliší od vlastností plug-in modulu. Pro použití tohoto sampleru již není třeba využívat hostovací aplikace, na druhou stranu je nutné mít vlastní MIDI klaviaturu pro jeho ovládání.

V práci byl realizovaný sampler popsán z pohledu jeho jednotlivých funkčních bloků, postupně byla rozebrána jejich softwarová implementace pomocí diagramů tříd, nejdůžitější metody tříd byly vysvětleny užitím vývojových diagramů. Je zde znázorněn mechanismus přiřazování zvuků k požadovaným tónům. Pro posun tónu byla jako nejvhodnější zvolena metoda PICOLA, proto je v diplomové práci obsažen popis její implementace obsahující jak samotnou metodu, tak i postup odhadu základního tónu.

Jelikož bylo součástí zadání diplomové práce při vývoji softwaru používat taktéž verzovacího systému a nástroje pro automatické generování dokumentace, obsahuje poslední část dokumentu stručný popis verzovacího systému Subversion a dokumentačního nástroje Doxygen.

11 Literatura

[1] ATASSI, H. Metody detekce základního tónu řeči. *Elektrorevue* [online]. 2008. Dostupný z WWW: <www.elektrorevue.cz>.

[2] BERNSEE, S.: *DSPDimension* [online]. 1999. Time Stretching And Pitch Shifting of Audio Signals – An Overview. Dostupné z WWW: <www.dspdimension.com>.

[3] BURGET, R.: *Abstraktní datové typy(1)*, Přednášky k předmětu teoretická informatika, Dostupné z WWW.:<www.vutbr.cz/elearning>

[4] COLLINS-SUSSMAN, B.; FITZPATRICK, B.; PILATO, C. *Version Control with Subversion* [online]. 2008. Dostupné z WWW: <www.collab.net>.

[5] DATTORRO, J.: *AES* [online]. 1997. Effect Design: Part 2: Delay-Line Modulation and Chorus. Dostupný z WWW: <www.aes.org>.

[6] DISCH, S; ZÖLZER, U.: *Workshop on Digital Audio Effects*. 1999. Modulation and delay based digitalaudio effects.

[7] D*oxygen* [online]. 2011. Doxygen manual. Dostupné z WWW: <www.doxygen.org>.

[8] CHEVEIGNE, A; KAWAHARA, H. YIN, a fundamental frequency estimator for speech and music. In *J. Acoust. Soc. Am., Vol. 111, No. 4, April 2002* [online] Dostupné z WWW: <audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf>.

[9] IKEDA, M. *PICOLA and TDHS* [online]. 2006 PICOLA and TDHS. Dostupné z WWW: <http://keizai.yokkaichi-u.ac.jp/~ikeda/research/picola.html>.

[10] KÁŇA, L.; SCHIMMEL, J.: *Studiová a hudební elektronika*. FEKT VUT Brno, Vlastnosti zvukových signálů, Dostupné z WWW: <www.vutbr.cz>.

[11] KŘUPKA, A. *Implementace digitálního metronomu s převzorkováním signálu v technologii VST.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 36 s. Vedoucí bakalářské práce Ing. Jaromír Mačák.

[12] SMÉKAL, Z.: *Číslicové zpracování signálů* [online] FEKT VUT Brno:, 2009. Dostupné z WWW: <www.vutbr.cz/elearning>.

[13] SMÉKAL, Z.; SYSEL, P. *Číslicové filtry* [online]. FEKT VUT Brno , 2004 . Dostupné z WWW: <www.vutbr.cz/elearning>.

[14] Yamaha corporation, Time-Scale Modification Method And Apparatus For Digital Audio Signals, United States Patent 6 519 567 B1, 2003)

[15] *Studiová a hudební elektronika*: Využití sampleru v nahrávacím studiu. Návod do laboratorního cvičení, Dostupný z WWW: <www.vutbr.cz/elearning/>

[16] ZÖLZER, U.,et al. *DAFX : Digital Audio Effects* Willey, 2002

Seznam zkratek

API	Application Programming Interface, *rozhraní pro programování aplikací*
DFT	Discrete Fourier Transform, *diskrétní Fourierova transformace*
DLL	Dynamic-Link Library, *dynamicky připojovaná knihovna*
FFT	Fast Fourier Transform, *rychlá Fourierova transformace*
GUI	Graphical User Interface, *grafické uživatelské rozhraní*
HTML	HyperText Markup Language, *značkovací jazyk pro hypertext*
IIR	Infinite Impulse Response, *nekonečná impulsní odezva*
MIDI	Musical Instrument Digital Interface, *digitální rozhraní pro hudební nástroje*
MSB	Most Significant Bit, *nejvýznamnější bit*
PCM	Pulse-Code Modulation, *pulzně-kódová modulace*
PICOLA	Pointer Interval Control Overlap and Add, *vzdáleností ukazatelů řízené přičtení přesahu*
SDK	Software Development Kit, *sada pro vývoj softwaru*
VST	Virtual Studio Technology, *technologie virtuálního studia*
XML	Extensible Markup Language, *rozšiřitelný značkovací jazyk*

Význam použitých symbolů

$r[n]$	modulační pilová funkce
$r_t[\tau]$	autokorelační funkce
$d_t[\tau]$	rozdílová funkce
$d'_t[\tau]$	rozdílová funkce normalizovaná kumulativním průměrem
$X(n,k)$	časově proměnné spektrum
$\lvert X(n,k)\rvert$	modul časově proměnného spektra
$\varphi(n,k)$	fáze časově proměnného spektra
$\tilde{X}(n,k)$	časově proměnné spektrum pásmově omezeného signálu
$X^*(n,k)$	komplexně sdružené časově proměnné spektrum
$H_k(e^{j(\Omega)})$	kmitočtová charakteristika diskrétního systému
$*$	operátor lineární konvoluce
$\Delta\varphi(n,k)$	rozdíl fáze dvou krátkodobých spekter signálu
$H(z)$	přenosová funkce diskrétního systému

Obsah přiloženého média

- *Matlab simulations* – adresář obsahuje simulace jednotlivých metod pro posun výšky tónu v prostředí Matlab
  - *Mod delay line* – adresář obsahuje simulace metody s využitím modulace zpožďovcí linky
    - *modDelayLine.m* – simulace metody pro posun výšky tónus využitím zpožďovací linky, výsledné soubory se zpracovanými zvuky se uloží do adresáře *Sounds\Output*
    - *modDelayLinePitchEst* – simulace metody pro posun výšky tónu s využitím zpožďovací linky, metoda je doplněná o automatické řízení délky zpožďovací linky v závislosti na základní periodě zpracovávaného zvuku. Výsledné soubory se zpracovanými zvuky se uloží do adresáře *Sounds\Output*
  - *Picola* – adresář obsahuje simulaci metody PICOLA
    - *picola.m* – skript pro simulaci metody PICOLA, výsledné soubory se zpracovanými zvuky se uloží do adresáře *Sounds\Output*
  - *Phase vocoder* – adresář obsahuje simulace metody s využitím fázového vokodéru, výsledné soubory se zpracovanými zvuky se uloží do adresáře *Sounds\Output*
    - *testPhaseVoc.m* – skript pro simulaci metody s využitím fázového vokodéru, výsledné soubory se zpracovanými zvuky se uloží do adresáře *Sounds\Output*
  - *Sounds* – adresář, kde jsou uloženy testovací zvuky, do podadresáře *Output* se ukládají výstupy jednotlivých simulačních skriptů
- *Sampler* – adresář obsahující projekt sampleru
  - *docSampler* – adresář obsahující dokumentaci vygenerovanou programem Doxygen
  - *juce* – zdrojové kódy knihovny JUCE
  - *SamplerJuce* – adresář obsahující zdrojové kódy projektu softwarového sampleru, který byl vytvořen v MS Visual Studio 2010
  - *vstsdk2.4* – zdrojové kódy VST API
- *SamplerProgram* – adresář obsahující zkompilovaný program sampleru
- *DiplomovaPrace.pdf* – elektronická verze diplomové práce
- *NavodKPouziti.pdf* – návod k použití softwarového sampleru