Digitalni gitarski efekti

Margan, Tino

Sveučilište u Rijeci - Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Tino Margan

Digitalni gitarski efekti

Završni rad

Mentor: doc. dr. sc. Miran Pobar

Rijeka, 15. rujna 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 1.6.2020.

Zadatak za završni rad

Pristupnik: Tino Margan

Naziv završnog rada: Digitalni gitarski efekti

Naziv završnog rada na eng. jeziku: Digital guitar effects

Sadržaj zadatka:

Proučiti i opisati tipične vrste gitarskih efekata. Opisati načine softverske (digitalne) simulacije analognih efekata. Izraditi primjer digitalnog efekta kao plugin za softver za obradu audio signala, opisati korištene alate i princip rada te komentirati dobivene rezultate.

Mentor

doc. dr. sc. Miran Pobar

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 1.6.2020.

(potpis pristupnika)

Sadržaj:

1. Uvod

Praksa je pokazala da je uz dovoljno znanja, vremena i opreme moguće uspješno emulirati gotovo svaki komad analogne opreme i pretvoriti ga u digitalnu inačicu. Ta digitalna inačica naziva se *plug-in* i koristi se unutar softvera za obradu audio signala, tj. digitalnih radnih audio stanica (engl. *DAW - Digital Audio Workstation*). Prednosti koje *plug-inovi* donose ima puno, a najveća je ta što ako želimo primijeniti željeni efekt na 15 audio kanala, ne treba nam 15 fizičkih jedinica istog efekta koje ćemo spojiti svaku na svoj kanal, već isti *plug-in* u softveru kopiramo na 15 audio kanala i dobijemo isti učinak. 15 fizičkih jedinica također puno košta, a *plug-in* jednom kupljen može se neograničeno koristiti. Ostale nezanemarive prednosti su i potreba za manje prostora (jer nije potrebno fizički smjestiti svu tu opremu), manja potrošnja struje koja bi ju inače pogonila, manji broj kablova za spajanje s kanalima na audio mikseru i manja mogućnost kvara opreme.

Slika 1 - plug-inovi u gotovo svakoj pojedinoj "traci" audio projekta

U tom digitalnom svijetu stvorilo se nezamislivo veliko tržište i počela je bitka koja tvrtka će napraviti zvukovnim karakteristikama točniju reprezentaciju svakog legendarnog audio miksera, kompresora, gitarskog pojačala, ekvilajzera i sl. To me potaknulo na razmišljanje zašto ne bih i sam pokušao napraviti digitalnu emulaciju nekog gitarskog efekta kojeg posjedujem u fizičkom obliku. Promotrit ćemo nekoliko efekata koji postoje, teoriju iza emulacije efekata i načine na koje ih je moguće i na koje ih nije moguće emulirati.

2. Efekti

Kada govorimo o električnim gitarama one su dizajnirane na način da magneti na tijelu gitare pretvaraju vibracije žica u signal i odašilju ga preko kabela u nekakvu vrstu pojačala (električne gitare same po sebi nisu glasne poput akustičnih gitara). Tijekom godina su se na putu između gitare i pojačala sve više počinjali nalaziti efekti koji modificiraju njen zvuk i daju joj nešto jedinstveno. U današnje vrijeme rijetko ćemo pronaći gitarski efekt, tj. gitarsku pedalu koja nije digitalna: digitalni efekti postale su odlične reprezentacije nekadašnjih analognih efekata, a pritom omogućuju i više vrsta modulacije zvuka od analognih pedala.[1] Nabrojat ćemo nekoliko gitarskih efekata i ukratko objasniti što rade. Pritom moramo naglasiti da se svi navedeni efekti mogu koristiti i za modulaciju zvuka drugih instrumenata, tj. ovisno o tome koji efekt želimo postići, mogu se koristiti za obradu bilo kojeg zvučnog signala.

2.1. Jeka

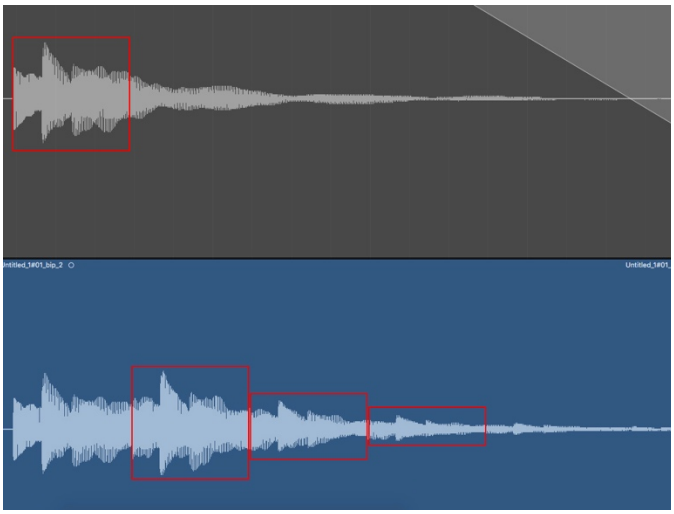
Instrumenti snimani u glazbenom studiju obično nemaju nikakvu jeku (engl. *reverb*) u svom signalu jer se u prostorijama koriste spužve, difuzori i razni materijali kako bi se spriječilo odbijanje signala od golih zidova i stvaranje neželjenih odjeka u snimci. Jeka kao efekt stavlja naglasak na kontrolirani odjek, tj. stvara privid prostorije u kojoj se nalazi instrument kojeg čujemo. Uglavnom je u svakoj verziji ovog efekta moguće kontrolirati nekoliko parametara. Trajanje jeke iskazano je u sekundama ili milisekundama i označava nam koliko će trajati jeka nakon početnog signala, npr. ako imamo signal koji traje jednu sekundu, a trajanje jeke namjestimo na dvije sekunde, izlazni signal ukupno će trajati tri sekunde. Veličina prostorije koju simuliramo drugi je parametar kojim se može manipulirati, a dojam veličine prostorije postiže se suptilnim ekvilajzerom: u manjim prostorijama zvuk se brže odbija od zidova i karakteriziraju ih više frekvencije, dok u većim prostorijama veće valne duljine (niže frekvencije) imaju prostora razviti se i time prevladavaju. Zadnji bitan parametar bio bi oblik prostorije (tj. broj zidova koji nas okružuje): zvuk će se različito odbijati u sobi trokutastog oblika za razliku od sobe četvrtastog ili čak heksagonalnog oblika pri čemu se ustvari koristi čak i blagi, skoro nečujni efekt odgode. S ta tri parametra možemo reći da imamo potpunu kontrolu nad signalom. Korištenjem jeke možemo simulirati da smo u dnevnom boravku, crkvi, pa čak i u šumi ili drugim otvorenim prostorima.

[1] Davies, P. (2018) *The secret history of guitar effects pedals* [Online]. Dostupno na: https://www.soundaffects.com/blog/2018/04/the-secret-history-of-guitar-effects-pedals/ [15. rujna 2020.]

2.2. Odgoda

Odgoda (engl. *delay*) još je jedan efekt koji stvara privid prostora. Kao što samo ime govori, ona odgađa originalni signal za zadano vrijeme. U praksi se odgoda koristi uz originalni signal što otvara beskrajne kreativne poteze koji se s tim efektom mogu napraviti. Odgoda se možda ne može čuti kao takva u sklopu cijele glazbene skupine, ali se koristi u gotovo svim pjesmama na odabranim instrumentima ili na vokalu baš iz razloga što pruža osjećaj prostranosti i popunjava praznine ili tiše dijelove u izvedbi. Na slici 2 možemo vidjeti da efekt ima dva glavna parametra koje možemo kontrolirati, a to su vrijeme nakon kojeg želimo čuti odgodu (*Note*, iskazano je u notnim vrijednostima (četvrtinka, osminka) jer je namješteno tako da prati tempo pjesme unutar koje se nalazi) i broj odgoda koje želimo čuti (*Feedback*, koliko odgoda, tj. koliko ponavljanja želimo čuti - 0% je jedna odgoda, a 100% ih je beskonačan broj). Na slici 3 vidimo snimljeni ulazni signal (gore) i izlazni signal (dolje) i kako izgleda izlazni signal nakon što se na ulazni primijeni odgoda. Crvenim kvadratima na izlaznom signalu označena su prva tri ponavljanja ulaznog signala i možemo vidjeti da su intervali ponavljanja jednakog trajanja te da se svako ponavljanje linearno stišava u zadanom vremenu.

Slika 2 - grafičko sučelje Delay plug-ina



Slika 3 - gore: početni signal, dolje: svako pojedino ponavljanje signala

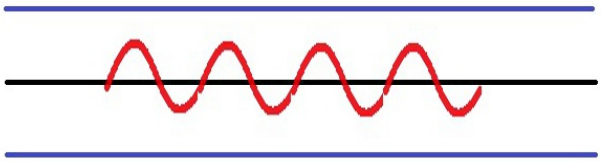
Kada bi *feedback* postavili na 100% signal bi teoretski trebao ostati iste glasnoće i ponavljati se u beskonačnost, međutim to se ne bi dogodilo sa signalom iz primjera; vidimo da ponavljanje signala započne prije nego se originalni signal u potpunosti stiša te kada se njihove glasnoće zbroje to se prenosi u iduće ponavljanje gdje im se glasnoća opet zbraja, itd. Na kraju dobivamo signal koji postaje glasniji u svakom ponavljanju jer se gradi povrh postojećeg. Funkcije *dry* i *wet* samo su balans glasnoće ulaznog i izlaznog signala. Da je *wet* postavljen na

100%, a *dry* na 0% tada u donjem signalu na slici 3 ne bi bilo ulaznog signala, već bi amplituda započinjala s prvim crvenim kvadratom. U suprotnom slučaju, da je *wet* postavljen na 0%, izlazni signal bio bi jednak ulaznom jer bi glasnoća odgode bila postavljena na nulu. *Color* parametar u ovoj inačici efekta nije ništa drugo nego filter koji propušta samo visoke frekvencije kada vrijednost ode u pozitivnom smjeru i samo niske frekvencije kada vrijednost ide u negativnom smjeru. Postavljanjem na nulu zadržavamo jednake karakteristike izlaznog signala ulaznom.

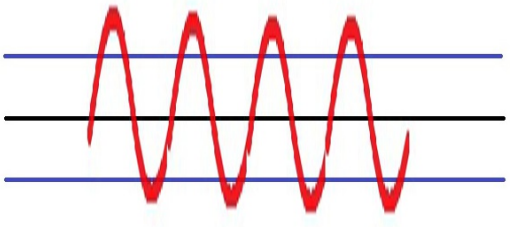
Postoji još nekoliko varijacija odgode, npr. *flanger* ili *chorus* - to su efekti koji imaju toliko kratko vrijeme odgode da ju uopće ne percipiramo kao odgodu, tj. pojedinačna ponavljanja se uopće ne čuju čime dobivamo dojam potpuno drukčijeg efekta.

2.3. Distorzija

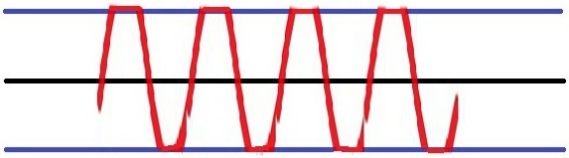
Na svaki spomen električne gitare uglavnom dolazi i asocijacija na efekt distorzije. Više je vrsta distorzije, a opisat ćemo *hard clipping* i *soft clipping* distorziju. Općenito se efekt postiže tako da se signal ekstremno pojača, toliko da vrhovi amplituda prelaze maksimalnu glasnoću koju primalac signala može podnijeti i time gube valovite sinusne oblike. Na slikama 4.1 do 4.4 vidimo što se događa sa signalom: u slučaju *hard clipping* distorzije na slici 4.3 ti se ekstremi amplitude pretvaraju u ravnu crtu, za razliku od *soft clipping* distorzije na slici 4.4 kod koje dosežu maksimum, ali ipak uspijevaju zadržati jedan mali dio valovitog svojstva. Uz električnu gitaru najčešće se povezuje *soft clipping* distorzija i upravo je to *plug-in* koji ćemo napraviti.



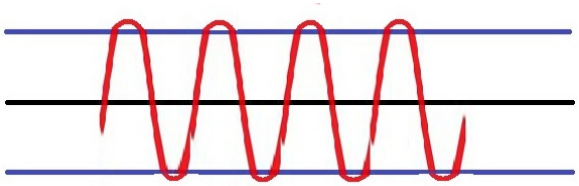
Slika 4.1 - početni signal, sinusni val



Slika 4.2 - isti signal kao na slici 4.1, ali pojačan preko granica

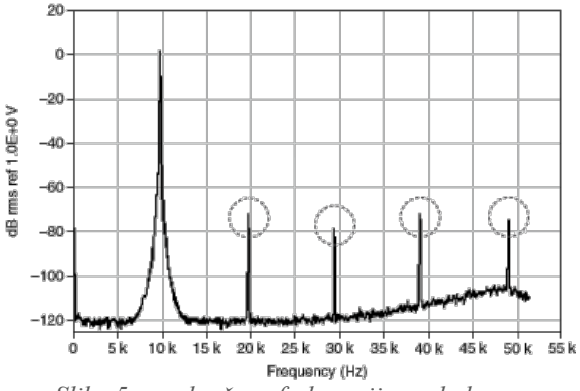


Slika 4.3 - signal sa slike 4.2 s potpuno odsječenim vrhovima (hard clipping)



Slika 4.4 - soft clipping - signal ipak zadržava neke karakteristike sinusoide

Uz rezanje vrhova amplitude većina efekata distorzije dodaje i razne harmonijske note koje obogaćuju (saturiraju) početni signal (slika 5). Te harmonijske note se ne čuju pojedinačno, ali doprinose sveukupnom signalu bez da opstruiraju zvuk ključne note. Jedini parametar kod osnovnog efekta distorzije je koja je granica (engl. *threshold*) izražena u dB do koje će efekt „odsijecati“ vrhove amplituda. Umjesto pomicanja granice, možda je lakše zamisliti da granica ostaje fiksna na 0 dB, a da signal pojačavamo toliko da vrhovi amplituda postaju, kako smo i rekli, ravne crte.



Slika 5 - zaokružene frekvencije su dodane na početnu notu, primijetimo kako su frekvencije jednakih međusobih udaljenosti (u Hz)

Efekt distorzije povezuje se s velikim tranzistorskim i cijevnim (*lampaškim*) gitarskim pojačalima koja ispod sebe imaju velike kabinete (kutije) s jednim, dva ili četiri zvučnika. Ako se ne oslanjamo na pojačala, već samo na digitalni efekt distorzije, uvelike će nam nedostajati zvuk koji proizvodi navedeni kabinet sa zvučnicima - to je ono što pretvara distorzirani signal u zvuk gitare kakav je svima poznat. To se također može simulirati na računalu tako da „uhvatimo“ impulsni odaziv zvučnika i signal gitare konvoluiramo s njim. Više o tome u poglavlju 4.3.

3. Algoritmi digitalne obrade signala

3.1. Nelinearni

Nelinearni algoritmi obrađuju ulazni signal ovisno o njegovoj glasnoći. Glasnoću ulaznog signala moguće je kontrolirati tako da povećamo ili smanjimo osjetljivost pretpojačala ili tako da povećamo ili smanjimo glasnoću samog signala kojeg reproduciramo. Recimo da želimo postići efekt kojim se signal tiši od -20 dB zanemaruje, tj. pretvara u tišinu. Što će se dogoditi ako nam nijedan dio signala nije tiši od -20 dB? Signal će ostati nepromijenjen, tj. neće se primjenjivati efekt tišine jer ne postoji dio signala koji zadovoljava taj uvjet. Sada recimo da glasnoću tog istog signala stišamo za nekoliko dB, tek toliko da najtiši dijelovi budu tiši od postavljene granice - tada se izvršava željeni efekt čime se pokazuje svojstvo nelinearnog algoritma koji strogo ovisi o glasnoći ulaznog signala. Primjer konkretnog efekta zove se *noise gate*, tj. zvučna vrata ili zvučna barijera koja se „zatvara“ ako signal nije dovoljno glasan da bi prošao tu barijeru.

3.2. Linearni

Za razliku od nelinearnih, linearni algoritmi obrađuju signal neovisno o njegovoj glasnoći. Ako je ulazni signal tih, obrađeni signal, tj. efekt će također biti tih, a ako ulazni signal pojačamo efekt će također proporcionalno biti glasniji. Primjeri su jeka, odgoda, ekvilajzer.

3.3. Vremenski zavisni

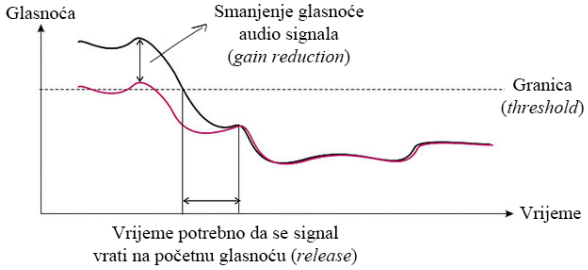
Promotrimo rad kompresora kao vremenski zavisnog algoritma. Kompresor je sprava kojoj je glavna funkcija automatizirano smanjiti (ili povećati) dinamički raspon audio signala.[2] Ljudski glas je primjer velikog dinamičkog raspona; od šapta do glasnog vikanja, kompresor sve može svesti na jednaku glasnoću. Postoji nekoliko stavki koje čine kompresor:

1. Granica (*threshold*) - izražena u dB, glasnoća iznad koje se izvršava kompresija, npr. granica od -10 dB znači da će sav signal tiši od -10 dB ostati netaknut
2. Količina kompresije (*ratio*) - izražena u omjeru glasnoće ulaznog i izlaznog signala, npr. 3:1 omjer znači da će signal koji prelazi granicu za 3 dB biti stišan na 1 dB iznad granice

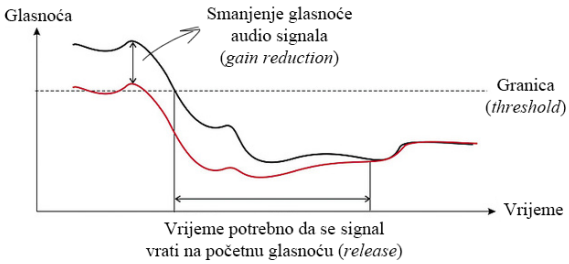
[2] White, F., Dick, T. (2016) *Analog compressor modeling* [Online]. Dostupno na: http://www2.ece.rochester.edu/~zduan/teaching/ece472/projects/2016/White_Dick_paper.pdf [15. rujna 2020.]

3. Vrijeme potrebno za izvršavanje kompresije (*attack*) - izraženo u milisekundama, određuje koliko vremena će proći prije negoli se signal počne komprimirati, npr. 10 ms znači da će kompresor biti u funkciji 10 ms nakon što amplituda prijeđe zadanu granicu
4. Vrijeme potrebno za povratak u „stanje mirovanja“ (*release*) - izraženo u milisekundama, određuje koliko vremena će proći prije negoli se komprimirani signal vrati na originalnu glasnoću, npr. 500 ms znači da će komprimiranom signalu trebati 500 ms da se vrati na neobrađenu glasnoću.

Napravimo primjer od vrijednosti koje smo naveli. Imamo promatrani signal koji u jednome trenutku dosegne amplitudu od -6 dB. To znači da 10 ms nakon što se ta amplituda pojavi, kompresor reagira i spušta glasnoću signala u zadanom omjeru (3:1). Sada kompresoru treba 500 ms da „popusti“, tj. da se vrati u stanje mirovanja. To znači da će sav signal koji se nalazi unutar tih 500 ms od amplitude koja je prešla granicu biti tiši nego što bi bio da te amplitude nije bilo. Prema tome, kompresor neće imati jednaki odaziv ako se odjednom dogodi veliki skok u glasnoći ili ako tog skoka nema. Ponovimo kako je naglasak na vremenu, a ne na glasnoći signala (pod pretpostavkom da je signal dovoljno glasan da pokrene rad kompresora).



Slika 6.1 - signalu je potrebno kratko vrijeme da dođe do originalne razine nakon što mu amplituda padne ispod granice



Slika 6.2 - signalu je potrebno znatno duže nego u slici 6.1 što znači da je "release" duži

3.4. Vremenski nezavisni

Za razliku od zavisnih, vremenski nezavisni algoritmi obrađivat će signal na isti način bez obzira je li se milisekundu prije dogodila velika amplituda ili je bila minuta tišine. Primjeri vremenski nezavisnih algoritama su jeka, odgoda, distorzija, ekvilajzer. To znači da ako nam je jeka kao efekt namještena da traje 30 sekundi, ona će se svakako čuti, ali ni po čemu neće utjecati na karakteristike signala koji također prolazi kroz jeku unutar tih 30 sekundi.

4. Emulacija efekata

Spomenuli smo kako je nekada bilo potrebno imati više fizičkih jedinica istog efekta kako bi se postigao jednak učinak kao danas s *plug-inovima*. Danas, godinama kasnije, veliki audio inženjeri tvrde kako dva ili više komada te iste opreme ne zvuče jednako i čak preferiraju pojedinu opremu za različite instrumente. To je posve normalno - oprema se troši, vlaga utječe na komponente koje ju čine, neki popravci zahtijevaju zamjenu starih komponenata, itd. Tvrdnjama da njihova oprema zvuči bolje nego od bilo koga drugoga potaknula je velike tvrtke koje se bave emulacijom efekata da dođu u njihov glazbeni studio i emuliraju točno tu spravu. Time su omogućili ljudima diljem svijeta da kupe digitalnu inačicu koju su proizveli i praktički koriste isti identični efekt koji koriste i najveća imena glazbene industrije.

Najveći izazovi s kojima se susreću timovi koji sudjeluju u izradi emulacija tih sprava su modeliranje nelinearnih signala i optimizacija računalne snage potrebne za obradu audio signala u stvarnom vremenu.[3] Nelinearne karakteristike obično su glavni razlog zašto je analogna oprema toliko poželjna - na dva jednaka signala reagirat će za nijansu drukčije i te male nepravilnosti daju signalu i krajnjem proizvodu nešto „organsko“. Međutim, što je tih nelinearnih karakteristika više, to je kompleksniji model potrebno izraditi, a on zahtijeva i koristi više procesorske snage.

4.1. Emulacija po komponentama

Pošto je sva analogna oprema napravljena od komponenata (kondenzatora, tranzistora, otpornika, itd.) za koje je poznato na koji način reagiraju na ulazni signal, emulacija bazirana na komponentama je početna, ako ne i glavna metoda modeliranja analogne opreme. Prvi korak bio bi provjeriti odgovaraju li sve komponente u kućištu sprave njenom shematskom prikazu.[4] Nakon toga potrebno je shvatiti kuda teče signal kroz komponente, analizirati ga i započeti s matematičkim jednadžbama koje bi odgovarale performansama i karakteristikama svake od komponenti. Neke jednadžbe su jednostavne, npr. ako komponenta čini signal dvostruko glasnijim onda signal pomnožimo s faktorom 2, ali za neke situacije je potrebno izmisliti jednadžbe koje će postići isti učinak kao što ga analogna sprava postiže. Primjer takve teže jednadžbe može biti nekakav fazni pomak koji se događa prilikom promjene nekoliko

[3] Lambert, M. (2010) *Plug-in modeling* [Online]. Dostupno na: https://www.soundonsound.com/techniques/plug-modelling [15. rujna 2020.]

[4] Waves.com (2019) *How Waves' modeling captures analog magic in a digital world* [Online]. Dostupno na: https://www.waves.com/how-waves-modeling-captures-analog-magic [15. rujna 2020.]

analognih filtara i za takve situacije potrebno je pronaći način koji će to prenijeti u matematički oblik. Nakon analize svih komponenti cijeli sustav se prebacuje u softver (npr. MATLAB) koji može mjeriti performanse svih jednadžbi i pokušati primijeniti sve to na ulazni testni signal koji mu pružimo. Od tuda se radi usporedba s originalnom spravom i ugađaju se parametri kako bi se došlo što bliže zvuku originalne sprave.

4.2. Emulacija *LTI* sustava

Kod linearnih vremenski nezavisnih sustava (engl. *LTI - Linear Time Invariant Systems*) proces se može pojednostaviti - njihov izlazni signal jednak je linearnoj kombinaciji individualnih signala, u ovom slučaju ulaznog signala i impulsnog odaziva efekta kroz kojeg signal prolazi.[5] Promjena ulaznog signala rezultirat će jednakom linearnom promjenom izlaznog signala. Teoretski zvuči jednostavno: ako imamo poznati ulazni signal koji prolazi kroz neku vrstu efekta iz kojeg izađe promijenjen onda je proces koji se događa u tom efektu razlika između izlaznog i ulaznog signala. Takve sustave možemo opisati jednom funkcijom, a to je impulsni odaziv (engl. *Impulse response*), tj. izlazni signal *LTI* sustava je konvolucija ulaznog signala s impulsnim odazivom sustava. U konvoluciju i dekonvoluciju nećemo previše ulaziti, tu se susrećemo s puno matematike, integralima i Fourierovim transformacijama, zbog čega postoje razni softveri koji pokušavaju odraditi težak posao za nas.

4.3. *Impulse Response Utility*

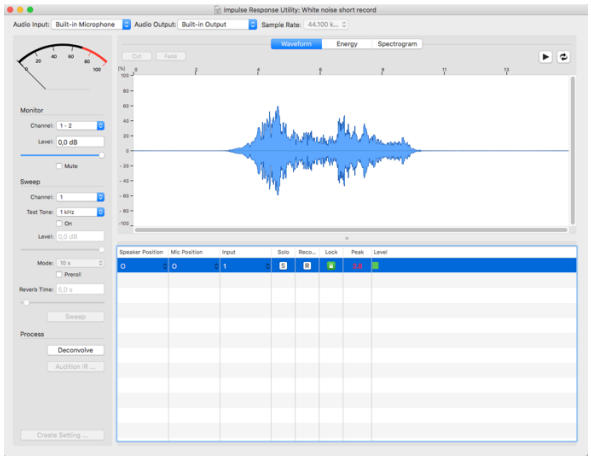
Apple unutar *Logic Pro X DAW*-a ima program naziva *Impulse Response Utility* koji radi dekonvoluciju signala u slučaju da želimo simulirati zvuk neke prostorije i prikazati kako bi zvuk koji reproduciramo zvučao u njoj. Postoje dva načina izrade impulsnog odaziva: reprodukcija beskonačno glasnog i beskonačno kratkog signala i reprodukcija logaritamskog sinusnog vala u rasponu od 20 Hz do 20 kHz.[6] Reprodukcija beskonačno glasnog i beskonačno kratkog signala moguća je samo teoretski i to bi prikupilo dovoljno informacija da se točno emulira promatrani *LTI* sustav. U praksi je to nemoguće, ali najbliže što tome možemo doći je reprodukcija bijelog šuma (engl. *white noise*) trajanja jednog uzorka (engl. *1 sample*). Bijeli šum simultano reproducira čitav frekvencijski spektar pa je najbolji kandidat za „hvatanje“ impulsnog odaziva. Druga varijacija bila bi pucanj startnog pištolja, međutim to je zvuk dužeg

[5] Brilliant.org *Linear time invariant systems* [Online]. Dostupno na: https://brilliant.org/wiki/linear-time-invariant-systems/ [15. rujna 2020.]

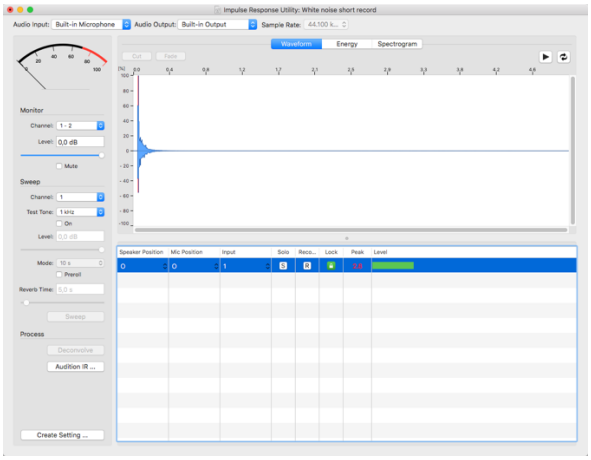
[6] Soundwoofer.se (2018) *How to create/record an impulse response* [Online]. Dostupno na: https://www.soundwoofer.se/blog/2018/03/29/how-to-create-record-an-impulse-response/ [15. rujna 2020.]

trajanja od jednog uzorka i ne karakteriziraju ga duboki tonovi pa u tom frekvencijskom spektru sustav ne bi bio dobro emuliran.

Time dolazimo do druge metode. Program kroz izlazni uređaj (zvučnike) reproducira logaritamski sinusni val u rasponu od 20 Hz do 20 kHz (engl. *sine sweep*) dok simultano kroz ulazni uređaj (mikrofon) snima kako taj isti sinusni val zvuči nakon što ga prostorija „obradi“. Poanta je ista kao u prethodnom primjeru: prostorija čiju emulaciju radimo uvelike utječe na sinusni val koji se reproducira, a mikrofon služi da zabilježi sve njene sitne utjecaje koje će kasnije računalo usporediti s prvotno reproduciranim sinusnim valom. *Sine sweep* je drukčiji jer nije beskonačno kratak, već traje 10 ili 50 sekundi (mogući odabiri u programu) te na dobivenom izlaznom signalu treba izvršiti proces dekonvolucije. U dekonvoluciji program usklađuje svaki snimljeni uzorak u vremenu i glasnoći i smješta sve uzorke na sami početak datoteke. Rezultirajući signal je impulsni odaziv koji možemo kombinirati i konvoluirati s bilo kojim ulaznim signalom i izlazni signal zvučat će kao da je snimljen upravo u prostoriji koju smo emulirali.[7]



Slika 7.1 - prikaz sine sweep-a koji je prošao kroz hardversku jedinicu



Slika 7.2 - proces dekonvolucije koji je izvršen na signalu sa slike 7.1 - dobiven je impulsni odaziv

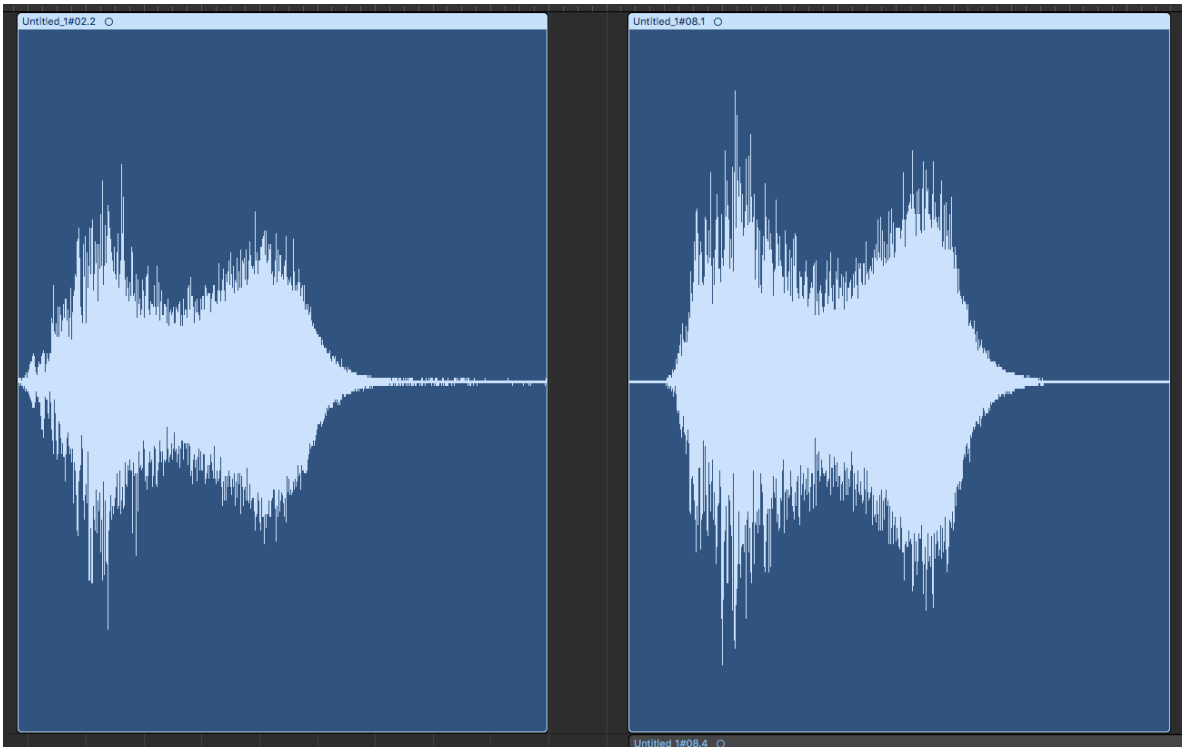
U svome posjedu imam gitarski efekt jeke, ali s dodatnim karakteristikama. Taj efekt ne samo da produži trajanje početnog signala i napravi nam dojam kao da se nalazimo u nekakvom prostoru nego tu jeku reproducira u nekoliko harmonijskih tonova (prva oktava, druga oktava, treća oktava, kvinta, itd.) uz originalni ton našeg signala. To nam pruža dojam kao da, primjerice, uz gitaru sviraju i violine, sintesajzeri, mnogi se za konkretan efekt usude reći i

[7] Apple.com *Impulse Response Utility user manual* [Online]. Dostupno na: https://help.apple.com/impulseresponseutility/mac/1.0.3/en/impulseresponseutility/usermanual/Impulse%20Response%20Utility%20User%20Manual%20(en).pdf [15. rujna 2020.]

nebeski i anđeoski glasovi. Spomenuti efekt pokušao sam dobiti u digitalnom obliku jer sam mislio da neće biti teško pomoću *Impulse Response Utility-a*, međutim rezultati nisu proizašli kako sam očekivao. Za sljedeće audio primjere namjestio sam da gitaru snimam u jednoj „traci" čisti izlazni signal, a u drugoj „traci" čisti izlaz iz gitarskog efekta. Zatim sam kreirao treću „traku" gdje sam duplicirao čisti signal gitare i na njega primijenio konvoluirani efekt koji sam napravio pomoću *sine sweep-a* unutar *Impulse Response Utility-a*.

	1. Čisti signal gitare
	2. Čisti signal hardverske jedinice
	3. *Plug-in* napravljen *Impulse Response Utility*-em

Kao što se može čuti, zvučni zapis pod brojem 3 puno je drukčiji od broja 2, a trebali bi zvučati jednako. Trajanje jeke moguće je namjestiti na samoj fizičkoj jedinici i što se trajanja tiče *Impulse Response Utility* točno emulira trajanje efekta (kako promijenimo na hardveru tako se promijeni i na *plug-inu)*. Nažalost ostale specifične karakteristike ovog efekta ne replicira dobro (nema velikog broja harmonijskih tonova koji obogaćuju početni signal i nema blage varijacije u tonu navedenih oktava i kvinti (engl. *detune*) zbog kojih se efekt ponaša drukčije svaki put). Karakteristike *LTI* sustava su takve da na jednake signale svaki put jednako reagiraju, ali sam isti signal pustio kroz hardver nekoliko puta i nijednom nije proizašao identičan (slika 8) što mi govori da sustav koji pokušavam emulirati nije *LTI* sustav.

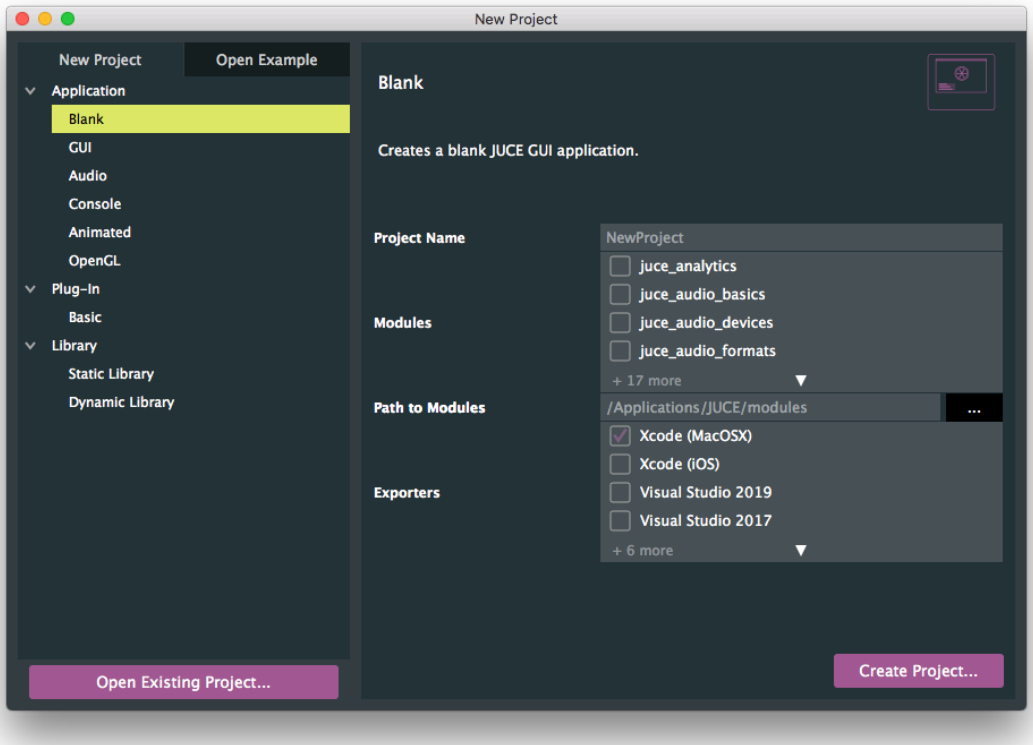


Slika 8 - zvuk lijeve i desne amplitude trebao bi biti jednak, ali i bez da slušamo, naizgled vidimo da nisu jednake, što znači da ne opisujemo LTI sustav

5. Izrada *plug-ina* u JUCE Frameworku

JUCE je C++ razvojni okvir (engl. *framework*) djelomično otvorenog koda (engl. *open-source*) koji se koristi za razvoj kako računalnih tako i mobilnih aplikacija, *plug-inova*, animacija i sl. JUCE je moguće koristiti na Windows operativnom sustavu, macOS-u i Linux-u, pa će se na isti način i izlazne aplikacije koje korisnik kreira moći pokrenuti na svim operativnim sustavima jednako, tzv. međuplatformske aplikacije (engl. *cross-platform applications*).[8] U JUCE-u je moguće i izrađivati videoigre zahvaljujući OpenGL sustavu te sam na njihovom forumu naišao na ljude koji su čak u audio *plug-ine* ubacivali videoigre poput Sudoku, 2048 ili Solitaire kao skrivena iznenađenja.

Pri otvaranju „Projucer“ aplikacije ekran dobrodošlice nudi nam izbor predložaka (slika 9) među kojima možemo izdvojiti izradu grafičkog sučelja za aplikaciju, izradu osnovnog audio grafičkog sučelja s osnovnim mogućnostima spajanja ulaznih i izlaznih signala u računalo, izradu animacija unutar grafičkog sučelja aplikacije i predmet ovog rada - izradu *plug-inova*.[9]



Slika 9 - početno sučelje JUCE framework-a prilikom kreiranja projekta

[8] Wikipedia.org (2020) *JUCE* [Online]. Dostupno na: https://en.wikipedia.org/wiki/JUCE [15. rujna 2020.]
[9] JUCE *Getting started with the Projucer* [Online]. Dostupno na: https://docs.juce.com/master/tutorial_new_projucer_project.html [15. rujna 2020.]

JUCE zatim automatski odabire module koji će nam biti potrebni u izradi odabranog predloška i kreira datoteke, npr. PluginProcessor.cpp u koju ide sva logika procesiranja ulaznih i izlaznih signala i PluginEditor.cpp koji možemo gledati kao datoteku za uređivanje izgleda *plug-ina* i povezivanja grafičkog sučelja s funkcijama aplikacije. Uz njih kreiraju se i datoteke s nastavkom .h (*header*) unutar kojih se deklariraju brojne klase koje JUCE smatra potrebnima za izradu projekta prema predlošku kojeg smo odabrali. Puno funkcija koje se automatski deklariraju u svom programu nisam niti koristio, npr. u datoteci PluginProcessor.cpp definiraju se funkcije vezane za ulaz, obradu (procesiranje) i izlaz MIDI (*Musical Instrument Digital Interface*) izvora koje, iako korisne u slučaju da planiramo koristiti MIDI kontrolere, za ovaj konkretan slučaj nisu potrebne. MIDI kontroleri ne proizvode zvučne valove, već prenose skupove podataka koje potom računalo interpretira onako kako korisnik zada (korisnik namješta zvuk koji želi čuti; to može biti od violine do bubnjeva, klavira i mnogih drugih sintetičkih zvukova). Između ostaloga, moguće je upravljati visinom note, trajanjem, jačinom, a računalni sintesajzer manipulira tim podacima i daje nam izlazni signal u zvučnom obliku. Velika većina današnje elektroničke ili pop glazbe oslanja se na MIDI podatke za zvukove koje čujemo u pjesmama.

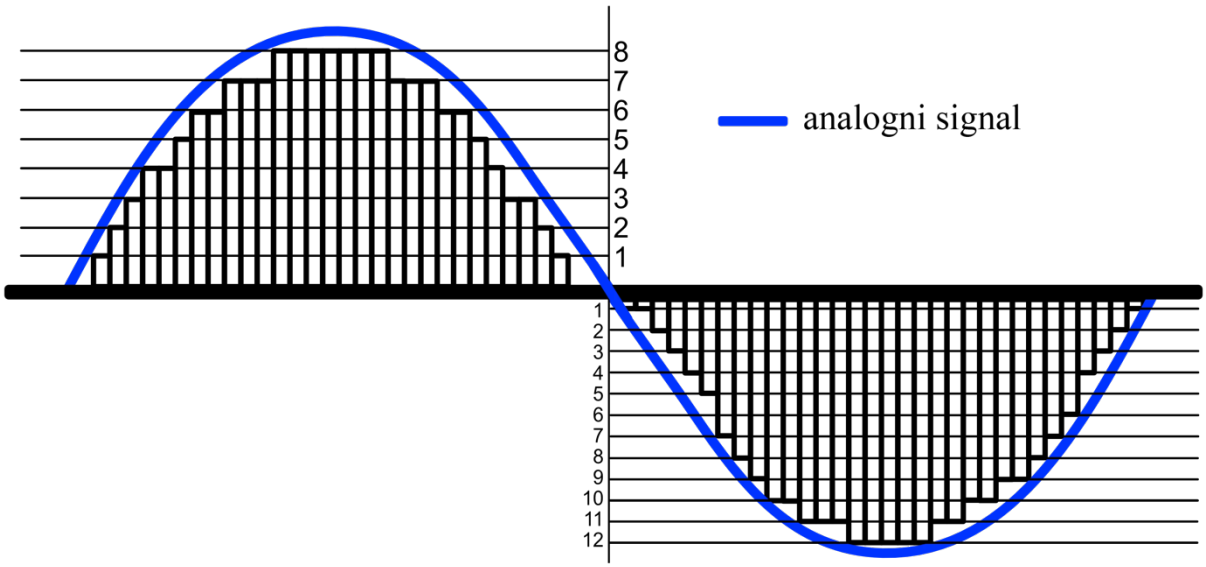
Ukratko ćemo opisati klase koje su korištene u izradi *plug-ina*, od kojih je najbitnija `AudioProcessor`. To je glavna klasa u kojoj su definirane, tj. iz koje proizlaze sve funkcije vezane uz procesiranje zvuka i *plug-in* općenito. Najbitnija funkcija unutar te klase je `processBlock` u kojoj se definira sve što aplikacija treba raditi, npr. sve modulacije ulaznog signala koje ćemo opisati u poglavlju 5.2. radimo pod tom funkcijom. `AudioProcessorEditor` je klasa koja služi kao uređivač grafičkog sučelja za navedeni `AudioProcessor`. Ona sadrži sve što bi očekivali za izradu grafičkog sučelja – veličina prozora aplikacije/*plug-ina*, kreiranje grafičkih elemenata (poput klizača, tj. potenciometara koje ćemo pokazati kako se kreiraju) kojima potom dajemo funkcionalnosti, kreiranje i uređivanje tekstualnih elemenata, itd.

5.1. Obrada zvuka u stvarnom vremenu

Digitalni signal definira se brojem uzoraka u sekundi (engl. *sample rate*) i brojem kvantizacijskih razina u jednom uzorku (engl. *bit depth*).[10] Ako zamislimo to kao koordinatni

[10] Brown, G. (2019) *Digital audio basics: sample rate and bit depth* [Online]. Dostupno na: https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html [15. rujna 2020.]

sustav, x-os predstavlja uzorke u vremenu, a y-os razinu kvantizacijskog bita u svakom zadanom uzorku. Na slici 10 možemo vidjeti u ilustrativne svrhe pretjerano uvećanu razliku između 16-bitnog signala i 24-bitnog signala. U stvarnosti nemamo brojke od 1 do 8, odnosno od 1 do 12 na svakoj polovici amplitude, već imamo ukupno 2^16 i 2^24 bitova, tj. 65536 i 16777216 razina na koje možemo smjestiti bit.[11] Kada pogledamo te brojke postane nam jednostavno shvatiti koliko bolju aproksimaciju zvučnog vala će napraviti 24-bitni signal u odnosu na 16-bitni kad ima 16711680 više razina na koje može smjestiti bitove u koordinatnom sustavu. Time je smanjena kvantizacijska greška i šum (distorzija) koja nastaje prilikom aproksimacija zvučnog vala.



Slika 10 - ilustracija razlike između signala različite količine kvantizacijskih bitova: lijevo je manji broj kvantizacijskih razina, a desno veći čime se postiže bolja aproksimacija zvučnog vala

Dolaskom CD-a na tržište industrijski standard za broj uzoraka u sekundi i broj kvantizacijskih bitova 1979. godine postaje 44,1 kHz, odnosno 44100 uzoraka u sekundi i 16-bitni format. Niti 10 godina kasnije, 1987. u profesionalnoj glazbenoj industriji standard digitalnog snimanja i procesiranja postaje 48000 uzoraka u sekundi, a pojavom DVD-a i Blu-ray diskova broj kvantizacijskih bitova pomiče se sa 16 na 24.[12] Danas se, zahvaljujući velikim pomacima u snazi procesora, teži većim brojem uzoraka u sekundi pa se uz 88200 i 96000 u rijetkim slučajevima susrećemo čak s 192000 uzoraka u sekundi što je veliko opterećenje za procesor, pogotovo ako odlučimo obrađivati signal u stvarnom vremenu s nekoliko *plug-inova*.

[11] Burk, P., Polansky, L., Repetto, D., Roberts, M., Rockmore, D. (1998) *Music and Computers* [Online]. Dostupno na: http://musicandcomputersbook.com/ [19. rujna 2020.]

[12] Wikipedia.org (2020) *44,100 Hz* [Online]. Dostupno na: https://en.wikipedia.org/wiki/44,100_Hz [15. rujna 2020.]

JUCE provjerava koji broj uzoraka nam je namješten u projektu preko funkcije `getNumSamples()` i jednostavnom `for` petljom prolazi kroz svaki uzorak u našem ulaznom signalu. Čim može dohvatiti svaki uzorak to znači da njime može i manipulirati na koji god način isprogramiramo. U slučaju napravljenog *plug-ina* isprogramirali smo da svaki uzorak „provlačimo" kroz istu formulu koja glasi:

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

pri čemu je x ulazni signal kojim manipuliramo.[13] Time mijenjamo svaki uzorak dok god postoji ulazni signal, a rezultat nam je distorzirani signal koji u slučaju gitare možemo upariti sa zvukom kabineta, kako smo i govorili u poglavlju 2.3., čime dobivamo sasvim zadovoljavajući, dobro zvučeći izlazni signal. Izraz α odnosi se na količinu distorzije koju želimo. Ako α postavimo na nulu, to znači da x množimo s nulom, odnosno ulazni signal postaje nula (ne dobivamo nikakav izlazni signal).

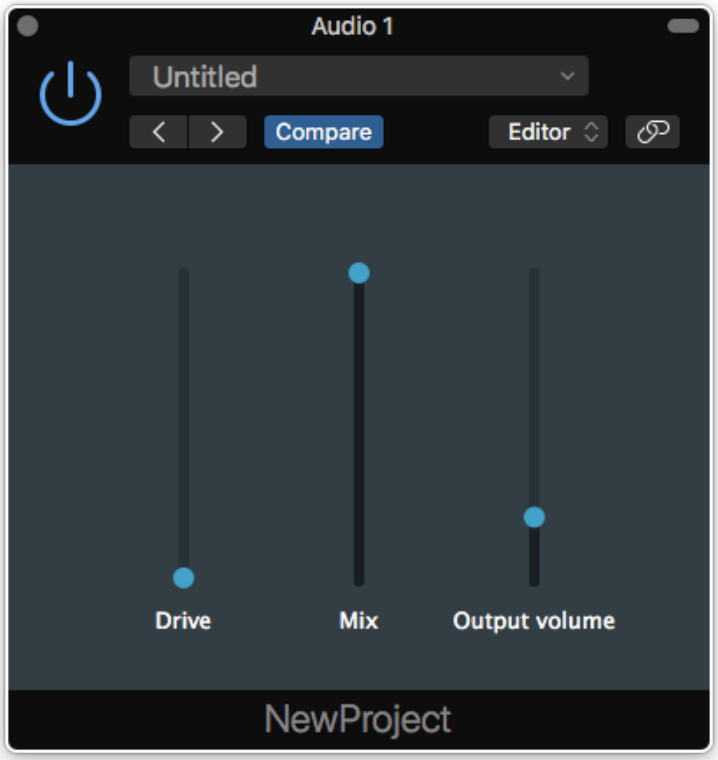
5.2. *Plug-in*

GUI naše aplikacije mijenjamo u datoteci PluginEditor.cpp gdje pomoću `juce::Graphics` dobivamo pristup raznim funkcijama, primjerice `setColour`, `setFont`, `drawText` kojima možemo dodavati i uređivati tekst kojeg bi obično stavili ispod pojedinog potenciometra kojeg kreiramo. Potenciometre kreiramo u istoj datoteci; za početak ih moramo kreirati i postaviti da budu vidljivi pomoću `addAndMakeVisible` funkcije.

```
addAndMakeVisible(driveKnob = new juce::Slider("Drive"));
```

[13] Hackaudio.com *Soft clipping* [Online]. Dostupno na: https://www.hackaudio.com/digital-signal-processing/distortion-effects/soft-clipping/ [15. rujna 2020.]

Ta funkcija stvara *Drive* potenciometar koji vidimo na slici 11 nakon čega možemo birati vrstu potenciometra, odnosno klizača (horizontalni, vertikalni, kružni, itd.) i smjestiti ga u prozor aplikacije, tj. *plug-ina*.

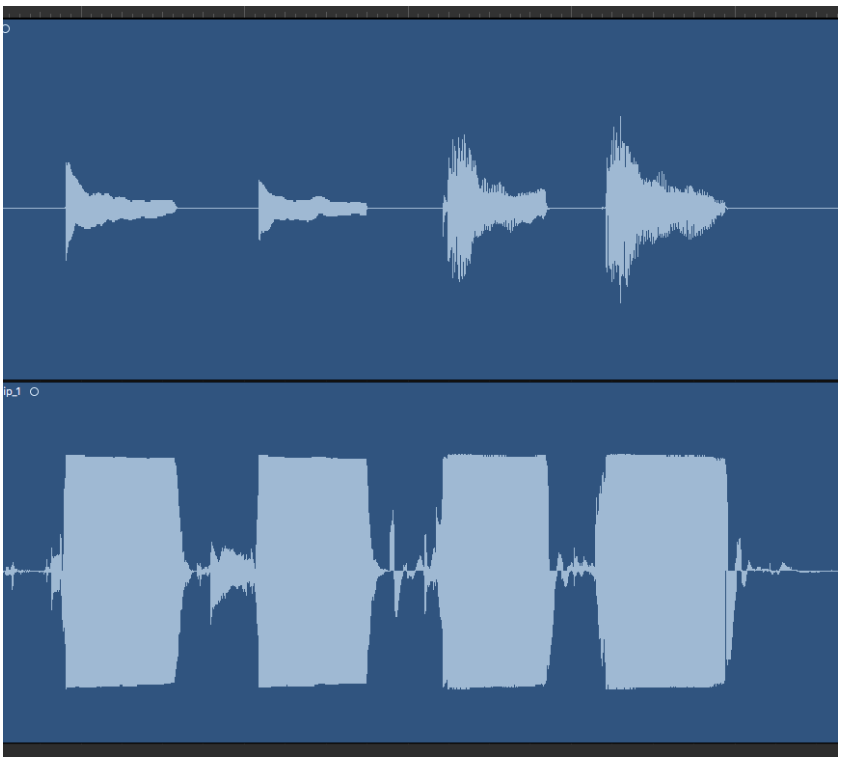


Slika 11 - grafičko sučelje napravljenog plug-ina

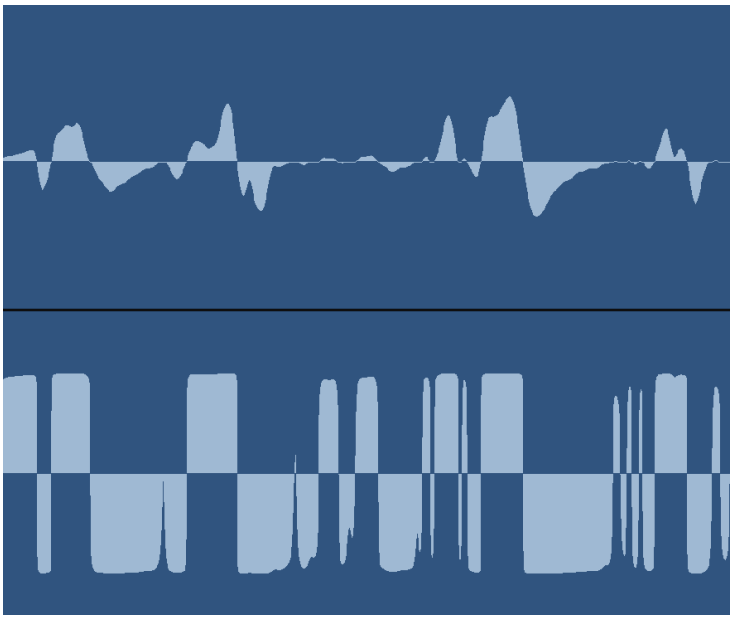
```
driveKnob->setSliderStyle(SliderStyle::LinearVertical);
```

Odabrao sam linearni vertikalni klizač jer mi se doimao najintuitivnijim. Ako odaberemo kružni potenciometar čak možemo birati hoćemo li mu mijenjati vrijednost kružnim, horizontalnim ili vertikalnim pokretima kursora. Preko funkcije `getState` prosljeđujemo podatke o vrijednosti kreiranih potenciometra formuli u kojoj manipuliramo tim podacima. Kao što vidimo na slici 11 dodao sam tri potenciometra kojima mogu upravljati da bih postigao zvuk koji želim: *Drive*, *Mix*, i *Output volume*. *Drive*, kako smo i rekli, nalazimo kao α u samoj funkciji distorzije koja se množi s vrijednosti ulaznog signala. Ako *drive* postavimo na nulu, cijeli taj izraz postaje nula i ne dobivamo nikakav izlazni signal. *Mix* je omjer „sirovog“ i obrađenog signala, ako je klizač namješten na 0% čut ćemo samo neobrađeni signal, dok ako je namješten na 100% neobrađeni signal nećemo čuti uopće. *Output volume* jednostavno je glasnoća izlaznog signala. Distorzirani signal u pravilu se doima glasnijim, tako da je dobro imati mogućnost podešavanja glasnoće unutar samog *plug-ina*. Na slikama 12.1 do 12.3 vidimo

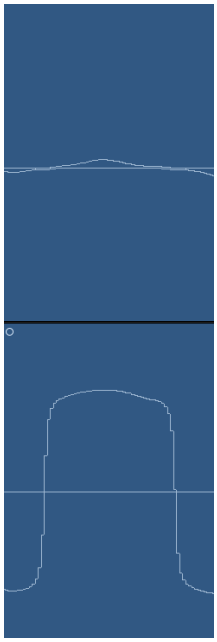
primjer distorzije na snimljenom zvuku gitare. Na gornjim dijelovima slika nalazi se ulazni signal, a na donjim dijelovima vidimo taj isti signal nakon što ga obradi napravljeni *plug-in*.



Slika 12.1 - gore: početni ulazni signal, dolje: distorzirani obrađeni signal

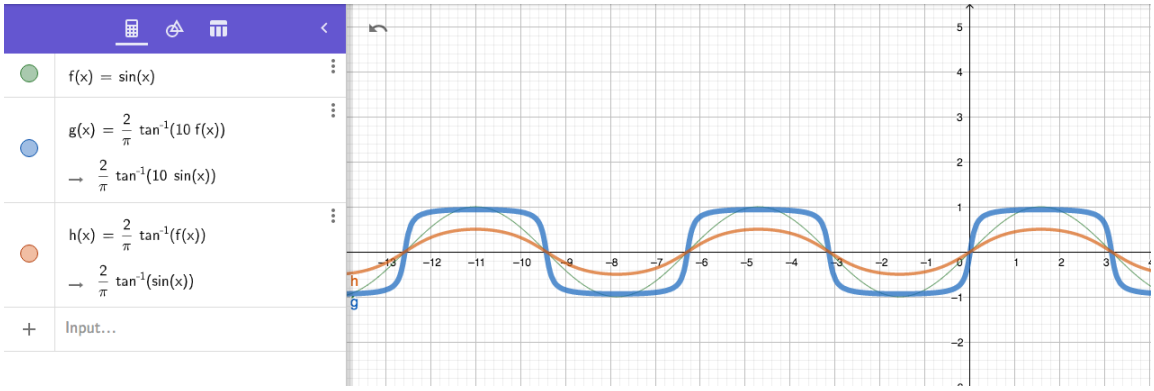


Slika 12.2 - uvećani zvučni val sa slika 12.1 da se vide ekstremna pojačanja u amplitudi signala



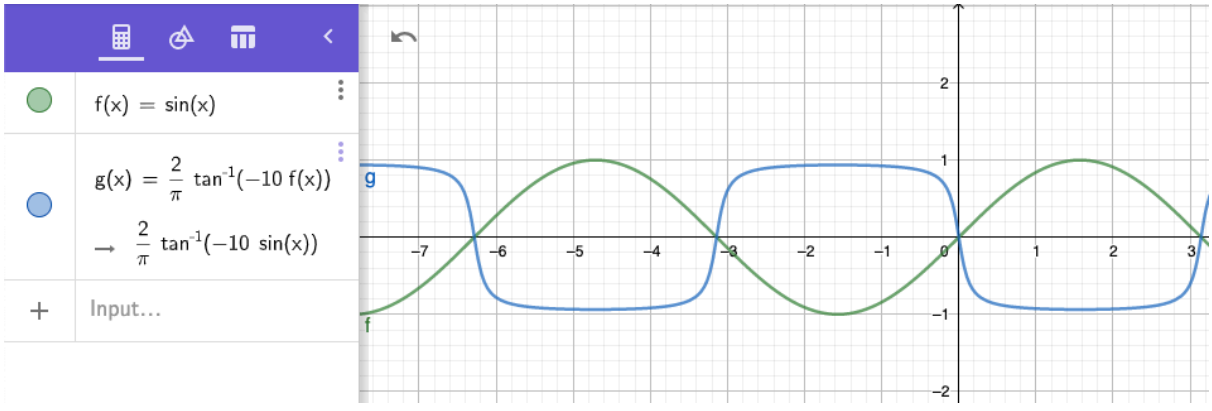
Slika 12.3 - maksimalno uvećana jedna amplituda signala da se dočara koliko je ustvari on pojačan

Pomoću GeoGebre pokušao sam eksperimentirati i pronaći koja bi funkcija mogla dati zanimljiv izlazni signal. Za ulaznu funkciju stavio sam obični sinusni val $f(x) = \sin x$, a zatim sam uzeo drugu funkciju $g(x)$ koja njome manipulira. Ako u GeoGebru unesemo funkciju koju smo prethodno naveli i zapišemo je u obliku $g(x) = \frac{\pi}{2}\tan^{-1}(\alpha f(x))$ na slici 13 možemo vidjeti što se događa s amplitudom u grafičkom smislu. Za α vrijednost najbolje je uzeti velike brojeve kako bi dobili stvarni dojam što se događa s valom u ekstremnim slučajevima te sam uzeo vrijednost 10, čime pokazujemo što smo i tvrdili u poglavlju 2.3., vrhovi amplituda postaju gotovo odsječeni, ali i dalje zadržavaju neke karakteristike sinusoide.



Slika 13 - podebljana plava arctan funkcija prikazana s velikim vrijednostima i narančasta arctan funkcija prikazana bez amplifikacije funkcijske vrijednosti

Negativne vrijednosti, kao što vidimo na slici 14 ne vrijedi uzimati jer, iako daju jednaki zvuk kao i pozitivne, ne možemo koristiti *mix* funkciju kako smo je namijenili. Kada bi *mix* klizač bio na 50% imali bi pola ulaznog signala i pola izlaznog signala, ali bi faza izlaznog signala bila obrnuta u odnosu na ulazni i time bi došlo do djelomičnog poništavanja signala.



Slika 14 – ista arctan funkcija kao na slici 13, ali s negativnim velikim vrijednostima

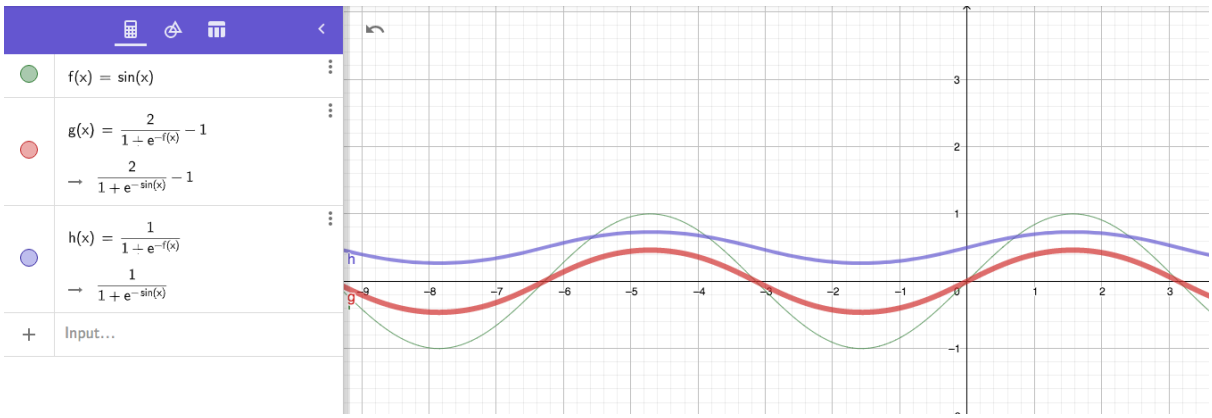
Zatim sam pokušao napraviti drugu verziju *plug-ina* sa sigmoidnom funkcijom čija formula glasi $f(x) = \frac{1}{1+e^{-x}}$, međutim s ovakvom formulom signal oscilira premalo i to oscilira isključivo u pozitivnim vrijednostima. Prema tome, koeficijente u formuli potrebno je malo modificirati kako bi amplituda prolazila x-os, tj. imala dovoljno velike oscilacije kojima prelazi x-os. Funkcija sada glasi:

$$f(x) = \frac{2}{1+e^{-x}} - 1,$$

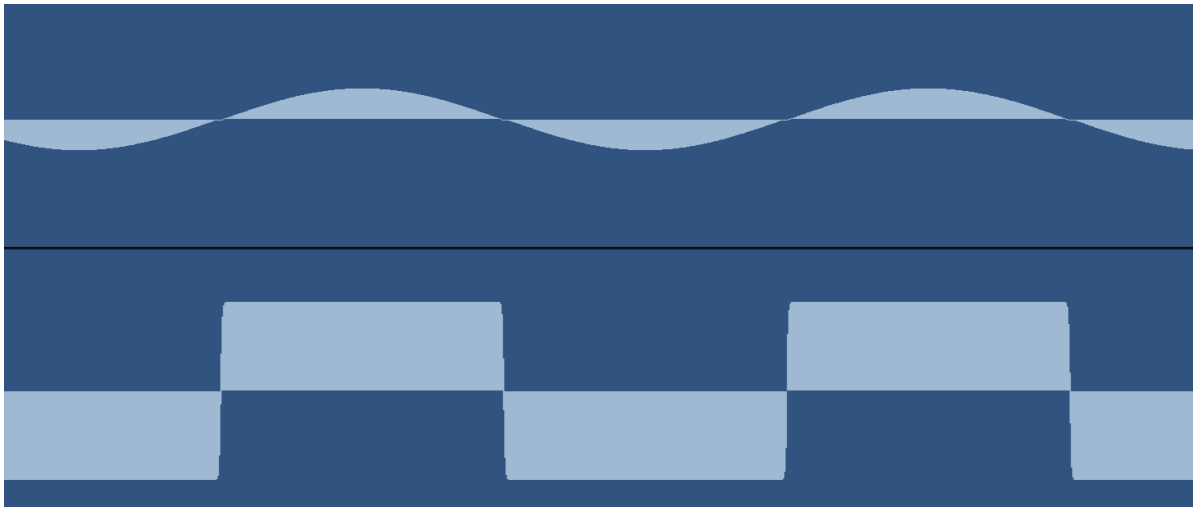
pri čemu je -1 negativan pomak na y-osi, a brojnik je postavljen na 2 kako bi raspon amplitude funkcije bio što sličniji prethodnoj *arctan* funkciji koju smo demonstrirali. Linija koda u kojoj manipuliramo signalom glasi:

```
((2/(1+exp(-(*channelData * drive)))))-1
```

pri čemu je `channelData` naš x, odnosno amplituda ulaznog signala u svakom trenutnom uzorku i, kako smo prethodno i opisali, putem `for` petlje mijenja se 44100 puta svake sekunde. Na slici 15 vidimo grafičku usporedbu funkcije prije i nakon što smo je modificirali, a na slici 16 vidimo kako izgleda ta funkcija „pretočena“ u *plug-in*.



Slika 15 - tanja ljubičasta linija prikazuje funkciju prije nego smo je modificirali, a deblja crvena krivulja pokazuje modificiranu funkciju



Slika 16 - prikaz izvornog sinusnog signala (gore) i obrađenog, distorziranog signala (dolje)

Što se tiče zvuka kojeg proizvodi ova funkcija, proveo sam test kako bih je usporedio s *arctan* funkcijom s nekoliko izvora zvuka (sinusni val u nekoliko frekvencija, gitara) i ne mogu reći da čujem razliku između njih, tj. amplituda ima jednake karakteristike u oba slučaja i logično je zaključiti da će zvučati vrlo, vrlo slično. Da je funkcija kompleksnija za izračun svakako bi za takvu neznatnu razliku odabrao funkciju koja zahtijeva manje procesorske snage.

6. Zaključak

Nakon nekoliko godina rada u glazbenoj industriji, tek sada mogu vidjeti koliko su zahtjevni procesi koji se obavljaju samo kako bi *plug-in* radio svoju funkciju i sve više mogu cijeniti snagu i optimizaciju procesora koji nerijetko mogu podnijeti rad višestrukih *plug-inova* po „traci“ na preko 20 simultanih audio izvora. Izuzetno me veseli što je ovakav *framework* dostupan i izuzetno jednostavan i za početnike koji se nikad nisu susreli izradom *plug-inova*, a najbolje od svega je što na kraju mogu isprobati svoju kreaciju i, uz malo znanja, dotjerati je da zvuči kako želim.

Također, dok me količina matematike u ovoj grani nije iznenadila, sve sam više počeo shvaćati koliko je ona bitna u manipulaciji zvukova i kako su sve te zvukovne karakteristike ustvari rezultat raznoraznih funkcija. Veselim se daljnjem poboljšanju, eksperimentiranju i radu na optimizaciji trenutnih *plug-inova*, kao i izradi novih, a potencijalno i učenju emuliranja sustava koji ne spadaju u *LTI* kategoriju. Mislim da imam puno prostora za daljnje pisanje i razradu ovog rada što se nadam da i hoće prerasti u diplomski rad uz puno učenja.

7. Literatura

Slike 4.1, 4.2, 4.3 i 4.4 - https://humbuckersoup.com/pedals/soft-clipping-vs-hard-clipping-difference/ [15. rujna 2020.]

Slika 5 - http://zone.ni.com/reference/en-XX/help/372058U-01/nirfsa/total_harmonic_distortion/ [15. rujna 2020.]

Slike 6.1 i 6.2 - https://www.soundonsound.com/sound-advice/q-what-release-settings-should-i-use-limiter [15. rujna 2020.]

Slika 10 - https://www.masteringthemix.com/blogs/learn/113159685-sample-rates-and-bit-depth-in-a-nutshell [15. rujna 2020.]

[1] Davies, P. (2018) *The secret history of guitar effects pedals* [Online]. Dostupno na: https://www.soundaffects.com/blog/2018/04/the-secret-history-of-guitar-effects-pedals/ [15. rujna 2020.]

[2] White, F., Dick, T. (2016) *Analog compressor modeling* [Online]. Dostupno na: http://www2.ece.rochester.edu/~zduan/teaching/ece472/projects/2016/White_Dick_paper.pdf [15. rujna 2020.]

[3] Lambert, M. (2010) *Plug-in modeling* [Online]. Dostupno na: https://www.soundonsound.com/techniques/plug-modelling [15. rujna 2020.]

[4] Waves.com (2019) *How Waves' modeling captures analog magic in a digital world* [Online]. Dostupno na: https://www.waves.com/how-waves-modeling-captures-analog-magic [15. rujna 2020.]

[5] Brilliant.org *Linear time invariant systems* [Online]. Dostupno na: https://brilliant.org/wiki/linear-time-invariant-systems/ [15. rujna 2020.]

[6] Soundwoofer.se (2018) *How to create/record an impulse response* [Online]. Dostupno na: https://www.soundwoofer.se/blog/2018/03/29/how-to-create-record-an-impulse-response/ [15. rujna 2020.]

[7] Apple.com *Impulse Response Utility user manual* [Online]. Dostupno na: https://help.apple.com/impulseresponseutility/mac/1.0.3/en/impulseresponseutility/usermanual/Impulse%20Response%20Utility%20User%20Manual%20(en).pdf [15. rujna 2020.]

[8] Wikipedia.org (2020) *JUCE* [Online]. Dostupno na: https://en.wikipedia.org/wiki/JUCE [15. rujna 2020.]

[9] JUCE *Getting started with the Projucer* [Online]. Dostupno na: https://docs.juce.com/master/tutorial_new_projucer_project.html [15. rujna 2020.]

[10] Brown, G. (2019) *Digital audio basics: sample rate and bit depth* [Online]. Dostupno na: https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html [15. rujna 2020.]

[11] Burk, P., Polansky, L., Repetto, D., Roberts, M., Rockmore, D. (1998) *Music and Computers* [Online]. Dostupno na: http://musicandcomputersbook.com/ [19. rujna 2020.]

[12] Wikipedia.org (2020) *44,100 Hz* [Online]. Dostupno na: https://en.wikipedia.org/wiki/44,100_Hz [15. rujna 2020.]

[13] Hackaudio.com *Soft clipping* [Online]. Dostupno na: https://www.hackaudio.com/digital-signal-processing/distortion-effects/soft-clipping/ [15. rujna 2020.]

Reon Fourie (2017) Audio Processing Tutorial: How To Create an AWESOME Distortion VST/AU Plugin In C++ (JUCE Framework). Dostupno na: https://www.youtube.com/watch?v=iNCR5flSuDs [15. rujna 2020.]

8. Prilozi

8.1. PluginProcessor.cpp

```
#include "PluginProcessor.h"
#include "PluginEditor.h"

DistorzijaAudioProcessor::DistorzijaAudioProcessor()
{
    state = new juce::AudioProcessorValueTreeState(*this, nullptr);

    state->createAndAddParameter("drive", "Drive", "Drive",
juce::NormalisableRange<float>(0.0, 3000.0, 0.0001), 1.0, nullptr,
nullptr);
    state->createAndAddParameter("blend", "Blend", "Blend",
juce::NormalisableRange<float>(0.0, 1.0, 0.0001), 1.0, nullptr, nullptr);
    state->createAndAddParameter("volume", "Volume", "Volume",
juce::NormalisableRange<float>(0.0, 3.0, 0.0001), 0.0, nullptr, nullptr);

    state->state = juce::ValueTree("drive");
    state->state = juce::ValueTree("blend");
    state->state = juce::ValueTree("volume");

void DistorzijaAudioProcessor::processBlock (juce::AudioBuffer<float>&
buffer, juce::MidiBuffer& midiMessages)
{
    juce::ScopedNoDenormals noDenormals;
    auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();

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

    float drive = *state->getRawParameterValue("drive");
    float blend = *state->getRawParameterValue("blend");
    float volume = *state->getRawParameterValue("volume");
    float pi = 3.14159;

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

```
        for(int sample = 0; sample < buffer.getNumSamples(); sample++)
        {
            float cleanSig = *channelData;
            *channelData = (((((2.0/pi)*atan(drive *
*channelData))*blend)+(cleanSig*(1.0-blend)))/2.0)*volume;
            channelData++;
        }
    }
}

juce::AudioProcessorValueTreeState& DistorzijaAudioProcessor::getState()
{
    return *state;
}

void DistorzijaAudioProcessor::getStateInformation (juce::MemoryBlock&
destData)
{
    juce::MemoryOutputStream stream(destData, false);
    state->state.writeToStream(stream);
}

void DistorzijaAudioProcessor::setStateInformation (const void* data, int
sizeInBytes)
{
    juce::ValueTree tree = juce::ValueTree::readFromData(data,
sizeInBytes);

    if(tree.isValid())
    {
        state->state = tree;
    }
}

juce::AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new DistorzijaAudioProcessor();
}
```