LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SKAŅAS EFEKTU MODUĻU IZSTRĀDE, IZMANTOJOT VST SDK

BAKALAURA DARBS

Autors: **Ilmārs Kazeks**
Stud. apl. ik08372
Darba vadītājs: Dr. dat. Jānis Zuters

RĪGA 2012

Anotācija

Bakalaura darba mērķis ir izpētīt skaņas efektu moduļu izstrādi, kas izstrādāti izmantojot VST SDK 2.4 (Virtual Studio Technology Software Development Kit).

Darba teorētiskajā daļā tiek apskatīta ar datoru veidotas mūzikas vēsture, VST interfeisa attīstība, iespējas un uzbūve. Tiek veikts testa moduļa teorētiskais plānojums un meklēti matemātiskie risinājumi.

Darba praktiskajā daļā, tiek veikta testa moduļa izstrāde un aprakstīts izstrādes process, kas detalizēti paskaidro izstrādes gaitu un izvēlētos risinājumus. Tiek testēts izstrādātais testa modulis.

Secinājumos tiek apkopotas svarīgākās izstrādes procesā iegūtās atziņas un iezīmētas perspektīvas.

Abstract

Theme: Sound effect plugin development using VST SDK

Chief aim of this bachelor's thesis is to research the development of sound effect modules that designed using the VST SDK 2.4 (Virtual Studio Technology Software Development Kit).

In this work theoretical part are researched computer music history, VST interface evolution, capabilities and structure. Are made theoretical planning and developed the mathematical solutions for test plugin.

In the practical part of this work are made test plugin and described in details the development process and the choice of solutions. Are tested the developed test plugin.

In conclusion are summarized the most important lessons learned of VST plugin development and outlined future perspectives.

Atslēgvārdi: modulis, VST, skaņa, efekts, izstrāde.
Keywords: plugin, VST, sound, effect, development.

SATURS

Attēlu saraksts

Apzīmējumi

VST (Virtual Studio Technology) - Diemžēl Latvijas Zinātņu akadēmijas (LZA) terminu datu bāzē pagaidām nav oficiāla tulkojuma šim terminam. Autora ieteikums VST tulkojumam latviski būtu sekojošs: **virtuāla studijas tehnoloģija,** kur **studija**, tiek domāta kā skaņu ieraksta un apstrādes studija.

DAW (Digital Audio Workstation) – Plašas funkcionalitātes lietojumprogramma, kas paredzēta digitāla audio signāla ierakstam, apstrādei un sekvencēšanai.

DLL (Dynamic Link Library) – Datnes veids, ar paplašinājumu *.dll, kas satur funkciju bibliotēku un citus atribūtus, kuriem var piekļūt uz Windows platformas bāzētas lietojumprogrammas.

SDK (Software Development Kit) - Palīgprogrammu (rutīnu) kopa, kas lietojumprogrammu izstrādātājam atvieglo programmu veidošanu, ievērojot konkrētās to darbības vides īpatnības (piemēram, grafisko lietotāja saskarni, operētājsistēmu, datu bāzu pārvaldības sistēmu u. c.).

API (Application Program Interface) - Lietojumprocesos izmantojama pilna operētājsistēmas funkciju specifikācija, kā arī šo funkciju izmantošanas procedūru apraksts.

MIDI (Musical Instrument Digital Interface) - Standartprotokols, kas nosaka muzikālās informācijas apmaiņu starp mūzikas instrumentiem un datoru. Dators ar šādu saskarni atšķirībā no parastā magnetofona saglabā mūzikas ierakstu nevis analogā, bet gan diskrētā formā, fiksējot taustiņu nospiedienus un vadības kodus.

GUI (Graphical User Interface) - Displeja formatēšanas veids, kas ļauj lietotājam izvēlēties komandas, palaist programmas, kā arī apskatīt datņu sarakstus un citus objektus, norādot to piktogrāfiskos attēlus (ikonas). Izvēli var izdarīt, izmantojot tastatūru vai peli. Grafiskā lietotāja saskarne piedāvā vidi, kas nodrošina tiešu dialogu ar datoru. Grafisko lietotāja saskarni izmanto, piemēram, operētājsistēma Microsoft Windows.

Ievads

Jebkuram kurš strādāja skaņu ierakstu studijā deviņdesmitajos gados vai agrāk, mūsdienu studija ir kā pavisam jauna un dīvaina pasaule. Jaunas, iepriekš šķietami nereālas iespējas ir kļuvušas reālas līdz ar datoru un to programmatūras bāzētu studiju laikmeta pienākšanu.

Ja datoru pirmsākumos digitālās skaņas kvalitāte būtiski atpalika no tās, ko tajā laikā piedāvāja analogās iekārtas, tad nu jau labu laiku kā digitālās skaņas kvalitāte ir līdzvērtīga analogajai. Tāpēc ir vērts pievērst uzmanību un pētīt mūsdienu datoru dotās iespējas skaņu inženierijā.

Viens no aktuālākajiem virzieniem šobrīd ir modulāras lietojumprogrammas, kas dod neierobežotos iespējas trešās puses izstrādātājiem izstrādāt moduļus lietojumprogrammām, būtiski paplašinot to funkcionalitāti, izmantojot visas tās unikālās priekšrocības, ko spēj sniegt mūsdienu dators.

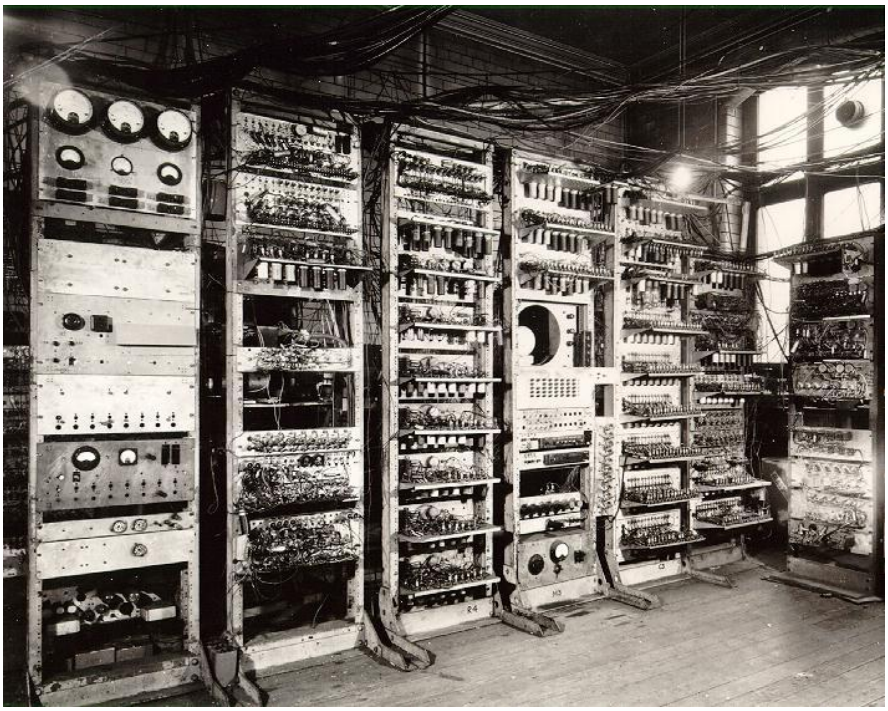
Ierakstu studijas aprīkojums kļūst ar vien pieejamāks un kompaktāks, jo lielos statīvus, pilnus ar „dzelžiem", aizstāj to programmiskās versijas jeb moduļi, kas imitē to darbību vai piedāvā pilnīgi jaunu funkcionalitāti, kas agrāk šķita nereāla.

Darba mērķis ir izpētīt vienu no šim tehnoloģijām, kas ļauj izstrādāt šādus moduļus. Šajā gadījumā tā ir Virtual Studio Technology jeb VST. Šī ir viena no populārākajām šāda veida tehnoloģijām tāpēc ir vērts izpētīt izstrādes procesu, lai pēc tam varētu realizēt nebijušus digitāla audio signāla apstrādes algoritmus vai simulēt reālu iekārtu darbību vienā iekārtā – datorā.

1. Ieskats ar datoru veidotas mūzikas vēsturē

1.1. Pirmais datora atskaņotais ieraksts

Tas notika Mančesteras Universitātē jau tālajā 1951 gadā. Lai to paveiktu, tolaik izmantoja Ferranti Mark 1 datoru. Pirmie skaņdarbi, ko tas spēja atskaņot bija God Save The Queen, Baa Baa Black Sheep un In The Mood. Ar to arī revolūcija varēja sākties!



Attēls 1-1 Ferranti Mark 1

1.2. Max Mathews

Par vienu no izcilākajiem, ar datoru veidotas mūzikas pionieriem, tiek uzskatīts Max Mathews. Viens no pirmajiem viņa sasniegumiem bija mūzikas lietojumprogrammas rakstīšana, kamēr viņš strādāja Bella laboratorijā ASV. Tas notika 1957 gadā. Lietojumprogramma ar nosaukumu „Music1" tika izmantota, lai radītu 17 sekunžu garu audio materiālu, kas tika atskaņots Ņujorkā uz IBM 704 datora. Vēlāk tika izveidotas arī Music2 (1958) un Music3 (1960) lietojumprogrammas, ar uzlabotu funkcionalitāti.

1.3. Commodore 64 parādīšanās

Liels progress mūzikas lietojumprogrammu izstrādē notika sešdesmitajos un septiņdesmitajos gados, bet kad parādījās Comodore C64 mājas dators, ar datoru veidota mūzika uzņēma jaunus apgriezienus. Galvenais kāpēc šis dators ir īpaši saistīts ar mūziku, ir tam speciāli izstrādātais SID mikroshēma, kas spēja sintezēt skaņu. SID mikroshēmas sintezētās skaņas ir populāras vēl

pat mūsdienās un tieši tāpēc ir pieejami Comodore 64 emulatori priekš mūsdienu operētājsistēmām.

Attēls 1-2 Comodore 64

1.4. MIDI protokola rašanās

1983. gadā tika izveidota pirmā MIDI protokola versija. MIDI izveide bija būtisks ieguvums mūziķiem, kas izmantoja datoru, lai veidotu mūziku. Šis protokols lāva analogajiem sintezatoriem, datoriem un citām iekārtām sazināties savā starpā. Vēl mūsdienās plaši tiek izmantots šis protokols.

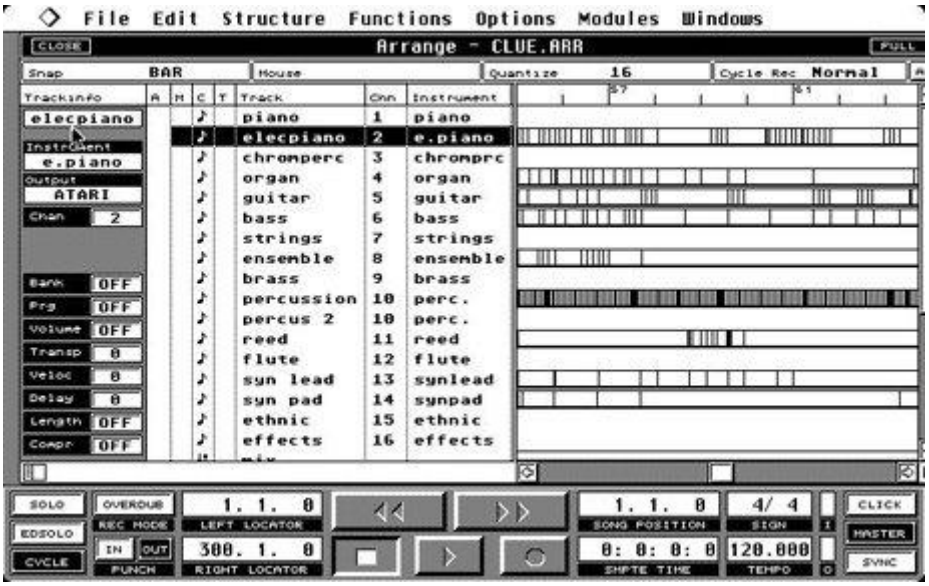
1.5. Atari prezentē ST

Mūsdienās mēs lietojam personālos datorus, makintoš datorus un citus, bet astoņdesmitajos gados katrs mūziķis, kas izmantoja datoru savai radošajai nodarbei, vēlējās sev Atari ST. Atari ST izstrādātāji paveica lielisku darbu, ievietojot MIDI portu savā datorā, kas deva iespējas kontrolēt dažādas, ar mūziku saistītas iekārtas, izmantojot datoru.

1.6. Steinberg izstrādā Cubase lietojumprogrammu

Kompānija Steinberg arī pirms Cubase izstrādes bija veiksmīgi darbojusies audio lietojumprogrammu izstrādē, izstrādājot Pro-24, taču īstu atzinību un popularitāti tā ieguva izstrādājot Cubase pirmo versiju. Tas notika 1989 gadā. Viena no galvenajām Cubase pirmās versijas īpašībām, bija horizontāli novietotais skaņas celiņu izkārtojums, kas ļāva ērti pārskatīt

un būvēt aranžējumus. Ātri vien šāds izkārtojums kļuva par vispopulārāko visu audio sekvencēšanas lietojumprogrammu vidū.



Attēls 1-3 Steinberg Cubase 1

1.7. Skaņas ieraksts datorā kļūst pieejams

Agrīnos deviņdesmitajos gados, pieejama sāka kļūt iespēja datorā ierakstīt digitālu audio signālu, jo līdz tam datoru izmantoja, lai sekvencētu skaņas no ārējām iekārtām, izmantojot MIDI protokolu. Tajā laikā Steinberg laiž klajā Cubase Audio priekš makintoš datoriem un vēl jau uzlabotu versiju arī priekš Antari Falcon datoriem.

1.8. Sound Blaster Pro skaņas karte

Pašas pirmās Creative Labs ļoti populārās Sound Blaster skaņas kartes tika prezentētas 1989 gadā, bet tieši Pro versijai tika pievienota iespēja veikt stereo skaņas ierakstu ar nolašu frekvenci 44.1 kHz. Katra nolase tika kodēta tikai ar 8 bitiem, bet nebija ilgi jāgaida līdz tika prezentētā 16 bitu versija. Tas notika pēc gada, kad tika prezentēta Sound Blaster 16 skaņas karte. Šī karte, teorētiski, ļāva mājas datoram ierakstīt CD kvalitātes skaņas signālu.

1.9. Virtuāla Studijas Tehnoloģija (VST)

Līdz ar kompānijas Steinberg lietojumprogrammas Cubase VST nākšanu klajā, 1997 gadā, kļuva iespējams ne tikai ierakstīt vairākus audio signāla celiņus, bet arī izmantot efektu moduļus audio signāla apstrādei. VST moduļi kļuva par alternatīvu reālām iekārtām. Drīz vien arī citas lietojumprogrammas piedāvāja līdzīgu funkcionalitāti.

1.10. Pirmais virtuālo instrumentu modulis

1999. gadā kompānija Steinberg jau atkal klajā nāca ar ļoti inovatīvu risinājumu līdz ar Cubase VST 3.7 versiju. Tajā bija iespējams izmantot skaņas sintēzes programmatūras moduļus. Mūsdienās šādi programmatūras moduļi ir ļoti plaši izplatīti. Viens no pirmajiem šādiem moduļiem bija Neon, kas bija vienkāršs analoga tipa skanējuma sintezators. Drīz pēc tam sekoja lavīnveida moduļu izstrādes bums gan komerciālu, gan bezmaksas.

Attēls 1-4 Steinberg Neon modulis

2. VST standarts

2.1. Raksturojums

Virtual Studio Technology (VST) ir kompānijas Steinberg izstrādāts moduļu interfeiss, lai integrētu audio sintezatorus un efektu procesorus audio redaktoros. VST un līdzīgas tehnoloģijas izmanto dažādus digitālo signālu apstrādes algoritmus, lai simulētu reālu iekārtu darbību, vai realizētu pilnīgi jaunus digitālā signāla apstrādes algoritmus ar programmatūras palīdzību. Eksistē tūkstošiem moduļu, kas lieto šo interfeisu gan komerciāli, gan bezmaksas, un VST tehnoloģiju atbalsta liels skaits audio redaktoru.

Moduļus ar VST interfeisu pārsvarā izmanto tā saucamās „digitālās audio darba stacijas" (no angļu val. *Digital Audio Workstation*, DAW), kas ir pamata programmatūra audio ierakstam un sekvencēšanai un ar šo moduļu palīdzību tiek paplašināta funkcionalitāte. Lielākā daļa moduļu satur grafisku saskarni, kas būtiski atvieglo lietojamību.

Starp moduļiem bieži sastopamas reālu iekārtu emulācijas, piemēram, Attēls 2-1 Fairchild 670 redzams reāls analogais skaņas kompresors, bet Attēls 2-2 T-Racks Limiter model 670 kompānijas T-Racks izstrādāts, šī paša kompresora VST modulis.

Attēls 2-1 Fairchild 670

Attēls 2-2 T-Racks Limiter model 670

Atšķirībā no VST instrumentiem, efektu procesori apstrādā ienākošo signālu un nav nepieciešams MIDI. Tomēr to izmanto, jo caur MIDI var padot ne tikai informāciju par notīm, bet arī dažādiem parametriem, kas ir gan sintezatoriem, gan efektu procesoriem, gan „sampleriem". Lielākā daļa „digitālās audio darba stacijas" atbalsta signāla maršrutēšanu, kas ļauj skaņas sintezatora moduļa izejošajam signālam pielietot kādu efektu moduli.

2.2. Attīstība

VST interfeisa specifikācija un SDK tika izdota 1997. g. Tas notika līdz ar kompānijas Steinberg „digitālās audio darba stacijas" Cubase 3.02. versijas iznākšanu, kuras sastāvā bija pirmie VST interfeisa moduļi.

VST interfeisa specifikācija tika atjaunota uz versiju 2.0 1999. g. Viena no jaunajām iespējām bija iespēja moduļiem saņemt MIDI informāciju, kas ļāva izstrādāt instrumentu moduļus. Neon bija pirmais VST instruments. Tas bija 16-balsu, 2 oscilatoru virtuāli-analogais sintezators.

2006. g. VST interfeisa specifikācija tika atjaunota uz versiju 2.4. Viena no jaunajām iespējām bija spēja apstrādāt audio signālu ar 64 bitu precizitāti.

2008. g. specifikācija tika atjaunota uz versiju 3.0. Dažas no jaunajām iespējam bija iespēja VST instrumentiem saņemt audio signālu un vairākas MIDI ieejas un izejas.

VST interfeisa specifikācija 2011. g. februārī tika atjaunota uz versiju 3.5. Viena no ievērojamākajām jaunajām iespējām ir katrai notij atsevišķi pievienot kāda parametra modulāciju.

2.3. VST moduļu veidi

Ir trīs VST moduļu veidi:

1. **VST instrumenti** – Tie ir skaņas sintezatori vai „sampleri”. Daži ir pilnīgi oriģināli un tikai kā programmiski moduļi pieejami, daži ir esošu iekārtu emulatori, piemēram, Native Instruments Pro-53 ir slavenā Prophet-5 sintezatora emulācija.

Attēls 2-3 Native Instruments Pro-53

Attēls 2-4 Prophet-5

2. **VST efekti** – skaņas kompresori, telpu un atbalss simulātori u.c. audio signāla apstrāde. Daži efekti neveic apstrādi, bet dod vizuālu priekšstatu par signālu.

Attēls 2-5 Cubase 4 RoomWorks telpas simulātors

Attēls 2-6 Waves PAZ-Analyzer signāla vizuālais analizators

3. **VST MIDI efekti** – Apstrādā MIDI protokola ziņojumus.

Attēls 2-7 Cubase MIDI Gate, VST MIDI efekts

2.4. Programmēšanas valoda

Steinberg izstrādātais VST SDK [1] ir C++ klašu kopums, kas balstīts uz C API. SDK ir brīvi lejupielādējams no Steinberg mājaslapas.

Ir pieejamas dažādas adaptācijas, piemēram, Delphi versija, kuru izstrādāja Frederic Vanmol, Java versija no jVSTwRapper projekta un .NET versija no Noise. Vēl viena .NET implementācija ir VST.NET. Šis, atvērtā koda projekts ietver arī programmēšanas ietvaru, kas atvieglo VST programmēšanu un palīdz strukturēt kodu.

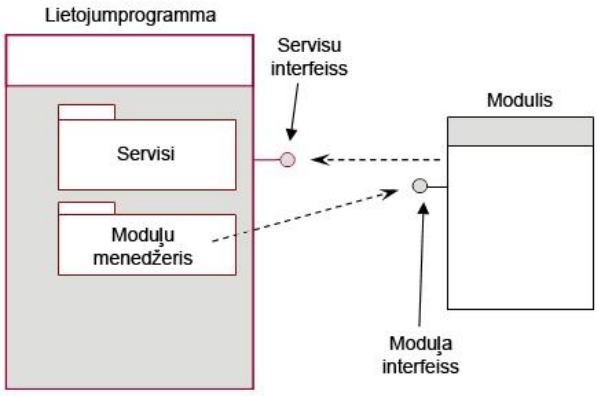
Ievērojama programmēšanas valoda, kas atbalsta VST ir Faust, kas ir speciāli radīta, lai programmētu moduļus digitālā signāla apstrādei. Līdz ar to bieži uzrāda labākus rezultātus ātrdarbībā kā C++.

Papildus Steinberg ir izstrādājuši VST GUI [2], kas arī C++ klašu kopums, ar kura palīdzību var veidot moduļu grafisko saskarni. Iekļautas ir pogu, bīdņu un citas klases. Tomēr šīs ir zema līmeņa klase, līdz ar to grafiskais izskats un animācija jāveido pašam izstrādātājam. Daudzi komerciālie un bezmaksas VST moduļi ir veidoti izmantojot Juce C++ ietvaru, pretēji izmantojot tiešus funkciju izsaukumus no VST SDK, lai varētu no viena koda veidot ne tikai VST interfeisa moduļus, bet arī AU, RTAS.

4. VST 2.4 skaņas efektu moduļu raksturojums

4.1. Vispārīgs raksturojums

Būtībā, VST efektu modulis ir tīra audio signāla datu apstrādes komponente, nevis pilnvērtīga lietojumprogramma. Tā ir komponente, kas tiek izmantota kā papildinājums kādai audio lietojumprogrammai. Audio lietojumprogramma nodrošina audio datu plūsmu, kas tālāk tiek apstrādā, izmantojot moduļa kodu. VST moduļu sasaiste ar lietojumprogrammu notiek pēc klasiska sadarbības principa, kas shematiski parādīts zemāk.



Attēls 4-1 Moduļu darbības princips

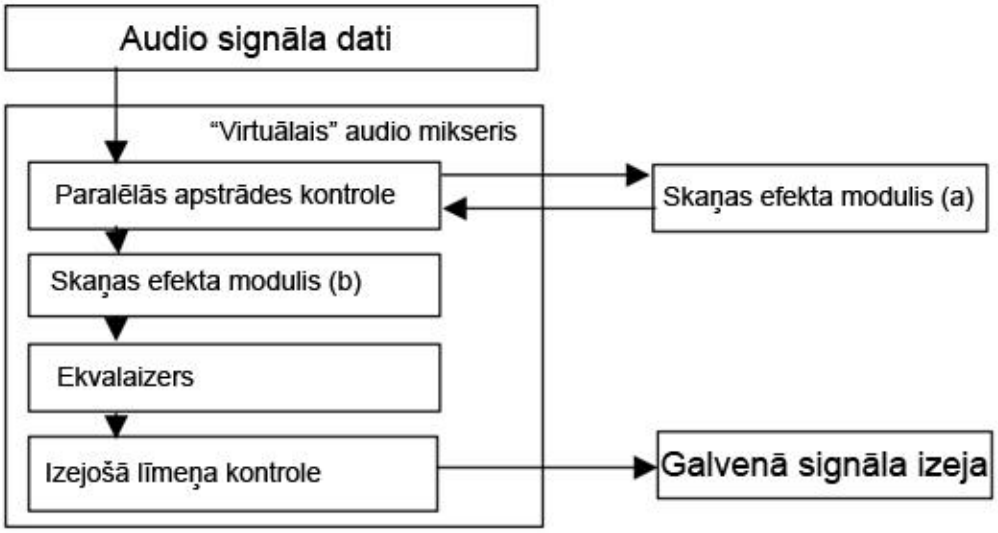
Īsāk sakot, lietojumprogramma nosūta datus modulim, kur tie tiek apstrādāti un atgriezti atpakaļ lietojumprogrammai. VST moduļi, koda izpildei, parasti izmanto datora CPU un tiem nav nepieciešami speciāli digitālā signāla apstrādes procesori. Digitālā audio signāla plūsma tiek sadalīta blokos, kurus lietojumprogramma secīgi piegādā modulim. Viena bloka izmēru nosaka un kontrolē pati lietojumprogramma un tai piesaistītās iekārtas. Par katra datu bloka apstrādi atbild vienīgi pats modulis, lietojumprogramma nesaglabā nekādu informāciju par datu bloku, ko modulis apstrādājis.

No lietojumprogrammas viedokļa, modulis ir kā „melnā kaste” ar patvaļīgu signāla ieeju un izeju skaitu (Audio vai MIDI) un noteiktiem kontroles parametriem. Lietojumprogrammai nav nepieciešams zināt neko par audio signāla datu apstrādi, ko veic modulis, lai izmantotu to. Modulis var izmantot jebkādus apstrādes procesa parametrus, kas definēti pašā modulī un atkarībā no lietojumprogrammas, kas to izmanto, tā var kontrolēt šos parametrus, veidojot to automātisko kontroli.

VST moduļa pirmkods ir platformneatkarīgs, taču kompilētais rezultāts ir atkarīgs no platformas arhitektūras. Windows vidē, VST modulis ir vairāku pavedienu DLL (Dynamic Link Library) datne. Noklusētā moduļu vietne tiek glabāta sistēmas reģistrā „HKEY_LOCAL_MACHINE\SOFTWARE\VST\VSTPluginsPath”.

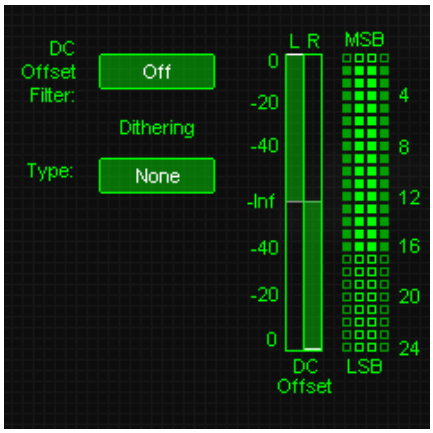
4.2. Audio signāla datu apstrāde

Audio signāla datu apstrāde tiek veikta, izmantojot vienu no trim VST SDK metodēm ar nosaukumiem *process(), processReplacing(), processDoubleReplacing()*. Metode *process()* pielieto, tajā definēto, datu apstrādes algoritmu un sasummē to ar izejas signālu. Savukārt *processReplacing()* un *processDoubleReplacing()* pilnībā aizstāj ienākošo signāla datu plūsmu un izvada apstrādāto signālu (pārraksta izejas buferi). Piemēram, Attēls 4-2 parāda tipisku audio signāla plūsmu lietojumprogrammā, kur moduļa (a) gadījumā tiks izsaukta metode process(), bet moduļa (b) gadījumā *processReplacing()* vai *processDoubleReplacing()*.



Attēls 4-2 Audio signāla plūsmas piemērs lietojumprogrammā

Audio dati, ko apstrādā VST modulis ir 32 bitu vai 64 bitu gari peldošā punkta dati. Ienākošā audio signāla datu plūsma ir atkarīga no audio bufera izmēra, respektīvi, tajā esošo nolašu skaita. Lai noskaidrotu kādā skaitliskā intervālā moduliem tiek padotas audio signāla datu vērtības, bija jāveic nelieli testi, jo dokumentācijā tas nav skaidri aprakstīts. Tā tika noskaidrots, ka svārstību amplitūda tiek raksturota, izmantojot vērtības diapazonā +/-1.



Attēls 4-3 Svārstību amplitūdas vērtību diapazona tests

Tomēr ir iespējams izmantot arī vērtības ārpus šī diapazona.

Visiem lietotājam pieejamie parametri, darbojas tieši vai netieši, ja tos ir automatizējusi pati lietojumprogramma. Tie ir 32 vai 64 bitu peldošā punkta dati. Šo parametru vērtībām vienmēr jābūt robežās no 0.0 līdz 1.0, neatkarīgi no to ārējās vai iekšējās reprezentācijas.

4.3. Lietotāja saskarne

Viss, kas saistās ar grafisko lietotāja saskarni, ir pilnībā nodalīts no audio signāla datu apstrādes. Ir iespējams arī nenodrošināt lietotāja saskarni pavisam, šajā gadījumā pati lietojumprogramma, kas izmanto šo moduli, izveido vienkāršu saskarni, kas attēlo ASCII simbolu virknes, kas raksturo parametru vērtības, apzīmējumus un nosaukumus. Šāda iespēja lieliski noder moduļa izstrādes un testēšanas procesā, jo ir iespējams to pilnvērtīgi izmantot un grafisko saskarni izstrādāt un pievienot vēlāk.

Nākošais lietotāja saskarnes līmenis ir, kad modulī ir definēta grafiskās saskarnes klase jeb redaktors. Tas dod praktiski neierobežotas iespējas izstrādāt ērtu un saprotamu grafisku lietotāja saskarni. Negatīvais aspekts tam ir tas, ka grafiskā saskarnes izstrādes laikā, ātri vien var nonākt līdz kodam, kas specifisks konkrētai platformai, līdz ar to, lai arī izstrādātai audio signāla datu apstrādes kods ir platformneatkarīgs, tad grafiskā saskarnes kods katrai platformai var nākties tikt modificēts.

Tieši tāpēc, lai risinātu šo problēmu, ir izveidoti programmēšanas ietvari, kas ļauj izstrādāt kodu, no kura var kompilēt grafisko saskarni praktiski jebkurai platformai, kas atbalsta VST. Tā, piemēram, VST GUI [2] ietvars, kas ir iekļauts VST SDK [1] komplektācijā, vai arī JUCE, kas ir ieguvis plašu popularitāti VST moduļu izstrādātāju vidū.

5. Praktiskā realizācija

Tā kā šī darba mērķis ir izpētīt skaņas efektu moduļu izstrādi, kas izstrādāti izmantojot VST SDK, tad ir nepieciešams arī praktiski mēģināt norealizēt kādu ideju. Praktiskās realizācijas process ļaus pilnvērtīgāk izprast VST SDK uzbūvi un iespējas, izstrādājot efektu moduļus. Praktiskajā sadaļā nav mērķis izstrādāt ļoti sarežģītu VST moduli, bet gan saprast izstrādes procesu, izstrādājot vienkāršas darbības principa moduli.

Sākumā mēģināšu teorētiski saprast, ko vēlos izstrādāt un kāds būs nepieciešamais algoritms. Pēc tam pētīšu veidus, kā iespējams šos algoritmu norealizēt un iegūt reālu rezultātu. Un visbeidzot, testēšu iegūto rezultātu, lai saprastu vai tas atbilst sākotnēji uzstādītajiem mērķiem un nav ieviesušās kļūdas. Tāpat arī mēģināšu izsecināt iespējamos trūkumus, kas būtu jāpilnveido izstrādes procesā.

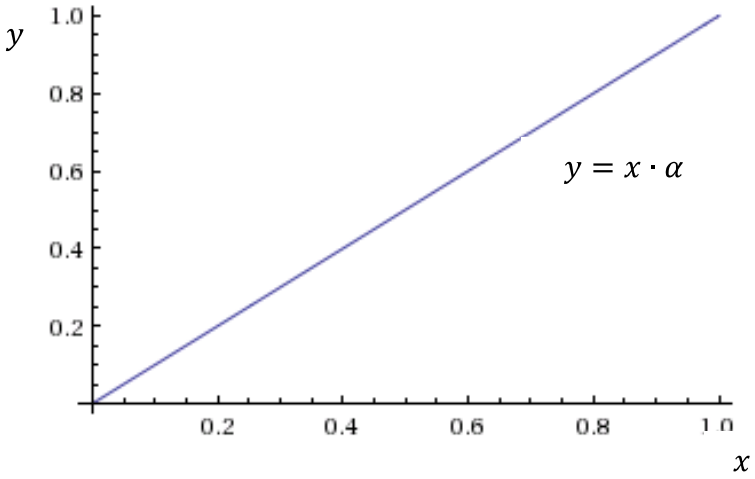
5.1. Teorētiskais plānojums

Lai izstrādātu kādu skaņas efektu moduli, izmantojot VST SDK, vispirms ir jāsaprot, kas ir tas rezultāts, ko vēlamies panākt un kā tam būtu jāskan. Tad jāmēģina atrast, vai jāizdomā savs algoritms skaņas apstrādei un jāmēģina, tas aprakstīt matemātiski, ar procesu raksturojošiem vienādojumiem vai algoritmiem.

Vēlējos, lai modulim ir skaņas skaļuma maiņas parametrs, kas maina ienākošā signāla skaļumu no 0 dB līdz –oo dB, kas ir pilnīgs klusums. Tā kā skaļuma maiņa ienākošajam signālam vēlējos aprakstīt ar lineāru vienādojumu, tad tas arī ir pietiekami vienkārši izdarāms ar formulu:

$$y = x \cdot \alpha$$

Kur y ir izejošā signāla līmenis, x ir ienākošā signāla līmenis, bet α ir koeficients, par kādu ienākošā signāla līmenis tiks izmainīts. Visiem šiem trim lielumiem, pēc VST SDK definīcijas, jābūt robežās no 0 līdz +1 ar C++ peldošā punkta skaitļa precizitāti, kas ir garumā 4 baiti un vienāds ar +/- 3.4e +/- 38 (~7 cipari). Tātad, ja ienākošā signāla līmenis ir konstants 0dB, jeb vērtība 1, tad skaļuma maiņas grafiks būtu kā Attēls 5-1



Attēls 5-1 Skaļuma līmeņa maiņas grafiks

Protams, reāli ienākošais signāls nekad nebūs konstants, tāpēc šis grafiks mainās atkarībā no tā.

Nākošais parametrs, ko vēlējos ieviest savā modulī ir stereo panorāmas balansētājs. Šo parametru vairs nevar raksturot tik vienkārši, izmantojot vienu lineāru vienādojumu. Kā jau minēts sadaļā 4.2, tad visu parametru vērtības ir pieļaujamas robežās no 0 līdz +1, no tā seko, ka stereo panorāmas viduspunkts būs vērtība 0.5. Ir jāsaprot, ka stereo panoramēšanas ideja ir, ka nobīdot to pa labi, pretējais stereo kanāls (kreisais) kļūst klusāks, attiecībā pret nobīdes pozīciju. Tieši tas pats notiek pie inversas situācijas. Tāpēc ir nepieciešami divi dažādi vienādojumi, kas katrs aprakta stereo panorāmas nobīdi uz savu pusi. Respektīvi, ja balanss ir nobīdīts pa labi, jeb parametra vērtība ir lielāka par 0.5, tad kreisā kanāla piereizināmajam koeficientam jāmainās robežās no 0 līdz +1, kur attiecīgi, ja 0, tad –oo dB, ja +1, tad 0 dB. Būtu vēlams, lai šo izmaiņu neraksturotu taisne, bet gan izliekta līkne, kas radītu reālistiskāku stereo panorāmas efektu.

Pēc nelielas izpētes internetā nonācu pie vienādojumiem, kas tieši raksturo augstāk minētos nosacījumus. Stereo panorāmas nobīdi pa labi var raksturot ar šādiem vienādojumiem:

$$y_L = sin(0.5 * \pi * \alpha)$$

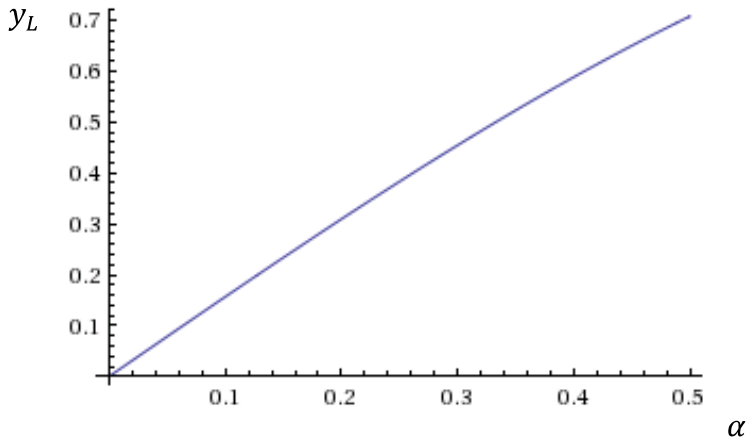
$$y_R = 1$$

Un attiecīgi nobīde pa kreisi:

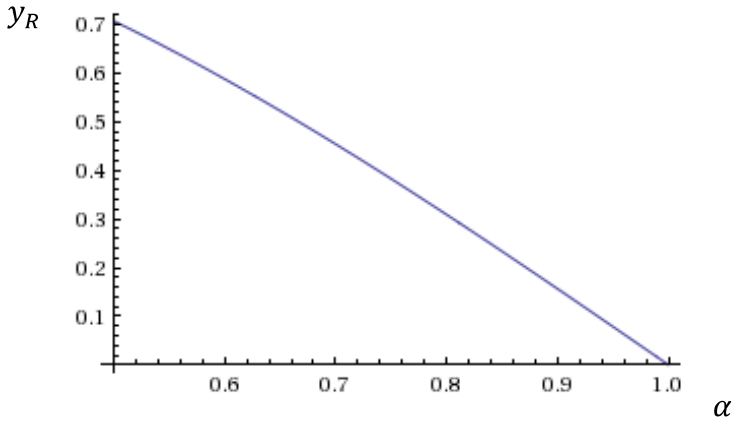
$$y_L = 1$$

$$y_R = cos(0.5 * \pi * \alpha)$$

Kur attiecīgi y_L un y_R apzīmē kreisā un labā stereo kanāla līmeņa izmaiņas koeficientu un α apzīmē nobīdes koeficienta vērtību. Funkcijas raksturojoši grafiki būtu kā Attēls 5-2, Attēls 5-3



Attēls 5-2 Stereo panoramēšanas pa labi, kreisā kanāla līmeņa izmaiņa



Attēls 5-3 Stereo panoramēšanas pa kreisi, labā kanāla līmeņa izmaiņa

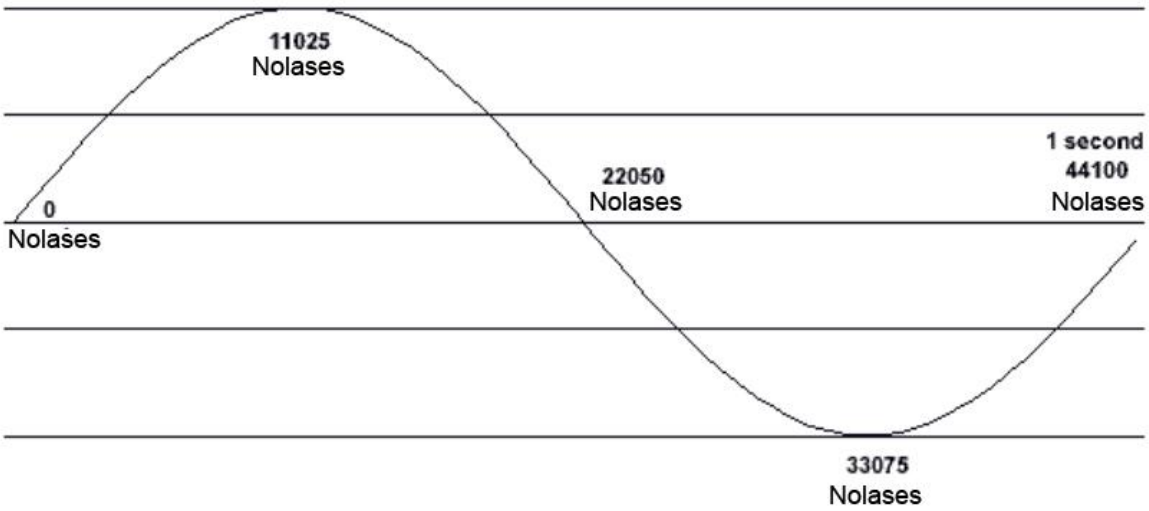
Jāatceras, ka pie centrētas stereo panorāmas abu kanālu līmeņi netiek izmainīti, respektīvi, koeficienti y_L un y_R ir vienādi ar +1, jeb 0 dB, pārrēķinot decibelos.

Turpinot par decibeliem, vēlējos, lai grafiskajā saskarnē skaļuma līmeņa izmaiņu attēlotu decibelos, ar vērtībām no 0 dB līdz –oo dB, tāpēc ir nepieciešama formula, kas pārrēķina vērtību diapazonu (0; +1) un attēlo to decibelos. Nonācu pie šādas formulas:

$$20 * \log_{10} \alpha$$

Kur α ir audio signāla līmenis vērtību diapazonā no 0 līdz +1.

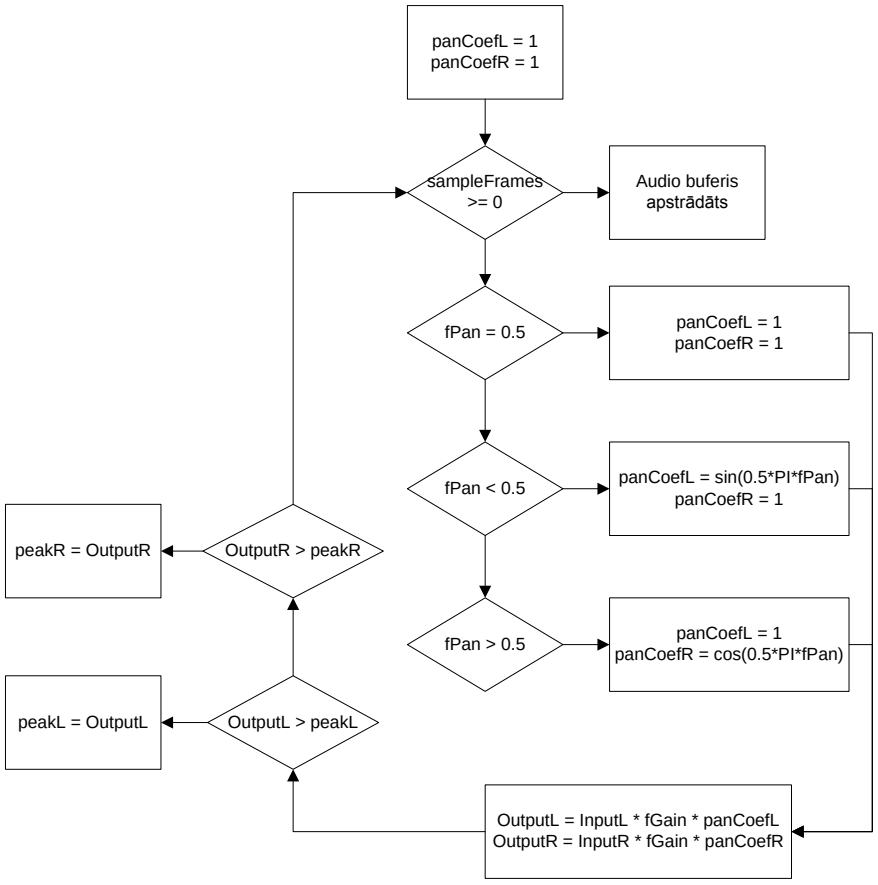
Pēdējais, ko šajā efektu modulī vēlējos iekļaut ir audio signāla pīķu līmeņa mērītājs. Zinot, ka vizuālu reprezentāciju nodrošina jau izstrādātas metodes, kas pieejamas VST GUI [2], tad nepieciešams ir saprast, kā tiek organizēt audio buferis un tas nodots efekta modulim. Digitālais audio signāls tiek kodēts nolasēs (*samples*) un to daudzumu sekundē izsaka nolašu daudzums (*sample rate*), atkarībā no uzstādījumiem, audio buferī tiek glabāts noteikt nolašu skaits, kas tad arī tiek padots modulim apstrādei. Lai labāk saprastu, kā tiek kodēts audio signāls, to attēloju shematiski.



Attēls 5-4 1 Hz svārstību perioda sadalījums nolasēs, ar nolašu biežuma frekvenci 44100 Hz

Tā kā vienas nolases ietvaros saņemto vērtību amplitūda var būt pilns spektrs un attēlot mērītājā vērtības ar atjaunošanās frekvenci, kas vienāda ar vienas nolasi ir lieki, tad pietiek glabāt, nolašu maksimālo vērtību, kura tiek inicializēta uz minimālo pie katras attēlošanas grafiskajā saskarnē. Pēc tam šī vērtība tiek lasīta, jau ar grafiskajā saskarnē uzstādīto nolases biežuma frekvenci.

Rezultātā iegūtā algoritma blokshēma ir šāda:



Attēls 5-5 Moduļa darbības algoritma blokshēma

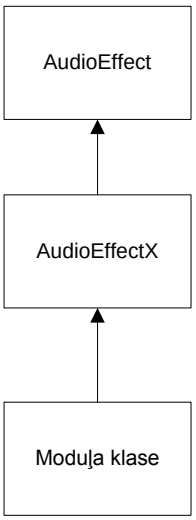
5.2. Izstrādes vide

Tā kā VST SDK ir C++ programmēšanas valodas klašu kopums, tad visloģiskāk programmēt moduļus būtu tieši programmēšanas valodā C++, lai arī ir pieejami ietvari, kas ļauj VST moduļus izstrādāt JAVA, Python, Delphi un citās programmēšanas valodās. Es izvēlējos un arī iesaku, tomēr izvēlēties C++ programmēšanas valodu, izstrādājot VST moduļus, jo tā ļauj veidot visoptimālāko kodu, jo nav nepieciešami nekādi pārveidojumi sadarbībai ar VST SDK.

Protams, uz dažādām platformām ir pieejamas dažādas izstrādes vides, bet būtiskākā komponente, kas ir nepieciešama ir C++ kompilators, pārējais ir gaumes jautājums. Šī testa projekta realizēšanai es izvēlējos Microsoft Visual C++ 2010 Express izstrādes vidi, kas ir bez maksas pieejama ikvienam.

5.3. Nepieciešamo SDK klašu metožu izvēle

5.3.1. Modulis



Attēls 5-6 Metožu mantošanas hierarhija

5.3.1.1. setProgramName(), getProgramName() - AudioEffect

Šīs metodes nodrošina izvēlētās programmas nosaukuma attēlošanu lietojumprogrammā, vai arī nolasa nosaukumu no tās. Abos gadījumos kā mainīgais tiek padots norāde uz simbolu virkni.

5.3.1.2. setParameter(), getParameter() - AudioEffect

Metode *setParameter()* saņem divus parametrus, kur pēc parametra *index* identificē parametru un attiecīgi uzstāda tā izmainīto vērtību. Savukārt metode *getParameter()* vienkārši atgriež aktuālo parametra vērtību, identificējot to pēc indeksa.

5.3.1.3. getParameterName() - AudioEffect

Metode *getParameterName()* pēc indeksa un norādes uz nosaukuma adresi atmiņā, nosaka un ievieto tā nosaukumu.

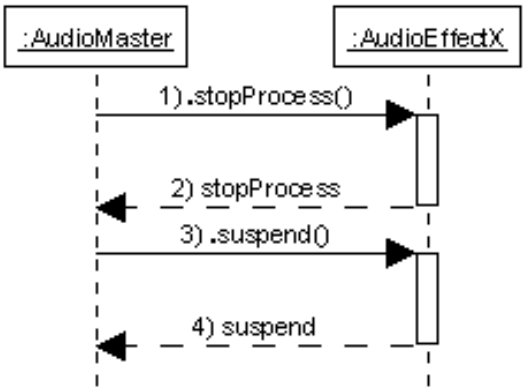
5.3.1.4. getParameterDisplay() - AudioEffect

Metode *getParameterDisplay()* pēc indeksa un norādes uz teksta adresi atmiņā, ievieto tā tekstu. Tā kā visi parametri ir skaitliskā formā, tad tos var nākties pārveidot teksta formā un šeit palīgā nāk metodes *dB2String()*, kas pārrēķina padoto vērtību decibelos un tad pārveido simbolu virknē un *float2String()*, kas peldošā punkta skaitli pārveido par simbolu virkni. Ir pieejamas arī citas pārveidošanas metodes, bet var izveidot arī savas.

5.3.1.5. getParameterLabel() - AudioEffect

Metode *getParameterLabel()* pēc indeksa un norādes uz nosaukuma adresi atmiņā, uzstāda parametra mērvienības.

5.3.1.6. suspend() , stopProcess() – AudioEffect



Attēls 5-7 Moduļa izslēgšanas metožu izsaukumi

Metode *suspend()* tiek izsaukta, ja modulis tiek izslēgts, bet *stopProcess()* tiek izsaukta, ja modulim vairs netiek padoti audio signāla dati.

5.3.1.7. getEffectName(), getProductString(), getVendorString(), getVendorVersion() - AudioEffectX

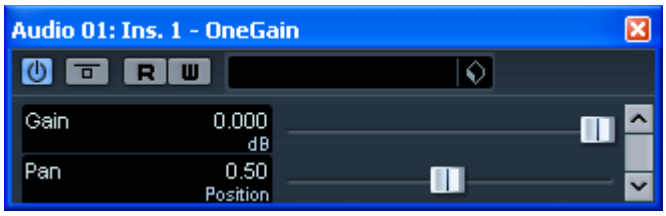
Šīm metodēm ir vairāk informatīvs nolūks, kas piedod tādu pabeigta moduļa izskatu. Tā piemēram getEffectName() uzstāda moduļa nosaukumu, kuru attēlos lietojumprogramma. Tad getProductString() uzstāda unikālu produkta identifikatoru, kuru var iegūt reģistrējot savu moduli kompānijas Steinberg mājas lapā. Tāpat *getVendorString()* un *getVendorVersion()* uzstāda attiecīgi moduļa izstrādātāju un versiju. Šos parametrus bieži izmanto lietojumprogrammas, lai strukturētu moduļus un grafiskās saskarnes informatīvajos laukos.

5.3.1.8. processReplacing() - AudioEffect

Tad nu beidzot esam nonākuši līdz metodei, kur notiek īstā „maģija", jeb audio signāla datu apstrāde. Metode saņem parametrus, kas ir norādes uz saņemto audio datu buferi un izejošo. Katra no šim norādēm rāda uz divdimensiju masīvu ar izmēru divi. Parasti šis norādes apzīmē ar *inputs, outputs*. Tāpat arī tiek saņemt parametrs, kas satur informāciju par to cik daudz nolases tiek turētas buferī. Šo parametru varam izmantot, lai noskaidrotu, kad esam beiguši apstrādāt buferī esošās nolases un varam beigt darbu. Parametru parasti apzīmē ar nosaukumu *sampleFrames.*

5.3.2. Grafiskā saskarne

VST moduļus ir iespējams lietot arī bez tiem speciāli izstrādātas grafiskās saskarnes, jo praktiski visas zināmās lietojumprogrammas, kas atbalsta VST moduļus, nodrošina elementāru parametru kontroles saskarni, arī ja nav izveidota speciāla grafiskā saskarne. Līdz ar to, tas ļoti atvieglo moduļu izstrādi, jo var izmēģināt un notestēt izstrādāto moduli, netērējot laiku tā grafiskās saskarnes izstrādei. Arī testa moduli sākumā nokompilēju bez grafiskās saskarnes, lai testētu tā darbības korektumu.

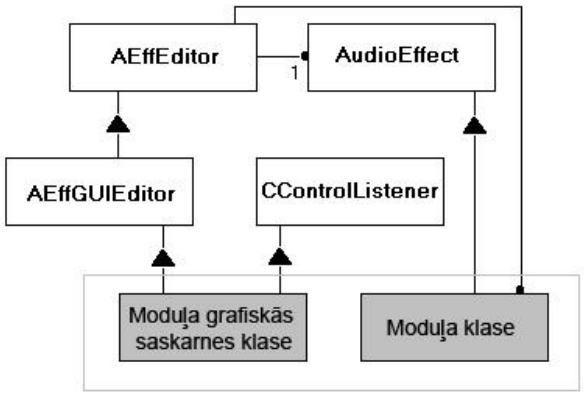


Attēls 5-8 "OneGain" modulis bez speciālas grafiskās saskarnes

Kā redzams attēlā Attēls 5-8 "OneGain" modulis bez speciālas grafiskās saskarnes, lietojumprogramma Cubase 5, bez problēmām arī moduļiem bez speciāli definētas grafiskās saskarnes, spēj izveidot vienkāršu parametru kontroles saskarni.

Tomēr lielākoties moduļu izstrādātāji izvēlas izstrādāt speciālas grafiskās saskarnes saviem moduļiem, jo tas ļauj būtiski uzlabot moduļa lietošanas ērtumu, ļauj realizēt unikālus kontroles risinājumus un visbeidzot, tā ir moduļa redzamā daļa, kas ir kā firmas zīme izstrādātājiem un spēj piesaistīt uzmanību un līdz ar to arī jaunus lietotājus. Tāpēc arī parauga modulim izstrādāju grafisko saskarni, kas satur tādus elementus, ko bez tās savādāk realizēt nemaz nevar.

Lai izstrādātu grafisko saskarni „OneGain" modulim nolēmu izmantot kompānijas Steinberg izveidoto C++ bibliotēku VST GUI [2], kas speciāli paredzēta VST un citu moduļu grafiskās saskarnes izveidei. Šī darba ietvaros izmantoju VST GUI 4.0.1 versiju.
Lai grafiskā saskarne veiksmīgi sadarbotos ar pašu moduli, jāievēro šāda klašu hierarhija:



Attēls 5-9 Klašu hierarhija grafiskās saskarnes izstrādei

5.3.2.1. open()

Metode *open()* nodrošina grafiskās saskarnes izveidi. Šīs metodes ietvaros tiek izveidoti visi saskarnes elementi, uzstādītas to noklusētās vērtības un pats galvenais, piesaistīti paša efekta parametri, kuru vērtības mainīsies atkarībā no grafiskās saskarnes elementu izmaiņām. Elementu veidi ir iepriekš definēti un pieejami VST GUI [2] bibliotēkā. Tie tiek definēti kā objekti ar saviem parametriem un metodēm. Tā kā VST GUI [2] bibliotēkā jau ir iekļauts atbalsts PNG paplašinājuma datnēm, grafiskajā saskarnē varam izmantot šī paplašinājuma attēlus. Lielākā priekšrocība šim formātam, izstrādājot grafisko saskarni, ir iespēja izmantot caurspīdīgumu. Pirmais, kas jāizveido, veidojot jaunu grafisko saskarni ir rāmja objekts *CFrame*, kuram tālāk pievienojam skata vai citu elementu objektus.

5.3.2.2. close()

Metode *close()* tiek izsaukta moduli aizverot. Piemēram, tajā ir svarīgi izdzēst rāmja objektu, ja tāds ir izveidots, bet to nevar darīt pa tiešo, jo pats objekts ir jāsaglabā, tāpēc izveidojam pagaidu objektu, kas norāda uz to, bet pašam piešķiram vērtību 0, lai lietojumprogrammai, nododot grafiskajai saskarnei informāciju par to, ka parametrs ir izmainīts, saskarne zinātu, ka nav atvērta un nekas nav jāmaina.

5.3.2.3. valueChanged()

Metode *valueChanged()* ir ļoti būtiska grafiskās saskarnes darbībā, jo tā nodrošina saiti ar pašu efekta moduli un nodod tam izmainītās parametru vērtības. Tā tiek izsaukt katru reizi, kad lietotājs veic kādas izmaiņas grafiskajā saskarnē. Šie izsaukumi tiek kontrolēti no virsklases *CControlListener*, kas tad arī nosaka, kurš elements, par kādu vērtību ir izmainīts un attiecīgi nodod šo informāciju metodei *valueChanged()*.

5.3.2.4. setParameter()

Metode *setParameter()* nodrošina atgriezenisko saiti modulim ar tā grafisko saskarni. Piemēram, ja lietojumprogramma ir automatizējusi kāda moduļa parametra vērtības maiņu noteiktā laika periodā, tad būtu vēlams, lai arī grafiskās saskarne mainītos līdzi, kuras moduļa parametram veic lietojumprogramma, bez grafiskās saskarnes starpniecības.

5.3.2.5. idle()

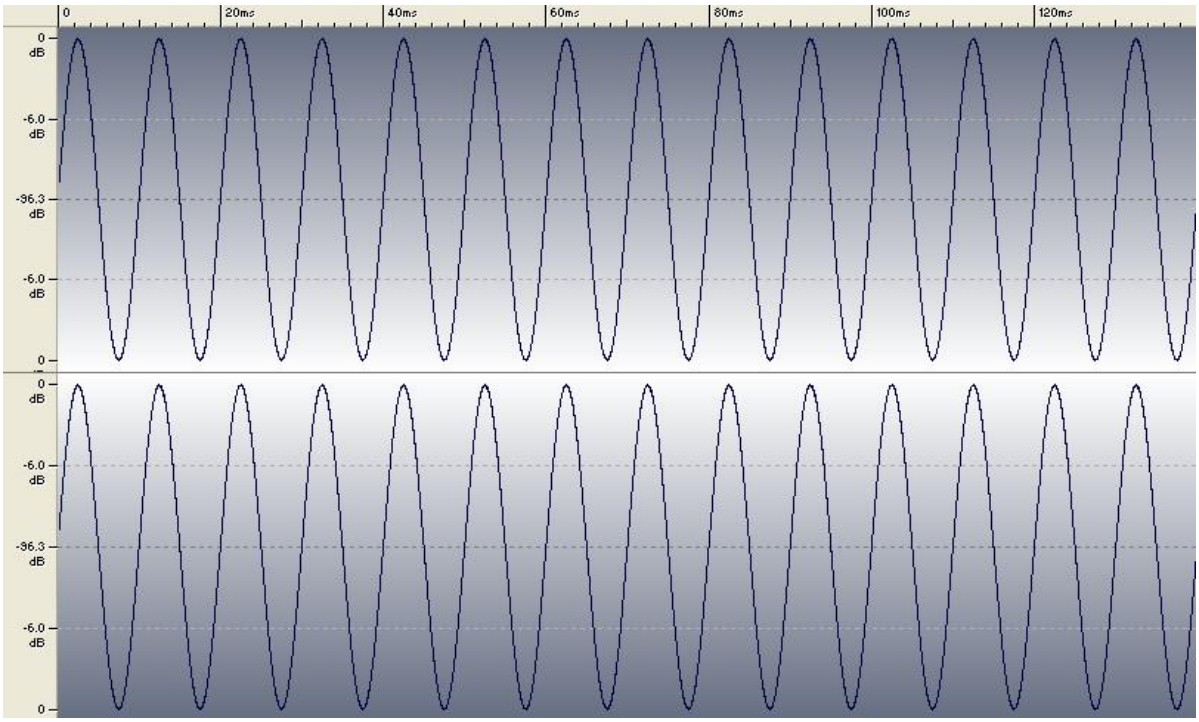
Būtiskākā metodes idle() īpašība ir, ka tā tiek izsaukta ar regulāru biežumu. Noklusētais biežums ir 30 Hz, kas ir 30 izsaukumi sekundē. Šī metode labi noder grafiskās saskarnes elementiem, kuru stāvoklis ir jāatjauno ar regulāru biežumu.

6. Rezultātu analīze

Rezultātā tika izstrādāts skaņas efektu modulis, izmantojot VST SDK 2.4 [1], kā arī tā grafiskā saskarne, izmantojot VST GUI [2].

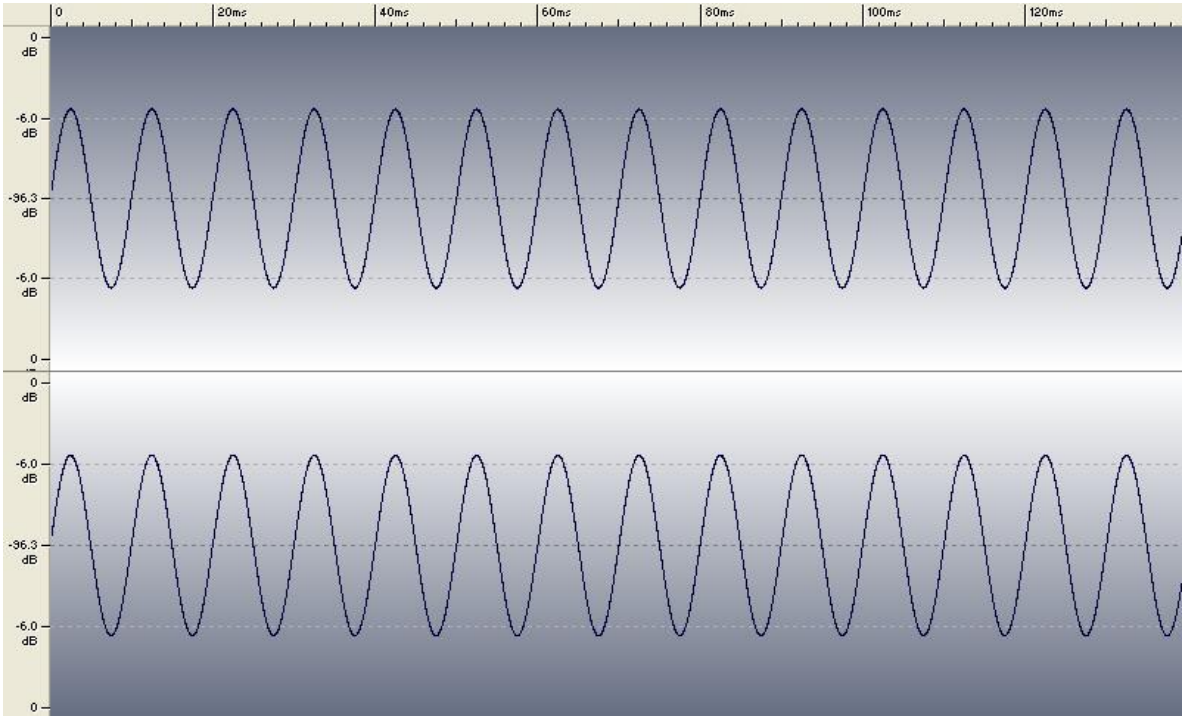
Ka tika plānots teorētiskajā plānojumā 5.1, tad moduļa galvenās īpašībām jābūt iespējai mainīt ienākošā signāla skaļuma līmeni no 0 dB, kas ir neizmainīts līmenis, līdz –oo, kas ir pilnīgi noklusināts ienākošā signāla līmenis. Tāpat modulī bija jābūt iespējai nobīdīt stereo panorāmu no centrētas līdz pilnīgi kreisai vai labai. Grafiskajā saskarnē bija jāiekļauj reāla laika skaļuma līmeņa mērītājās.

Lai pārliecinātos par skaļuma līmeņa maiņas darbības korektumu, veicu pārbaudi par atskaiti ņemot sinusoīdu ar svārstību frekvenci 100 Hz un amplitūdu 0 dB. Tā kā modulis apstrādā stereo signālu, tad abos signāla kanālos tika ģenerēta identiska sinusoīda.



Attēls 6-1 Sinusoīdas 100 Hz, 0 dB

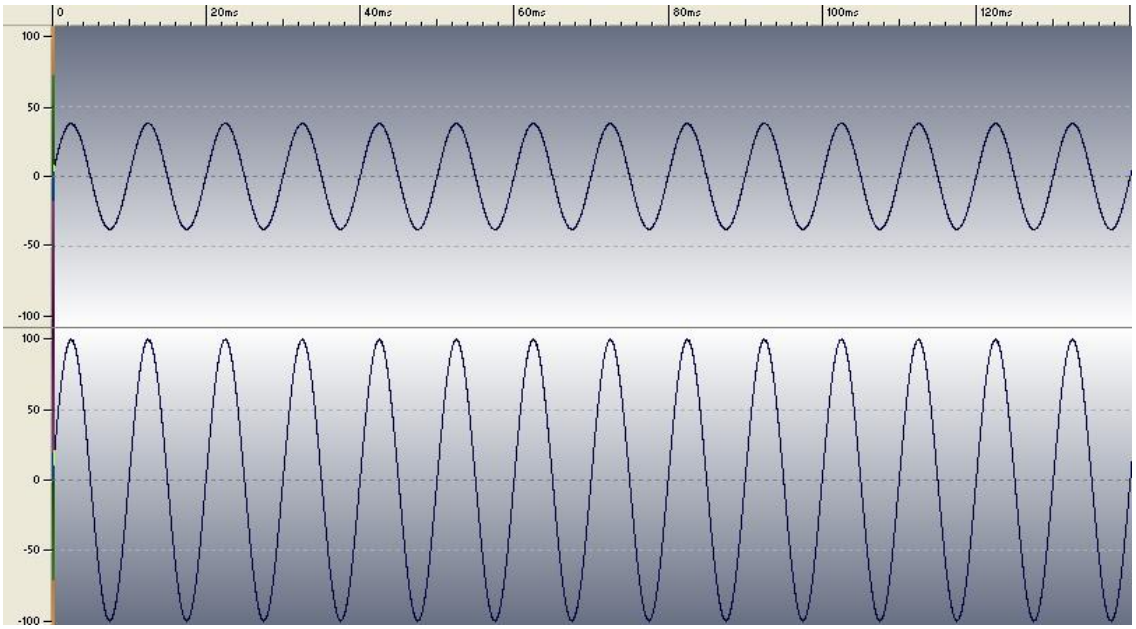
Izmainot modulī skaļuma līmeni uz -5 dB, arī sinusoīdu amplitūdai vajadzētu samazināties par 5 dB.



Attēls 6-2 Sinusoīdas 100 Hz, -5 dB

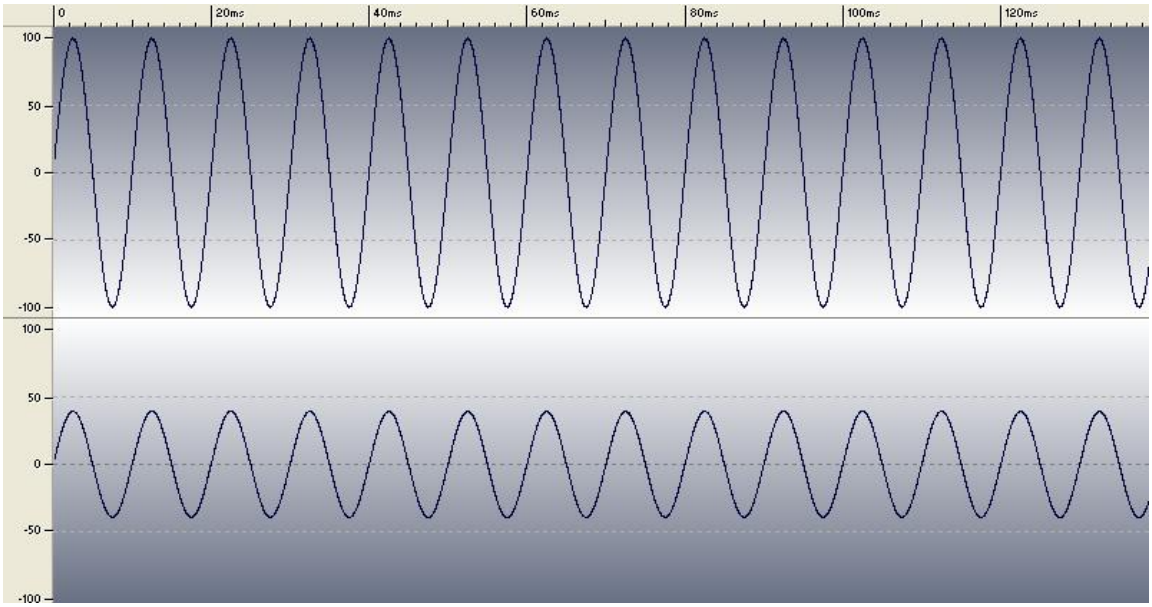
Kā redzams grafikā, tad samazinot modulī skaļuma līmeni par 5 dB, tad arī abu sinusoīdu amplitūda samazinājās par 5 dB, kas arī bija mērķis, izstrādājot skaļuma līmeņa maiņas parametru modulim.

Kā nākamais parametrs, kas jāpārbauda ir stereo panorāmas nobīde gan pa labi, gan pa kreisi. Sāksim ar nobīdi pa labi. Izmantosim to pašu 100 Hz, 0 dB sinusoīdu. Nobīdot stereo panorāmu pa labi, labajam stereo kanālam vajadzētu palikt nemainīgam, taču kreisajam vajadzētu tikt samazinātai svārstību amplitūdai. Testam nobīdīsim stereo panorāmu 50% pa labi.



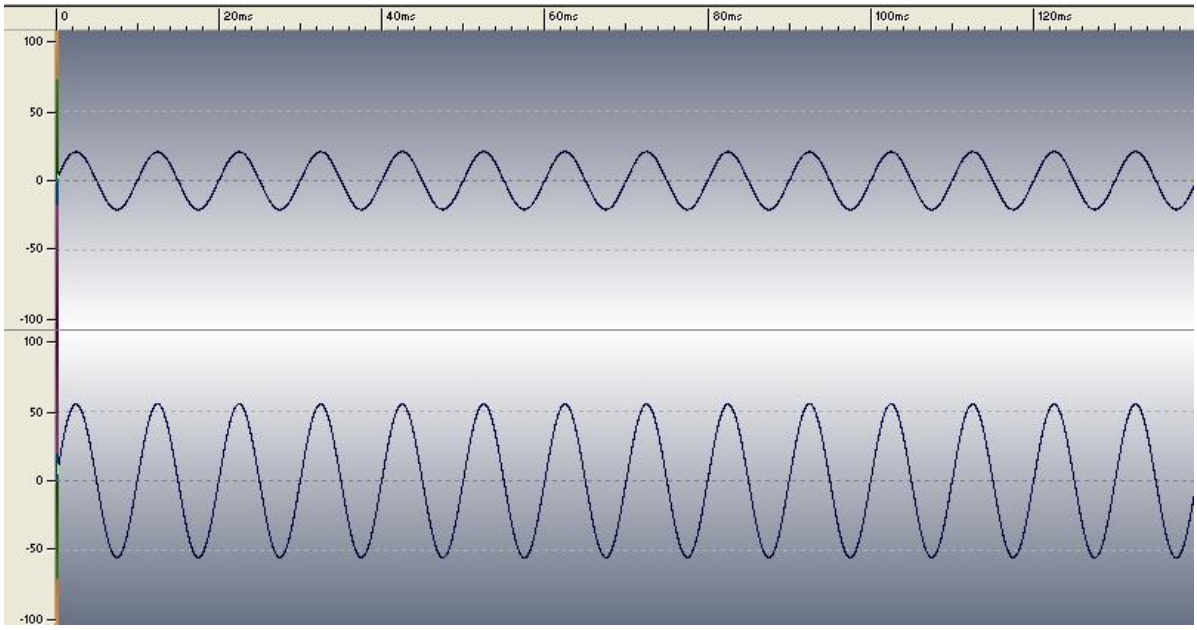
Attēls 6-3 Sinusoīdas 100 Hz, 0 dB, 50% pa labi

Kā tika paredzēts, tad labais kanāla amplitūda netika izmainīta, savukārt kreisā kanāla amplitūda tika samazināt nedaudz vairāk par 50%, tas tāpēc, ka pielietojam nelineāru vienādojumu amplitūdas koeficienta rēķināšanai. Pretēju rezultātu iegūstam, ja stereo panorāmu nobīdām 50% pa kreisi.



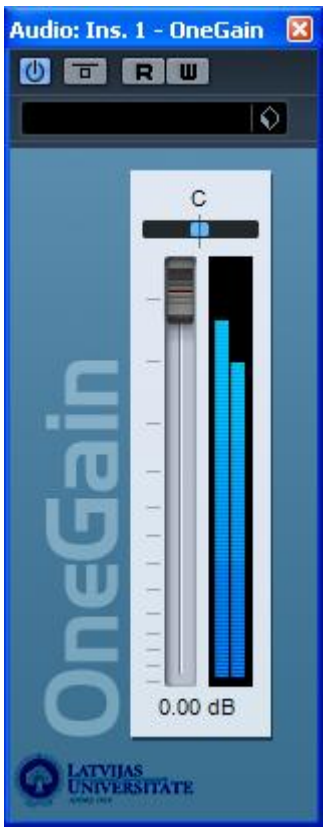
Attēls 6-4 Sinusoīdas 100 Hz, 0 dB, 50% pa kreisi

Pārbaudīsim arī parametru sadarbību. Piemēram, samazināsim kopējo skaļuma līmeni par 5 dB un nobīdīsim stereo panorāmu 50% pa labi.



Attēls 6-5 Sinusoīdas 100 Hz, -5 dB, 50% pa labi

Kā var redzēt grafikā, tad vispirms ir samazināta abu kanālu signāla līmenis un tad attiecīgi kreisā kanāla līmenis samazināts vēl par 50%, jo stereo panorāma tika nobīdīta pa labi.



Attēls 6-6 Izstrādātā moduļa grafiskā saskarne

Aplūkosim izstrādāto moduļa grafisko saskarni. Kā var redzēt Attēls 6-6, tad moduļa grafiskā saskarne satur divus parametrus, kas ir vertikāli bīdāms skaļuma regulators un horizontāli bīdāms stereo panorāmas regulators. Tāpat saskarne satur divus informācijas laukus, kur viens attēlo skaļuma līmeņa izmaiņu decibelos, bet otrs stereo panorāmas nobīdes virzienu un nobīdes vērtību procentos. Pa labi no skaļu līmeņa maiņas regulatora atrodas izejošā signāla līmeņa stereo mērītājs. Šim mērītājam gan vairāk ir informatīva nozīme, jo nav speciāli kalibrēts, lai tā rādījumi būtu ļoti precīzi.

Visbeidzot modulis tika testēts vairākās lietojumprogrammās, kas atbalsta VST, lai pārliecinātos par tā savietojamību. Izvēlētās lietojumprogrammas bija Cubase 5, Ableton Live 8 un WaveLab 6. Visās šajās lietojumprogrammās modulis tika atpazīts un varēja atvērt tā instanci. Galvenās atšķirības tika novērotas skaņas līmeņa pīķu mērītājā, kura dinamika Cubase 5 lietojumprogrammā bija jūtami ātrāka. Skaņas kvalitātes ziņā atšķirības netika saklausītas.

7. Secinājumi

Lai arī VST SDK [1] versija 2.4 nav pati jaunākā, tomēr tā vēl joprojām ir pati izplatītākā un to atbalsta visvairāk lietojumprogrammas, tāpēc to izvēlējos moduļa izstrādei. Iepazīstot pašu SDK nācās secināt, ka tā automātiski ģenerētā dokumentācija ir grūti saprotama, ja nav iepriekšējas pieredzes šāda veida izstrādes procesā. Tāpēc praktiskās izstrādes posmā nācās diezgan daudz eksperimentēt un strādāt ar kļūdu paziņojumiem, lai atrastu veidus kā tos novērst.

Darba praktiskās izstrādes laikā sapratu, ka būtiskas ir labas C++ programmēšanas valodas zināšanas, tāpēc nācās arī atsvaidzināt savējās, lai varētu veikt praktisko izstrādes procesu. Protams, ir iespējams izstrādāt moduļus arī citās programmēšanas valodās, tomēr rekomendējama ir tieši C++ programmēšanas valoda.

Tā kā darba mērķis nebija izstrādāt sarežģītu moduli, bet gan fokusēties uz veiksmīgas izstrādes procesu, tad kopumā, darba rakstīšanas laikā izstrādātā testa moduļa izstrādi var uzskatīt par veiksmīgu, jo gala rezultātā modulis bija pilnībā funkcionējošs un saderīgs ar lietojumprogrammām, kas atbalsta VST standartu.

Turpmākie pētījumu virzieni varētu būt ne tieši saistīti ar VST standartu un ar to saderīgu moduļu izstrādi, bet tieši digitālā audio signāla apstrādi un algoritmu izstrādi, kurus tad pēc tam var norealizēt VST moduļa veidā, kas ir savietojams ar daudzām digitālā audio signāla apstrādes lietojumprogrammām. Tāpat plaša un interesanta sfēra, kur turpināt pētījumus ir VST instrumentu moduļu izstrāde, kur parādās skaņas sintēzes algoritmu pielietojums.

8. Izmantotā literatūra un avoti

1. Steinberg 3rd party developer [tiešsaiste]. Pieejams: http://www.steinberg.net/en/company/developer.html
2. VSTGUI Sourceforge repozitorijs [tiešsaiste]. Pieejams: http://sourceforge.net/projects/vstgui/
3. C++ VST Plugin Tutorials [tiešsaiste]. Pieejams: http://learnvst.wordpress.com/
4. Teragon Audio: How to make VST Plugins in Visual Studio [tiešsaiste]. Pieejams: http://www.teragonaudio.com/2010/02/how-to-make-vst-plugins-in-visual.html
5. Euphoria Audio - Steinberg Example VST Plugin Walkthrough [tiešsaiste]. Pieejams: http://www.euphoriaaudio.com/tutorials/vstexample/vstexample7.php
6. KVR Forum: DSP and Plug-in Development [tiešsaiste]. Pieejams: http://www.kvraudio.com/forum/viewforum.php?f=33
7. Plugin Developer [tiešsaiste]. Pieejams: http://www.plugindeveloper.com/
8. Sampling rate - Wikipedia, the free encyclopedia [tiešsaiste]. Pieejams: http://en.wikipedia.org/wiki/Sampling_rate
9. Virtual Studio Technology - Wikipedia, the free encyclopedia [tiešsaiste]. Pieejams: http://en.wikipedia.org/wiki/Virtual_Studio_Technology
10. Definition and Explanation of a .DLL file [tiešsaiste]. Pieejams: http://support.microsoft.com/kb/87934
11. A brief history of computer music [tiešsaiste]. Pieejams: http://www.musicradar.com/news/tech/a-brief-history-of-computer-music-177299

9. Pielikumi

9.1. „OneGain” moduļa pirmkods

9.1.1. OneGain.h

```
//-------------------------------------------------------------------------------------------------------
// Versija 0.1
//
// Kategorija          : VST 2.x SDK Modulis
// Datnes nosaukums : OneGain.h
// Izstrādātājs               : Ilmārs Kazeks
// Apraksts                   : Stereo modulis, kas maina signālā līmeni robežās [-oo, 0dB]
//                                    Stereo panorāmas nobīdīšana, robēžas (100L, C, 100R)
//                                    Signāla līmeņa mērītājs.
//-------------------------------------------------------------------------------------------------------

#ifndef __OneGain__
#define __OneGain__

#include "public.sdk/source/vst2.x/audioeffectx.h"
#include "OneGainEditor.h"

#define PI 3.14159265

//Parametri
enum
{
	kGain = 0,
	kPan,
	kNumParams
};

//-------------------------------------------------------------------------------------------------------
class OneGain : public AudioEffectX
{
public:
	OneGain (audioMasterCallback audioMaster);
	~OneGain ();

	// Apstrāde
	virtual void processReplacing (float** inputs, float** outputs, VstInt32 sampleFrames);
	virtual void suspend();
	virtual VstInt32 stopProcess();

	// Programmas
	virtual void setProgramName (char* name);
	virtual void getProgramName (char* name);

	// Parametri
	virtual void setParameter (VstInt32 index, float value);
	virtual float getParameter (VstInt32 index);
	virtual void getParameterLabel (VstInt32 index, char* label);
	virtual void getParameterDisplay (VstInt32 index, char* text);
	virtual void getParameterName (VstInt32 index, char* text);
	virtual bool getEffectName (char* name);
	virtual bool getVendorString (char* text);
	virtual bool getProductString (char* text);
	virtual VstInt32 getVendorVersion ();

protected:
	float fGain;
	float fPan;
	char programName[kVstMaxProgNameLen + 1];
};
#endif
```

9.1.2. OneGain.cpp

```
//-------------------------------------------------------------------------------------------------------
// Versija 0.1
//
// Kategorija          : VST 2.x SDK Modulis
// Datnes nosaukums : OneGain.cpp
// Izstrādātājs               : Ilmārs Kazeks
// Apraksts                   : Stereo modulis, kas maina signālā līmeni robežās [-oo, 0dB]
//                                    Stereo panorāmas nobīdīšana, robežas (100L, C, 100R)
//                                    Signāla līmeņa mērītājs.
//-------------------------------------------------------------------------------------------------------
#include "OneGain.h"

// Globālie mainīgie priekš skaļuma līmeņa grafiskā mērītāja sasaistes
// ar audio signāla datu plūsmu
float peakL, peakR = 0;

//-------------------------------------------------------------------------------------------------------
AudioEffect* createEffectInstance (audioMasterCallback audioMaster)
{
	return new OneGain (audioMaster);
}

//-------------------------------------------------------------------------------------------------------
OneGain::OneGain (audioMasterCallback audioMaster)
: AudioEffectX (audioMaster, 1, 2)	// 1 programma, 2 parametri
{
	setNumInputs (2);		// stereo ieeja
	setNumOutputs (2);		// stereo izeja
	setUniqueID ('Gain');  // identifikators
	canProcessReplacing ();		// atbalsta signāla plūsmas aizstāšanu

	fGain = 1.f;			// noklusētais koeficients 0 dB
	fPan = 0.5f;			// noklusētais koeficients "Centrēts"

	vst_strncpy (programName, "Default", kVstMaxProgNameLen);	// noklusētais programmas
nosaukums

	editor = new OneGainEditor (this); // izveido grafiskās saskarnes instanci
}

//-------------------------------------------------------------------------------------------------------
OneGain::~OneGain ()
{
	// šoreiz te nekas nav jādara
}

//-------------------------------------------------------------------------------------------------------
void OneGain::setProgramName (char* name)
{
	vst_strncpy (programName, name, kVstMaxProgNameLen);
}

//-----------------------------------------------------------------------------------------
void OneGain::getProgramName (char* name)
{
	vst_strncpy (name, programName, kVstMaxProgNameLen);
}

//-----------------------------------------------------------------------------------------
void OneGain::setParameter (VstInt32 index, float value)
{
	switch(index)
	{
		case kGain:	fGain = value; break;
		case kPan:
			// Noapaļo vērtību līdz 2 cipariem aiz komata
			float prec = pow(10.f,2);
			value = ceil(value*prec)/prec;
			fPan = value;
			break;
	}

	if (editor)
		((AEffGUIEditor*)editor)->setParameter (index, value);
}
```

```
//-----------------------------------------------------------------------------------------
float OneGain::getParameter (VstInt32 index)
{
	switch(index)
	{
		case kGain: return fGain; break;
		case kPan: return fPan; break;
	}
	return 0.f;
}

//-----------------------------------------------------------------------------------------
void OneGain::getParameterName (VstInt32 index, char* label)
{
	switch(index)
	{
		case kGain: vst_strncpy (label, "Gain", kVstMaxParamStrLen); break;
		case kPan: vst_strncpy (label, "Pan", kVstMaxParamStrLen); break;
	}
}

//-----------------------------------------------------------------------------------------
void OneGain::getParameterDisplay (VstInt32 index, char* text)
{
	switch(index)
	{
		case kGain:
			//Pārrēķinam fGain vērtību decibelos
			dB2string (fGain, text, 5);   break;
		case kPan:
			float2string(fPan, text, 4); break;
	}

}

//-----------------------------------------------------------------------------------------
void OneGain::getParameterLabel (VstInt32 index, char* label)
{
	switch(index)
	{
		case kGain: vst_strncpy (label, "dB", kVstMaxParamStrLen); break;
		case kPan: vst_strncpy (label, "Position", kVstMaxParamStrLen); break;
	}
}

//------------------------------------------------------------------------
bool OneGain::getEffectName (char* name)
{
	vst_strncpy (name, "OneGain", kVstMaxEffectNameLen);
	return true;
}

//------------------------------------------------------------------------
bool OneGain::getProductString (char* text)
{
	vst_strncpy (text, "OneGain", kVstMaxProductStrLen);
	return true;
}

//------------------------------------------------------------------------
bool OneGain::getVendorString (char* text)
{
	vst_strncpy (text, "Ilmars Kazeks", kVstMaxVendorStrLen);
	return true;
}

//-----------------------------------------------------------------------------------------
VstInt32 OneGain::getVendorVersion ()
{
	return 1000;
}

void OneGain::suspend()
{
	peakL = peakR = 0.f;
}
```

```
VstInt32 OneGain::stopProcess()
{
	peakL = peakR = 0.f;
	return 1;
}


//-----------------------------------------------------------------------------------------
void OneGain::processReplacing (float** inputs, float** outputs, VstInt32 sampleFrames)
{
    float* in1  =  inputs[0];
    float* in2  =  inputs[1];
    float* out1 = outputs[0];
    float* out2 = outputs[1];

	float panCoefL = 1.f;
	float panCoefR = 1.f;

    while (--sampleFrames >= 0)
    {
		// Stereo panorāmas nobīdes koeficienta aprēķināšana
		if(fPan == 0.5f)
		{
			panCoefL = 1.f;
			panCoefR = 1.f;
		}
		if(fPan < 0.5f)
		{
			panCoefL = (float)sin(0.5*PI*fPan);
			panCoefR = 1.f;
		}
	    if(fPan > 0.5f)
		{
			panCoefL = 1.f;
			panCoefR = (float)cos(0.5*PI*fPan);
		}

		// Signāla skaļuma maiņa (0db; -oo) un panoramizācija
		(*out1++) = (*in1++) * fGain * panCoefL;
	    (*out2++) = (*in2++) * fGain * panCoefR;

		// Audio skaļuma pīķa noteikšana konkrētajai nolasei
		if(*(out1-1) > peakL){peakL = *(out1-1);}
		if(*(out2-1) > peakR){peakR = *(out2-1);}
	}
}
```

9.2. „OneGain” moduļa grafiskās saskarnes pirmkods

9.2.1. OneGainEditor.h

```
//-------------------------------------------------------------------------------------------------
// Versija 0.1
//
// Kategorija          : VST GUI
// Datnes nosaukums : OneGainEditor.h
// Izstrādātājs               : Ilmārs Kazeks
// Apraksts                   : Grafiskā saskarne OneGain modulim
//-------------------------------------------------------------------------------------------------

#ifndef __OneGainEditor__
#define __OneGainEditor__

#include "plugin-bindings/aeffguieditor.h"

class OneGainEditor : public AEffGUIEditor, public CControlListener
{
public:
        OneGainEditor (void*);
        virtual ~OneGainEditor();

        // Mantotais no AEffGUIEditor
        bool open (void* ptr);
        void close ();

        void setParameter (VstInt32 index, float value);
        void valueChanged (CControl* pControl);

        void idle();

private:
        // Kontroles
        CVerticalSlider* kGainFader;
        CHorizontalSlider* kPanFader;
        CParamDisplay* kGainDisplay;
        CParamDisplay* kPanDisplay;
        CVuMeter* vuMeterL;
        CVuMeter* vuMeterR;

        // Norādes skaļuma līmeņa mērītājam
        float* toPeakL;
        float* toPeakR;
};


#endif
```

9.2.2. OneGainEditor.cpp

```
//-------------------------------------------------------------------------------------------------------
// Versija 0.1
//
// Kategorija          : VST GUI
// Datnes nosaukums : OneGainEditor.h
// Izstrādātājs               : Ilmārs Kazeks
// Apraksts                   : Grafiskā saskarne OneGain modulim
//-------------------------------------------------------------------------------------------------------

#ifndef __OneGainEditor__
#include "OneGainEditor.h"
#endif

#ifndef __OneGain__
#include "OneGain.h"
#endif

#include <stdio.h>
#include "public.sdk/source/vst2.x/audioeffectx.h"

extern float peakL;
extern float peakR;

enum {
	// Grafiskās saskarnes attēli
	OneGainBGId = 128,
	GainBGId,
	GainCapId,
	PanBGId,
	PanCapId,
	vuOnId,
	vuOffId,

	// Pozīcijas
	GainFaderX = 76,
	GainFaderY = 54,
	PanFaderX = 66,
	PanFaderY = 36,

	GainDisplayX = 68,
	GainDisplayY = 272,
	GainDisplayXWidth = 53,
	GainDisplayHeight = 15,

	PanDisplayX = 68,
	PanDisplayY = 18,
	PanDisplayXWidth = 53,
	PanDisplayHeight = 15,

	vuMeterX = 102,
	vuMeterY = 57
};

//-------------------------------------------------------------------------------
bool float2dBConvert (float value, char string[256], void* userData)
{
	if(value <= 0)
		sprintf (string, "-oo dB");
	else
		sprintf (string, "%.2f dB", (float)(20. * log10 (value)));
	return true;
}

bool float2PanConvert (float value, char string[256], void* userData)
{
	if(value == 0.5)
		sprintf (string, "C");
	else if(value == 0)
		sprintf (string, "R 100");
	else if (value < 0.5f)
		sprintf (string, "R %.0f", ((0.5-value)/0.5)*100);
	else
		sprintf (string, "L %.0f", ((value-0.5)/0.5)*100);
	return true;
}
```

```
//-----------------------------------------------------------------------------------------
AEffGUIEditor* createEditor (AudioEffectX* effect)
{
	return new OneGainEditor (effect);
}

//-----------------------------------------------------------------------------------------
OneGainEditor::OneGainEditor (void* ptr)
: AEffGUIEditor (ptr)
{
	// Saskarnes objekti
	kGainFader = 0;
	kPanFader = 0;
	kGainDisplay = 0;
	kPanDisplay = 0;
	vuMeterL = 0;
	vuMeterR = 0;

	// Inicializē moduļa izmēru
	rect.left   = 0;
	rect.top    = 0;
	rect.right  = 153;
	rect.bottom = 336;

	// Skaļuma līmeņa mērītāja vērtību norādes
	toPeakL = &peakL;
	toPeakR = &peakR;
}

OneGainEditor::~OneGainEditor ()
{
}

//-----------------------------------------------------------------------------------------
bool OneGainEditor::open (void* ptr)
{
	// !!! Vienmēr jāizsauc virsklase no kuras mantotā funkcija tiek pārslogota !!!
	AEffGUIEditor::open (ptr);

	//-- Izveidojam jaunu rāmi un skatu ar fona attēlu
	CRect frameSize (0, 0, 153, 336);
	CFrame* newFrame = new CFrame (frameSize, ptr, this);
	CBitmap* OneGainBG = new CBitmap (OneGainBGId);
	CView* myView = new CView (CRect (0, 0, OneGainBG->getWidth (), OneGainBG->getHeight
()));
    myView->setBackground (OneGainBG);
	myView->idleRate = 100;
	newFrame->addView (myView);
	OneGainBG->forget();

	// -- Skaļuma līmeņa regulators
	// Attēli
	CBitmap* GainHandle = new CBitmap (GainCapId);
	CBitmap* GainBody = new CBitmap (GainBGId);

	// Pozīcija
	int minPos = GainFaderY;
	int maxPos = GainFaderY + (int)GainBody->getHeight () - (int)GainHandle->getHeight() +
10;

	// Regulators
	CRect size (GainFaderX, GainFaderY, GainFaderX + GainBody->getWidth (), GainFaderY +
GainBody->getHeight ());
	kGainFader = new CVerticalSlider (size, this, kGain, minPos, maxPos, GainHandle,
GainBody);
	kGainFader->setValue (effect->getParameter(kGain));
	kGainFader->setDefaultValue(1.f);
	newFrame->addView (kGainFader);
	GainHandle->forget();
	GainBody->forget();

	// -- Stereo panoramas regulators
	// Attēli
	CBitmap* PanHandle = new CBitmap (PanCapId);
	CBitmap* PanBody = new CBitmap (PanBGId);
```

```
        // Pozīcija
        minPos = PanFaderX;
        maxPos = PanFaderX + (int)PanBody->getWidth () - (int)PanHandle->getWidth();

        // Regulators
        size (PanFaderX, PanFaderY, PanFaderX + PanBody->getWidth(), PanFaderY + PanBody-
>getHeight());
        kPanFader = new CHorizontalSlider (size, this, kPan, minPos, maxPos, PanHandle,
PanBody);
        kPanFader->setValue (effect->getParameter(kPan));
        kPanFader->setDefaultValue(0.5f);
        newFrame->addView (kPanFader);
        PanHandle->forget();
        PanBody->forget();

        // -- Skaļuma līmeņa vērtības attēlojums
        size (GainDisplayX, GainDisplayY, GainDisplayX + GainDisplayXWidth, GainDisplayY +
GainDisplayHeight);
        kGainDisplay = new CParamDisplay (size, 0, kCenterText);
        kGainDisplay->setFont (kNormalFontSmall);
        kGainDisplay->setFontColor (kBlackCColor);
        kGainDisplay->setTransparency(true);
        kGainDisplay->setValue (effect->getParameter(kGain));
        kGainDisplay->setValueToStringProc(float2dBConvert);
        newFrame->addView (kGainDisplay);

        // -- Stereo ponorāmas pozīcijas attēlojums
        size (PanDisplayX, PanDisplayY, PanDisplayX + PanDisplayXWidth, PanDisplayY +
PanDisplayHeight);
        kPanDisplay = new CParamDisplay (size, 0, kCenterText);
        kPanDisplay->setFont (kNormalFontSmall);
        kPanDisplay->setFontColor (kBlackCColor);
        kPanDisplay->setTransparency(true);
        kPanDisplay->setValue (effect->getParameter(kPan));
        kPanDisplay->setValueToStringProc(float2PanConvert);
        newFrame->addView (kPanDisplay);

        // -- Skaļuma līmeņa mērītājs
        // Attēli
        CBitmap* vuOnBitmap  = new CBitmap (vuOnId);
        CBitmap* vuOffBitmap = new CBitmap (vuOffId);

        // Kreisā kanāla meritajs
        size (vuMeterX, vuMeterY, vuMeterX + vuOnBitmap->getWidth(), vuMeterY + vuOnBitmap-
>getHeight() - 1);
        vuMeterL = new CVuMeter (size, vuOnBitmap, vuOffBitmap, 120);
        vuMeterL->setDecreaseStepValue (0.01f);
        newFrame->addView(vuMeterL);

        // Labā kanāla mērītājs
        size.offset(8, 0);
        vuMeterR = new CVuMeter (size, vuOnBitmap, vuOffBitmap, 120);
        vuMeterR->setDecreaseStepValue (0.01f);
        newFrame->addView(vuMeterR);

        vuOnBitmap->forget ();
        vuOffBitmap->forget ();

        frame = newFrame;
        return true;
}

//-----------------------------------------------------------------------------------------
void OneGainEditor::close ()
{
        //-- Aizverot moduli, nepieciešams izdzēst rāmi.
        //-- Pirms izdzēšam rāmi uzstādām tam vertību 0, lai gadījumā, ja setParameter()
        //-- mēģin atjaunot to, nenotiktu kļūda
        CFrame* oldFrame = frame;
        frame = 0;
        oldFrame->forget ();
}

//-----------------------------------------------------------------------------------------
void OneGainEditor::valueChanged (CControl* pControl)
{
        //-- Nododam informāciju efektam, ka grafiskajā saskarnē ir meinītās parametru vērtības
        effect->setParameterAutomated (pControl->getTag (), pControl->getValue ());
```

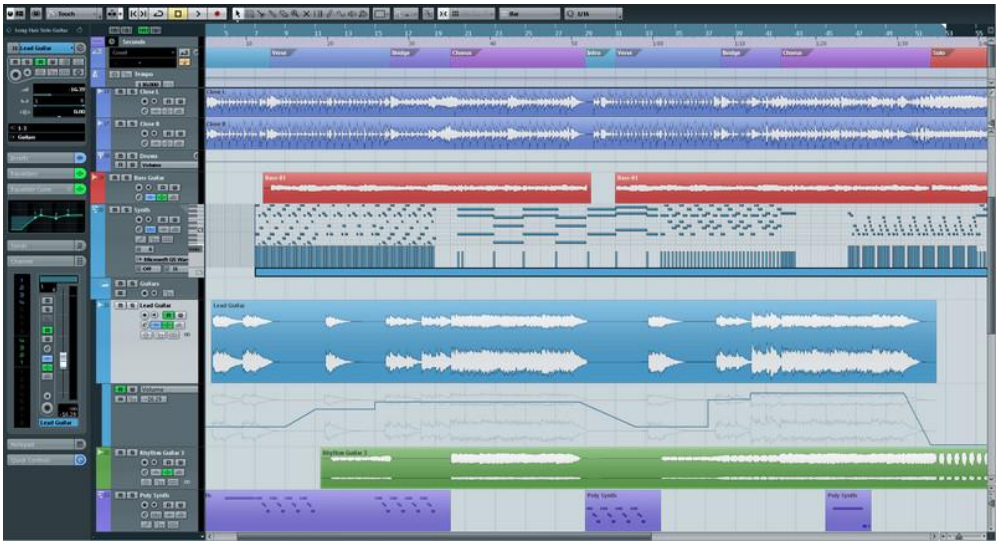
```
}

//-----------------------------------------------------------------------------------------
void OneGainEditor::setParameter (VstInt32 index, float value)
{
	//-- setParameter tiek izsaukta, kad lietojumprogramma automatizējusi kādu parametru
	//-- Tāpēc arī grafiskajai saskarnei jāreaģē uz to un jāizmaina elementu vērtības
	if (frame && index < kNumParams)
	{
		switch(index)
		{
			case kGain:
				kGainFader->setValue (value);
				kGainDisplay->setValue (value);
				break;
			case kPan:
				kPanFader->setValue(value);
				kPanDisplay->setValue (value);
				break;
		}
	}
}

void OneGainEditor::idle()
{
	AEffGUIEditor::idle();
	vuMeterL->setValue(*toPeakL);
	vuMeterR->setValue(*toPeakR);
	*toPeakL = *toPeakR = 0;
}
```

9.3. Populārākās lietojumprogrammas, kas atbalsta VST standartu

Šajā nodaļā pieminētas tikai dažas no daudzajām lietojumprogrammām, kas atbalsta VST standartu.

Cubase

Saite: http://www.steinberg.net/en/products/cubase/start.html



Attēls 9-1 Steinberg Cubase 6

Ableton Live/Suite

Saite: http://www.ableton.com/

Attēls 9-2 Ableton Live 8

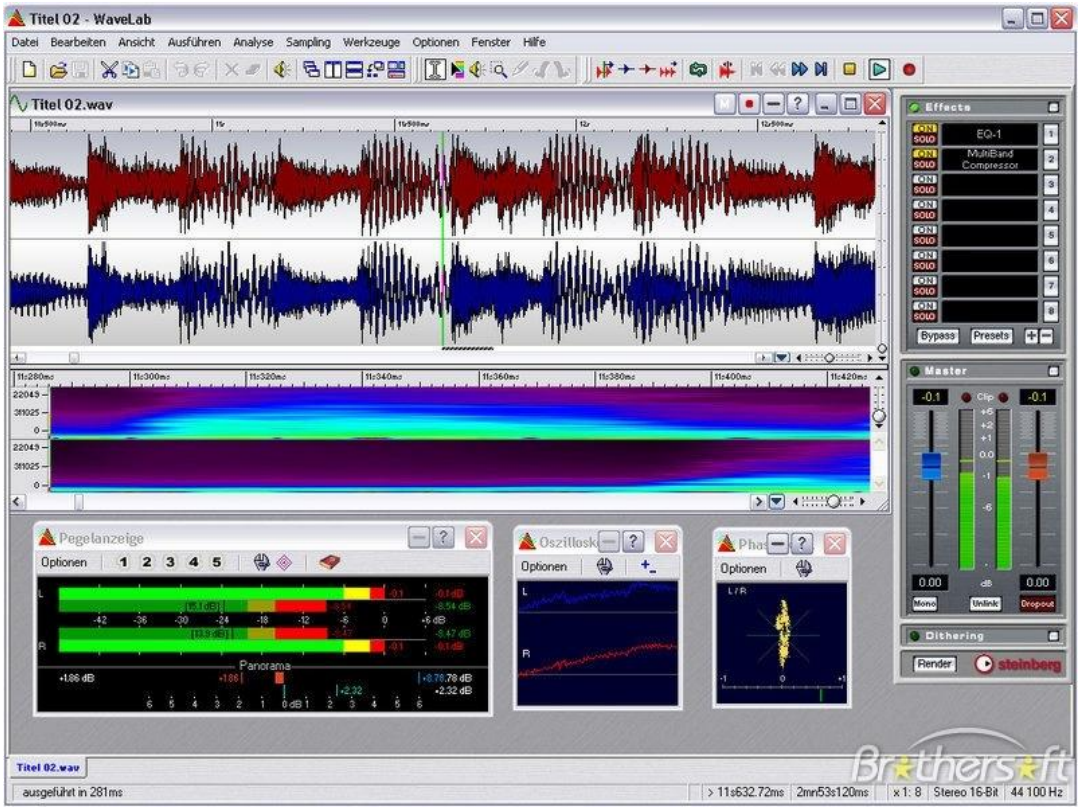
Sony Sound Forge

Saite: http://www.sonycreativesoftware.com/soundforgesoftware

Attēls 9-3 Sony Sound Forge 10

WaveLab

Saite: http://www.steinberg.net/en/products/wavelab.html



Attēls 9-4 WaveLab 6

Dokumentārā lapa

Bakalaura darbs „Skaņas efektu moduļu izstrāde, izmantojot VST SDK” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Ilmārs Kazeks _____ 28.05.2012.

Rekomendēju darbu aizstāvēšanai
Vadītājs: Dr. dat. Jānis Zuters _____

Recenzents: _____

Darbs iesniegts Datorikas fakultātē 28.05.2012.
Metodiķe:

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē
__.06.2012. prot. Nr. __, vērtējums ___ (_____)

Komisijas sekretārs/sekretāre: _____