

Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Tereza Cihelková

Harmonický posun výšky tónu

Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Holan Tomáš, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové systémy

Praha rok 2011

Děkuji svému vedoucímu za trpělivost a ochotu pomoci, firmě Audiffex a jejím spolupracovníkům, zejména inženýru Michalu Trzosovi a panu Luboru Přikrylovi, za odborné konzultace a doktoru Jiřímu Schimmelovi za poskytnutí šablony pro vývoj VST modulu.

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 3.8.2011

Název práce: Harmonický posun výšky tónu

Autor: Bc. Tereza Cihelková

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Tomáš Holan, Ph.D.

Abstrakt: Tato práce se zabývá návrhem a implementací hudebního efektu sloužícího pro harmonický posun frekvence jednohlasého zpěvového signálu. Efekt umožňuje v reálném čase generovat ze vstupního signálu dva hlasy s posunem výšky tónu při zachování formantů. Míru posunu hlasů je možné ovládat pomocí MIDI ovladače. Efekt je implementován jako VST modul ve formě dynamicky linkované knihovny. Tato práce také obsahuje teoretickou část zahrnující úvod do technik DSP (digitálního zpracování signálu).

Klíčová slova: harmonický posun frekvence, VST modul, změna výšky tónu, zachování formantů

Title: Harmonic pitch shifting

Author: Bc. Tereza Cihelková

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D.

Abstract: This thesis describes the design and implementation of musical effect for pitch shifting of monophonic singing signals. The effect can generate two pitch shifted voices from input signal in real-time, while preserving formants. Amount of the shift can be controlled via MIDI controller. The effect is implemented as VST module in the form of dynamic-link library. This work also includes theoretical introduction to related DSP techniques.

Keywords: harmonic pitch shifting, VST module, pitch shifting, formant preservation

Obsah

Předmluva	1
1 Úvod do problematiky	2
1.1 Zvuk jako mechanické vlnění.....	2
1.1.1 Periodické vlnění.....	2
1.1.2 Aperiodické a kvaziperiodické vlnění.....	3
1.1.3 Rezonance	3
1.2 Hudební tóny	3
1.3 Jak funguje lidský sluch	5
1.3.1 Vnímání výšky tónu	5
1.3.2 Vnímání hlasitosti tónu	6
1.4 Jak funguje lidský hlas	7
1.5 Digitální zpracování zvukového signálu	8
1.5.1 Digitalizace	9
1.5.2 Fourierova analýza	11
1.5.3 Frekvenční spektrum.....	15
1.5.4 Spektrální obálka.....	16
1.5.5 Okénkové funkce	17
2 Jak změnit výšku tónu.....	19
2.1 Algoritmy manipulující se signálem v časové doméně.....	19
2.1.1 Změna výšky tónu pomocí změny vzorkovací frekvence.....	20
2.1.2 SOLA	22
2.1.3 PSOLA	25
2.2 Algoritmy manipulující se signálem ve frekvenční doméně.....	27
2.2.1 Fázový vokoder.....	27
3 Zvolené řešení	33
3.1 Algoritmus pro hledání základní frekvence	34
3.1.1 Popis algoritmu YIN	35
3.1.2 O Implementaci.....	39
3.2 Posun výšky.....	49
3.2.1 Lentův algoritmus	49
3.2.2 O Implementaci.....	53
3.2.3 Jak lze pomocí Lentova algoritmu měnit formanty	57

3.3	Implementace řešení jako modulu.....	58
3.3.1	Technologie VST	59
3.3.2	VST šablona	61
3.3.3	Ovládání pomocí MIDI.....	61
3.3.4	Popis projektu	62
3.4	Popis výsledného efektu	65
3.4.1	Ovládání efektu	66
4	Porovnání s existujícími řešeními	70
4.1	Profesionální knihovny a aplikace pro posun výšky hlasu.....	70
4.2	VST efekty pro posun výšky hlasu.....	72
4.2.1	Placené efekty	72
4.2.2	Neplacené efekty	74
4.3	Zhodnocení	75
	Závěr	76
	Seznam použité literatury.....	77
	Seznam použitých knihoven	80

Předmluva

S rostoucí výkonností počítačů bylo umožněno i amatérským nadšencům postavit si doma malé nahrávací studio za rozumnou sumu peněz. Dříve analogové efekty se dnes implementují jako software, jehož výrobní náklady jsou nižší a lépe se šíří. Jedním z těchto případů jsou i efekty pro změnu výšky tónu.

Algoritmy pro posun výšky tónu se vyvíjí již desítky let. Každý z nich funguje dobře jen pro určitý typ zvuků. V kategorii zpracování hudebního signálu v reálném čase při zachování formantů ani drahá profesionální řešení nezvládají danou úlohu bez chyb pro velké změny ve výšce tónu. Jelikož mě práce se zvukem v počítači zajímá již delší dobu, stejně jako zpěv, rozhodla jsem se v rámci diplomové práce vytvořit efekt, který by umožňoval změnu výšky zpívaného tónu v reálném čase a bylo by ho možné používat se standardním nahrávacím software například pro generování doprovodných hlasů.

Tato práce je organizována následujícím způsobem:

- První kapitola je úvodem do pojmů souvisejících se změnou výšky tónu a základů digitálního zpracování zvuku.
- Druhá kapitola se věnuje některým význačným technikám pro změnu výšky tónu.
- Třetí kapitola se zabývá popisem implementace vlastního řešení, založeného na principech představených v prvních dvou kapitolách.
- Ve čtvrté kapitole je výsledný efekt porovnán s některými existujícími řešeními.
- Závěrečná kapitola podá celkové shrnutí práce a zamyslí se nad možností jejího dalšího rozšíření.
- Na přiloženém cd jsou k dispozici nahrávky, na které je z práce odkazováno a také návod pro použití efektu i s popisem instalace s vybranými hudebními editory.

1 Úvod do problematiky

V této kapitole si krátce zopakujeme, co je to zvuk, jak funguje lidský sluch a hlas a přiblížíme si některé základní pojmy a techniky z oblasti digitálního zpracování zvuku. Pokud čtenář v této oblasti již znalosti má, může přejít rovnou ke kapitole 2, která se zabývá technikami pro změnu výšky tónu.

1.1 Zvuk jako mechanické vlnění

Jelikož chceme zpracovávat zvuk, měli bychom si říci, co to vlastně je. **Zvuk** je podélné¹ mechanické **vlnění**. Můžeme si ho představit jako vibraci nebo vlnění molekul vzduchu, popřípadě jiného prostředí, které má dostatečný počet částic na jednotku délky. Pokud by v prostředí nebyl dostatečný počet částic, například ve vakuu, tak se jím žádné mechanické vlnění – tedy ani zvuk – šířit nemůže. Vlnění nebo vibrace jsou vyvolány pohybem nějakého objektu, například reproduktorů, struny, hlasivek, úderem paličky na buben. Jelikož mezi částicemi prostředí existují vazebné síly, tak jakmile je jedna z částic vychýlena, způsobí vychýlení dalších částic v nejbližším okolí. Tím se tvoří oblasti, kde je mnoho částic tlačeno na sebe a oblasti, kde je naopak částic málo. Tyto komprese a uvolnění se šíří kolem zdroje zvuku v kruzích. Zvuková vlna tedy vzniká jako rychle se měnící tlak v prostředí. Velikosti změn v tlaku, které vytváří zvuk, v poměru ke konstantnímu tlaku vzduchu se říká **amplituda**. Čím vyšší amplituda je, tím hlasitější se nám zvuk jeví.

1.1.1 Periodické vlnění

Pokud se pohyb, který vyvolal zvukovou vlnu, identicky opakuje v pravidelných intervalech, vzniká takzvané periodické vlnění. Závislost intenzity takového vlnění na čase se dá popsat periodickou funkcí. V takových případech můžeme tvrdit, že zvuková vlna má určitou **periodu**. Počtu cyklů, které proběhnou za jednotku času, říkáme **frekvence** zvuku. Pro periodu T a frekvenci f platí, $f = 1/T$. Jednotkou frekvence je Hertz, značí se Hz . Čím pomaleji zdroj zvuku kmitá, tím delší má zvuk periodu a tím nižší má frekvenci. Naopak kmitá-li zdroj zvuku rychle, je perioda zvuku krátká a frekvence vysoká.

¹ V pevných látkách se zvuk může šířit i jako příčné mechanické vlnění.

Někdy se definice zvuku omezuje pouze na podélné mechanické vlnění s frekvencí z rozsahu 16 Hz až 20 000 Hz. Přibližně v tomto rozsahu je zvuk slyšitelný pro zdravého člověka. Zvuku s frekvencí pod 16 Hz se pak říká **infrazvuk** (slyší ho například sloni) a zvuku s frekvencí nad 20 000 Hz se říká **ultrazvuk** (vnímají ho například netopýři nebo delfinovití).

1.1.2 Aperiodické a kvaziperiodické vlnění

Většina zvuků v přírodě je aperiodických. To znamená, že pohyb, který zvukovou vlnu vyvolal, se v čase měnil. Takový zvuk vnímáme například jako hluk nebo šum.

Zajímavé je, že pokud se podíváme na zvuky produkované hudebními nástroji² nebo hlasivkami, tak zjistíme, že také nejsou dokonale periodické, přesto je ale jako hluk nebo šum nevnímáme. Je to dáno tím, že na krátkých časových úsecích (pár desítek milisekund) se jeví jako téměř dokonale periodické. To stačí k tomu, abychom u nich byli schopni rozlišit frekvenci a vnímat je jako hudební tóny. Takovému druhu vlnění, které je na krátkých časových úsecích téměř dokonale periodické, se říká **kvaziperiodické**.

1.1.3 Rezonance

Objekt, který může vibrovat s určitou frekvencí, když je rozpořhybován, také začne vibrovat, zasáhne-li ho zvuková vlna o této frekvenci. Například někteří operní pěvci byli známí tím, že uměli hlasem rozbít sklo. Když mají vibrace zvuku nesené částicemi prostředí stejnou frekvenci jako je přirozená³ frekvence objektu, do kterého naráží, tak vybudí objekt k oscilaci s vysokou amplitudou, stejně jako můžeme rozhoupat houpačku. A právě tento jev, kdy je jeden objekt rozvibrován poté, co ho zasáhne zvuková vlna o určité frekvenci, se v akustice nazývá rezonance. Říkáme, že objekt rezonuje.

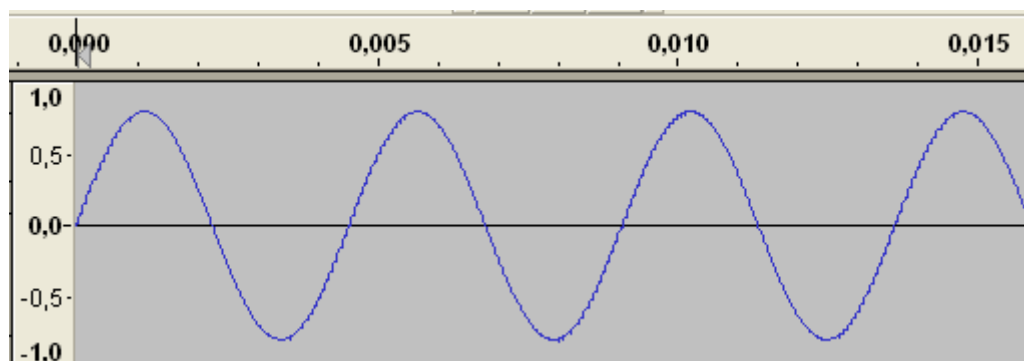
1.2 Hudební tóny

Hudební tóny jsou podmíněné periodickým nebo kvaziperiodickým vlněním. **Výška tónu** je dána jeho frekvencí. Tóny se dále dělí na **jednoduché** a **složené**.

² S výjimkou perkusivních nástrojů.

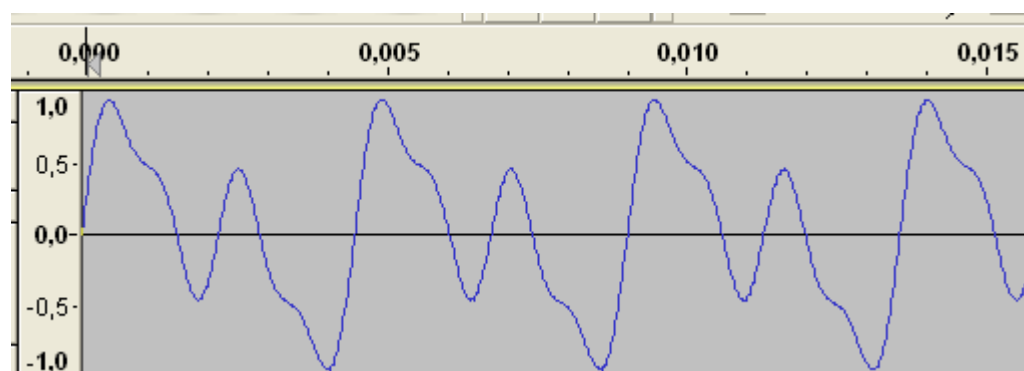
³ Někdy se také říká „vlastní“.

Jednoduchý tón má harmonický průběh a jeho grafem závislosti intenzity na čase je křivka podobná grafu sinové funkce s fázovým posunem.



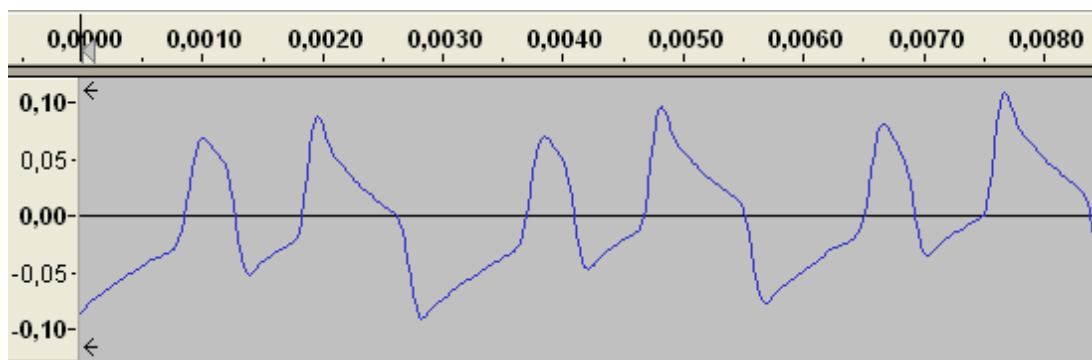
Obrázek 1.1. Příklad grafu reprezentujícího jednoduchý tón o frekvenci 220 Hz.

Jednoduché tóny jsou však vzácné. Častěji se setkáváme s tónem složeným z více tónů jednoduchých. Jeho průběh je také periodický, ale složitější, jak ilustruje obrázek 1.2.

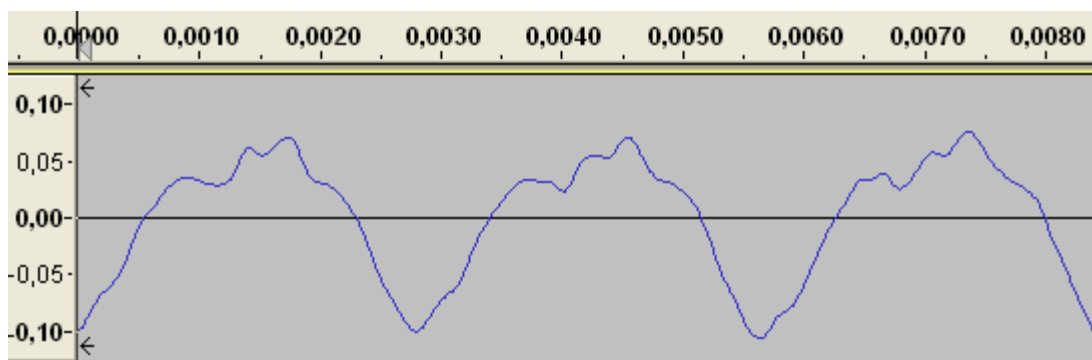


Obrázek 1.2. Příklad grafu reprezentujícího složený tón o základní frekvenci 220 Hz a vyšších harmonických složkách 440 Hz a 880 Hz.

Složené tóny obsahují kromě **základní frekvence** – která určuje vnímanou výšku tónu – ještě vyšší **harmonické složky**. Vyšší harmonické složky jsou celočíselné násobky základní frekvence, která je někdy také označována jako první harmonická složka. Poměr mezi těmito harmonickými složkami pak určuje vnímanou **barvu tónu**. Díky barvě tónu je člověk schopný odlišit od sebe tóny o stejné intenzitě a výšce hrané různými nástroji. Na obrázcích 1.3 a 1.4 jsou příklady průběhu tónu F3 hraného různými nástroji.



Obrázek 1.3. Průběh tónu F3 hraného na elektrickou kytaru.



Obrázek 1.4. Průběh tónu F3 hraného na klavír.

1.3 Jak funguje lidský sluch

Lidské ucho vnímá zvukové vlny v rozsahu frekvencí 16–20 000 Hz a nejcitlivější je na tóny v oblasti okolo 1 000–3 000 Hz (mluvené slovo).

Zvukové vlny přechází přes vnější zvukovod ucha a naráží na bubínek, který rozkmitají. Vibrace se dále přenáší skrze střední ucho díky třem malým, navzájem se dotýkajícím kůstkám, kterými jsou: kladívko, kovadlinka a třmínek. Tyto kůstky rovněž zesilují amplitudu vibrací. Poslední ze sluchových kůstek – třmínek – je předává skrze oválné okénko do hlemýžďe. Hlemýžď má tvar šnečí ulity a různé jeho části rezonují s různými frekvencemi. Když nějaká část hlemýžďe začne rezonovat, tak nerv příslušný dané oblasti přenesení signál do mozku. My jej vnímáme jako určitý tón. Většina zvuků ale rezonuje ve více částech hlemýžďe najednou. Takový zvuk vnímáme jako složený (obsahuje vyšší harmonické složky) nebo jako akord (souzvuk různých tónů).

1.3.1 Vnímání výšky tónu

Výška tónu závisí na základní frekvenci tónu. Přitom frekvence tónu je objektivně a jednoznačně změřitelná veličina, zatímco výška tónu je do jisté míry

subjektivní vjem. Vztah mezi základní frekvencí a výškou tónu je přibližně logaritmický. Například rozdíl ve výšce mezi tóny o frekvencích 100 Hz a 200 Hz je pro člověka stejně velký jako rozdíl ve výšce mezi tóny 1 000 Hz a 2 000 Hz. To, na čem skutečně záleží, není rozdíl mezi frekvencemi, ale jejich poměr. Poměr frekvencí, který je rovný 2, nazýváme oktáva. Říkáme, že tón o frekvenci 1 000 Hz je o oktávu nižší než tón o frekvenci 2 000 Hz a naopak tón o frekvenci 2 000 Hz je o oktávu vyšší než tón o frekvenci 1 000 Hz. Jelikož v chromatické stupnici má oktáva 12 půltónů, odpovídá posun frekvence o půl tónu nahoru vynásobení stávající frekvence $\sqrt[12]{2}$ a posun o půl tónu dolů vydělení stávající frekvence $\sqrt[12]{2}$.

Zajímavostí je, že i v případě, kdy základní frekvence ve složeném tónu není přítomna, mozek je dle [20] schopný si ji, za určitých okolností, „domyslet“ a tedy zvuk vnímat, jako by tam byla.

Naopak u velmi silných vyšších harmonických složek, může dojít k tomu, že mozek určí výšku tónu na základě jiné frekvence než frekvence základní. Přesto se v literatuře, která se zabývá zpracováním zvuku, často pojmy výška tónu a základní frekvence zaměňují.

Výška tónu také, do určité míry, závisí dle [20] na akustickém tlaku, zejména u frekvencí nižších než 1 000 Hz a vyšších než 2 000 Hz. Vnímaná výška nižších tónů se snižuje zároveň se zvyšujícím se akustickým tlakem a výška vyšších tónů se naopak zvyšuje se zvyšujícím se akustickým tlakem.

Nejmenší, uchem rozlišitelný, rozdíl mezi frekvencemi závisí dle [20] na velikosti frekvencí. Například v oktávě mezi 1000–2 000 Hz je roven zhruba 3,6 Hz. V oktávě mezi 62–125 Hz je už menší, asi 2 Hz. Nejmenší slyšitelný rozdíl byl stanoven testováním – skupině subjektů byly přehrány dva tóny rychle za sebou a testovaní měli říci, zda mezi nimi slyší rozdíl. Přitom v případě, že jsou tóny zahrány zároveň, je rozlišovací schopnost lepší. Lidský sluch dokáže rozeznat asi 1 400 zvuků v závislosti na jejich frekvenci.

1.3.2 Vnímání hlasitosti tónu

Vnímaná hlasitost tónu závisí na velikosti tlakových změn, které zvuková vlna na bubínek vyvíjí. Nicméně hlasitost je vjem subjektivní a závisí na citlivosti

sluchu jedince. Také platí, že tóny, o různých frekvencích a stejné velikosti tlakových změn, se nám jeví jako různě hlasité.

Aby bylo možno se zvuky objektivně pracovat, byl zaveden pojem úrovně intenzity zvuku. Pokusy bylo zjištěno, že jednoduchý tón o frekvenci 1 000 Hz je průměrným člověkem slyšitelný, pokud vyvolá akustický tlak o velikosti:

$$p_0 = 2 * 10^{-5} Pa \quad (1)$$

Jako **akustický tlak** označujeme rozdíl mezi okamžitým tlakem způsobeným přítomností zvuku a atmosférickým tlakem.

Dosazením akustického tlaku p zkoumaného zvuku a (1) do (2) získáme hladinu akustického tlaku L_p , jejíž jednotka je označována jako **bel**. Přesný

$$L_p = 10 \log_{10}(p/p_0) \quad (2)$$

Jedním z důvodů k logaritmizaci je skutečnost, že ucho má přibližně logaritmickou odezvu. Běžně se ovšem pracuje s desetkrát menší jednotkou nazývanou **decibel**, značí se **dB**.

Práh slyšitelnosti je zhruba 0 dB - záleží na frekvenci a na citlivosti sluchu jedince. Práh bolestivosti je asi 120 dB [2].

1.4 Jak funguje lidský hlas

V hrtanu jsou dvě pružné blány, které se nazývají hlasivky. Často se mylně uvádí, že hlasivky jsou svaly. To není pravda. Pohyb hlasivek je zajišťován sadou svalů a chrupavek umístěných v hrtanu kolem hlasivek. Vzduch, při cestě do a z plic, prochází právě mezi těmito dvěma blánami. Pokud jsou od sebe hlasivky odtažené a uvolněné, pak žádný tón nevzniká. Ve skutečnosti se od sebe oddalují jen na jednom konci a na druhém jsou stále spojené. Tvarem tedy připomínají písmeno „V“. Pokud se k sobě hlasivky dostatečně přiblíží a napnou, začnou při výdechu (nebo i nádechu) kmitat. Tím vzniká tón, jehož základní frekvence je daná délkou a napnutím hlasivek. Čím víc jsou hlasivky okolním svalstvem napínány, tím vyšší frekvence vzniká. Ve skutečnosti je tento proces složitější, ale toto přiblížení je dostačující. Ženy mívají hlasivky o něco kratší než muži, a proto mají výše položené hlasy.

Rozsah lidského hlasu, stejně jako elasticita hlasivek, není neomezený. Průměrný rozsah hlasu je 80–1100 Hz, což zhruba odpovídá e2–c6. S přibývajícím věkem rozsah hlasu kvůli kostnatění chrupavek a ochabování svalstva klesá.

Dle výzkumu, dokumentovaného ve [3], je doba potřebná ke změně tónu, o 4 tóny a více, přibližně 55 ms.

Cestou ven z těla prochází vzduch třemi rezonančními dutinami – hrtanovou, ústní a nosní. Kromě ústní dutiny mají všechny dutiny víceméně neměnnou vlastní frekvenci. Vlastním frekvencím těchto rezonančních dutin se říká formanty. Formant ústní dutiny je možné měnit pozicí a tvarem jazyka, rtů a zubů. To nám umožňuje rozlišovat jednotlivá písmena.

Pokud je základní frekvence tónu, produkovaného hlasivkami, vyšší než formanty, nedochází prakticky k žádné rezonanci. Proto je vysoko zpívajícím operním pěvcům špatně rozumět.

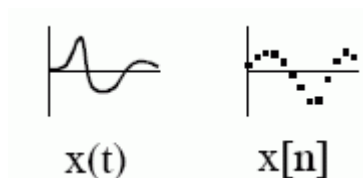
1.5 Digitální zpracování zvukového signálu

Co je to **signál**? Definovat pojem signál přesně není jednoduché. Nejobecnější tvrzení by bylo, že signál je tok informací. V této práci budeme vycházet z definice volně převzaté z [21], která říká, že:

Signál je jakýkoliv fyzikální úkaz, který se dá modelovat jako funkce jedné, či více proměnných s reálnými nebo komplexními hodnotami.

Pokud funkce závisí na jediné proměnné, říkáme, že je signál jednorozměrný. Mezi jednorozměrné signály patří i zvuk. Pokud funkce závisí na více proměnných, říkáme, že je signál vícerozměrný. Například obraz je dvourozměrný signál. V této práci se budeme zabývat pouze zvukovými – jednorozměrnými – signály.

Signálu, který je spojitý ve všech rozměrech, říkáme **analogový** a zapisujeme ho s kulatými závorkami například $x(t)$. Signálu, který je diskrétní ve všech rozměrech říkáme **digitální** a zapisujeme ho s hranatými závorkami například $x[n]$.



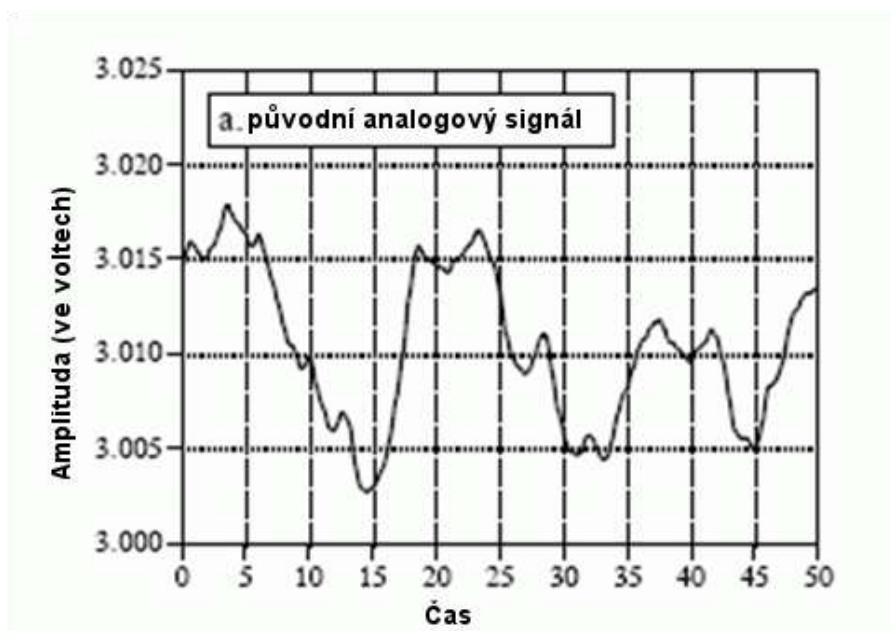
Obrázek 1.5. Příklad analogového a diskrétního signálu a jejich značení. Převzato z [1].

Digitální zpracování signálu – ve většině odborné literatury označované anglickou zkratkou **DSP** (Digital Signal Processing) – je souhrn algoritmů, vzorců a technik používaných k manipulaci s digitálními signály [1]. Důvodem pro manipulaci s takovými signály může být například snaha zlepšit kvalitu obrazu, rozpoznat a generovat řeč nebo zkomprimovat data pro uložení a přenos.

1.5.1 Digitalizace

Digitalizace je proces, který ze spojitého, analogového, signálu vytvoří signál digitální, se kterým je pak počítač schopný pracovat. Výsledkem procesu digitalizace je reprezentace původního, analogového, signálu jako sekvence čísel. Digitalizace probíhá v zařízeních, kterým se říká **analogově digitální převodník** (zkratka A/D, v angličtině ADC). Opačný převod, z digitálního signálu na analogový, zajišťuje **digitálně analogový převodník** (zkratka D/A, v angličtině DAC).

Digitalizace se skládá ze dvou částí – vzorkování a kvantování.



Obrázek 1.6. Analogový signál. Převzato z [1].

Pokud se podíváme na obrázek 1.6, který reprezentuje graf analogového signálu, máme na ose x čas a na ose y okamžitou amplitudu. Během vzorkování dochází k převodu nezávislé proměnné (v tomto případě času) ze spojité na diskrétní. Časová vzdálenost mezi dvěma po sobě jdoucími prvky se nazývá **vzorkovací interval**⁴ T a převrácená hodnota se nazývá **vzorkovací frekvence** $f_s = 1/T$. Vzorkovací frekvence udává počet vzorků za vteřinu v Hz.

Každý vzorek reprezentuje amplitudu, kterou signál měl v čase daném indexem vzorku. Nicméně tato hodnota může být v číslicových počítačích uložena jen s omezenou přesností. Pokud bude pro uložení hodnoty každého vzorku vyhrazeno 12 bitů, můžeme například ukládat celá čísla od 0 do 4 095. To znamená, že hodnota 2 560 mohla původně být amplitudou o velikosti 2 560,01 stejně tak jako 2 560,00. Tomuto procesu, kdy je závislá proměnná (zde amplituda) převedená ze spojité na diskrétní, říkáme **kvantování**. Kvantování tedy určitým způsobem degraduje signál [1]. Každý ze vzorků digitalizovaného signálu může mít chybu maximálně rovnou $\pm 1/2$ LSB (Least Significant Bit, česky Nejméně významný bit). LSB reprezentuje vzdálenost mezi dvěma po sobě jdoucími kvantovacími úrovněmi. Ve většině případů kvantování vede k přidání určitého množství šumu k signálu. Čím více bitů máme k dispozici pro uložení hodnot vzorků, tím vyšší úroveň přesnosti dosáhneme.

Nyquistův-Shannonův teorém dle [1,5] říká:

V čase spojité signál $x(t)$, který obsahuje jen frekvence menší než f_{max} , může být plně rekonstruován ze své navzorkované verze $x[n] = x(nT)$, pokud pro vzorkovací frekvenci $f_s = 1/T$ platí, že $f_s \geq 2f_{max}$.

Polovinu vzorkovací frekvence budeme označovat jako **Nyquistovu frekvenci** a dvojnásobek šíře frekvenčního pásma frekvenčně limitovaného signálu budeme nazývat **Nyquistova vzorkovací frekvence**⁵.

⁴ Nebo také vzorkovací perioda.

⁵ Pojmy Nyquistova frekvence a Nyquistova vzorkovací frekvence nejsou standardizované, a tudíž je různí autoři používají v mírně odlišném významu [1]. Proto se musí vždy definovat.

Pokud by vstupní signál obsahoval i frekvence vyšší než Nyquistova frekvence, došlo by k jevu, kterému se říká **aliasing** [1]. Při aliasingu jsou frekvence vyšší než polovina vzorkovací frekvence zaměněny za frekvence mezi 0 Hz a polovinou vzorkovací frekvence. Před vstupem signálu do A/D je nutné tyto vysoké frekvence odstranit pomocí dolní propusti (anglicky low-pass filter) [1].

V teorii tedy digitalizací signálu, který neobsahuje frekvence vyšší než je Nyquistova a při zvolení dostatečného množství bitů pro kvantování, signál téměř nepoškodíme. V praxi bývá problém zejména s implementací vhodné dolní propusti, jelikož neexistuje analogový filtr, který by uměl ze signálu odebrat zvolené frekvence, aniž by na signálu vytvořil nějaké artefakty [1].

1.5.2 Fourierova analýza

Fourierova analýza je rodina matematických technik, které jsou všechny založené na rozkladu signálu na sinové a kosinové funkce, hromadě někdy označované jako sinusoidy. Sinusoida⁶ je přitom jakákoliv funkce mající formu $x(t) = A \sin(\omega t + \varphi)$, kde A je amplituda, φ je počáteční fáze, $\omega = 2\pi f$ a f je frekvence. Například i jednoduchý tón je možné matematicky popsat jako sinusoidu.

Podle terminologie převzaté z [1] existují čtyři kategorie Fourierovy analýzy lišící se podle toho, zda je signál, se kterým pracují, spojitý nebo diskrétní a zda je periodický či aperiodický.

Signál	Aperiodický	Periodický
Spojitý	Fourierova Transformace	Fourierova řada
Diskrétní	V čase diskrétní Fourierova transformace	Diskrétní Fourierova Transformace

Tabulka 1.1. Kategorie Fourierovy analýzy

⁶ V některých publikacích je jako sinusoida označován čistě graf funkce sinus a nikoliv funkce.

Všechny druhy Fourierovy analýzy pracují s nekonečným signálem. Abychom mohli použít Fourierovu analýzu jen na úsek signálu, tak s ním buď pracujeme jako by byl z obou stran do nekonečna doplněn nulami, tudíž by nám vznikl signál aperiodický, nebo jako by se zpracovávaný úsek do nekonečna opakoval, v tomto případě by vznikl signál periodický.

K syntéze⁷ aperiodického signálu je zapotřebí nekonečné množství sinusoid. V počítači můžeme pracovat jen s konečnou informací. Proto se v DSP používá pouze Diskrétní Fourierova Transformace (DFT). Pokud budeme v následujících kapitolách referovat Fourierovu analýzu, vždy budeme mít na mysli DFT.

Princip Diskrétní Fourierovy Transformace

Jean-Baptiste Joseph Fourier přišel na to, že jakkoliv komplexní signál se dá reprezentovat jako suma sinusoid. Pro diskrétní periodický signál má tato suma konečný počet členů.

DFT zjišťuje přítomnost referenční frekvence v diskrétním signálu tím, že vzorky signálu vynásobí hodnotami navzorkované sinusoidy, o referenční frekvenci a jednotkové amplitudě, a výsledné součiny sečte. Pokud signál danou frekvenční složku obsahuje, bude výsledek vysoký, pokud ji neobsahuje, bude blízky nule. Principiálně je to podobné rezonanci. Referenční frekvence je tu v roli rezonátoru, který je excitován vstupním signálem.

Pokud by se dalo zajistit, že všechny frekvenční složky signálu budou začínat s nulovou počáteční fází, stačilo by, jako referenční funkci, použít sinus. Zapsat by se to dalo jako:

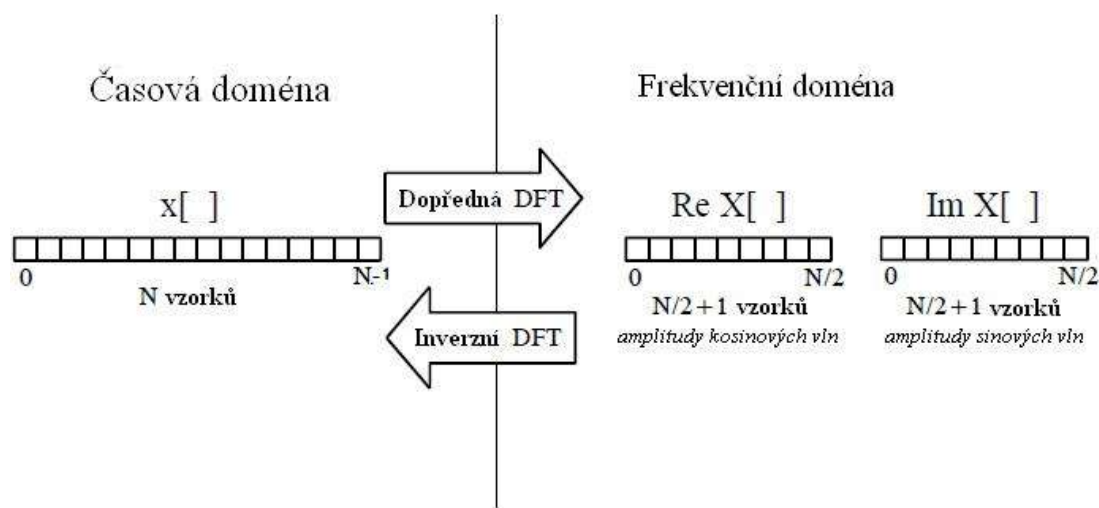
```
int bin = (testovanaFrekvence * pocetVzorku) / vzorkovaciFrekvence;
for (int i = 0; i < pocetVzorku; i++) {
    koeficientAmplitudy += sin(bin * 2 * PI * i / pocetVzorku) *
        signal[i];
}
```

Kde *signál* je pole obsahující vzorky testovaného signálu.

⁷ Syntéza je proces inverzní k analýze a bude podrobněji popsána v dalších částech kapitoly.

Jelikož ale nevíme, jakou počáteční fázi jednotlivé frekvenční složky signálu mají, používá DFT jako referenční jak sinovou tak kosinovou funkci. Ze získaných hodnot sum součinů pro sinus a kosinus dané frekvence lze pak dopočítat skutečnou amplitudu frekvenční složky i její fázi.

Reálná DFT je typ DFT, který k reprezentaci vstupních a výstupních signálů používá reálná čísla. DFT mění vstupních N prvků na dvakrát $N/2+1$ prvků [1]. Prvky na vstupu tvoří signál, který chceme rozložit a prvky na výstupu reprezentují amplitudy sinových a kosinových vln určitých frekvencí. O vstupním signálu říkáme, že leží v časové doméně, to proto, že vstupem jsou většinou vzorky odebírané v pravidelných časových intervalech. Frekvenční doménu zase používáme pro popsání amplitud sinových a kosinových vln.



Obrázek 1.7. Naznačený převod mezi časovou a frekvenční doménou, převzato z [1]

Frekvenční doména obsahuje stejné informace jako časová, jen v jiné formě. Pokud známe jednu doménu, můžeme vypočítat druhou. Proces rozkladu vstupního signálu se nazývá dekompozice, analýza, dopředná DFT nebo jen DFT. Pokud naopak známe frekvenční doménu a chceme z ní vypočítat původní signál, tak mluvíme o syntéze nebo inverzní DFT. Obojí, analýza i syntéza, může být reprezentována formou rovnic v počítačových algoritmech.

Počet vzorků v časové doméně se většinou značí N . N může být jakékoliv přirozené kladné číslo, ale většinou se volí jako mocnina dvojky - např. 128, 512,

1024. Jsou pro to dva důvody: digitální data se ukládají za pomoci binárního adresování a nejefektivnější algoritmus výpočtu DFT, Rychlá Fourierova Transformace (anglicky Fast Fourier Transformation FFT) většinou operuje s N , které je mocninou dvou. Typicky je potřeba, aby N bylo mezi 32 a 4 096.

Rovnice pro výpočet syntézy⁸ dle [1]:

$$x[n] = \sum_{k=0}^{N/2} \text{Re}X[k] \cos(2\pi kn/N) - \text{Im}X[k] \sin(2\pi kn/N)$$

Rovnice pro výpočet analýzy dle [1]:

$$\text{Re}X[k] = \frac{2}{N} \sum_{n=0}^{N-1} x[n] \cos(2\pi kn/N)$$

$$\text{Im}X[k] = \frac{-2}{N} \sum_{n=0}^{N-1} x[n] \sin(2\pi kn/N)$$

Časová doména:

$x[n]$ je diskretní, periodický signál.

n jde od 0 k $N - 1$, kde N je počet prvků $x[n]$.

Frekvenční doména:

$\text{Re}X[k]$ je diskretní, periodický signál.

$\text{Im}X[k]$ je diskretní, periodický signál.

k jde od 0 k $N/2$, kde N je počet prvků $x[n]$.

[1].

⁸ Před použitím rovnice pro syntézu je potřeba hodnoty $\text{Re}X[0]$ a $\text{Re}X[N/2]$ vydělit dvěma

Z dat frekvenční domény dopočítáme frekvenci v Hz pro index k a vzorkovací frekvenci f_s jako:

$$f_k = k * \frac{f_s}{N} \quad (3)$$

Magnitudu v dB dle [1,19] jako:

$$magnituda_k = 20 * \log_{10}(2 * \sqrt{ReX[k]^2 + ImX[k]^2}/N)$$

Fázi v radiánech dle [1] jako:

$$phase_k = \arctan(ImX[k]/ReX[k])$$

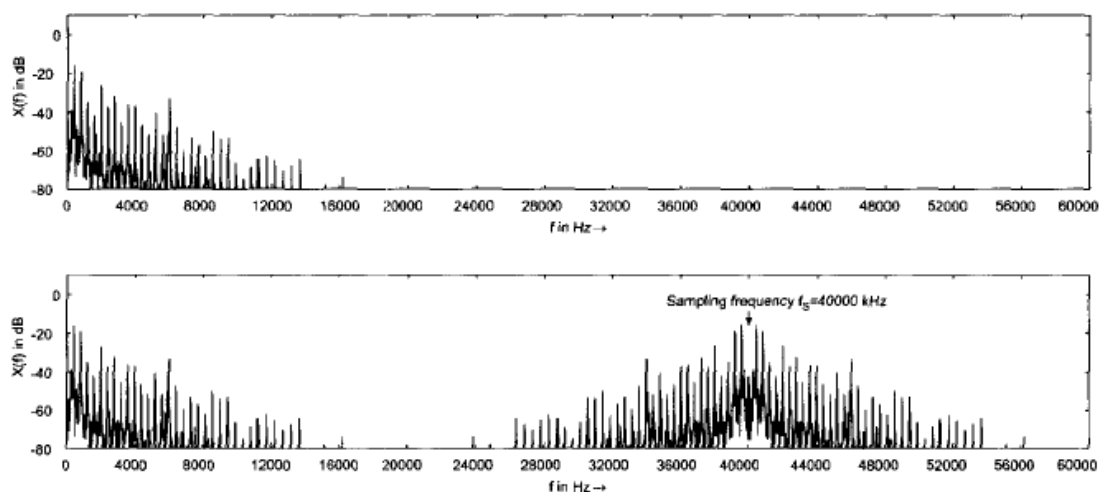
Z rovnice (3) je vidět, že frekvence pro jednotlivé indexy jsou od sebe vzdáleny o $\frac{f_s}{N}$. Tyto fixní frekvence ovšem nemusí přesně odpovídat frekvencím přítomným v signálu. Pokud leží skutečná frekvence mezi dvěma fixními, rozprostře se její energie na okolní fixní frekvence. Z DFT tedy nelze vyčíst rovnou přesné hodnoty harmonických složek signálu.

Dále je z rovnice (3) vidět, že frekvenční rozlišení DFT záleží na vzorkovací frekvenci a počtu vzorků signálu.

Pro efektivní výpočet DFT existují takzvané Fast Fourier Transform, zkráceně FFT, algoritmy. Nejznámějším algoritmem FFT často používaným pro výpočty na počítači je Cooley-Tukey [4], který má složitost $O(N * \log_2 N)$. Existuje několik dalších modifikací FFT. Vážnější zájemci o danou problematiku mohou nalézt další informace v [1] a [6].

1.5.3 Frekvenční spektrum

Frekvenční spektrum signálu popisuje zastoupení jednotlivých frekvencí v signálu.



Obrázek 1.8. Ukázka frekvenčního spektra analogového a digitalizovaného signálu. Na ose x jsou frekvence v Hz, na ose y jsou magnitudy v dB. Převzato ze [6].

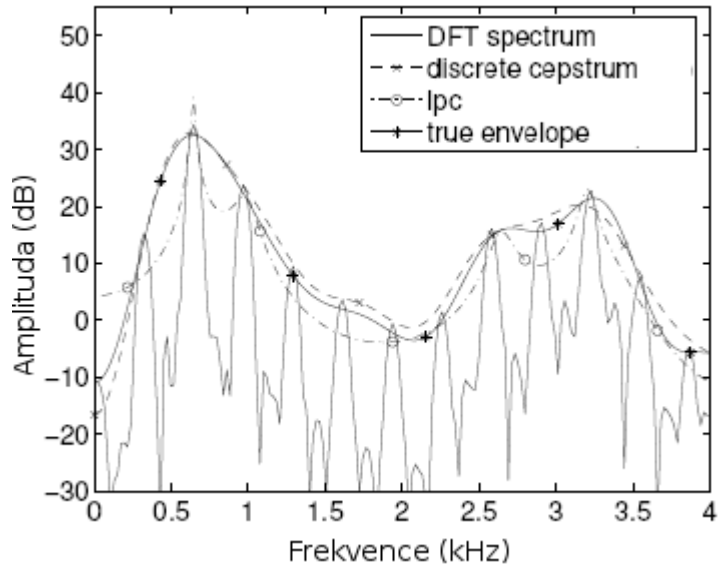
Na obrázku 1.8 je ukázka frekvenčního spektra pro analogový signál a následně pro ten samý signál navzorkovaný vzorkovací frekvencí 40 000 Hz. Zrcadlový obraz části spektra od 0 do 20 000 Hz kolem poloviny vzorkovací frekvence a opakování této části a části od 0 do 20 000 Hz na násobcích vzorkovací frekvence jsou důsledkem samotného vzorkování. Detailnější vysvětlení lze nalézt v [1] a [6].

1.5.4 Spektrální obálka

Máme-li frekvenční spektrum krátkého úseku například jednohlasého zpěvového signálu, tak nejvyšší budou magnitudy frekvencí blízkých základní frekvenci a vyšším harmonickým složkám zpívaného tónu. Spektrální obálka je potom křivka, která prochází těmito lokálními maximy. Existuje několik způsobů, jak spektrální obálku spočítat, a proto pro stejné spektrum může existovat několik různých, ale do značné míry podobných, spektrálních obálek.

Když měníme výšku zvuku a nechceme změnit formanty, tak je potřeba, aby zůstal tvar spektrální obálky pro posunutý zvuk stejný, jako pro zvuk původní. Toho různé algoritmy dosahují různými způsoby, jak uvidíme v kapitole 2.

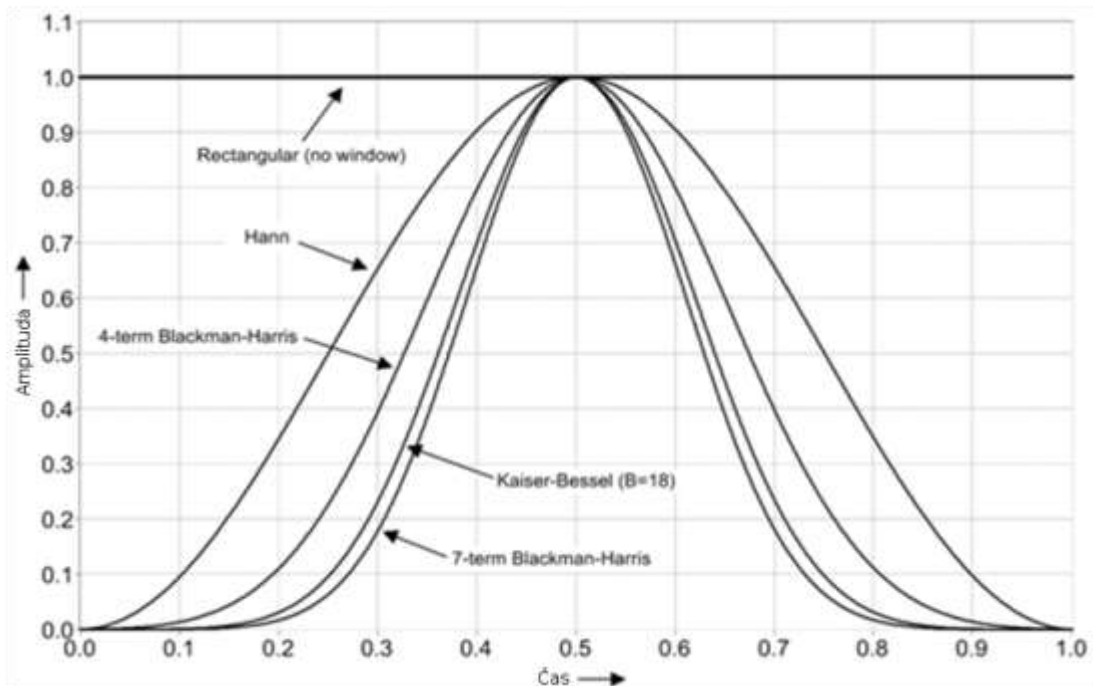
Pro detailnější popis spektrální obálky a metody výpočtu spektrální obálky doporučuji [11] a [12].



Obrázek 1.9. Příklady křivek spektrální obálky signálu vypočítané čtyřmi různými metodami. Převzato z [12].

1.5.5 Okénkové funkce

V DSP označuje pojem „okénková funkce“ většinou funkci, která má nenulové hodnoty jen pro nějaký zvolený interval. Okénkové funkce se většinou používají pro výběr části signálu. Pokud je vstupní signál vynásoben okénkovou funkcí, bude výsledný signál nulový všude, kromě zvoleného intervalu – zbude tedy „okénko“ signálu. Často se okénkové funkce také používají k plynulému utlumení či zhlazení části signálu. Některé z těchto funkcí lze vidět na obrázku 1.10.



Obrázek 1.10. Příklady okénkových funkcí. Převzato z [10].

Jednou z okénkových funkcí zobrazených na obrázku 1.10 je Hannova funkce⁹, která bude v následujících kapitolách používána u několika algoritmů. Hannova funkce je dána jako:

$$Hann(n) = 0,5 * \left(1 - \cos\left(\frac{2\pi n}{N-1}\right) \right)$$

Kde N je délka okna ve vzorcích.

⁹ Bývá označována také jako Hannovo okno.

2 Jak změnit výšku tónu

Jak bylo řečeno v kapitole 1.3, „Jak funguje lidský sluch“, výška tónu je komplexní subjektivní vjem, který není jednoznačně měřitelný. Změnou výšky tónu v DSP většinou myslíme změnu základní frekvence tónu a ostatních frekvenčních složek, při zachování poměrů mezi nimi. To znamená, že u složených tónů nemůžeme změnit pouze základní frekvenci tónu a ostatní frekvenční složky nechat beze změny. Stejně tak nemůžeme vzít jednotlivé složky frekvenčního spektra tónu a posunout je všechny o stejný počet hertzů.

Mějme například složený tón o základní frekvenci 220 Hz s vyššími harmonickými složkami 440 Hz a 880 Hz a chtějme ho posunout o čistou kvartu¹⁰ nahoru. Nestačí jen změnit základní frekvenci na 293,66 Hz¹¹ a ostatní složky ponechat stejné, jelikož by se tím změnil poměr mezi jednotlivými harmonickými složkami. Podobně pokud bychom ke každé složce přičetli 73,66 Hz, posunuli bychom sice základní frekvenci na správnou hodnotu, ale opět bychom změnili poměr mezi jednotlivými složkami. Výsledkem by pak byl disharmonický, kovový zvuk. Takovýto postup by fungoval pouze pro jednoduchý tón.

Správným řešením by bylo vynásobit každou frekvenční složku, přítomnou ve zvuku, hodnotou $\sqrt[12]{2^5}$. V následujících kapitolách si ukážeme některé metody pro změnu výšky tónu a vysvětlíme si, proč to není tak jednoduché, jak by se zdálo.

Existují dva základní přístupy k manipulaci s výškou nebo délkou zvuku, lišící se tím, v jaké doméně se zvukovým signálem pracují.

2.1 Algoritmy manipulující se signálem v časové doméně

Výhodou zpracování signálu v časové doméně je to, že implementace je vcelku přímočará a zvuková data jsou měněna ve stejném formátu, v jakém jsou nahrávána a přehrávána. Nevýhodou zpracování v časové doméně je tendence ke vzniku artefaktů připomínajících ozvěnu [22]. Důvod vzniku těchto artefaktů si vysvětlíme u popisu jednotlivých algoritmů.

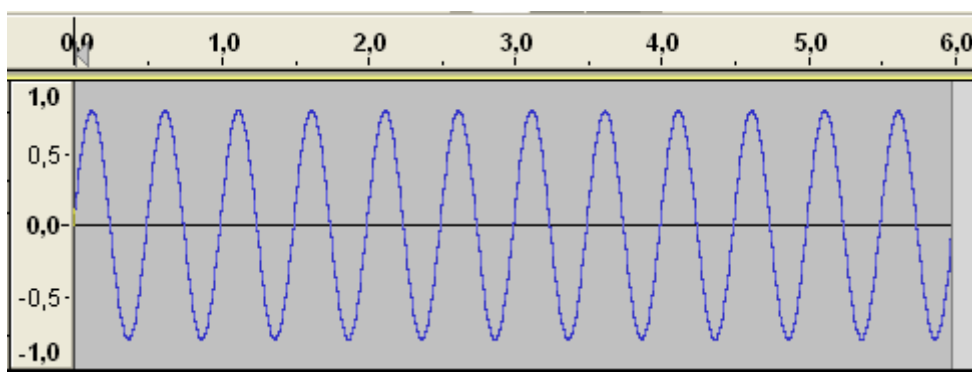
¹⁰ Interval velikosti 5 půltónů.

¹¹ Odpovídá vynásobení původní frekvence hodnotou $\sqrt[12]{2^5}$.

2.1.1 Změna výšky tónu pomocí změny vzorkovací frekvence

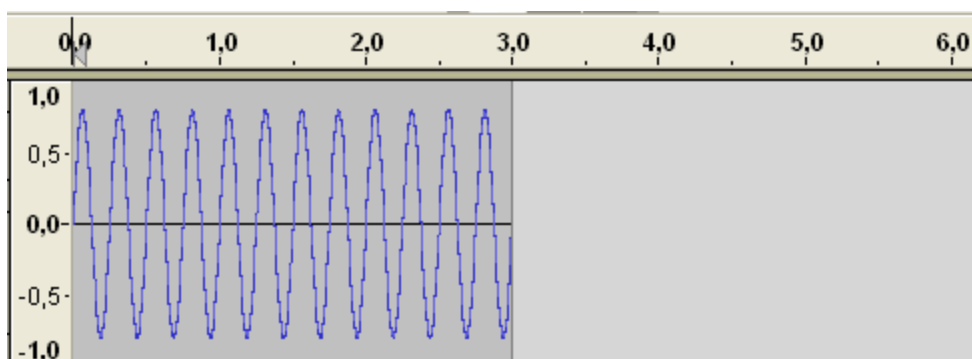
Jako mnoho metod DSP byla i tato inspirována procesy z analogového světa. Magnetofony umožňují přehrávání o různých rychlostech a tím je možné docílit změny výšky tónu pomocí změny rychlosti přehrávání. Budeme-li přehrávat kazetu rychleji, výška tónu stoupne a délka záznamu se zkrátí. Budeme-li přehrávat kazetu pomaleji, výška tónu klesne a délka záznamu se prodlouží. Co se tedy při původní rychlosti stalo během časové periody T , se nyní odehraje za T/v , kde v je relativní rychlost přehrávání.

Při původní rychlosti trvá přehrávání nahrávky zobrazené na obrázku 2.1 6 sekund.



Obrázek 2.1. Signál trvající 6 s. Pouze ilustrační, takto nízká frekvence by nebyla uchem slyšitelná, tudíž se dle definice nejedná o zvuk.

Nyní tu samou nahrávku přehrajme za 3 sekundy relativní rychlostí $v = 2$.



Obrázek 2.2. Ilustruje zvýšení frekvence signálu při zkrácení doby přehrávání na 3 s.

Tím, že zvýšíme rychlost přehrávání na dvojnásobek, zároveň dvojnásobně zvýšíme frekvenci a dostaneme tak o oktávu vyšší tón. Toto platí jak pro jednoduché signály, jako v našem příkladě, tak pro signály daleko složitější, jelikož změna rychlosti přehrávání z v_{puv} na v_{nova} odpovídá vynásobení každé frekvenční složky hodnotou $v = v_{nova}/v_{puv}$ a tudíž jsou zachovány poměry mezi jednotlivými složkami.

Jak takového efektu docílit s digitalizovaným signálem? Předpokládejme nyní, že nemáme možnost měnit frekvenci ADC ani DAC. Pokud má být signál přehrán¹² frekvencí například 44 kHz a my chceme, aby se jeho délka zkrátila na polovinu¹³, musíme počet vzorků v signálu o polovinu zmenšit. Můžeme si to představit tak, že každý druhý vzorek ze signálu odstraníme. Nejprve ale musíme pomocí dolní propusti¹⁴ zajistit, že signál nebude obsahovat frekvence vyšší, než je čtvrtina přehrávací frekvence, aby nedošlo k aliasingu. Tím, že změním počet vzorků, ale přehrávací frekvence zůstane stejná, zkrátíme dobu přehrávání na polovinu a zároveň posuneme výšku každé frekvenční složky zvuku o oktávu nahoru.

Naopak pokud chceme zvuk zpomalit a jeho výšku tak snížit, je potřeba vzorky přidat. Ke vzorkům, které potřebujeme navíc, si dopomůžeme interpolací. Interpolací je víc druhů, často se ale používá lineární.

Harmonické složky $f_i = i * f_{základní}$ jsou přitom škálovány podle: $f_{i_{nova}} = v * f_i$, přičemž amplitudy se nemění: $a_{i_{nova}} = a_i$.

Závěr

Tuto metodu je možné použít jak pro jednohlasé, tak pro vícehlasé zvukové signály, ale pro účely této práce je nevhodná hned z několika důvodů. Za prvé nechceme při změně výšky složek zvuku měnit i jeho délku. Za druhé tato metoda

¹² Přehrávací a vzorkovací frekvence jsou většinou stejné.

¹³ Což odpovídá přehrávání relativní rychlostí $v = 2$.

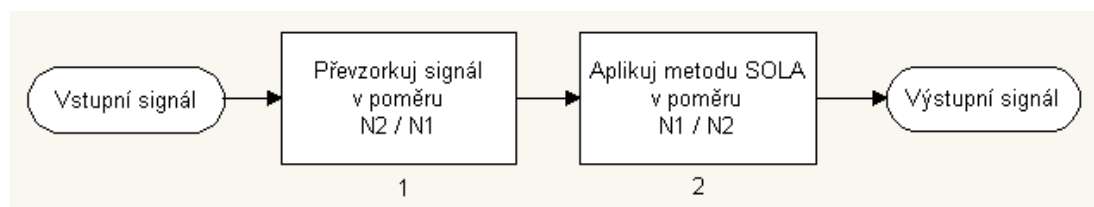
¹⁴ Dolní propust je druh filtru, který do výsledného signálu propustí jen frekvenční složky, které jsou nižší než určitá daná frekvence.

nezachovává formanty – škáluje spektrální obálku zvuku, jelikož nemění amplitudy nových frekvencí. Tím se mění charakter zvuku a při posunu výšky směrem nahoru potom vzniká dojem, že zvuk vydává někdo velmi malý¹⁵ a naopak při posunu výšky směrem dolů se zdá, že zvuk vydává někdo obrovský. U některých hudebních nástrojů posun formantů tolik nevaří, jelikož i ve skutečnosti se u nich formanty mění téměř proporčně se změnou výšky tónu. U zpěvu a hlasového projevu obecně jsou ovšem formanty na výšce tónu závislé mnohem méně.

Dále tím, že dochází k opakování nebo vynechání některých vzorků, může při větších změnách výšky docházet k problémům s opakováním nebo vymizením výrazných změn v signálu, které trvají velmi krátce – například úder paličky. Čím větší změna, tím víc je tento problém patrný.

2.1.2 SOLA

SOLA, neboli Synchronous Overlap and Add, je metoda sloužící pro změnu délky zvuku při zachování jeho výšky. Pokud takovou metodu zkombinujeme s převzorkováním popsaným v kapitole 2.1.1, které mění výšku i délku zvuku, dostaneme algoritmus pro změnu výšky zvuku bez ovlivnění doby jeho trvání.



Obrázek 2.3. Vývojový diagram popisující proces změny výšky zvuku pomocí algoritmu SOLA a převzorkování. N_1 je původní počet vzorků, N_2 je nový počet vzorků. Bloky 1 a 2 by mohly být zařazeny i v opačném pořadí.

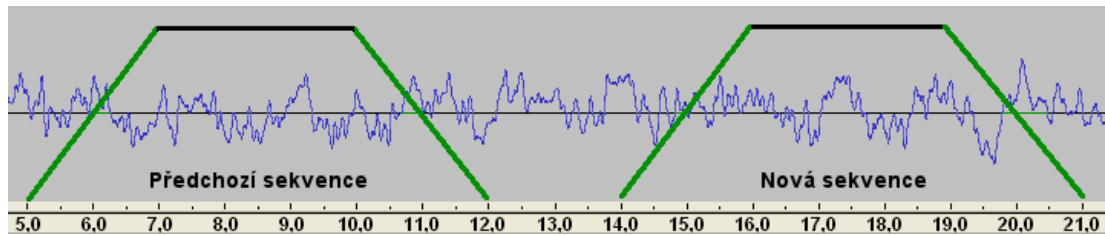
SOLA se dá rozdělit do dvou částí. V první části – analýze – dojde k rozdělení vstupního signálu do sekvencí po několika vzorcích. V druhé části – syntéze – jsou tyto sekvence opět spojeny.

¹⁵ V zahraniční literatuře často označováno jako „The chipmunk effect“.

Analýza

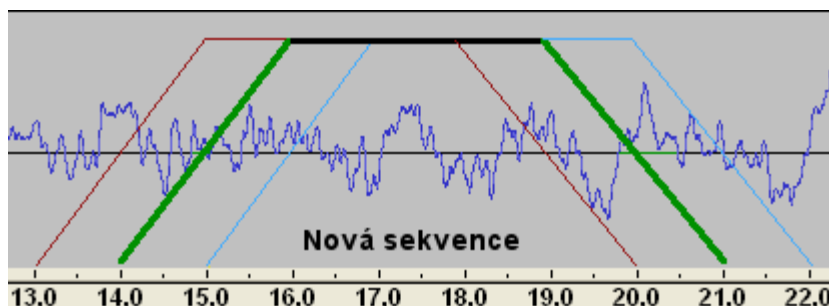
Vstupní signál je rozdělen do sekvencí délky n vzorků. Začátky jednotlivých sekvencí jsou od sebe vzdáleny o z vzorků. Poměr n/z musí být stejný jako poměr žádané a původní délky signálu. Pokud chceme signál prodloužit, budou některé vzorky ve více sekvencích, pokud chceme signál zkrátit, některé vzorky budou vynechány.

Při syntéze budeme potřebovat, aby se sekvence částečně překrývaly z důvodu zajištění lepší návaznosti v amplitudě, proto potřebujeme k délce sekvence n přidat ještě p vzorků, kde p odpovídá délce překryvu. Celková délka jednoho zpracovávaného úseku pak bude rovna $n + p$. Pro délku překryvu stačí, pokud je zlomkem délky úseku [23]. Rozdělení signálu do sekvencí ilustruje obrázek 2.4.



Obrázek 2.4. Ukázka rozdělení signálu do sekvencí. Na ose x jsou čísla vzorků. Sekvence mají délku $n = 5$, překryv $p = 2$ a vzdálenost mezi začátky sekvencí je $z = 9$. Zelené části označují, kde se budou sekvence při skládání překrývat.

Další opatření pro zvýšení plynulosti přechodu z jedné sekvence do druhé je umožnění výběru začátku každé nové sekvence z více kandidátů, jak ilustruje obrázek 2.5.

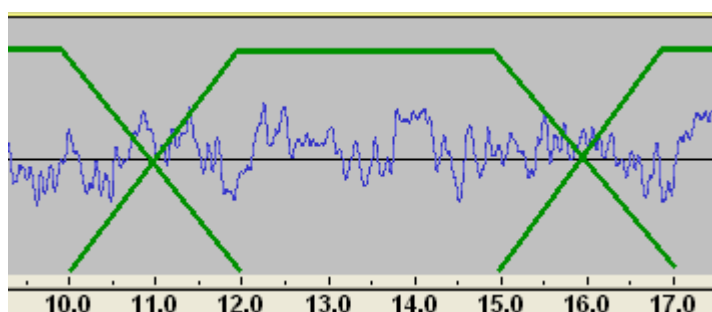


Obrázek 2.5. Výběr začátku sekvence. Zeleně označená je sekvence s počátkem daným vztahem $\text{počátek} = i * z$, kde i je index sekvence. Alternativní pozice pro tu samou sekvenci jsou vyznačeny červeně a modře. Interval pro volbu pozice začátku sekvence je zde tedy $\langle -1; +1 \rangle$.

Pro nalezení nejvhodnější pozice pro začátek nové sekvence se většinou používá vzájemná korelace. Pro každý možný začátek nové sekvence se počítá vzájemná korelace posledních p prvků předchozí sekvence s prvními p prvky nové sekvence, která by začínala na zvažovaném kandidátovi. Tudiž je to autokorelace částí, které se mají při syntéze překrývat. Nakonec se vybere ten kandidát, pro kterého má vzájemná korelace nejvyšší hodnotu. Mezi kandidáty patří určitý počet vzorků z okolí předpokládaného začátku nové sekvence. Čím víc kandidátů povolíme, tím větší je šance, že na sebe sousední sekvence budou dobře navazovat. Na druhou stranu pokud kandidátů bude příliš – skutečné vzdálenosti mezi začátky jednotlivých sekvencí se začnou příliš lišit od z – výsledný signál bude nestabilní [23].

Syntéza

Při syntéze skládáme sekvence za sebe v jejich původním pořadí s překryvem velikosti p .



Obrázek 2.6. Syntéza sekvencí. Překryv $p = 2$.

Překrývající se části nemůžeme jen sečíst dohromady – tím bychom uměle zvýšili amplitudu. Místo toho vynásobíme překryv na začátku sekvence lineárně stoupající funkcí od 0 do 1, čímž docílíme efektu běžně označovaného anglickým výrazem „fade in“. A překryv na konci sekvence lineárně klesající funkcí od 1 do 0, což se anglickou terminologií označuje jako „fade out“. Tím docílíme lepší návaznosti sekvencí.

Závěr

Nyní máme metodu pro změnu délky signálu, kterou můžeme použít k vrácení změny v délce trvání zvuku způsobené převzorkováním. Stejně jako převzorkování funguje SOLA pro jednohlasé i vícehlasé zvuky. Pořád nám ale zůstávají problémy s posunem formantů výsledného zvuku a artefakty způsobenými opakováním vzorků [23].

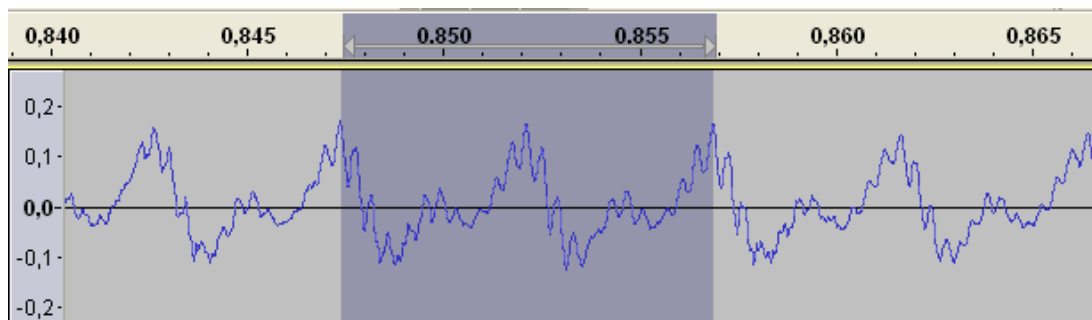
2.1.3 PSOLA

PSOLA neboli Pitch-synchronous Overlap and Add je ze stejné rodiny algoritmů, jako SOLA. Na rozdíl od algoritmu SOLA je ale navržena speciálně na zpracovávání hlasu a vychází z předpokladu, že vstupní zvuk je charakterizován výškou tónu. Dá se použít jak pro změnu doby trvání zvuku, tak pro změnu výšky tónu bez potřeby použití převzorkování, avšak kvůli závislosti na přítomnosti silné základní frekvence funguje správně jen pro jednohlasé zvuky. Zde si popíšeme jen variantu pro změnu výšky tónu.

Podobně jako SOLA se sestává ze dvou hlavních částí – analýzy a syntézy.

Analýza

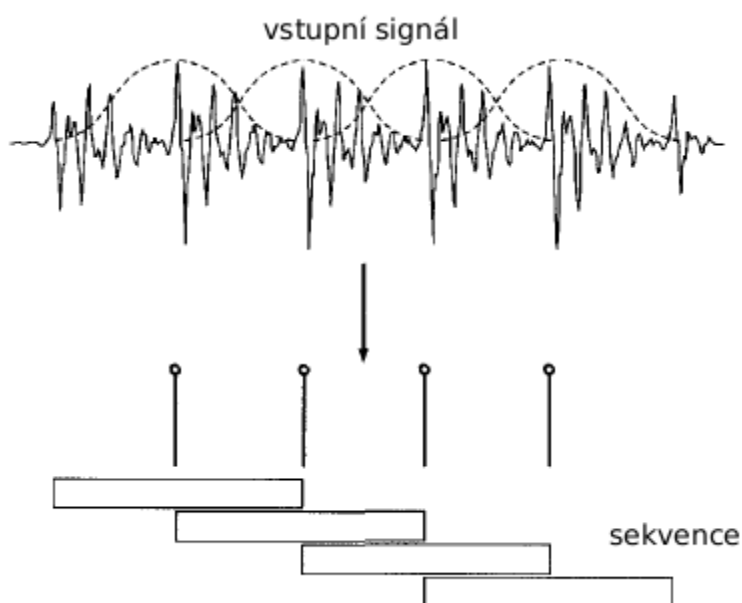
Vstupní signál rozdělíme na sekvence délky $2 * p$ vzorků, kde p je délka periody v signálu ve vzorcích. Délka periody se v různých částech signálu může lišit. Po sobě jdoucí sekvence se vždy překrývají o délku jedné periody a každá sekvence je vycentrovaná na vzorek s maximální amplitudou v rámci jedné periody. Střed i -té sekvence budeme značit c_i . Výběr sekvence je naznačen na obrázcích 2.7 a 2.8.



Obrázek 2.7. Výběr úseku o délce $2 * p$ vycentrovaného na maximální amplitudu periody.

V aperiodických částech signálu – například šum nebo souhlásky – rozdělíme signál na konstantně dlouhé části, opět z poloviny se překrývající. Můžeme například použít délku poslední známé periody.

Obdobou použití funkcí fade in a fade out u syntézy algoritmu SOLA, je použití Hannova okna na každou vyjmutou sekvenci signálu.



Obrázek 2.8. Rozklad vstupního signálu do sekvencí. Každá sekvence je vycentrovaná na vzorek s maximální amplitudou v rámci periody. Převzato z [6].

Syntéza

Abychom dosáhli změny výšky tónu, potřebujeme změnit vzdálenost mezi centry jednotlivých sekvencí – tím změníme délku period a tedy základní frekvenci

signálu. Chceme-li například zkrátit periodu o 50%, budou se při syntéze původní sekvence překrývat o $1,5 p$. Navíc, aby zároveň nedošlo ke zkrácení délky zvuku, tak některé sekvence použijeme vícekrát. Naopak pokud bychom chtěli délku periody o 50% prodloužit, překrývaly by se sekvence při syntéze jen o $0,5 p$ a některé sekvence bychom úplně vynechali.

Označme vzorky, na kterých budou vycentrované sekvence při syntéze, jako c'_k pro k -tou sekvenci syntézy. Pro každé nově vypočtené centrum c'_k potřebujeme najít sekvenci, kterou na něj vycentrovat. Vybereme sekvenci, pro kterou je $|c_i - c'_k|$ minimální, kde c_i je centrum i -té sekvence z analýzy.

Závěr

Tato metoda zachovává spektrální obálku původního signálu nezměněnou, pouze ji převzorkuje na nových frekvencích [6]. To jinými slovy znamená, že se zachovají formanty. Stále hrozí, že při velkých změnách výšky vzniknou artefakty připomínající ozvěnu. U zpěvu je ale toto nebezpečí menší, jelikož ve zpěvovém signálu se nevyskytují rychlé přechody, jako jsou například u bicích nástrojů nebo i u kytary, kde by se mohlo stát, že se zopakuje úder prstu nebo trsátka o strunu.

Tato metoda je velmi citlivá na výběr kvalitního algoritmu pro zjištění periody v signálu [6, 7]. Jakákoliv nepřesnost v určení periody silně snižuje kvalitu výsledného zvuku.

2.2 Algoritmy manipulující se signálem ve frekvenční doméně

Jak bylo řečeno v úvodu kapitoly 2, výšku tónu je možné posunout tím, že každou frekvenční složku vynásobíme určitým číslem. K tomu ale potřebujeme jednotlivé frekvenční složky znát. V následující kapitole si popíšeme, jak frekvenční složky najít a jaké problémy jsou s tímto přístupem spojené.

2.2.1 Fázový vokoder

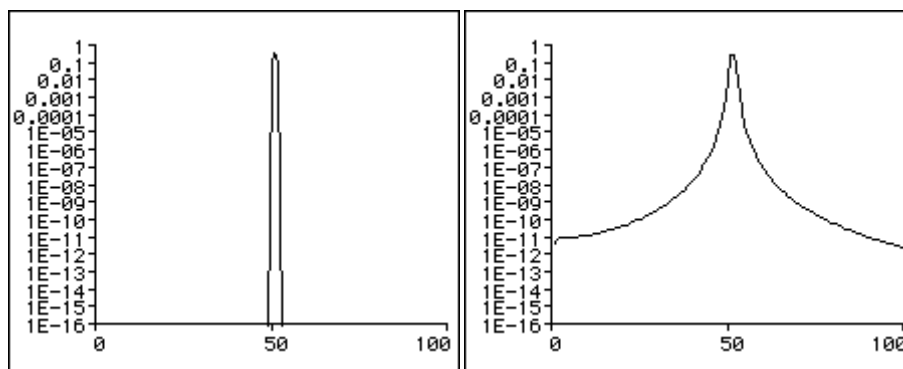
Fázový vokoder je algoritmus zpracovávající signál ve frekvenční doméně. Kromě změny výšky zvuku se dá použít k mnoha dalším hudebním efektům. Tato kapitola se bude věnovat popisu implementace fázového vokoderu pro účely změny výšky tónu pomocí DFT. Formálnější popis této i jiných implementací je možné nalézt v [6, 13].

Algoritmus fázového vokoderu má tři základní části: analýzu, transformaci a syntézu. Během analýzy dochází k převodu signálu z časové do frekvenční domény. Během transformace může dojít ke změně frekvenčního spektra a během syntézy dochází k převodu z frekvenční domény zpět do časové.

Analýza

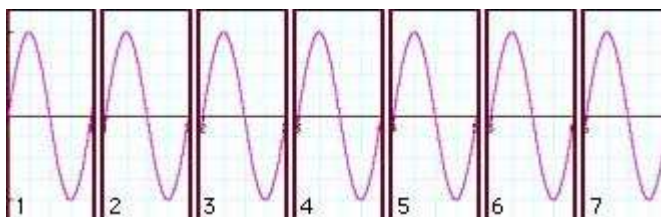
Většina signálů – včetně zpěvového – je kvaziperiodická. To znamená, že přesto, že se v čase výrazně mění, tak na krátkých úsecích jsou téměř dokonale periodické. Proto nemá smysl uvažovat o DFT celého signálu, při které by se informace o změnách ztratily. Místo toho provedeme DFT na mnohem kratších úsecích signálu – oknech. Tento způsob použití diskrétní Fourierovy transformace na malé úseky signálu se v DSP nazývá Short-Time Fourier Transform (STFT) signálu.

Problém s použitím STFT je ten, že používá fixní tabulku frekvencí. Pokud má například okno STFT délku 1024 vzorků a vzorkovací frekvence je 44 100 Hz, tak vzdálenost mezi jednotlivými přesně reprezentovatelnými frekvencemi bude $44100 / 1024$ což je přibližně 43 Hz. Tudiž pokud je ve zkoumaném signálu například frekvence 66 Hz, tak se její magnituda rozdělí mezi okolní fixní frekvence, jak ilustruje obrázek 2.9.



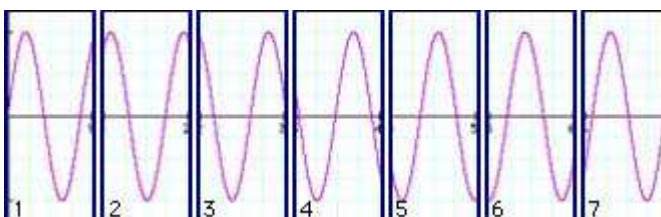
Obrázek 2.9. Frekvenční spektrum pro signál obsahující frekvenci odpovídající některé z fixních frekvencí STFT a frekvenční spektrum pro signál obsahující frekvenci, která je někde mezi dvěma fixními. Převzato z [19].

Pokud frekvence v signálu bude odpovídat některé z fixních frekvencí STFT, pak to bude znamenat, že v každém okně STFT bude začínat se stejnou fází. Pro ilustraci na obrázku 2.10 začíná s nulovou počáteční fází.



Obrázek 2.10. Rozdělení signálu v časové doméně do oken STFT. Frekvence signálu odpovídá některé fixní frekvenci STFT. Převzato z [19].

Pokud bude skutečná frekvence někde mezi dvěma fixními, bude v každém okně začínat s jinou fází, jak ilustruje obrázek 2.11. Z rozdílu fází fixních frekvencí ve dvou po sobě jdoucích oknech se pak dá dopočítat skutečná frekvence, jak si ukážeme později.



Obrázek 2.11. Rozdělení signálu v časové doméně do oken STFT. Frekvence signálu neodpovídá žádné z fixních frekvencí STFT. Převzato z [19].

Dalším problémem je, že pokud signál začíná v každém okně s jinou fází, tak kvůli nenávaznostem na hranicích okna se do spektra během STFT přidávají frekvence navíc. Proto se na okna signálu před vstupem do STFT aplikuje okénková funkce. Smyslem použití okénkové funkce je signál na krajích pozvolna zesílit a pak opět ztlumit. Tím se docílí lepší návaznosti a dojde k redukci počtu falešných frekvencí, které se do spektra během STFT přidávají navíc.

Abychom byli schopni jednoznačně určit rozdíl ve fázi fixní frekvence mezi dvěma po sobě jdoucími okny, potřebujeme, aby se okna signálu částečně překrývala. Většinou se používá překryv alespoň 75%.

Důvod je následující. Fáze se počítá pomocí funkce atan2^{16} , která má obor hodnot $(-\pi; +\pi)$. Tudíž i kdyby skutečný rozdíl ve fázi frekvence mezi dvěma po sobě jdoucími okny byl větší než π , tak to nezjistíme, protože to je mimo obor funkce. Čím větší mírou se tedy budou okna překrývat, tím menší je pravděpodobnost, že nám rozdíl ve fázi takto „přeteče“. Pokud by k přetečení došlo, mělo by to za následek špatný výpočet skutečných frekvencí a následně, pokud bychom manipulovali s frekvenčním spektrem, slyšitelné chyby ve výsledném zvuku [19].

Nyní bychom, ovšem kvůli překryvu oken, měli nenulový rozdíl ve fázi i u frekvencí, které odpovídají našim fixním. Abychom toto spravili, je potřeba od již vypočteného rozdílu fází odečíst předpokládaný posun ve fázi zavedený překryvem. Ten se k fixní frekvenci s indexem i snadno spočítá jako:

$$p_i = 2 * \pi * i * \frac{(N - \text{překryv})}{N}$$

Kde N je velikost okna ve vzorcích a překryv je velikost překryvu oken ve vzorcích.

Skutečná frekvence pro fixní frekvenci s indexem i se potom spočítá jako:

$$f'_i = \frac{f_s}{N} * \left(i + \text{diff} * \frac{p_f}{2\pi} \right)$$

Kde f_s je vzorkovací frekvence, diff je celkový rozdíl ve fázi pro po sobě jdoucí okna pro fixní frekvenci s indexem i a $p_f = \frac{N}{N - \text{překryv}}$ je faktor překryvu.

Říkali jsme, že pokud skutečná frekvence neodpovídá přesně žádné fixní frekvenci, tak se její magnituda rozprostře na nejbližší fixní frekvence. f'_i nám říká, jaká skutečná frekvence se na fixní frekvenci f_i takto „zaokrouhlila“. Mějme například dvě sousední fixní frekvence o velikostech $f_1 = 20$ Hz a $f_2 = 40$ Hz a

¹⁶ Funkce $\text{atan2}(x,y)$, pro reálná x a y , která nejsou obě zároveň nulová, vrací v radiánech velikost úhlu svíraného kladnou částí osy x roviny a úsečkou procházející počátkem soustavy souřadnic a bodem (x,y) .

silnou frekvenci skutečně obsaženou v signálu o velikosti $f = 30$ Hz. Právě tato skutečná frekvence by nám poté měla vyjít jako hodnota pro f'_1 a f'_2 .

Je stále možné, že u některých fixních frekvencí dojde k přetečení, takové frekvence by už ale měly být relativně daleko od skutečné frekvence a tudíž budou mít nízkou magnitudu.

Transformace

V případě, že nepotřebujeme zachovat formanty, stačí, když zde vynásobíme získané skutečné frekvence škálovacím faktorem. Například pokud chceme posunout zvuk o oktávu dolů, tak vynásobíme frekvence faktorem 0,5. Pokud o oktávu nahoru, tak frekvence vynásobíme faktorem 2. Magnitudy zachováme.

Jediné, na co si musíme dát při škálování pozor, je, abychom nepočítali frekvence vyšší než je Nyquistova frekvence a nedošlo k aliasingu.

V případě, že formanty zachovat chceme, je nutné vypočítat spektrální obálku původního signálu a amplitudu pro každou novou frekvenci určit podle hodnoty spektrální obálky pro danou frekvenci. V praxi se používá ještě o něco složitější koncept. Pro detailnější popis použití a výpočtu spektrální obálky doporučuji [12].

Syntéza

Abychom získali výstupní signál, stačí vrátit všechny změny, které jsme udělali během analýzy. Tudíž pro získání rozdílu mezi fázemi spočítáme pro každý index i z intervalu $\langle 0; N/2 \rangle$, kde N je velikost okna:

$$p'_i = \left(\frac{\left(\left(f'_i - \left(\frac{f_s}{N} * i \right) \right) * N * 2\pi \right)}{f_s} * p_f \right) + \left(2\pi * i * \frac{N - \text{překryv}}{N} \right)$$

Kde f'_i je skutečná frekvence pro index i , p_f je faktor překryvu oken a překryv je velikost překryvu oken.

K výsledku přičteme fázi vypočtenou pro předchozí okno a stejný index i . Tím získáme fázi pro každou fixní frekvenci.

Nyní tedy známe pro každý index i magnitudu i fázi. Z nich dopočteme hodnoty pro kosinovou a sinovou část transformace jako $magnituda * \sin(fáze)$ a $magnituda * \cos(fáze)$.

Zbývá už je provést inverzní STFT, použít na výsledek opět okénkovou funkci, jakou jsme použili při extrakci signálu a překrýt okna a sečíst.

Závěr

Fázový vokoder je podstatně výpočetně náročnější než uvedené metody pracující v časové doméně. Na rozdíl od metod pracujících v časové doméně použitím fázového vokoderu nevznikají na signálu artefakty podobné ozvěnám. Použití fázového vokoderu může v signálu měnit vztah mezi fázemi jednotlivých frekvenčních složek a způsobovat tak jiné slyšitelné artefakty [15, 22, 23]. Pro odstranění těchto artefaktů existují různá vylepšení [13, 15, 24, 25, 26, 27]. Tato vylepšení nadále zvyšují výpočetní náročnost algoritmu. Ta nejefektivnější z těchto vylepšení jsou vázána patenty a obchodními tajemstvími [24]. Pro detailnější informace o této problematice doporučuji k prostudování [6, 13].

Pomocí fázového vokoderu je možné měnit výšku i u vícehlasých zvuků.

3 Zvolené řešení

Praktickou částí zadání bylo implementovat algoritmus pro harmonický posun frekvence zpěvového signálu, umožnit řízení posunu pomocí MIDI a celé řešení implementovat jako modul, který bude schopný spolupracovat se standardním nahrávacím software. Harmonickým posunem je zde myšleno to, že jsou při posunu zachovány vztahy mezi harmonickými složkami. Signál, se kterým chceme pracovat, je jednohlasý a relativně pomalu se měnící¹⁷. Všechny čtyři metody popsané v kapitole 2 jsou pro splnění tohoto zadání použitelné, ale jen poslední dvě – PSOLA a fázový vokoder – umožňují zachovat formanty, což je u práce se zpěvovým signálem důležité.

Aby bylo možné s fázovým vokoderem dosáhnout dostatečně kvalitních výsledků, bylo by potřeba aplikovat na něj různá vylepšení, z nichž mnohá jsou součástí obchodních tajemství (viz závěr kapitoly 2.2.1). Navíc ani veřejně publikovaná vylepšení nejsou ve většině případů doprovázena ukázkami, podle kterých by bylo možné jejich kvalitu porovnat. Proto jsem se rozhodla zabývat se dále algoritmem PSOLA. Po dalším čtení a konzultacích jsem si nakonec k implementaci vybrala algoritmus, který poprvé navrhl Keith Lent v [7] a který, jak si ukážeme v kapitole 3.2.1, je metodě PSOLA velmi podobný, přestože se oba algoritmy vyvinuly nezávisle na sobě. Je vhodný pro změnu výšky jednohlasého hudebního signálu se zachováním formantů a stejně jako PSOLA pracuje v časové doméně. K jeho správné funkci je nezbytný kvalitní algoritmus na vyhledávání základní frekvence v signálu. Algoritmů na určování základní frekvence v signálu bylo navrženo nespočetné množství, žádný z nich však není dokonalý a každý je vhodný jen pro určitý typ signálu. K. Lent ve zmíněném článku navrhuje použít detekci průchodů nulou¹⁸ na signál vyfiltrovaný pásmovou propustí. Takovýto postup je vhodný ale jen pro signály s velmi omezeným rozsahem frekvencí – jako je řeč.

¹⁷ Při zpěvu jsou změny ve výšce zpívaných tónů většinou pomalejší než třeba při sólu na elektrickou kytaru. Stejně tak ve zpěvu není nic tak výrazného a rušivého jako úder trsátka atp.

¹⁸ Anglicky „zero-crossing“. Hledáme místa, kde se mění znaménko okamžité amplitudy signálu.

Řešení se dá tedy rozdělit do tří hlavních částí:

- nalezení základní frekvence v signálu
- změna výšky tónu
- implementace ve formě modulu a MIDI ovládání

3.1 Algoritmus pro hledání základní frekvence

Signál, se kterým chceme pracovat, je kvaziperiodický. Tudíž na krátkých úsecích je téměř dokonale periodický. Jelikož základní frekvence periodického signálu je inverzní k jeho periodě, dá se hledání základní frekvence snadno převést na hledání periody v signálu. Pokud tedy v signálu $x[n]$ existuje nějaká perioda délky d , musí platit, že $x[n] = x[n + d]$. Samozřejmě toto by byl ideální případ. Když pracujeme s reálnými signály, tak tato rovnost většinou neplatí, ale dá se tvrdit, že $x[n]$ a $x[n + d]$ si budou často velmi podobné. Právě nedokonalosti v periodicitě jsou to, co tolik ztěžuje určování základní frekvence. Můžou být způsobené například přidáním šumu z dechu narážejícího na mikrofon, hluku na pozadí, změnou tvaru zpěvového traktu¹⁹ nebo nestálým dechem zpěváka, který způsobuje výkyvy v amplitudě při zachování frekvence – jinými slovy zpěvák ten samý tón zpívá chvíli hlasitěji, chvíli tišeji.

Další komplikací je to, jak člověk vnímá výšku tónu. Již jsme si řekli, že ne vždy základní frekvence odpovídá vnímané výšce tónu. Nicméně k tomu nedochází příliš často, a proto se v DSP mnohdy pojmy „hledání základní frekvence tónu“ a „hledání výšky tónu“ používají pro monofonní zvuky záměnně. Tohoto přístupu se bude držet i tato práce a nadále budeme postupovat, jako by si základní frekvence a vnímaná výška tónu vždy odpovídaly.

¹⁹ Stačí pohyb jazyka, úst, měkkého patra apod.

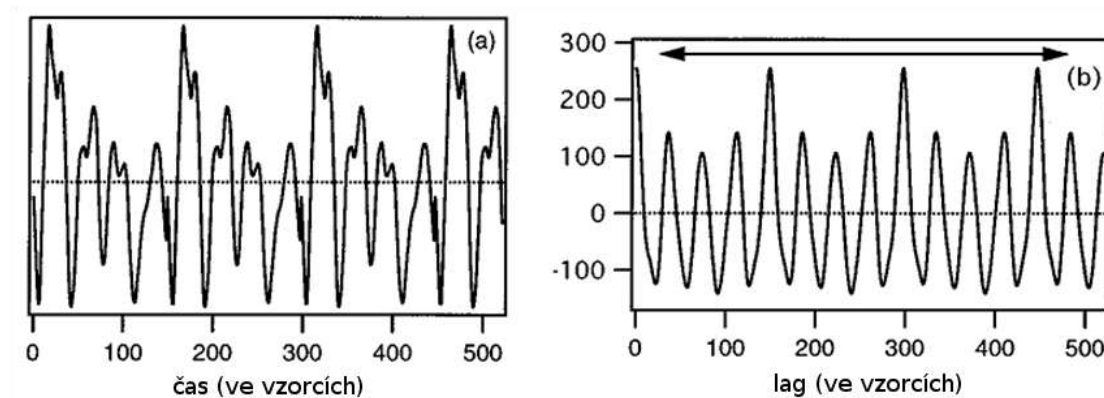
3.1.1 Popis algoritmu YIN

Pro hledání základní frekvence tónu byla nakonec použita variace algoritmu YIN, který byl poprvé popsán v [8]. YIN pracuje v časové doméně a vychází z autokorelační metody, kterou v pěti²⁰ krocích optimalizuje.

Při použití autokorelační metody rozdělíme signál na okna v délce W . Typicky má okno délku několika milisekund. Autokorelační funkce pro diskrétní signál x_t může být definována jako:

$$r_t(lag) = \sum_{j=t+1}^{t+W} x_j x_{j+lag} \quad (4)$$

Kde t je časový index a lag je délka testované periody ve vzorcích. Pokud je lag roven délce skutečné periody signálu v daném časovém úseku nebo některému jejímu celočíselnému násobku, tak hodnota $r_t(lag)$ bude vysoká. Pro hodnoty lag lišící se od skutečné periody či jejích celočíselných násobků, bude hodnota autokorelační funkce nižší.



Obrázek 3.1. a) okno signálu. b) hodnoty autokorelační funkce pro signál z a) pro různé hodnoty proměnné lag . Převzato z [8].

Jak je vidět na obrázku 3.1, perioda je dlouhá přibližně 150 vzorků. Autokorelační funkce pak má maxima vždy na celočíselných násobcích periody. Pokud, tedy, chceme najít periodu signálu, budeme hledat mezi hodnotami lag , pro které má autokorelační funkce lokální maximum. U autokorelační metody máme typicky stanovenou horní a dolní mez pro hledání periody. Pokud je dolní mez moc

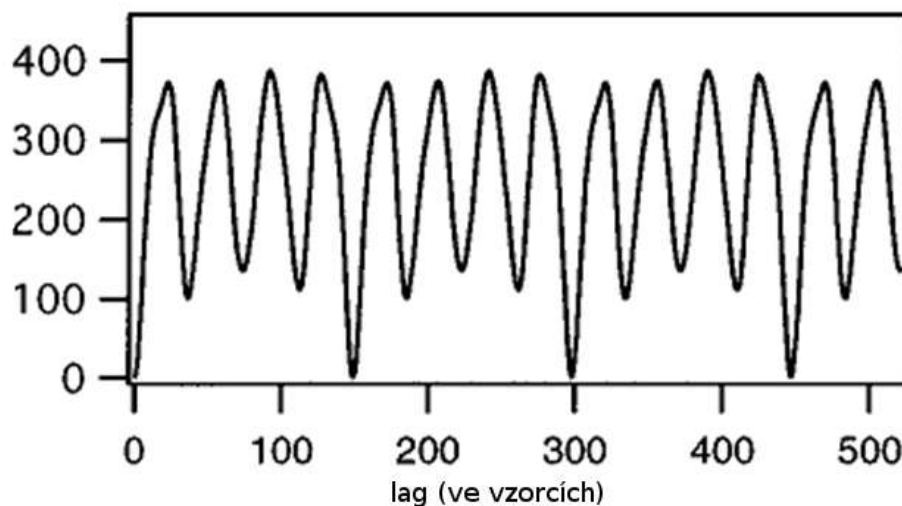
²⁰ Text článku [8] je rozdělen do šesti kroků, optimalizačních je ale jen pět.

nížká, hrozí, že chybně vybereme lag odpovídající vrcholku autokorelační funkce kolem nuly²¹. Pokud je naopak moc vysoká, hrozí, že vybereme například dvojnásobek nebo trojnásobek skutečné periody. U autokorelační metody pro hledání základní frekvence bohužel k těmto omylům dochází příliš často na to, aby byla reálně použitelná [8].

A.de Cheveigné a H. Kawahara v [8] tvrdí, že chybovost klesne z 10% na 1,95% pokud se autokorelační funkce nahradí auto-diferenční funkcí tvaru:

$$d_t(lag) = \sum_{j=t+1}^{t+W} (x_j - x_{j+lag})^2 \quad (5)$$

Vysvětlují to tím, že auto-diferenční funkce je méně citlivá k výkyvům amplitudy způsobeným drobnými změnami hlasitosti. Toto je zároveň první krok optimalizace.



Obrázek 3.2. Auto-diferenční funkce pro signál z obrázku 3.1 a). Převzato z [8].

U auto-diferenční funkce nás zajímají naopak nejnižší hodnoty a ne vrcholky, jako tomu bylo u auto-korelační funkce. Stejně jako u auto-korelační funkce opět hrozí vybrání příliš nízké nebo příliš vysoké hodnoty lag . Kromě problémů s nulovou hodnotou pro lag velikosti nula, existuje ještě problém s poklesy hodnot

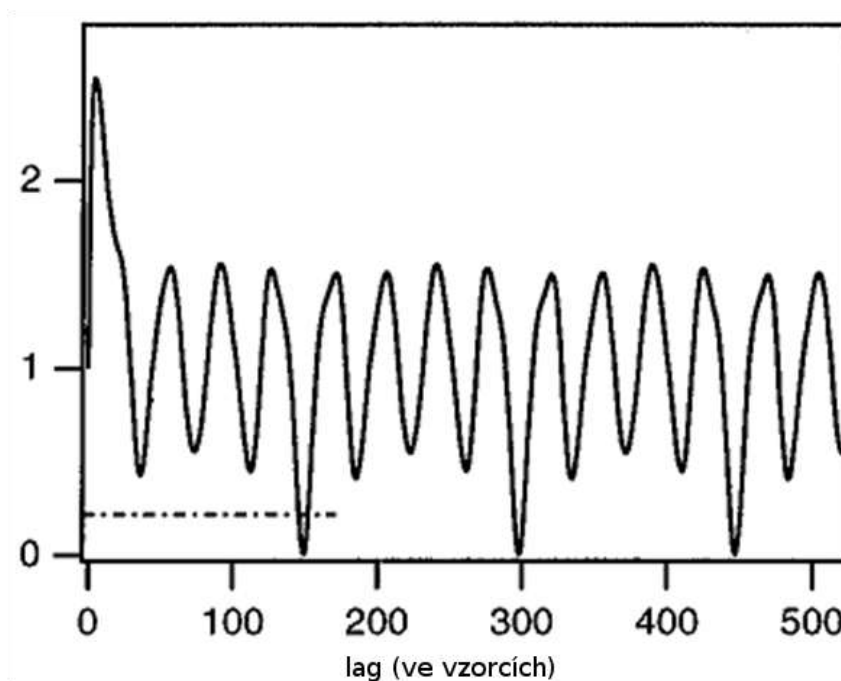
²¹ Navíc většina zkoumaných signálů nebude mít tak dokonale periodický průběh, jako signál na obrázku 3.1 a proto bude hodnota autokorelační funkce pro lag rovný nebo blízký nule dokonce vyšší než pro lag rovný délce skutečné periody.

auto-diferenční funkce pro lag odpovídající periodě prvního formantu, v případě, že je první formant hodně silný. Tento problém se nevyřeší nastavením mezí, protože frekvenční rozsahy prvního formantu a základní frekvence se často překrývají [8].

Proto A.de Cheveigné a H. Kawahara v [8] ve druhém kroku optimalizace navrhuji následující modifikaci auto-diferenční funkce:

$$d'_t(lag) = \begin{cases} 1 & \text{pro } lag = 0, \\ d_t(lag) / [(1/lag) \sum_{j=1}^{lag} d_t(j)] & \text{jinak.} \end{cases} \quad (6)$$

Výsledná funkce se liší od původní auto-diferenční funkce tím, že hodnotou začíná na 1 místo na 0 a její hodnoty zůstávají vysoké pro malá lag a pod 1 klesnou jen pokud $d_t(lag)$ klesne pod průměr. Při použití této funkce podle [8] klesne chybovost z 1,95% na 1,69% díky tomu, že se sníží počet případů, kdy je chybně vybrán příliš nízký lag a navíc odpadá potřeba dolní meze.



Obrázek 3.3. Pozměněná auto-diferenční funkce pro signál z obrázku 3.1 a).
Převzato z [8].

Třetí krok optimalizace se zabývá případem, kdy dojde k tomu, že je chybně vybrán příliš vysoký lag ²². Stát se to může zejména kvůli nedokonalé periodičnosti signálu na daném úseku.

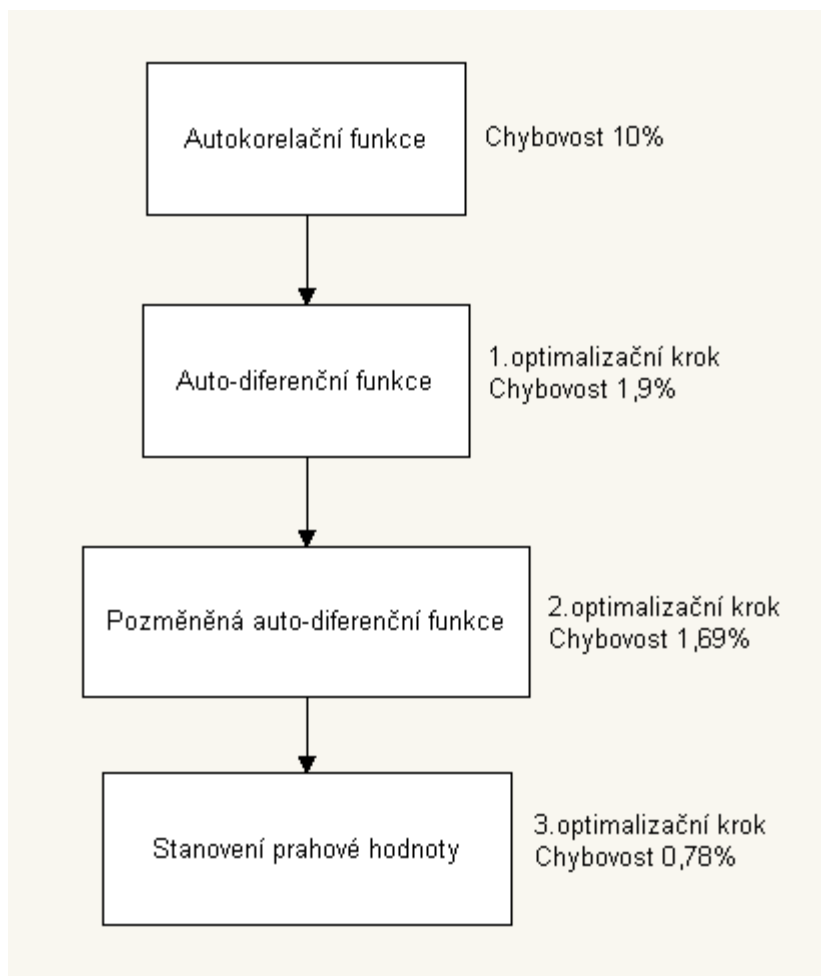
²² Nejčastěji některý vyšší násobek správné periody.

A.de Cheveigné a H. Kawahara v [8] navrhnou stanovit prahovou hodnotu a jako výslednou periodu zvolit nejmenší *lag*, pro nějž hodnota upravené auto-diferenční funkce klesne pod zvolený práh. Pokud hodnota auto-diferenční funkce pro žádný *lag* neklesne pod práh, tak je vybrán *lag*, pro který má auto-diferenční funkce na úseku signálu, daném časovým indexem t a velikostí okna W , globální minimum. Prahová hodnota navrhaná v článku je 0,1. S touto hodnotou podle [8] klesne chybovost z 1,69% na 0.78%.

Zatím celou dobu pracujeme s celočíselnými hodnotami proměnné *lag*, není ale nic, co by nám zaručovalo, že skutečná perioda signálu neleží někde mezi dvěma sousedními hodnotami *lag*. Čtvrtý krok optimalizace se zabývá přesnějším určením periody pomocí parabolické interpolace. Díky tomu ve výsledku můžeme dostat přesnější, neceločíselnou periodu. Tento krok optimalizace nebyl v rámci této práce použit, jelikož neceločíselná perioda by komplikovala implementaci následného algoritmu pro posunu výšky tónu. V závislosti na vzorkovací frekvenci tato skutečnost může a nemusí být problém. Obecně platí, že čím je vzorkovací frekvence vyšší, tím vyšší je rozlišení v periodě a tím menší je nepřesnost způsobená hledáním periody jen mezi celými čísly.

Poslední, pátý, navrhaný krok optimalizace spočívá v prohledávání odhadů periody pro časové indexy z nejbližšího okolí právě zkoumaného časového indexu t , konkrétně z $[t - T_{max}/2, t + T_{max}/2]$, kde T_{max} je maximální očekávaná perioda. Jelikož cílem této práce je implementovat algoritmus pracující v reálném čase a máme k dispozici vždy jen část signálu, kterou je potřeba okamžitě zpracovat, není možné porovnávat výsledek pro daný časový index s výsledky pro indexy budoucí. Proto tento krok nebyl implementován.

Obrázek 3.4 shrnuje vývoj algoritmu YIN a použité kroky optimalizace. Autoři vychází z autokorelační funkce (4), kterou se v prvním optimalizačním kroku rozhodnou nahradit auto-diferenční funkcí (5). Ve druhém optimalizačním kroku ji dále upravují na (6), aby snížili počet případů, kdy je chybně vybrán příliš nízký *lag* a zbavili se potřeby dolní meze. Ve třetím optimalizačním kroku stanovují prahovou hodnotu, aby omezili počet případů, kdy je chybně vybrán příliš vysoký *lag*.



Obrázek 3.4. Vývoj algoritmu YIN, použité kroky optimalizace.

3.1.2 O Implementaci

V této kapitole si popíšeme některá implementační rozhodnutí a důvody, které k nim vedly. V neposlední řadě se tato kapitola zabývá prezentací vlastních vylepšení, která nejsou součástí původního algoritmu.

Volba délky okna

Nejdelší naležitelná perioda v signálu, při zpracování algoritmem YIN, je rovná velikosti okna W . Pro každý časový index potřebuje YIN $2 * W$ vzorků. Jelikož se tato práce zabývá zpěvovým signálem, můžeme maximální délku periody, kterou potřebujeme detekovat, určit z rozsahu lidského hlasu. Jak bylo řečeno v kapitole 1.4, základní frekvence lidského hlasu, až na výjimečné případy, neklesne pod 80 Hz. Potřebujeme tedy detekovat jen periody kratší než 12,5 ms.

V závislosti na aktuální vzorkovací frekvenci f_s , určíme délku okna W a zároveň i délku maximální periody ve vzorcích jako:

$$W = f_s/80 \quad (7)$$

W zaokrouhlíme na celé číslo.

Rozlišení v periodě

Z rovnice (7) je vidět, že čím nižší je vzorkovací frekvence, tím nižší je W a v důsledku toho i rozlišení v periodě. Použitá vzorkovací frekvence záleží na nastavení hudebního editoru, který efekt²³ hostuje. Většina editorů uživatelé umožňuje nastavit vzorkovací frekvenci a často i bez ručního nastavení, automaticky používá vysoké frekvence jako 44100 Hz, při kterých je rozlišení v periodě dostatečné.

Pokud by bylo z nějakého důvodu nutné algoritmus přizpůsobit pro práci s nízkými vzorkovacími frekvencemi, bylo by vhodné zvážit použití nadvzorkování signálu k umělému zvýšení rozlišení v periodě.

Snížení zpoždění

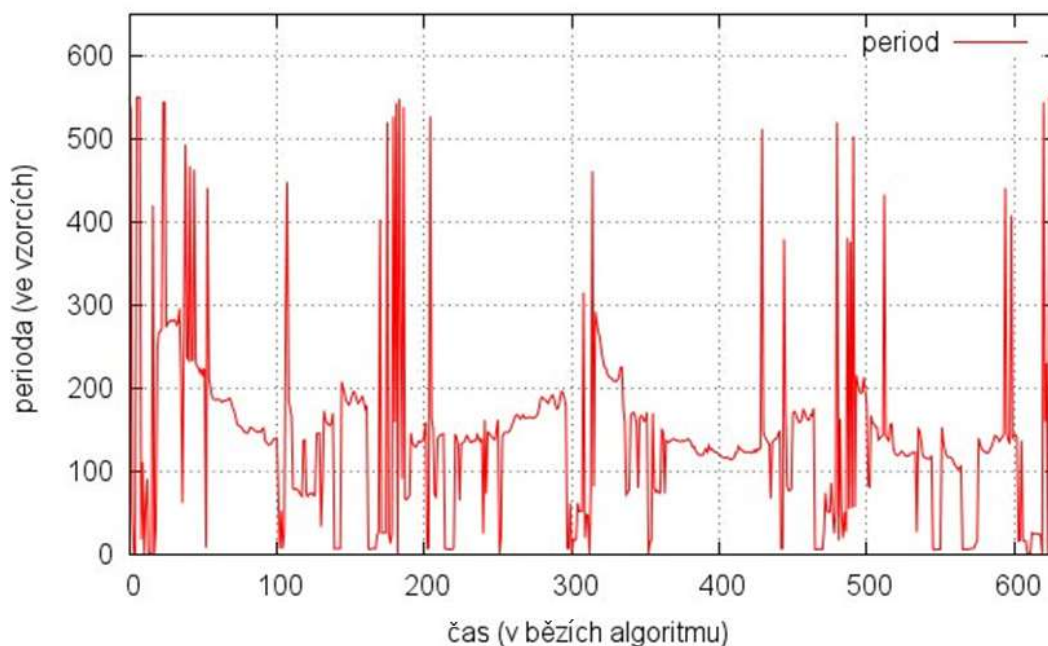
Jelikož potřebujeme detekovat periodu o délce až 12,5 ms a YIN potřebuje mít k dispozici alespoň dvě periody signálu, potřebujeme do vyrovnávací paměti před prvním během algoritmu načíst vzorky odpovídající 25 ms. To už je ale nezanedbatelné zpoždění. Lidský sluch začne vnímat zpoždění výstupního signálu už zhruba od 20 ms jako rušivé. Abychom zpoždění snížili, necháme poprvé algoritmus proběhnout jen s polovičním počtem vzorků a zbytek vyrovnávací paměti doplníme nulami. V důsledku toho budou správně detekovatelné jen frekvence nad 160 Hz.

Tato úprava má vliv pouze na prvních 12,5 ms signálu. Všechny další běhy už budou operovat na skutečných datech v plné délce 25 ms. Vzhledem k tomu, že mezi zapnutím efektu a začátkem zpěvu je většinou určitá pauza, tak to často nevádí. Každopádně vyšší latence by byla v daném případě rušivější, než možná chvilková nepřesnost.

²³ Vztah mezi modulem, v jehož podobě má být efekt dle zadání implementován, a hostitelskou aplikací bude blíže popsán v kapitole 3.1.1.

Vylepšení detekce periody

Ve zbytku kapitoly si popíšeme přidaná vylepšení detekce periody. Pro ilustraci byl algoritmus spuštěn před a po některých změnách na zvukovém souboru *mic.wav*, který lze najít na přiloženém cd v adresáři „Přílohy“. Zvuk byl nahrán přes levný mikrofon určený pro internetové hovory.



Obrázek 3.5. Hodnoty vrácené algoritmem pro nalezení periody v první verzi, kdy byl implementován algoritmus YIN, popsany v kapitole 3.2.1 bez posledních dvou kroků optimalizace.

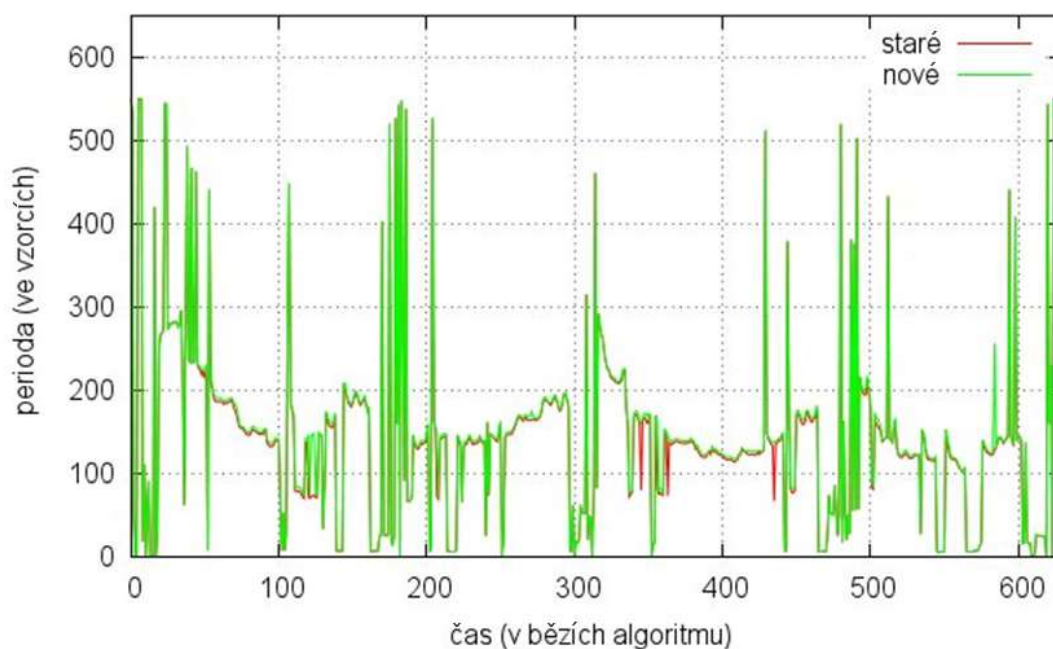
Z obrázku 3.5 je vidět, že algoritmus často vybere příliš vysokou nebo nízkou hodnotu. Důvody jsou zpravidla dva:

- Zkoumaná část signálu je téměř aperiodická – může být způsobeno přítomností velkého množství šumu, třeba při souhláskách, nebo přechody mezi tóny či formanty u změny samohlásek nebo naprostou nepřítomností zpěvového signálu.
- Je přítomná silná druhá harmonická složka, a jelikož má vyšší frekvenci, tak má nižší periodu a algoritmus mylně vybere ji, když pro ni hodnota pozměněné auto-diferenční funkce klesne pod zvolený práh.

Nahrávka *mic-yin.wav* v adresáři „\Přílohy\3.1.2\01 - YIN“ ilustruje dopad chybného výběru periody na kvalitu zvuku při změně výšky tónu. Jelikož jsou chyby patrnější pro větší změnu výšky tónu, jsou tóny v nahrávce posunuty o oktávu nahoru. To samé platí pro všechny další nahrávky v této kapitole.

Úprava třetího optimalizačního kroku algoritmu YIN – lokální minimum

Jak již bylo řečeno v kapitole 3.1.1, YIN vybírá první hodnotu, která spadá pod zvolený práh. Ovšem lepší je vybírat z hodnot pod prahem lokální minimum, tím se mírně zvýší přesnost odhadu periody, jak je vidět na obrázku 3.6. Již víme, že rozlišení v periodě je omezené vzorkovací frekvencí. Proto i chyba o velikosti několika málo vzorků, při odhadu periody, může znamenat velký rozdíl.



Obrázek 3.6. Hodnoty vrácené algoritmem pro nalezení periody ve druhé verzi, kdy byl přidán výběr lokálního minima. Červeně jsou hodnoty z předchozí verze, zeleně jsou nové hodnoty.

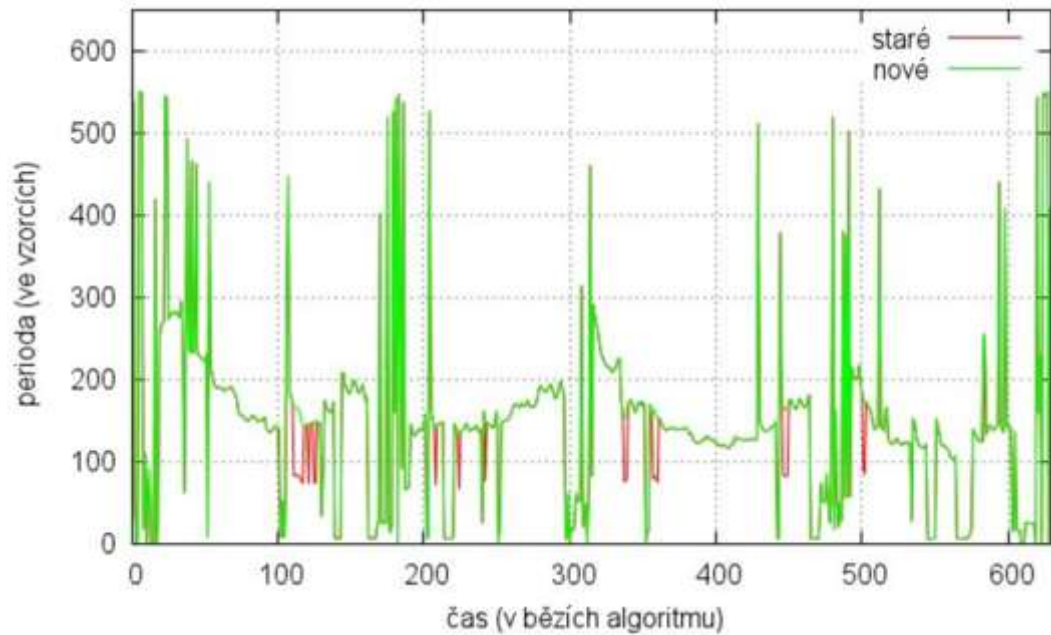
Úprava třetího optimalizačního kroku algoritmu YIN – více kandidátů

Tento krok se zabývá situací, kdy algoritmus mylně vybere periodu odpovídající druhé harmonické složce. Druhá harmonická složka bývá silná například při použití zpěvové techniky, které se anglicky říká „belting“. Přesné popisy této techniky se v literatuře liší, ale v zásadě se jedná o použití silné opory

dechu o bránici k vyzpívání not ze středu rozsahu zpěváka plným tónem, který připomíná hrudní rejstřík. Tato metoda je velmi rozšířená mezi světovými populárními zpěvačkami.

Po spuštění algoritmu YIN na několika vlastních nahrávkách o známé periodě se ukázalo, že pokud se do signálu dostane silná vyšší harmonická složka, je velmi pravděpodobné, že YIN vybere periodu odpovídající právě jí namísto skutečné periody. Jelikož YIN byl původně navrhován a testován na řečových signálech, tak toto nebyl problém. Avšak v případě použití algoritmu YIN na zpěv, to již problém byl. Proto byl do algoritmu, implementovaného v rámci této práce, přidán výběr výsledné periody z více kandidátů. Jelikož problém bývá jen s druhou harmonickou složkou, stačí vzít jako kandidáty jen hodnoty proměnné *lag* odpovídající prvním dvěma lokálním minimům pozměněné auto-diferenční funkce, která se ocitnou pod prahem. Pokud žádná taková nejsou, použije se, stejně jako v původním algoritmu, *lag* odpovídající globálnímu minimu.

Máme-li dva kandidáty, tak vybereme prvního, pokud hodnota pozměněné auto-diferenční funkce pro druhý není o víc jak 0,1 nižší, než pro první. Pokud je nižší, tak vybereme druhého kandidáta. Tato podmínka vede k velkému zlepšení detekce periody pro signály se silnou druhou harmonickou složkou. Pro ty ostatní naopak vede k mírnému zhoršení v případě nedostatečné periodicity signálu. Proto bylo použití tohoto kroku parametrizováno a uživatel má možnost nastavit si efekt tak, aby co nejlépe vyhovoval zpěvové technice, kterou používá.



Obrázek 3.7. Hodnoty vrácené algoritmem pro nalezení periody ve třetí verzi, kdy byl přidán výběr ze dvou kandidátů. Červeně jsou hodnoty z předchozí verze, zeleně jsou nové hodnoty.

Na obrázku 3.7 je vidět pokles množství chyb způsobených výběrem druhé harmonické složky. Pro zvukovou ilustraci je k dispozici nahrávka *mic-2Harmonicka.wav* v adresáři „Přílohy\3.1.2\03 - 2Harmonicka“. Rozdíl je slyšitelný zejména v čase 2 s na slově „for“.

Návaznost v periodě

Jak bylo řečeno v kapitole 1.4, minimální doba potřebná ke změně výšky tónu o 4 tóny je pro lidský hlas zhruba 55 ms. Teoreticky bychom tedy mohli omezit rozsah přípustných výsledků pro jeden běh algoritmu na základě výsledků běhů předchozích. Neboli vrátí-li nám jeden běh algoritmu periodu 100 a následující běh vrátí periodu 200, víme, že došlo k chybě, protože změna o oktávu není v tak krátkém čase možná. Problém je, že neumíme rozhodnout, zda chyba nastala v předchozím, nebo současném běhu. Proto zavádíme další podmínky, které musí být splněny k tomu, aby byl odhad periody označen za chybný. Výběr periody na základě návaznosti se používá ve dvou případech:

- Žádná z hodnot pozměněné auto-diferenční funkce nespadá pod prahovou hodnotu a minimální hodnota pozměněné auto-diferenční funkce

na zkoumaném úseku signálu je vyšší než zvolená mez²⁴ a navíc *lag*, pro který pozměněná auto-diferenční funkce nabývá minima, není z povoleného rozsahu, daného výslednou periodou předchozího běhu programu. Pokud periody v posledních alespoň třech bězích navazovaly²⁵, použijeme jako výsledek periodu vrácenou předchozím během. Pokud periody nenavazovaly, budeme předpokládat, že zkoumaná část je aperiodická a obsahuje šum a vrátíme nulu.

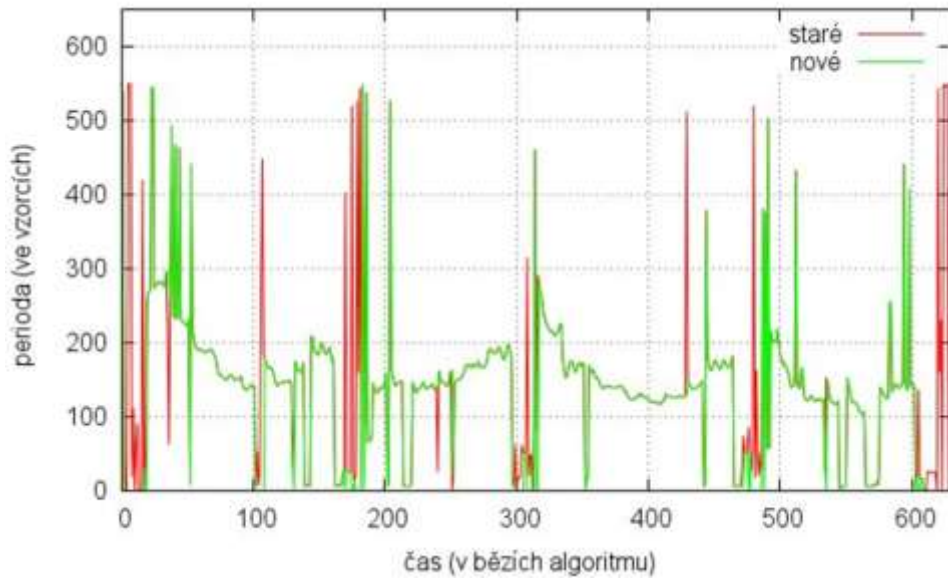
- Vybíráme ze dvou kandidátů. Pokud je některý z kandidátů z přípustného intervalu a navíc periody v posledních alespoň třech bězích navazovaly, vybereme tohoto kandidáta. Jinak kandidáty zpracujeme postupem popsaným v odstavci „Úprava třetího optimalizačního kroku algoritmu YIN – více kandidátů“.

Jelikož tím, že vybereme periodu z předchozího běhu nebo kandidáta, který na ni navazuje, zvýšíme uměle proměnnou, která počítá, kolik běhů bez přerušení navazovalo, mohlo by se stát, že bychom začali opakovat jednu periodu pořád dokola. Proto se v další proměnné uchovává informace o tom, kolikrát jsme si výběr periody vynutili popsanými pravidly. Změnu výsledku výběru periody na základě návaznosti povolíme jen dvakrát za sebou. Pokud by k ní mělo dojít potřetí v řadě, tak se místo toho všechny proměnné udržující informace o návaznosti v periodě napříč běhy algoritmu vynulují.

Z obrázku 3.8 je vidět, že došlo ke značné eliminaci omylů ve výběru periody, zejména v oblastech obsahujících šum, jelikož nyní nenutíme algoritmus vybírat *lag* odpovídající globálnímu minimu pozměněné auto-diferenční funkce i v případech, kdy je signál aperiodický.

²⁴ V současné implementaci má prahová hodnota velikost 0,1 a horní mez pro minimum je 0,5.

²⁵ To znamená, že nebyly vzdálené o víc, nežli přípustný interval.

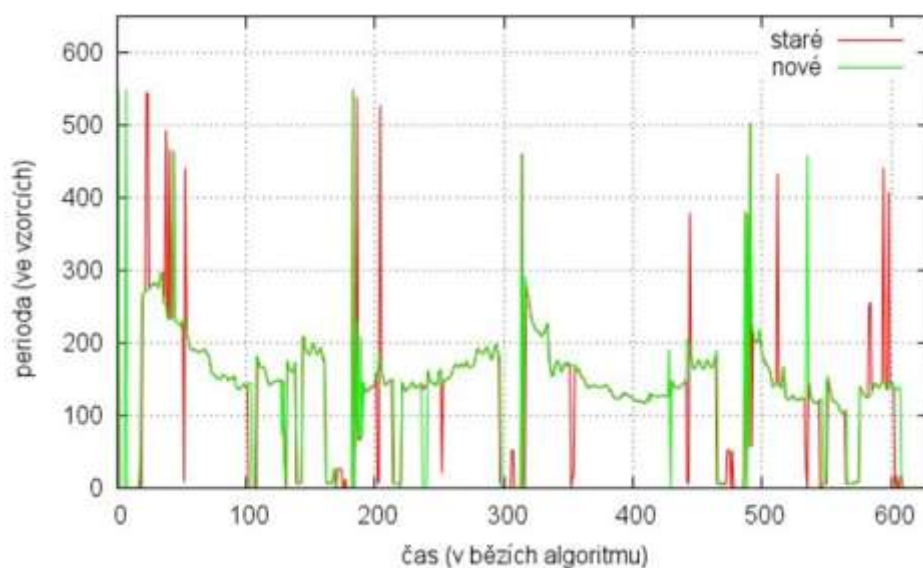


Obrázek 3.8. Hodnoty vrácené algoritmem pro nalezení periody ve čtvrté verzi, kdy byla přidána podmínka pro návaznost v periodě.

Dolní propust

Pokud signál obsahuje hodně šumu, je značně ztížena detekce periody. Proto byla na začátek samotného algoritmu pro odhad periody přidána dolní propust, která utlumí vysokofrekvenční složky a zlepší tak periodicitu signálu.

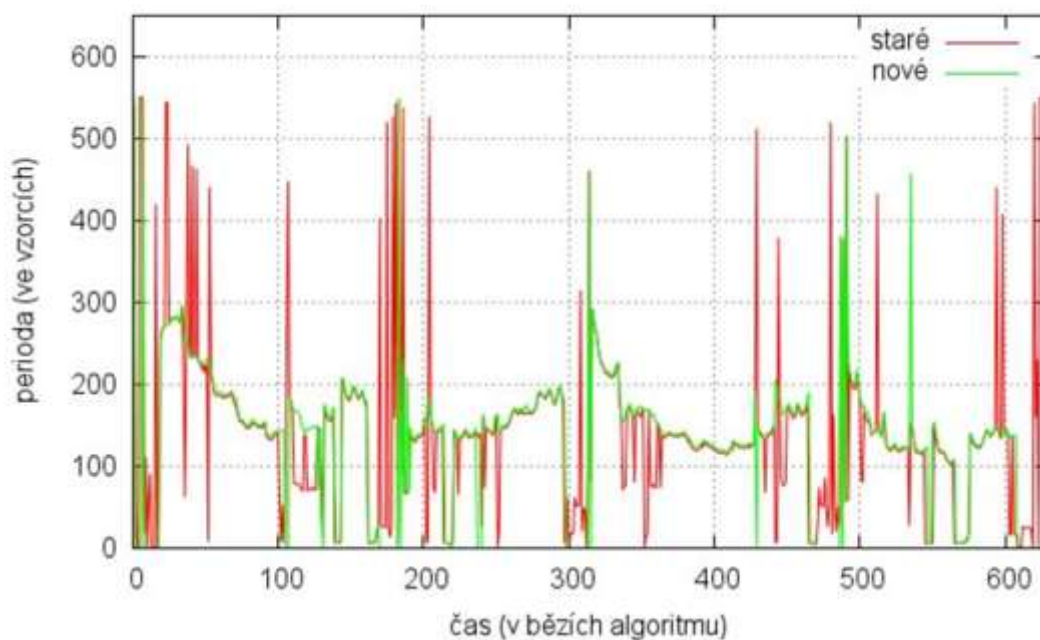
Jak je vidět z obrázku 3.9, po přidání dolní propusti klesl počet omylů v odhadu periody oproti předchozímu kroku a kvůli změně tvaru průběhu signálu naopak v několika málo případech chyba vznikla.



Obrázek 3.9. Hodnoty vrácené algoritmem pro nalezení periody v páté verzi, kdy byla přidána dolní propust.

Pro zvukovou ilustraci změny kvality výsledného zvuku je k dispozici nahrávka *mic-dolniPropust.wav* v adresáři „\Přílohy\3.1.2\05 - DolniPropust“. Změna je patrná zejména v první a čtrnácté vteřině nahrávky.

Obrázek 3.10 ilustruje rozdíl mezi neupravenou verzí algoritmu YIN z obrázku 3.5 a verzí po aplikaci všech dosud popsaných vylepšení.



Obrázek 3.10 Srovnání výsledků původní a vylepšené verze.

Mediánový filtr

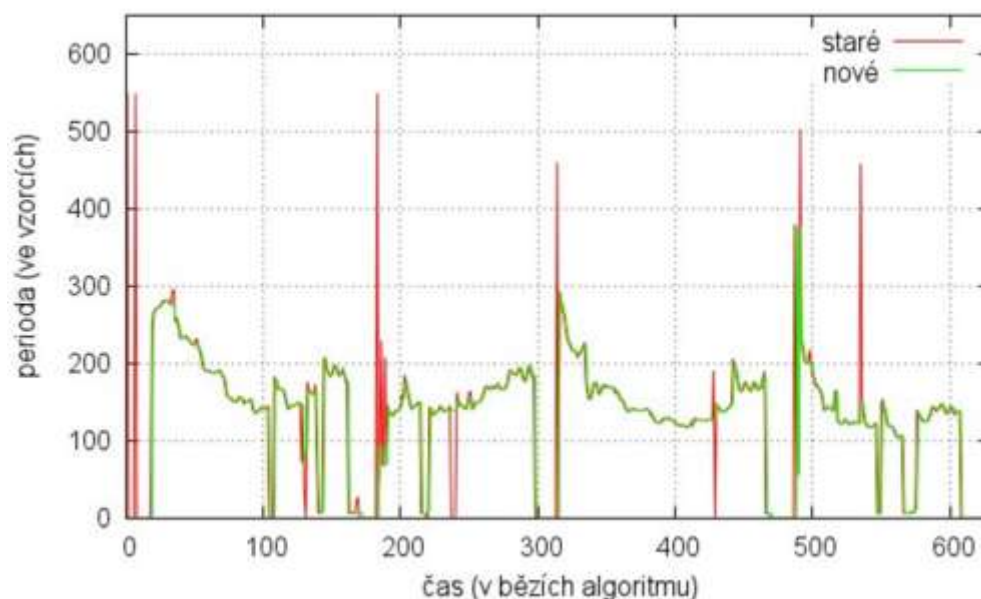
Stále ojediněle dochází k vybrání špatné periody. Takovéto chyby se dají omezit při použití mediánového filtru. Mediánový filtr si pamatuje výsledky posledních n běhů algoritmu pro určení periody a po přidání nové hodnoty vrátí hodnotu, která je mediánem hodnot z posledních n běhů. Použitím mediánového filtru dojde k vymizení ojedinělých omylů algoritmu pro určení periody, ale zároveň dojde k přidání určitého zpoždění v reakci na změny výšky tónu v signálu.

Použijme například mediánový filtr délky 3 a necht' obsahuje hodnoty: 168, 170, 166. Naposled vložená byla hodnota 166 a zároveň je i mediánem. Pokud nyní dojde ke změně výšky tónu – například perioda klesne na 100 – tak přidáme do filtru hodnotu 100 a zároveň dojde ke smazání nejstarší hodnoty 168 a jako medián se nám vrátí hodnota 166. Pokud v dalším běhu přijde hodnota 101, tak už se nám vrátí správně jako medián. Tudíž reakce na změnu výšky tónu je při použití mediánového

filtru délky 3 posunuta o jeden běh algoritmu pro určení periody. To už je nezanedbatelné a projeví se to sníženou kvalitou výstupního signálu. Nečekané výkyvy ve výšce tónu, způsobené občasným omylem algoritmu pro určení periody, jsou tak nahrazeny jakýmsi rozdvojením hlasu patrným zejména v místech, kde se mění výška tónu. Toto je možné slyšet na nahrávce *mic-medianovyFiltr.wav* umístěné v adresáři „\Přílohy\3.1.2\06 - MedianovyFiltr“.

Ve sborech je tento problém méně patrný, než oktávový omyl algoritmu pro učení periody. Proto je možnost použití mediánového filtru parametrizována a uživatel si ji může zapnout, pokud používá efekt ke generování sborů nebo vypnout, pokud používá efekt pro posunutí výšky jediného hlasu.

Obrázek 3.11 ilustruje vliv použití mediánového filtru délky 3. Je vidět, že rozptyl hodnot se snížil. Pořád jsou přítomny poklesy hodnot k nule, což je ovšem v pořádku, protože ty označují místa, kde zpěvák nezpívá nebo vyslovuje déle trvající souhlásku²⁶.



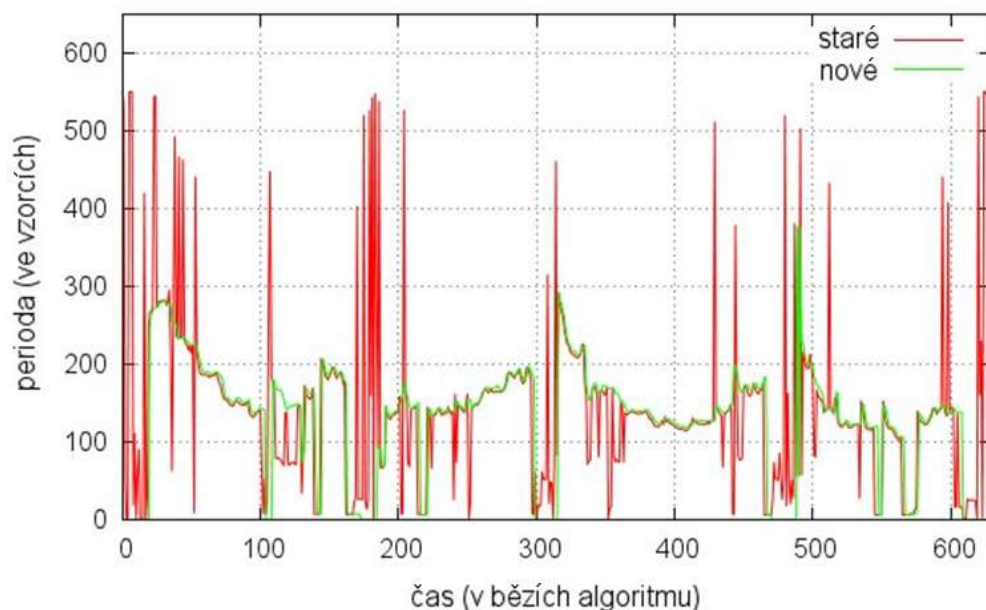
Obrázek 3.11. Hodnoty vrácené algoritmem pro nalezení periody v šesté verzi, kdy byl přidán mediánový filtr.

Závěr

U výsledného algoritmu dochází pro zpěvový signál k podstatně méně omylům, než u algoritmu původního. Některé změny provedené v algoritmu mohou

²⁶ Souhlásky mají stejný charakter jako šum, jsou neznělé.

za určitých podmínek přesnost odhadu periody naopak zhoršit. Proto jsou tyto kroky parametrizované a uživatel má možnost je obejít. Obrázek 3.12 ilustruje rozdíl mezi výsledky první a šesté verze.



Obrázek 3.12. Hodnoty vrácené algoritmem pro nalezení periody v první a šesté verzi.

Číselně nejlepší výsledky má šestá verze. Poslechově, subjektivně, zní nejlépe většinou pátá verze.

3.2 Posun výšky

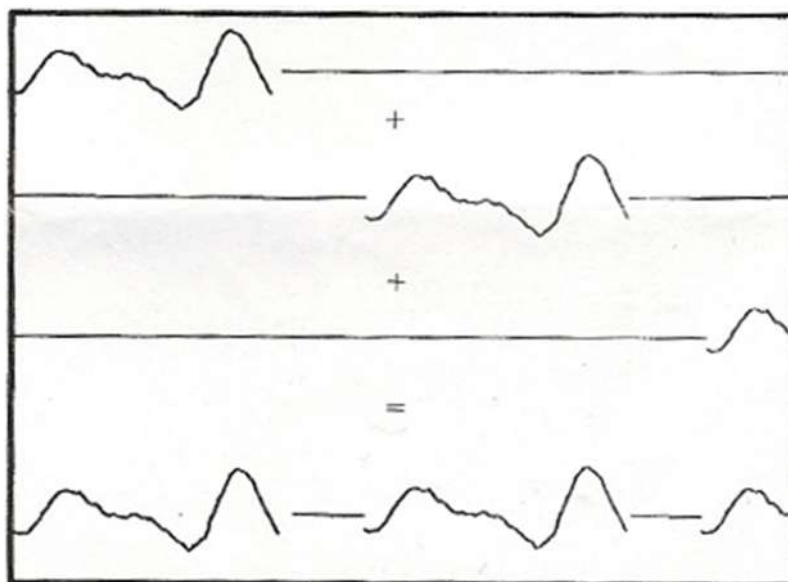
Pro samotnou změnu výšky tónu byla použita část Lentova algoritmu. Pomocí Lentova algoritmu je možné měnit jak výšku zvuku, tak jeho délku. V kapitole 3.2.1. si popíšeme část Lentova algoritmu, pro změnu výšky zvuku. V kapitole 3.2.3. se krátce podíváme, jak by bylo možné pomocí změny délky trvání zvuku upravovat formanty nezávisle na změně výšky zvuku.

3.2.1 Lentův algoritmus

Algoritmus navrhovaný K. Lentem v [7] je víceméně zjednodušená verze algoritmu PSOLA, popsaného v kapitole 2.1.3. Součástí Lentova algoritmu je navíc hledání periody v signálu, které ale v této práci bylo nahrazeno algoritmem YIN.

Stejně jako PSOLA, manipuluje Lentův algoritmus s jednotlivými periodami signálu a to tak, že pokud je potřeba výšku tónu snížit, periodu prodlouží a pokud je potřeba výšku tónu zvýšit, periodu zkrátí. Tím by se samozřejmě změnila i délka trvání zvuku, proto je potřeba některé periody vynechat nebo zopakovat.

Prodloužení periody provádí tak, že původní periodu doplní na novou velikost nulami. Zkrácení periody dosahuje tím, že nechá původní periody, aby se částečně překrývaly. Na rozdíl od převzorkování, popsaného v kapitole 2.1.1., tedy zachovává tvar původní periody téměř²⁷ beze změny a díky tomu se zachovávají formanty. Zájemcům o detailní rozbor Lentova algoritmu stran zachovávání formantů doporučuji k prostudování článek [9].

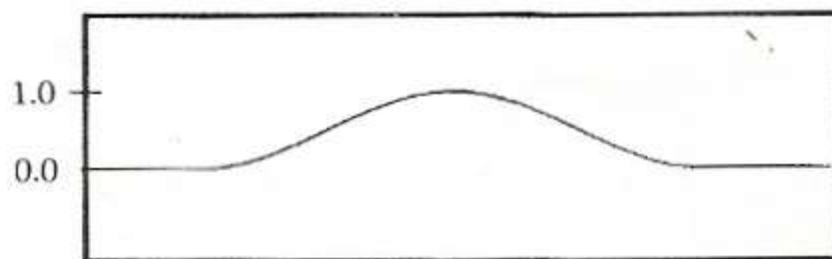


Obrázek 3.13. Jak by to vypadalo při snížení výšky tónu, pokud by se jednotlivé periody jen poskládaly za sebe. Převzato z [7].

Jak je vidět na obrázku 3.13, pokud by se perioda zvuku při prodloužení jen doplnila nulami, mohly by mezi úseky nul a původních period vzniknout velké nenávaznosti. Tím by se do výsledného signálu podle [7] přidaly nežádoucí vysokofrekvenční složky, které by značně poškodily kvalitu výsledného zvuku.

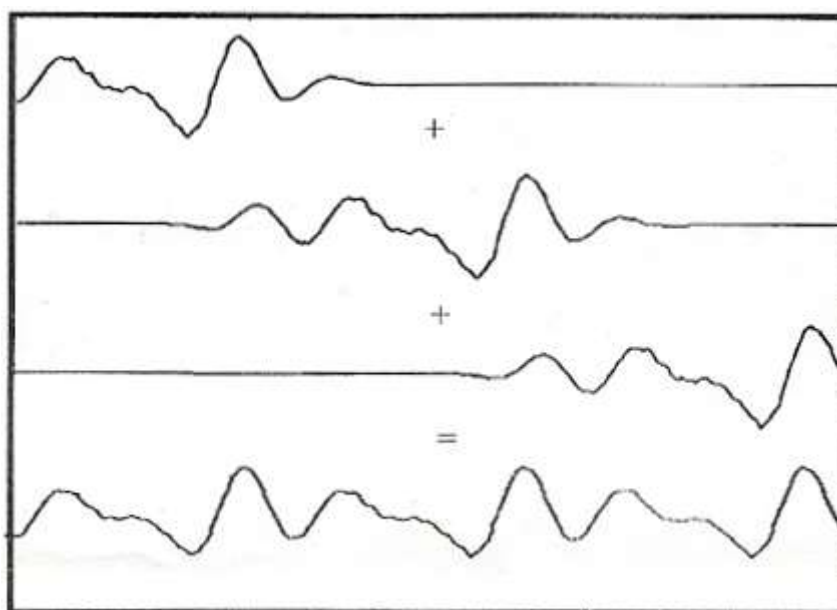
Tento problém se řeší v Lentově algoritmu, stejně jako u algoritmu PSOLA a mnoha dalších, použitím okénkové, konkrétně Hannovy, funkce.

²⁷ Ke změně dochází jen při zvýšení výšky tónu v důsledku částečného překryvu.



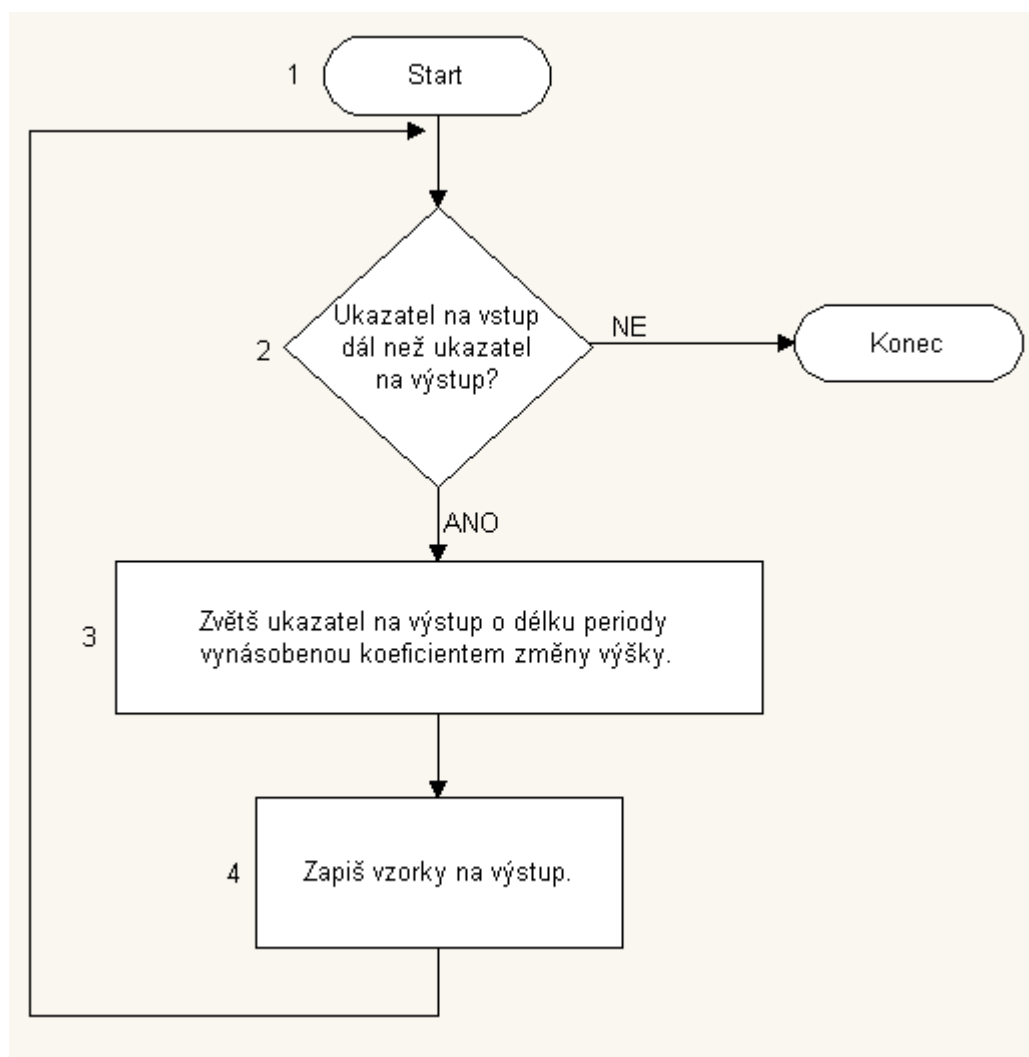
Obrázek 3.14. Tvar Hannova okna. Převzato z [7].

Při použití Hannova okna se ze vstupního signálu vždy vyjme úsek v délce dvou period. Přitom vzdálenost mezi začátky jednotlivých úseků je rovna délce jedné periody. Jinými slovy úseky se z poloviny překrývají. Díky symetričnosti Hannova okna ovšem platí, že pokud tyto úseky se stejným překryvem posčítáme opět dohromady, získáme zpátky původní signál, jak ilustruje obrázek 3.15. Na rozdíl od algoritmu PSOLA, bereme ze vstupního signálu úseky, aniž bychom je centrovali na lokální maximum periody.



Obr 3.15. Součet tří úseků vyjmutých ze vstupního signálu pomocí různě zpožděného Hannova okna. Převzato z [7].

Díky použití Hannova okna jsou hodnoty vyjmutých úseků na koncích rovny nule. Proto když prodlužujeme periodu, můžeme bez obav z nenávaznosti doplnit za takto vyjmutý úsek vzorky s nulovou hodnotou.



Obrázek 3.16. Zjednodušený vývojový digram části Lentova algoritmu pro zpracování jedné periody zvuku.

Diagram na obrázku 3.16 ukazuje průběh zpracování periody zvuku. Na začátku v **bloku 1** máme délku periody, vstupní pole vzorků zvuku s ukazatelem nastaveným na začátek nové periody a výstupní pole s ukazatelem nastaveným na začátek poslední zapsané periody a neobsazenými hodnotami nastavenými na nuly. V **bloku 2** testujeme, zda je ukazatel výstupního pole dál, než ukazatel vstupního pole. Pokud je, tak končíme. Pokud není, tak pokračujeme **blokem 3**, kde posuneme ukazatel na výstupní pole o délku periody vynásobenou koeficientem změny výšky tónu dopředu. Tudiž pokud výšku tónu snižujeme, bude výsledný posun větší než jedna perioda, pokud zvyšujeme, tak bude naopak menší. To vede k tomu, že v závislosti na velikosti posunu některé úseky délky jedné periody

zopakujeme nebo některé naopak přeskočíme. V **bloku 4** potom aplikujeme na vstupní signál Hannovo okno v délce dvou period vystředěné na ukazatel vstupního pole a výsledek přičteme k výstupnímu signálu s vycentrováním na ukazatel výstupního pole. Operace probíhající v bloku 4 by se daly zapsat jako:

```
for (int n = - perioda; n < perioda; n++) {  
    Výstup[n + výstupní_ukazatel] += Vstup[n + vstupní_ukazatel]  
        * Hann[n / perioda];  
}
```

3.2.2 O Implementaci

V této kapitole si popíšeme některá implementační rozhodnutí, vlastní vylepšení, vytvořené pomocné struktury a změny oproti algoritmu popsanému v 3.2.1.

Kruhová vyrovnávací paměť

K vnitřní reprezentaci vzorků pro vstup a výstup a pro usnadnění manipulace s nimi byla vytvořena třída implementující kruhovou vyrovnávací paměť. Součástí třídy je pole pevné velikosti a dva ukazatele – jeden pro čtení a jeden pro zápis. Mezi ukazatelem pro čtení a pro zápis může uživatel nastavit zpoždění maximálně o jedna menší, než je velikost vnitřního pole. Mimo jiné poskytuje třída implementující kruhovou vyrovnávací paměť metody pro přidání a vyzvednutí vzorků. Slovo kruhový značí, že pokud některý z ukazatelů dojde na konec vnitřního pole, je automaticky přenastaven na jeho začátek, aniž by se o to musel uživatel starat.

Třída starající se o posun výšky tónu používá dvě instance kruhových vyrovnávacích pamětí, jednu pro vstupní a jednu pro výstupní signál. Důvodem k použití kruhových vyrovnávacích pamětí je hlavně to, že potřebujeme pracovat i se vzorky z minulých period.

Použité kruhové vyrovnávací paměti pro vstupní a výstupní vzorky mají velikost vnitřního pole v násobcích maximální očekávané periody. Při vytvoření jsou vnitřní pole kruhových vyrovnávacích pamětí předvyplněna nulami.

Zpracování aperiodického signálu

Lentiv algoritmus posouvá výšku tónu všech částí signálu bez ohledu na velikost periody. To znamená, že se snaží posouvat výšku i tam, kde je jen šum.

Při zkracování periody se tedy může stát, že v původně aperiodické části signálu periodu uměle vytvoří. Jaký dopad to má na výsledný zvuk je možné slyšet na nahrávce *mic-chybneSykavky.wav*, kterou lze nalézt na přiloženém cd v adresáři „\Přílohy\3.2.2\01 - chybné sykavky\“. Zejména v části kolem 0:13 na slovech „christmas tree“ je to dobře slyšitelné.

Z tohoto důvodu byla do algoritmu přidána podmínka, která zajišťuje, že pokud je zjištěná perioda mimo rozsah lidského hlasu, zpracuje se daná část signálu beze změny délky periody.

Jak přidání této podmínky ovlivnilo kvalitu zvuku, je možné slyšet na nahrávce *mic-opraveneSykavky.wav*, která je k nalezení na přiloženém cd v adresáři „\Přílohy\3.2.2\02 - opravené sykavky\“.

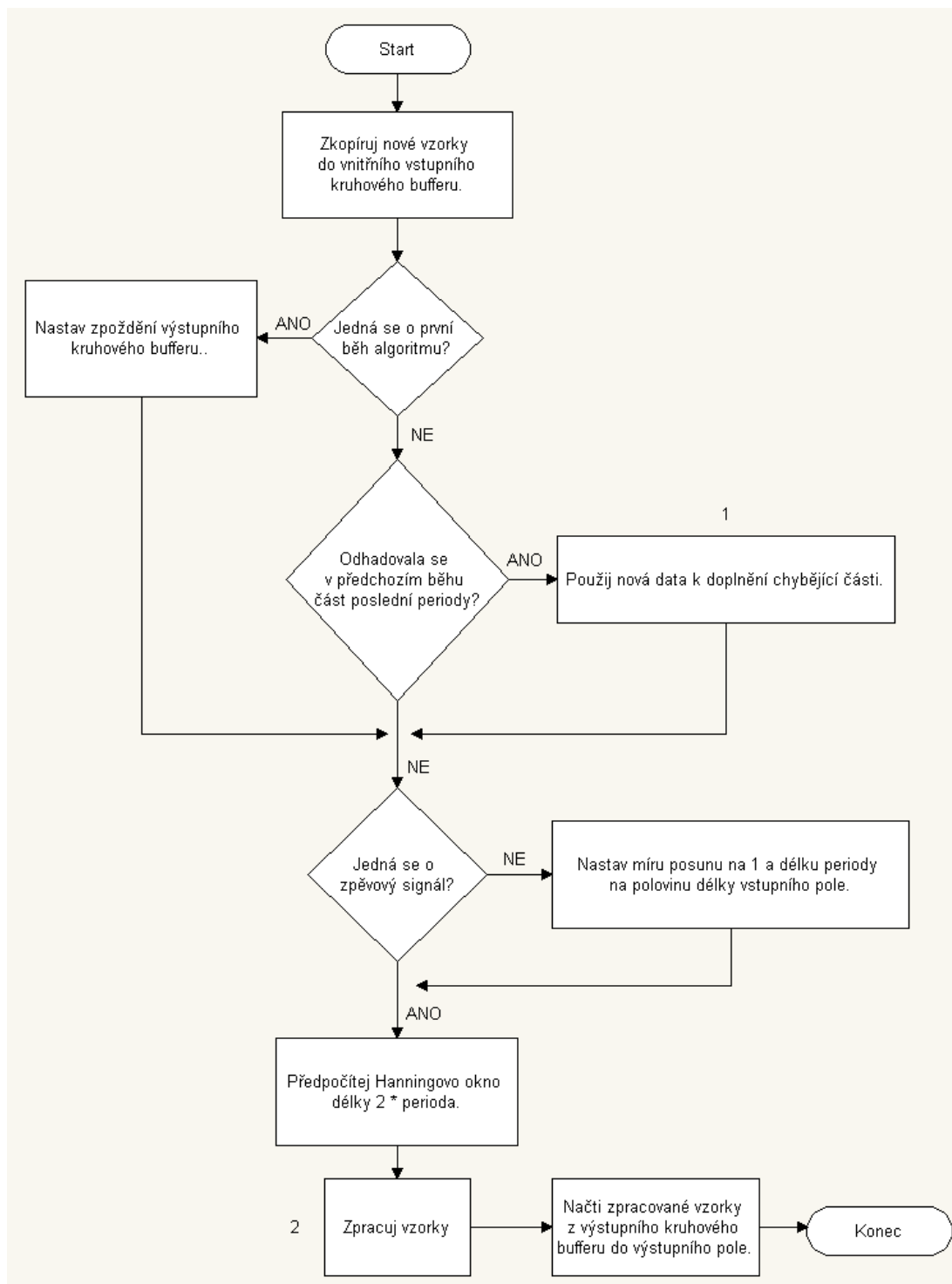
Snížení zpoždění

Na vstupu do algoritmu je pole vzorků v délce maximální očekávané periody. Není tedy nijak zaručeno, že velikost vstupního pole bude násobkem skutečné periody. Přitom Lentův algoritmus pracuje s celými periodami. Co se tedy stane se vzorky, které přebývají? Standardním řešením by bylo zpracovat je v dalším běhu algoritmu, až přijdou další data. Z jednoho běhu by nám tak mohlo zbyť nezpracovaných až *maximální očekávaná perioda – 1* vzorků. To by způsobilo pevné zpoždění v délce trvání jedné maximální periody, což je pro minimální frekvenci 80 Hz zhruba 12,5 ms. K tomu se ještě přidává další zpoždění velikosti 12,5 ms, způsobené tím, že Lentův algoritmus používá Hannovo okno v délce dvou period a potřebuje přistupovat k datům v délce jedné periody z předchozího běhu. Vzhledem k tomu, že dopředu velikost periody neznáme, musíme předpokládat, že může mít délku až 12,5 ms.

Celkově by se nám zpoždění způsobené algoritmem pro posun výšky tónu nasčítalo na 25 ms a to je hodně. Za cenu mírné nepřesnosti je možné zpoždění, způsobené přebývajícimi vzorky, snížit.

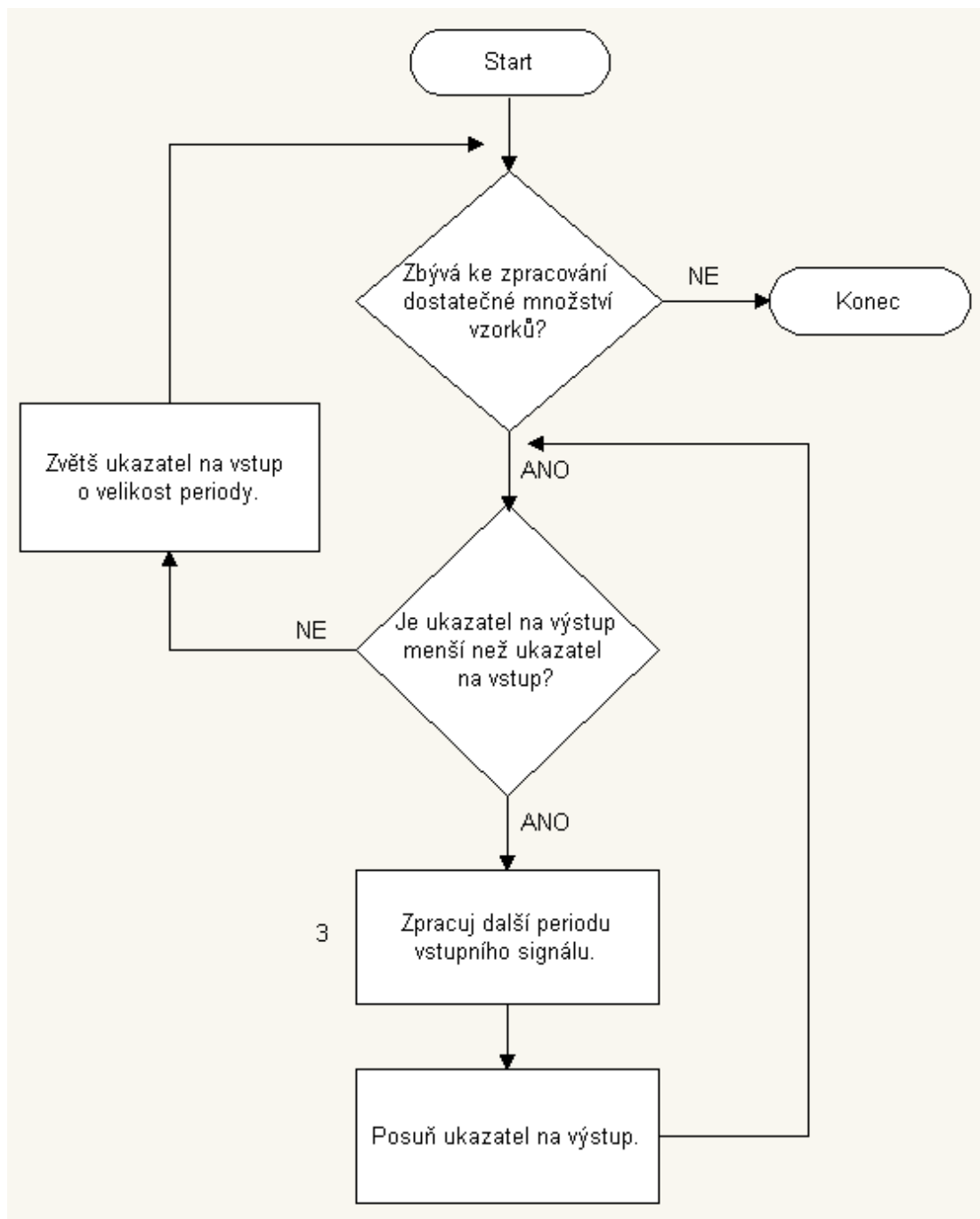
Pokud nám přebývají nějaké vzorky, doplníme je do délky celé periody daty z předchozí periody. Za předpokladu, že velikost periody byla algoritmem pro periody odhadnuta správně, budou data dobře navazovat. Takto uměle vytvořenou periodu potom zpracujeme, jako všechny ostatní. Tím se může snížit zpoždění způsobené přebývajícimi vzorky až na nulu. Tudíž celkové zpoždění algoritmu pro posun výšky tónu je potom 12,5 ms.

Maximální procentuelní množství kopírovaných vzorků je navíc parametrizované, a tak si uživatel, pokud nepracuje v reálném čase, může nastavit počet kopírovaných vzorků na 0% a získat tak maximální přesnost za cenu vyššího zpoždění.



Obrázek 3.17. Vývojový digram algoritmu pro posun výšky tónu. Zpracování jednoho okna signálu.

Vývojový diagram na obrázku 3.17 popisuje zpracování vzorků při posunu výšky zvuku. Se snižováním zpoždění prostřednictvím doplňování period souvisí ještě **blok 1**. Pokud se v předchozím běhu algoritmu doplňovala nějaká perioda, tak aby se odhadnutá data nemíchala s novými daty, na která by nemusela úplně dobře navazovat, nahradí se v poslední zapsané periodě uměle doplněná část správnými, nově příchozími daty. Jelikož se nahrazení děje jen pro poslední periodu, nepřidá se tím žádné zpoždění.



Obrázek 3.18. Diagram detailu bloku 2 z obrázku 3.16.

Lineární interpolace

V **bloku 3** diagramu z obrázku 3.18 zpracováváme dvě periody signálu. Novou a tu, která jí předcházela. Zjednodušený kód bloku 5 by mohl vypadat jako:

```
for (int n = -perioda; n < perioda; n++) {
    if ((n - zbývá) < 1) {
        vstupni_pointer -= perioda;
    }
    vzorek = vstupni_kruhova_pamet[vstupni_pointer + n] *
        Hann[n + perioda];
    vystupni_kruhova_pamet[floor(vystupni_pointer) + n] = koef[0]
        * vzorek;
    vystupni_kruhova_pamet[floor(vystupni_pointer) + n + 1] =
        koef[1] * vzorek;
}
```

Kde *perioda* je délka právě zpracovávané periody ve vzorcích, *zbývá* je počet vzorků zbývajících ke zpracování ve vstupním poli, *Hann* je pole předpočítaných hodnot Hannova okna a *koef* je pole koeficientů pro lineární interpolaci. Tu používáme, jelikož ukazatel na výstup nemusí být celočíselný, a proto pro zvýšení přesnosti hodnotu vzorku ze vstupu rozdělíme mezi dva výstupní vzorky.

Závěr

Díky snížení zpoždění na polovinu, je možné použít algoritmus pro běh v reálném čase. Dále tím, že neměníme úseky, které neobsahují zpěvový signál, a přidáním lineární interpolace, bylo dosaženo mírného zlepšení v kvalitě zvukového výstupu. Maximální doporučená míra posunu pro hlas je +/- oktáva. Zvláště při posunu směrem dolů, jsou na signálu slyšet artefakty způsobené prodloužením periody doplněním nulami. Ukázky posunu je možné nalézt na příloženém cd v adresáři „\Přílohy\3.2.2\04 – závěr\“. Kromě ukázek na nahrávce, na které jsme si zatím demonstrovali všechny vývojové fáze efektu, je v adresáři k nalezení ještě ukázka posunu na vyčištěné studiové nahrávce. Originál nahrávky je k dispozici v adresáři „\Přílohy\“ pod názvem *studio.wav*.

3.2.3 Jak lze pomocí Lentova algoritmu měnit formanty

Tato kapitola je krátkým popisem možného rozšíření efektu do budoucna. Nezávislá změna formantů nebyla součástí zadání této práce, proto tento postup nebyl ve výsledném efektu implementován, nicméně jeho funkčnost byla ověřena

na prototypu. Pokud čtenáře zajímá jen současný stav efektu, může pokračovat kapitolou 3.3.

Jak jsme se již krátce zmínili v úvodu kapitoly 3.2, je možné pomocí Lentova algoritmu měnit nejen výšku zvuku, ale i jeho délku. Toho by se dalo docílit jednoduchou úpravou podmínky v rozhodovacím **bloku 2**, diagramu na obrázku 3.16. Stačilo by tam porovnávat, zda je ukazatel na výstup menší než ukazatel na vstup vynásobený koeficientem změny délky. Pokud bychom tedy chtěli signál například o polovinu prodloužit, byl by koeficient délky roven 1,5. Úpravou této podmínky bychom dosáhli toho, že by bylo možné měnit výšku zvuku nezávisle na jeho délce.

Jak tedy změnit formanty nezávisle na změně výšky zvuku?

V kapitole 2.1.1. jsme si ukázali, jak změnit délku trvání zvuku zároveň se změnou jeho výšky a formantů pomocí převzorkování. Odpovědí tedy je spojit převzorkování s Lentovým algoritmem.

Před vstupem do části Lentova algoritmu, která posouvá délku a výšku zvuku, by se musela každá perioda zvuku převzorkovat ve zvoleném poměru. Následkem převzorkování by došlo ke zkrácení nebo prodloužení periody. Před vstupem do Lentova algoritmu by stačilo upravit délku periody na novou hodnotu a nastavit koeficient pro změnu délky zvuku v opačném poměru, než v jakém jsme převzorkovali a ke koeficientu pro změnu výšky tónu přičíst posun způsobený převzorkováním. Tím bychom se dostali zpět na původní délku, původně zamýšlený posun ve výšce, ale zároveň přidaný posun formantů.

Formanty tímto způsobem nelze měnit libovolně. Prakticky takovouto změnou formantů lze simulovat jen změnu velikosti rezonančních dutin a délky hrdla. Tudíž při nadzvorkování by zvuk zněl, jak by ho zpíval někdo větší než původní zpěvák a při podzvorkování naopak jako by zpíval někdo menší.

3.3 Implementace řešení jako modulu

Součástí zadání bylo implementovat efekt jako modul, který na základě objektového rozhraní spolupracuje se standardním nahrávacím software.

Mezi nejrozšířenější typy modulů pro nahrávací programy patří moduly založené na VST (Virtual Studio Technology). Dají se upravit pro použití na Windows, Mac OS X i Linux, nicméně většinou jsou používány hlavně pro Windows. Pro Mac OS X se častěji používá formát AU (Audio Units).

Algoritmus popsany v předchozích kapitolách byl nakonec implementován jako VST modul pro Windows ve formě dynamicky linkované knihovny (DLL). Pro tvorbu modulu bylo použito VST SDK od firmy Steinberg Media Technologies ve verzi 2.4 a šablona pro vytváření VST efektů od doktora Jiřího Schimmela z Vysokého Učení Technického v Brně ve verzi 1.2.

VST SDK, šablona i výsledný efekt je psán v jazyce C++. Pro vývoj efektu bylo použito vývojové prostředí Visual Studio Express 2008.

3.3.1 Technologie VST

VST je rozhraní umožňující integraci softwarových audio efektů a syntezátorů s hudebními editory. Toto rozhraní bylo vyvinuto firmou Steinberg, která je ve světě počítačového zpracování hudby známá hlavně díky hudebnímu editoru Cubase. Softwarovým audio efektům využívajícím toto rozhraní se říká VST moduly nebo pluginy.

Existují tři druhy VST modulů:

- VST instrument, který hudbu generuje – například různé syntezátory nebo samplery.
- VST MIDI efekt, který zpracovává MIDI zprávy.
- VST efekt, který zpracovává zvukový vstup.

K tomu, aby bylo možné modul používat, je zapotřebí hostitelská aplikace – takzvaný VST host. Mezi VST hosty v současné době patří většina profesionálních i poloprofesionálních aplikací určených pro práci s hudbou. VST host se mimo jiné stará o komunikaci se zvukovou kartou a MIDI ovladači a získaná data potom dává

k dispozici VST modulu, který s nimi může manipulovat. VST modul je tedy komponenta, která se stará čistě o zpracování zvukových dat, nikoliv o jejich správu.

VST moduly musí zpravidla pracovat s tím, co jim VST host poskytne. To znamená, že pod některými VST hosty nemusí fungovat všechny funkce. Také záleží jen na hostovi, jaká se použije vzorkovací frekvence, kolik kanálů budou mít data posílaná do modulu nebo po kolika vzorcích najednou je bude modulu posílat. Modul se tedy musí přizpůsobit stávajícím podmínkám a nemůže spoléhat na nějaké konkrétní nastavení.

VST SDK²⁸ [28] je soubor C++ tříd stavících na C API²⁹, které zjednodušují vývoj VST modulů i VST hostů. VST SDK zjednodušuje vývoj modulů do takové míry, že v jednodušších případech stačí pro výrobu efektu podědit jednu třídu a implementovat těla několika málo metod. Takový efekt nebude mít žádné parametry nebo uživatelské rozhraní. Pro vývoj grafického uživatelského rozhraní efektu potom slouží knihovny z VSTGUI³⁰.

Zdrojový kód VST modulu je nezávislý na platformě, ale typ výsledného souboru závisí na architektuře platformy. Na Windows je VST modul dynamicky linkovanou knihovnou, na Mac OS X je VST modul bundlem a na BeOS a SGI (pod MOTIF, UNIX) je VST modul sdílenou knihovnou.

Oficiální dokumentaci k VST SDK a VSTGUI lze nalézt na přiloženém cd v adresáři „\Text a dokumentace\Dokumentace VST SDK\“.

Všechny typy modulů musí dědit ze třídy *AudioEffectX* a podle typu modulu musí implementovat některé z jejích metod. Modul implementovaný v rámci této práce je VST efektem a nejdůležitější z metod, které musí implementovat, se jmenuje *ProcessReplacing*. Tuto metodu volá host vždy, když se mu naplní vyrovnávací paměť novými daty.

²⁸ Virtual Studio Technology Software Development Kit

²⁹ C Application Programming Interface

³⁰ Virtual Studio Technology Graphic User Interface

3.3.2 VST šablona

VST šablona [29] od doktora Jiřího Schimmela staví na VST SDK a dále usnadňuje tvorbu VST modulů. Hlavním důvodem pro použití šablony byla skutečnost, že šablona umožňuje vytvořit jednoduché grafické uživatelské rozhraní pro efekt, které se jinak přes VSTGUI vytváří složitě. Díky této šabloně je možné zobrazit různé ovládací prvky ve formě jednoduchých posuvníků, což je pro demonstraci algoritmu postačující. Nevýhodou šablony je to, že umožňuje použití popisků jen velmi omezené délky a rozložení prvků lze ovlivnit pouze nastavením počtu povolených řádků. Pokud je potřeba složitější GUI, je nutné pracovat přímo s VST SDK a VSTGUI.

Šablona je připravena k okamžitému použití. Kromě tříd samotné šablony obsahuje balíček, ve kterém je šablona dodávána, i potřebné třídy z VST SDK, grafiku pro GUI a projekt pro Visual Studio 2005. Po načtení projektu do Visual Studia stačí dát „Build->Build Solution“, zvolit název řešení a dojde k vytvoření souboru dynamicky linkované knihovny, který reprezentuje jednoduchý VST efekt, umožňující měnit hlasitost zvukového signálu. Odpadá tedy zdlouhavé nastavování vlastností projektu. Jediné, co je potřeba ve vlastnostech projektu nastavit, je cesta k VST hostu, ve kterém si přejeme efekt ladit a v závislosti na nastavení hostitelského programu i cesta, kam se má soubor dynamicky linkované knihovny vygenerovat.

Ve větším detailu je tvorba VST modulu s pomocí šablony popsána v dokumentaci šablony, která je k dispozici na příloženém cd v adresáři „\Text a dokumentace\Dokumentace VST šablony\“ pod názvem *VST template.pdf*.

3.3.3 Ovládání pomocí MIDI

Součástí zadání práce bylo pokusit se vytvořit jednoduchý systém pro ovládání míry posunu pomocí MIDI.

VST efekt nemůže běžně přijímat MIDI zprávy, aby toto bylo možné, je potřeba nastavit několik příznaků, které VST hosta informují o tom, že má efektu MIDI zprávy přeposílat. Bohužel některé hostitelské aplikace toto nastavení ignorují.

Pokud host umožní efektu přijímat MIDI zprávy, pamatuje si efekt seznam právě hraných MIDI not a podle počtu aktivních hlasů používá poslední hrané noty k řízení míry posunu výšky tónu pro jednotlivé hlasy. Míra posunu se určí jako rozdíl mezi hranou notou a základní frekvencí v signálu. Pro tento účel si efekt drží tabulku pro převod mezi MIDI notami a frekvencemi. Pro generování tabulky se využívá skutečnosti, že MIDI nota 0 odpovídá frekvenci 8.1757989156 Hz a frekvence pro notu o půl tónu vyšší se v rovnoměrně temperovaném ladění získá jako:

$$f_{nova} = f_{stara} * \sqrt[12]{2}$$

Kde f_{nova} je o půl tónu vyšší než f_{stara} .

Výsledný koeficient posunu se potom určí jako:

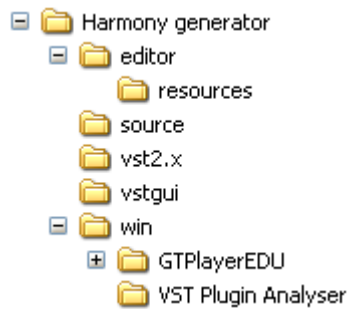
$$koeficient\ posunu = f_{noty} / f_{zpěvu}$$

Víceméně tedy dojde k tomu, že nové hlasy budou mít výšku odpovídající hraným notám. Jediné omezení je, že rozdíl mezi hraným a zpívaným tónem nesmí být větší než oktáva, jelikož to je maximální povolená míra posunu.

Vylepšením by mohlo být načítat ze zpráv i údaje o rychlosti stisku klávesy a podle nich nastavovat hlasitost jednotlivých hlasů. Na druhou stranu by to mohlo vést k nechtěným výkyvům v hlasitosti hlasů.

3.3.4 Popis projektu

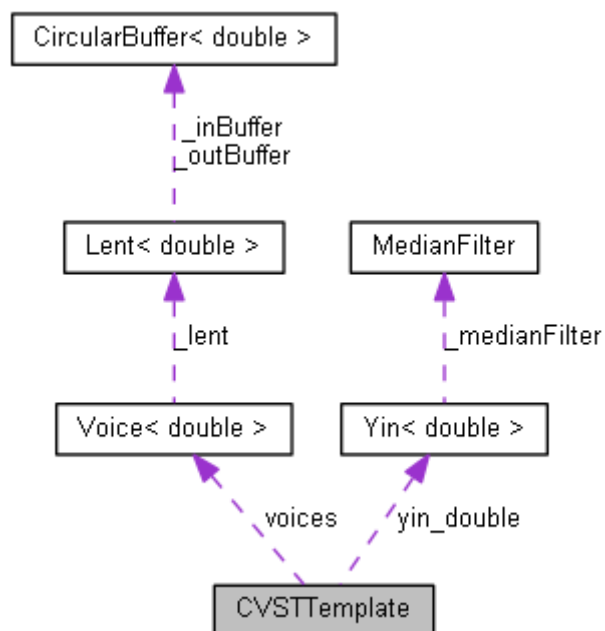
Generátor harmonií vznikl rozšířením projektu VST šablony. Přibylo několik nových tříd a část stávajících byla upravena. Součástí projektu jsou i třídy z VST SDK, které šablona používá. Zdrojové soubory včetně projektu pro Visual Studio 2008 lze nalézt na příloženém cd v adresáři „\Projekt\“.



Obrázek 3.19. Adresářová struktura projektu Harmony generator.

Popis obsahu adresářů:

- „vst2.x“ a „vstgui“ obsahují třídy z VST SDK. Tyto třídy by nikdy neměly být měněny.
- „editor“ obsahuje třídu a zdroje pro grafické uživatelské rozhraní. Je součástí šablony.
- „source“ obsahuje zdrojové kódy šablony, které byly určeny pro manipulaci uživatelem a nové třídy vytvořené pro generátor harmonií.
- „win“ obsahuje ladící programy dodávané se šablonou a projekt efektu upravený pro Visual Studio 2008.



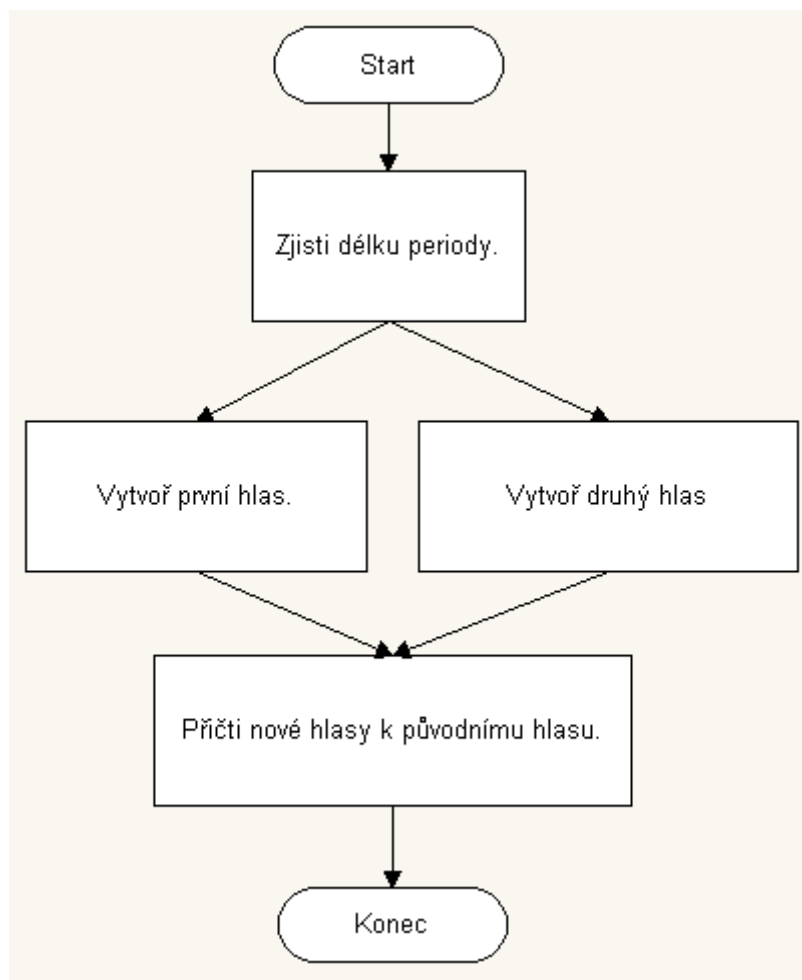
Obrázek 3.20. Diagram nových a změněných tříd projektu. Vygenerováno pomocí nástroje Doxygen.

Popis tříd:

- „CVSTTemplate“ je hlavní třídou šablony. Dědí ze třídy *AudioEffectX*, která patří do VST SDK. Přes metody v této třídě komunikuje efekt s VST hostem.
- „Yin“ implementuje algoritmus pro hledání periody v signálu popsany v kapitole 3.1.
- „MedianFilter“ implementuje mediánový filtr popsany v kapitole 3.1.
- „Lent“ implementuje algoritmus pro posun výšky tónu popsany v kapitole 3.2.
- „CircularBuffer“ implementuje pomocnou strukturu pro reprezentaci vyrovnávací paměti, popsanou v kapitole 3.2.2.
- „Voice“ reprezentuje jeden přídavný hlas. Má vlastní instanci třídy pro posun výšky tónu, pamatuje si údaje o míře posunu, hlasitosti a dalších parametrech vztahujících se ke každému hlasu.

Vygenerovanou dokumentaci zdrojového kódu lze nalézt na přiloženém cd v adresáři „\Text a dokumentace\Doxygen - dokumentace kódu“. Dokumentaci lze zobrazit v internetovém prohlížeči po otevření souboru *index.html*.

Díky tomu, že algoritmy popsány v kapitolách 3.1.2 a 3.2.2 byly implementovány v podobě C++ tříd, je jednoduché přidat další hlasy. Diagram na obrázku 3.21 ilustruje zjednodušený průběh signálu efektem.



Obrázek 3.21. Zjednodušený vývojový diagram efektu.

3.4 Popis výsledného efektu

Výsledný efekt umožňuje vytvoření až dvou hlasů s výškou lišící se až o oktávu od původního hlasu. Díky optimalizacím popsaným v kapitolách 3.1.2 a 3.2.2 bylo zpoždění vytvářené efektem mezi vstupním a výstupním signálem sníženo z původních 50 ms na 25 ms a efekt je tedy možné použít v reálném čase. Detekce periody v signálu prošla řadou vylepšení a je nyní schopná pracovat i s mírně zašuměnými signály a signály se silnou druhou harmonickou složkou. Důsledkem zakázání opakování aperiodických částí signálu došlo ke zlepšení kvality zpracování sykavek. Efekt je implementovaný jako VST modul a proto je ho možné použít s velkým množstvím různých aplikací pro práci se zvukem. Díky objektově orientovanému přístupu je možné efekt v budoucnu snadno rozšiřovat.

3.4.1 Ovládání efektu

Efekt má ovládací prvky rozdělené do 4 sloupců, postupně si popíšeme prvky v každém z nich.

V prvním sloupci, který je vidět na obrázku 3.22 jsou ovládací prvky týkající se nastavení celého efektu a původního hlasu.

- Posuvník „Bypass“ nabývá hodnot „On“ a „Off“. Pokud je nastavený na „On“, tak signál efektem prochází beze změny. Při zapnutí a opětovném vypnutí funkce „Bypass“ dojde k resetování dat v efektu.
- Posuvník „Gain“ nastavuje hlasitost původního hlasu v procentech.
- Posuvník Mute nabývá hodnot „On“ a „Off“. Pokud je nastavený na „On“, tak je původní hlas umlčený.



Obrázek 3.22. První sloupec ovládacích prvků efektu.

Ovládací prvky ve druhém a třetím sloupci mají ten samý význam, jen každý sloupec ovládá jiný hlas. Proto si popíšeme jen ten druhý, který je na obrázku 3.23.

- Posuvník „Pitch 1“ nastavuje míru posunu výšky prvního hlasu v půltónech. Rozsah má od -12 půltónů do +12 půltónů.
- Posuvník „Gain 1“ nastavuje hlasitost prvního hlasu v procentech.
- Posuvník „Mute 1“ nabývá hodnot „On“ a „Off“. Pokud je nastavený na „On“, tak je hlas umlčený.



Obrázek 3.23. Druhý sloupec ovládacích prvků efektu.

Ovládací prvky ve čtvrtém sloupci slouží k nastavení parametrů algoritmu pro hledání periody v signálu. Všechny nabývají hodnot „On“ a „Off“, jak ilustruje obrázek 3.24.

- Posuvník „Median“ zapíná a vypíná použití mediánového filtru. Zapnutí mediánového filtru se hodí v případech zašuměného nebo jinak poškozeného signálu, ale doporučuje se jen pro generování sborů, nikoliv pro posun sólového hlasu, protože na hlase vytváří artefakty znějící jako rozdvojení hlasu.
- Posuvník „Lowpas“ zapíná a vypíná použití filtrování dolní propustí. Zlepšuje kvalitu detekce pro zašuměné signály, pro většinu signálů vede ke zlepšení přesnosti detekce.
- Posuvník „2nd H.“ zapíná a vypíná kontrolu přítomnosti silné druhé harmonické složky v signálu. Hodí se zejména při zpěvu se silnou oporou bránice, kdy dochází k vyšší rezonanci v rezonančních dutinách. Po změně nastavení posuvníku je nutné resetovat efekt. Pro resetování efektu zapněte a vypněte funkci „Bypass“.



Obrázek 3.24. Čtvrtý sloupec ovládacích prvků efektu.

Pátý sloupec obsahuje jen dva ovládací prvky, jak je vidět na obrázku 3.25.

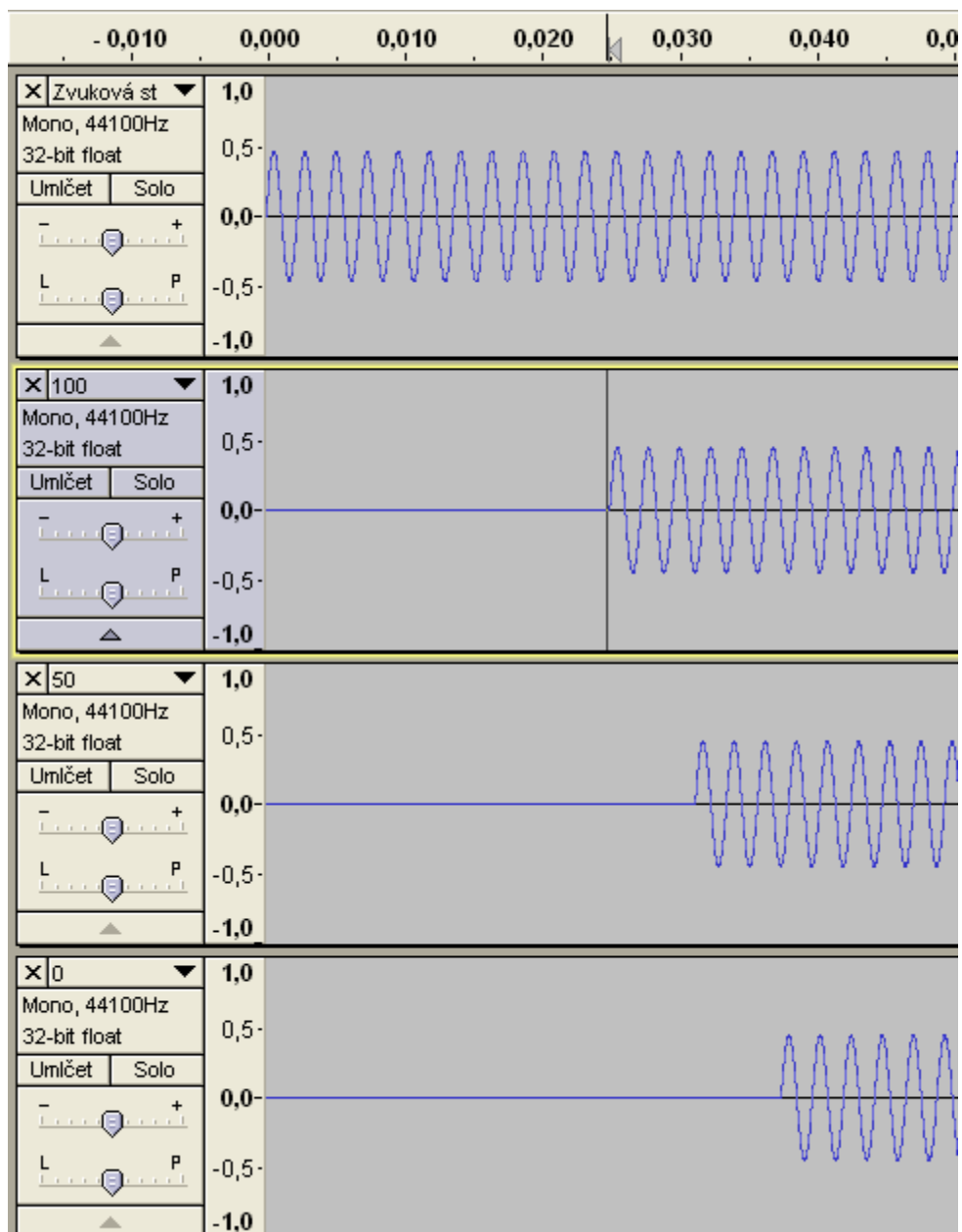
- Posuvník „Aprox.“ nabývá hodnot od 0% do 100% a ovlivňuje maximální počet vzorků, které se v algoritmu pro posun výšky zvuku budou doplňovat do neúplné periody. Pokud je nastavený na 0% tak se jakákoliv data zbylá z jednoho běhu algoritmu přenáší do dalšího běhu a zpoždění generované efektem tím stoupne o 12,5 ms. Pokud je posuvník nastavený na 100%, tak se použijí v každém běhu všechny vzorky a zpoždění generované efektem zůstává na celkových 25 ms. Pro použití v reálném čase je doporučeno mít posuvník nastavený na 100%, jelikož to zajišťuje nejnižší latenci. Nastavení 0% se zase dá použít pro maximální přesnost efektu. Pro většinu signálů je ale rozdíl v kvalitě téměř neznatelný.

Obrázek 3.25 ilustruje vliv nastavení posuvníku na velikost zpoždění. Po změně nastavení posuvníku je nutné resetovat efekt, jelikož je potřeba přepočítat velikost některých vyrovnávacích pamětí. Pro resetování efektu zapněte a vypněte funkci „Bypass“.

- Posuvník „Midi“ zapíná a vypíná ovládání pomocí MIDI kontroleru. Při zapnutém MIDI ovládání efekt určuje míru posunu jednotlivých hlasů z posledních dvou hraných not. Hlasy potom absolutně posouvá na hrané noty, pokud je míra posunu v rozsahu jedné oktávy. Pokud je rozdíl mezi zpívanou a hranou notou větší, než jedna oktáva, tak efekt posune hlas na stejnou notu, ale z oktávy ve které je nota zpívaná.



Obrázek 3.25. Pátý sloupec ovládacích prvků efektu.



Obrázek 3.26. Vliv nastavení posuvníku „Aprox“ na zpoždění generované efektem. V první stopě je původní signál, v druhé stopě je signál z efektu s posuvníkem „Aprox“ nastaveným na 100%, ve třetí stopě je signál z efektu s posuvníkem „Aprox“ nastaveným na 50% a v poslední stopě je signál z efektu s posuvníkem „Aprox“ nastaveným na 0%.

Návod na použití efektu s různými VST hosty lze nalézt na přiloženém cd v adresáři „\Text a dokumentace\“ pod názvem *Návod na použití.pdf*.

4 Porovnání s existujícími řešeními

Efekt umožňuje vytvářet harmonie, nejsložitějším úkolem práce ovšem bylo samotné sestavení algoritmu pro změnu výšky hlasu se zachováním formantů. Proto budeme efekt s existujícími řešeními porovnávat na základě schopnosti změnit výšku hlasu a ne na základě schopnosti vytvářet harmonie.

Existující řešení se dají rozdělit do několika kategorií.

- 1) Profesionální knihovny a aplikace pro posun výšky hlasu neimplementované ve formě modulů
- 2) VST efekty pro posun výšky hlasu
 - a. Placené
 - b. Volně dostupné

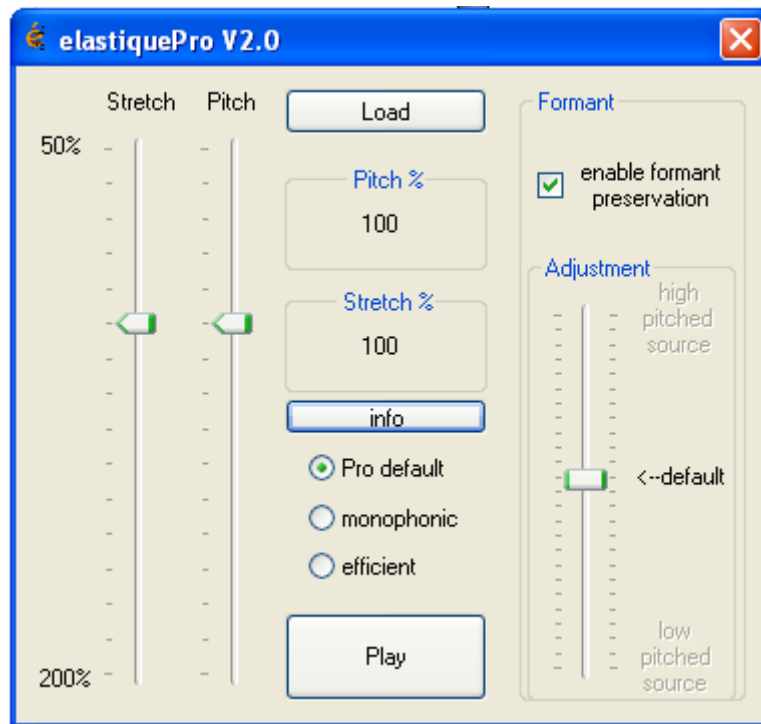
Jelikož kvalita zvuku se nejlépe určí poslechem a její hodnocení je tedy do značné míry subjektivní, bylo každé prezentované řešení otestováno na oktávových změnách na souborech *mic.wav* a *studio.wav*, které lze najít na přiloženém cd v adresáři „\Přílohy\“. Mnoho z testovaných řešení obsahuje kromě samotného posunu zvuku ještě další funkce, jako je možnost přidání ozvěny, nezávislé změny formantů a další. Funkce, které nejsou porovnatelné s řešením implementovaným v rámci této práce, nebudou testovány.

Jak si s posuny poradilo řešení představené v kapitole 3, je možné slyšet na nahrávkách, které se nalézají v adresáři „\Přílohy\4\Vlastní řešení\“ na přiloženém cd.

4.1 Profesionální knihovny a aplikace pro posun výšky hlasu

Élastique 2 Pro

Élastique 2 Pro od firmy Zplane development je soubor knihoven pro změnu délky trvání a výšky zvuku. Těchto knihoven využívá dle [14] velké množství profesionálních hudebních editorů jako je Cubase, Ableton Live, Reaper a další. Pro otestování kvality produktu poskytuje Zplane development zadarmo ke stažení jednoduchý program, do kterého je možné načíst zvukový soubor a otestovat změnu výšky a délky zvuku.



Obrázek 4.1. Okno testovacího programu *Élastique 2 Pro*.

Jediné nastavení, při kterém je možné zachovat formanty je „Pro default“. Výsledky pro posun zvuků o oktávu nahoru a dolů při tomto nastavení je možné najít na příloženém cd v adresáři „\Přílohy\4\Elastique 2 Pro\“.

Výrobce bohužel neuvádí, jaký typ algoritmu používá, ale vzhledem k podobnosti zvuku s řešením implementovaným v rámci této práce je pravděpodobné, že se také jedná o metodu pracující v časové doméně. Posun výšky zvuku směrem dolů má slyšitelně horší kvalitu, než posun výšky zvuku směrem nahoru. V čase 4 vteřin je na nahrávce *mic-oktavaNahoru.wav* slyšet typ chyby, který je většinou způsobený špatným odhadem periody.

Dirac 3

Další knihovnou pro posun výšky zvuku a změny doby jeho trvání je Dirac 3 od DSP Dimensions. Knihovna Dirac se vyvíjí od roku 2005 a verze 3 vyšla na podzim 2010. Stejně jako knihovna *Élastique* je Dirac dle [16] používán v mnoha profesionálních hudebních editorech. Zdarma je k dispozici verze Dirac 3 Le, která má některá omezení, ale kvalita zpracování zvuku je stejná, jako v placené verzi Dirac 3 Pro.

Pro účely testování poskytuje výrobce připravený C++ projekt pro Visual Studiu 6. V něm je potřeba změnit jen cesty ke vstupním a výstupním souborům a nastavit zda chceme měnit čas, výšku, formanty nebo vše najednou. Dále je možné nastavit kvalitu v závislosti na požadované rychlosti zpracování. Jelikož nebylo uvedeno, zda je pro běh v reálném čase potřeba snížená kvalita, byl posun testován pro kvalitu maximální.

Výsledky posunu zvuků o oktávu nahoru a dolů při nastavení na maximální kvalitu výstupu je možné najít na příloženém cd v adresáři „\Přílohy\4\Dirac 3 Le“.

Stejně jako u předchozí knihovny výrobce neuvádí detaily implementace, ale je slyšet, že charakter výsledného zvuku je jiný než u předchozí testované nahrávky. Při posunu směrem dolů se zdá, jako by se do nahrávky zkopíroval původní hlas, což je slyšet zejména na nahrávce *mic-oktavaDolu.wav*.

4.2 VST efekty pro posun výšky hlasu

4.2.1 Placené efekty

Harmony Engine Evo

Harmony Engine Evo [18] od firmy Antares Audio Technologies patří mezi známé profesionální efekty pro generování harmonií. V plné verzi umožňuje vytvoření 4 přidavných hlasů. U každého hlasu je možné nezávisle měnit formanty, nastavovat hlasitost, přidat vibrato a mnoho dalších efektů. Pro porovnání s efektem implementovaným v rámci této práce budeme testovat jen posun jednoho hlasu se zachováním formantů o oktávu nahoru a dolů.

Výsledky posunu zvuků o oktávu nahoru a dolů je možné najít na příloženém cd v adresáři „\Přílohy\4\Harmony Engine Evo“.

Harmony Engine Evo také umožňuje ovládat míru posunu pomocí MIDI stejným způsobem jako zde implementované řešení.



Obrázek 4.2. Nastavení jednoho hlasu v Harmony Engine Evo.

Mu Voice

Mezi další placené efekty pro generování harmonií patří Mu Voice od Mu Effects. Podobně jako Harmony Engine Evo umožňuje Mu Voice měnit formanty nezávisle na výšce tónu, umožňuje ovládání efektu pomocí MIDI ovladače a mnoho dalšího. Opět budeme testovat jen kvalitu posunu výšky jednoho hlasu. Bohužel Mu Voice poskytuje k testování jen verzi, která v náhodných intervalech zvuk utlumuje. I tak ale byly pořízeny nahrávky, které jsou k nalezení na přiloženém cd v adresáři „\Přílohy\4\Mu Voice“.



Obrázek 4.3. Okno efektu Mu Voice.

Stejně jako všechna předchozí řešení má Mu Voice větší problém s posunem hlasu dolů, než nahoru. Artefakty, které na zvuku při posunu směrem dolů vytváří, jsou dokonce výraznější než u ostatních řešení. Z neznámého důvodu se zdá, že Mu Voice zní lépe při běhu v reálném čase, pokud by měl čtenář zájem efekt si vyzkoušet, demo verze je ke stažení na adrese <http://www.mu-technologies.com/>.

4.2.2 Neplacené efekty

V kategorii neplacených efektů nebylo možné nalézt modul, který by byl schopný při posunu výšky hlasu zachovat formanty. Alespoň pro ukázkou toho, jak zní posun výšky tónu bez zachování formantů, byl použit efekt Choralozoide [17] od výrobce Numerikart.

Výsledky posunu zvuků o oktávu nahoru a dolů je možné najít na příloženém cd v adresáři „Přílohy\4\Choralozoide\“.



Obrázek 4.4. Okno efektu Choralozoide.

4.3 Zhodnocení

Řešení implementované v rámci této práce je, z hlediska funkcionality, porovnatelné jen s placenými efekty a knihovnami. Porovnání z hlediska kvality zvuku je ponecháno na čtenáři, jelikož, jak bylo řečeno v úvodu kapitoly, se jedná o věc subjektivní.

Oba efekty, představené v kapitole 4.2, mají kromě funkce pro posunu výšky hlasu mnoho dalších funkcí navíc. Některé z nich by nebylo složité přidat do řešení z kapitoly 3 – například možnost měnit formanty nezávisle na změně výšky hlasu. Způsob, kterým by se toho dalo docílit, byl popsán v kapitole 3.2.3. Přidání změny formantů a například dalších možností ovládání míry posunu výšky hlasu, by tedy mohlo být předmětem budoucích vylepšení, která jsou ale nad rámec této práce.

Z praktického hlediska by bylo dobré k efektu vytvořit profesionální grafické uživatelské rozhraní. V případě harmonizací by bylo zajímavé zaměřit se na takzvanou „humanizaci“ přidanych hlasů. Jedná se o cílené zavedení časových posunů nástupu hlasů, chyb v intonaci a dalších věcí, které mají za úkol vytvořit přirozenější, méně dokonale znějící, sbor.

Závěr

Jedním z cílů práce bylo seznámit čtenáře s vybranými pojmy z oblasti zpracování digitálního signálu a s nejnámějšími algoritmy pro změnu výšky tónu. Téma je to velmi obsáhlé, a proto jsem vybrala jen oblasti související s praktickou částí práce, kterou byl návrh a implementace efektu, který je schopný v reálném čase změnit výšku tónu a umí spolupracovat se standardním nahrávacím software. Pro získání detailnějších a komplexnějších informací z oblasti DSP doporučuji čtenáři k prostudování zejména [1] a [6].

V praktické části práce jsem navrhla a implementovala hudební efekt, který umožňuje přidat k původnímu hlasu dva další, vzdálené maximálně v intervalu oktávy od hlasu původního. U hlasů jsou zachovávány formanty. Dále je efekt do určité míry možné nastavit podle techniky, kterou zpěvák používá, podle míry hluku v okolí a podle toho, zda chceme efekt použít pro generování sborů nebo jen pro posun výšky jednoho hlasu. Míru posunu výšky jednotlivých hlasů je možné ovládat prostřednictvím MIDI ovladače. Efekt je implementován jako VST modul, v podobě dynamicky linkované knihovny. Díky tomu může efekt spolupracovat s většinou současných programů, pro nahrávání a editaci zvuku.

V kategorii volně dostupných VST modulů nebyl nalezen žádný, který by byl s implementovaným efektem porovnatelný z hlediska funkčnosti, kterou nabízí. V kategorii placených produktů již s čím porovnávat je. Jelikož nejdůležitější je u hudebního efektu kvalita zpracovaného zvuku, nelze efekty porovnávat objektivně. Proto jsou na přiloženém cd nahrávky ilustrující posuny výšky hlasu provedené pomocí jednotlivých efektů popsaných v kapitole 4.

Do budoucna by bylo možné vytvořit k efektu profesionální grafické uživatelské rozhraní. Dále by bylo zajímavé zaměřit se na zlepšení kvality zvuku při posunu výšky hlasu směrem dolů. Jak bylo patrné z nahrávek odkazovaných v kapitole 4, s kvalitou snížení výšky hlasu mají problémy i některé profesionální produkty, tudíž se jedná o oblast, která by si zasloužila více pozornosti.

Seznam použité literatury

[1] W. SMITH, Steven. *The Scientist & Engineer's Guide to Digital Signal Processing*. 1. California : California Technical Pub., 1997. 626 s.

[2] *Jreichl.com* [online]. 2006, 2011 [cit. 2011-04-14]. Encyklopedie fyziky. Dostupné z WWW: <<http://fyzika.jreichl.com/>>.

[3] SUNDBERG, J. Data on maximum speed of pitch changes. *Quarterly Progress and Status Report : STL-QPSR*. 1973, 14, 4, s. 039-047. Dostupný také z WWW: <<http://www.speech.kth.se/qpsr>>.

[4] In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-04-14]. Dostupné také z WWW: <cs.wikipedia.org/wiki/Rychlá_Fourierova_transformace>.

[5] ROCCHESO, Davide. *Introduction to Sound Processing* [online]. Università di Verona : Dipartimento di Informatica, March 20, 2003. 191 s. Oborová práce. Università di Verona. Dostupné také z WWW: <<http://www.scienze.univr.it/~rocchess/SP/sp.pdf>>.

[6] ZÖLZER, Udo. *DAFX - Digital Audio Effects*. Chichester (England) : John Wiley & Sons, 2002. 533 s. ISBN 0-471-49078-4.

[7] LENT, Keith. An Efficient Method for Pitch Shifting Digitally Sampled Sounds. *Computer Music Journal*. 1989, 13, 4, s. 7. Dostupný také z WWW: <<http://www.jstor.org/stable/3679554>>.

[8] DE CHEVEIGNE, Alain; KAWAHARA, Hideki. YIN, a fundamental frequency estimator for speech and musica. *Acoustical Society of America*.. April 2002, 111, 4, s. 14

[9] BRISTOW-JOHNSON, Robert. A Detailed Analysis of a Time-Domain Formant-Corrected Pitch-Shifting Algorithm. *J. AudioEng.Soc.*, May 1955, 43, 5, s. 340-352.

[10] *National Instruments* [online]. sep. 2008 [cit. 2011-04-14]. Ni Developer Zone. Dostupné z WWW: <<http://zone.ni.com/devzone/cda/tut/p/id/4844>>.

[11] SCHWARZ, Diemo. *Spectral Envelopes in Sound Analysis and Synthesis* [online]. Paris : IRCAM , 1998. Kapitoly 1.1 - 12.3 s. Diplomová práce. IRCAM. Dostupné také z WWW: <http://recherche.ircam.fr/anasyn/schwarz/da/specenv/Spectral_Envelopes.html>.

[12] RÖBEL, A.; RODET, X. EFFICIENT SPECTRAL ENVELOPE ESTIMATION AND ITS APPLICATION TO PITCH SHIFTING AND ENVELOPE PRESERVATION : Analysis/Synthesis Group. In *Conference on Digital Audio Effects*. Madrid, Spain : Proc. of the 8th Int., September 20-22, 2005. s. 6.

[13] J. LAROCHE; M. DOLSON. Improved phase vocoder time-scale modification of audio. *IEEE Trans. on Audio and Speech Processing*, Vol. 7, No. 3, May 1999.

[14] *Zplane Development* [online]. 2011 [cit. 2011-04-14]. Music Creation. Dostupné z WWW: <<https://www.zplane.de/index.php?page=description-elastique>>.

[15] LAROCHE, Jean; DOLSON, Mark. New Phase-Vocoder Techniques for Pitch-Shifting, Harmonizing and Other Exotic Effects. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 1999, Oct., s. 17-20.

[16] BERNSEE, Stephan M. *Dirac3 Technology* [online]. 2011 [cit. 2011-04-14]. Use Cases. Dostupné z WWW: <http://dirac.dspdimension.com/Dirac3_Technology_Home_Page/Dirac3_Technology.html>.

[17] *GERVIC.COM* [online]. 2008 [cit. 2011-04-14]. Synth/Marker. Dostupné z WWW: <<http://www.gersic.com/plugins/index.php?daPlug=1218>>.

[18] *Antarestech* [online]. 2011 [cit. 2011-04-14]. Harmony Engine. Dostupné z WWW: <http://www.antarestech.com/products/harmony_engine-evo.shtml>.

[19] BERNSEE, Stephan M. *The DSP Dimension* [online]. 1999-09-21 [cit. 2011-05-09]. Pitch Shifting Using The Fourier Transform. Dostupné z WWW: <<http://www.dspdimension.com/admin/pitch-shifting-using-the-ft/>>.

[20] PLACK, Christopher J.; OXENHAM, Andrew J.; FAY, Richard R. *Pitch: neural coding and perception*. New York : Springer, 2005. 364 s.

[21] FAROOQUI, Zainab Arif; SRIVASTAVA, Jyoti. *Scribd* [online]. 2011 [cit. 2011-08-01]. Continuous time & discrete signals. Dostupné z WWW: <<http://www.scribd.com/doc/52465571/CTS-DTS>>.

[22] BERNSEE, Stephan M. *The DSP Dimension* [online]. 1999-09-21 [cit. 2011-05-09]. Time Stretching And Pitch Shifting of Audio Signals – An Overview. Dostupné z WWW: <<http://www.dspdimension.com/admin/time-pitch-overview/>>.

[23] PARVIAINEN, Olli. Time and pitch scaling in audio processing. *Software Developer's Journal* [online]. 2006, 4, [cit. 2011-05-09]. Dostupný z WWW: <<http://www.surina.net/article/time-and-pitch-scaling.html>>.

[24] DE GÖTZEN, Amalia; BERNARDINI, Nicola; ARFIB, Daniel. Traditional (?) implementations of a Phase-Vocoder: The Tricks of the trade. *COST-G6 Conference on Digital Audio Effects*. 2000, DAFX-00, s. 1-7

[25] HVASS PADERSEN, Magnus Erik. *The Phase-Vocoder and its Realization*. Aarhus N, Denmark, 2003. 35 s. Oborová práce. University of Aarhus.

[26] RÖBEL, A. A Shape-Invariant Phase Vocoder for Speech Transformation. *Conference on Digital Audio Effects*. 2010, DAFx-10, s. 1-8.

[27] JENKINS, Kenneth. *Steps Toward a Better Phase Vocoder*. Princeton, 2009. 4 s. Referát. Princeton University.

Seznam použitých knihoven

[28] *Steinberg* [online]. 2011 [cit. 2011-04-15]. 3RD PARTY DEVELOPER.

Dostupné z WWW: <<http://www.steinberg.net/en/company/developer.html>>.

VST SDK verze 2.4

[29] C++ šablona pro vývoj VST efektu od doktora Jiřího Schimmela z Vysokého Učení Technického v Brně, verze 1.2.