Izrada programa za virtualni efekt u audio produkciji

Ćurić, Krešimir

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:867858

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krešimir Ćurić

Izrada programa za virtualni efekt u audio produkciji

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krešimir Ćurić

Matični broj: 44940/16-R

Studij: Informacijski sustavi

Izrada programa za virtualni efekt u audio produkciji

ZAVRŠNI RAD

Mentor:

Dr. sc. Mladen Konecki

Varaždin, rujan 2019.

Krešimir Ćurić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Audio inženjering nezaobilazan je dio glazbenog svijeta te glazbene industrije kao takve no i svakodnevnog života danas. Rapidan razvoj digitalne glazbene opreme povećao je pristupačnost audio inženjeringa široj korisničkoj zajednici što je rezultiralo povećanjem interesa, a u konačnici i povećanjem tržišta audio inženjeringa te glazbene industrije. Audio inženjering i razvoj virtualnih efekata strahovit je potencijal i svakako mu se treba pružiti veća pozornost što nastoji bar djelomično učiniti ovaj završni rad.

Ključne riječi: zvuk; audio inženjering; glazba; distorzija; predpojačalo; vst; virtualni efekt;

Sadržaj

1. Uvod

Glazba je već tisućljećima neizostavan dio ljudskoga života, ljudske socijalizacije te shvaćanja svijeta oko sebe kao takvog. Začeta u malim plemenskim zajednicama kao prijenos emocija te razmišljanja putem napetih opna i grla nastavila je pratiti civilizacije sve do danas. U početku je bila izvođena isključivo uživo, dakako zbog manjka tehnologije, a danas imamo tu privilegiju ponovno preslušavati ono nekad uživo izvedeno. Uistinu, rijetko tko zastane i razmisli o toj činjenici – o činjenici da je reprodukcija zvučnih ili bilo kakvih drugih materijala zaista luksuz, luksuz koji danas uživamo.

U 18. stoljeću ljudi su otpočeli s razvojem tehnologija za snimanje zvuka. Spomenuta napeta opna bubnja snimljena su već 1921. godine od strane Robert Sutherland Rattraya na cilindru napravljenom od voska. Snimka je nastala u Ghani, a ista se čuva danas Velikoj Britaniji te predstavlja jedno od najvećih blaga audio inženjerstva [1]. Uz snimku iz Ghane u to vrijeme nastalo je još nekoliko zvučnih zapisa na cilindrima od voska - kasnijih godina tehnologija je rapidno uznapredovala pa se zvuk počeo nasnimavati na magnetske vrpce što je bio revolucionaran korak audio inženjerstva. Godinu po godinu, pogonjeno britkim idejama te povijesnim okolnostima kvaliteta snimljenog zvuka postepeno se podizala. Naposljetku razvojem računala te njihovim unaprjeđenjima zvuk se snimio u digitalnom obliku kao niz nula i jedinica.

Računala su ta koja su gotovo u potpunosti zamijenila magnetske trake te svu popratnu studijsku opremu koja je prije bila korištena. Danas, toliki niz godina kasnije – velika većina zvučnih zapisa snimljena su upravo pomoću računala te adekvatnih programa. Količina kućnih studija te entuzijasta za snimanje zvuka nikada nije bila veća, a s porastom interesa dolazi i porasta dotične industrije. Sva studijska oprema koja je nekad korištena sada je replicirana u obliku računalnih programa, imajući to na umu ne čudi da je prodaja analognih komada studijske opreme znatno opala dok je prodaja njihovih virtualnih replika dotakla svoj dosadašnji vrhunac. Tomu je primarno tako jer su virtualne replike, odnosno računalni programi daleko jeftiniji za proizvesti, a time onda i za kupiti – za razliku od svojih analognih preteča. Gledajući dakle s ekonomske strane, tržište za audio inženjering postoji i njegovi korisnici su zaista vjerni istom – no, važnost glazbe kojoj je audio inženjering medij za lakše i brže dopiranje do ljudi u potpunosti je neizostavna. Prodaja virtualnih replika ili pak analognih preteča opreme za snimanje industrija je za sebe i tržišno ima svoj veliki udio, ali prilika da gotovo svatko u bilo kojem kutku svijeta iznese svoje ideje, emocije i težnje u rekordno brzom vremenu putem glazbe i računala prava je vrijednost audio inženjeringa kakvog danas poznajemo – audio inženjeringa pogonjenog računalima.

Osobno sam jedan od entuzijasta audio inženjeringa. Od svoje desete godine počeo sam izrađivati svoje zvučne snimke. Prvotno u potpunosti trivijalne, a kasnije ponešto kompleksnije. Veliku količinu vremena sam provodio i provodim u raznim glazbenim studijima i moja volja i želja za audio inženjeringom i glazbom čini se neće tako lako splasnuti. Studirajući na fakultetu počeo sam gajiti veliku ljubav spram programiranju i činilo mi se potpuno logično spojiti ju sa starim hobijem – audio inženjeringom. Upravo od tuda dolazi motivacija za pisanje i obrađivanje ove teme.

U nadolazećim stranicama nastojat ću opisati izradu programa za virtualni efekt u audio produkciji. Konkretno bavit ću se izradom virtualnog efekta predpojačala. Predpojačalo je jedna od komponenata gitarskog pojačala koje između ostalog modificira zvuk na tako da oblikuje frekvencije ulaznog gitarskog signala na različite načine. Težište i fokus virtualnog efekta kojeg ću izraditi bit će na distorziji – jednog od modifikatora zvuka kojeg predpojačalo može pružati te uglavnom uvijek pruža. Odluka koji virtualni efekt izraditi nije bila nimalo jednostavna, no s obzirom na to da sam gitarist - s predpojačalima, samim time i distorzijama imao sam i više negoli mnogo iskustva.

Kako je ranije spomenuto virtualni efekti uvelike zamjenjuju studijsku opremu, no isti zamjenjuju i gitarističku pa tako i ostalu opremu raznih instrumentalista. Virtualni efekt pojačala odnosno predpojačala za gitaru daleko je jeftiniji za izraditi od pravog, analognog dvojnika. Pridodamo li tomu još i činjenicu da virtualni efekt teži 0 kilograma, zaista se radi o „ugodnijem“ komadu opreme.

2. Metode i tehnike rada

Za izradu virtualnog efekta odlučio sam koristiti programski jezik C++. Tomu je tako jer smo na fakultetu utrošili dobar dio vremena učeći isti, a i gledajući sa strane performansi C++ zasigurno je dostojan zadatku. Dodamo li još činjenicu i da je programski jezik C++ gotovo pa industrijski standard za izradu virtualnih efekta te ostalih programskih rješenja vezanih uz audio inženjering te audio industriju kao takvu, odabir nimalo ne čudi.

Izraditi virtualni efekt bilo kojeg tipa ne bi bilo nimalo jednostavno da se isti izrađuje bez ikakvog programskog okvira, naime programiranje određenih funkcionalnosti bilo bi poprilično teško, dugotrajno te repetitivno. Svaki virtualni efekt ima nekoliko zajedničkih koncepata, a isti su se preslikali iz svojih analognih preteča, izuzev njih nekoliko. Tako primjerice svaki virtualni efekt ima određen broj ulaza te izlaza. Kada govorimo o ulazima i izlazima mislimo na ulazne i izlazne utore za zvučne signale – u analognom svijetu ti ulazi i izlazi bili bi manifestirani kao utori za kablove, dok promatrajući digitalno radi se o logičkim ulazima i izlazima koji rukuju digitalnim signalima zvuka. Svaki virtualni efekt radi na određenoj računalnoj arhitekturi – bilo 32 bitnoj ili 64 bitnoj. Nadalje postoji još nekoliko zajedničkih koncepata, no istih ću se dotaknuti kada za to dođe vrijeme. Bit je da bi se svi ti zajednički koncepti iznova i iznova morali programirati kada god bi se izrađivao neki virtualni efekt, a s obzirom na to da se ti koncepti dotiču vrlo „niske“ razine programiranja takav pothvat bio bi zaista težak. Srećom, za programski jezik C++ izrađeno je rješenje u obliku programskog okvira JUCE koji u potpunosti ima pokrivene spomenute zajedničke funkcionalnosti „niske“ razine. Na osobi koja izrađuje virtualni efekt ostaje „samo“ izraditi grafičko sučelje te isprogramirati samu logiku iza konverzije ulaznog zvučnog signala u izlazni.

Kako se svaki virtualni efekt uglavnom koristi u svrhe modificiranja zvuka tijekom snimanja - za testiranje i korištenje istih potrebno je imati neki programski alat za snimanje i baratanje zvukom. U mome slučaju taj alat bit će besplatni programski alat Ardour te komercijalni alat Cubase 5. Oba alata nude mogućnosti snimanja, uređivanja te reproduciranja snimljenih zvučnih signala, a upravo se ti zvučni signali unutar spomenutih alata te svih ostalih programskih ekvivalenata oblikuju uz pomoć virtualnih efekata.

Za pisanje te uređivanje samog programskog koda koristit ću Visual Studio 2017. Sam kod može se pisati i unutar uređivača teksta samog programskog okvira JUCE, no znatno bolje se snalazim u programskom alatu Visual Studio 2017 pa sam odlučio kod ne pisati u uređivaču teksta kojeg nudi programski okvir JUCE.

Virtualni efekt mora imati i grafičko sučelje pa ću komponente istog izraditi uz pomoć besplatnog programskog alata GIMP.

3. Audio inženjering

Audio inženjering se bavi snimanjem, uređivanjem te reprodukcijom zvuka pomoću mehaničkih ili elektroničkih sredstava. Kao takav smatra se podgranom znanosti o zvuku, no zbog svoje specijalizacije nije okupiran aktivnostima zvuka unutar prostora u smislu kontrole buke odnosno zvučne izolacije i slično. Audio inženjering u svojoj primjeni također je dakako uvelike prožet i fizikom, prosto jer je središnja okupacija samog – zvuk, fizička pojava.

3.1. Povijest snimanja zvuka

Različiti autori povijest snimanja zvuka dijele na različite načine, no napravi li se presjek njihovih razmišljanja povijest snimanja zvuka može se grupirati u tri ere obilježene značajkama vrijednima spomena, kronološkim redom to bi bile:

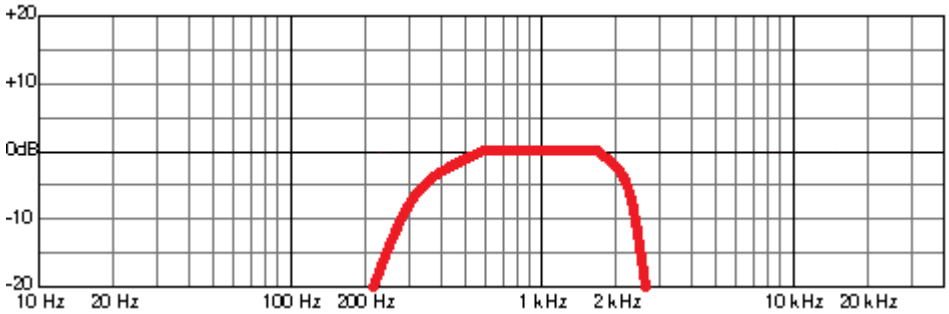
1. Akustična era
2. Magnetska era
3. Digitalna era

Svaka od spomenutih era prožeta je ljudima te idejama koje su oblikovale audio inženjering, no i sam svijet koji danas poznajemo. U nastavku slijedi pregled istih radi dobivanja konteksta u koji spada audio inženjering te specijalizirano virtualni efekti kao takvi.

3.1.1. Akustična era

Pri samome početku svi izumi odnosno uređaji za snimanje zvuka bili su posve mehanički te uvelike ograničeni. Svaki od tih uređaja zvuk bi snimao tako da bi prikupljao zračni pritisak stvoren od strane zvučnih valova te bi kao reakciju na taj pritisak pomoću posebne igle urezivao reprezentaciju zvučnih valova u određeni materijal, primjerice najčešće vosak. Urezani podaci vizualno su predstavljali snimljeni zvuk, no isti se u samome početku nije mogao reproducirati – svejedno, pothvat „zaustavljanja“ zvuka na određeni medij predstavio je revolucionaran iskorak. Snimljeni zvuk mogao se tumačiti na način da su glasniji zvukovi uzrokovali veću amplitudu, a viši zvukovi veću frekvenciju ucrtavanja. Nešto kasnije izumom fonografa od strane Thomas Alva Edisona [2] zvuk se naposljetku mogao i reproducirati. Problem koji se ukazao kod fonografa te svih ostalih izuma ove ere bio je taj što je zvuk koji se snimao morao biti poprilično glasan, a i frekvencijski spektar koji se mogao zabilježiti bio je vrlo uzak. Istraživanjem literature mišljenje autora se uglavnom podudara da su tadašnji uređaji mogli snimiti odnosno zabilježiti frekvencijski spektar između 250 te 2500 herca. Prilažem graf koji prikazuje frekvencijski spektar koji ljudsko uho može čuti, na istom sam

aproksimativno crvenom bojom ucrtao udio spektra koji su tadašnji uređaji mogli zabilježiti – zaista se dade uvidjeti da je to vrlo šturi komad. Na apscisi se nalaze frekvencije (Hz), dok je na ordinati amplituda istih izražena u decibelima (db).



Slika 1: Aproksimativni frekvencijski spektar fonograma

Frekvencijski spektar koji se mogao zapisati uvelike se proširio nakon što se rog koji je do tada hvatao zvuk zamijenio prvim inačicama mikrofona kakvog danas poznajemo u eri koji neki autori nadodaju te nazivaju električnom erom. Istraživanja te napori za napredak su se nastavili te je otpočela Magnetska era snimanja zvuka.

3.1.2. Magnetska era

Iako je i prije 1930. godine bilo pokušaja snimanja zvuka na osnovi energije magnetskog polja niti jedan od tih pokušaja nije bio posve uspješan. Otprilike 1930. godine u Njemačkoj je razvijen snimač koji je radio na principu magnetske trake, isti taj snimač nije bio dostupan ostatku svijeta zbog povijesnih razloga sve do pada nacionalsocijalističke Njemačke. Snimanje pomoću magnetske trake u potpunosti je promijenilo frekvencijski spektar koji se mogao ovjekovječiti, uz to amplituda ulaznih signala odnosno samog zvuka više nije morala biti niti približno velika kao kod primjerice fonograma. Već 1950. godine pa nadalje magnetska traka postala je de facto standardni medij za zvuk. Istraživanja su se nastavila, a tehnologija za snimanje zvuka rapidno je napredovala. Otkrivena je tehnologija za snimanje više kanalnih snimaka [2] iste su se mogle i uređivati prostim rezanjem traka i lijepljenjem istih i slično. Kada govorimo o višekanalnom snimanju tu govorimo o broju zvukova koji snimljeni odvojeno mogu završiti u jednoj finalnoj snimci, reproducirajući se pritom paralelno. Analogan scenarij u programiranju bila bi primjerice višedretvenost gdje je svaka dretva zadužena za određeni posao pri čemu se poslovi izvode istovremeno, paralelno. Osim višekanalnih snimača u ovoj eri razvijeni su još mnogi uređaji za modificiranje zvuka koji je pridolazio kroz mikrofone, tako su primjerice razvijeni ekvilajzeri - uređaji za izjednačavanje amplitude frekvencija u

frekvencijskom spektru, zatim kompresori – uređaji za izjednačavanje dinamike samog snimljenog zvuka te mnogo drugih uređaja koji se i dan danas koriste. Također, upravo je u ovoj eri snimljeno veliko bogatstvo glazbenih uradaka koji se i danas rado preslušavaju poput – The Beatlesa, The Rolling Stonesa, Jimi Hendrixa te mnogih drugih.

3.1.3. Digitalna era

Razvojem računala audio inženjering doživio je kao i mnoga druga područja ljudske djelatnosti potpunu preobrazbu. Cjelokupni koncept snimanja zvuka promijenio se u nevjerojatnom periodu od svega petnaest do dvadesetak godina kada je sva prethodno godinama razvijana oprema zamijenjena računalnim programima, konkretizirano virtualnim efektima. Veliki studijski kompleksi zamijenjeni su gotovo višestruko manjim prostorima – snimanje zvuka postalo je dostupno gotovo svakome. Audio inženjering današnjice, samim time snimanje zvuka kao takvo još je u digitalnoj eri.

3.2. Značaj audio inženjeringa

Možda sami nismo niti svjesni, no živimo u vremenu kada količina multimedije u svakodnevnom životu nikada nije bila veća naprosto smo konstantno u doticaju s produktima audio inženjeringa. Jednostavno govoreći svaki multimedijski sadržaj čiji je sastavni dio i zvuk na jedan ili drugi način produkt je audio inženjeringa. Cjelokupni radijski i televizijski program, svaki sadržaj na platformama za strujanje video i audio materijala, svaka pjesma naših omiljenih umjetnika itd. - proizvodi su audio inženjeringa. Snimljeni zvukovi moraju biti uređeni prije negoli dospiju do ušiju slušatelja pa tu govorimo o aktu audio inženjeringa, no realnost je da je sama mogućnost snimanja audio inženjering kao takav – dakako sve zbog povijesti i značajnih ljudi dotaknutih nešto ranije.

Značaj audio inženjeringa do sada nikada nije bio veći, no šira javnost mu možda nikada nije pružala manje pozornosti. Modificiranjem zvuka te upravljanjem frekvencijama mogu se postići razni efekti od jednostavnog podizanja slušljivosti materijala do promjene raspoloženja slušatelja što čini audio inženjering vrlo moćnim alatom. Rapidnim porastom količine zvučnih sadržaja te zahtjeva za zvučnim priredbama bilo koje vrste, raste i razvoj alata audio inženjeringa što u konačnici rezultira i porastom tržišta zvuka kao takvog. Glazbena industrija jedna je od najvećih na svijetu, što automatizmom pretvara i audio inženjering u istu takvu – time rečeno ne smije se zanemariti značaj istoga, naprotiv treba mu se pružiti znatno veća pažnja i pozornost. Audio inženjering nudi ekonomske potencijale, no i socijalno kulturološke potencijale koji se ne smiju ostati u sjeni.

3.3. Snimanje zvuka

Snimanje zvuka oduvijek se temeljilo na nastojanju da se uhvate zvučni valovi (frekvencije) koje proizvodi pojedini objekt u prostoru – bilo instrument, osoba ili što drugo. Izumom mikrofona zvučni val bi se nastojao uhvatiti mikrofonom koji je sačinjen od žičane zavojnice koja je pričvršćena za membranu – relativno tanku opnu i magneta kojim je sama zavojnica okružena. Zvučni valovi uzrokuju podražaje membrane što kao posljedicu ima prijenos energije na žičanu zavojnicu. Gibanje žičane zavojnice unutar magnetskog polja stvara električni signal koji se pomoću kabla prenosi kao analogni signal. Taj analogni signal dospio bi na ulazne utore analogne studijske opreme gdje bi bio modificiran, dakako na različite načine – zavisno o kojem se komadu studijske opreme se radi. Nakon modificiranja analognog signala, novonastali modificirani analogni signal prenio bi se na medij za čuvanje snimljenog zvuka – u magnetskoj eri primjerice na magnetsku traku.

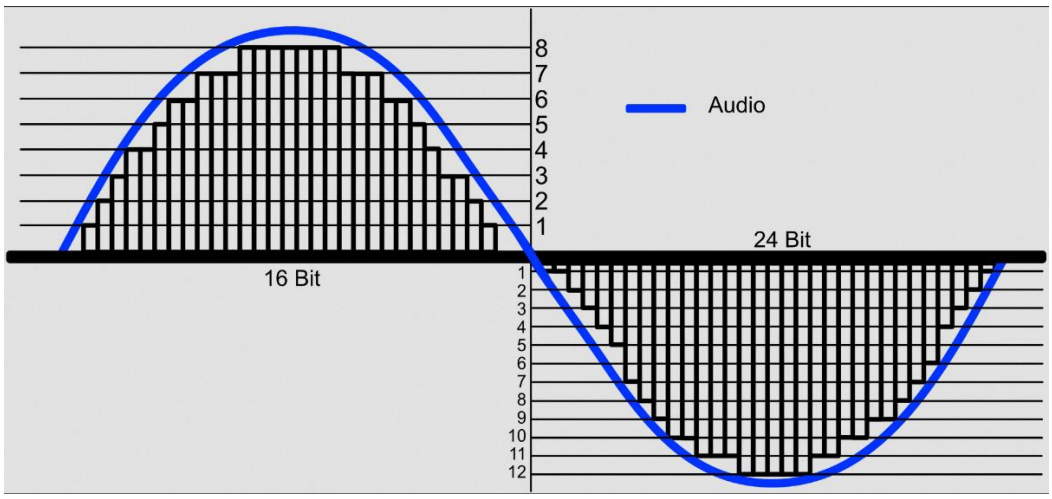
S obzirom na to da snimanje zvuka kao takvo nije predmet ovog završnog rada u nastavku neću previše vremena posvećivati analognom snimanju, no radi dobivanja konteksta o virtualnim efektima vrijedno je spomenuti digitalno snimanje.

Analogni signal nastao snimanjem mikrofonom i kod digitalnog snimanja prenosio bi se primjerice kablom, no analogni signal kao takav računalo ne može „razumjeti“ pa se isti mora pretvoriti u niz jedinica i nula – u nešto računalu „razumljivo“. Analogni signal, u suštini voltaža, kod digitalnog snimanja pomoću kabla dospijeva do zvučne kartice koja dospjelu voltažu konvertira u binarni kod. Prilikom konverzije koju provodi zvučna kartica u obzir se za kvalitetu snimke uzimaju dvije stvari, a to su frekvencija uzrokovanja (engl. *sample frequency*) te dubina bita (engl. *bit depth*). Nakon što se analogni konvertira u digitalni signal njime se može baratati u adekvatnim programskim alatima. Primarno, potreban je programski alat za samo snimanje, uređivanje te reproduciranje zvuka – poznatiji kao digitalna audio radna stanica. Digitalna audio radna stanica nadomješta neizostavne komponente glazbenog studija iz magnetske ere – konkretno magnetsku traku za snimanje i primjerice osnovne komponente konzole za uređivanje zvuka. Unutar digitalne audio radne stanice nadalje se pomoću virtualnih efekata nadomješta popratna oprema iz magnetske ere – oprema poput ekvilajzera, kompresora, predpojačala, distorzija i slično. Upravo se kombinacijom digitalne audio radne stanice i virtualnih efekata postiže ekvivalent nekadašnjem glazbenom studiju, no sve unutar računala spram nekad sobe ili čak cijelih korporacijskih kompleksa.

U nastavku slijedi nekoliko riječi o dubini bita te frekvenciji uzrokovanja, jer su neizostavna komponenta prilikom digitalnog snimanja i potrebno je odabrati iste prilikom stvaranja projekta unutar digitalne audio radne stanice, odnosno unutar programskih alata.

3.3.1. Dubina bita (engl. bit depth)

Dubina bita zapravo u suštini predstavlja koliko razina glasnoće, gledano sa strane dinamike zvuka – same amplitude, se može snimiti. Najčešće dubine snimanja su 16 ili 24 bita – postoje i „veće“ dubine, dakako sami iznosi rastu eksponencijalno. Kako bi se jednostavnije predočila dubina bita prilažem ilustraciju koju su izradili autori internetske stranice *„Mastering the Mix“*, 29. ožujka 2016. godine.

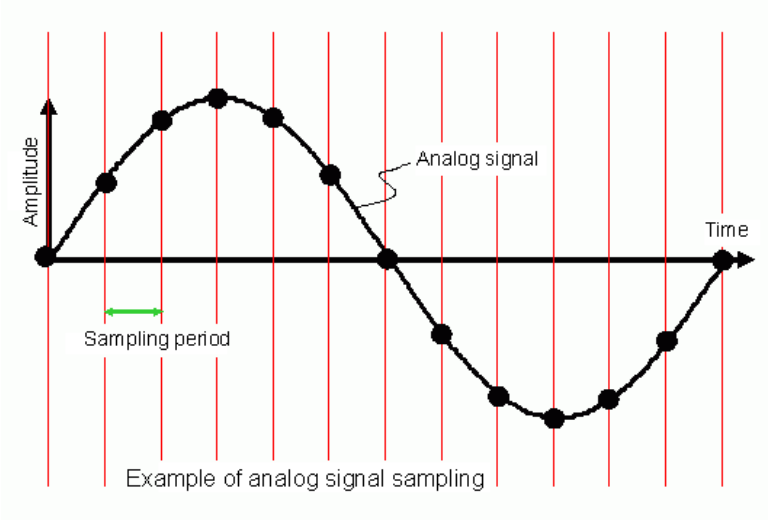


Slika 2: Dubina bita (Mastering the Mix, 2016.)

Udio plave sinusoide predstavlja analogni zvuk. Crni pravokutnici koji opisuju udio sinusoide s njezine unutarnje strane predstavljaju frekvenciju uzrokovanja, no ista trenutno nije predmet promatranja. Naime s lijeve strane ordinate dade se uvidjeti 16 bitna dubina bita što bi značilo da postoji 65.536 (2^{16}) različitih glasnoća koje mogu biti zapisane – konkretno 32.768 razina glasnoće sa svake strane apscise. Zapravo razmislimo li malo to je zaista puno različitih razina glasnoće, uistinu mnogo dinamike koja se može očuvati – većina glazbe danas u konačnom formatu do slušatelja dolazi upravo u 16 bitnoj dubini, primjerice audio diskovi, mp3 datoteke i sl.. No, postoji omanji problem – pojedini instrumenti ipak stvaraju toliko drastične oscilacije prilikom performansa da i 65.536 različitih glasnoća ne mogu uhvatiti sve te profinjene razlike – kao primjer može se navesti bubanj. Iz spomenutog razloga uglavnom se za snimanje pojedinih instrumenata koristi 24 bitna dubina bita što rezultira daleko većim brojem različitih glasnoća koje mogu biti zapisane – konkretno njih 16.777.216. Konverzije iz 24 bitne dubine bita u 16 bitnu dubinu bita i obratno dakako su moguće i one se itekako rade. Primjer gdje finalni proizvod, odnosno finalni zvuk ostaje pri 24 bitnoj dubini bita jest filmska industrija – DVD i sl..

3.3.2. Frekvencija uzrokovanja (engl. sample rate)

Sljedeći izrazito bitan aspekt prilikom digitalnog snimanja jest frekvencija uzrokovanja. Frekvencija uzrokovanja možda je i jednostavnija za shvatiti, a zapravo predstavlja broj uzoraka koji se uzimaju iz analognog signala tijekom jedne sekunde. S obzirom na to da nije moguće analogni signal tumačiti u digitalnom svijetu – njegova reprezentativna sinusoida se opisuje točkama (uzorcima) čijim spajanjem može nastati aproksimativni prikaz analognog signala – digitalni signal.



Slika 3: Frekvencija uzrokovanja (Renesas Electronics Corporation, 2019)

Na apscisi se nalazi vrijeme dok je na ordinati prikazana amplituda pristiglog analognog signala. Svaka udaljenost između paralelnih crvenih pravaca predstavlja frekvenciju uzrokovanja. Svaka crna točka na dijelu sinusoide, na analognom signalu predstavlja jedan od uzoraka. Spajanjem tih točaka dobivamo digitalnu reprezentaciju analognog zvuka – vrlo grubo govoreći. Sasvim je logično, nadalje, da što više tih točaka odnosno uzoraka ima to će kvaliteta zvuka biti bolja – jer naprosto onda digitalni signal bolje precrtava inicijalni analogni signal. Dakako više uzoraka označava i više podataka što rezultira i većim zauzećem memorije – što treba imati na umu jer je ista ograničena. Standardne frekvencije uzrokovanja u audio inženjeringu danas su 44.1 kHz te 48 kHz, no više nije neobično ni susresti 96 kHz kao odabranu frekvenciju uzrokovanja.

4. Virtualni efekt predpojačala

Kako sam u samom uvodu natuknuo konkretno ću se baviti izradom virtualnog efekta predpojačala pa bih se prije same izrade htio u nekoliko riječi osvrnuti na predpojačala, distorziju koju pružaju te virtualne efekte kao takve.

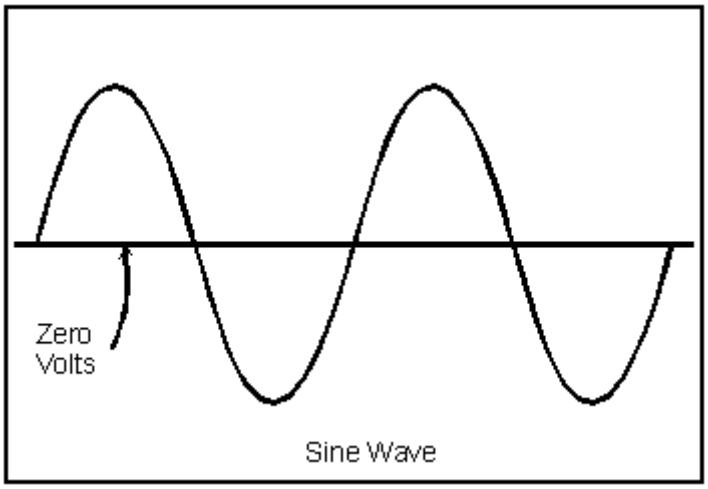
Gledajući činjenično tanka je linija između predpojačala i obične distorzije u digitalnom gitarističkom svijetu, virtualni efekt koji ću izraditi može se shvatiti i kao efekt distorzije – no prilikom izrade imao sam na umu predpojačala te sam samo oblikovanje zvuka (distorziju) više nastojao približiti zvuku koji bi nudila predpojačala negoli primjerice gitarski efekt distorzije.

4.1. Predpojačalo

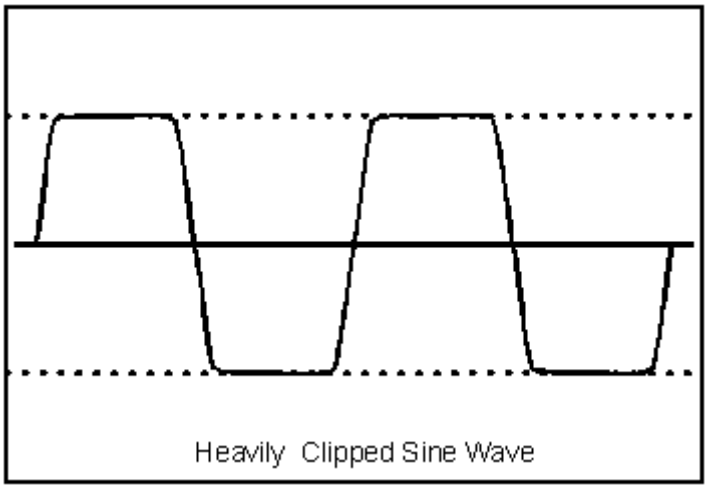
Predpojačala kao takva su električna pojačala čija je zadaća pretvarati slabi električni signal u signal koji će biti dovoljno jak i postojan za daljnju uporabu. Bez predpojačala ulazni signali bili bi vrlo tihi i u mnogo slučajeva vrlo nejasni i nedefinirani. U slučaju električne gitare predpojačala su neizostavan dio gitarskih pojačala. Ona u gitarističkom svijetu oblikuju zvuk i zapravo su zaslužna za većinski dio zvuka električne gitare. Signal pripremljen od strane predpojačala unutar gitarskih pojačala se šalje prema glavnom pojačalu (engl. *power amp*) koje znatno pojačani signal konačno emitira putem zvučnika, dakako titranjem membrane. Naprosto u digitalnom svijetu nije potrebno pojačavati signal u svrhu stvaranje same snage zvuka koja će pomicati membranu pa onda samo glavno pojačalo nije nužno niti potrebno simulirati.

Predpojačala uglavnom dolaze s nekoliko glavnih modifikatora zvuka od kojih su temeljni modifikator glasnoće (engl. *volume*) te modifikator zasićenosti zvuka (engl. *gain*). Modifikator glasnoće kao takav jest modifikator koji povećava ili smanjuje elongaciju amplitude ulaznog signala do željene odnosno moguće granice – sve zavisno o predpojačalu te korisniku istog. Zatim predpojačala gotovo uvijek nude modifikator zasićenosti ulaznog signala – modifikator koji zasićuje amplitudu ulaznog signala do određene razine elongacije nakon koje jednostavno odstrani višak signala. Upravo spomenuto odstranjivanje viška signala, odnosno prezasićenost se u konačnici onda manifestira kao distorzija – no o tome nešto više kasnije. Naposljetku predpojačala vrlo često još imaju i ekvilajzere (engl. *equalizer*) čija je zadaća oblikovati te izjednačiti frekvencijski spektar ulaznog signala. Ekvilajzer kao takav nije nužna sastavna komponenta predpojačala jer se isti uglavnom izrađuje kao zasebna komponenta unutar digitalnog, no i analognog svijeta.

Modifikacije zvuka koje predpojačalo pruža mogu se zorno predočiti grafom. Prilažem dva grafa preuzeta s web stranice *Geo-Fex Cybernetic Music* u trenutku izrade ovog rada. Prvi graf predočuje ulazni signal dok drugi graf predočuje apliciranje modifikatora zasićenosti zvuka na početni ulazni signal.



Slika 4: Ulazni signal (Geo-Fex Cybernetic Music, 2019)



Slika 5: Ulazni signal, modifikator zasićenosti (Geo-Fex Cybernetic Music, 2019)

Još se štošta može reći o predpojačalima, no ovo će biti dovoljno kako bi se stekao kontekst u koji ona pripadaju. Za samu implementaciju virtualnog efekta bit će značajnija distorzija i oblici u kojima dolazi – no prije distorzije par riječi o virtualnom efektu, što oni zapravo jesu?

4.2. Virtualni efekt

Iako sam se već dotakao virtualnih efekata prilikom komparacije digitalne te analogne audio produkcijske opreme držim da se još ponešto mora reći o istima jer virtualni efekt kao takav čini srž ovog rada. Virtualni efekti kako se već dade natuknuti iz prethodnih stranica zapravo su postali dio audio inženjeringa prilikom uvoda računala u domenu. Virtualni efekti su ništa drugo negoli emulacije analognih efekata – odnosno komada glazbene opreme. Dakle kada u audio produkciji kažemo efekt zapravo mislimo na komad glazbene opreme koji pruža neki oblik modifikacije zvuka, no možemo se referirati i na samo oblikovanje zvuka kao takvog uz pomoć tih istih komada glazbene opreme, sve zavisno o kontekstu u kojem se termin upotrebljava. Efekti se dijele u nekoliko skupina, a to su: modulacijski efekti, vremenski orijentirani efekti, efekti spektra, filteri te naposljetku dinamički efekti u koje spadaju distorzije pa samim time čine okosnicu predpojačala koja su od interesa za ovaj rad [3]. Svaki od tih efekata do nedavno bio je produkt određenog komada glazbene ili studijske opreme – ovisno želi li se specificirati. Danas se tih istih efekata može dočepati u obliku računalnih programa ili proširenja računalnih programa što je zapravo češći slučaj. Digitalne audio radne stanice su postale jezgra modernih audio studija, a virtualni efekti su računalni programi koji proširuju te digitalne audio radne stanice. Ukoliko se unutar digitalne audio radne stanice snimi pet odvojenih zvukova svaki unutar svojih kanala onda se svaki od tih kanala može modificirati virtualnim efektima, što onda zapravo modificira zvukove koji se puštaju kroz te kanale.

Slika 6: Cubase 5, Digitalna audio radna stanica

Na *Slika 6: Cubase 5, Digitalna audio radna stanica* može se vidjeti primjer digitalne audio radne stanice, konkretno programski alat Cubase 5 – programsko rješenje njemačke firme Steinberg. Ukoliko se pomnije promotri *Slika 6: Cubase 5, Digitalna audio radna stanica* može se uočiti digitalna konzola za snimanje s dva kanala. Prvi vidljivi kanal na sebi ima tri virtualna efekta od kojih je jedan otvoren – otvorena je druga inačica (Kodiak) virtualnog efekta kojeg sam razvio u svrhu ovog rada. Na drugom vidljivom kanalu također se nalazi jedan virtualni efekt. Otvoreni projekt snima se na frekvenciji uzrokovanja od 44.1 kHz dok je dubina bita 24 bita – što je vidljivo u lijevom donjem kutu.

Ponavljam, jer isto držim izuzetno bitnim – svaki efekt ima određen broj ulaza te izlaza. Kada govorimo o ulazima i izlazima mislimo na ulaze ili izlaze za zvučne signale – u analognom svijetu ti ulazi i izlazi bili bi manifestirani kao utori za kablove, dok „gledano“ digitalno radi se o logičkim ulazima i izlazima koji rukuju digitalnim signalima zvuka na programskoj razini. Taj faktor zapravo određuje može li efekt biti stereo ili se radi isključivo o mono efektu.

4.3. Distorzija

Svaka od spomenutih grupacija efekata (modulacijski efekti, vremenski orijentirani efekti, efekti spektra, filteri te dinamički efekti) nalazi svoju primjenu i zadužena je za posebnu zadaću. Kako sam kazao dinamički efekti od interesa su za ovaj rad – oni modificiraju ponašanje amplitude, korigiraju te rukuju elongacijom iste, barataju zasićenošću signala te pripomažu kontroli amplitudnih oscilacija te njihovih diferencija [4]. Distorzija kao jedan od dinamičkih efekata prisutna je još od samih početaka električnih glazbenih izvedbi. Distorzija je, kako sam natuknuo prilikom opisivanja predpojačala, zapravo prezasićenost signala koja rezultira odstranjivanjem dijela amplitude sinusoide zvučnog vala. U samim početcima distorzija je bila neželjena nuspojava, no dolaskom *rock n' roll* glazbe distorzija je postala u potpunosti svjetski trend te i danas drži svoju popularnost. Distorzija se manifestira u nekoliko oblika – svaki od oblika nosi svoje karakteristike te s istima nalazi i svoje mjesto u glazbi. Okvirno oblici distorzija se mogu podijeliti u tri skupine: *overdrive, distortion* i *fuzz* – termini su naprosto kao takvi ušli u hrvatski jezik te nemaju svoj uvaženi službeni prijevod. Kako ne bih previše izašao iz opsega rada neću zalaziti u detalje svake od spomenutih skupina, no htio bih se osvrnuti podrobnije na distorzije skupine *distortion* iz jednostavnog razloga jer se upravo te distorzije zapravo najčešće mogu pronaći u predpojačalima gitarskih pojačala. Razlika u zvuku svake od spomenutih skupina distorzija zapravo proizlazi iz različitih nelinearnih funkcija koje barataju zvukom.

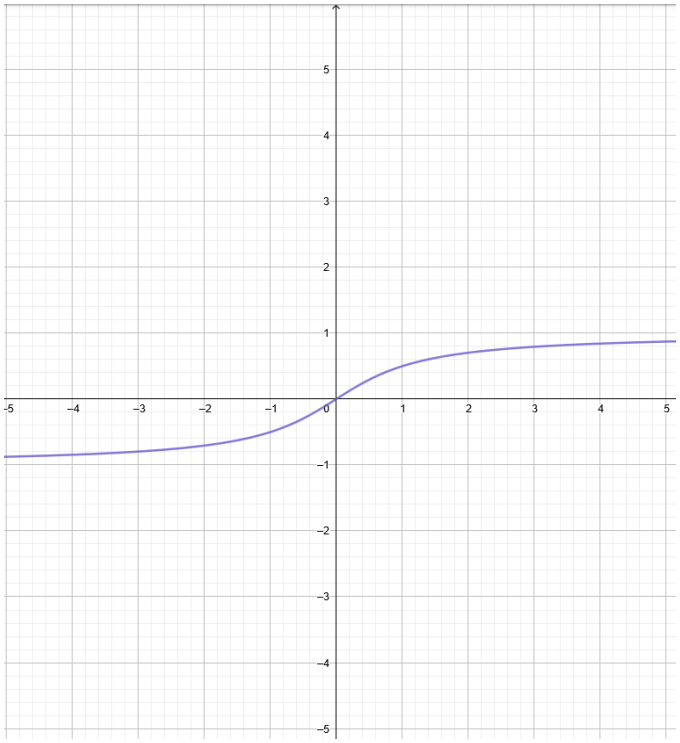
4.3.1. Nelinearne funkcije distorzije

Svaka distorzija, odnosno distorzije kao takve temelje se na nelinearnim matematičkim funkcijama koja zapravo u svojoj manifestaciji dovode do prezasićenosti signala. Primjer takve funkcije, a ujedno i nelinearna funkcija koju sam koristio prilikom izrade virtualnog efekta je:

$$f(x,\alpha) = \frac{2}{\pi} \cdot tan^{-1}(x) \cdot \alpha$$

$$\alpha \in [0,1]$$

Zapravo je riječ o nelinearnoj funkciji koja je vrlo učestala kod virtualnih efekata distorzije. Posebice onih iz skupine *distortion*. Argument x iz date funkcije predstavlja zapravo ulazni signal, dok je argument α zapravo glasnoća samog izlaznog signala. Prilažem grafičku reprezentaciju date funkcije – graf je izrađen u alatu GeoGebra.



Slika 7: Karakteristična krivulja

Grafom prikazana krivulja – *Slika 7: Nelinearna funkcija*, odnosno reprezentacija spomenute funkcije $f(x,\alpha)$ unutar audio inženjeringa se naziva karakteristična krivulja distorzije [4] ,a zapravo opisuje ponašanje ulaznog signala te njegovu konverziju u izlazni signal. Iskusniji

matematičar već bi pogledom na funkciju $f(x, \alpha)$ mogao zaključiti kakve karakteristike bi zvuk oblikovan tom funkcijom mogao nositi, no samo jednim pogledom na graf iste može pružiti zorniju predodžbu. Distorzija koja nastaje funkcijom $f(x, \alpha)$ karakterno bi spadala u *soft-clipping* distorzije, što zapravo označava da ne postoje oštrija rezanja amplitude ulaznog signala, već se sam signal „ravnomjerno“ prilagođuje zadanom okviru elongacije amplitude. Takva distorzija u konačnici rezultira prirodnijim zvukom koji nije nužno previše agresivan – takva karakteristika odviše odgovara tonu gitarskog predpojačala što je dakako i razlog zašto sam se osobno odlučio baš za tu funkciju odnosno za tu karakterističnu krivulju. Bilo kakva distorzija *hard-clipping* karakteristike prouzročila bi preagresivan zvuk koji se ne bi mogao koristiti u gitarskim predpojačalima. Primjer *hard-clipping* distorzije može se vidjeti na grafu: Slika 5: Ulazni signal, modifikator zasićenosti.

Vrlo pozitivna stvar jest ta što se funkcije koje predstavljaju karakterističnu krivulju distorzije mogu doslovno prepisati u kod kao takve te na taj način implementirati. U samoj srži to označava da se programsko rješenje za virtualni efekt distorzije relativno jednostavno dade implementirati imajući na umu adekvatan programski okvir te njegovu popratnu dokumentaciju.

U nadolazećim stranicama podarit ću pozornosti programskom okviru za C++ naziva JUCE. Radi se o izuzetno popularnom odabiru za razvoj virtualnih efekata, te izrazito popularnom programskom okviru unutar audio inženjeringa kao takvog – prosto iz razloga što JUCE kao okvir nudi još izrazito mnogo mogućnosti, no o tome više u nastavku.

5. Programski okvir JUCE

Programski okvir JUCE dostupan je za korištenje od 2004. godine kada je objavljena njegova prva inačica. Sam programski okvir razvio je Jules Storer, C++ programer čiji je kod uvelike utjecao na razvoj digitalnog audio inženjeringa. JUCE je posebnu popularnost stekao upravo zbog činjenice da se radi o okviru čiji se kod može izvršavati neovisno o operacijskom sustavu koji ga izvršava (engl. *cross-platform*) – štoviše kada se govori o operacijskim sustavima programski okvir JUCE podržava čak i razvoj mobilnih aplikacija. Ljudstvo koje razvija JUCE sam okvir na svojim mrežnim stranicama proziva vođom *cross-platform* audio aplikacija, a tomu u prilog ide i činjenica da je kod napisan uz pomoć JUCE programskog okvir podržan od strane sljedećih operacijskih sustava:

- Linux, kernel 2.6 i noviji
- Windows XP, Vista, 7, 8, 10
- Mac OS X, verzija 10.5 i novije
- Android, NDK-v5 i noviji
- iOS verzija 3 i noviji

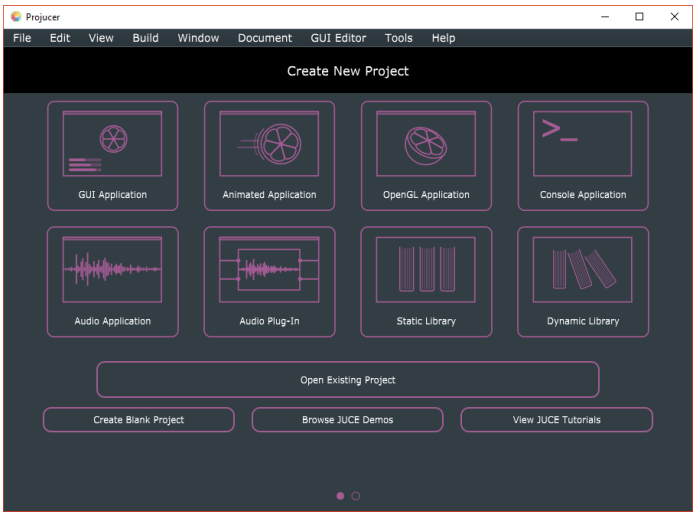
Također, JUCE službeno podržava sljedeće kompajlere - iako zajednica audio programera tvrdi da još nekoliko kompajlera stabilno kompajlira JUCE kod:

- GCC, verzija 4.0 i novije
- Microsoft Visual Studio – Visual C++ 2013 i noviji
- LLVM – LLVM Clang
- MinGW

5.1. Mogućnosti

Kao programski okvir, JUCE nudi izuzetan broj klasa koje pružaju razne metode pomoću kojih se mogu izrađivati elementi grafičkog sučelja, parsirati grafički te zvučni elementi. JUCE također može s lakoćom raditi i s XML te JSON datotekama, nudi metode za kriptografiju, mrežne aplikacije te mnogo drugih stvari. Radi se o vrlo svestranom okviru te je kao takav zaokupio pozornost programera diljem svijeta. Naposljetku i najvažnija činjenica za ovaj rad je da JUCE nudi i klase za izrađivanje audio virtualnih efekata te podržava velik broj formata virtualnih efekata – konkretno: VST, VST3, AU (Audio Units), RTAS i AAX.

Još valja spomenuti da programski okvir JUCE nudi i svoje razvojno okruženje Projucer koje rukuje JUCE projektima. Nakon što korisnik, odnosno programer specificira pojedinosti svog projekta, Projucer stvara kolekciju datoteka ovisno o pojedinostima projekta.

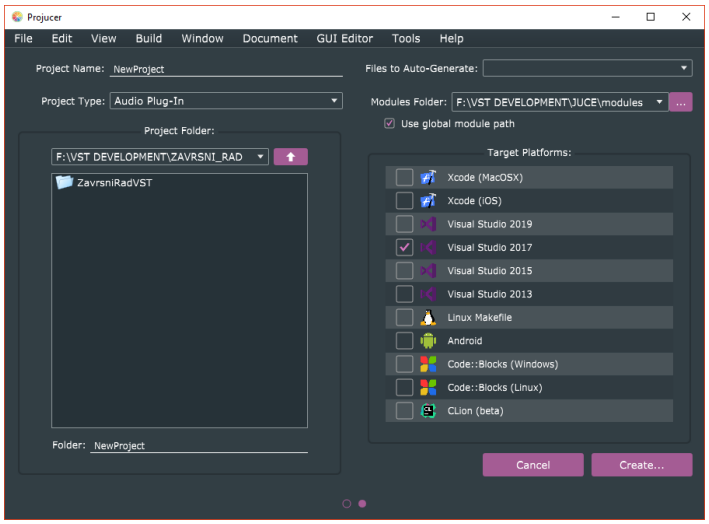


Slika 8: Projucer, stvaranje projekta

Kao što se na *Slika 8: Projucer, stvaranje projekta* može vidjeti – Projucer nudi zaista veliku paletu mogućnosti: stvaranje GUI aplikacija te mnogo čega drugog pa čak i OpenGL aplikacija.

5.2. Inicijalno korištenje

Kako bi se programski okvir JUCE zapravo uopće mogao koristiti prvotno je potrebno preuzeti JUCE sa službene web stranice. Preuzeti okvir potrebno je prvo kompajlirati primjerice u programskom alatu Visual Studio 2017 nakon čega se može koristiti razvojno okruženje Projucer koje će biti potrebno za stvaranje projekata. Kada se otvori Projucer korisnik se susreće s prozorom prikazanim na *Slika 8: Projucer, stvaranje projekta* gdje zapravo odabire tip projekta koji želi stvoriti. Nakon odabira korisnika Projucer navigira kroz stvaranje projekta što naposljetku dovodi do pisanja koda.



Slika 9: Projucer, pojedinosti projekta

6. Izrada virtualnog efekta

Kako bismo započeli izradu virtualnog efekta potrebno je u izborniku razvojnog okruženja Projucer odabrati stvaranje *Audio Plugin* projekta – *Slika 8: Projucer, stvaranje projekta.* Nakon ispunjavanja pojedinosti projekta poput odabira željenog kompajlera, arhitekture samog virtualnog efekta i sl. Projucer stvara kod programskog okvira JUCE. Konkretno u slučaju virtualnih efekata inicijalno se stvaraju četiri datoteke:

- PluginProcessor.cpp
- PluginProcessor.h
- PluginEditor.cpp
- PluginEditor.h

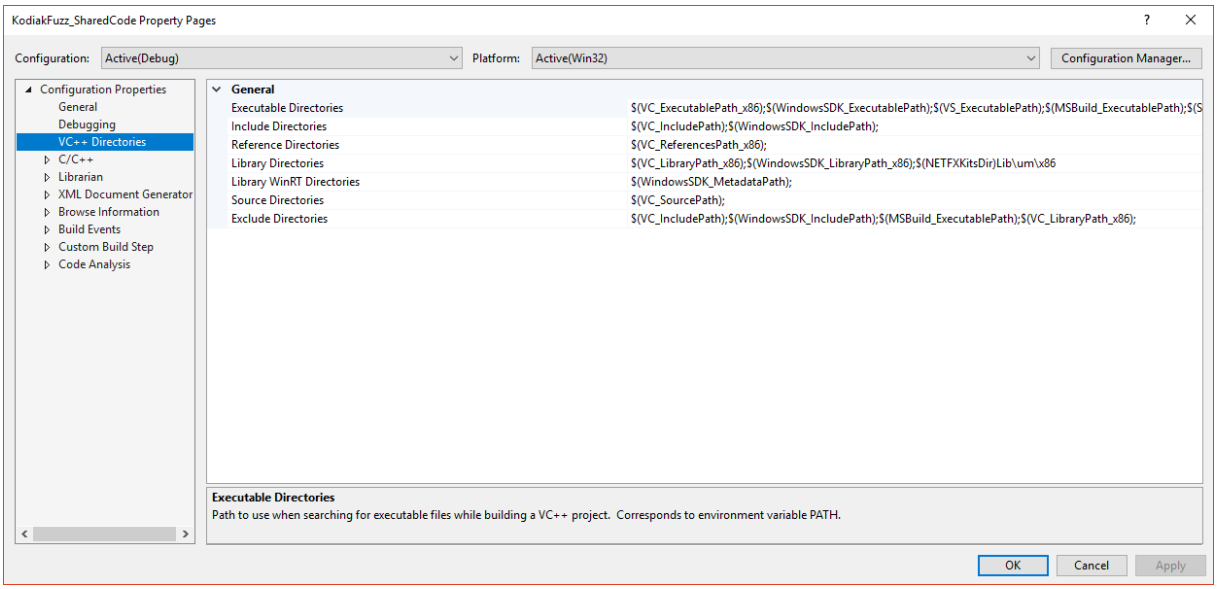
Datoteke *PluginEditor.cpp* i *PluginEditor.h* su datoteke unutar kojih se uglavnom barata elementima grafičkog sučelja samog virtualnog efekta dok se u datotekama *PluginProcessor.cpp* te *PluginProcessor.h* implementira sama jezgra samog virtualnog efekta – modifikacija te alteracija ulaznog zvuka. Prije negoli se otpočne pisanje koda, potrebno je još unutar razvojnog okruženja Projucer još jednom provjeriti postavke projekta te eventualno podesiti pojedinosti poput broja ulaznih te izlaznih kanala – ovisno jesu li ulazi odnosno izlazi mono ili stereo. Nakon što su postavke prilagođene projektu te situaciji može se otpočeti s programiranjem – u mojem slučaju unutar programskog alata Visual Studio 2017.

6.1. Steinberg SDK

S obzirom na to da sam virtualni efekt odlučio implementirati koristeći operacijski sustav Windows 10 – naprosto jer većina audio studija koristi upravo taj operacijski sustav bilo je jednostavno i odlučiti koji format virtualnog efekta izabrati. Odabir je pao na VST (*Virtual Studio Technology*) format – koji je najkorišteniji format virtualnih efekata u odabranom operacijskom sustavu (*Windows 10*). Htio bih samo dodati da sam inicijalno htio virtualni efekt izrađivati koristeći se Linux Manjaro operacijskim sustavom te AU formatom virtualnog efekta, sam razvoj bio bi gotovo identičan – no tržišno gledajući to još uvijek nema smisla, jer je audio inženjering na Linux distribucijama još nije zaprimio puni zamah – iako tu, osobno smatram, leži izrazit potencijal. macOS s druge strane usko je vezan uz sklopovsku arhitekturu kojoj pristup nemam.

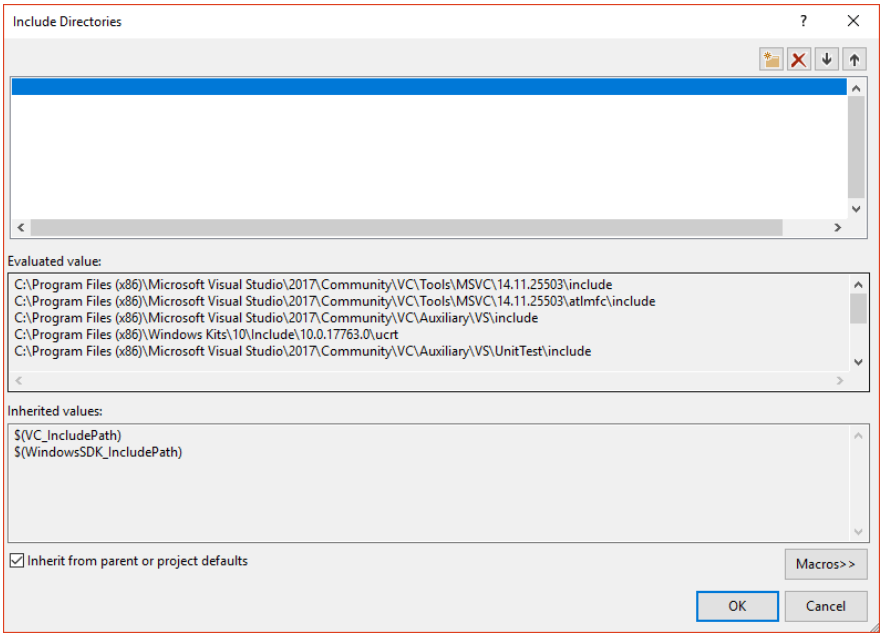
VST kao format razvijen je od strane Steinberg Media Technologies godine 1996., a iste te godine Steinberg je također objavio i SDK (Software Development Kit) za isti format što je omogućilo entuzijastima razvoj virtualnih efekata. Ne bih zalazio u povijest niti pojedinosti VST

formata virtualnih efekata, no za samu implementaciju virtualnog efekta u VST formatu bitno je znati da projekt mora imati pristup Steinberg priboru za razvijanje softvera (SDK). Unutar rješenja Visual Studio 2017 potrebno je unutar obilježja rješenja odabrati direktorije koji će biti uključeni u sam projekt – konkretno potrebno je odabrati Steinberg SDK direktorije.



Slika 10: Visual Studio 2017, Property Pages

Pribor za razvijanje softvera unutar Property Pages prozora dodajemo klikom na Include Directories, prilikom čega se otvara prozor za sam odabir direktorija. Dodavanjem direktorija u Property Pages zapravo smo dodali pribor za razvijanje softvera u naš Visual Studio 2017 projekt.



Slika 11: Visual Studio 2017, Include Directories

Zbog činjenice da JUCE kao programski okvir samostalno generira minimalno dva projekta unutar Visual Studio rješenja potrebno je uključiti Steinberg SDK za svaki pojedini projekt kako bi virtualni efekt u konačnici bilo moguće kompajlirati. Nakon uključivanja svih potrebnih direktorija odnosno pribora za razvijanje softvera moguće je započeti pisati kod, odnosno implementirati sam virtualni efekt.

6.2. Izrada grafičkog sučelja virtualnog efekta

Prilikom izrade grafičkog sučelja kako sam već natuknuo uglavnom se piše kod unutar datoteka *PluginEditor.cpp* te *PluginEditor.h* jer su one primarno zadužene za korisničko sučelje virtualnog efekta.

Iz razloga što virtualni efekt predpojačala mora primarno imati dvije kontrole (glasnoća te količina zasićenosti signala) bilo je u interesu stvoriti dva elementa grafičkog sučelja pomoću kojih bi korisnik mogao alterirati ulazni signal virtualnog efekta. Kao i gotovo sve ostalo unutar koda vezanog uz audio inženjering implementacija ovih dvaju grafičkih elemenata izvedena je pomoću pokazivača koji su zapravo prekriveni u jednoj od klasa koje nudi JUCE programski okvir – klasa o kojoj govorim naziva se *ScopedPointer*. Velika uporaba pokazivača i referenci unutar programiranja vezanog za audio inženjering proizlazi iz činjenice da su performanse izrazito bitne kod programskih rješenja spomenute domene.

Implementacija grafičkih elemenata koji će zapravo biti dvije kontrole na grafičkom sučelju započela je u *PluginEditor.h* datoteci gdje su deklarirane dvije varijable stereotipizirane klase *ScopedPointer<Slider>* nakon čega su također deklarirane dvije varijable stereotipizirane klase *ScopedPointer<AudioProcessorValueTreeState::SliderAttachment>*.

```
ScopedPointer<Slider> gainControl;
ScopedPointer<Slider> volumeControl;


ScopedPointer<AudioProcessorValueTreeState::SliderAttachment>
gainAttachment;
ScopedPointer<AudioProcessorValueTreeState::SliderAttachment>
volumeAttachment;
```

Nakon deklaracije varijabli potrebno je inicijalizirati objekte spomenutih stereotipiziranih klasa unutar datoteke *PluginEditor.cpp* – konkretno unutar konstruktora.

```
addAndMakeVisible(gainControl = new Slider("Gain"));
addAndMakeVisible(volumeControl = new Slider("Volume"));
```

Metoda *addAndMakeVisible()* je metoda tipa void te kao argument prima pokazivač na *juce::Component* objekt, opcionalno spomenuta metoda još prima i cjelobrojni argument koji zapravo predstavlja u kojem vizualnom planu će se komponenta postaviti, odnosno u kojem sloju grafičkog sučelja će se komponenta pronaći – vrlo slično kao z-index svojstvo unutar CSS-a kod web programiranja. Pozivom metode *addAndMakeVisible()* – objekt na koji pokazuje proslijeđeni pokazivač se dodaje te postaje vidljiv unutar grafičkog sučelja samog virtualnog efekta. S obzirom na to da instance klase *Slider* posjeduju nekoliko obilježja ista se mogu modificirati određenim metodama iz spomenute klase. Budući da rukujemo pokazivačima iste je prilikom poziva metoda potrebno dereferencirati. Metode koje sam pozivao za promjenu izgleda komponenti grafičkog sučelja, odnosno za modificiranje njihovih obilježja su: *setSliderStyle()* i *setTextBoxStyle()*.

```
(*gainControl).setSliderStyle(Slider::Rotary);
(*gainControl).setTextBoxStyle(Slider::NoTextBox, true, 1 , 1);

(*volumeControl).setSliderStyle(Slider::Rotary);
(*volumeControl).setTextBoxStyle(Slider::NoTextBox, true, 1 , 1);
```

Metoda *setSliderStyle()* omogućuje promjenu tipa izgleda same komponente. Kao argument metoda prima cjelobrojni broj – u ovome slučaju element enumeracije *Slider::Rotary.* „Rotary Sliders“ su komponente grafičkog sučelja koje nalikuju kontrolama sučelja analognih predpojačala – stoga je odabir pao upravo na taj tip. Nadalje metoda *setTextBoxStyle* prima nekoliko argumenata, no samo je jedan od interesa s obzirom na moj odabir. Nisam htio imati tekstualni okvir vezan uz svoje grafičke komponente za kontrolu samog virtualnog efekta pa sam iste maknuo prosljeđujući kao prvi argument element enumeracije *Slider::NoTextBox*. Kako bih još dodatno poradio na dizajnu samih komponenti za kontrolu virtualnog efekta odlučio sam im promijeniti i boju.

```
getLookAndFeel().setColour(Slider::thumbColourId, Colours::darkgray);
```

Sljedeći korak prilikom izrade grafičkog sučelja bio je postaviti visinu i širinu samog virtualnog efekta – konkretno na 800 piksela širine te 400 piksela visine.

```
setSize (800, 400);
```

Nakon što je postavljena dimenzija virtualnog efekta bilo je potrebno skalirati same grafičke komponente za kontrolu virtualnog efekta. Spomenuto nisam učinio unutar konstruktora već u metodi *resized()* zbog konvencije koja je zadana u dokumentaciji programskog okvira JUCE.

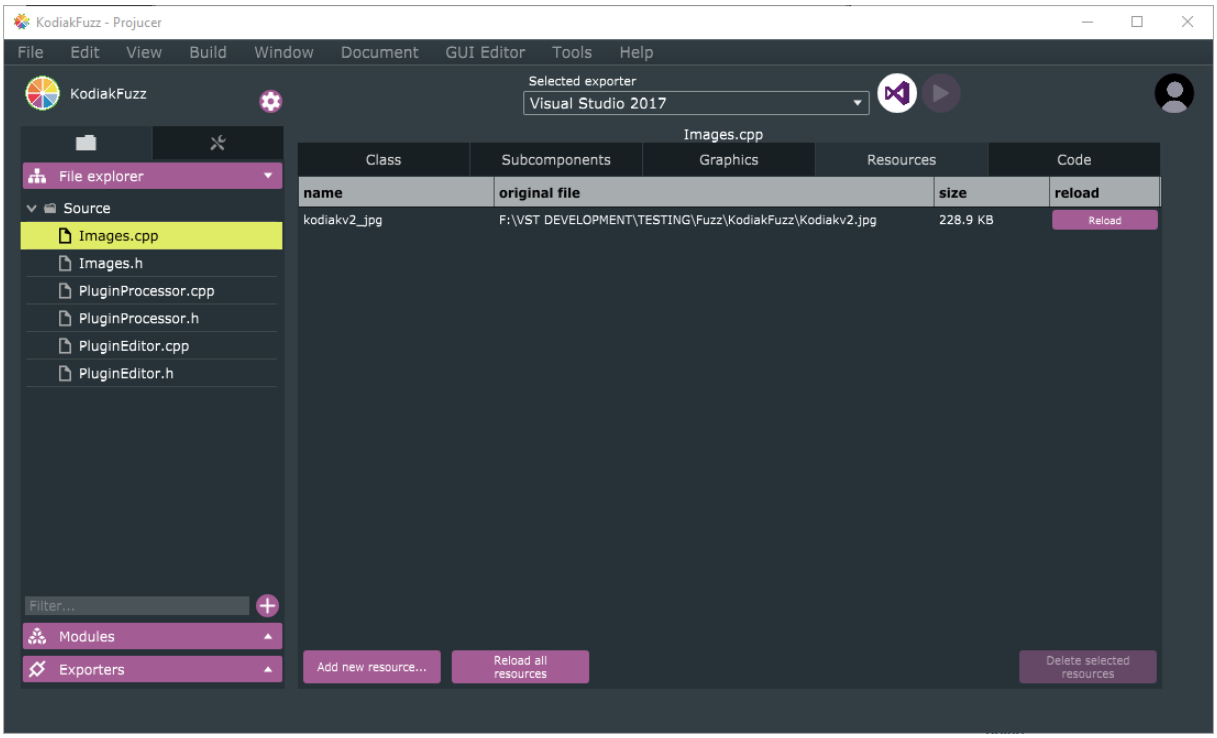
```
void KodiakFuzzAudioProcessorEditor::resized()
{
	// This is generally where you'll want to lay out the positions of any
	// subcomponents in your editor..

	// Controls, components:
	(*gainControl).setBounds(308, 325, 60, 60);
	(*volumeControl).setBounds(435, 325, 60, 60);

}
```

Konkretne brojeve za pozicioniranje komponenti dobio sam prostim isprobavanjem dok nisam bio zadovoljan pozicijom te dimenzijom istih. Metoda za promjenu dimenzije te pozicije pojedine komponente je kao što se gore može vidjeti metoda *setBounds()*.

U želji da dodatno doradim dizajn grafičkog sučelja virtualnog efekta odlučio sam istome postaviti pozadinsku sliku koju sam izradio unutar programskog alata GIMP. Glavnu ulogu u dodavanju pozadinskih slika ili bilo kakvih dodatnih komponenti grafičkog sučelja koji nisu standardni dio JUCE programskog okvira – odigrava razvojno okruženje Projucer. U izborniku datoteka razvojnog okruženja Projucer može se pronaći korijenski direktorij zadanog naziva *Source*. Desnim klikom na direktorij *Source* nudi se nekoliko opcija od kojih valja odabrati *Add New GUI Component*. Odabirom spomenute opcije Projucer stvara dvije nove datoteke koje se konvencionalno nazivaju *Images.cpp* i *Images.h*. Otvori li se *Images.cpp* unutar Projucer razvojnog okruženja nudi se nekoliko opcija, između ostalog i dodavanje novog resursa – konkretno slike. Nakon dodavanja resursa, Projucer projekt se mora spremiti.



Slika 12: Projucer, Dodavanje slike

Dodavanjem slike unutar razvojnog okruženja prvi je korak dodavanja pozadinske slike za virtualni efekt u izradi. Potrebno je dodati ponešto koda. Prvotno je unutar *PluginEditor.h* datoteke potrebno dodati uključivanje *Images.h* datoteke te dodati privatnu varijablu koja će sadržavati konkretnu pozadinsku sliku.

```
#include "Images.h"

...

Image backgroundImage;
```

Sljedeći korak je u konstruktoru kojeg možemo pronaći u datotetci *PluginEditor.cpp* valja iz memorije (Projucer rukuje memorijom vezanom za vanjske resurse) željenu sliku spremiti u privatnu varijablu koju smo deklarirali u *PluginEditor.h* datoteci. Opisano možemo učiniti koristeći metodu *getFromMemory* koja kao argument prima binarni resurs te njegovu veličinu – u konkretnom slučaju sliku zapisanu kao binarni kod te veličinu iste.

```
	backgroundImage = ImageCache::getFromMemory(Images::kodiakv2_jpg,
Images::kodiakv2_jpgSize);
```

Posljednji korak je prikazati samu pozadinsku sliku, a to se prema konvenciji čini unutar metode *paint koristeći metodu drawImage* nad objektom g koji pripada klasi *Graphics* koja predstavlja kontekst za rukovanje grafičkim komponentama virtualnog efekta.

```
void KodiakFuzzAudioProcessorEditor::paint (Graphics& g)
{
	g.drawImage(backgroundImage, 0, 0, 800, 400, 0, 0, 800, 400);
}
```

Kako bi se započelo s procesiranjem zvuka bilo je još potrebno napraviti konkretnu poveznicu između grafičkog sučelja virtualnog efekta te samog algoritma za procesiranje zvuka. Opisano se pomoću programskog okvira JUCE radi takozvanim *Attachmentima* ili ako bi se slobodno moglo prevesti poveznicama. Prilikom inicijalizacije prosljeđuje se stanje virtualnog efekta, identifikacijski string za datu komponentu te pokazivač na objekt odnosno komponentu u ovome slučaju *Slider*.

```
gainAttachment = new
AudioProcessorValueTreeState::SliderAttachment(p.getState(), "Gain",
*gainControl);

volumeAttachment = new
AudioProcessorValueTreeState::SliderAttachment(p.getState(), "Volume",
*volumeControl);
```

Može se primijetiti da se kao prvi argument prosljeđuje povratna vrijednost metode *getState()* koja je pozvana nad objektom *p*. Objekt *p* zapravo je argument same metode unutar koje je napisan gore priloženi kod, a on je zapravo referenca na sam procesor virtualnog efekta – objekt odnosno sama implementacija koja je zadužena za modificiranje zvuka unutar virtualnog efekta. Metodu *getState()* potrebno je samostalno implementirati, no implementacija iste se može pronaći u dokumentaciji programskog okvira JUCE jer se stanje procesora vrlo često dohvaća. Ono zapravo označuje stanje u kojem se u datom trenutku nalazi sam virtualni efekt – pritom govorimo o eventualnim spremljenim postavkama samog virtualnog efekta (engl. *presets*). *getState()* metoda se prema standardima implementira unutar *PluginProcessor.h* datoteke. Prvotno se unutar private bloka deklarira objekt stereotipizirane klase *ScopedPointer<AudioProcessorValueTreeState>* čiji se getter nakon toga deklarira unutar public bloka.

```
...
private:
ScopedPointer<AudioProcessorValueTreeState> state;
...

...
public:
// GETTER:
AudioProcessorValueTreeState& getState();
...
```

Sama implementacija gettera te sama inicijalizacija stanja odnosno objekta *state* nalazi se unutar *PluginProcessor.cpp* datoteke – konkretno unutar konstruktora.

```
{
      state = new AudioProcessorValueTreeState(*this, nullptr);
      (*state).createAndAddParameter("Gain", "Gain", "Gain",
NormalisableRange<float>(0.f, 200.f, 0.001), 0.0, nullptr, nullptr);
      (*state).createAndAddParameter("Volume", "Volume", "Volume",
NormalisableRange<float>(0.f, 0.2f, 0.001), 0.0, nullptr, nullptr);

      (*state).state = ValueTree("Gain");
      (*state).state = ValueTree("Volume");
}
```

Prilikom inicijalizacije objekta *state* u sam konstruktor klase *AudioProcessorValueTreeState* prosljeđuje se pokazivač na procesor te opcionalni menadžer za povratak prethodnih postavki stanja (zapravo omogućava implementaciju *undo* gumba). U svojoj implementaciji koristio sam pokazivač **this* s obzirom na to da se gore priloženi kod već piše unutar procesora te *nullptr* jer ne želim koristiti menadžer za povratak prethodnih postavki stanja. Nadalje potrebno je stvoriti same parametre koji su korišteni unutar virtualnog efekta – ovi parametri se kontroliraju pomicanjem komponenti grafičkog sučelja koje su prethodno izrađene. Parametri se stvaraju metodom *createAndAddParameter()* koja prima pozamašan broj argumenata, no zapravo se

radi o domenama koje parametri mogu poprimiti te nazivlju istih. Naposljetku potrebno je još implementirati getter koji je deklariran u public bloku *PluginProcessor.h* datoteke.

```
AudioProcessorValueTreeState&  KodiakFuzzAudioProcessor::getState()
{
      return *state;
}
```

Ukoliko se želi baratati stanjima procesora to se može učiniti unutar dolje navedenih metoda. Rad sa stanjima zapravo je rad s XML datotekama koje spremaju korisničke postavke virtualnog efekta.

- void KodiakFuzzAudioProcessor::getStateInformation (MemoryBlock& destData)
- void KodiakFuzzAudioProcessor::setStateInformation (const void* data, int sizeInBytes)

Sad kada je izrađeno grafičko sučelje te je implementirana poveznica između grafičkog sučelja te pozadine samog virtualnog efekta može se otpočeti s implementacijom algoritama za procesiranje, modifikaciju te alteriranje zvuka. Spomenuti algoritmi spadaju pod DSP (*Digital signal processing*) algoritme. Imajući u obzir audio inženjering, DSP algoritmi se bave provedbom seta kalkulacija koje zapravo rezultiraju manipulacijom digitalnog zvuka. Uz spomenuto DSP algoritmi imaju još pozamašnu upotrebu – primjerice:

- Enkripcija te dekripcija govora
- Prepoznavanje govora
- Poništavanje pozadinske buke

6.3. Procesiranje zvuka pomoću virtualnog efekta

Zbog činjenice da se cjelokupni DSP algoritam za procesiranje zvuka unutar virtualnog efekta odvija unutar jedne metode, odlučio sam tu istu cijelu metodu priložiti te ju potom podrobnije opisati.

```
void KodiakFuzzAudioProcessor::processBlock (AudioBuffer<float>& buffer,
MidiBuffer& midiMessages)
{
    auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();


    for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
        buffer.clear (i, 0, buffer.getNumSamples());

      float gainValue = *(*state).getRawParameterValue("Gain");
      float volumeValue = *(*state).getRawParameterValue("Volume");


      for (int channel = 0; channel < totalNumInputChannels; ++channel)
      {
        auto* channelData = buffer.getWritePointer (channel);

            for (int sample = 0; sample < buffer.getNumSamples(); sample++)
            {
                  *channelData *= gainValue;
                  *channelData =
                  ((2.f/float_Pi)*atan(*channelData))*volumeValue;

                  channelData++;
            }

      }
}
```

Metoda *processBlock* u potpunosti je zadužena za rukovanje sa zvukom. Kao argument prima dva buffera koja su zapravo nešto kao dva privremena spremnika. Prvi argument *AudioBuffer<float>& buffer* je zapravo ulazni audio signal, dok je drugi argument MIDI signal, no isti nisam koristio prilikom izrade virtualnog efekta predpojačala jer ne rukujem MIDI signalima već samo digitalnim zvukom. Sama metoda otpočinje dohvaćanjem broja ulaznih te izlaznih kanala samog virtualnog efekta.

```
      auto totalNumInputChannels  = getTotalNumInputChannels();
      auto totalNumOutputChannels = getTotalNumOutputChannels();
```

Potom se prazne bufferi kako bi se primarno oslobodila memorija.

```
      for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
        buffer.clear (i, 0, buffer.getNumSamples());
```

Sljedeći korak je dohvatiti vrijednosti parametara koje je korisnik podesio koristeći komponente grafičkog sučelja. Ako se podsjetimo to su bile dvije *Slider* komponente, jedna za glasnoću te jedna za količinu prezasićenosti signala. Domena parametara određena je prilikom stvaranja istih u *PluginProcessor.cpp* datotetci.

```
float driveValue = *(*state).getRawParameterValue("Gain");
float volumeValue = *(*state).getRawParameterValue("Volume");
```

Nakon dohvaćanja korisničkih parametara potrebno je za svaki ulazni kanal (konkretno sada je samo jedan) dohvatiti sadržaj buffera. Ono što se zapravo dohvaća je ulazni zvuk virtualnog efekta.

```
for (int channel = 0; channel < totalNumInputChannels; ++channel)
      {
        auto* channelData = buffer.getWritePointer (channel);
      ...
```

Zatim je potrebno za svaki uzorak (engl. *sample*) unutar buffera oblikovati sam zvučni signal. Broj uzoraka zavisit će o postavkama digitalne audio radne stanice – konkretno o frekvenciji uzrokovanja. Algoritam koji sam odabrao množi ulazni signal s parametrom količine prezasićenosti zvuka te potom na isti ulazni signal primjenjuje nelinearnu funkciju koja će zapravo stvoriti željeni efekt distorzije koji je potreban za svrhe izrade virtualnog efekta predpojačala.

```
for (int sample = 0; sample < buffer.getNumSamples(); sample++)
      {
            *channelData *= gainValue;
            *channelData =
            ((2.f/float_Pi)*atan(*channelData))*volumeValue;

            channelData++;
      }
```

Ovih par linija dovoljnih su za ostvarivanje efekta distorzije pomoću virtualnog efekta, no pisanje istih nije bilo nimalo jednostavno – barem ne meni, još relativno neiskusnom programeru. Čitanje dokumentacije bilo je na trenutke pomalo zbunjujuće te je sam pothvat izrade algoritma za procesiranje zvuka bio dosta mukotrpan. Imajući na umu da se nakon svakog uspješnog kompajliranja sam virtualni efekt morao učitati te isprobati u digitalnoj audio radnoj stanici mogu samo reći da je testiranje ovog DSP algoritma bilo izrazito dugotrajno te je iziskivalo strpljenje.

6.4. Testiranje virtualnog efekta

Kako bi se virtualni efekt testirao potrebno ga je, kako sam već natuknuo, učitati unutar digitalne audio radne stanice. Virtualni efekt učitava se na jedan od audio kanala mix konzole digitalne audio radne stanice te isti onda tamo modificira ulazni signal. U praksi to znači da se nakon bilo koje promjene nakon kompajliranja novog koda, virtualni efekt mora iznova učitavati svaki put te testirati unutar digitalne audio radne stanice. Ova činjenica uvelike usporava razvoj, dokumentacija JUCE-a nudi alternativne opcije – no htio sam u svakome trenu biti u potpunosti siguran da će sam virtualni efekt raditi u „stvarnom“ okruženju.

Sam virtualni efekt predpojačala testirao sam pomoću digitalne audio radne stanice Ardour, no i pomoću Cubase 5 digitalne audio radne stanice u lokalnom glazbenom studiju. I sesije od nekoliko sati sviranja i produciranja glazbe u potpunosti su bile podržane od strane razvijenog virtualnog efekta.

Kako je sam virtualni efekt zamišljen kao predpojačalo bez ekvilajzera uz isti se, barem tako držim, mora koristiti neki vanjski ekvilajzer kako bi se odstranile nepotrebne frekvencije – naročito u niskom frekvencijskom spektru (50 Hz – 200Hz). Također s obzirom na to da se radi o gitarskom predpojačalu potrebno je radi dobivanja upotrebljivog zvuka koristiti virtualni efekt simulacije kabineta odnosno zvučnika – primjerice besplatni virtualni efekt *keFIR* koji sam koristio prilikom testiranja.

Kombinacijom triju spomenutih virtualnih efekata – razvijenog predpojačala kojeg sam nazvao *Kodiak* prema najvećoj podvrsti mrkog medvjeda, ekvilajzera te simulacije gitarskog kabineta (zvučnika) dobio se vrlo iskoristljiv zvuk za snimanje. Ulazni signal gitare prolazio je kroz sljedeći lanac opreme:

Slika 13: Lanac virtualnih efekata

Za kraj htio bih još priložiti dvije slike, sam konačni dizajn izrađenog virtualnog efekta predpojačala *Kodiak* te sliku virtualnog efekta *Kodiak* unutar digitalne audio radne stanice Cubase 5.

Slika 14: Sučelje virtualnog efekta predpojačala „Kodiak“

Slika 15: „Kodiak“ unutar Cubase 5

7. Zaključak

Razvitkom programskog okvira JUCE programiranje virtualnih efekata te ostalih komponenata audio inženjeringa postalo je lakše negoli ikada. Uvid u sam princip rada virtualnih efekata kao takvih omogućen je i glazbenim te audio entuzijastima poput mene samog, a upravo je to skupina korisnika koja bi mogla uroditi novim idejama – prosto zato što se ta skupina korisnika do sada nije mogla tako lako okušati u programiranju komponenti audio inženjeringa.

Digitalni audio inženjering je sadašnjost i budućnost glazbene industrije, a rapidan razvoj glazbene industrije samo govori o ekonomskom potencijalu unutar same domene. Kao što sam spomenuo nekoliko puta – broj kućnih studija nikada nije bio veći, a unutar tih istih kućnih studija se koristi gotovo isključivo digitalna oprema te virtualni efekti poput ovog razvijenog u svrhu ovog završnog rada. Interesantna činjenica je da je razvojem elektronske glazbe došlo do još većeg porasta kućnih studija što se i dalje nastavlja – a takvim rastom interesa dolazi isključivo i do rasta potrošnje u odabrani hobi ili profesiju ovisno o potrošenom vremenu i volji pojedinca.

Osobno, opet napominjem, sam nevjerojatan zaljubljenik u glazbu te audio inženjering i pothvat izrađivanja vlastitog virtualnog efekta gotovo pa mi je bio dječački san koji sam eto uspio ispuniti. Namjeravam nastaviti raditi na virtualnom efektu predpojačala kojeg sam izradio dodajući ekvilajzer sekciju te još nekoliko opcija poput ugrađenog simulatora kabineta i sl.. Potencijal programiranja u svrhe audio inženjeringa je ogroman te neiskorišten – namjeravam utrošiti još velike količine vremena u proučavanje istoga jer držim temu izrazito zanimljivom te izazovnom.

Programski okvir JUCE zajedno sa svojim razvojnim okruženjem Projucer nevjerojatan je alat za razvoj komponenata audio inženjeringa i toplo se nadam da će nastaviti razvijati u nadolazećim godinama. *Kodiak* se pokazao u potpunosti stabilnim virtualnim efektom što samo pokazuje kvalitetu programskog okvira. Ostale programske alate koji su korišteni u svrhu razvoja smatram da ovdje ne treba pretjerano spominjati jer ih je jednostavno moguće izostaviti ili zamijeniti, no uvelike su ubrzali i potpomogli razvoj, to su primjerice: GIMP, Visual Studio 2017, Cubase 5, Ardour ...

Audio inženjering je znatno moćnija komponenta društvenog te kulturnog života negoli se to možda čini – to je ljudska djelatnost koja zahtjeva znatno veću pozornost jer nudi strahovitu snagu, bilo ekonomsku ili naprosto utjecajnu. Utjecaj audio inženjeringa je nezanemariv, audio inženjering je postao sveprisutan, a njegov rast i razvoj će samo postajati intenzivniji. Audio inženjering je jednostavno postao sastavni dio današnjeg života.

Popis literature

[1] David Hendy, „Noise, A Human History of Sound and Listening“, Profile Books, 2013.

[2] Steve Jones, „Rock Formation, Music Technology and Mass Communication“, SAGE Publications, 1992.

[3] Metin Bektas, „Audio Effects, Mixing and Mastering“, Metin Bektas, 2014.

[4] Joshua D. Reiss, Andres P. McPherson, „Audio Effects, Theory, Implementation and Application“, CRC Press, 2015.

[5] Martin Robinson, „Getting Started With JUCE“, Packt Publishing, 2013.

[6] David Gouveia, „Getting Started with C++ Audio Programming for Game Development“, Packt Publishing, 2013

[7] Will Prkle, „Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory“, Focal Press ,2012.

[8] Richard Boulanger, Victor Lazzarini, „The Audio Programming Book“, The MIT Press, 2010.

Popis slika