Programsko rješenje generičkog alata za sintezu zvuka

Košmerl, Daniel

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1463

PROGRAMSKO RJEŠENJE GENERIČKOG ALATA ZA SINTEZU ZVUKA

Daniel Košmerl

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1463

PROGRAMSKO RJEŠENJE GENERIČKOG ALATA ZA SINTEZU ZVUKA

Daniel Košmerl

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1463

Pristupnik:	**Daniel Košmerl (0036540037)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Marko Horvat

Zadatak: **Programsko rješenje generičkog alata za sintezu zvuka**

Opis zadatka:

Sinteza zvuka jedan je od načina generiranja zvuka uporabom analognog ili digitalnog sklopovlja te raznovrsnih programskih rješenja. Ovaj se proces široko primjenjuje u suvremenoj glazbenoj produkciji jer je njime moguće dobiti zvuk kakav nije moguće proizvesti stvarnim izvorima u obliku glazbenih instrumenata i/ili izvođača vokalista. Stoga u mnogim žanrovima suvremene glazbe na ovaj način dobiveni zvukovi upotpunjuju i oplemenjuju zvuk dobiven tradicionalnim putem, a u nekima se sintetizirani zvuk upotrebljava kao jedini način dobivanja zvučnog sadržaja. Zvučna sinteza svoju primjenu nalazi u brojnim drugim područjima kao što su sinteza govora iz teksta, modeliranje stvarnih izvora zvuka i sl. U ovome je radu potrebno prikazati i izraditi programsko rješenje generičkog alata za sintezu zvuka. U teorijskom je dijelu potrebno opisati osnovne karakteristike pojedinih dijelova sintetizatora, što uključuje oscilatore za generiranje osnovnih valih oblika kao što su sinusni, pravokutni i pilasti, metode sinteze zvuka koje će biti implementirane u sintetizatoru kao što su aditivna, subtraktivna i sinteza frekvencijskom modulacijom, te postupke oblikovanja signala u vidu uobičajenih metoda frekvencijske i dinamičke obrade audiosignala. U programskom je rješenju potrebno implementirati sve navedeno te ostvariti mogućnost učitavanja i spremanja postavki sintetizatora, kao i sučelje kojim će sintetizator moći primati podatke putem MIDI protokola.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

Cilj projekta bio je naučiti proizvesti zvuk u digitalnom okruženju – osobnom računalu, te napraviti alat za demonstraciju naučenog. To uključuje generiranje i spremanje signala u računalu, obrada signala i interakcija s operacijskim sustavom računala kako bi se osiguralo konzistentno slanje i primanje signala u stvarnom vremenu.

Slični programi se često koriste u produkciji glazbe i dizajnu zvuka, otkad su procesori na osobnim računalima postali dovoljno moćni za obradu signala u stvarnom vremenu bez potrebe za specijaliziranim, skupim sklopovljem za obradu signala.

Takav alat mora podržavati reagiranje na korisničke ulaze koji mogu uključivati ulaze preko MIDI protokola, te mijenjanje, dodavanje i brisanje dijelova lanca obrade signala preko korisničkog sučelja.

Svrha tog alata je pružanje korisniku što više slobode pri izradi i dizajnu zvuka, time što omogućuje izgradnju bilo kakvog lanca obrade signala (koristeći komponente definirane u alatu).

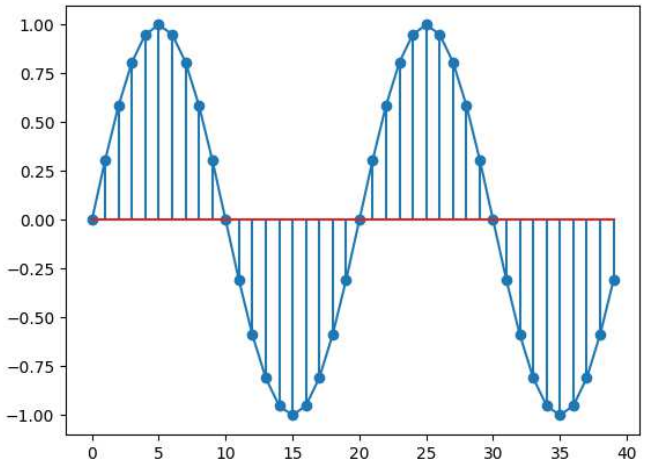
1. Digitalni signal i zvuk

1.1. Pohranjivanje zvuka u računalu

Zvuk koji čuje ljudsko uho opisuje se zvučnim tlakom koji predstavlja promjenu atmosferskog tlaka u vremenu u području frekvencija i amplituda te promjene koje ljudsko uho može registrirati. Ako želimo takvu funkciju pohraniti u računalu, moramo koristiti aproksimaciju – periodično uzimanje uzoraka.

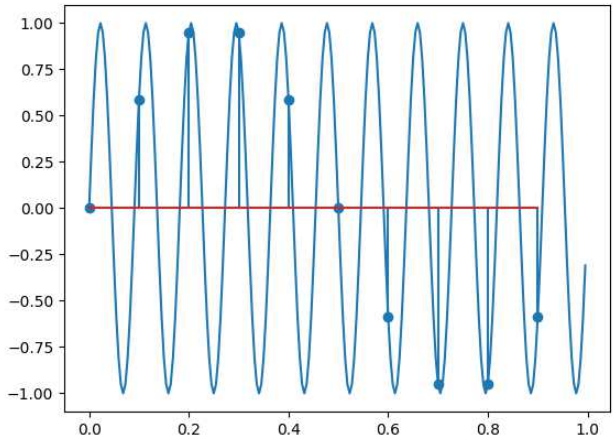
1.1.1. Uzorkovanje

Vremenski interval između pojedinih uzetih uzoraka nazivamo periodom uzorkovanja, a recipročnu vrijednost toga perioda frekvencijom uzorkovanja f_s. Na slici Sl. 1.1 pokazan je primjer uzorkovanja sinusoidnog signala. Dobiveni uzorci mogu se pohraniti u računalu kao niz binarnih brojeva.



Sl. 1.1 Uzorkovanje signala

Naravno, iz ovako uzorkovanog signala ne možemo rekonstruirati originalni signal bez gubitka informacije. Ukoliko osnovni signal koji se uzorkuje ima određeni spektralni sadržaj na frekvencijama većim od polovice frekvencije uzorkovanja, za sadržaj na tim frekvencijama te pri kasnijoj digitalno-analognoj pretvorbi događa se višeznačnost (engl. aliasing), odnosno bit će rekonstruiran sadržaj kojeg nije bilo u izvornom signalu, kako je prikazano na slici Sl. 1.2.



Sl. 1.2 Primjer višeznačnosti signala (engl. aliasing) – uz uzorkovanje sinusoidnog signala frekvencije f nedovoljno visokom frekvencijom uzorkovanja f_s, pri digitalno-analognoj pretvorbi umjesto izvornog sinusoidnog signala bit će rekonstruiran njegov alias frekvencije $f_a = f_s - f$

Činjenicu da ne možemo točno rekonstruirati frekvencije veće od polovice frekvencije očitavanja govori nam Nyquistov teorem. Kako bi se izbjegla pojava izobličenja uzrokovanog višeznačnošću, za svaki je signal koji se uzorkuje potrebno odabrati odgovarajuću frekvenciju uzorkovanja, a spektralni sadržaj signala potrebno je ograničiti niskopropusnim filtrom.

Ako znamo da je najveća frekvencija koju ljudsko uho može čuti oko 20 kHz, možemo odabrati frekvenciju očitavanja veću od 40 kHz i nećemo izgubiti nikakvu bitnu informaciju. U praksi su najčešće korištene frekvencije očitavanja 44,1kHz (standard za pohranjivanje na CD) i 48 kHz.

1.1.2. Kvantizacija

Budući da funkcija promjene zvučnog tlaka u vremenu teoretski može poprimiti bilo koju vrijednost u skupu realnih brojeva, i vrijednosti uzetih uzoraka također su realni brojevi koje nije moguće pohraniti u računalu jer bi za njihov prikaz bio potreban beskonačan broj bitova. Moramo koristiti konačan broj bitova za spremanje pojedinog uzorka, što nam daje konačan broj vrijednosti koje uzorak može poprimiti. To znači da će neki realni brojevi morati poprimiti istu diskretnu vrijednost, te gubimo preciznost. Takvo preslikavanje realnih brojeva u diskretne vrijednosti nazivamo kvantizacija.

Cijeli raspon vrijednost koje može poprimiti analogni signal potrebno je pokriti određenim brojem diskretnih kvantizacijskih razina. Broj raspoloživih razina ovisi o broju bitova te uz N bitova na raspolaganju imamo 2^N kvantizacijskih razina. Svođenjem amplitude uzoraka

na određen broj diskretnih vrijednosti predstavljenih kvantizacijskim razinama čini se određena pogreška kao razlika stvarne amplitude uzorka i vrijednosti najbliže kvantizacijske razine. Ova se pogreška manifestira kao šum kvantizacije. Odnos signal-šum između uzorkovanog i kvantiziranog signala te šuma kvantizacije može se ugrubo izračunati iz broja bitova *N* kao 6**N* [dB].

U praksi se pri kvantizaciji audiosignala za potrebe glazbene i govorne produkcije koristi 16, 24 ili 32 bita, što u sva tri slučaja daje dovoljno veliki odnos signal-šum pri kojem je šum kvantizacije nečujan.

1.1.3. Zahtjevi na memoriju pri pohrani digitalnog audiosignala

Pretpostavimo da želimo pohraniti jednu sekundu audiosignala kvantiziranog sa 16 bitova po uzorku te uz frekvenciju uzorkovanja od 44,1 kHz. U današnjoj se glazbenoj produkciji široko upotrebljava stereo format, što znači da je potrebno pohraniti dva zvučna kanala – jedan za lijevi, drugi za desni zvučnik, kako bismo dobili stereo efekt zvuka. Broj bitova korištenih za jednu sekundu audiosignala računamo jednadžbom (1).

$$16\frac{bit}{uzorak} * 44100\frac{uzorak}{sekunda} * 2(\#kanala) = 1411200\frac{bit}{sekunda} \quad (1)$$

Ovo je standard pohranjivanja zvuka na CD [1].

Dodatno se čistom audiosignalu dodaju metapodaci te dodatni bitovi uz pomoć kojih se otkrivaju i ispravljaju pogreške, a signal se kodira tzv. kanalnim kodovima prilagođenim određenom načinu prijenosa ili pohrane signala.

1.2. Generiranje digitalnog signala

Ovim postupkom možemo uzorkovati bilo koju matematičku funkciju, što nam omogućuje da generiramo zvukove kakvi ne mogu nastati u prirodi. Najzanimljivije su nam periodične funkcije jer one modeliraju titranje zraka koje čini zvuk. Uređaje, uključujući i programske objekte, koji generiraju periodički signal nazivamo oscilatori. Osnovni valni oblici koje generiramo oscilatorima su sinusoidni, pravokutni, trokutasti i pilasti.

Sinusoidni val računa se iz formule $x(t) = A\sin(2\pi ft + \varphi)$, gdje *A* predstavlja amplitudu vala, *f* frekvenciju, a φ početnu fazu vala. Uzimanjem uzoraka te funkcije frekvencijom očitavanja dobivamo diskretni signal $x[n]$.

Kvadratni val dobivamo iz formule $x(t) = A(-1)^{\lfloor 2ft \rfloor}$.

Trokutasti val: $x(t) = 2A\left|2\left(ft - \left\lfloor ft + \frac{1}{2}\right\rfloor\right)\right| - 1$.

Pilasti val: $x(t) = 2A\left(ft - \left\lfloor ft + \frac{1}{2}\right\rfloor\right)$.

1.3. Sinteza zvuka

Jedan generirani signal nije pretjerano zanimljiv sam po sebi, no možemo ga koristiti u raznim postupcima sinteze zvuka.

1.3.1. Aditivna sinteza

Uzimanjem više oscilatora s različitim valnim oblicima, frekvencijama i glasnoćama, te sumiranjem njihovih izlaza možemo dobiti nove, bogatije zvukove.

Sumiranje signala je jednostavna operacija – pojedini uzorak sumiranog signala je zbroj tog uzorka svih signala koji se sumiraju, odnosno $x[n] = a[n] + b[n]$. Pri tome treba paziti da zbroj ne bude veći od maksimalne vrijednosti koju uzorak može poprimiti.

Pretpostavimo da sumiramo dva sinusoidna vala iste amplitude i frekvencije, no jedan je pomaknut u fazi od drugog za π. Sumirani signal $x[n]$ biti će nula za svaki $n \in N$. Kažemo da su signali u protu fazi i da se poništavaju. U tom slučaju korisnije nam je oduzeti signale, odnosno pomnožiti jednog s -1 i onda zbrojiti.

1.3.2. Subtraktivna sinteza

Za razliku od aditivne sinteze, subtraktivna sinteze koristi filtre – uređaje koji uklanjaju neželjene frekvencije. Time iz već bogatih zvukova ili šumova možemo dobiti bolje kontrolirani zvuk.

Šumom nazivamo signal nastao slučajnim generiranjem. U slučaju digitalnog signala, možemo ga dobiti tako da svaki uzorak signala bude slučajno generiran broj.

Takav signal nije periodičan, a spektralna analiza pokazati će da sadrži sve frekvencije spektra. Iz toga možemo filtrima izbaciti ili reducirati neželjene frekvencije dok ne dobijemo zvuk kakav nam odgovara.

1.3.3. Frekvencijska modulacija

Zanimljivi zvukovi nastaju kada se frekvencija signala mijenja u ovisnosti o drugom signalu. Drugim riječima, jedan signal modulira frekvenciju drugog signala. Izlazni signal može se opisati jednadžbom (2).

$$y(t) = A\sin(2\pi f t + A_m \sin 2\pi f_m t) \tag{2}$$

1.4. Obrada audiosignala

Nakon generiranja signala moguće je primijeniti razne postupke obrade signala u svrhu dodatnog oblikovanja dobivenog zvuka. Uzmimo neki generirani digitalni signal $x[n]$. Nad njime možemo primijeniti neku operaciju čiji izlaz daje obrađeni signal $y[n]$. Takav sustav predstavljamo diferencijskom jednadžbom (3).

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] - a_1 y[n-1] - \cdots - a_M y[n-M] \tag{3}$$

Ova jednadžba nam govori da pojedini uzorak izlaznog signala može ovisiti o uzorcima i ulaznog i izlaznog signala tog sustava, no smije ovisiti samo o ulaznim i izlaznim uzorcima koji su prethodili trenutnome i o trenutnom uzorku na ulazu. Koeficijenti b_0, …, b_N i a_0, …, a_N definiraju sustav. [2]

1.4.1. Filtriranje

Koeficijenti diferencijske jednadžbe sustava mogu se odabrati tako da sustav ima neku željenu frekvencijsku karakteristiku, odnosno utjecaj na spektralne komponente signala. Time možemo definirati filtre koji uklanjaju ili prigušuju neke komponente spektra. Dijelimo ih na visokopropusne (propuštaju frekvencije iznad neke granične frekvencije), niskopropusne (propuštaju frekvencije ispod granične), pojasnopropusne (propuštaju frekvencije između donje i gornje granične frekvencije) i pojasne brane (uklanjaju frekvencije između donje i gornje granične).

1.4.2. Dinamička obrada signala

Stvarni zvukovi mijenjaju se kroz vrijeme. Često počinju glasno, te se s vremenom stišavaju prema tišini. Za modeliranje ovakvog ponašanje koristimo ovojnice – vremenski ovisne

sustave koji mijenjaju glasnoću signala, odnosno množe ga s promjenjivim koeficijentom pojačanja.

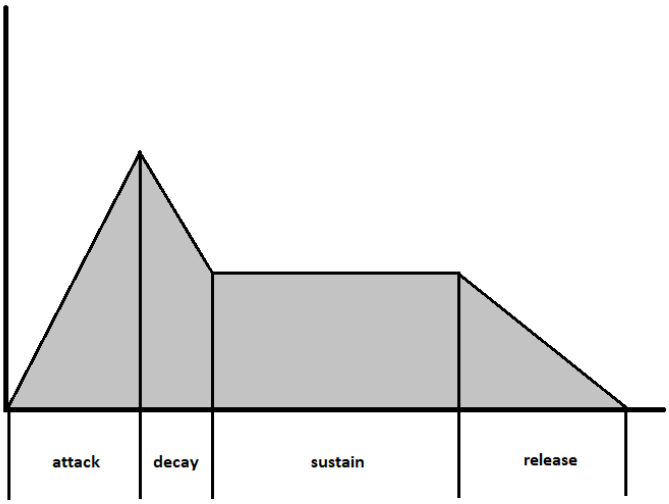
ADSR je primjer takve ovojnice. Skraćenica dolazi od engleskih imena varijabli sustava: attack, decay, sustain i release. Na slici Sl. 1.3 prikazan je graf pojačanja ovojnice kroz vrijeme.

Attack je vrijeme potrebno da pojačanje postigne maksimalnu vrijednost. U vremenu $t = 0$ pojačanje ovojnice uvijek je jednako nuli, te do vremena određeno attack varijablom t_A postupno raste (najčešće linearno).

Sustain je vrijednost pojačanja do kojeg ovojnica dolazi nakon vremena attack i decay t_A+t_D.

Decay je vrijeme potrebno da se pojačanje spusti od maksimalne vrijednosti do vrijednosti određene sustain varijablom. U vremenu $t = t_A$ pojačanje se smanjuje (najčešće linearno) do sustain vrijednosti s, do koje dolazi u vremenu $t = t_A + t_D$.

Release je vrijeme potrebno da se pojačanje sa sustain vrijednosti s spusti u nulu. Sustav mora prvo primiti poruku da je signal otpušten. Nakon što je poruka primljena, pojačanje se (najčešće linearno) spušta do nule.



Sl. 1.3 ADSR ovojnica

ADSR je poseban slučaj jer ovisi o primanju poruka za početak i kraj signala. Pojačanje s kojim množimo signal može biti izlaz bilo koje funkcije ovisne o vremenu i drugim varijablama i porukama.

2. VST sučelje

Nakon što smo u memoriju spremili signal, moramo ga nekako poslati na zvučnike da bismo ga čuli. Zvuči jednostavno, no različita računala i operacijski sustavi obavljaju ovu operaciju na različite načine. Umjesto da programiramo različit kod ovisno o okruženju, bolje rješenje je korištenje već prevedenih programa koji mogu pokrenuti naš alat kao potprogram i obavljaju komunikaciju s operacijskim sustavom za nas. To znači da moramo napraviti samo jednu binarnu datoteku i osigurati pravilnu komunikaciju sa sučeljem domaćinskog programa.

2.1. Povezivanje potprograma

Svi operacijski sustavi na neki način omogućuju dinamičko povezivanje binarnih datoteka kako bi se funkcije iz jedne datoteke (biblioteke) mogle pozvati iz druge bez potrebe za prevođenjem u jednu datoteku.

Na operacijskom sustavu Windows, binarne datoteke bez ulaza namijenjene za pozivanje iz drugih datoteka nazivamo DLL (*Dynamic-link library*) i imaju nastavak .dll. Operacijski sustav nudi funkcije za povezivanje biblioteka: `LoadLibrary()` za učitavanje DLL-a u adresni prostor procesa i `GetProcAddress()` za uzimanje pokazivača na funkciju. `GetProcAddress()` kao parametar uzima ime tražene funkcije, što znači da proces mora unaprijed znati imena funkcija koje će koristiti. Dakle, za pravilno korištenje biblioteke moraju unaprijed biti definirana sva imena funkcija dostupnih za pozivanje – aplikacijsko sučelje ili API (*Applications programming interface*) [3].

2.2. VST tehnologija

1996. godine tvrtka Steinberg Media Technologies razvila je programsko sučelje VST (*Virtual studio technology*) za rješavanje upravo ovakvog problema – sučelje koje povezuje domaćinski program DAW (*Digital audio workstation*) preveden za neko okruženje i .vst datoteku koju dinamički učitava. Datoteka .vst može se dinamički povezati na svakom operacijskom sustavu jer se ponaša kao .dll na Windows OS-u, Mach-O Bundle na Mac OS X i package na Linuxu [4].

VST sučelje definira sve potrebne funkcije za komunikaciju s operacijskim sustavom uključujući inicijalizaciju i uništavanje programa, procesiranje signala, čitanje namještene frekvencije uzorkovanja, stvaranje grafičkog sučelja i primanje ulaza iz grafičkog sučelja, audio kanala ili MIDI protokola. Točne komponente sučelja mogu se pronaći u Steinbergovoj službenoj dokumentaciji VST sučelja [4].

3. MIDI protokol

MIDI (*Musical Instrument Digital Interface*) označava standardizirani industrijski protokol za komunikaciju i sinkronizaciju između elektroničkih glazbenih instrumenata, računala i ostale elektroničke opreme.

Svrha MIDI protokola je slanje MIDI poruka iz instrumenta u stvarnom vremenu, a na programeru je da implementira reakcije na te poruke. Budući da projekt koristi VST sučelje, MIDI poruke, ako su primljene, automatski dolaze kroz ulazni kanal sučelja.

3.1. Struktura MIDI protokola

Pojedina MIDI poruka je niz okteta (8 bitova), čiji prvi oktet je statusni i određuje vrstu poruke, dok su ostali okteti podatkovni i sadrže informaciju poruke. Dijele se na dvije podvrste: kanalne i sistemske poruke [5].

3.1.1. Kanalne MIDI poruke

MIDI protokol sastoji se od 16 kanala što znači da može provoditi MIDI poruke o 16 instrumenata istovremeno. Zbog toga poruke koje se odnose samo na jedan kanal moraju u statusnom oktetu odrediti na koji kanal se odnose. Statusni oktet je u tom slučaju podijeljen na dva dijela od 4 bita. Prva 4 određuju vrstu poruke, a druga 4 čine 4-bitni broj od 0 do 15 koji predstavlja indeks kanala na koji se poruka odnosi.

Preslikavanje bitova statusnog okteta u vrstu poruke vidimo u Tablica 3.1.

Gornja 4 bita statusnog okteta	Donja 4 bita statusnog okteta	Osnovna status poruka	Broj dodatnih podatkovnih okteta	Kategorija poruke
1000 = 8	0000=0 ÷ 1111=F broj (logičkog) MIDI kanala na koji se poruka šalje	NOTE OFF	2	KANALNA
1001 = 9		NOTE ON	2	
1010 = A		AFTERTOUCH	2	
1011 = B		CONTROL CHANGE	2 ÷ 3	
1100 = C		PROGRAM (PATCH) CHANGE	2	
1101 = D		CHANNEL PRESSURE	1	
1110 = E		PITCH WHEEL	2 ÷ 3	
1111 = F	Poruka podklase	SYSTEM		SISTEMSKA

Tablica 3.1 Kanalne MIDI poruke

3.1.2. Sistemske MIDI poruke

Sistemske poruke su one koje se ne odnose na specifičan kanal. Uključuju poruke za sinkronizaciju, naredbe za odabir pjesme i slično.

U ovom projektu nikakve sistemske MIDI poruke nemaju implementiranu reakciju.

3.2. Reagiranje na poruke

Najvažnije poruke su *Note On* i *Note Off* jer se odnose na specifičnu tipku na klavijaturi i nose informaciju kada je koja tipka pritisnuta odnosno otpuštena, te kojom jačinom.

Note On poruka ima oblik:

- Statusni oktet: `1011 CCCC`
- Podatkovni oktet 1: `0PPP PPPP`
- Podatkovni oktet 2: `0VVV VVVV`

Bitovi `CCCC` predstavljaju indeks kanala (od 0 do 15), `PPP PPPP` predstavlja notu koja je pritisnuta (od 0 do 127), a `VVV VVVV` predstavlja jačinu kojom je pritisnuta (od 0 do 127).

Note Off poruka istog je oblika osim u statusnom oktetu koji je oblika `1000 CCCC`.

Iz pritisnute note možemo dobiti njenu frekvenciju koristeći formulu (4).

$$f(n) = 440 * 2^{(n-69)/12} \tag{4}$$

Program sada može, kada primi *Note On* poruku, aktivirati oscilator kakav je definiran u prvom poglavlju, s frekvencijom dobivenom iz pritisnute note i pomnožen s jačinom pritiska (podijeljenu s 127 kako bi signal ostao između 0 i 1). Taj oscilator nastavlja titrati dok ne primi *Note Off* poruku za tu notu.

4. Programski jezik Rust

Većina programa i sučelja za sintezu i obradu zvuka pisana je koristeći jezike C i C++. Samo sučelje VST pisano je u tim jezicima, a najpopularniji način izrade VST-ova je C++ okruženje JUCE koje definira mnoge korisne funkcije za lakšu izradu VST-a.

To ne znači da je izrada VST-ova ograničena samo na C++. Jedan od ciljeva ovog projekta je pokazati da se ovakav program može napraviti neovisno o korištenim alatima, jezicima i okruženju.

Jezik Rust odabran je zbog dobrih performansi, što je važno za sustave koji reagiraju u stvarnom vremenu, olakšano upravljanje memorijom i aktivna zajednica audio programera koji redovito razvijaju nova okruženja i alate za audio programiranje.

4.1. Okruženje nih-plug

Glavna funkcija ovog okruženja je lakša izrada VST programa, slično kao JUCE okruženje u C++-u. U okruženju je definiran `Plugin` trait (slično sučelju u objektno orijentiranim jezicima) koji treba implementirati za glavni objekt programa jer će se funkcije definirane u sučelju povezati na funkcije VST sučelja. Specifično, sučelje nudi funkcije za inicijalizaciju, uništavanje, stvaranje grafičkog sučelja i, najvažnije, funkciju za obradu jednog bloka uzoraka. Ta funkcija `process()` prima kao parametar sa VST sučelja ulazni blok uzoraka (kod sintetizatora koji nemaju ulaze, ovaj blok biti će ispunjen nepoznatim vrijednostima) i treba ga popuniti vrijednostima koje će se poslati na izlaz sučelja.

4.2. Biblioteka FunDSP

Ova biblioteka nudi mnogo korisnih funkcija vezanih uz obradu signala. Svaka od tih funkcija može se instancirati kao objekt koji implementira `AudioNode`, što nam daje generički objekt s definiranim ulazima i izlazima. Na primjer, funkcija `sine()` stvara objekt tipa `Sine` koji implementira `AudioNode`, te ima jedan ulaz (frekvenciju) i jedan izlaz (izlazni signal). Moguće je i definirati nove strukture i implementirati `AudioNode` za njih, ulaze i izlaze, te ih povezivati s postojećim `AudioNode` objektima.

Biblioteka također nudi strukturu mreža – `Net32`. Mreža se može modificirati tijekom izvođenja programa, dodavanjem, brisanjem, povezivanjem i modificiranjem `AudioUnit` elemenata. `AudioUnit` je dinamička verzija `AudioNode` objekta koja se alocira na gomili, a ulazi i izlazi ne moraju biti poznati za vrijeme prevođenja nego tek pri stvaranju objekta.

Procesiranje signala u mreži može se odvojiti na posebnu dretvu. Time je osigurano da promjene u mreži tokom izvođenja ne stvaraju nagle prekide u signalu jer dretva neće primiti promjenu dok ne završi procesiranje trenutnog bloka.

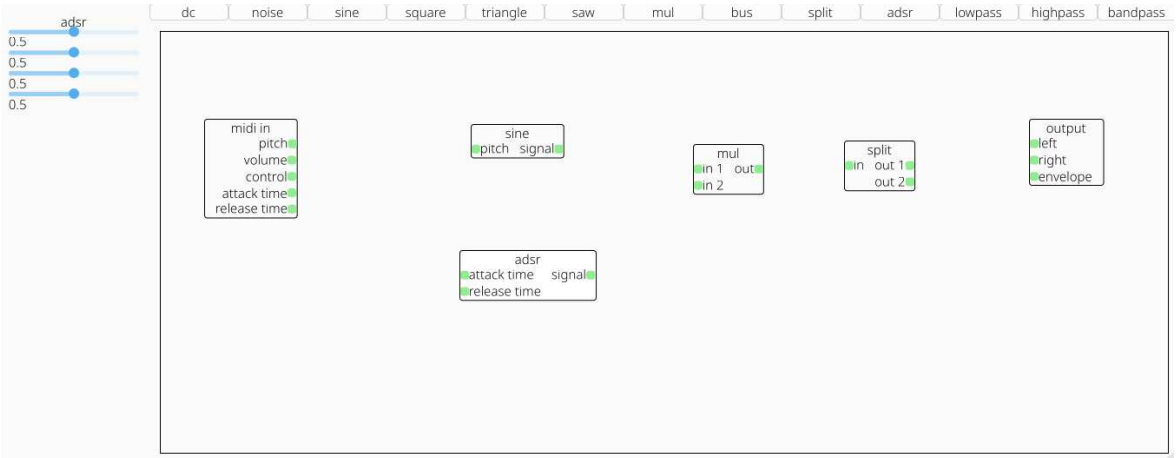
4.3. Biblioteka Vizia

Ovo je biblioteka za izradu grafičkog sučelja. Glavna prednost ove biblioteke je jednostavna integracija s nih-plug okruženjem.

Izradi grafičkog sučelja posvećeno je najmanje vremena jer je fokus projekta na sintezi i obradi zvuka. Ipak, implementirane su osnovne kontrole za modificiranje mreže stvorene bibliotekom FunDSP.

5. Opis gotovog alata

Alat preko grafičkog sučelja na slici Sl. 5.1 definira i modificira mrežu obrade signala koja tokom cijelog izvođenja programa generira izlazni signal koji odlazi na izlaz definiran VST sučeljem, te u stvarnom vremenu reagira na korisničke ulaze preko grafičkog sučelja i MIDI protokola.

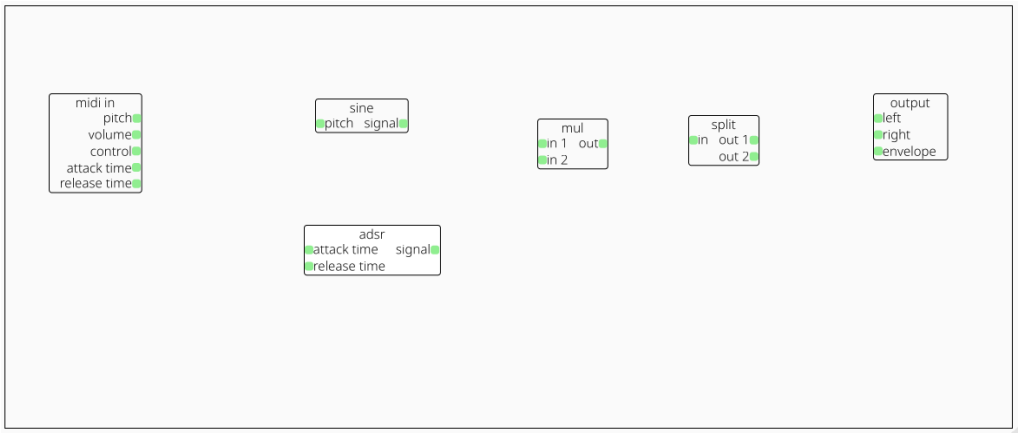


Sl. 5.1 Grafičko sučelje programa

5.1. Elementi grafičkog sučelja

5.1.1. Prostor za uređivanje grafa

Središnji dio grafičkog sučelja zauzima prostor za uređivanje grafa, prikazan na slici Sl. 5.2.



Sl. 5.2 Prostor za uređivanje grafa

Svaki element ovog prostora predstavlja čvor u mreži obrade signala. Čvorovi imaju ulazne kanale s lijeve i izlazne kanale s desne strane čvora. Zeleni krug pored kanala služi za

povezivanje tog kanala s drugim. Pritiskom na njega kanal se odabire, osim ako je neki kanal već odabran, u kojem slučaju se ti kanali povezuju, što znači da se izlaz iz izlaznog kanala prosljeđuje u ulaz ulaznog kanala. Nije moguće povezati dva ulazna niti dva izlazna kanala.

Graf (a time i prostor) uvijek sadrži točno jedan čvor za primanje MIDI poruka („midi in“) i jedan čvor za slanje izlaznog signala na VST sučelje („output“). Ostali čvorovi mogu se dinamički stvoriti odabirom u izborniku i pritiskom na prazno mjesto u prostoru za uređivanje, na kojem će se taj čvor stvoriti.

5.1.2. Izbornik čvorova

Na vrhu grafičkog sučelja nalazi se izbornik čvorova prikazan na slici Sl. 5.3.

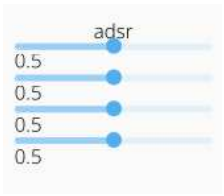
Sl. 5.3 Izbornik čvorova

Pritiskom na element izbornika on se odabire za stvaranje, što se radi pritiskom na prazno mjesto u prostoru za uređivanje.

5.1.3. Lista parametara

Prikaz s lijeve strane sučelja na vrhu prikazuje ime odabranog elementa, što može biti čvor u grafu, čvor odabran za stvaranje ili kanal odabran za spajanje. U slučaju da je odabran čvor grafa, ispod imena pokazuju se parametri čvora (ako postoje). Ti parametri mogu se modificirati, a mreža odmah reagira na te promjene.

Primjer na slici Sl. 5.4 pokazuje se pritiskom na ADSR čvor u grafu.



Sl. 5.4 Parametri ADSR čvora

Zaključak

U sklopu ovog rada stvoren je alat koji obavlja zadaću jednostavnog sintetizatora zvuka. Implementirane su sve potrebne komponente: sučelje za interakciju s korisnikom, reagiranje na poruke iz priključenog MIDI instrumenta i, najvažnije, generiranje i obrada audiosignala. Velik naglasak projekta bio je upravo na fleksibilnosti mreže obrade signala, koja se oslanja na funkcije iz biblioteke FunDSP. To omogućava jednostavno proširivanje alata novim komponentama i čvorovima mreže obrade signala, čak i implementaciju kompozicije čvorova (podmreža) za bolju preglednost i jednostavnije korištenje velikih mreža.

Alat se još može proširiti dodavanjem mogućnosti spremanja i učitavanja konfiguracija mreža sa računala kako se ne bi morala ponovno izgraditi mreža pri svakom korištenju alata. Nažalost, ova funkcionalnost nije implementirana jer bi zahtijevala razvoj posebnog formata za spremanje konfiguracije mreže, te alata za čitanje konfiguracijske datoteke.

Još jedna mogućnost za proširivanje alata bio bi prostor za analizu generiranog signala. Postoje moćne biblioteke za spektralnu analizu signala u Rust-u kao što su RustFFT i RealFFT. Ova komponenta nije napravljena jer bi grafički prikaz analiziranog signala u stvarnom vremenu bio zahtjevan za implementirati.

Ukratko, ovaj alat može biti koristan ako korisnik treba brzo i jednostavno dizajnirati specifičan zvuk koristeći neke od implementiranih komponenti. Može se koristiti kao instrument u glazbenoj produkciji ili za dizajn generičkih zvučnih efekata.

Literatura

[1] Reiss, J. D., McPherson, A. P. *Audio Effects: Theory, Implementation and Application*. 1. izdanje.

[2] Zölzer, U., *DAFX: Digital Audio Effects*. 2. izdanje.

[3] Pirkle, W. C., *Designing Audio Effect Plug-Ins in C++*. 1. izdanje.

[4] Steinberg Media Technologies, *VST3 API Documentation* https://steinbergmedia.github.io/vst3_doc/vstsdk/index.html; pristupljeno 11. rujna 2024.

[5] Vandenneucker, D., *MIDI Tutorial for programmers* https://www.cs.cmu.edu/~music/cmsip/readings/MIDI%20tutorial%20for%20programmers.html; pristupljeno 11. rujna

Sažetak

U ovom radu opisan je postupak izrade generičkog alata za sintezu zvuka. Objašnjena je teorija iza sinteze zvuka, uključujući generiranje i osnovna obrada signala i načini pohrane signala u računalu. Korištena je tehnologija VST za interakciju između programa i operacijskog sustava i MIDI protokol za primanje ulaza preko vanjskih uređaja. Za samu implementaciju alata korišten je programski jezik Rust, okruženje nih-plug za interakciju s VST sučeljem, biblioteka FunDSP za sintezu i obradu signala i biblioteka Vizia za izradu grafičkog sučelja.

Summary

This paper describes the process of creating a generic tool for audio synthesis. It contains the theory behind sound synthesis including signal generation and processing and means of storing the signal in a computer. The tool uses VST technology to interact with the operating system and recieves inputs from external instruments using MIDI protocol. Iti s implemented using Rust programming language, nih-plug framework for interacting with the VST interface, FunDSP library for signal generation and processing and Vizia library to create the user interface.

Privitak

Instalacija programske podrške

Budući da je projekt pisan u programskom jeziku Rust, za prevođenje je potrebno instalirati kompajler za njega. Taj kompajler može se preuzeti sa službene stranice organizacije Rust: https://www.rust-lang.org/tools/install, gdje se također nalaze i upute za instalaciju.

Nakon instalacije kompajlera, projekt se gradi pomoću komande:

```
cargo xtask bundle synth --release
```

Nakon prevođenja, gotov program u .vst3 formatu trebao bi se nalaziti u datoteci:

```
./target/bundled
```