UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Kladnik

Aplikacija za tvorbo prostorskega zvoka z uporabo zbirke meritev na modelu glave

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Rozman Robert

Ljubljana 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00345/2012
Datum: 05.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ KLADNIK**

Naslov: **APLIKACIJA ZA TVORBO PROSTORSKEGA ZVOKA Z UPORABO ZBIRKE MERITEV NA MODELU GLAVE**

AN APPLICATION FOR GENERATION OF SPATIAL SOUND USING THE COLLECTION OF HEAD MODEL MEASUREMENTS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Prostorsko zaznavanje zvoka je ena najpomembnejših lastnosti človekovega sluha. Zato je ideja o uporabi te lastnosti na vseh področjih komunikacije človek-stroj zelo zanimiva. Razvijte aplikacijo za tvorbo prostorskega zvoka s pomočjo zbirke opravljenih meritev na umetnem modelu glave. Aplikacija naj bo zasnovana na uporabi javno dostopne zbirke meritev MIT-KEMAR in implementirana v programskem jeziku Python. Najprej ustvarite razvojno okolje, ki bo tudi prenosljivo med različnimi operacijskimi sistemi. Končna aplikacija naj omogoča enostavno pretvorbo posameznih zvokov v ustrezne prostorske oblike in njihovo sestavo v kompleksnejše prostorske zvočne scene. Aplikacija naj bo enostavno uporabna na Windows operacijskih sistemih.

Mentor: viš. pred. dr. Robert Rozman

Dekan: prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

Spodaj podpisani Matej Kladnik, z vpisno številko 63050143, sem avtor diplomskega dela z naslovom:

Aplikacija za tvorbo prostorskega zvoka z uporabo zbirke meritev na modelu glave

S svojim podpisom zagotavljam, da

- sem diplomsko nalogo izdelal samostojno pod mentorstvom viš. pred. dr. Roberta Rozmana
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronskega dela v zbirki »Dela FRI«

V Ljubljani, dne 15. marca 2013 Podpis avtorja:

Zahvala

Za mentorstvo in pomoč pri izdelavi diplomske naloge se zahvaljujem viš. pred. dr. Robertu Rozmanu. Posebna zahvala gre Jasni za pomoč pri prevajanju in lektoriranju. Prav tako zahvala tudi staršem za podporo pri študiju.

Hvala!

Kazalo

Povzetek

Namen diplomskega dela je bil razviti aplikacijo za tvorbo 3D zvoka. To je zvrst prostorskega zvoka, ki vzbuja občutek prostorskosti na podlagi sestave dveh ustrezno spremenjenih različic zvoka za vsako uho posebej. V tem se 3D zvok razlikuje od splošnega pojmovanja »prostorskega zvoka« (*surround sound*), ki je običajno ustvarjen s pomočjo virov zvoka na različnih mestih v prostoru. V diplomskem delu smo za implementacijo sistema uporabili na spletu javno dostopno zbirko MIT-KEMAR. Le-ta vsebuje odzive na enotin impulz merjene na umetnem modelu glave – HRIR, ki so potrebni za tvorbo 3D prostorskega zvoka.

V prvem delu je predstavljena tehnika merjenja in organizacija podatkov v zbirki. Sledi njena uporaba, in sicer najprej v enostavnem programu za numerično računanje in procesiranje signalov – Scilab-u. Napisali smo funkcijo za prikaz časovnega poteka odziva na enotin impulz za levo in desno uho iz zbirke. Primerjali smo odzive za različne lokacije izvora zvoka. Nato smo s pomočjo nove funkcije tvorili zvočne datoteke s 3D učinkom.

V osrednjem delu smo s pomočjo orodja Oracle VirtualBox ustvarili virtualni razvojni sistem, ki je prenosljiv in uporaben na različnih operacijskih sistemih. Pri razvoju aplikacije je bil uporabljen programski jezik Python, ki je že vsebovan v Linux operacijskem sistemu nameščenem na virtualnem sistemu. V programsko okolje so bili dodani še moduli za branje zvočnih datotek, delo z matrikami in posebnimi večdimenzionalnimi polji, risanje grafov ter dodaten nabor grafičnih gradnikov. Sledil je razvoj glavne aplikacije, ki poleg tvorbe posameznih 3D zvočnih datotek omogoča tudi njihovo sestavo v kompleksnejše zvočne scene. Predstavljene so tudi vse njene pomembnejše funkcije. Na koncu smo z uporabo modula Py2exe naredili še končno obliko aplikacije, ki teče brez nameščanja Python okolja na Windows operacijskih sistemih.

Ključne besede: prostorski zvok, 3D zvok, MIT-KEMAR, procesiranje signalov, virtualni sistem, Python

Abstract

The purpose of the thesis was to develop an application for the generation of 3D sound. 3D sound is a type of spatial sound that adds spatial impression by generating two, slightly different versions of the sound for both ears. Term *Surround sound* in contrary means "spatial sound" produced by transmission of sound waves from several spatially placed sources in the room.

In the thesis we developed application based on the online public library MIT-KEMAR. The library contains head related impulse responses (HRIRs) measured on artificial head model that are necessary for the generation of 3D spatial sound.

The first part of the thesis focuses on the measurement technique and data organization in the library. Use of data from the library is demonstrated in a simple program function, developed in Scilab – environment for numerical computation and signal processing. A function for the graphical demonstration of the impulse responses for left and right ear from the library was written. In addition, responses from different locations of the sound sources were compared. Afterwards, using a new function, 3D sound files were created.

The second part of the thesis shows creation of a virtual development system using software tool Oracle VirtualBox. Such system is highly portable and can be used on various operating systems. Main application was developed in Python programming language that is already included in Linux distribution installed on virtual system. Python modules for reading of sound files, matrices and multidimensional arrays operations, graph drawing and other graphical user interface elements were added. Later on, main application was implemented which in addition to generation of elementary 3D sounds also combines them into more complex sound scenes. Using Py2exe module, we produced final version of application that can run without installing Python environment on Windows operating systems.

Keywords: spatial sound, 3D sound, MIT-KEMAR, signal processing, virtual system, Python

Seznam uporabljenih kratic in simbolov

3D	*3-dimensional* (3-dimenzionalno)
AIFF	*audio interchange file format* (vrsta formata zvočnih datotek)
CentOS	*community enterprise operating system* (prosto dostopna distibucija operacijskega system Linux)
DPS	digitalno procesiranje signalov
FFT	*fast Fourier transform* (hitri Fourierov transform)
GUI	*graphical user interface* (uporabniški grafični vmesnik)
HRIR	*head related impulse response* (impulzni odziv modela snemalne glave)
HRTF	*head related transfer function* (prenosna fukncija modela snemalne glave)
KEMAR	*knowles electronics manikin for acoustic research* (vrsta modela snemalne glave za zvočne raziskave)
MIT	*Massachusetts Institute of Technology* (Tehnološki inštitut v Massachusetts-u)
ML	*maximum length* (maksimalna dolžina)
OS	*operating system* (operacijski sistem)
SNR	*signal to noise ratio* (razmerje med signalom in šumom)
SSH	*secure shell* (protokol za varno prijavo)
Tix	*tk interface extension* (raŽširitveni modul za tk)

Poglavje 1

Uvod

Prostorski zvok je del vsakdanjega življenja. S pomočjo prostorske zaznave dobimo osnovne informacije o svetu okoli nas. Zaznavanje prostora temelji na lastnostih človeškega ušesa. Obe ušesi namreč slišita nekoliko drugačen zvok. Razlika med zvokoma nastane zaradi odbojev v uhlju in zgornjem delu telesa. Ta razlika je odvisna tudi od smeri in višine izvora zvoka. Če izvor ni direktno pred nami, pride prav tako do razlike prihoda in jakosti zvoka med levim in desnim ušesom. Iz opisanih razlik človekov sluh poleg samega zvoka izlušči še informacijo o poziciji izvora zvoka.

Razvoj simulacije prostorskega zvoka sega v leto 1940, ko je bil posnet prvi animirani film s 3D zvočnimi učinki. Ti filmi imajo posnete zvoke na več snemalnih napravah, ki so različno razporejene v prostoru. Nato pa s podobno razporeditvijo zvočnikov, kot so bile razporejene snemalne naprave pri snemanju, film predvajajo v dvorani (*surround sound*). Prostorski zvok je lahko tudi sredstvo, s katerim obogatimo računalniško animacijo, aplikacijo,...

Prostorskost lahko dosežemo tudi s prevajalnimi funkcijami, merjenimi s pomočjo snemalnega modela glave, ki ima levi in desni model uhlja z mikrofonom za snemanje zvoka. S pomočjo primerjave med posnetim zvokom in izvornim zvokom določimo odzive na enotin impulz za vsako uho posebej. Meritve so opravljene v točno določenih pozicijah izvora zvoka v zvočno neodbojnem prostoru. V diplomski nalogi bomo uporabili zbirko MIT-KEMAR [2], ki vsebuje opisane odzive na enotin impulz. Ker je zbirka pridobljena z dejanskimi meritvami vsebuje natančne rezultate, vendar je njena uporaba računsko zelo zahtevna. Kljub svoji natančnosti na vsakega poslušalca deluje različno – vsak ima namreč nekoliko različno obliko uhlja od snemalnega modela. Zato neredko v praksi želimo drugačne, nekoliko manj natančne, a hitrejše postopke. To lahko dosežemo z uporabo bolj enostavnih modelov preoblikovanja zvoka pri njegovem prostorskem zaznavanju. Mi smo izbrali že omenjen pristop z uporabo zbirke meritev.

Cilj naloge je narediti aplikacijo z grafičnim uporabniškim vmesnikom, s katerim lahko tvorimo poljubno 3D zvočno sceno. V aplikaciji uporabimo mono zvočne datoteke, katere

obdelamo s pomočjo postopka konvolucije za vsako uho posebej. Nato posnetka za levo in desno uho kot ločena kanala združimo v stereo zvočno datoteko. Rezultat je zvočna datoteka s 3D učinkom. Te datoteke lahko kasneje poljubno sestavljamo zaporedno eno za drugo ali pa vzporedno eno ob drugo. Aplikacija omogoča tudi izvoz zvočne datoteke.

V sodobnem času se je zanimanje za 3D predelavo zvoka povečalo in postaja vse bolj pomembno. Bodisi zaradi razvoja računalniških iger, pripomočkov za slabovidne ali pa v navidezni ali obogateni resničnosti.

Poglavje 2

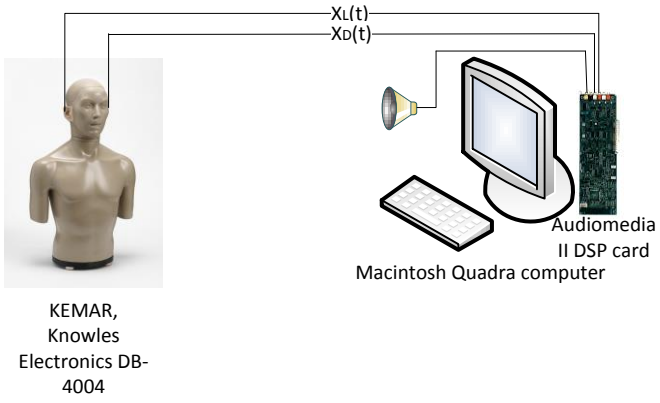
Knjižnica za tvorbo 3D zvoka MIT-KEMAR

Za simulacijo 3D zvoka je bila uporabljena javno dostopna knjižnica MIT-KEMAR, ki je nastala na Massachusetts Institute of Technology v ZDA. Dobljena je s pomočjo snemalnega modela glave KEMAR, ki vsebuje mikrofone za prostorsko snemanje. Zvok je bil generiran z Realistic Optimus Pro 7 zvočnikom oddaljenim 1,4 metra od modela za zaznavanje. Pri frekvenci vzorčenja 44,1 kHz so bile uporabljene največje možne dolžine psevdo-naključnih dvojiških zaporedij za pridobivanje impulznih odzivov. Meritve so potekale na 710 različnih pozicijah v prostoru. Rezultati meritev so odzivi na enotin impulz za levo in desno uho.

2.1 Tehnika merjenja

Sistem za izvedbo meritev je bil sestavljen iz zvočne kartice, *anti-aliasing* filtra, ojačevalnika, zvočnika in sobe v kateri je snemalna glava s pripadajočimi mikrofoni in predojačevalniki. Merjenje je potekalo z Macintosh Quadcore računalnikom, v katerega je vgrajena Audiomedia II DPS kartica, ki ima 16-bitna stereo A/D in D/A pretvornika. Meritve so izvedli s frekvenco vzorčenja 44,1 kHz. Izhodni signal enega od obeh kanalov kartice so preko ojačevalca signala povezali na Realistic Optimus Pro 7 zvočnik. Sprejemni del sestavlja snemalni model glave (KEMAR – *Knowles Electronics Manikin for Acoustic Research* DB-4004, levi uhelj DB-061, desni uhelj DB-065, mikrofon Etymotic ER-11) in Etymotic ER-11 predojačevalnik. Izhoda slušalk sta povezana v stereo vhod kartice Audiomedia. Povezava med snemalno glavo in računalnikom je prikazana na sliki 1. Signali, poslani na avdio izhod (zvočnike), se pojavijo na vhodih mikrofonov v snemalnem modelu glave. Rezultat meritev je *Head related transfer function* – HRTF oziroma *Head related impulse response* – HRIR. HRTF je sistemska oziroma prevajalna funkcija med izvornim in zaznanim zvokom. Spada med prenosne funkcije, ki opisujejo relacije med vhodi in izhodi linearnih časovno invariantnih sistemov. HRIR je ustrezen odziv na enotin impulz. S pomočjo FFT in HRTF dobimo HRIR oziroma odziv na enotin impulz, ki vsebuje značilnosti celotnega merilnega sistema, vključno z odboji v akustičnem prostoru in vplivom zvočnika. Vpliv odbojev v prostoru odpravimo tako, da zagotovimo njihovo čim večjo zakasnitev – to pomeni, da se

njihov vpliv pojavi na mikrofonih za odzivom snemalnega modela glave, ki traja nekaj milisekund. Odziv zvočnika izmerimo ločeno in naredimo inverzni filter, ki v povezavi s HRTF meritvijo ustvari signal, ki ni popačen.



Slika 1: Prikaz povezave med računalnikom ter snemalno glavo

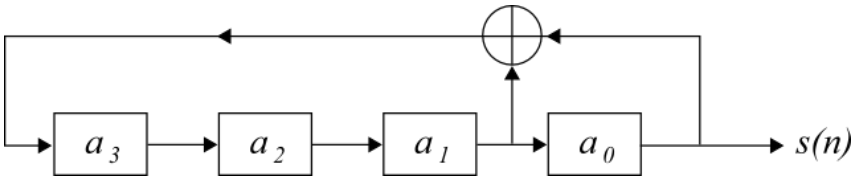
Vsi odzivi so bili pridobljeni z uporabo ML zaporedij, ki so opisana v podpoglavju 2.2. Dolžina zaporedij N je 16383 vzorcev, kar ustreza 14-bitnemu registru. Dve sestavljeni zaporedji tvorita 2 * N dolgo zaporedje vzorcev, ki se predvaja iz zvočne kartice in posname na obeh vhodnih kanalih. Posneto ML zaporedje vključuje hrup, nelinearnosti v sistemu ter časovno prekrivanje. Za zmanjšanje hrupa je potrebno uporabiti daljša zaporedja ali pa vzeti povprečja odzivov na krajša ML zaporedja. Vpliv nelinearnosti zmanjšamo s povprečenjem več različnih meritev. Vpliv časovnega prekrivanja pa v splošnem zmanjšamo z uporabo daljših ML zaporedij. Ker so bile v našem primeru meritve narejene v zvočno neodbojnem prostoru in z dovolj dolgimi ML zaporedji, časovno prekrivanje ni bil problem. Izbran je bil 16383 dolg vzorec, ki je dal dober odziv, brez pretirane časovne in prostorske zahtevnosti. Izmerjeno razmerje med signalom in šumom (SNR) je bilo 65 dB [2].

2.2 ML zaporedja

Ta zaporedja spadajo med psevdo-naključna dvojiška zaporedja. So zaporedja bitov tvorjena z linearnimi povratnimi premičnimi registri. Za procesiranje signalov se stanji 0 in 1 preslikata v ± 1. Z m številom registrov dobimo zaporedje dolžine $2^m - 1$. ML zaporedja običajno uporabljamo za merjenje odziva na enotin impulz [4].
Slika 2 prikazuje shemo pomičnega registra dolžine 4, ki tvori ML zaporedje po naslednjem izrazu:

$$a_k[n+1] = \begin{cases} a_0[n] + a_1[n], k = 3 \\ a_{k+1}[n] \end{cases} \quad (1)$$

n – časovni indeks
k – pozicija v registru
+ – seštevanje po modulu 2



Slika 2: Pomični register dolžine 4 za tvorbo ML zaporedij

2.3 Inverzni filter

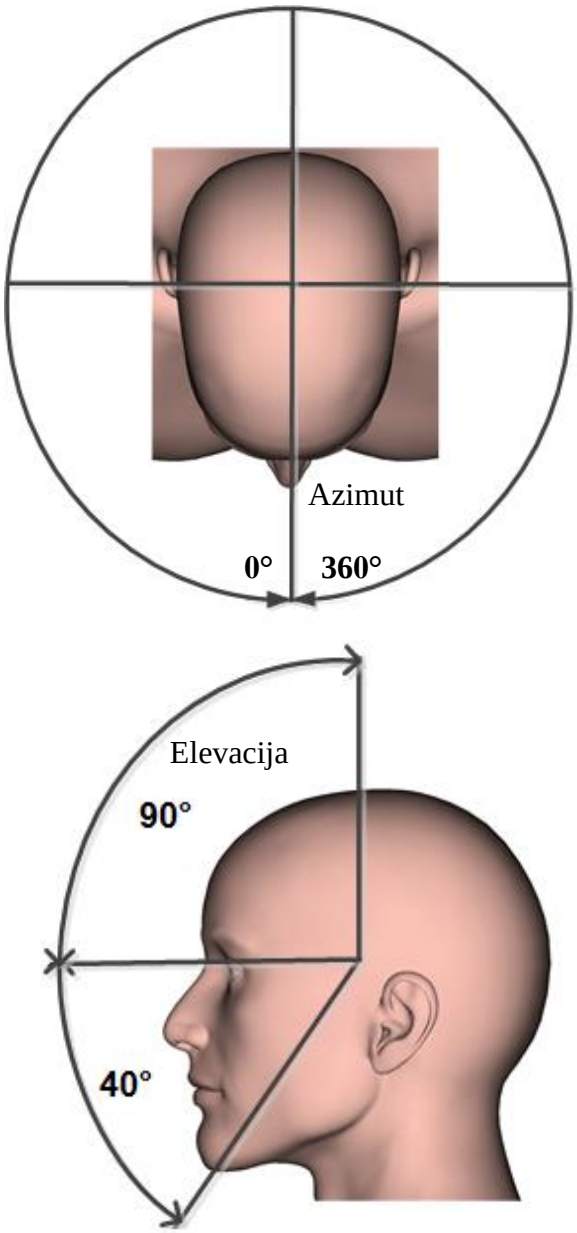
Filtri so sistemi, ki običajno iz signala odstranijo neželene komponente. Največkrat se uporabljajo za odstranjevanje določenih frekvenčnih komponent z namenom, da se odpravijo motnje in/ali zmanjša hrup iz ozadja. Včasih pa delovanje sistemov neželeno spremeni signal, ki ga poskušamo povrniti v prvotno stanje. Temu postopku rečemo inverzno filtriranje ali dekonvolucija; pomeni pa odpravo učinkov konvolucije na obdelanem signalu. Konvolucija je operacija med vhodnim signalom in prenosno funkcijo sistema, ki jo ponazarja odziv na enotin impulz – *h(n)*. Enačba (2) prikazuje enačbo konvolucije, kjer je *y(n)* signal, ki ga dobimo iz poljubnega vhodnega signala *x(n)* in odziva na enotin impulz *h(n)*.

$$y(n) = h(n) * x(n) \tag{2}$$

V zbirki MIT-KEMAR so tehniko inverznega filtriranja uporabili za odpravo vpliva lastnosti Optimus 7 zvočnika na izvedeno HRTF meritev.

2.4 Merilni postopek

V zvočni komori je bila v pokončen položaju nameščena snemalna glava, ki se preko računalnika lahko zavrti do določenega azimuta. Zvočnik je postavljen na stojalo. S tem je omogočena namestitev njegove višine glede na glavo. Od zvočnika je glava oddaljena 1.4 metra. Za vsako višino posebej so narejene meritve. Lokacija izvora zvoka se izraža v polarnem zapisu. Sfero okrog omenjenega snemalnega modela so z meritvami vzorčili od -40 stopinj (pod vodoravno ravnino) do +90 stopinj (neposredno nad). Za to postavko bomo uporabljali izraz elevacija. Na vsaki višini so merili vseh 360 stopinj okrog glave z enakim korakom. Temu parametru pravimo azimut. Slika 3 prikazuje kote meritev glede na glavo. Skupno so meritve opravil na 710 lokacijah okrog glave. V tabeli 1 so prikazani podatki o številu meritev in povečanju azimuta ob določeni meritvi.



Slika 3: Prikaz kota azimuta in elevacije glede na glavo

Elevacija	Število meritev	Povečanje azimut
-40	56	6,43
-30	60	6,00
-20	72	5,00
-10	72	5,00
0	72	5,00
10	72	5,00
20	72	5,00
30	60	6,00
40	56	6,43
50	45	8,00
60	36	10,00
70	24	15,00
80	12	30,00
90	1	XX

Tabela 1: Število meritev in korak povečanja azimuta za vsako elevacijo

KEMAR snemalni model glave ima dva različna modela uhljev. Tako z meritvami dobimo dva različna kompleta simetričnih prenosnih funkcij. Poleg merjenja impulznega odziva zvočnika so prav tako opravili meritve z različnimi vrstami slušalk.
Kot že zgoraj opisano, je vsaka meritev prinesla 16383 točkovni odziv na enotin impulz s frekvenco vzorčenja 44,1 kHz. Večina tega signala je nepomembna zaradi oddaljenosti merilnika in vpliva predvajalno-snemalnega sistema. Da bi zmanjšali velikost nabora nepomembnih podatkov brez izgube potencialno zanimivih, so se odločili, da zavržejo prvih 200 in shranijo sledečih 512 vzorcev. Odzivi so shranjeni kot 16-bitna predznačena cela števila z najpomembnejšim bajtom na nižjem naslovu. Dinamični razpon 16-bitnih števil (96 dB) presega izmerjeno razmerje med šumom in signalom meritve, ki je znašal 65 dB [2].

2.5 Organizacija podatkov v zbirki

Odzivi so shranjeni v mape glede na višino. Vsaka mapa ima ime 'elevEE', kjer je EE kòt višine v stopinjah. V njih so datoteke z formatom imena 'XEEeAAAa.dat', kjer predstavlja X levi (L) ali desni (D) odziv. EE je kòt višine glede na glavo v stopinjah med -40 ter 90, AAA pa predstavlja azimut prav tako v stopinjah med 0 in 355. Elevacija in azimut označujeta lokacijo izvora zvoka glede na glavo. To pomeni, da elevacija ter azimut 0 stopinj predstavljata lokacijo izvora direktno pred, višina 90 stopinj pomeni navpično nad, višina 0 in

azimut 90 stopinj pa skrajno desno od glave. Na primer datoteka R-20e270a.dat je odziv sistema z izvorom 20 stopinj pod vodoravno ravnino in 90 stopinj v levo od glave. Odzivi zvočnikov in slušalk so shranjeni v stisnjeni mapi »headphones+spkr«. Obstaja tudi zgoščena verzija knjižnice odzivov, ki smo jo uporabili mi. Struktura podatkov je organizirana podobno po mapah in s podobnimi imeni. Razlika z zgornjo zbirko je v tem, da datoteke niso razdeljene v rezultate za levi in desni uhelj, ampak vsebuje oba odziva in je za polovico krajša (256 vzorcev). Ime datotek se začne z H nato sledi kòt višine glede na glavo in AAA, ki predstavlja azimut (primer: H10e050a.wav). Ta kòt je le med 0 in 180 stopinj saj lahko s pomočjo simetrije tvorimo 3D zvok za obe strani [2].

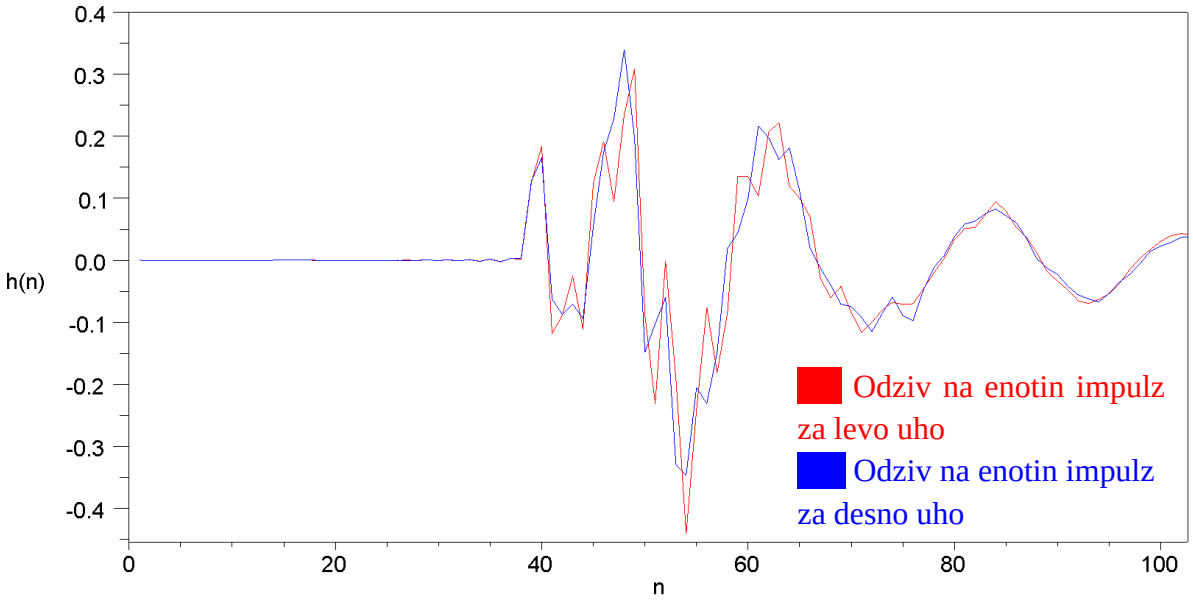
Poglavje 3

Procesiranje zvočnih signalov v okolju Scilab

Scilab je odprt programski paket za visokonivojsko numerično računanje in deluje na vseh OS. Lahko ga uporabljamo za procesiranje signalov, statistične analize, izboljšave slik. Sestavljajo ga trije glavni deli: tolmač, knjižnice in funkcije. Specializiran je za računanje z matrikami (seštevanje, množenje, trasponiranje, inverz,…). Na enostaven način tvorimo rezultate in rišemo dvodimenzionalne in tridimenzionalne grafe. Prav tako ima odprto razvojno okolje, ki omogoča uporabnikom razvoj lastnih funkcij in knjižnic. V diplomski nalogi smo uporabili verzijo 5.2.2 z dodatkom, ki je dostopen na spletni strani *http://laps.fri.uni-lj.si/dps/dps_lab_2010/index.html* in ustvari hierarhijo delovnih map.
Na začetku praktičnega dela diplomske naloge smo preverili lastnosti, ki veljajo za ušesi v odvisnosti od lokacije izvora zvoka. Napisali smo funkcijo v kateri narišemo časovni potek signala za levi in desni odziv ušesa iz zbirke KEMAR. Za vhodne podatke smo si izbrali odzive z elevacijo 0° in azimute 0°, 45°, 90°, 135°, 180°, 225°, 270° in 315°. Ti koti ponazarjajo smer lokacije izvora po vrsti kot si sledijo zgoraj:

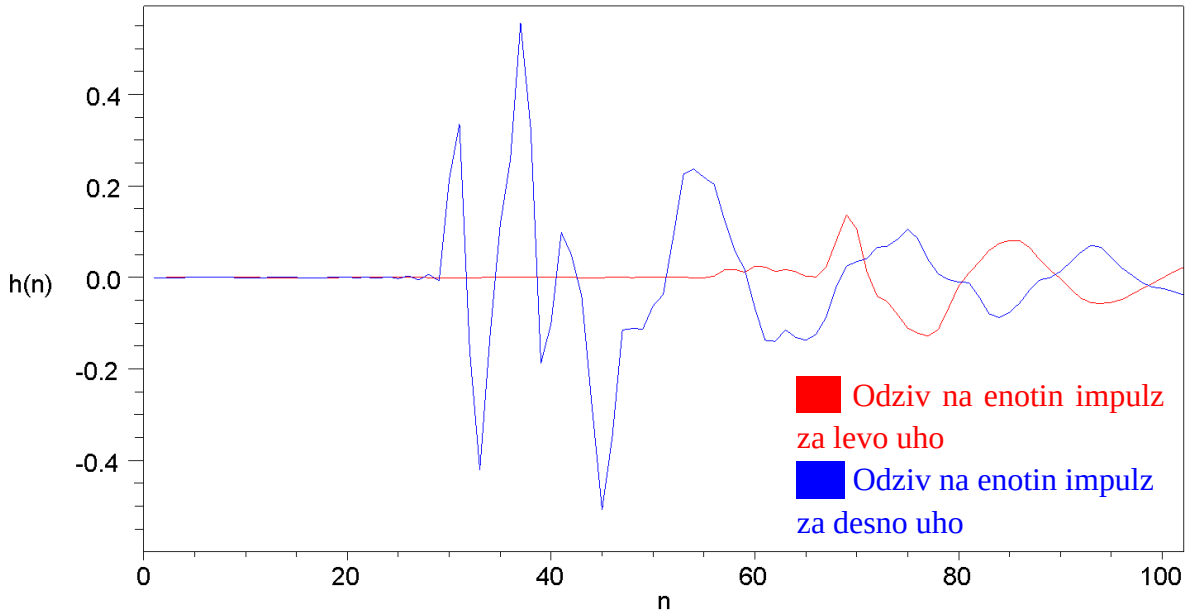
- direktno pred nami,
- med lego direktno pred nami in skrajno desno lego,
- skrajno desna lega,
- med skrajno desno lego in za nami,
- za nami,
- med lego za nami in skrajno levo lego,
- skrajno leva lega,
- med skrajno levo lego in lego direktno pred nami.

Časovna poteka odzivov pri azimutu 0° za levo in desno uho se skoraj povsem prekrivata. Ravno tako pri azimutu 180°. Časovni potek odzivov za lego direktno pred nami je viden na sliki 4, na sliki 5 pa za skrajno desno lego. Iz slike 4 se dobro vidi časovni zamik prihoda zvoka v obe ušesi, ki je približno po prvih 35 vzorcih (nekaj milisekund).



Slika 4: Časovni potek odzivov na enotin impulz pri azimutu 0°

Pri kotih azimuta 45°, 90°, 135°, 225°, 270° in 315° se pojavi razlika časa prihoda zvoka v posamezen uhelj. Med časovnimi poteki se pojavi simetrija glede na lokacijo izvora. Časovni poteki parov so zrcalni, odziva za levo in desno uho sta torej ravno obratna pri kotih 45° in 135°, 90° in 270° ter 135° in 315°. Na sliki 5 je prikazan časovni potek pri azimutu 90°, kjer se najbolje vidi razlika v času prihoda zvoka med ušesoma (približno 35 vzorcev). Razlog je v tem, da se zaradi razdalje med ušesoma, oblike glave in velikosti glave najprej pojavi zvok na desnem ušesu, šele nato na levem. Zaradi te razlike je tudi amplituda odziva pri bližnjem ušesu višja.

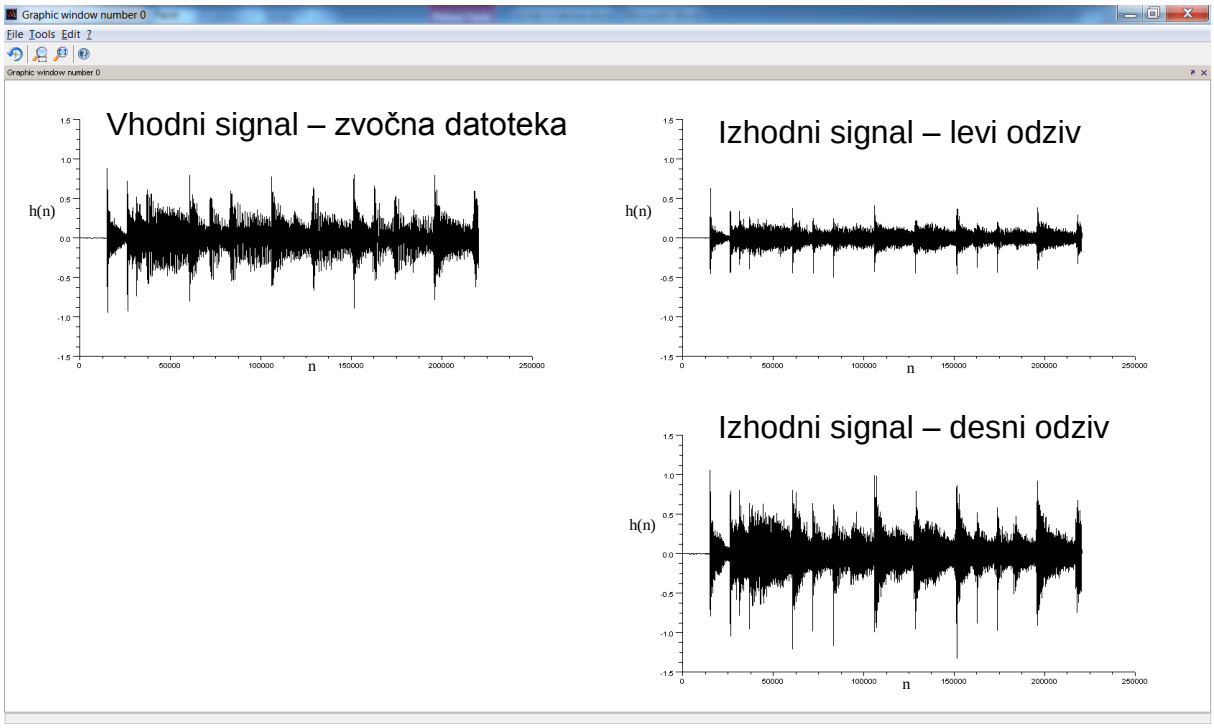


Slika 5: Časovni potek odzivov na enotin impulz pri azimutu 90°

Nato smo napisali funkcijo, s katero dodamo zvočni datoteki 3D učinek. Vhodna podatka sta mono zvočna datoteka ter datoteka z levim in desnim odzivom na enotin impulz iz zbirke MIT-KEMAR. S pomočjo teh dveh datotek in konvolucije dobimo prostorski zvok, ki ga shranimo v izhodno spremenljivko. Pri tem moramo paziti, da uporabimo konvolucijo za vsak kanal posebej. Izhod je dvovrstična matrika, ki se shrani v stereo datoteko s prostorskim zvokom. Za ponazoritev smo dodali še grafe vhodnih in izhodnih podatkov. Slika 6 prikazuje enostavno funkcijo za Scilab, s katero tvorimo 3D zvok. Na sliki 7 pa je rezultat funkcije iz slike 6.

```
1  function [y] = zvok(mono_zvok,odziv)
2  x = loadwave(mono_zvok);
3  h = loadwave(odziv);
4  h_l = convol(h(1,:),x);
5  h_d = convol(h(2,:),x);
6  y(1,:) = h_l;
7  y(2,:) = h_d;
8  wavwrite(y,44100,'stereo.wav');
9  a = gca();
10 a.data_bounds = [0,-1.5 ; length(x),1.5];
11 subplot(221); plot2d(x);
12 title('Mono sound')
13 a = gca();
14 a.data_bounds = [0,-1.5 ; length(y(1,:)),1.5];
15 subplot(222); plot2d(y(1,:));
16 title('Left speaker')
17 a = gca();
18 a.data_bounds = [0,-1.5 ; length(y(1,:)),1.5];
19 subplot(224); plot2d(y(2,:));
20 title('Rigth speaker')
21 endfunction
```

Slika 6: Funkcija za tvorbo 3D zvoka



Slika 7: Prikaz vhodnega in izhodnega signala ob tvorbi prostorskega zvoka

Poglavje 4

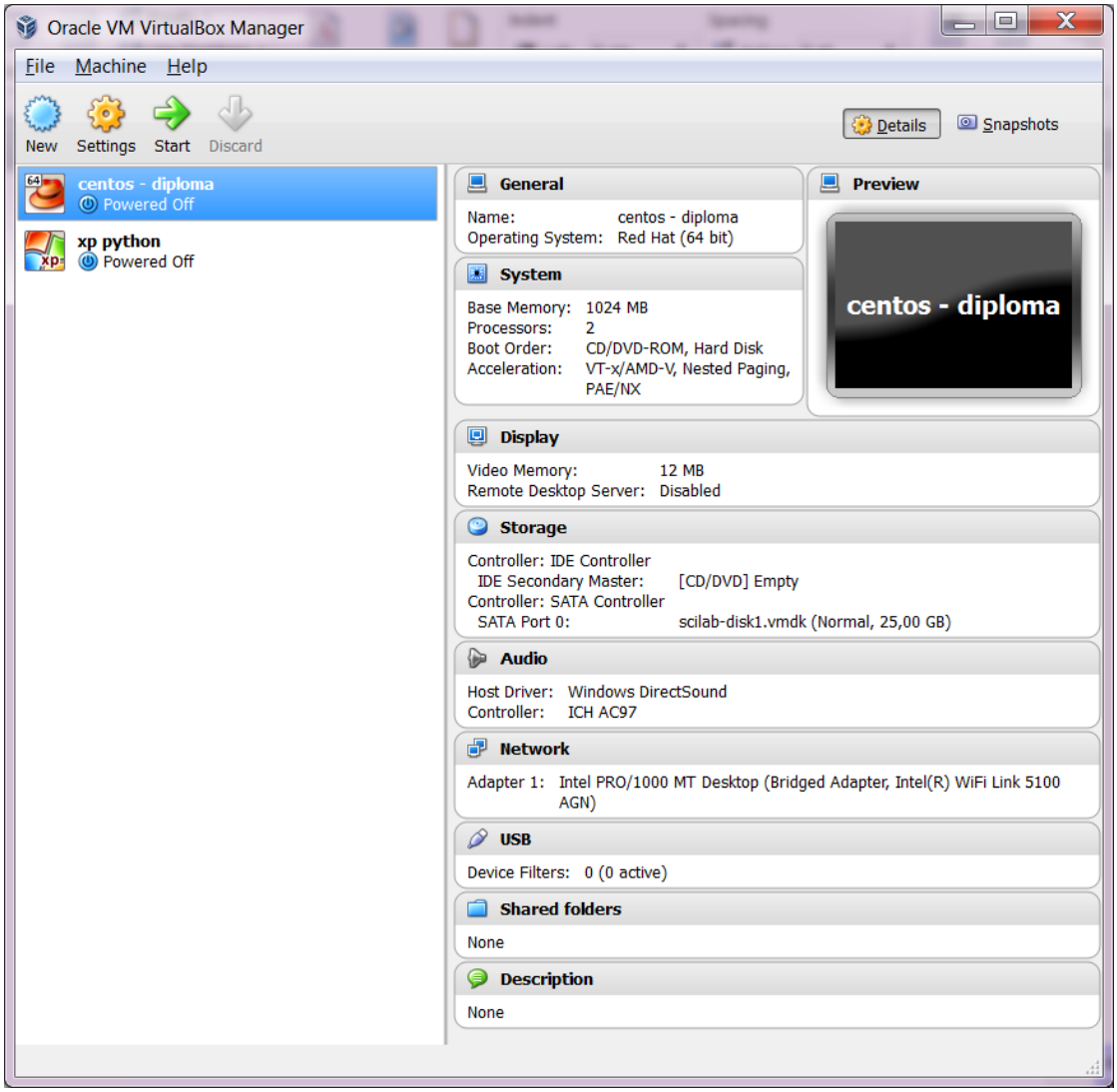
Virtualizacija razvojnega okolja

S pomočjo virtualizacije lahko na vsakem dovolj zmogljivem računalniku enostavno naredimo navidezni (virtualni) računalnik ali pa celo več takih računalnikov. Nam je virtualizacija služila, da nismo posegali v obstoječi operacijski sistem oziroma smo vzpostavili prenosljivo razvojno okolje. Za realizacijo navideznega računalnika smo uporabili orodje VirtualBox, ki ga namestimo na že obstoječi operacijski sistem. Zgoraj omenjeno orodje je prikazano na sliki 8.

Možnosti in značilnosti, ki jih zagotavlja VirtualBox:

- Zagon več operacijskih sistemov hkrati. Tako lahko poganjamo programsko opremo napisano za različne sisteme brez ponovnega zagona računalnika. Namestimo lahko starejše operacijske sisteme tudi, če nam naša oprema tega ne dovoljuje več.
- Lažja namestitev programske opreme. Prodajalci programov uporabljajo virtualne stroje za pošiljanje celotne konfiguracije končnim uporabnikom. Recimo popolna rešitev poštnega strežnika na pravi stroj. Namestitev in konfiguracijo poštnega strežnika naredimo na navideznega. Tako na nek način zapakiramo sistem in z njim vse funkcije, ki smo mu jih dodelili. Postane neodvisen paket, katerega lahko poljubno uvažamo na drug računalnik.
- Testiranje in reševanje podatkov ob nesrečah. Ob tvorjenju navideznega stroja, le tega vidimo neodvisno od gostitelja. Lahko ga poljubno zamrznemo, varnostno kopiramo in premikamo med želenimi gostitelji. Se pravi, da lahko poganjamo enak navidezni stroj na različnih mestih. S tem omogočimo testiranje programov.
- Kreiranje posnetkov. S pomočjo te funkcije shranimo določeno stanje navideznega računalnika. Na ta način nam je zelo olajšano delo z okoljem. Recimo učenje uporabe novega operacijskega sistema in eksperimentiranje v njem. Če si okolje »pokvarimo«, se enostavno vrnemo na prejšnji posnetek in se izognemo ponovnim namestitvam in konfiguraciji sistema. Ko katerega med njimi ne potrebujemo več, ga lahko pri vklopljenem sistemu izbrišemo.

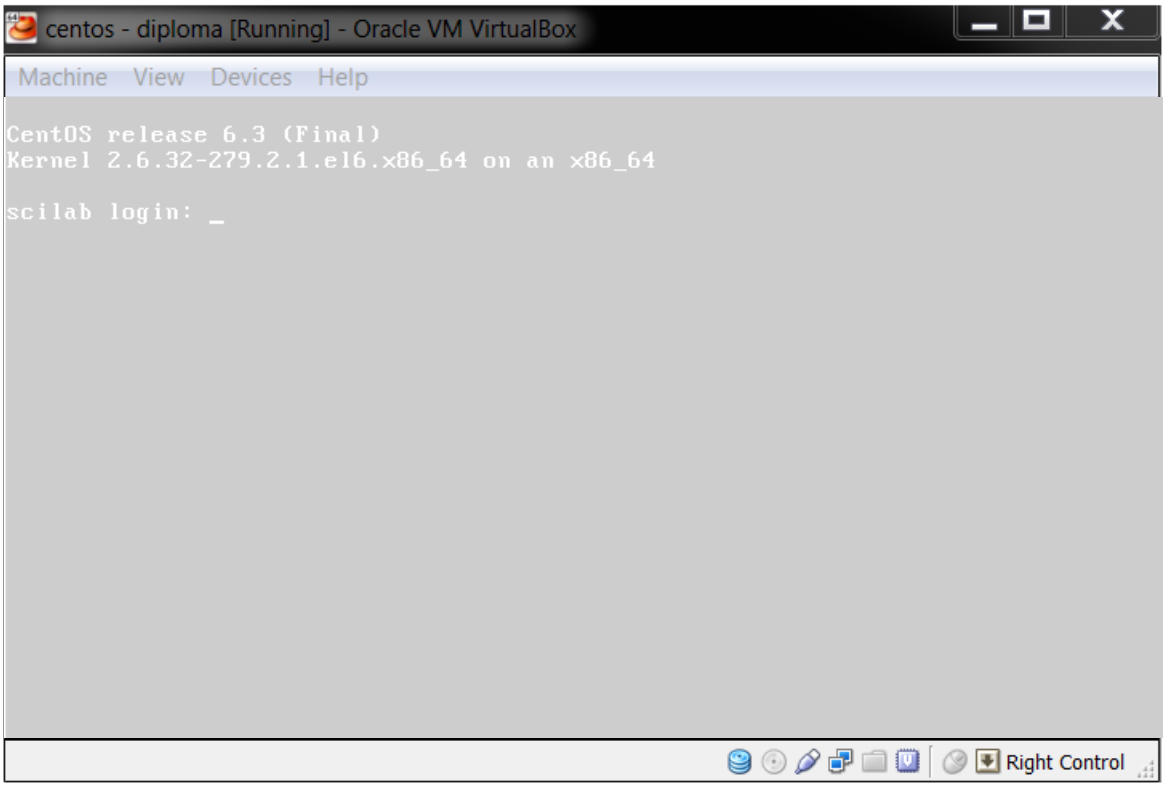
- Infrastrukturna konsolidacija. Z navideznimi stroji se lahko zmanjšajo stroški strojne opreme in električne energije. Računalniki večino časa uporabljajo minimalno svojih resursov. S tem zapravljamo strojno opremo in energijo. Namesto več fizičnih računalnikov, ki le delno uporabljajo svojo opremo, jih je mogoče zapakirati na le nekaj gostiteljev [8].



Slika 8: Orodje VirtualBox

4.1 Ustvarjanje navideznega računalnika

Na fizični računalnik (4GB spomina, 250GB trdi disk, Intel procesor) smo namestili programsko opremo VirtualBox. Ustvarili smo navidezni sistem (1GB spomina, 25GB trdi disk) ter nanj namestili 64-bitni operacijski sistem CentOS 6, v katerega je že integrirano okolje Python, ki smo ga potrebovali za nadaljnje delo. Na sliki 9 je prikazano orodje VirtualBox z nameščenim CentOS 6 operacijskim sistemom.



Slika 9: Operacijski sistem CentOS na VirtualBox

4.1.1 Operacijski sistem CentOS

CentOS je ena izmed vrst Linuxovih operacijskih sistemov. Je prosto dostopna distribucija, ki izhaja iz različice imenovane Fedora. Zaradi odprtosti ga lahko po potrebi izboljšujemo in prilagajamo. Linux smo izbrali, ker je enostaven za namestitev. Programske pakete lahko nameščamo sproti po potrebi. Prav tako pri namestitvi nimamo težav z iskanjem in nameščanjem gonilnikov, saj za to poskrbi operacijski sistem sam. Ker je zelo stabilen, ga ni potrebno tolikokrat ponovno zaganjati. Stabilnost sistema nam nudijo preverjeni paketi, ki so preizkušeni v različici iz katere izhajajo. Virtualni sistemi z grafičnimi vmesniki so računsko zahtevni. Za naše delo pa potrebujemo le razvojno okolje z programskim jezikom Python. Dobra lastnost Linux OS je dostopnost preko SSH protokola. S pomočjo tega protokola in sistema »X-Windows« lahko oddaljeno zaganjamo grafične programe.

Poglavje 5

Python programski jezik

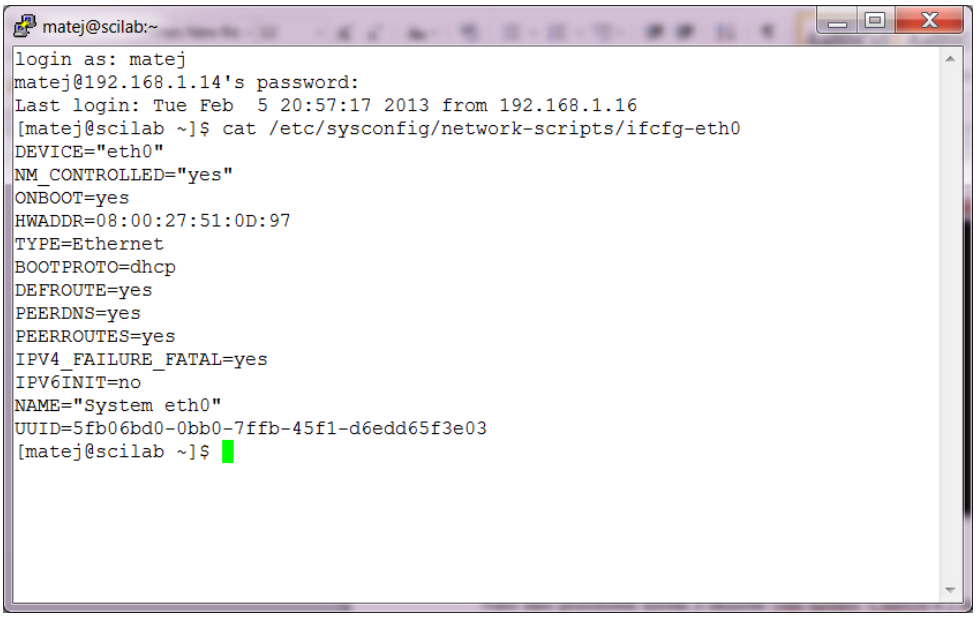
Python je odprtokodni skriptni jezik, ki ga je razvil Guido van Rossum v začetku 90-ih let. Poimenoval ga je po filmu Monty Python. Je preprost, enostaven, objektno usmerjen jezik, ki s pomočjo velikega števila podatkovnih struktur zagotavlja visoko raven programiranja. Uporaben je za hiter razvoj aplikacij in se odlikuje z dobro berljivostjo izvorne kode. Prosto dostopen je za vse vrste operacijskih sistemov. Podpira orodja operacijskega sistema, kot so na primer spremenljivke okolja, vtičnice, procesi, niti in regularni izrazi. Na voljo imamo ogromno število standardnih modulov in paketov. V nadaljevanju bomo opisali uporabljene module.

Uporabljeni moduli:

- *Tkinter* : Povezava programskega jezika Python na Tk GUI orodje, ki je znano odprto kodno orodje za gradnjo grafičnih elementov. Razvit je bil kot razširitev Tcl skriptnega jezika. Omogoča knjižnice osnovnih elementov za oblikovanje grafičnih uporabniških vmesnikov v različnih programskih jezikih. Ponuja ogromno število pripomočkov potrebnih za razvoj namiznih aplikacij, kot so gumb, meni, platno, okvir, drsnik, seznamsko polje, itd.. Podobno kot Python je neodvisen od platforme, torej ga lahko prenesemo na vse platforme. Je tudi prilagodljiv, saj imajo skoraj vse funkcije gradnikov možnost prilagajanja med ustvarjanjem ali morda celo kasneje skozi ukaze. Veliko možnosti je mogoče shraniti v zbirko podatkov, zaradi česar je enostaven za nastavljanje videza aplikacije [5].
- *Os* : Modul omogoča prenosni način uporabe operacijskega sistema odvisno od funkcionalnosti. Uporabimo ga, če želimo brati ali pisati v datoteke ter manipulirati poti. Z njim lahko beremo vrstice v ukazni vrstici in tvorimo začasne datoteke ali mape.
- *Sys* : Z njim dostopamo do nekaterih spremenljivk, ki jih uporablja tolmač, in do funkcij, ki so povezane s tolmačem. Ta modul je vedno na voljo.

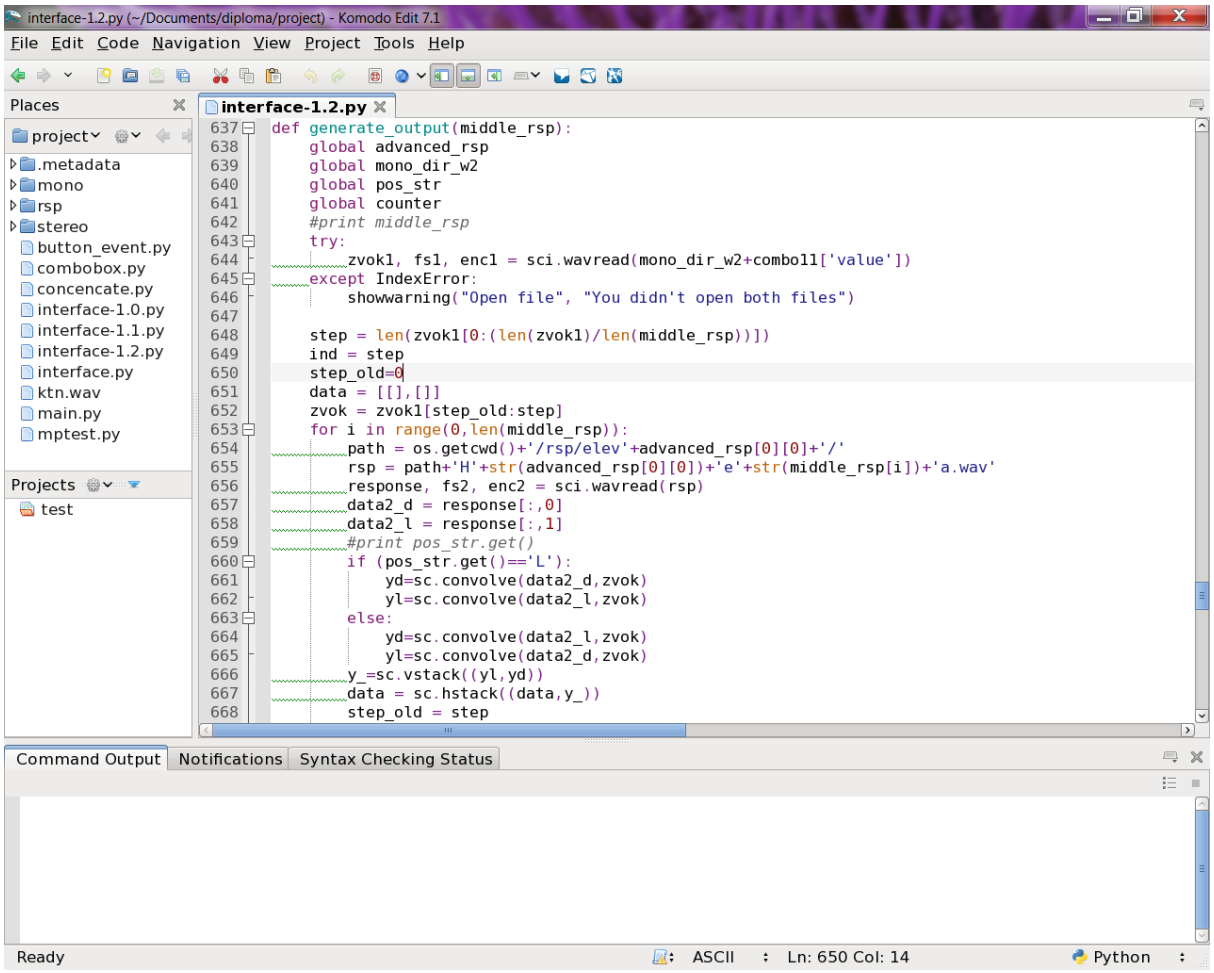
- *NumPy* : Razširitev programskega jezika Python s podporo za velike, večdimenzionalne matrike in polja skupaj s knjižnico matematičnih funkcij za operacije z njimi, kot so funkcije linearne algebre in Fourierjeve transformacije.
- *SciPy* : Knjižnica algoritmov in matematičnih orodij za programski jezik Python. Vsebuje module za optimizacijo, linearno algebro, integracijo, interpolacijo, FFT, procesiranje signalov in obdelavo slik, itd. Osnovna struktura podatkov v SciPy je večdimenzionalno polje NumPy modula.
- *Audiolab scikits* : Paket za branje, pisanje in predvajanje zvočnih datotek. Uvoz in izvoz poteka neposredno v in iz NumPy polj. Podpira avdio formate Wave, AIFF, ogg,...
- *Matplotlib* : Knjižnica za Python in njegovo razširitev NumPy za risanje grafov.
- *Tix* : Tix modul omogoča dodatno bogat nabor gradnikov za Tkinter. Čeprav ima standardna Tk knjižnica številne uporabne pripomočke, so daleč od popolne. Ta knjižnica ponuja večino pogosto potrebnih pripomočkov, ki manjkajo v standardnem Tk modulu.
- *Py2exe* : Razširitveni modul, ki pretvori Python skripte v Windows izvršne programe. S tem lahko zaganjamo aplikacije, napisane v Pythonu, brez njegove namestitve na računalniku.

Za dostop preko SSH povezave smo najprej preverili mrežne nastavitve. Če le-te ob namestitvi OS niso bile spremenjene je potrebno popraviti datoteko /etc/sysconfig/network-script/ifcfg-eth0, in sicer tako, da ob zagonu nastavimo IP (popravimo parameter spremenljivki ONBOOT v »*yes*«). Mrežne nastavitve so prikazane na sliki 10.



Slika 10: Konfiguracija mrežnih nastavitev v okolju CentOS

Nato smo posodobili sistem z ukazom *yum update*. CentOS 6.2 ima že vgrajen Python verzijo 2.6.6. Potrebno je še namestiti module potrebne za razvoj aplikacije. Z Python verzijo 2.6.6 so združljivi paketi NumPy 1.6.1, SciPy 0.10.1, scikits.audioloab 0.11.0 in matplotlib 1.1.1. Za operacijski sistem so značilni rpm paketi, ki so namestljivi z ukazom *rpm –i ime_paketa*. Za lažje programiranje smo namestili in uporabljali urejevalnik Komodo Edit. Slika 11 prikazuje urejevalnik Komodo edit.

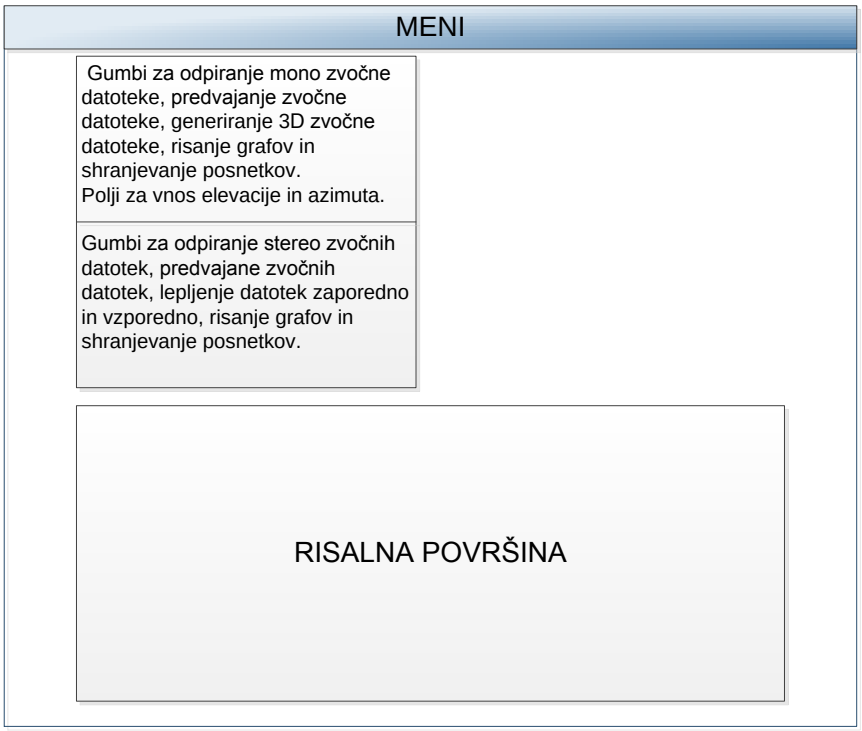


Slika 11: Komodo Edit

5.1. Postopek izdelave programa

Kot smo že v uvodu napisali, je namen naloge v Pyhon-u napisati aplikacijo z grafičnim vmesnikom za tvorjenje 3D zvokov s pomočjo KEMAR zbirke. Vsebovati mora funkcije za tvorbo zvokov z vsemi možnimi kombinacijami elevacij in azimutov za levo in desno uho. Poleg tega mora obstajati možnost predvajanja zvoka in prikazovanja uvoženih zvokov. Uvodoma smo naredili skico grafičnega vmesnika. Kot je prikazano na sliki 12 smo razdelili glavni grafični vmesnik na tri dele:

- zgornji del za tvorbo enostavnih in naprednih 3D zvočnih posnetkov,
- srednji del za lepljenje posnetkov zaporedno in vzporedno,
- spodnji za prikaz uporabljenih posnetkov oziroma podatkov.

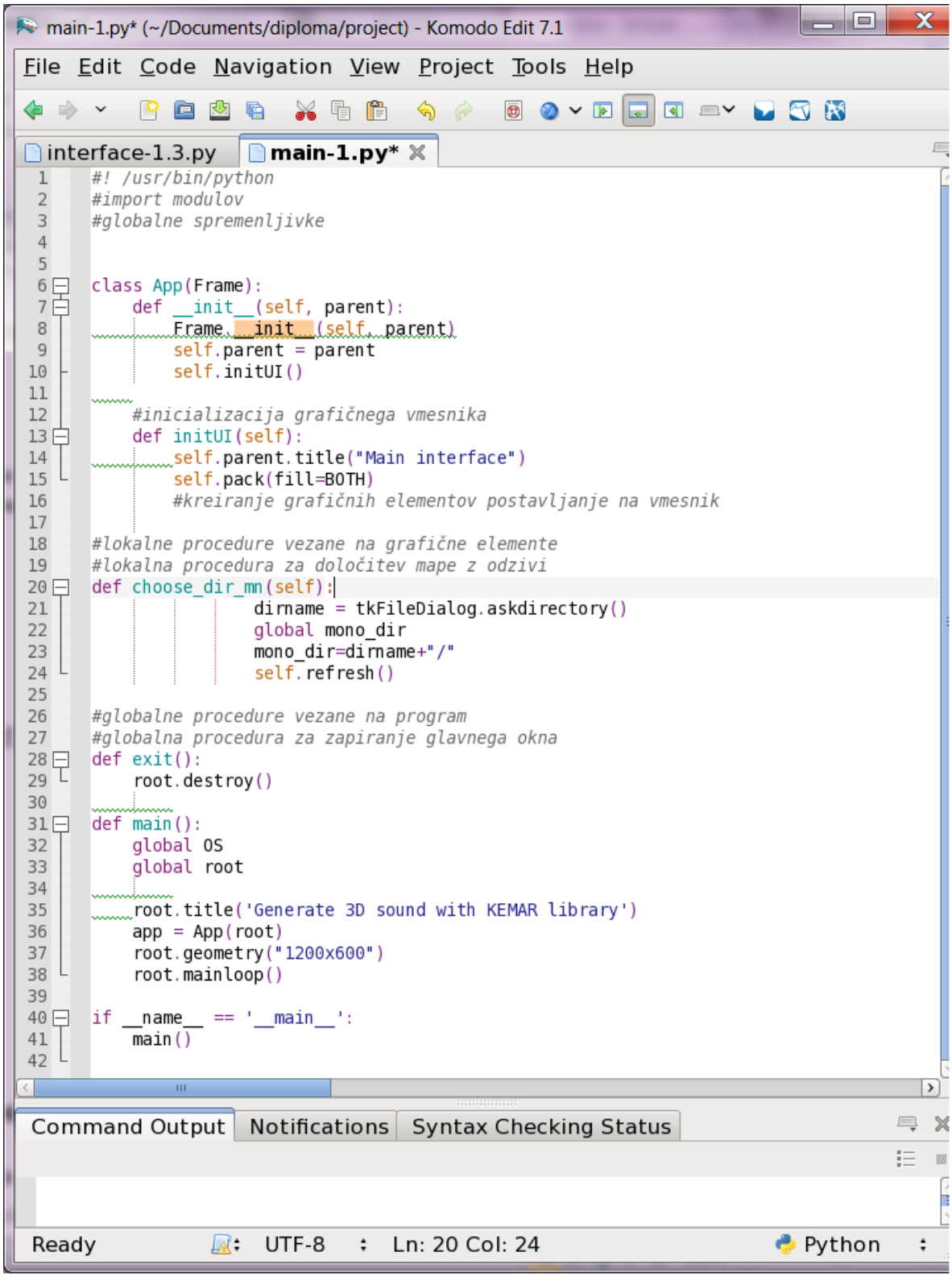


Slika 12: Skica grafičnega vmesnika

5.1.1 Osnovna struktura programa

Najprej smo ustvarili prazen okvir in nanj dodali besedila in gumbe za ustvarjanje, predvajanje, shranjevanje zvokov. Napisali smo še metode, ki so potrebne za operacije nad gumbi. Nastajal je vse kompleksnejši program. Dodali smo še del za sestavljanje zvokov in risalno površino. V prvih verzijah programa smo imeli gumb za izbiranje datoteke z zvoki in odzivi, nato smo jih zamenjali z gradniki uporabniškega vmesnika. To so polja z drsniki in vnosnimi polji za vnos elevacije in azimuta. S tem smo morali nekoliko spremeniti funkcije

ter jih nekaj dodati. Prav tako smo dodali izbirno vrstico, ki ponuja možnosti spremembe poti zvočnih datotek in osnovno pomoč pri uporabi programa. Kodo z osnovno strukturo programa vidimo na sliki 13.



Slika 13: Koda z osnovno strukturo programa

5.1.2 Funkcija za tvorbo 3D zvočnih scen

Zaradi že implementirane funkcije za konvolucijo je tvorba 3D zvoka enostavna. Najprej moramo le prebrati zvočno datoteko in odziv, kar prav tako naredimo z že vgrajeno funkcijo *wavread*. Primer funkcije za tvorbo zvočne datoteke s 3D učinkom je na sliki 14. V drugem delu smo implementirali lepljenje posnetkov. Razvili smo vzporedno in zaporedno sestavljanje zvočnih datotek.

```
def to_3D_right(self):
    global response
    global zvok1
    global mono_dir
    global rsp_dir
    global counter
    self.validate()
    try:
        zvok1, fs1, enc1 = sci.wavread(mono_dir+self.combo1['value'])
        #response, fs2, enc2 = sci.wavread(rsp_dir+self.combo2['value'])
        response, fs2, enc2 = sci.wavread(rsp_dir+"H"+str(elev_text[:-1])+"e"+str(az_text)+"a.wav")

        data2_l = response[:,0]
        data2_d = response[:,1]
        yl=sc.convolve(data2_l,zvok1)
        yd=sc.convolve(data2_d,zvok1)
        y=sc.vstack((yl,yd))
        y=y.transpose()
        sci.wavwrite(y, 'stereo.wav',44100, 'pcm16')
        open_plot_window(y)
        counter = 1
    except IndexError:
        showwarning("Open file", "You didn't open both files")
```

Slika 14: Primer kode za tvorbo 3D zvočne datoteke

5.1.3 Funkcija za vzporedno sestavljanje zvočnih scen

Iz programske kode na sliki 15 je razviden klic funkcije *wavread*, ki iz zvočne datoteke prebere število kanalov, frekvenco vzorčenja in zapise zvočnih signalov za vsak kanal. Nato definiramo novo spremenljivko *y*, v katero smo shranili rezultat. Preden začnemo sestavljati vsebino prebranih podatkov, preverimo ali smo odprli dve eno-kanalni ali dvo-kanalni datoteki. V primeru odprtja dveh eno-kanalnih datotek enostavno zlepimo vektorja skupaj z funkcijo *hstack*. Kadar pa zaporedno kopiramo dve dvo-kanalni datoteki, moramo s pomočjo funkcije *hstack* najprej zložiti levi in nato desni kanal posebej. Naposled še zgradimo novo matriko iz levega in desnega dela z funkcijo *vstack*, ki naredi matriko z dvema stolpcema. Vsak stolpec predstavlja enega od kanalov. Da lahko posnetek shranimo in poslušamo moramo matriko transponirati.

```
def paste_together_S(self):
        global zvok_s1
        global zvok_s2
        global counter
        zvok_s1, fs1, enc1 = sci.wavread(st_dir+self.combo3['value'])
        zvok_s2, fs1, enc1 = sci.wavread(st_dir+self.combo4['value'])
        y = array([])
        if zvok_s1.ndim == 1 and zvok_s2.ndim ==1:
            y=sc.hstack((zvok_s1,zvok_s2))
            counter = 1
        elif zvok_s1.ndim == 2 and zvok_s2.ndim ==2:
            yl=sc.hstack((zvok_s1[:,0],zvok_s2[:,0]))
            yd=sc.hstack((zvok_s1[:,1],zvok_s2[:,1]))
            y=sc.vstack((yl,yd))
            y=y.transpose()
            counter = 1
        else:
            showwarning("Open file", "You didn't open two mono or stereo sounds\n")
        if (len(y) == 0):
            showwarning("Open file", "You didn't open two stereo sounds\n")
        else:
            sci.wavwrite(y,'stereo.wav',44100,'pcm16')
            open_plot_window(y)
```

Slika 15: Primer kode za vzporedno sestavljanje zvočnih datotek

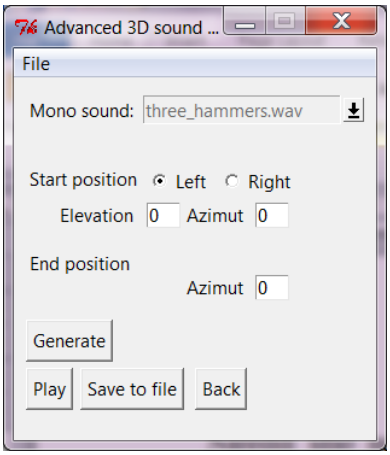
5.1.4 Funkcija za zaporedno sestavljanje zvočnih scen

Kopiranje dveh zvočnih datotek zaporedno smo implementirali podobno kot vzporedno. Pri tej operaciji moramo biti pozorni na dolžine posnetkov. Koda za zaporedno sestavo zvočnih datotek je na sliki 16. Prav tako najprej preberemo podatke o zvočnih datotekah. Nato glede na število kanalov in dolžino vektorjev sestavimo novo matriko s funkcijo *sum*, ki sešteva elemente vektorjev. Ker nismo bili zadovoljni le z enostavnimi 3D zvoki, s samo enim uporabljenim odzivom, smo želeli tvoriti naprednejše, bolj realne posnetke. Torej uporabiti več odzivov na eni zvočni datoteki. Naredili smo nov gumb, v katerem smo klicali proceduro za ustvarjanje novega okna. Ob vsakem tvorjenju novega posnetka smo prikazali njegov časovni potek.

```
def paste_together_P(self):
        global zvok_s1
        global zvok_s2
        global counter
        y = array([])
        counter = 1
        zvok_s1, fs1, enc1 = sci.wavread(st_dir+self.combo3['value'])
        zvok_s2, fs1, enc1 = sci.wavread(st_dir+self.combo4['value'])
        if zvok_s1.ndim == 1 and zvok_s2.ndim ==1:
            counter = 1
            if zvok_s1.shape[0] <= zvok_s2.shape[0]:
                y=sum([zvok_s1,zvok_s2[0:zvok_s1.shape[0]]],axis=0)
            else:
                y=sum([zvok_s1[0:zvok_s2.shape[0]],zvok_s2],axis=0)
        elif zvok_s1.ndim == 2 and zvok_s2.ndim ==2:
            counter = 1
            if zvok_s1.shape[0] <= zvok_s2.shape[0]:
                yl=sum([zvok_s1[:,0],zvok_s2[0:zvok_s1.shape[0],0]],axis=0)
                zvok_s2[0:zvok_s1.shape[0],0]=sum([zvok_s1[:,0],zvok_s2[0:zvok_s1.shape[0],0]],axis=0)
                yd=sum([zvok_s1[:,1],zvok_s2[0:zvok_s1.shape[0],1]],axis=0)
                zvok_s2[0:zvok_s1.shape[0],1]=sum([zvok_s1[:,1],zvok_s2[0:zvok_s1.shape[0],1]],axis=0)
                yl=zvok_s2[:,0]
                yd=zvok_s2[:,1]
            else:
                yl=sum([zvok_s1[0:zvok_s2.shape[0],0],zvok_s2[:,0]],axis=0)
                zvok_s1[0:zvok_s2.shape[0],0] = sum([zvok_s1[0:zvok_s2.shape[0],0],zvok_s2[:,0]],axis=0)
                yd=sum([zvok_s1[0:zvok_s2.shape[0],1],zvok_s2[:,1]],axis=0)
                zvok_s1[0:zvok_s2.shape[0],0] = sum([zvok_s1[0:zvok_s2.shape[0],1],zvok_s2[:,1]],axis=0)
                yl=zvok_s1[:,0]
                yd=zvok_s1[:,1]
            y=sc.vstack((yl,yd))
            y=y.transpose()
        else:
            showwarning("Open file", "You didn't open two mono or stereo sounds\n")
        if (len(y) == 0):
            showwarning("Open file", "You didn't open two stereo sounds\n")
        else:
            sci.wavwrite(y,'stereo.wav',44100,'pcm16')
            open_plot_window(y)
```

Slika 16: Primer kode za zaporedno sestavljanje zvočnih datotek

Za tvorjenje naprednih odzivov (izraz napredno je pojasnjen v naslednjem podpoglavju) moramo najprej prebrati podatke o elevaciji, začetnem in končnem azimutu ter strani. Nato preverimo veljavnost vpisanih podatkov. Za vpisano višino potrebujemo še vmesne azimute. Le-te dobimo s pomočjo globalnih tabel *elev* in *elev40*. Prva vsebuje podatke o elevacijah in koraku med azimuti. Recimo mapa odzivov za elevacijo 20 stopinj vsebuje datoteke z azimutom s korakom 5 stopinj. Torej azimute 0, 5, 10, 15, 20, 25, …, 170, 175, 180. Druga tabela pa ima vpisane vse azimute za elevaciji -40 oziroma 40 stopinj, saj azimuti niso linearni. Ko preberemo še vmesne podatke, lahko naredimo izhodni zvok. Okno za kreiranje naprednih 3D zvočnih datotek je na sliki 17.



Slika 17: Okno za tvorbo naprednih zvočnih scen

5.1.5 Funkcija za napredno tvorjenje 3D zvoka

Izraz napredno tvorjenje v tej diplomski nalogi pomeni zvočne datoteke, na katerih je bilo po delih naenkrat uporabljenih več zaporednih odzivov. Na ta način simuliramo premikanje izvora zvoka. Funkcija za napredno tvorbo je na sliki 18. V postopku tvorjenja izhoda najprej odpremo zvočno datoteko in jo razdelimo na dele v odvisnosti od izbora začetnega in končnega azimuta ter razpona med njima. Nato nastavimo pot do odzivov s pomočjo prebrane elevacije. V tabeli imamo shranjene vse odzive, ki jih uporabimo za tvorjenje 3D zvoka. Pri tem pazimo na pravilne klice iz prebrane poti. Zvok dobimo s pomočjo konvolucije za levi in desni kanal; enako kot v glavnem delu programa.

```
def generate_output(middle_rsp):
    global advanced_rsp
    global mono_dir_w2
    global pos_str
    global counter
    #print middle_rsp
    try:
        zvok1, fs1, enc1 = sci.wavread(mono_dir_w2+combo11['value'])
    except IndexError:
        showwarning("Open file", "You didn't open both files")

    step = len(zvok1[0:(len(zvok1)/len(middle_rsp))])
    ind = step
    step_old=0
    data = [[],[]]
    zvok = zvok1[step_old:step]
    for i in range(0,len(middle_rsp)):
        path = os.getcwd()+'/rsp/elev'+advanced_rsp[0][0]+'/'
        rsp = path+'H'+str(advanced_rsp[0][0])+'e'+str(middle_rsp[i])+'a.wav'
        response, fs2, enc2 = sci.wavread(rsp)
        data2_d = response[:,0]
        data2_l = response[:,1]
        #print pos_str.get()
        if (pos_str.get()=='L'):
            yd=sc.convolve(data2_d,zvok)
            yl=sc.convolve(data2_l,zvok)
        else:
            yd=sc.convolve(data2_l,zvok)
            yl=sc.convolve(data2_d,zvok)
        y_=sc.vstack((yl,yd))
        data = sc.hstack((data,y_))
        step_old = step
        step = step+ind
        zvok = zvok1[step_old: step]
    data=data.transpose()
    sci.wavwrite(data, 'stereo.wav',44100, 'pcm16')
    open_plot_window(data)
    counter = 1
```

Slika 18: Funkcija za tvorbo naprednih 3D zvokov

5.1.6 Distribucija programa na operacijske sisteme Windows

Kot že omenjeno, je Python jezik, ki ga lahko uporabljamo na vseh operacijskih sistemih. Zato smo želeli preveriti delovanje in distribucijo programa na operacijskem sistemu Windows. Ugotovili smo, da program deluje tudi na drugem operacijskem sistemu. Problem predstavlja uporaba modula Os, ki uporablja funkcionalnosti operacijskega sistema. Te se seveda med Linux in Windows OS razlikujejo. Zaradi tega smo v program dodali globalno spremenljivko OS, na podlagi katere kličemo različne ukaze odvisne od OS. Nato smo s pomočjo modula Py2exe delali distribucije skript. To naredimo tako, da napišemo nov program v Pythonu. Slika 20 prikazuje primer programa za distribucijo na Windows OS. Kličemo ga iz Python ukazne vrstice z parametrom py2exe. V programu uvozimo module, potrebne za prevajanje, ter napišemo poti za podatkovne datoteke (grafične knjižnice).

*D:\dropbox\main_window-1.3-64bit\py2exe.py - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X

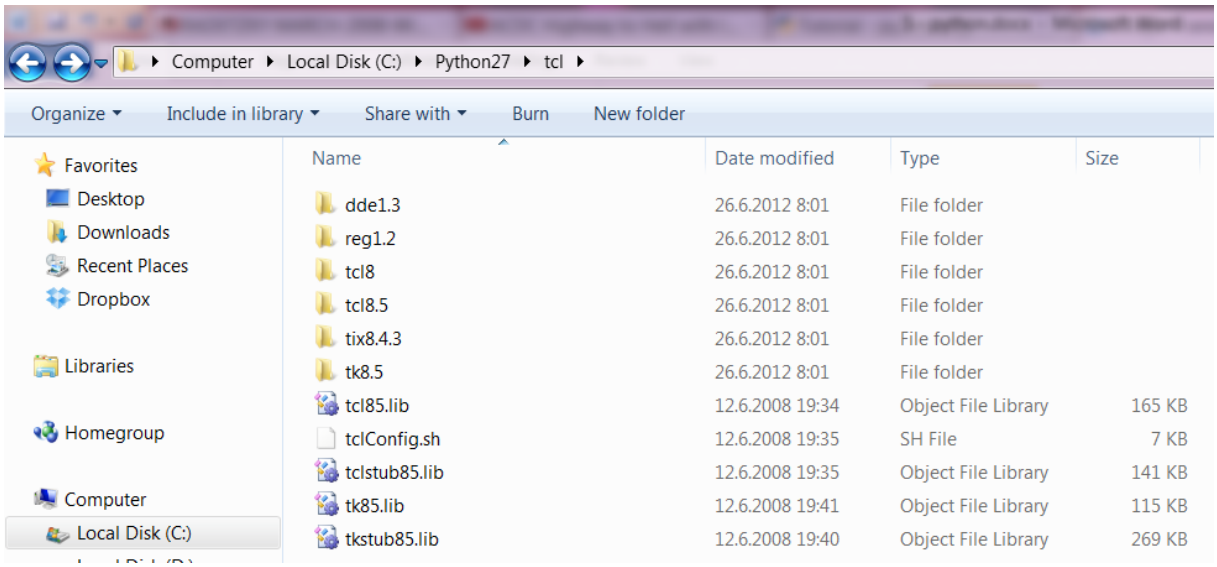
py2exe.py

```
1  #import modulov
2  #zaženemo v python ukazni lupini z ukazom python ime_skripte.py py2exe
3  from distutils.core import setup
4  import py2exe
5  from distutils.filelist import findall
6
7  def files(folder):
8      for path in glob.glob(folder+'/*'):
9          if os.path.isfile(path):
10             yield path
11 data_files=[
12     ('.', glob.glob(sys.prefix+'/DLLs/tix81*.dll')),
13     ('tcl/tix8.1', files(sys.prefix+'/tcl/tix8.1')),
14     ('tcl/tix8.1/bitmaps', files(sys.prefix+'/tcl/tix8.1/bitmaps')),
15     ('tcl/tix8.1/pref', files(sys.prefix+'/tcl/tix8.1/pref')),]
16
17 setup(console=['main_interface.py'],    #ime programa
18     data_files=matplotlib.get_py2exe_datafiles(),
19     name='3D_sound',
20     version='1.3',
21     description="main window",
22     author="Matej",
23     )
```

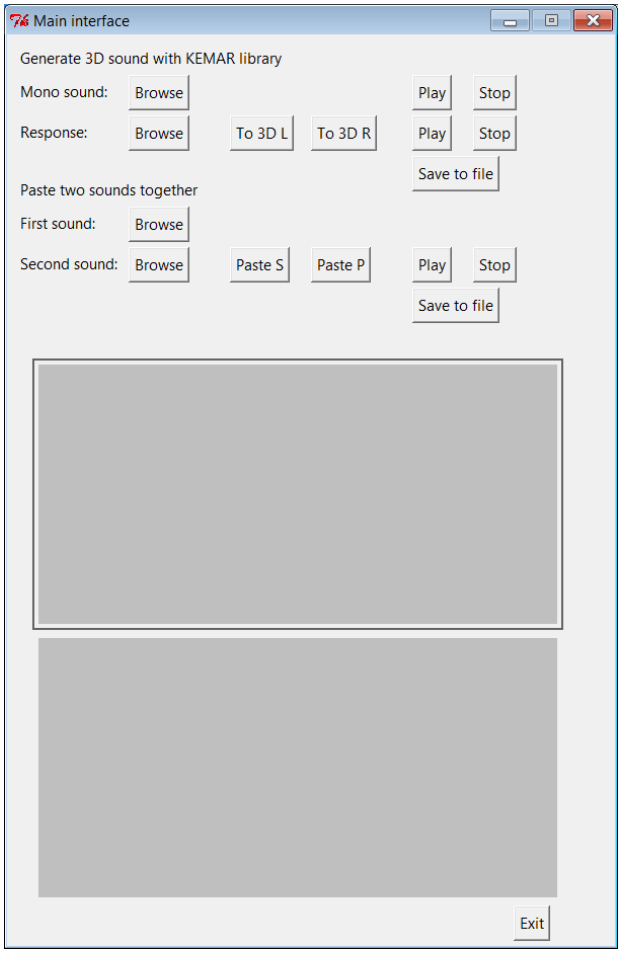
Python file | length : 722 | lines : 23 | Ln : 1 | Col : 16 | Sel : 0 | 0 | Dos\Windows | ANSI | INS

Slika 19: Prikaz kode programa za kreiranje distribucije

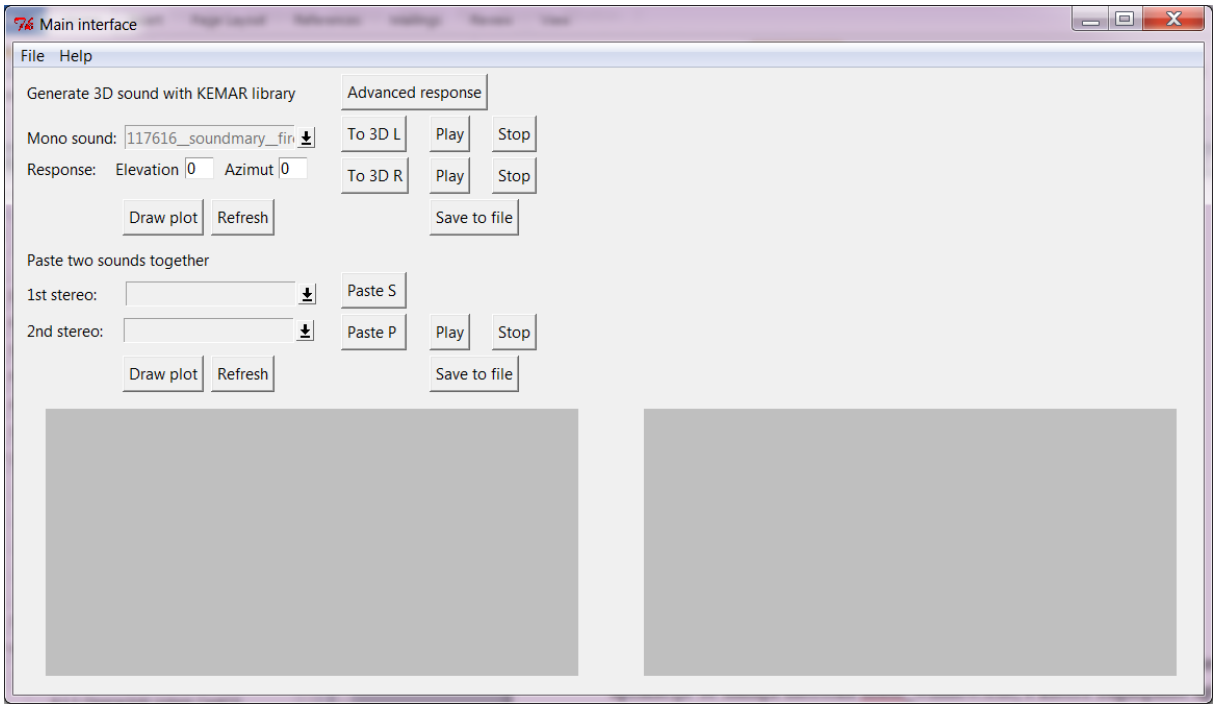
Narediti smo morali dve različici za 32- in 64-bitne Windows OS, saj skripte potrebujejo različne knjižnice za delovanje. Rezultat sta dve mapi build in dist. Mapa build je, medtem ko se aplikacija pakira, delovna. Po končani distribuciji jo lahko izbrišemo. Datoteke v drugi mapi pa so potrebne za zagon programa. Po tem, ko smo dodali v program elemente iz modula Tix, smo morali dodati v mapo tcl tudi vse knjižnice potrebne za zagon teh elementov. Vse potrebne mape vidimo na sliki 20. Na slikah 21 in 22 pa sta aplikaciji dobljeni s pomočjo modula py2exe (prva in zadnja različica).



Slika 20: Knjižnice za Windows okolje



Slika 21: Prva verzija programa

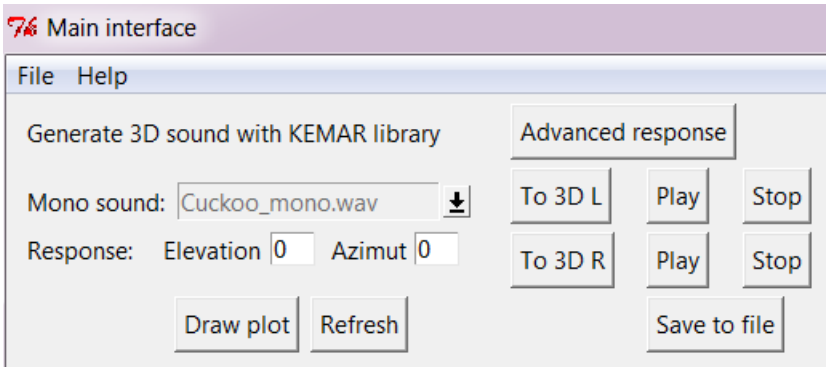


Slika 22: Zadnja verzija programa

Poglavje 6

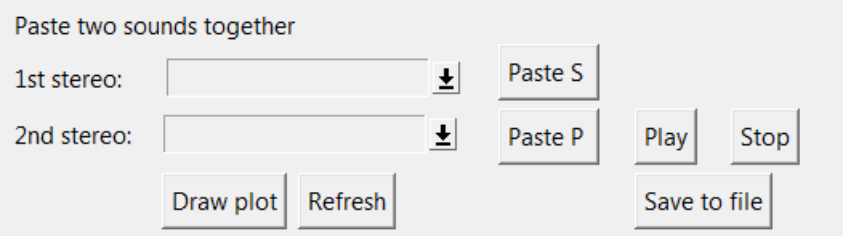
Uporaba aplikacije za tvorbo 3D zvočnih scen

V mapi dist aplikacije (glej podpoglavje 5.1.6) se nahaja datoteka main_window.exe, s katero zaganjamo aplikacijo. Glavno okno vsebuje meni z dvema izbirama in sicer *file* ter *help*, kar prikazuje slika 23. Prva je izbira mape z mono zvočnimi datotekami, stereo zvočnimi datotekami ter izhod iz aplikacije. Druga pa dostop do pomoči za uporabo aplikacije. Pod menijem je sklop gumbov za tvorjenje enostavnih 3D zvokov (poimenovali smo ga *Generate 3D sound with KEMAR library)*. Za enostavne 3D zvoke velja, da je na eni zvočni datoteki uporabljen natanko en odziv. Da dobimo enostavne 3D zvočne datoteke, moramo najprej izbrati mono zvočno datoteko. V podmapo mono, kjer se aplikacija nahaja, lahko dodamo svoje datoteke. Da jih vidimo v aplikaciji, moramo vrednosti v seznamu osvežiti z gumbom z napisom *refresh*. Lahko pa tudi sami izberemo mapo z zvočnimi datotekami v meniju (izbira te mape je opisana v uvodu tega poglavja). Po izboru mape in zvočne datoteke v seznamu, vpišemo elevacijo in azimut v stopinjah v polji za besedilo. Nato z uporabo gumba z napisom »*To 3D L*« oziroma »*To 3D R*« ustvarimo novo 3D zvočno datoteko. Če želimo uporabiti impulzni odziv za levo uho, to naredimo z gumbom z napisom »*To 3D L*«, za desno pa uporabimo gumb z napisom »*To 3D R*«. Zvočne datoteke lahko poslušamo. Zgornji gumb z napisom *play* (glej sliko 23) predvaja trenutno izbrano mono zvočno datoteko. Gumb pod njim pa predvaja nazadnje ustvarjeno 3D zvočno datoteko. Ob vsakem od teh gumbov je gumb z napisom stop, ki vsako posamezno predvajanje ustavi. Po tvorbi 3D zvočnih datotek lahko narišemo časovne prikaze signalov z uporabo gumba z napisom »*Draw plot«* ali pa shranimo tvorjeno 3D zvočno datoteko (gumb z napisom »*save to file*«).



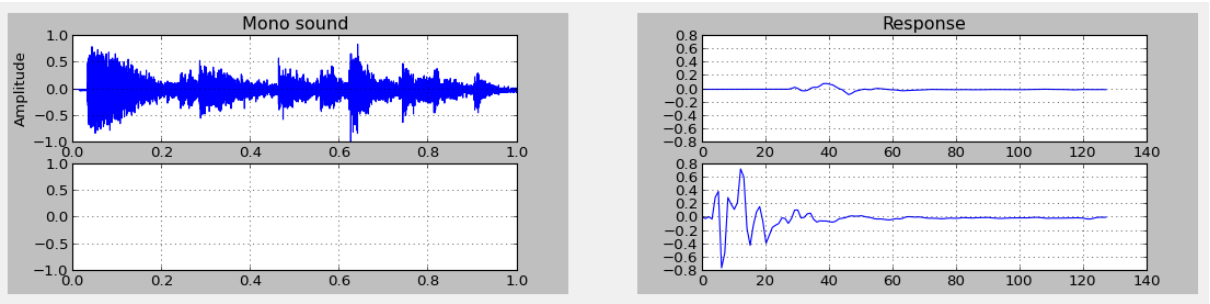
Slika 23: Sklop gumbov za tvorjenje enostavnih 3D zvokov

Osrednji del aplikacije, prikazan na sliki 24, je namenjen sestavljanju zvočnih datotek. Vsebuje dva seznama za izbor zvočnih datotek. Če ju želimo izbrati, moram najprej v meniju izbrati mapo z zvočnimi datotekami (glej začetek poglavja meni *file*). Po izboru dveh, lahko narišemo časovni prikaz signalov. To naredimo z gumbom, ki ima napis »*Draw plot*«. Z gumboma z napisi »*Paste S*« in »*Paste P*« sestavljamo zvočne datoteke. Zgornji gumb (*Paste S*) sestavi skupaj dve zvočni datoteki eno za drugo. Postavi ju v sekvenco. Spodnji gumb (*Paste P*) pa zvočni datoteki postavi eno ob drugo, torej vzporedno. Gumb z napisom »*Refresh«* ponastavi vsebino iz trenutno izbrane mape z stereo zvočnimi datotekami. Enako kot v zgornjem sklopu imamo možnost predvajanja nazadnje sestavljene zvočne datoteke. Prav tako je poleg tega gumba tudi stop gumb ter gumb za shranjevanje zvočne datoteke.



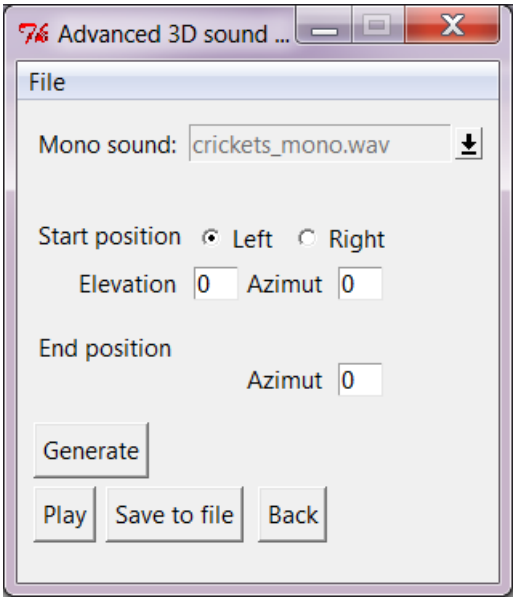
Slika 24: sklop gumbov za sestavljanje zvočnih datotek

Spodnji del je brez gradnikov grafičnega vmesnika. Vsebuje dve risalni površini, ki narišeta časovni prikaz signalov. Oba časovna prikaza sta na sliki 25. Ob kliku na gumb za risanje v zgornjem delu aplikacije, dobimo izris signala trenutno izbrane mono zvočne datoteke na levi strani in vpisanega odziva na desni strani. Če kliknemo na gumb za izris signala v delu aplikacije, kjer sestavljamo zvočne datoteke, dobimo na levi strani prvo na desni pa drugo izbrano zvočno datoteko.



Slika 25: Risalna površina aplikacije

Na voljo imamo tudi napredno tvorbo zvokov, ki je na zgornjem delu aplikacije, z napisom »*Advanced response*«. Ob uporabi gumba se nam odpre novo okno, ki je na sliki 26. Vsebuje meni, ki ima podobno kot glavno okno možnost izbora mape z mono zvočnimi datotekami (»*Change mono source*«) in možnost zaprtja okna (*back*). Pod menijem je seznam mono zvočnih datotek, v katerem si izberemo želeno. Začetni položaj izvora zvoka, ki ga bomo uporabili za tvorjenje 3D zvoka, določimo tako, da označimo »*Left«* oziroma »*Right«*. Če izberemo možnost »*Left*« bomo začeli tvoriti 3D zvok za levo uho, če »*Right*« pa za desno. Določiti moramo še začetno elevacijo in azimut v stopinjah (polja ob napisu »*Elevation*« in »*Azimut«*) ter končni azimut v stopinjah (polje ob napisu »*End position azimut«*). Ob kliku na gumb za tvorjenje (z napisom »Generate«) ustvarimo napredni 3D zvok, ki je tvorjen iz več zaporednih odzivov.



Slika 26: Okno za napredno tvorbo 3D zvočnih datotek

Poglavje 7

Zaključek, sklepne ugotovitve

Cilj diplomske naloge je bil sprogramirati aplikacijo z grafičnim vmesnikom, ki tvori in predvaja 3D zvoke s pomočjo knjižnice MIT-KEMAR. Po analizi dokumentacije knjižnice smo najprej naredili enostavno rešitev v okolju Scilab, s katero smo preverili delovanje izbranih metod in knjižnice.

Zgoščena knjižnica MIT-KEMAR vsebuje 354 datotek z odzivi, katere uporabimo bodisi kot leve bodisi kot desne odzive. S tem si poenostavimo in zmanjšamo delo z vhodnimi datotekami.

Izbira virtualizacije z Oracle VirtualBox in programskega okolja Python se je izkazala kot zelo dobra rešitev. Čeprav smo morali najprej postaviti CentOS operacijski sistem, smo le minimalen čas namenili njegovi konfiguraciji. Potrebno je bilo nastaviti omrežje za dostop in namestiti potrebne module za že vsebovan Python programski jezik v sistem. Z moduli nam je bilo olajšano branje in shranjevanje zvočnih datotek, uvoz parametrov in podatkov iz datotek v vektorje in matrike, sestavljanje matrik s podatki iz vhoda ter shranjevanje podatkov na izhod.

Ker pa smo se prvič srečali s tem programskim jezikom, smo več časa potrebovali za programiranje.

Grafični vmesnik smo pripravili za Linux in Windows OS, s čimer smo zagotovili širšo uporabo aplikacije. Po tvorjenju nekaj enostavnih zvočnih shem ugotovimo, da s knjižnico ustvarimo dober približek realnih razmer. Ob tvorjenju naprednih zvokov pa zaradi množice odzivov pride bolj do izraza slabost tega pristopa, ki je večja računska in prostorska zahtevnost.

Literatura

[1] (2011) Antonie Lefebvre, "An introduction to Python programing with NumPy, SciPy, and Matplotlib/Pylab", dostopno na http://koen.me/research/teaching-asci/python-workshop-slides.pdf

[2] (1994) Bill Gardner and Keith Martin, "HRTF Measurements of a KEMAR Dummy-Head Microphone", dostopno na http://sound.media.mit.edu/resources/KEMAR/hrtfdoc.txt

[3] (2013) Fred L. Drake, Jr, "Python documentation", dostopno na http://docs.python.org/2/

[4] (2003) Jens He, "Impulse response measurement using MLS", dostopno na http://jenshee.dk/signalprocessing/mls.pdf

[5] (2013) John W. Shipman, "Tkinter 8.5 reference: a GUI for Python", dostopno na http://infohost.nmt.edu/tcc/help/pubs/tkinter.pdf

[6] (2010) Konzorcij Scilab (Michaël Baudin), "Introduction to Scilab", dostopno na http://www.scilab.org/content/download/247/1702/file/introscilab.pdf

[7] (2011) NumPy comunity, "NumPy reference ", dostopno na http://docs.scipy.org/doc/numpy/numpy-ref.pdf

[8] (2012) Več avtorjev, "Oracle VM VirtualBox User Manual", dostopno na http://dlc.sun.com.edgesuite.net/virtualbox/4.2.0_BETA1/UserManual.pdf

[9] (2011) Več avtorjev, "Python py2exe tutorial", dostopno na http://www.py2exe.org/index.cgi/Tutorial