VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Filip Švihálek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VIZUALIZACE ZVUKU

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — **Bc. Filip Švihálek**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Kamil Říha, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Bc. Filip Švihálek ***ID:*** 164955
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Vizualizace zvuku

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte možnosti vizualizace zvuku na základě jeho parametrického popisu. Navrhněte a implementujte metodu pro vytvoření syntetického obrazu/objektu, jehož parametry budou měněny na základě časově proměnných parametrů zvuku (hlasitost, spektrální složení, tempo…).
Možné nástroje pro implementaci: PureData, OpenCV, OpenGL a MS Visual C++.
Jako výsledek práce je požadováno naprogramování kompletního vizualizačního řetězce obsahujícího měření parametrů (tempo, hlasitost apod.) a jejich vizualizace formou komplexního syntetického obrazu, případně obrazu reálného, avšak s komplexně propracovanou metodikou změny obrazových parametrů ovládaných extrahovanými parametry hudebními.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ, R. C.; WOODS R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002.

[2] BRADSKI, G.; KAEHLER A.: Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc. USA 2008, ISBN: 978-0-596-51613-0.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Kamil Říha, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Bibliografická citace:

ŠVIHÁLEK, Filip. *VIZUALIZACE ZVUKU*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2019, 45s. Vedoucí diplomové práce: doc. Ing. Kamil Říha, Ph.D.

Prohlášení

„Prohlašuji, že svou semestrálnípráci na téma Vizualizace zvuku jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené semestrální práce dále prohlašuji, že v souvislosti s vytvořením této semestrální práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **14. prosince 2018** ……
podpis autora

Poděkování (nepovinné)

V této sekci je možné uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.). Příklad poděkování:

Děkuji vedoucímu diplomové (bakalářské) práce Prof. Ing. Jiřímu Novotnému, CSc. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **10. května 2018**

……
podpis autora

Obsah

Úvod ... 10

1 Zvuk ... 12

1.1 Parametry zvuku z hlediska fyziky ... 12

1.2 Parametry zvuku z hlediska lidského slyšení ... 12

1.2.1 Parametr frekvence ... 12

1.2.2 Hlasitost ... 13

1.2.3 Tempo ... 15

2 digitální zpracování zvuku ... 16

2.1 Vzorkování ... 16

2.2 Kvantování ... 16

2.3 Kódování ... 17

3 Váhové filtry ... 18

3.1 Frekvenční váhování ... 18

4 ADSR obálka ... 20

5 Open gl ... 21

5.1 Renderování ... 21

5.2 FreeGLUT ... 21

5.2.1 Inicializace okna ... 21

5.2.2 Zobrazení obsahu ... 22

5.2.3 Mazání obsahu ... 22

5.2.4 Určování barvy ... 23

5.2.5 Velikost a pozice okna ... 23

5.2.6 Základní geometrické útvary ... 23

5.2.7 Průhlednost (Blending) ... 25

6 Knihovna bass ... 27

6.1 Podporované formáty ... 27

6.2 Základní funkce ... 27

6.3 Chybové kódy ... 29

7 Praktická část ... 30

7.1 Stanovení tempa ... 30

7.2 Spektrum a amplituda ... 31

7.3 Zobrazení FFT spektra ... 31

7.4 Zobrazení amplitudy ... 35

7.5 Zobrazení tempa ... 36

7.6 Ovládání ... 39

Závěr ... 40

Seznam příloh ... 43

A Obsah přiloženého CD ... 43

A Obsah přiloženého cd ... 44

Seznam obrázků

1.1 Křivky stejné hlasitosti …… 13
3.1 Modulové kmitočtové charakteristiky váhových filtrů…… 16
4.1 Obálka typu ADSR…….. 17
5.1 Povolené útvary převzato …… 24
5.2 Konstrukce polygonu převzato …….…….. 25
7.1 Vykreslení kružnice……. 32
7.2 Vykreslení FFT spektra na kružnici …… 33
7.3 Vykreslení zrcadlově převráceného FFT spektra …… 35
7.4 Vykreslení amplitudy…… ……. 36
7.5 Vykreslení zobrazení tempa ……..……. 38
7.6 Finální animace …….…… …… 39

ÚVOD

Tato diplomová práce se věnuje problematice zpracování časově proměnných parametrů zvuku a jejich zobrazení v reálném čase. Schopnost zpracování dat v reálném čase se v dnešní době může zdát jako maličkost, vzhledem k tomu, že téměř každé dva roky se rychlost procesorů násobně zvyšuje. Již v minulém století prohlásil Gordon Moore empirické pravidlo o exponenciálním růstu výpočetní techniky. Jenomže stejně jako stoupá výkon techniky, stoupají také uživatelské nároky. Požadované výpočty jsou čím dál náročnější. To samé platí pro výpočty v reálném čase, kdy je požadováno, aby daný výpočet trval pro člověka nepostřehnutelnou dobu. Jako příklad si můžeme vzít virtuální realitu. Ta musí zpracovávat obrovské množství dat tak, aby navodila uživateli vjem, že se nejedná o fikci, ale aby měl pocit, že se opravdu nachází v jiném světě.

I když by se zdálo, že pro zpracování vizualizace zvuku je již výkon více než dostačující, stále můžeme chtít lepší a datově náročnější vizualizaci, která počítače dostihne. Jako zářný příklad si můžeme vzít moderní počítačové hry. Každým rokem vývojáři pracují na reálnějších modelech prostředí tak, aby uživatele vtáhli do hry co nejvíce. Stejně s propracovanější grafikou rostou výpočetní nároky na její zpracování. Tento fakt je možné pozorovat na počítačích starších tří a více let. Pokud na nich budete chtít spustit nejmodernější hru s plnými detaily, a třeba i ve 4k rozlišení, budete mít nejspíše problém. Samozřejmě vždy existují komponenty, které si s takovou hrou poradí, nicméně takhle vysoký výkon je vykoupen také vysokou cenou.

Příklad můžete najít na jednom z největších výrobců grafických karet. Řeč je o americké společnosti NVIDIA. Ta v roce 2018 představila nejnovější grafické katy s označením RTX. Zkratka RTX je odvozena od pojmu Ray Tracing. Tato technologie je známá již dlouhou dobu, ale doposud byla použitelná pouze pro zpracování obrazu mimo reálný čas. Ray Tracing umožňuje reálné nasvícení scény simulováním fyzického chování světla. To znamená, že počítá výslednou barvu pixelů sledováním trasy světla včetně odrážení od objektů a procházení průhlednými nebo částečně průhlednými objekty. Nová řada těchto grafických karet má k dispozici tolik výkonu, že dokáže provádět tyto výpočty v reálném čase.

Tato práce je ve své podstatě rozdělena na několik částí. Část první je věnována hlavně teorii toho, co je to vlastně zvuk, jak jej chápeme a jaké parametry jsme u něj schopni vnímat. Parametry jsou popsány jak z fyzikálního hlediska, tak z hlediska vnímání člověkem.

Část druhá se věnuje popisu zvuku a jeho interpretaci z digitálního hlediska. Jsou zde popsány základní úkony pro digitalizaci a také s tím související chyby.

V třetí části je pozornost zaměřena na váhové filtry, jejich křivky a popis matematickými rovnicemi. V části čtvrté je stručně popsána ADSR obálka a její složky.

Pátá část je věnována prostředí OpenGL. Tato část obsahuje krátké seznámí s prostředím a následně jsou popisovány základní procesy, funkce a příkazy.

Poslední teoretická část, s pořadovým číslem 6, je věnována audio knihovně BASS. V kapitole budete seznámeni se základními funkcemi, podporovanými formáty a

možnými chybami.

Poslední část 7 je částí praktickou. Jsou zde popsány algoritmy a implementace měřených parametrů a také jejich implementace jejich vizualizace. Tato kapitola obsahuje také úryvky kódu a jejich grafické výstupy.

1 ZVUK

Zvuk má pro člověka nemalý význam při každodenních činnostech. Slouží nejen ke komunikaci a orientaci v prostoru, ale také jako mohutný zdroj informací, který náš mozek zpracovává v reálném čase. Z hlediska fyziky se jedná o mechanické vlnění částic v pružném prostředí, schopné šířit se ve skupenství pevném, kapalném i plynném. Šíření zvuku probíhá díky pružnosti prostředí, ve kterém se při kmitání částic vytvářejí místa s relativním zhuštěním nebo zředěním částic. Ty poté postupují od zdroje kmitání předáváním kinetické energie částic prostředí rychlostí zvuku, která je za podmínek, kdy je teplota vzduchu $t = 20°C$ a atmosférický tlak $p_0 = 101{,}3$ kPa, rovna hodnotě 343 m/s.

1.1 Parametry zvuku z hlediska fyziky

Frekvence, také označována jako kmitočet, je jedním ze základních parametrů a udává nám počet, s jakým se zvuková vlna opakuje za určitý časový úsek. Pro její vyjádření používáme následujíc vztah:

$$f = \frac{1}{T}\ [\text{Hz}], \tag{1.1}$$

kde T je perioda a dále pak vztah:

$$f = \frac{c_0}{\lambda}\ [\text{Hz}], \tag{1.2}$$

kde c_0 je rychlost zvuku a λ je vlnová délka.
Jelikož se jedná o vlnění, musí také vykazovat maximální hodnotu. Takovou hodnotu poté nazýváme **amplituda**, nebo výkmit.
Pro parametrický popis vlnění ještě zbývá parametr nazývaný **fáze**. Fáze vlny je bezrozměrná veličina, která nám udává místo, kde se vlna nachází v čase nula.
Tyto tři parametry jsou nedílnou součástí pro popis zvuku z hlediska lidského slyšení.

1.2 Parametry zvuku z hlediska lidského slyšení

1.2.1 Parametr frekvence

Kmitočet vnímáme sluchovým ústrojím jako výšku tónu. Jedná se o subjektivní vlastnost zvuku, nicméně vztah mezi výškou tónu a frekvencí je exaktní. To nám zaručuje, že bude například klavír správně naladěn, stejně jako orchestr. K tomu se využívá takzvaného referenčního tónu, označovaného jako komorní tón. Jedná se o oficiální standard, používaný po celém světě, který nám říká, že tón A1 má mít frekvenci 440 Hz.

1.2.2 Hlasitost

Hlasitostí zvuku máme na mysli subjektivní škálu, na které jsme schopni pomocí sluchového vnímání seřadit zvuky od tichých po hlasité [1]. Hlasitost zvuku souvisí s veličinou nazvanou intenzita zvuku, kterou definujeme následovně:

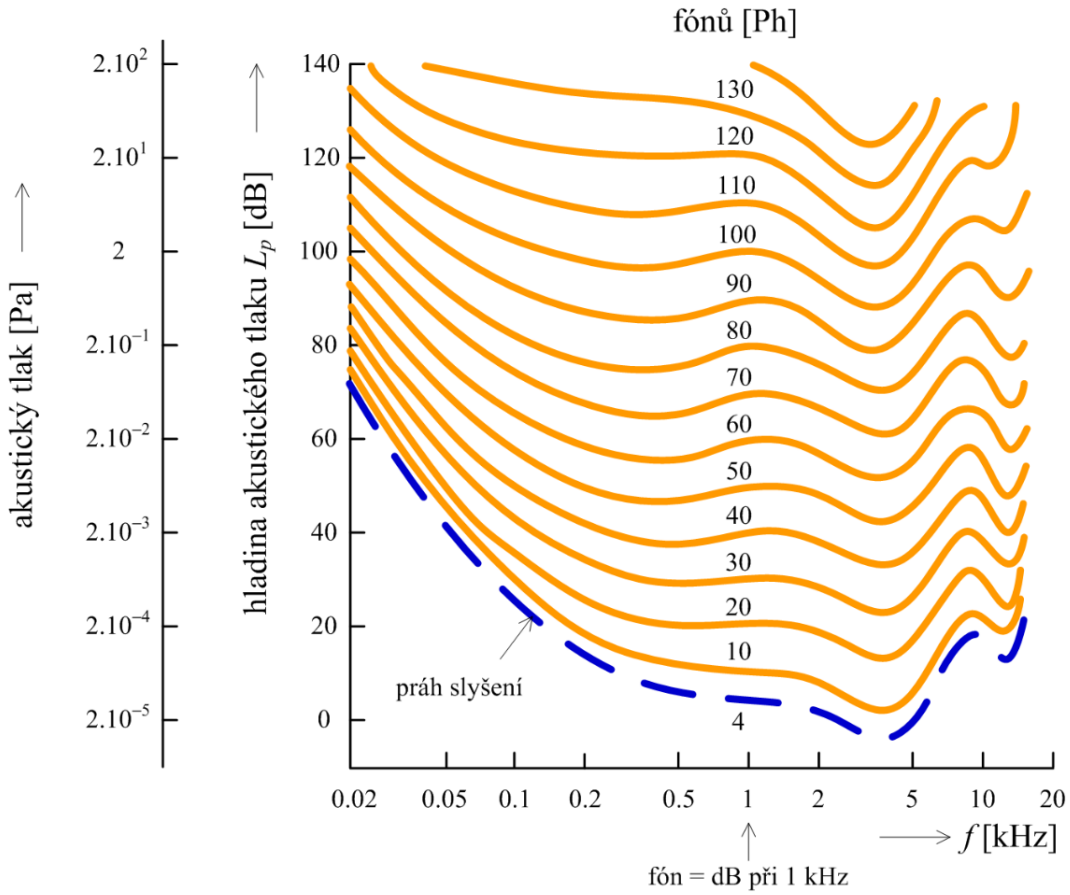
$$I = \frac{1}{T_{max}} \int_0^{Tmax} pvdt \ \ [\mathrm{W/m^2}], \tag{1.3}$$

kde $T_{max} \gg 1/f_n$ je kmitočet nejnižší spektrální složky složeného zvuku, p je akustický tlak a v je akustická rychlost. Vzhledem k tomu, že lidské ucho vnímá násobky akustických veličin jako přírůstky, je vhodné pro vyjádření akustických veličin použít logaritmické hladiny. Pro hladinu intenzity zvuku tedy máme daný vztah [2]

$$L_I = 10 \log \frac{I}{I_0} \ [\mathrm{dB}], \tag{1.4}$$

kde I_0 je vztažná hodnota 10-12 W/m2, která je definována pro frekvenci 1 kHz. Jak bylo řečeno výše, hlasitost je subjektivní veličina a je třeba podotknout, že dva zvuky o stejné hladině intenzity nemusíme slyšet stejně hlasitě. To je způsobeno hned několika faktory. Běžně se uvádí, že slyšitelné pásmo frekvencí pro člověka se nachází v intervalu od 20 Hz až do 20 kHz. Když však vezmeme v potaz věk, víme, že s procesem stárnutí dochází k jisté degradaci sluchu. Obecně se dá říci, že osoby ve věku 40 až 50 let většinou neslyší zvuky, které dosahují frekvence vyšší než 15 kHz [3].

Dalším faktorem, který ovlivňuje vjem hlasitosti, je frekvence zvuku (tónu). Vztah (2.2) je tedy platný pouze pro tóny o frekvenci 1 kHz a pro zjištění intenzity hlasitosti dalších frekvencí je zapotřebí subjektivního srovnání hlasitosti daného tónu s referenčním tónem o kmitočtu 1 kHz. Tímto srovnáním byly získány Kingsburyho křivky stejné hlasitosti [3]. Tyto křivky nám říkají, jaký je poměr mezi hladinou hlasitosti určitého tónu o dané frekvenci a hladině hlasitosti tónu s frekvencí 1 kHz.



Obr. 1.1: Křivky stejné hlasitosti z [3].

Na obrázku (1.0) vidíme, že číselné hodnoty pro kmitočet 1 kHz ve fónech korespondují s hodnotami hladiny akustického tlaku. To však platí jen pro onen referenční tón s kmitočtem 1 kHz. I když jsou křivky stejné hlasitosti zjištěny na základě subjektivního srovnání, stále neodpovídají subjektivnímu vjemu hlasitosti. Například, rozdíl 5 fónů je na nižších hladinách hlasitosti téměř nepozorovatelný, ale ve vyšších hladinách hlasitosti je již podstatný [2]. Stejně tak nevnímáme subjektivně rozdíl 80 a 40 fónů jako dvojnásobnou hlasitost. Další problém nastává při zjišťování výsledné hladiny hlasitosti, pokud se vjem skládá z několika čistých tónů. V tomto případě nejsme schopni získat výslednou hladinu hlasitosti tím, že sečteme dílčí hlasitosti jednotlivých tónů [2].

S tímto vědomím byla zavedena jednotka hlasitosti **son**. Vztah mezi sonem a fónem je takový, že hlasitost 1 son odpovídá hladině hlasitosti 40 fónů u tónů s frekvencí 1 kHz. Dalším subjektivním zkoumáním hladin hlasitostí a vnímanou hlasitostí člověkem byl odvozen vztah [3]:

$$N = 2^{(L_N - 40)/10} \text{ [son]}, \tag{1.5}$$

kde N je hlasitost v sonech a L_N je hladina hlasitosti ve fónech.

1.2.3 Tempo

Dalším parametrem, který je lidské slyšení schopno identifikovat je tempo. Ve světě hudby je tempo klíčové, udává nám totiž jak rychle či pomalu bude skladba znít. Metronomicky se tempo označuje BPM z anglického *Beats per minute* – údery za minutu a udává nám počet taktových dob za minutu.

2 DIGITÁLNÍ ZPRACOVÁNÍ ZVUKU

Digitálním zpracováním zvuku máme na mysli převod signálu z oblasti analogové do oblasti číslicové. Stejně jako digitální zpracování jiných signálů se musí řídit určitými pravidly tak, abychom nepřišli o důležitá data a vyvarovali se přílišnému zkreslení. Tento proces převodu rozdělujeme do tří částí: vzorkování, kvantování a kódování [6].

2.1 Vzorkování

Procesem vzorkování převádíme signál spojitý v čase na sérii čísel. Dle Nyquistova teorému je nutné dodržet poučku $f_{VZ} > 2f_{MAX}$, kde f_{VZ} je vzorkovací kmitočet a f_{MAX} je nejvyšší harmonická složa obsažená v signálu. Touto poučkou dle Nyquista docílíme toho, že signál nebude znehodnocen.

Jelikož analogový signál může nabývat nekonečně mnoha hodnot, je třeba, aby se rozdělil na vzorky (od toho pojem vzorkování). Vzorky se poté odebírají s určitou frekvencí f_{VZ} definovanou výše. Jak již bylo řečeno, oblast slyšení se nachází v pásmu 20 Hz až 20 kHz. Tedy je důležité, aby byl vzorkovací kmitočet vetší než 40 kHz. Pokud se v signálu nachází harmonická složka, která má kmitočet $f > 2f_{MAX}$, dochází k ději nazývanému aliasing. Proto se před A/D převodník zapojuje filtr typu dolní propust (nazývaný anti-aliasingový), který harmonické složky překračující dvojnásobek vzorkovací frekvence odstraní [5]. Z tohoto důvodu se běžně používají vzorkovací kmitočty 44,1 kHz a 48 kHz, potažmo jejich celočíselné násobky 88,2 kHz, 96 kHz, 176,4 kHz a 192 kHz.

2.2 Kvantování

Vzorkováním jsme převedli analogový signál na vzorky diskrétní v čase se spojitou úrovní. Procesem kvantování převádíme tyto vzorky z úrovně spojité na úroveň diskrétní. Kvantizací tedy zaokrouhlujeme spojité hodnoty získaných vzorků na kvantovací hladiny, kde je počet kvantovacích hladin dán počtem bitů následného kódování n. Tento počet označujeme jako bitovou hloubku či bitové rozlišení. Kvantovací krok Q definujeme následujícím vztahem:

$$Q = \frac{1}{(2n-1)}, \tag{2.1}$$

kde n je počet bitů převodníku bez znaménkového bitu. Při kvantizaci dochází k chybě nazývané kvantizační chyba. Ta je způsobena diskrétními úrovněmi výstupního signálu kvantovače. Maximální chybou převodníku je polovina hodnoty nejmenšího kvantovacího kroku. V procesu kvantizace definujeme také kvantizační šum. Jedná se od odchylku nakvantovaného signálu od původního a pro harmonické signály je definován takto:

$$SNR = 6{,}02n + 1{,}76 \text{ [dB]}. \tag{2.2}$$

V běžných případech se pro výpočet kvantovacího šumu používá odhad založený na velikosti kvantovacího kroku:

$$SNR = 20 \log \frac{1}{Q} \text{ [dB]} \tag{2.3}$$

V případě audio signálů používáme bitové hloubky 16, 20 a 24 bitů.

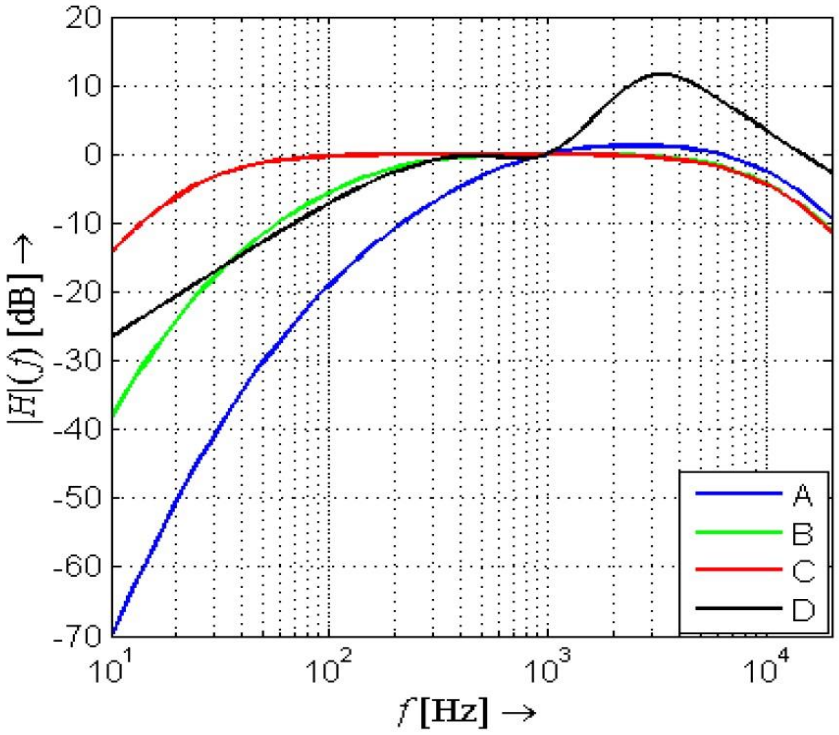
2.3 Kódování

Posledním krokem potřebným k digitalizaci je proces kódování. Tímto procesem se převedou vzorky diskrétní v čase i úrovni na binární číslo. V oblasti zpracování zvuku se výhradně používá PCM tedy pulzně – kódová modulace (z anglického Pulse Code Modulation). Důvod převážného využití tohoto typu kódování tkví v tom, že při něm nedochází ke ztrátě informace či omezení šířky pásma rychlých přechodů. Digitální systémy, které zpracovávají audio signály používají k výpočtu úrovně signálu v dB relativní úroveň digitálního signálu dBFS (dB relitave to Full Scale). Maximální hodnota digitálního signálu, kterou může převodník zpracovat, odpovídá nominální úrovni 0 dBFS.

3 VÁHOVÉ FILTRY

3.1 Frekvenční váhování

Vzhledem k tomu, že citlivost lidského vnímání zvuku je kmitočtově závislá je zapotřebí, abychom se k této závislost přiblížili. Z tohoto důvodu používáme frekvenční váhové filtry, jejichž kmitočtová charakteristika je aproximací křivky inverzní ke křivce stejné hlasitosti. Tyto křivky se poté sčítají se spektrem získaným pomocí FFT. Běžně využívány jsou filtry typu A, které se aplikují při signálech dosahujících nízkých hodnot hladin akustických tlaků. Při vyšších hladinách akustických tlaků se pak využívá filtr typu C [1].



Obr. 3.1: Modulové kmitočtové charakteristiky váhových filtrů A, B, C a D převzato z [3]

Kmitočtovou charakteristiku váhového filtru typu C získáme pomocí dvou pólů na nízkém kmitočtu f_1, dále dvou pólů na vysokém kmitočtu f_4 a dvou nul na frekvenci 0 Hz. Filtr typu A následně vychází z filtru typu C a to tak, že se kmitočtová charakteristiku filtru C doplní dvěma vysokofrekvenčními propustmi prvního řádu. Rovnice pro filtr typu C je následující:

$$C(f) = 20\log\left[\frac{f_4^2 f^2}{(f^2+f_1^2)(f^2+f_4^2)}\right] - C_{1000}\ [\text{dB}]. \qquad (3.1)$$

Rovnici pro filtr typu A definujeme obdobně:

$$A(f) = 20\log\left[\frac{f_4^2 f^2}{(f^2+f_1^2)(f^2+f_4^2)\sqrt{(f^2+f_2^2)(f^2+f_3^2)}}\right] - A_{1000}\ [\mathrm{dB}], \qquad (3.2)$$

kde C_{1000} *a* A_{1000} jsou normující konstanty, které reprezentují elektrický zisk tak, aby kmitočtové funkce měly zisk 0 dB na kmitočtu 1 kHz. Jejich hodnoty aproximované na 0,001 dB jsou C_{1000} = -0,062 dB *a* A_{1000} = -2,000 dB. Pro kmitočty pak platí následující hodnoty: f_1 = 20,598997 Hz, f_2 = 107,65265 Hz, f_3 = 737,86223 Hz a f_4 = 12194,217 Hz [7].

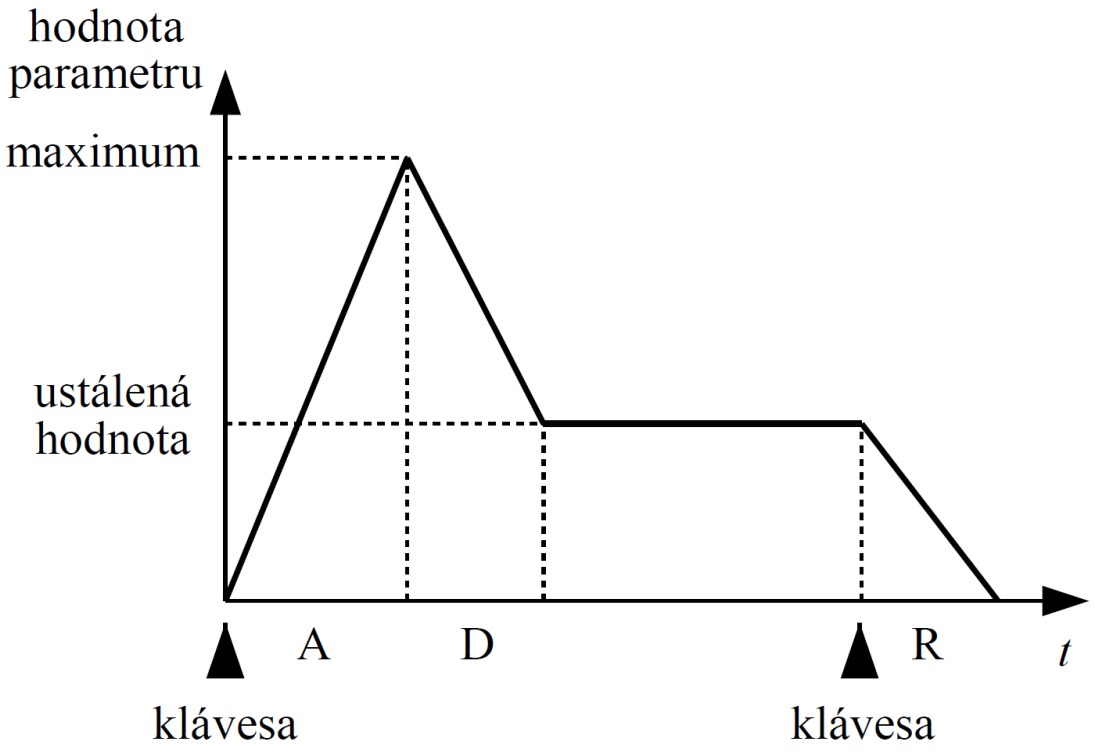
Podle zvoleného váhového filtru, doplníme jeho písmeno do indexu veličiny a za její jednotku následovně [1]:

$$L_A = 10\log\frac{I}{I_0}\ \ [\mathrm{dB(A)}]. \qquad (3.3)$$

4 ADSR OBÁLKA

ADSR obálka jednak ovlivňuje barvu zvuku, ale také je klíčem k určení tempa skladby. Název obálky je odvozen ze čtyř složek, z kterých se skládá.

- **Attack time** neboli doba náběhu je čas, který je potřebný k tomu, aby po stisknutí klávesy zvuk dosáhl své maximální hodnoty.
- **Decay time** je doba, za kterou klesne maximální hodnota na hodnotu ustálenou, tj. sustain.
- **Sustain level** je hlasitost zvuku při stlačené klávese. Hodnota hlasitosti zůstává konstantní až do doby, kdy je klávesa puštěna
- **Release time** je čas od puštění klávesy, za který hlasitost klesne na nulu.



Obr. 4.1: Obálka typu ADSR převzato z [6]

5 OPEN GL

OpenGL je vývojové prostředí určené pro grafický hardware. Jeho hlavní vlastnosti jsou přímočarost a hardwarová nezávislost. Díky tomu je možná implementace napříč hardwarovými platformami. Pro dosáhnutí těchto vlastností je OpenGL tvořeno jako nízko-úrovňové prostředí. To mimo jiné znamená, že neobsahuje vysoko-úrovňové příkazy, které by specifikovaly komplikované tvary. Například modely letadel, aut atd. Pro vytvoření komplikovaných modelů je zapotřebí, aby uživatel model „složil“ z primitivních geometrických útvarů čítajících tečky, čáry a polygony. Díky tomu nabývá prostředí obrovské výhody, jelikož uživatel může ovlivňovat i ty nejzákladnější příkazy, které mohou být v ostatních prostředích uživatelsky ne-editovatelné. Na druhou stranu zde vzniká také nevýhoda tkvící ve složitosti programu. Je třeba totiž myslet na to, že systém neudělá nic za vás, ale je nutné mu zadat přesně dané pokyny. Tyto pokyny jsou sami o sobě poměrně jednoduché a lze říci, že slouží ke změně stavu, který ovládá objekty a způsob, jakým je OpenGL vytváří.

5.1 Renderování

Procesem zvaným renderování (anglicky rendering) dochází k vytvoření obrazu ze zadaných dat či modelu, který je složen z primitivních geometrických objektů. Takový model je pak definován jeho vrcholy, respektive souřadnicemi vrcholů. Po dokončení procesu renderování vzniká obraz tvořený **pixely**. Pixel považujeme za nejmenší viditelný prvek, který je hardware schopen zobrazit. Informace o pixelech jsou uloženy v části paměti zvané **bitplane**. Jedná se o část paměti, jež obsahuje jeden bit informací pro každý zobrazený pixel. Tyto bity jsou dále uspořádány do tzv. framebufferu. Framebuffer obsahuje veškeré informace o tom, jaké barvy a intenzity mají všechny zobrazené pixely nabývat.

5.2 FreeGLUT

Freeglut je volně šiřitelná knihovna a alternativa k OpenGL Utility Toolkit (GLUT), která přestala být podporována již v roce 1998. Knihovna Freeglut byla vytvořena za účelem obstarávání systémových procesů potřebných pro vytváření oken, inicializaci OpenGL obsahu a zpracovávání vstupních událostí. Zjednodušeně se dá říci, že tato knihovna usnadňuje uživateli práci tím, že za pomocí jednoduchých příkazů vykonává příkazy komplexní, které jsou uživateli skryty [10].

5.2.1 Inicializace okna

Aby bylo možné vůbec zobrazit vymodelované objekty na obrazovce je zapotřebí několik základních příkazů sloužících k inicializaci okna.

- Funkce **glutInit** slouží k samotné inicializaci Freeglut knihovny a měla by být volána vždy před ostatními příkazy.
- **GlutInitDisplayMode** slouží pro výběr barevného modelu a typ bufferu. V této diplomové práci je používán barevný model RGBA (anglicky Red, Green, Blue, Alpha) a dvojitý buffer (anglicky Double Buffer). Jak již z názvu napovídá, dvojitý buffer obsahuje dva buffery, přičemž je vždy zobrazen právě jeden. V druhém bufferu probíhá renderování. Jakmile je renderování dokončeno, buffery se vymění. Díky této funkčnosti je na displeji zobrazena vždy kompletně vykreslená scéna. V případě použití jednoduchého bufferu (anglicky Single Buffer) dochází k instantnímu vykreslování na displej což mimo jiné způsobí, že objekty vykreslené jako poslední budou zobrazeny po kratší časový úsek než objekty, které byly vykresleny jako první.
- **GlutInitWindowPosition** určuje pozici vytvořeného okna počínaje levým, horním rohem.
- **GlutInitWindowSize** určuje velikost okna zadanou v pixelech.
- **GlutCreateWindow** vytvoří okno s obsahem tvořeným OpenGL.

5.2.2 Zobrazení obsahu

Nejdůležitější funkcí pro zobrazení obsahu je funkce **glutDisplayFunc**. Jedná se o funkci se zpětným voláním (anglicky Callback Function) do které je třeba vložit veškeré příkazy pro překreslení obsahu. Tyto příkazy se pak provedou vždy, když je potřeba překreslit obsah. Zavoláním funkce **glutMainLoop** se poté zobrazí všechna vytvořená okna a začne se provádět proces renderování. Jelikož se jedná o smyčku, bude se opakovat do nekonečna, nebo dokud ji nepřerušíte.

5.2.3 Mazání obsahu

Proces renderování by se dal připodobnit ke kreslení na papír ovšem s tím rozdílem, že při kreslení na papír začínáte kreslit již na danou barvu, kterou papír měl. V případě renderování je nejprve potřeba programu říci, jakou výchozí barvu má mít. K tomuto účelu slouží funkce **glClearColor**.

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
```

V prvním řádku se nastaví barva na černou. Druhý řádek obstará to, že se celé okno přemaže na černou barvu nastavenou v řádku prvním. Toho je docílenou parametrem GL_COLOR_BUFFER_BIT. Ten nám udává, že funkce glClear přemaže právě buffer obsahující informace o aktuální barvě.

5.2.4 Určování barvy

Vzhledem k tomu, že popis modelu je v OpenGL nezávislý na popisu barvy, je možné každou část modelu barevně odlišit. Jelikož byl v diplomové práci zvolen barevný model RGBA, lze pro určení barvy použít příkaz **glColor3f**. Příkaz je nutné volat se třemi parametry, jenž určují výsledný poměr barev. GlColor3f přijímá desetinná čísla ležící v intervalu hodnot <0.0;1.0>.

```
glColor3f(0.0, 0.0, 0.0);
glColor3f(1.0, 1.0, 1.0);
glColor3f(1.0, 0.0, 0.0);
glColor3f(0.0, 1.0, 0.0);
glColor3f(0.0, 0.0, 1.0);
```

Pořadí zadávaných barev je RGB, tedy červená, zelená, modrá. V prvním řádku bude výsledná barva černá, v řádku druhém bílá a poté následuje červená, zelená a modrá barva. Výsledné zabarvení tedy vytvoříme vhodným poměrem jednotlivých barev.

5.2.5 Velikost a pozice okna

V podkapitole 6.2.1 bylo popsáno, jak se inicializuje okno, tedy v jakém místě obrazovky a jak velké se vytvoří. Po vytvoření okna je samozřejmě na uživateli, jak s ním bude dále manipulovat. Může ovlivnit jeho velikost i pozici. Díky knihovně Freeglut proběhne při jakékoliv změně okna automatická notifikace a zavolá se zpětná funkce registrovaná ve funkci **glutReshapeFunc**. Funkce glutReshapeFunc předává registrované funkci dva argumenty, určující šířku a výšku změněného okna. V registrované funkci by pak neměly chybět následující příkazy:

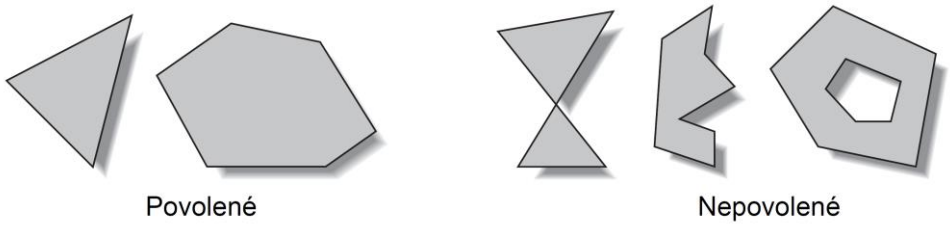
- **GlViewport** je příkaz pomocí něhož nastavíte část okna, ve které bude probíhat zobrazení scény.
- **GlMatrixMode** slouží pro zvolení typu maticového režimu. V této práci se využívá výhradně režim GL_PROJECTION, tedy režim projekční matice. Tento režim nám poskytuje přístup k způsobu, jakým se scéna vykresluje.
- Příkazem **glLoadIdentity** resetujeme projekční matici do původního stavu.
- **GluOrtho2D** nastaví ortogonální zobrazení scény

5.2.6 Základní geometrické útvary

Jak již bylo řečeno v úvodu kapitoly, OpenGL je nízko-úrovňové prostředí. Je tedy nutné, aby uživatel finální model složil z jednoduchých geometrický útvarů sestávajících se z bodů, úseček a polygonů. Všechny tyto útvary jsou v OpenGL definovány souřadnicemi jejich vrcholů. Tyto souřadnice mají však konečnou přesnost a zde narážíme na první limitaci. Konečná přesnost totiž znamená, že zde vznikne odchylka při zaokrouhlování.

Druhou a důležitější limitací jsou samotné zobrazovače. Nejmenší zobrazitelnou jednotkou jsou totiž pixely, a i když se může zdát, že pixel je dostatečně malý, není tomu tak. Například přesnost souřadnic bodu může být natolik velká, že pokud se změní jen nepatrně, způsobí to vykreslení stejného pixelu.

- Body jsou v OpenGL representovány vrcholy, jejichž souřadnice jsou dány trojčlennou skupinou desetinných čísel. Tato skupina zastává souřadnice x, y a z. V případě, že je bod zadán pouze prvními dvěma souřadnicemi, považuje se souřadnice z za nulovou.
- Úsečky taktéž reprezentujeme jejími vrcholy. V tomhle případě se jedná o souřadnice počátečního a koncového vrcholu. V OpenGL je možné vytvořit také sérii spojených úseček.
- Polygony lze vytvořit jako uzavřenou smyčku několika úseček, které jsou definovány jejich souřadnicemi počátečních a koncových bodů. Tvorba polygonů může být mnohdy složitá, proto je limitována samotným OpenGL. V případě první limitace OpenGL nedovoluje konstruovat polygon, jehož segmenty složené z úseček přes sebe přesahuj. Musí tedy lícovat. Druhou limitací je podmíněná konvexnost vytvořeného tělesa. To znamená, že když spojíme kterékoli dva body uvnitř tělesa, musí tato spojnice taktéž náležet uvnitř tělesa.



Obr. 5.1: Povolené útvary převzato z [8]

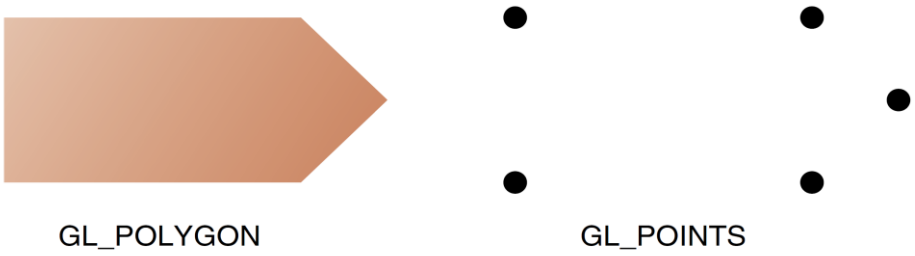
Pro tvorbu geometrického útvaru slouží příkaz **glBegin**, který je následně ukončen příkazem **glEnd**. Mezi začátek a konec příkazu pak pomocí řádků **glVertex2f** přiřazujeme jednotlivým vrcholům souřadnice. Počet těchto řádků budu následně značit písmenem n. Příkaz glBegin přijímá následující argumenty:

- GL_POINTS vykreslí jednotlivé body, kde n značí počet těchto bodů.
- GL_LINES považuje každou dvojici n za nezávislou úsečku.
- GL_LINES_STRIP vytvoří spojitou skupinu úseček.
- GL_LINES_LOOP vytváří spojitou skupinu úseček s rozdílem, že pojí poslední segment s prvním, aby vznikla smyčka.
- GL_TRIANGLES vytvoří z každé trojice n nezávislý trojúhelník.
- GL_TRIANGLES_STRIP vytvoří spojené trojúhelníky tak, že poslední dvě zadané souřadnice předchozího trojúhelníku tvoří zároveň první dvě souřadnice následujícího trojúhelníku.

- GL_TRAINGLES_FAN vytvoří spojené trojúhelníky tak, že mají pouze jednu společnou souřadnici. Pokud je tedy počet trojúhelníku M, pak je počet vrcholů M+2.
- GL_QUADS vytvoří z každé čtveřice n nezávislý čtyřúhelník.
- GL_QUAD_STRIP vytvoří skupinu spojitých čtyřúhelníků.
- GL_POLYGON vytvoří jeden konvexní polygon, kde n definuje tento polygon.

Ukázka tvorby polygonu s pěti vrcholy:

```
glBegin(GL_POLYGON);
glVertex2f(0.0, 0.0);
glVertex2f(0.0, 3.0);
glVertex2f(4.0, 3.0);
glVertex2f(6.0, 1.5);
glVertex2f(4.0, 0.0);
glEnd();
```



Obr. 5.2: Konstrukce polygonu převzato z [8]

5.2.7 Průhlednost (Blending)

Efekt průhlednosti je možné pozorovat přímo v operačním systému Windows, kdy skrze jedno okno vidíte okno druhé. Stejný efekt lze pozorovat v případě nasazení zabarvených brýlí. Pokud jsou skla brýlí zabarvené do červena a mají propustnost 90 % znamená to, že barvy, které skrze ně vidíte, jsou kombinací 10 % červené barvy skel brýlí a 90 % barvy daného objektu, na který se díváte. Tato technika je hojně využívaní právě v počítačové grafice. Abychom mohli efekt průhlednosti využít v OpenGL je nutné použít barevný model RGBA. V průběhu procesu zprůhlednění jsou hodnoty barev nově přicházejících dat (zdroj) zkombinovány s hodnotami barev, které jsou již uloženy ve framebufferu (destinace) a zobrazeny pixelem. Celkový proces zprůhlednění je rozdělen do dvou fází. V první fázi je zapotřebí definovat, jakým způsobem jsou vypočítány faktory průhlednosti zdroje a destinace. Ve fázi druhé jsou spočítány jejich kombinace. Pro lepší pochopení lze operace popsat matematicky.

Nechť Z_R, Z_G, Z_B, Z_A jsou faktory průhlednosti zdroje, D_R, D_G, D_B, D_A jsou faktory průhlednosti destinace, R_Z, G_Z, B_Z, A_Z jsou RGBA hodnoty zdroje a R_D, G_D, B_D,

A_D jsou RGBA hodnoty destinace. Pak jsou výsledné RGBA hodnoty spočítány následovně: ($R_Z Z_R + R_D D_R$, $G_Z Z_G + G_D D_G$, $B_Z Z_B + B_D D_B$, $A_Z Z_A + A_D D_A$).

Pro definování způsobu výpočtu faktorů průhlednosti slouží dvě funkce **glBlendFunc** a **glBlendFuncSeparate**. V rámci programu je použita pouze první z jmenovaných, proto bude popsána pouze první. Funkce glBlendFunc přijímá dva argumenty. První argument indikuje, jakým způsobem spočítat faktor průhlednosti zdroje a argument druhý indikuje způsob výpočtu faktoru průhlednosti destinace. Přehled parametrů přijímaných funkcí glBlendFunc je uveden v tabulce 5.1.

Tab.: 5.1 Parametry průhlednosti

Parametr	RGBA faktor
GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)
GL_SRC_COLOR	(R_Z, G_Z, B_Z, A_Z)
GL_ONE_MINUS_SRC_COLOR	(1, 1, 1, 1) – (R_Z, G_Z, B_Z, A_Z)
GL_DST_COLOR	(R_D, G_D, B_D, A_D)
GL_ONE_MINUS_DST_COLOR	(1, 1, 1, 1) – (R_D, G_D, B_D, A_D)
GL_SRC_ALPHA	(A_Z, A_Z, A_Z, A_Z)
GL_ONE_MINUS_SRC_ALPHA	(1, 1, 1, 1) – (A_Z, A_Z, A_Z, A_Z)
GL_DST_ALPHA	(A_D, A_D, A_D, A_D)
GL_ONE_MINUS_DST_ALPHA	(1, 1, 1, 1) – (A_D, A_D, A_D, A_D)
GL_CONSTANT_COLOR	(R_C, G_C, B_C, A_C)
GL_ONE_MINUS_CONSTANT_COLOR	(1, 1, 1, 1) – (R_C, G_C, B_C, A_C)
GL_CONSTANT_ALPHA	(A_C, A_C, A_C, A_C)
GL_ONE_MINUS_CONSTANT_ALPHA	(1, 1, 1, 1) – (A_C, A_C, A_C, A_C)
GL_SRC_ALPHA_SATURATE	(*i*, *i*, *i*, 1)

Posledním krokem k vytvoření průhlednosti je její zapnutí, které se provede příkazem **glEnable**(GL_BLEND) [8].

6 KNIHOVNA BASS

BASS je audio knihovna použitelná v programech na platformách Win32, MacOSX a Linux. Jejím účelem je poskytovat nástroje pro efektivní práci se zvukem, jeho samplováním, streamováním a nahráváním.

6.1 Podporované formáty

Popisovaná audio knihovna podporuje poměrně velké množství formátů zvukových souborů. V případě nutnosti doplnění dalších formátů, které knihovna defaultně nepodporuje, je možné využít podporu zásuvných modulů. Přehled podporovaných formátů:

- MPEG 1.0, 2.0 a 2.5 Layer III (MP3), Layer I (MP1), Layer II (MP2)
- standardní RIFF a RF64 (WAV)
- veškeré PCM formáty v bitovém rozlišení 8 až 32 bitů (WAV, AIFF)
- podpora vícekanálových formátů (OGG, WAV, AIFF)

6.2 Základní funkce

Nejdříve je potřeba inicializovat výstupní zařízení. Tento úkon provádí funkce **BASS_Init**, která vyžaduje na vstupu pět parametrů. BASS_Init(device, freq, flags, win, clsid). Parametr device slouží ke zvolení výchozího zařízení k přehrávání v případě hodnoty -1. V případě zadání hodnoty 0 nebude k dispozici žádný zvuk. Druhým parametrem volíme výstupní vzorkovací frekvenci. Parametr na třetím pořadí slouží k vybrání kombinace rozšiřujících parametrů čítajících:

- BASS_DEVICE_8BITS použije 8bitové rozlišení na místo 16bitového.
- BASS_DEVICE_MONO výsledný zvuk bude mít pouze jeden kanál.
- BASS_DEVICE_STEREO limituje počet kanálů na maximálně dva.
- BASS_DEVICE_3D zapne 3D funkcionalitu.
- BASS_DEVICE_LATENCY vypočte velikost zpoždění mezi požadavkem na spuštění přehrávání a samotným začátkem přehrávání. Také vypočítá minimální doporučenou velikost bufferu.
- BASS_DEVICE_CPSPEAKERS použije ovládací panely systému Windows pro detekci počtu reproduktorů.
- BASS_DEVICE_SPEAKERS vynutí zapnutí funkce pro přidělení reproduktorů.
- BASS_DEVICE_NOSPEAKER způsobí ignorování uspořádání reproduktorů.

První ze základních funkcí je načtení zvukového souboru a vytvoření streamu. To je možné pomocí funkce **BASS_StreamCreateFile**. Ta očekává na vstupu celkem pět parametrů. Bass_StreamCreateFile (mem, file, offset, length, flags). Prvním parametr lze nastavit, zda se vzorky načtou z paměti, či ze souboru. Druhý parametr očekává v případě

čtení ze souboru jeho název a v případě čtení z paměti očekává adresu v paměti. Třetím parametrem lze volit počátek streamu, který je ve výchozím stavu 0. Čtvrtý parametr nám naopak udává konec streamu, tedy zda se mají načíst všechna data. Pátým a posledním parametrem lze volit kombinaci následující rozšiřujících parametrů:

- BASS_SAMPLE_FLOAT pro samplovaní dat bude použit 32bitový datový typ s plovoucí čárkou (float).
- BASS_SAMPLE_MONO dekóduje soubor jako jediný kanál tedy mono. Možné využít pouze pro soubory OGG, MP3, MP2 nebo MP1.
- BASS_SAMPLE_SOFTWARE vynutí nepoužití hardwarového DirectSound (komponenta společnosti Microsoft).
- BASS_SAMPLE_3D zapne 3D funkcionalitu. Ta musí být povolena i při inicializaci a zároveň musí být stream mono.
- BASS_SAMPLE_LOOP bude po načtení posledního vzorku načítat zase první vzorek
- BASS_SAMPLE_FX zapne starší implementaci DirectX 8 efektů.
- BASS_STREAM_PRESCAN provede skenování souboru pro přesnější manipulaci při přetáčení. Pouze pro soubory MP3, MP2, MP1.
- BASS_STREAM_AUTOFREE způsobí automatické uvolnění streamu po skončení přehrávání.
- BASS_STREAM_DECODE dekóduje vzorky dat bez jejich přehrání.
- BASS_SPEAKER_*xxx* slouží pro přidělení reproduktorů různým kanálům.
- BASS_ASYNCFILE způsobí asynchronní čtení souboru. V případě zapnutí je soubor načítán do bufferu paralelně s dekódováním.

Další použitá funkce **BASS_ChannelGetData** slouží pro získání dat aktuálního vzorku signálu, nebo jeho FFT reprezentaci. Funkce při volání vyžaduje na vstupu celkem tři parametry. BASS_ChannelGetData(handle, buffer, length). První parametr určuje, odkud data získat. Druhý parametr udává, kam získaná data vložit a parametr třetí určuje velikost a typ získaných dat:

- Počet bajtů až do 268435455
- BASS_DATA_FLOAT vrací data s plovoucí čárkou (float)
- BASS_DATA_FF256 vzorek o velikosti 256 bajtů vrací 128 hodnot FFT.
- BASS_DATA_FFT512 vzorek o velikosti 512 bajtů vrací 256 hodnot FFT.
- BASS_DATA_FFT1024 vzorek o velikosti 1024 bajtů vrací 512 hodnot FFT.
- BASS_DATA_FFT2048 vzorek o velikosti 2048 bajtů vrací 1024 hodnot FFT.
- BASS_DATA_FFT4096 vzorek o velikosti 4096 bajtů vrací 2048 hodnot FFT.
- BASS_DATA_FFT8192 vzorek o velikosti 8192 bajtů vrací 4096 hodnot FFT.
- BASS_DATA_FFT16384 vzorek o velikosti 16384 bajtů vrací 8192 hodnot FFT.
- BASS_DATA_FFT32768 vzorek o velikosti 32768 bajtů vrací 16384 hodnot FFT.

- BASS_DATA_FFT_COMPLEX vrací komplexní FFT výsledek, tedy reálnou i imaginární složku.
- BASS_DATA_FFT_INDIVIDUAL provede FFT pro každý kanál zvlášť. Velikost výstupních dat je pak násobena počtem kanálů.
- BASS_DATA_FFT_NOWINDOW způsobí, že na vzorkovaná data nebude aplikováno Hannovo váhovací okno.
- BASS_DATA_FFT_NYQUIST vrací extra hodnotu pro Nyquistovu frekvenci.
- BASS_DATA_FFT_REMOVEDC provede odstranění stejnosměrné složky ze vzorkovaných dat.

V případě, že není zvolen příznak BASS_DATA_FFT_COMPLEX, vrací funkce pouze reálné složky první polovinu FFT spektra [12].

6.3 Chybové kódy

V průběhu používání knihovny BASS se může stát, že narazíte na její limity. V takovém případě lze příkazem BASS_ErrorGetCode získat kód chyby. Nejčastější chybové kódy jsou popsány v tabulce 6.3

Tab.: 6.3 Chybové kódy

Chybový kód	Chybová hláška	Vysvětlivka
0	BASS_OK	Všechno v pořádku
1	BASS_ERROR_MEM	Chyba pamětí
2	BASS_ERROR_FILEOPEN	Chyba při otevírání souboru
3	BASS_ERROR_DRIVER	Chyba ovladače zvukové karty
4	BASS_ERROR_BUFLOST	Nenalezen buffer
5	BASS_ERROR_HANDLE	Chyba při vytváření streamu
6	BASS_ERROR_FORMAT	Nepodporovaný formát

7 PRAKTICKÁ ČÁST

V této kapitole budou popsány a rozebrány jednotlivé části vytvořeného programu.

7.1 Stanovení tempa

Tempo nám udává, jak rychle bude skladba hrát. Z hlediska vnímání tempa člověkem je to rytmus, který si vydupáváme nohou, popřípadě tleskáním. Tyto pohyby jsou ovládány mozkem na základě signálu, který zpracovávají naše uši. Signál má určitou energii, která roste s hlasitostí. Abychom však mohli detekovat tempo je třeba, aby se v signálu projevovala pravidelná energie větší, než je energie průměrná.

Z hlediska zpracování signálu je to podobné. Jednoduché by bylo nalézt špičky v signálu a spočítat jejich vzdálenost. Počet takovýchto vzdáleností v intervalu šedesáti vteřin by nám pak udával počet dob za minutu. Problém by nastal v okamžiku, kdy by skladba obsahovala například cinkání. Projevem cinkání je velký nárust ve spektru signálu, který zamaskuje hledané špičky a tím by byl výpočet tempa nesprávný.

Jedním z možných algoritmů je hledání špiček signálu v několika sub-pásmech a následné nalezení vzorků s instantní energií větší než energie průměrná. Tento algoritmus vymyslel autor Frédéric Patin a je dostupný z [13].

Prvním krokem je výpočet energie jednotlivých vzorků z FFT spektra. Tento výpočet je nutný provést pro všechny kanály a následně tyto energie sečíst. V případě této diplomové práce však funkce BASS_ChannelGetData vrací již sloučené vzorky FFT spektra obou kanálů. Následně je zapotřebí takto získaná data uložit do bufferu. Pro lepší přesnost detekce tempa má buffer velikost odpovídající jedné sekundě hudby. To v našem případě odpovídá velikosti vzorkovací frekvence vydělenou počtem vzorků, tedy 44100/2048 = 21,53. Tento výpočet je nutné provést pro všechna sub-pásma.

Druhý krok spočívá ve výpočtu průměrné energie pro všechna sub-pásma. Následuje porovnání energie aktuálního vzorku s průměrnou energií. Pokud vzorek překročí vypočtenou průměrnou energii, můžeme ho považovat za beat. To ovšem nemusí být nutně pravda, protože vyšší energie může také znamenat pouze hlasitější pasáž skladby. Abychom se této chybě vyvarovali, je před průměrnou energii ještě vložen práh, který detekci tempa zpřesní. Stále je však možné, že se ve skladbě projeví nástroje způsobující vyšší energii, například cinkot. Takový typ zvuku bez problémů překročí práh a algoritmus ho nesprávně určí jako beat. Z tohoto důvodu se ještě porovnávají časy, respektive pozice vzorků překračujících daný práh. Pozice pak musí odpovídat dosavadnímu průměru mezi dvěma beaty.

Implementovaná detekce tempa vychází z implementace autora Williama Vennese dostupná z [14]. Původní implementace však počítá odhad tempa pouze z jednoho sub-pásma. Z tohoto důvodu byl původní kód rozšířen o výpočet tempa v rámci třiceti-dvou sub-pásem a následně byl pomocí mediánu vypočítán výsledný odhad. Implementovaná metoda má výhodu v nepříliš náročném výpočtu a dá se tedy spouštět

v reálném čase. Díky tomu je detekce tempa schopná reagovat na různé pasáže skladby, což je pro výslednou vizualizaci klíčové. Jediným nedostatkem je přesnost závislá na typu žánru skladby. U žánrů čítajících rock, taneční hudbu, hip-hop a jim podobné vykazuje metoda dobré a vcelku přesné výsledky. Ovšem u žánrů, kde tempo není tak přímočaré, jako třeba u vážné hudby, metoda poněkud zaostává a výsledky jsou zatíženi vcelku velkou nepřesností.

7.2 Spektrum a amplituda

Při tvorbě FFT spektra v podstatě převádíme časovou oblast na oblast frekvenční. Každá frekvenční složka spektra je pak reprezentována její velkostí. Pokud tedy máme data obsahující 2048 vzorků a vzorkovací frekvenci 44100 Hz, pak je frekvence prvního vzorku rovna 44100/2048, tedy 21,53 Hz. Frekvence druhého vzorku je poté dvojnásobek frekvence vzorku prvního. FFT spektrum získáme z knihovny BASS pomočí příkazu BASS_ChannelGetData.

Pro získání amplitudy právě přehrávaného vzorku využijeme další funkci z knihovny BASS s názvem BASS_ChannelGetLevelEx. Funkce očekává na vstupu celkem čtyři parametry. BASS_ChannelGetLevelEx(handle, levels, length, flags). Prvním parametrem určíme, odkud data získat. Druhý parametr nám udává název proměnné, získaná data vložit. Třetím parametrem ovládáme množství dat pro výpočet amplitudy. Množství je zadáváno v sekundách, přičemž maximum je jedna sekunda. Poslední parametr slouží k rozšíření funkčnosti a přijímá jeden či kombinaci vícero následujících parametrů:

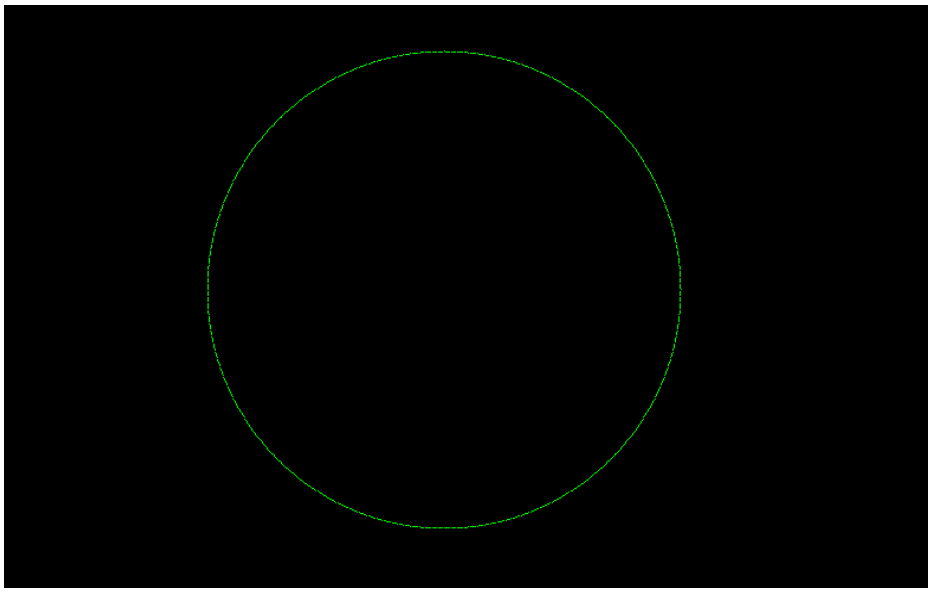
- BASS_LEVEL_MONO vrátí amplitudu jednoho kanálu (mono)
- BASS_LEVEL_STEREO vrátí amplitudy levého a pravého kanálu zvlášť
- BASS_LEVEL_RMS místo špičkového hodnoty vrací střední hodnotu.

7.3 Zobrazení FFT spektra

Základem celé animace je kruh. Jedna z možností, jak v OpenGL vytvořit kruh je za pomocí for cyklu a goniometrických funkcí sinus, cosinus. Při větším počtu kružnic není tato metoda úplně vhodná, jelikož je výpočetně náročnější, nicméně pro řešení jednodušších vizualizací je dostačující.

```
glBegin(GL_POINTS);
int points = 1024;
for (int i = 0; i < points; i++)
{
  angle= PI * 2 * i / points;
  glColor3f(0.0, 1.0, 0.0);
  float x1 = cos(angle) *  (radius);
  float y1 = sin(angle) * (radius);
  glVertex2f(x1, y1);
}
glEnd();
```

Ve for cyklu se postupně začnou vykreslovat body, jejichž počet je dán proměnnou points. Souřadnice x každého bodu je dána funkcí cosinus a souřadnice bodu y funkcí sinus. Obě tyto souřadnice jsou vynásobeny proměnnou radius, který odpovídá poloměru kruhu. Čím více bude v proměnné points definováno bodů, tím bude výsledný kruh vypadat celistvěji. Barva bodů je zvolena příkazem glColor3f a v našem případě bude zelená.

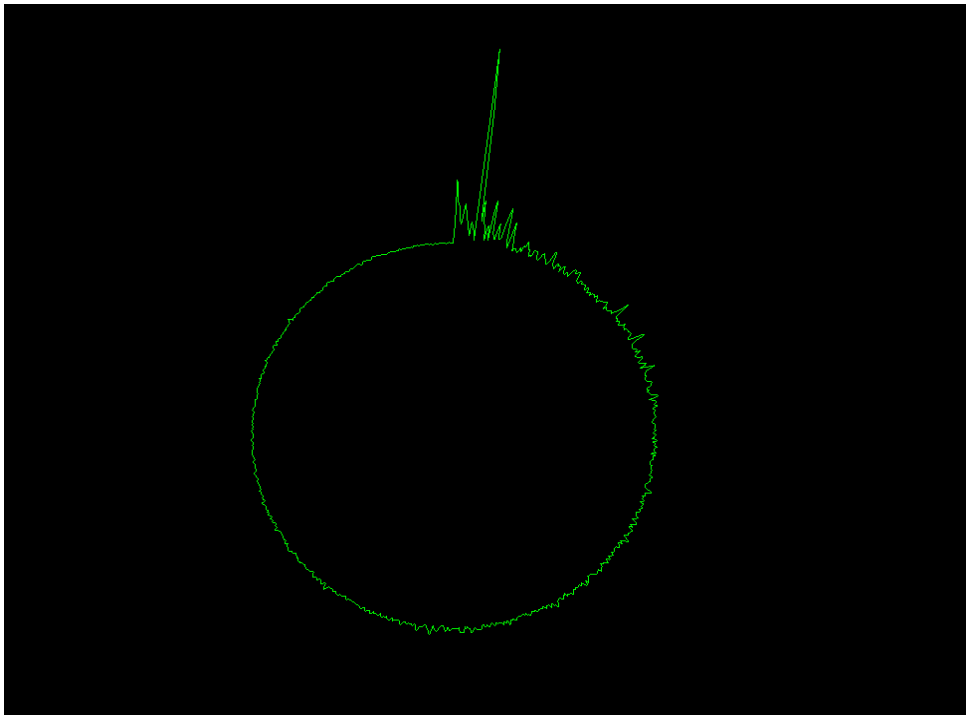


Obr. 7.1: Vykreslení kružnice

Pro vykreslení FFT spektra využijeme proměnnou radius, kterou vynásobíme složkou spektra.

```
glBegin(GL_LINE_LOOP);
int points = 1024;
float amplitude = 600;
for (int i = 0; i < points; i++)
{
  theta = PI * 2 * i / points – PI/2;
  glColor3f(0.0, 1.0, 0.0);
  float x1 = cos(angle) *  (radius + (fft[int(i)] * amplitude));
  float y1 = sin(angle) * (radius + (fft[int(i)] * amplitude));
  glVertex2f(x1, y1);
}
glEnd();
```

Jelikož je složka spektra poměrně malá, nebyla by ve výsledku moc vidět. Proto vynásobíme každou složku spektra ještě proměnnou amplitude.



Obr. 7.2: Vykreslení FFT spektra na kružnici

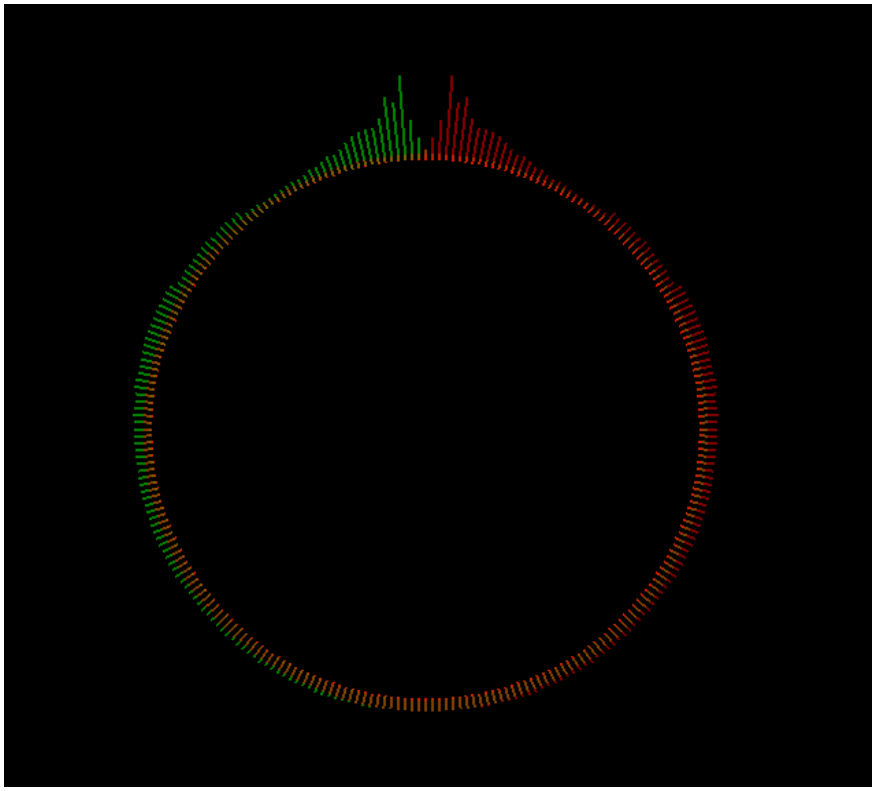
V tomto bodě máme tedy vykreslené FFT spektrum do tvaru kružnice. Výsledek není po estetické stránce moc dobrý, je tedy zapotřebí spektrum upravit.

```
for (int i = 0; i < halfFFT; i++)
{
  pomFFT[i] = 0;
  int count = i + 1;
  for (int j = 0; i + j < halfFFT && j < count; j++)
  {
    pomFFT[i] = pomFFT[i] + fft[i + j];
   }
   pomFFT[i] = pomFFT[i] / count;
}
for (int i = 0; i < halfFFT; i++)
{
 editFFT[i] = editFFT[i];
}
```

Úprava FFT spektra spočívá ve zkombinování několika složek do jedné složky. Konkrétně je každá n-tá složka součtem složek předchozích a vydělená jejich počtem. Takovou úpravou docílíme kompaktnějšího zobrazení spektra, jež působí lepším estetickým dojmem.

```
glBegin(GL_LINE_LOOP);
float  halfFFT = 4096/2;
for (int i = 0; i < halfFFT; i++)
{
  angle = PI*i/128 -PI/2;
  glColor4f(0, 1, 0, 0.5);
  float x1 = cos(angle) *  (radius);
  float y1 = sin(angle) * (radius);
  glVertex2f(x1, y1);
  float x2 = cos(angle) * (radius + -(editFFT[i] * ampltiude));
  float y2 = sin(angle) * (radius + -(editFFT[i] * ampltiude));
  glVertex2f(x2, y2);
}
glEnd();
```

Abychom neměli pouze „obrys“ FFT spektra, ale paprsek směřující od poloměru kruhu, přidáme další souřadnice x1 a x2. Přidáním dalšího for cyklu s tím rozdílem, že úhel bude s opačným znaménkem, vytvoříme dvě FFT spektra zrcadlově převrácená podle osy y.



Obr. 7.3: Vykreslení zrcadlově převráceného FFT spektra

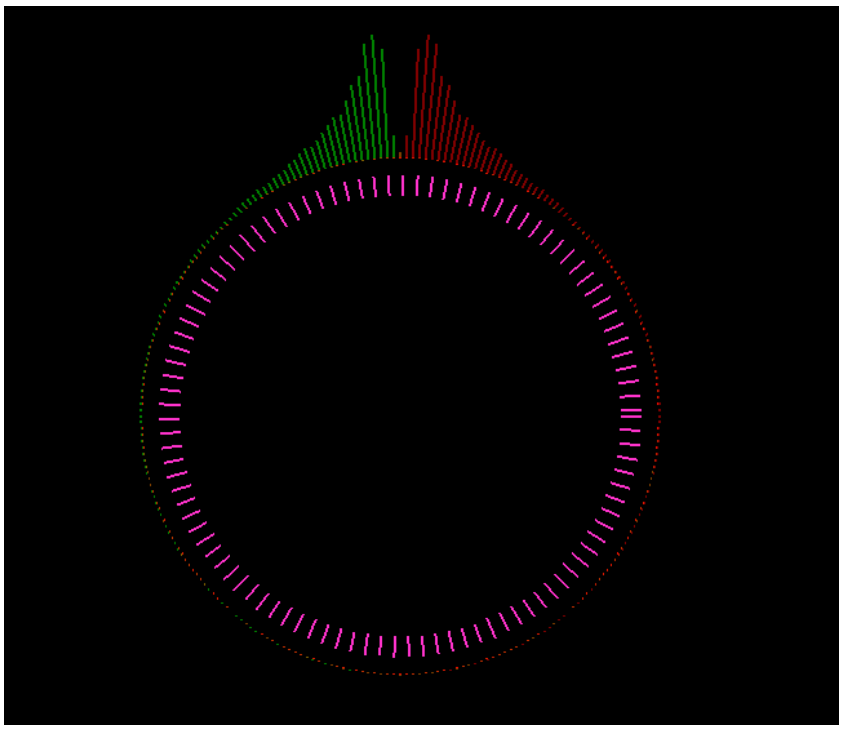
7.4 Zobrazení amplitudy

V kapitole 8.2 bylo popsáno, jak získat hodnotu amplitudy právě přehrávaného vzorku. Nyní bude popsán kód pro její zobrazení.

```
….
for (int i = 0; i < halfFFT/10; i++)
                {
                        radius = 150+ dBaOut;
                        angle = 2*i*PI/ (halfFFT / 10);
                        glColor3f(1.0, 0.2, 0.8);
                        float x1 = cos(angle) * radius;
                        float y1 = sin(angle) * radius;
                        glVertex2f(x1, y1);
                        float x2 = cos(angle) * (radius - (editFFT[3])*100);
                        float y2 = sin(angle) * (radius - (editFFT[3])*100);
                        glVertex2f(x2, y2);
                }
….
```

Pro vizualizaci amplitudy byl vytvořen další kruh s menším poloměrem a menším počtem bodů. Poloměr tohoto kruhu se bude měnit v závislosti na amplitudě právě přehrávaného vzorku skladby, kde hodnota této amplitudy je obsažená v proměnné dBaOut. Aby byl výsledný efekt výraznější, je od poloměru ještě odečtena třetí složka upraveného FFT spektra, která je dále násobena stem.

Jak bylo výše vysvětleno, třetí složka upraveného FFT spektra obsahuje složku druhou i první. Tyto složky odpovídají frekvenčnímu rozsahu 0 – 64,59 Hz, tedy basům v hudební terminologii. Třetí složka FFT spektra nebyla zvolena náhodou. Poněvadž cinkavé zvuky oplývají větší energií, bude se poloměr kružnice více roztahovat. Aby nebyla vizualizace jednotvárná a nenabývala rozměrů pouze jedním směrem, byla zvolena právě basová část spektra, která „prodlouží“ paprsky směrem do středu kružnice.



Obr. 7.4: Vykreslení amplitudy

7.5 Zobrazení tempa

Inspiraci k zobrazení tempa jsem našel u běžných analogových hodin, jelikož se bezprostředně pojí k vyjádření času. Jednoduchou implementaci analogových hodin, ze které kód vychází, lze najít v seznamu literatury [15].

```
void printInsideCircle(int ax, int ay, int bx, int by)
        {
          float da = ay - ax;
          float db = by - bx;
          float x1 = ax;
          float y1 = bx;
          float step = 0;
          step = abs(da);
          float xdx = da / step;
          float ydy = db / step;
          for (float i = 1; i <= step; i++)
                {
                  glVertex2f(x1, y1);
                  x1 += xdx;
                  y1 += ydy;
                }
}
```

Funkce printInsideCircle očekává na vstupu čtyři parametry, jenž tvoří souřadnice [x, y] dvou bodů. Prvním krokem je výpočet vzdálenosti souřadnic obou bodů. Absolutní hodnota vypočtené vzdálenost je posléze přiřazena maximální možné inkrementaci ve for cyklu. Druhým krokem je výpočet hodnoty, která se bude k jednotlivým souřadnicím přičítat, respektive odčítat, při každém projití for cyklu. Tato hodnota bude nabývat pouze čísla 1 nebo -1 podle kvadrantu, ve kterém se souřadnice jednotlivých bodů momentálně nacházejí. O vykreslení takto vytvořeného modelu se stará funkce displayInsideCircle.

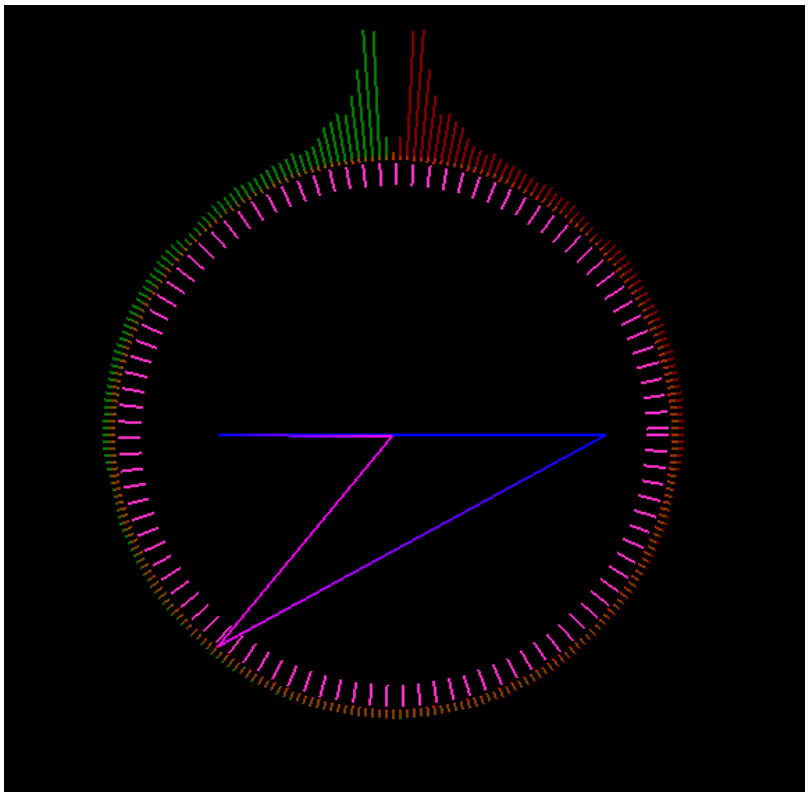
```
void displayInsideCircle(int ms)
{
  int x = 0; int y = 0;
  int insideRadius = 200;
  int secondx, secondy;
  secondy = y + (insideRadius * sin(insideDegree));
  secondx = x + (insideRadius * cos(insideDegree));
  glBegin(GL_LINE_LOOP);
  glPushMatrix();
  glColor3f(0.0, 0.0, 1.0);
  printInsideCircle(secondy, secondx, x, y );
  glEnd();
  insideDegree -= 0.0008*ms*beatsPerMinute / 60;
  glPopMatrix();
}
```

Tato funkce očekává jeden vstupní parametr, který bude popsán později. Jak je vidět, počáteční body začínají v souřadnicích [0, 0], což můžeme označit jako střed kružnice. Koncový bod je dán proměnnou insideRadius. Takto definované body tvoří délku úsečky začínající ve středu kružnice a končící v poloměru insideRadius. Vynásobením poloměru goniometrickou funkcí sinus, respektive kosinus, určíme bod na kružnici, ve kterém se bude nacházet koncový bod úsečky. Aby nebyla úsečka statická, ale „obíhala“ kružnici, je zapotřebí měnit úhel insideDegree.

Zde se konečně dostáváme k použití tempa skladby. Tempo, původně uvedeno v počtu úderů za minutu, je převedeno do počtu úderů za sekundu. Následně je vynásobena konstantou 0.0008 a parametrem ms.

Parametr ms je přijímán na vstupu funkce a určuje v milisekundách čas, za který funkce proběhla. Získán je příkazem glutGet(GLUT_ELAPSED_TIME) a slouží výhradně k tomu, aby rychlost „obíhání“ úsečky byla závislá pouze na tempu skladbu. V případě nepoužití tohoto parametru by byla rychlost „obíhání“ na různých hardwarech jiná, neboť se liší výkonem a tedy i dobou zpracování.

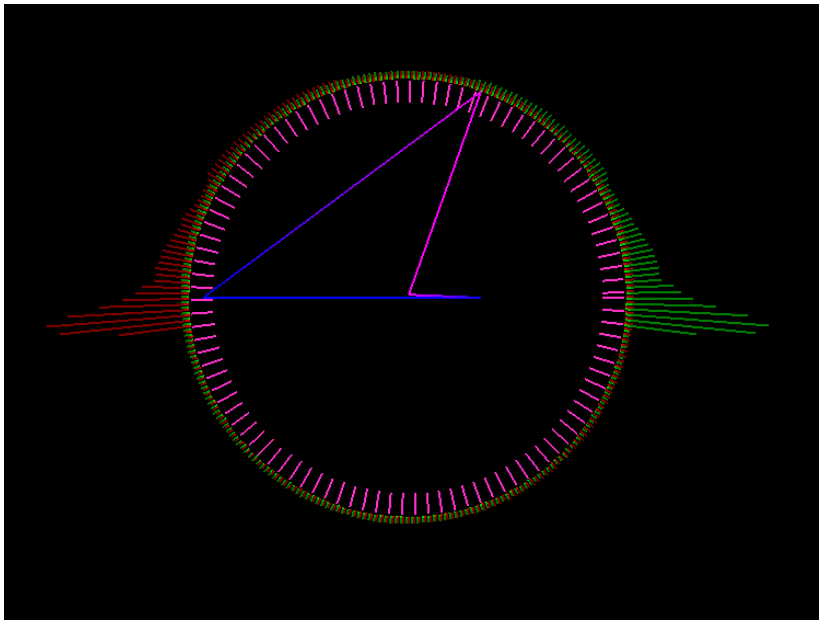
Nyní tedy máme vytvořenou rotující úsečku. Pokud však přidáme do funkce vícero příkazů glVertex2f(x1, y1) s různým pořadím souřadnic, získáme mnohem zajímavější rotující útvar.



Obr. 7.5: Vykreslení zobrazení tempa

Aby nebyla na tempu závislá pouze rotace vnitřního objektu, byly uvedeny do pohybu i FFT spektra. Ty však nerotují kolem celé kružnice, ale jen po „své“ půlkružnice. FFT

spektrum na levé straně se tedy pohybuje v intervalu $(\pi/2\,;3\,\pi/2)$ a FFT spektrum na straně druhé v intervalu $(3\,\pi/2\,;\pi/2)$. Pohyb je opět způsoben změnou úhlu, ve které se jako v předchozím příkladě promítá tempo a parametr ms.



Obr. 7.6: Finální animace

7.6 Ovládání

Výsledný program je spustitelný pod operačním systémem Windows, jako soubor exe. Při kopírování programu je nutné zkopírovat i přiložený obsah, jelikož obsahuje nezbytné knihovny. Po spuštění se otevře okno pro výběr souboru. Zde je zapotřebí, aby uživatel vybral podporovaný audio formát. Všechny podporované formáty knihovnou BASS jsou popsaný v kapitole 6.1. V případě, že zvolí nepodporovaný formát, zobrazí se dialogové okno s hláškou „Chybný nebo žádný soubor“. Po kliknutí na tlačítko „OK“ se program ukončí a je třeba jej spustit znovu. Stejně tak se program ukončí po dohrání skladby.

ZÁVĚR

Práce byla rozdělena do kapitol tak, jak k ní bylo přistupováno. Tedy nejprve nastudování teorie a poté samotná implementace. Po implementaci měření parametrů bylo třeba vymyslet, jak bude vypadat vizuální stránka výsledného modelu. Jako základ se mi zamlouvala myšlenka analogových hodin, která by reprezentovala tempo přehrávané skladby, které je s časem úzce spjato. V tomhle bodě bylo tedy stanoveno, že základem bude kruh. Následně bylo zapotřebí vymyslet zobrazení dalších parametrů tak, aby byl výsledný model celistvý a jednotlivé komponenty se vzájemně nenarušovali. Abychom toho mohli docílit, byly všechny objekty spjaty s kružnicí, která je samotným základem modelu. Tato stránka věci je samozřejmě do určité míry subjektivní, ale myslím, že výsledný objekt je poměrně celistvý.

Celkově se dá říci, že program dosahuje poměrně dobrých výsledků, animace je plynulá a jednotlivé komponenty nekazí dojem z celku. Je také poměrně zřejmé, jakými parametry jsou ovládány jednotlivé komponenty.

Největší úskalí pro mne bylo samotné prostředí OpenGL. Bylo zapotřebí pochopit správnou syntaxi a systém souřadnic. Tyto věci považuji za esenciální pro vlastní tvorbu v popsaném, grafickém prostředí. Je faktem, že OpenGL je nízko-úrovňové a dá se tam tak tvořit, co budete chtít, ovšem bez limitů to samozřejmě není. Tahle obrovská výhoda je zároveň nevýhodou pro lidi, kteří s prostředím začínají. Naštěstí je k dispozici nepřeberné množství tutoriálů, které jsou mnohdy cestou k úspěchu.

Přes veškerá úskalí se tedy nakonec podařilo vytvořit vizualizaci hudby. Po estetická stránce, i když je ryze subjektivní, může model nabývat dojmu jednoduchosti. To je jeden z bodů, který bych programu vytkl. Pro esteticky lepší formu je ovšem nutné ovládat pokročilejší techniky v prostředí OpenGL, na které bohužel nezbyl čas.

Literatura

[1] ČSN EN 61672-1: *Elektroakustika – Zvukoměry – Část 1*: *Technické požadavky*. Česká technická norma, Český normalizační institut, listopad 2003.

[2] SCHIMMEL, J. *Elektroakustika*. Brno: VUT Brno, 2012, 167 s. ISBN: 978-80-214-4450-8.

[3] SMÉKAL, Z. *Analýza signálů a soustav*. Brno: VUT Brno, 2012, 251 s. ISBN: 978-80-214-4453-9.Šablona pro BP/DP a prezentace v2.63 [online]. Brno: FEKT VUT, 2017 [cit. 2017-03-06].

[4] MÜLLER, M. *Fundamentals of music processing*. 1. New York, NY: Springer Berlin Heidelberg, 2015. ISBN 9783319219448.

[5] SCHEDL, M., *GOMEZ, E., URBANO*, J. Music Information Retrieval: Recent Developments and Applications. Vol. 8. Boston, USA: Now Publishers, 2014. ISBN 978-1601988065.

[6] SCHIMMEL, J. *Studiová a hudební elektronika*. Brno, 2015. Skriptum. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[7] ČSN EN 61260: *Elektroakustika - Oktávové a zlomkooktávové filtry*. Česká technická norma, Český normalizační institut, červenec 1997.

[8] KNEES, P., SCHEDL, M. *Music similarity and retrieval: an introduction to audio- and web-based strategies.* Vol. 36. New York, NY: Springer Berlin Heidelberg, 2016. ISBN 9783662497203.

[9] DAVE SHREINER, *OpenGL Programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*, 7th ed., Boston, MA 02116: Pearson Education, Inc., 2009. ISBN 978-0-321-55262-4.

[10] http://www.opengl-tutorial.org [online]: http://www.opengl-tutorial.org/assets/pdf/opengl_tutorial_2017_06_07.pdf

[11] http://freeglut.sourceforge.net/ [online]: http://freeglut.sourceforge.net/

[12] BASS library [online]: https://www.un4seen.com/doc/#bass/bass.html

[13] BPM algoritmus [online]: http://www.flipcode.com/misc/BeatDetection Algorithms.pdf

[14] Beat detection [online]: http://www.williamvennes.com/beat-detection.html

[15] OpenGL cloc [online]: https://github.com/sprintr/opengl-examples/blob/master/OpenGL-Clock-Animated.cpp

SEZNAM PŘÍLOH

A Obsah přiloženého CD

A OBSAH PŘILOŽENÉHO CD

Přiložené CD obsahuje výsledný program spustitelný souborem exe. Dále obsahuje kód ve formátu Visual Studia.