Tallinna Ülikool

Digitehnoloogiate Instituut

Heli manipuleerimine veebilehel Web Audio API’t kasutades

Bakalaureusetöö

Autor: Kevin Münter

Juhendaja: Andrus Rinde

Autor: ..... „ ..... „2016

Juhendaja: ..... „ ..... „2016

Instituudi direktor: ..... „ ..... „2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

----------	----------
(kuupäev)	(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)

(autori nimi)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   _____
   _____
   _____

   (*lõputöö pealkiri*)

   mille juhendaja on _____,

   (*juhendaja nimi*)

   säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas/Haapsalus/Rakveres/Helsingis, _____

(digitaalne) allkiri ja kuupäev

Sisukord

SISSEJUHATUS **5**

1 WEB AUDIO API **7**

1.1 Varasemad võimalused heli manipuleerimiseks veebilehel 8

1.2 Web Audio API ülesehitus 10

1.3 Audio kontekst 11

1.4 Heli sõlmed 12

1.5 Modulaarne marsruutimine 14

1.6 Heli allikas 15

1.6.1 Sagedusgeneraator *15*

1.6.2 Helifailid *16*

1.7 Filtrid 17

2 WEB AUDIO API KASUTAMINE VEEBILEHEL **19**

2.1 Sagedusgeneraatori kasutamine 19

2.2 Audio puhver 21

2.3 Heli otse failist 22

2.4 Mikrofoni sisendi kasutamine 23

2.5 Web audio API analüüsimisvahend 23

2.6 Näidete kokkuvõte 24

KOKKUVÕTE **25**

SUMMARY **26**

KASUTATUD KIRJANDUS **27**

LISAD **30**

Lisa 1 Fikseeritud sagedusega sagedusgeneraator 30

Lisa 2 Fikseeritud sagedusega neli erinevat helilainet 31

Lisa 3 Muudetava sagedusega ja helitugevusega sagedusgeneraator 33

Lisa 4 Audio puhver 35

Lisa 5 Audio puhver koos helitugevuse ja kiiruse muutmisega 36

Lisa 6 Audio puhvri heli läbi filtri 37

Lisa7 Kolm erinevat heli kolmest puhvrist 39

Lisa 8 Heli otse failist 41

Lisa 9 Mikrofoni signaali kasutamine 42

Lisa 10 Analüüsimisvahend 42

Sissejuhatus

Erinevate heliklippide kasutamine on muutunud igasuguse rakenduste, operatsioonisüsteemide lahutamatuks osaks. Ka veebilehtedele on heliklippe alati lisada püütud, takistuseks on siinjuures olnud erinevad tehnoloogilised piirangud. HTML5 on muutnud ka heli veebilehtede tavapäraseks osaks.

Nii nagu töölauarakenduste puhul, nii ka veebirakenduse kasutajaliideses peab kasutaja saama määrata, kuidas heliklippe mängitakse, reguleerida helitugevust ja palju muud. Varasemad tehnoloogiad võimaldasid heliklippe kasutada vaid taustaks ning see võis inimesed ära tüüdata. Et seda vältida, on arendatud vahendeid, et veebilehe külastaja saaks ise muuta, kas heli mängib või mitte, kui kõvasti see mängib ning veel palju võimalusi. Arendamise käigus on jõutud välja lausa veebipõhiste helitöötlusvahenditeni.

Probleemina nägi autor seda, et uut tehnoloogiat, mis võimaldab kasutajal heli manipuleerida veebilehtedel, ei tunta. Varem tuli heli kasutamiseks veebilehtedel kasutada erinevaid vahendeid, näiteks *Flash* või *QuickTime*, mis võimaldasid vähesel määral heliga manipuleerida kuid vajasid pistikprogramme. Hetkel kasutatakse laialdaselt HTML5 <*audio*> elementi, kuid sellega saab heli vähesel määral manipuleerida. Hiljuti arendati välja uus vahend nimega *Web Audio API*, mida pidevalt täiendatakse ning mis kõrvaldab eelmiste vahendite paljud puudused ja lisab veel erinevaid võimalusi juurde.

Selles töös uurib autor, kuidas saab lisada ja manipuleerida helisid veebilehel kasutades *Web Audio API*'t. Milliseid võimalusi pakub hiljuti välja tulnud ning pidevalt arendamises heli manipulatsioonivahend. Lisaks näitab autor erinevate näidetega, kuidas kasutada vastavat vahendit veebilehel, et selle töö lugeja saaks ettekujutuse, kui keeruline või lihtne heliga seonduvaid toiminguid teha on.

Teema valiku aluseks on autori huvi teada saada, kuidas saab manipuleerida heli veebilehtedel hiljuti välja tulnud *Web Audio API*'ga ning seda informatsiooni teistega jagada.

Sihtgrupiks on selle töö puhul kõik inimesed, kellel on huvi veebilehtede arendamise ning selle juures *Javascript*'i kasutamise vastu.

Töö eesmärgiks on anda ülevaade *Web Audio API* võimalustest ning kasutamisest. Lühidalt tutvustada, millised olid esimesed võimalused heli kasutamiseks veebilehtede. Lisaks testida mõningaid olulisemaid vahendeid ning anda lugejale juhiseid, kuidas neid vahendeid kasutada.

Eesmärkide saavutamiseks annab autor kirjanduse põhjal ülevaate *Web Audio API* vahenditest ning kasutamisest veebilehel. Peale selle loob ise näited, kus kasutab erinevaid võimalusi, mida *Web Audio API* pakub. Lõpuks selgitab lugejale, kuidas samasuguseid veebilehti ise saab luua.

1 Web Audio API

2014 aastal hakati brauserite loojate poolt täielikult toetama HTML5 <*audio*> elementi, mis võimaldab kõikides tänapäevastes brauserites heli mängida. (Smus, Web Audio API, 2013)

HTML5 <*audio*> elemendi kasutamine laseb kasutajal heliklipi esitust teatud määral kontrollida. Kasutaja saab määrata korduste arvu, kas kuvada kontrollnupud (*play* ja *pause* nupp), kas heli laadida siis, kui veebileht laeb, kas heli hakkab automaatselt kohe mängima või mitte.

HTML5 <*audio*> elemendiga lisatud heliklippide manipuleerimiseks on mitmeid kordi proovitud teha piisavalt võimast *API't*. Üks esimesi märkimisväärsemaid proove nende piirangute vähendamiseks on *Audio Data API*. See on prototüübitud ja disainitud *Mozilla Firefox*'is. (Smus, Web Audio API, 2013)

Mozilla alustas <*audio*> elemendiga ning laiendas selle *JavaScript*'i *API*'t lisades erinevaid lisavalikuid. Sellel *API*'l on limiteeritud audio graafik (erinevate sõlmede kogus audio kontekstis on piiratud). Alates aastast 2013 soositakse *Firefox*'is *Web Audio API*'t. (Smus, Web Audio API, 2013)

Web Audio API'ga alustati tööd 25 märtsil 2011, kui alustas tööd grupp nimega *Audio Working Group*, mille eesmärgiks oli arendada täiendatud audio võimalustega *API* kliendipoole skript. (W3C, 2011)

Web Audio API on kõrgetasemeline *JavaScript*'i rakenduste programmeerimise kasutajaliides (*Application Programming Interface*), mida saab kasutada heli töötlemiseks ja sünteesimiseks veebiprogrammides. Heli töötlemisega tegeleb *Assembly*/C/C++ kood brauseri sees, kuid *API* laseb kasutajal seda kontrollida *JavaScript*'i abil. (Metivier, Web Audi API Basics, 2014)

See *API* laseb kasutajal genereerida või manipuleerida mistahes olemasolevat heli. See on võimas vahend, sellel on olemas isegi enda ajastussüsteem, mis võimaldab poolesekundilise täpsusega taasesitamist See tähendab, et aega mõõdetakse *API*-s sekundites, kuid kõiki ajaga toimuvaid toiminguid saab komakoha täpsusega paika panna. (Memo, 2015) Vastava *API* põhiline eesmärk on lisada kasutajatele veebilehtedele samad võimalused heli manipuleerimiseks, mida leiab kaasaegsete mängude audiomootorites ning lisaks mõned

miksimis-, töötlemis- ja filtreerimistööd, mis leidub heli töötlemisprogrammides. (Smus, Getting Started with Web Audio API, 2013)

1.1 Varasemad võimalused heli manipuleerimiseks veebilehel

Tänapäeval on heli muutunud täiesti tavaliseks osaks veebilehest. Ajalooliselt sai heli veebilehele lisada vaid taustamuusikana ning kasutajal puudus selle üle kontroll. Hiljem võeti kasutusele pistikprogrammid, mis võimaldasid kasutajal vähesel määral heliga manipuleerida.

Üks esimesi viise, kuidas sai panna veebilehtedel heli taustaks mängima, oli <*bgsound*> elemendi kasutamine. See element saabus koos *Internet Explorer*'iga aastal 1995 See tähendab seda, et niipea, kui veebileht avati, hakkas ka muusika mängima ning jäigi mängima. Kasutajal puudus kontroll taustamuusika üle, kas seda peatada või esitada. See element oli saadaval ainult Internet *Explorer*'il ning seda ei standardiseeritud ning ükski teine veebibrauser ei võtnud seda üle. (Smus, Web Audio API, 2013)

Sellel elemendil on neli atribuuti: (Contributors, <bgsound>, 2016)

- *balance* - määratakse stereobalanss kõlarite vahel väärtustega -10 000 kuni +10 000
- *loop* - määratakse, mitu korda taustaheli korratakse. Väärtus *infinite* paneb igavesti korduma
- *src* – määratakse kasutatava helifaili asukoht. Failitüübiks saab olla vaid *.wav*, *.au* või *.mid*
- *volume* - määratakse helitugevus -10 000 ja 0 vahel.

Nendest atribuutidest on kohustuslik vaid *src* atribuut, sest sellega määratakse mängitava helifaili asukoht.

Kood, mida kasutatakse, et lisada <*bgsound*> elemendiga taustaheli, näeb välja selline (Koodinäide 1):

```
<bgsound src="sound2.wav" balance="1000" volume="500" loop="infinite">
```

Koodinäide 1 <*bgsound*> elemendi kasutamine

Netscape'il oli sarnane element nimega <*embed*>, mis tuli välja koos *Netscape Navigator* veebilehitsejaga aastal 1994, mille kasutamise illustreerib järgnev koodinäide (Koodinäide 2):

```
<embed src="hello.wav" />
```

Koodinäide 2 <*embed*> elemendi kasutamine

Selle koodiga lisati helifail veebilehele ning kuvati ka mõned heli kontrollivad nupud. Mis nupud täpsemalt, olenes sellest, millist pistikprogrammi kasutaja kasutas. Seega veebilehe looja ei saanud täpselt määrata, mis nupud kuvatakse tema lehel (Joonis 1). (Oliver, 1999)

Joonis 1 <embed> elemendi nupud internet explorer'il

Lisaks <*bgsound*> elemendile oli ka <*object*> element, mis tuli koos HTML4'ga 1997, kuid HTML5 kasutuselevõtuga kaotas see element palju atribuute. HTML5'ga töötavad atribuudid on: (Contributors, <object>, 2016)

- *data* – määrab ära faili asukoha, mida tahetakse veebilehele lisada
- *form* – määrab ära, millisesse klassi lisatav objekt kuulub
- *height* – määrab ära objekti kõrguse
- *name* – määrab ära objekti nime
- *type* – määrab ära, mis tüüpi andmeid soovitakse lisada
- *usemap* – määrab ära, millist pilti kasutatakse objekti näitamiseks veebilehel
- *width* – määrab ära objekti laiuse pikslites

Kood, millega saab kasutada <*object*> elementi, on järgmine (Koodinäide 3).

```
<object width="400" height="400" data="helloworld.swf" name="hello"></object>
```

Koodinäide 3 <*object*> elemendi kasutamine

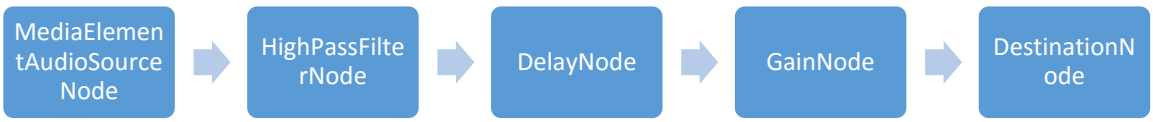
Aastal 1996 tuli välja *Macromedia Flash* 1.0. *Flash* võimaldas esimesena kõikidel veebilehtedel taustaheli mängida ning manipuleerida. Sellel oli kahjuks ka tagasilööke. Näiteks *Flash* vajas mängimiseks eraldi pistikprogramm. (Smus, Web Audio API, 2013)

1.2 Web Audio API ülesehitus

Selleks, et kasutada *Web Audio API*'t, peab kõigepealt looma konteineri, mille nimeks on *AudioContext* (edaspidi audio kontekst). Meeles peab pidama seda, et vaja läheb tavaliselt vaid ühte audio konteksti, kuid luua võib mitu ning sinna sisse saab laadida mistahes arv helisid. (Metivier, Web Audi API Basics, 2014)

Audio kontekst on vahelüli heliallika ja lõpliku väljundi vahel. Seda võib võrrelda, kui ühendust kitarri ja kõlari vahel. Näiteks kitarrist ühendab oma kitarri juhtme abil efektiseadme külge, mis muudab heli. Seejärel ühendab Ta järgmise juhtme seadme küljest, kas järgmise efekti külge või siis kõlari külge. Tulemuseks on see, et kui kitarrist mängib, siis heli signaalid liiguvad kitarrist efektiseadmeni, kus neid muudetakse ning edasi liigub juba muudetud helisignaal, kas järgmisse seadmesse või kõlarisse ning kuulajaskond kuuleb muudetud heli. Samamoodi töötab ka *Web Audio API*. (Memo, 2015)

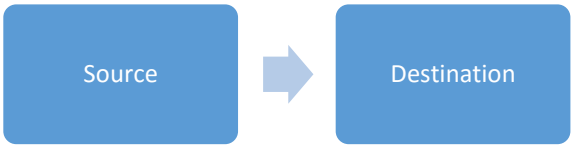
Kõik heliga seotud toimingud, manipulatsioonid toimuvad audio konteksti sees. Põhilised toimingud tehakse niinimetatud audio sõlmedega (*audio node*), mis on omavahel ühendatud ning moodustavad heli marsruutimise graafi. Mitmeid heli allikaid, mis on erinevat tüüpi, saab kasutada ühe audio konteksti sees. See laseb kasutajal luua komplekssseid heli funktsioone ja dünaamilisi efekte. (Joonis 2) (Contributors, Web Audio API, 2016)



Joonis 2 Heli liikumine läbi mitme sõlme

1.3 Audio kontekst

Web Audio API on ehitatud audio konteksti idee ümber. Audio kontekst on heli sõlmedest moodustatud suunatud graaf, mis näitab ära, kuidas helisignaal liigub allikast lõppsõlme. Siis, kui heli liigub läbi erinevate sõlmede, võib selle omadusi muuta. Kõige lihtsam audio kontekst on otseühendus algsõlmest lõppsõlme (Joonis 3). (Smus, Web Audio API, 2013)



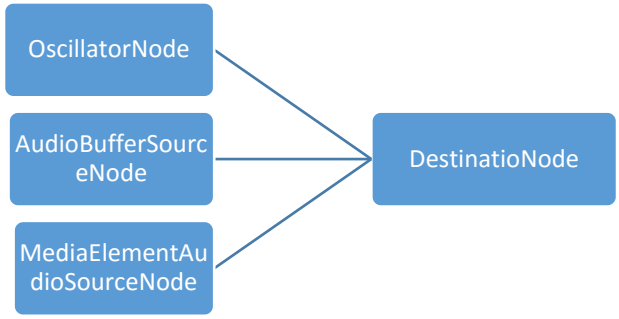
Joonis 3 Audio kontekst kahe sõlmega

Tavaline tööjada, kuidas veebilehele heli lisada failist, oleks selline: (Joonis 4) (Contributors, Web Audio API, 2016)

- Luuakse audio kontekst
- Selle sees luuakse allikas, näiteks helifail
- Siis lisatakse erinevaid filtreid
- Peale seda valitakse väljund, näiteks kõlar
- Lõpuks ühendatakse allikad efektisõlmedega ning need omakorda väljundiga. Ühendamine toimub *connect* käsklusega

Joonis 4 Audio kontekst koos efektiga

Kontekst võib olla ka väga keerukas, kuna allika ja lõpu vahel võib olla väga palju sõlmi, mis muudavad, analüüsivad või sünteesivad heli enne väljundisse jõudmist. Kui näiteks sõlme tuleb mitu sissetulevat heli, siis *API* lihtsalt mängib neid kõiki korraga (Joonis 5). (Smus, Web Audio API, 2013)

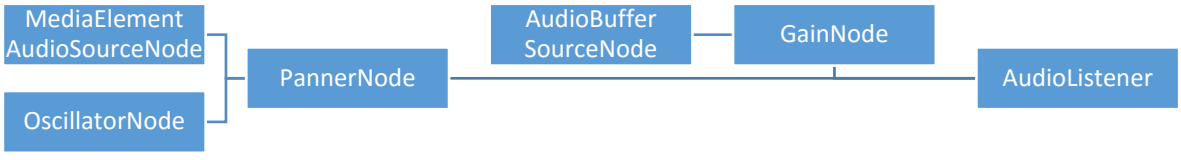


Joonis 5 Mitu allikat ühte lõppsõlme

1.4 Heli sõlmed

Kogu audio kontekst koosneb heli sõlmedest (*AudioNode*), seega saab öelda, et heli sõlmed on audio konteksti põhitalad. Sõlmedeks loetakse ka heli allikat ning väljundit. Minimaalselt saab sõlmi olla kaks ning maksimumi piiri neil ei ole. Minimaalselt on vaid allikasõlm ja lõppsõlm ning otseühendus nende vahel.

Heli sõlmede (*AudioNode*) kasutajaliides on üldine kasutajaliides, mis esindab helitöötlusmoodulit, nagu näiteks heli allika element. (Joonis 6) (Contributors, AudioNode, 2015)



Joonis 6 Üks võimalik heli marsruutimise graafik

Heli sõlmel on sisendid ja väljundid, igaühel etteantud arv kanaleid. Sõlm, millel pole ühtegi sisendit ja üks või mitu väljundit, on allika sõlm (*SourceNode*). Iga sõlm töötleb heli erinevalt, kuid põhimõte on selline: sõlm loeb sisendit, teeb helitöötlust ning genereerib uue

signaali, mille ta väljutab, või lihtsalt laseb heli endast läbi, olenevalt sõlmest. (Contributors, AudioNode, 2015)

Sõlmi, mida saab luua, on nelja erinevat tüüpi: (Smus, Web Audio API, 2013)

- **Allikate sõlmed** (*Source nodes*) - heli allikad nagu näiteks audio puhvrid, otse heli sisestused, <*audio*> elemendid, sagedusgeneraatorid ja *JavaScript'*i protsessorid
- **Modifitseerimise sõlmed** (*Modification nodes*) - filtrid, *JavaScript'*i protsessorid
- **Analüüsimise sõlmed** (*Analysis nodes*) - analüüsijad ja *JavaScript'*i protsessorid
- **Sihtpunkti sõlmed** (*Destination nodes*) - heli väljundid ja ühenduseta töötlemise puhvrid

Allikad ei pea olema alati failipõhised, vaid võivad olla ka reaalaja sisendid, näiteks muusikainstrumendid või mikrofon. Või näiteks täielikult sünteesitud heli. (Smus, Web Audio API, 2013)

Modifitseerimissõlmi on palju erinevaid. Nendest saadetakse helisignaal läbi ning, kui see juhtub, siis vastavalt sellele, milliseid sõlmi läbitakse, heli muutub. Järgnevalt mõned modifitseerimissõlmed mida kasutatakse: (Metivier, Web Audi API Basics, 2014)

- ***GainNode*** **-** võimendab sissetuleva heli signaali
- ***DelayNode*** **-** viivitab sissetuleva heli signaali ehk kui signaal tuleb sõlme sisse siis väljub ta sealt vastavalt kasutaja ette antud sekundite pärast
- ***AudioBufferNode*** **-** salvestatakse lühikesed heliklipid, tavaliselt lühemad, kui 45 sekundit. Seda kasutades ei pea lehe korduval kasutamisel heli uuesti laadima, vaid see asub puhvris ning hakkab sealt mängima
- ***PannerNod*** **-**stereobalanss, mis paneb paika heli allika asukoha, selle liikumise ja suuna ruumis
- ***ConvolverNode*** **-** lisab helile järelkaja, nagu heli põrkuks erinevatelt objektidelt tagasi, näiteks kontserdisaal
- ***DynamicsCompressorNode*** **-** lisab dünaamilise kompressiooni efekti ehk vähendab heli signaalis liiga kõrgel olevaid helisid et neid paremini kuulda oleks.
- ***BiQuadFilterNode*** **-** lisab lihtsaid filtreid, toonide kontrollimisvahendeid ja graafilisi ekvalaisereid

- ***WaveshaperNode*** **-** lisab helisignaalile moonutusi

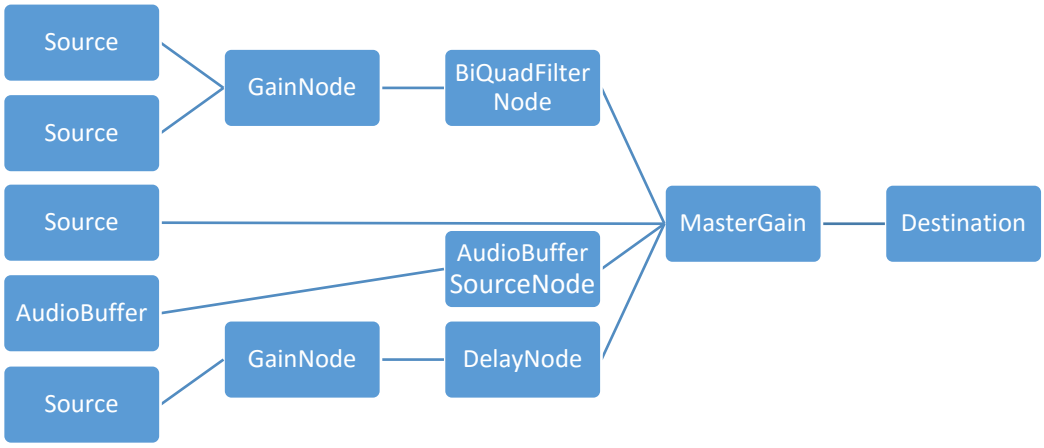
1.5 Modulaarne marsruutimine

Modulaarne marsruutimine (*Modular Routing*) laseb kasutajal muuta erinevate sisendite helisid vastavalt sellele, kuidas ta ise soovib. Näiteks, mängudel ühendatakse erinevad helid kokku, et soovitud tulemus saada. Allikateks oleksid näiteks taustamuusika, mängu heliefektid, *UI* (*User Interface*) helid ning kaasmängijate vestlused. *Web Audi API* eraldab erinevatest allikates tulnud helid ning laseb kasutajal igat heli muuta eraldi, või kõiki koos. (Smus, Web Audio API, 2013)

Kui kasutaja ei soovi kuulata taustamuusikat, saab ta selle eraldi välja lülitada, ilma, et lülitaks välja mängu heliefektid. Kui kõik helid on liiga valjud, saab kõikide helide tugevust ka korraga vaiksemaks keerata. (Smus, Web Audio API, 2013)

Näitena toob autor siinkohal konteksti, mis võtab heli viiest erinevast allikas, muudab kahe helitugevust eraldi. Kahel allikal on ka efekti sõlm ning kõigi helitugevust saab ka korraga muuta. Sobib väga hästi mängule, kus on kahe erineva tegelase hääl, mida peab moonutama. Lisaks on taustamuusika, mis tuleb läbi puhvri, mingisuguse masina hääl ning tegevuse hääl, mis peab saabuma kuulajani viivitusega.

All oleval joonisel (vt Joonis 7) on välja toodud viis heli allikat, mis erinevat teed pidi jõuavad *MasterGain* sõlme. Ülevalt alustades on kaks allikat, mis lähevad kokku ühte *GainNode'*i, kus saab nende helide signaali tugevust muuta. Seejärel liigub signaal edasi *BiQuadFilterNode'*i, kus sellele lisatakse kasutaja poolt valitud filter. Seda läbides jõuab signaal *MasterGain'*i. Nendeks allikateks sobivad hästi sagedusgeneraatorid, mis koos loovad polüfoonilise heli. Kolmas allikas liigub otse *MasterGain'*i. See oleks taustamuusika, sest seda ei ole vaja muuta. Neljandaks on audio puhvrist tulev signaal, mis liigub ka otse *MasterGain'*i. See oleks lühike heliklipp, näiteks relva lask. Viimane signaal tuleb läbi *GainNode'*i ja *DelayNode'*i ning jõuab *MasterGain'*i. Selleks allikaks sobib inimese hääle kaja. *MasterGain* sõlmes saab kõikide kokkutulnud signaalide tugevust muuta ning seejärel siirdub signaal lõppsõlme, milleks tavaliselt on kõlar.



Joonis 7 Mitmest allikast heli liikumine ühte lõppsõlme

1.6 Heli allikas

AudioSourceNode esindab heli allikat. See on heli sõlm (*AudioNode*), millel pole ühtegi sisendit ning on üks väljund. Põhiliselt on olemas kahte tüüpi allikasõlmi. (Metivier, Web Audi API Basics, 2014)

- ***Oscillator*** -sagedusgeneraator, mis genereerib kindla sagedusega võnkumisi
- ***Audio files*** -erinevad heli failid (näiteks .wav ja .mp3), mis laetakse ning mängitakse ette

Eraldi allikana saab võtta ka mikrofoni, mille kaudu võib tulla helisignaal otse mikrofonist ning liigub kõlarisse.

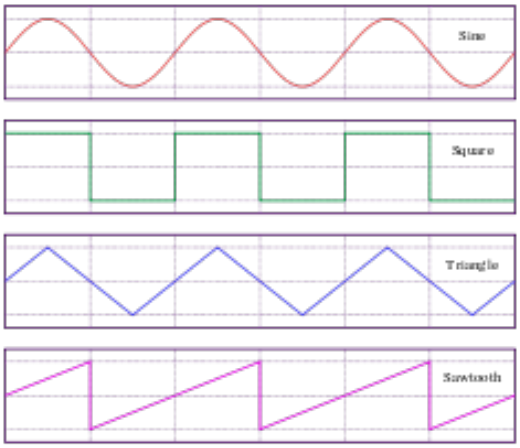
1.6.1 Sagedusgeneraator

Oscillator (sagedusgeneraator) on vooluring, mis loob korduvat signaali. Heli filtrite ja kontrollerite kõrval on see üks põhielemente, mida läheb vaja modernsetel analoogsüntesaatoritel. *OscillatorNode* on sõlm, mis on heliallika eest ning tekitab perioodilise helilaine. *Web Audio API* sagedusgeneraatori vaikimisi kasutatav sagedus on 200 hertsi. (Metivier, Web Audio API Oscillators, 2014)

Põhilised helilainekujud, mida saab genereerida, on järgnevad (Joonis 8):

- ***Sine*** - siinuslaine (*sine*)

- ***Square*** - ruutlaine (*square*)
- ***Sawtooth*** - saehammaslaine (*saw*)
- ***Triangle*** - kolmnurklaine (*triangle*)



Joonis 8 Helilained

Lisaks võnkekõvera kujule, saab määrata ka selle sagedust. See number, mis lisatakse, näitab tegelikku sagedust hertsides. Juhul, kui tahetakse luua polüfoonilisi helisid, saab korraga mitu erineva sagedusega heli genereerida. Selle jaoks tuleb sisestada mitu generaatorit heliallikana. (Metivier, Web Audio API Oscillators, 2014)

1.6.2 Helifailid

API toetab paljusid helifailiformaate, näiteks *WAV, MP3, AAC, OGG* ja teisigi. (Smus, Getting Started with Web Audio API, 2013)

Et heli saada veebilehele failist, peab esmalt laadima helifaili *XMLHttpRequest* meetodiga üles ning seejärel seda dekodeerima *context.decodeAudioData'*ga, et saaks heli laadida *AudioBuffer*'isse. (Smus, Web Audio API, 2013)

AudioBuffer on nagu füüsilise mälu regioon, mida kasutatakse, et ajutiselt paigutada infot, kuni seda viiakse ühest kohast teise. Hetkejuhul on infoks heli helifailist. (Metivier, Web Audio API Audio Buffer, 2014) *AudioBuffer* mahutab umbes 45 sekundit heli.

Üks näide, kuidas kirjutada funktsiooni, mis laeb heli veebilehele, on selline (Koodinäide 4):

```
var request = new XMLHttpRequest();
```

```
reguest.open('GET', url, true);
request.responseType = 'arraybuffer';

request.onload = function(){
      context.decodeAudioData(request.response, function(theBuffer) {
            buffer = theBuffer;
      }, onError);
}
Request.send();
```

Koodinäide 4 helifaili laadimine

Kui fail on *AudioBuffer'*is, tuleb luua *AudioBufferSourceNode'*i. Lihtsamalt seletades on *AudioBufferNode* nagu plaadimängija ning *AudioBuffer* on plaat, millele on muusika salvestatud. Kuna kasutatakse *Web Audio API*'t siis peab ühendama allikasõlme lõppsõlmega, et mingisugust heli hakkaks kostuma. Lõppsõlmeks on siinkohal „*destination*". (Memo, 2015)

Selleks, et heli kostma hakkaks, läheb vaja alljärgnevat funktsiooni: (Koodinäide 5):

```
function playSound(buffer) {
      var source = context.createBufferSource();
      source.buffer = buffer;
      source.connect(context.destination);
      source.start(0);
}
```

Koodinäide 5 funktsioon heli mängima panemiseks

1.7 Filtrid

Filtreid kasutades saab kasutaja moonutada sissetulevat heli. Üks viis selle tegemiseks on lisada *BiquadFilterNode* sõlme (d) heli allika ja lõppsõlme vahele. See sõlm kasutab erinevaid filtreid, mida põhiliselt kasutatakse selleks, et kasutaja saaks manipuleerida, mis helikõrgust edasi lasta ning mida mitte. (Smus, Getting Started with Web Audio API, 2013)

Toetatud filtritüübid on: (Contributors, BiquadFilterNode, 2015)

- ***Low pass filter*** - madalpääsfilter: sagedused allpool piiri lastakse läbi, üleval pool piiri lõigatakse ära
- ***High pass filter*** - kõrgpääsfilter: sagedused ülevalpool piiri lastakse läbi, allpool piiri lõigatakse ära

- ***Band pass filter*** - ribapääsfilter: sagedused, mis asuvad etteantud piiride sees, lastakse läbi, väljapool piiri lõigatakse ära
- ***Low shelf filter*** - madalsagedusfilter: sagedusi allpool piiri kas võimendatakse või alandatakse, ülevalpool piiri olevad sagedusi ei muudeta
- ***High shelf filter*** - kõrgsagedusfilter: sagedusi ülevalpool piiri kas võimendatakse või alandatakse, allpool piiri olevaid sagedusi ei muudeta
- ***Peaking filter*** - vahemiksagedusfilter: sagedusi etteantud piirides võimendatakse või vähendatakse, väljaspool piiri sagedusi ei muudeta
- ***Notch filter*** - kitsastõkkefilter: sagedused, mis asuvad etteantud piiridest väljaspool, lastakse läbi
- ***All pass filter*** - kõike läbilaskev filter: kõik sagedused lastakse läbi, kuid nihutab heli signaali ajas.

2 Web Audio API kasutamine veebilehel

Järgnevalt demonstreerib autor mõningate olulisemate vahendite kasutamist veebilehel kasutades selleks *Web Audio API*'t. Demonstreerimise eesmärgiks on näidata *API* võimalikult paljusid võimalusi ja vahendeid. Demonstreerimiseks loob autor sagedusgeneraatori ja audio puhvri demosid. Lisaks veel läbi <*audio*> elemendi tulevat heli ning kuidas saada heli läbi mikrofoni. Veel demonstreerib autor helisignaali visualiseerimist.

Sagedusgeneraator aitab kasutajal luua mono- ja polüfoonilisi helisid. Audio puhvrit on vaja lühikeste heliklippide salvestamiseks, et neid ei peaks korduvalt laadima. Kasutaja peab saama muuta helitugevust ning soovi korral peatama või käivitama heli mängimise. <audio> elementi kasutatakse pikemate helide mängimiseks, mida ei saa teha läbi puhvri. Mikrofoni demo on selle jaoks, et näidata, kuidas otse saadav heli kohe ette mängida ning visualisatsiooni saab kasutada veebilehe illustreerivamaks muutmiseks.

Kõik näited on kätte saadavad leheküljel http://www.tlu.ee/~tapper12/Web%20Audio%20API/index.html

Näidete loomisel leidis autor abi internetist lehekülgedelt https://developer.mozilla.org/en-US/docs/Web/API, http://middleearmedia.com/ ning https://www.w3.org/2011/audio/drafts/1WD/WebAudio/

2.1 Sagedusgeneraatori kasutamine

Järgnevalt demonstreerib autor sagedusgeneraatori kasutamist. Et paremini oleks aru saada, lõi autor kolm demo lehte. Esimesel on vaid siinuslaine generaator. Teisel lehel saab valida nelja erineva laine vahel ning kolmandal lehele saab nende lainete sagedust ja helitugevust muuta.

Siinuslaine generaatori lehel (*Sine Oscillator*) on heli allikaks loodud üks sagedusgeneraator, mis loob siinuslaine 200 Hz. Autor lõi ka kaks nuppu, start ja stop, et saaks generaatori käivitada ja seisma panna. Näite loomise aluseks oli see, et luua ühe sagedusgeneraatoriga leht, mida saavad lugejad aluseks võtta ning soovi korral ise proovida ning lisada sinna veel generaatoreid või muid lisasid (vt Lisa 1).

Kõigepealt lõi autor muutujad *Audio Context*'i, sagedusgeneraatori, *gain node*'i, start ja stop nupu jaoks. Seejärel funktsiooni, kus loodi *Audio Context*, *gain node*, sagedusgeneraator ning antakse sagedusgeneraatorile ette, mis tüüpi lainet see peab genereerima, mis sagedusel ning ühendatakse generaator *gain node*'i külge, et oleks teada, mis tugevusega heli mängima peab ning käivitatakse generaator.

Järgnevalt luuakse start ja stop funktsioonid, kus esimeses ühendatakse *gain node* lõppsõlmega ning teises ühendatakse see lahti.

Kõige lõpus on funktsioon, mis deaktiveerib start nupu, kui *gain node* on ühendatud lõppsõlmega ning, kui vajutada stop nuppu, siis muutub start nupp jälle aktiivseks.

Helilaine on sagedusel 200 Hz. Nendes näidetes liigub helisignaal läbi kolme sõlme- *source node*, *gain node* ja *destination node*.

Vastavat näidet ei olnud väga keeruline valmistada. Tähele tuleb panna seda, et peale start nupu vajutamist, tuli see deaktiveerida. Seda selle tõttu, et kui mitu korda vajutada vastavale nupule, käivitub mitu siinuslainegeneraatorit, kuid välja lülitab stop nupuga vaid ühe.

Teise näitena lõi autor demo, kus ei oleks vaid siinuslaine, vaid saaks ka valida teiste lainete seast, näiteks hammaslaine, kolmnurklaine ning ruutlaine (vt Lisa 2).

Kõik neli helilainet on sagedusel 200 Hz ning nende vahel saab valida kasutades veebilehel olevaid nuppe.

Kolmanda demona, kasutades sagedusgeneraatorit lõi autor näite, kus saab veel ka muuta kasutajal heli tugevust ning selle sagedust (vt Lisa 3).

Start nupu vajutamisel liigub helisignaal *source node*'ist *gain node*'i ning sellest *destination node*'i. Juhul, kui kasutaja muudab helitugevust, siis seda muudetakse *gain node*'is. Sagedus muutmisel muutub *source node*'ist väljastatava heli sagedus.

Vastaval näitel tuleb kindlasti tähele panna seda, et *gain node*'i väärtuseks tuleb määrata liugnupul olev väärtus (*gainSlider.value*), muidu liugnupp ei toimi. Sama kehtib ka sageduse liugnupu kohta. Ning kindlasti peab *gain node*'i ühendama *source node*'i ja *destination node*'i vahele.

2.2 Audio puhver

Järgnevalt demonstreerib autor lugejale audio puhvri kasutamist. Audio puhvrit kasutatakse selleks, kui soovitakse lühikesi heliklippe mängida veebilehel. Esimeses näites mängib lühike heliklipp läbi audio puhvri. Teises näites on lisatud ka helitugevuse ja kiiruse liugnupud ning kolmandas näites läbib heli enne lõppsõlme jõudmist filtri.

Audio puhvri näidetes lõi autor audio puhvri, mille kaudu mängida veebilehel heli. Selleks lõi autor muutujad *Audio Context*'i, allika, start ja stop nupu jaoks (vt Lisa 4).

Selleks, et lasta heli läbi audio puhvri koos start ja stop nupuga, läheb vaja viite funktsiooni. *Data* funktsioonis luuakse *buffer source node* ning *HMLHttpRequest*'iga küsitakse heli fail. Peale faili küsimist, käivitub uus funktsioon, kus määratakse ära, et helifailist saadud info on see, mida peab kasutama. Selle funktsiooni sees käivitub järgmine funktsioon, mis dekodeerib helifaili info nii, et selle saab sisestada audio puhvrisse. Selles funktsioonis määratakse audio puhver heli allikaks ning ühendatakse ära lõppsõlmega. Lisaks on seal ka juures see, et heli mängitakse igavesti (source.*loop = true*).

Kindlasti pöörata tähelepanu sellele, et *data ()* oleks kirjutatud start funktsiooni sisse, muidu ei hakka heli mängima. Lisaks ka see, et *start* ja *stop* funktsioonides *source.start(0)* ja *source.stop(0)* sulgudes olev null tähistab sekundeid, peale mida kas alustatakse heli mängimist või lõpetatakse see.

Järgnevas näites lisas autor lisaks eelnevale ka võimaluse muuta heli tugevust ning selle ettemängimise kiirust. Selle jaoks võttis autor kasutusele muutujad *playbackControl* ja *gainSlider*. Esimeses funktsioonis tuleb kohe luua lisaks heliallikale ka *gain node*. Lisaks veel helifaili dekodeerimisfunktsiooni sees tuleb luua *source.playback.value*. Esimese väärtust loetakse *gainSlider.oninput* funktsiooniga, mis kontrollib lehel oleva liuguri väärtust ning vastavalt sellele muudab *gain node*'i väärtust. *Playback value* kontrollitakse *playbackControl.oninput* funktsioonis, samuti liuguri kauda, mis asub lehel (vt Lisa 5).

Dekodeerimisfunktsioonis ühendatakse ära *source node gain node*'iga ning pannakse viimase väärtuseks *gainSlider.value*. *Source.playbackRate.value* väärtuseks pannakase *playbackControl.value*.

Erilist tähelepanu tuleb pöörata sellele, et *data* funktsioonis ning *playbackControl.oninput* ja *gainSlider.oninput* funktsioonides peavad olema *gain.gain.value = gainSlider.value* ning *source.playbackRate.value = playbackControl.value*. Vastasel juhul ei saa kasutada liugnuppe helitugevuse ning taasesituskiiruse muutmiseks.

Järgnevas demos võttis autor aluseks eelmise näite koodi ning muutis seda selliselt, et enne, kui heli jõuab lõppsõlme, läbib see madalpääsfiltrit. Selles näites sättis autor lävendiks 1000 Hz (vt Lisa 6).

Selle jaoks lõi autor muutja *low pass filter*'i jaoks. Esimese funktsioonis luuakse vastab filter *audioContext.createBiquadFilter()*. Seejärel kinnitatakse filtri tüüp ning selle filtri sageduse väärtus. Vastavas näites saab kasutaja muuta ka filtri sagedust, et kuulata, kuidas filter erinevatel sagedustel heli muudab.

Et heli nüüd enne lõppsõlme läbiks filtrit, tuleb dekodeerimisfunktsioonis ühendada filter *gain node*'i ja *destination*'i vahele, ehk *gain.connect(low_pass_filter)* ning *low_pass_filter.connect(audioContext.destination)*. Selle tulemusel muudetakse filtri sõlmes heli ning väljundist tuleb heli juba muudetud kujul. Sellisel viisil saab ka teisi filtreid ühendada helisignaali teekonna külge.

Viimases audio puhvri demos on kasutatud kolme erinevat helifaili ning nad tulevad kolmest erinevast audio puhvrist (vt Lisa 7). Siiski kõik see töötab vaid ühes audio konteksti sees. Et seda saavutada, tuleb kogu koodi kolm korda kirjutada ning anda erinevad muutujad.

2.3 Heli otse failist

Kui kasutajal on vaja laadida suuremaid ja pikemaid helifaile veebilehele, mis ületavad 45 sekundit, siis ei saa seda teha läbi audio puhvri, vaid tuleb kasutada allikana mõnda teist atribuuti- Näiteks järgnevas näites kasutas autor HTML5 <*audio*> elementi (vt Lisa 8).

Selle jaoks lõi autor muutujad *Audio Context*'i, audio, *source node*'i ning *gain node*'i jaoks. Allikaks lõi autor seekord *Media Element Source*'i, mis võtab ette määratud helifaili ning hakkab seda mängima. Kuna autor kasutas <audio> elementi, siis sellel elemendil saab lisaks ära määrata selle, kas kuvada brauseri poolt saadavad kontrollnupud, mida kasutada heli mängimise alustamiseks ja peatamiseks ning helitugevuse muutmiseks. Selle jaoks tuleb

kirjutada <*audio controls*>. Selles näites autor just neid kasutas ning selle tõttu puudus vajadus ise luua nupud ning funktsioonid, mida kasutada heli mängima panemiseks ja peatamiseks. Lisaks tuleb määrata helifail asukoht *source* ning selle tüüp *type*. Selle näite puhul on *source src='Audio.mp3'* ning *type='audio/mp3'*.

2.4 Mikrofoni sisendi kasutamine

Selles näites lõi autor näite, mis kuulab arvutiga ühendatud mikrofoni ning seejärel väljastatakse heli arvuti kõlaritesse kasutajale kuulamiseks (vt Lisa 9).

Et saada kätte mikrofoni kaudu heli, peab kasutama *getUserMedia* meetodit. Selle jaoks tuleb esmalt luua muutuja audio konteksti ja allika jaoks. Praegusel juhul on allikaks uus *URL*. *Window.URL.createObjectURL*. Sellega luuakse *URL* just selle jaoks, et saaks kuulata seda, mida mikrofoni lastakse. Iga kord, kui vajutada start nuppu, luuaks üks ning stop nupuga eemaldatakse see. Et läbi nuppude kontrollida tegevust, vajas autor *addEventListener* funktsiooni, mille sees *navigator.getUserMedia* küsib, mida tahetakse saada, kas *audio* või *video* või mõlemat, siis mida sellega peale hakata ning, mis saab siis, kui ei saada mikrofonist andmeid. Selle jaoks tuleb funktsioonis vastavalt siis kirjutada, kas *audio:true, video:true* või mõlemad kirja panna. Stop funktsioonis peatatakse mikrofoni kasutamine ning sellest saadud heli edastamise.

2.5 Web audio API analüüsimisvahend

Web Audio API laseb kasutajal heli analüüsida erinevatel viisidel. Näiteks sageduse põhiselt ja aja põhiselt. Lisaks ka helitugevuse põhiselt. Järgnevalt kasutas autor analyzer node'i helitugevuse analüüsimiseks, et näidata mängitava heliklipi helitugevust visuaalselt.

Selles näites lõi autor visuaalse näite, kuidas *Web Audio API* analüüsib edastatavat heli ning väljastab sellest koostatud visualisatsiooni (vt Lisa 10).

Selle näite juures *canvas*'t. Seda selle jaoks, et oleks koht, kus saaks kuvada *analyzer node*'i poolt analüüsitud helilainet. Seejärel loob autor valmisoleku funktsioon, mille sees loob uue audio kontekst. Selle funktsiooni sees on ka *start* funktsioon, kus kutsutakse välja allika, analüüsija ning skripti ja amplituudi sõlmi tegev funktsioon. Lisaks on seal

script.onaudioprocess funktsioon. Selle sees on *analyzer.getByteTimeDomainData()*, mis kopeerib heli helilaine ning saadab selle *Unit8Array*'sse. Lisaks kontrollitakse, kas muusika mängib ning, kui mängib, siis käivitatakse *drawTime* funktsioon, mis joonistab helilaine ekraanile. *Start* funktsiooni lõpus kontrollitakse, kas audio on puhvris või ei ole. Kui on, mängitakse see sealt, kui ei ole, siis laetakse see sinna *load* funktsiooniga. Peale start funktsiooni on stop funktsioon, mille sees peatatakse heli mängimine ja visualisatsiooni kuvamine.

Erilist tähelepanu tuleb pöörata sellele, et *analyzer node*'i juures peab kasutama *amplitude = new Uint8Array*(*analyzer.frequencyBinCount*), et *Uint8Array* ja *frequencyBinCount* oleks ühe pikkused. Lisaks on vaja *script processor*'it. Seal on kolm parameetrit: *buffer size*, *number of input channels* ja *number of output channels*. Esimese väärtus saab olla vaid 256, 512, 1024, 2048, 4096, 8192 või 16384. Kui seda ei ole määratud, siis valib API ise sobiliku variandi. Mida suurem number, seda vähem on heli katkendlikkust.

2.6 Näidete kokkuvõte

Web Audio API kasutamine ei ole väga raske. Kõigepealt tuleb endale selgeks teha tööjada, kuidas ja mida mis asi teeb. Kui see on selge, on juba koodi kirjutada palju lihtsam ning keerukamad ülesanded saavad ka lahendatud.

Tänu sellele, et API't pidevalt uueneb, lisanduvad sinna uued võimalused, millega tuleb olla kursis. Juhul, kui on muudatus tehtud, ei pruugi vanema koodiga enam API vahendeid tööle saada. Hea näide selle juures on mikrofoni signaali kasutamine. Google Chrome ei lase kasutada *getUserMedia()* enam ebaturvalistel lehekülgedel, vaid nõuab turvalist ühendust. Samas Mozilla Firefox seda ei nõua. Selle tõttu on mikrofoni signaali kasutamise näide töötav vaid Mozilla Firefox'iga.

Autor kasutas enda koodis *navigator.getUserMedia* kuid seda enam ei soovitata. Selle asemele on tulnud *navigator.mediaDevices.getUserMedia* kuid viimane on alles eksperimenteerimisjärgus ning ei pruugi olla stabiilne. Selle tõttu tekib ka vastuolu, kumba neist siis kasutada. Autor soovitab kasutada hetkel veel vana versiooni ning kui uus täielikult valmis, siis koodis väiksed muudatused teha.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli tutvustada lugejale suhteliselt uut ning pidevalt uuenevat heli manipuleerimisvahendit *Web Audio API*, mida kasutatakse veebilehtedel. Selle jaoks pidi autor välja otsima ning tõlkima materjale, et saada täpselt teada, mida vastav vahend võimaldab ning, kuidas seda kõike koodi abil tööle panna ja veebilehel heli manipuleerida.

Esmalt tutvustas autor väga lühidalt varasemaid võimalusi, millega sai veebilehel olevat heli manipuleerida. Seejärel tutvustas autor vahendit *Web Audio API*. Autor selgitas, mis on audio kontekst, selle sees olevad sõlmed, ning mida tähendab modulaarne marsruutimine. Lisaks tõi autor välja, kuidas saab selle vahendiga heli veebilehele - kasutades sagedusgeneraatorit või helifaili ning mikrofoni. Autor katsetas ja demonstreeris erinevate heliallikate kasutamist ja helile efektide lisamist veebilehel. Neid demosid luues peab meeles pidama, et alustada tuleb alati audio konteksti loomisest, mis on kogu vahendi alustala. Kui see on loodud, saab kasutaja sinna panna, mis tahes vahendid, alustades erinevate sagedusgeneraatorite loomisega ning lõpetades heli laadimise veebilehele läbi erinevate filtrite.

Et lugeja saaks paremini aru, kuidas midagi luua, valmistas autor 10 näidet kasutades erinevaid vahendeid: audio puhver, *analyzer node*, *gain node*, sagedusgeneraator ja teisedki. Näited valis autor eesmärgiga demonstreerida võimalikult palju erinevaid võimalusi ja vahendeid, et lugeja saaks piisava ja adekvaatse ülevaate, mida *Web Audio API* pakub.

Kõige rohkem tuleb tähelepanu pöörata sellele, et kõik kasutatavad elemendid oleksid audio konteksti sees. Lisaks, et *source node* oleks ühendatud *destination node*'iga. Juhul kui on vahepeal ka teisi sõlmi, siis peavad need korrektselt olema omavahel ühendatud.

Lõputöö andis autorile palju teadmisi juurde, kuidas saab kasutada heli veebilehel ning selle kasutamise raskusastmest. Lisaks, mida kõike võimaldab pidevalt arenev *Web Audio API* ning, kuidas seda kasutada. Lisaks sai autor juurde oskusi koodi kirjutamisel - kuidas kirjutada lühikest ja kergelt arusaadavat ning töötavat koodi.

Summary

Title: Manipulation of Sound on Web Page using Web Audio API

The objective of this thesis is to introduce to the reader recently come out and regularly updated sound manipulation tool Web Audio API. To do that, author had to look up and translate a lot of material to fully understand, what that tool can do and how to use it on a web page.

At first, author briefly introduced some of the other tools to manipulate sound on web page, which were used before Web Audio API. After that author introduced Web Audio API tool. Author explained, what is Audio Context, different nodes inside it and what is modular routing. In addition, the author pointed out, how to load sound on a web page using oscillator, sound file or microphone. Author experimented and demonstrated using different sound sources and adding different effects to sound. Creating those demos, everybody has to remember, that the first thing to do, is to create an audio context. When that is created, user can start putting in different nodes, from source nodes to filter nodes.

So that readers could understand better, how to create something, author created ten different examples showing different variations of using this tool: audio buffer, analyzer node, gain node, oscillator and more. Author chose those demos to show as many Web Audio API tools, as possible, so the reader could get the best and adequate coverage of Web Audio API.

Whoever uses Web Audio API, has to pay attention to that every node created has to be in audio context. And that the source node has to be connected to destination node. If there are more than two nodes, they have to be connected to each other correctly.

Thanks to this thesis, the author learnt a lot about using sound on web page and how difficult it is to manipulate it. Furthermore, what possibilities does constantly evolving Web Audio API have and how to use these. Also author learnt how to write short and easily understandable code.

Kasutatud kirjandus

Contributors, M. <bgsound>. Loetud 16 veebruar 2016 aadressil https://developer.mozilla.org/en-US/docs/Web/HTML/Element/bgsound

Contributors, M. <object>. Loetud 26 aprill 2016 aadressil https://developer.mozilla.org/en-US/docs/Web/HTML/Element/object

Contributors, M. AudioNode. Loetud 27 jaanuar 2016 aadressil https://developer.mozilla.org/en-US/docs/Web/API/AudioNode

Contributors, M. BiquadFilterNode. Loetud 27 jaanuar 2016 aadressil https://developer.mozilla.org/en-US/docs/Web/API/BiquadFilterNode

Contributors, M. Web Audio API. Loetud 27 jaanuar 2016 aadressil https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

Memo, S. The Web Audio API: What Is It? Loetud 13 jaanuar 2016 aadressil http://code.tutsplus.com/tutorials/the-web-audio-api-what-is-it--cms-23735#disqus_thread

Metivier, O. Play a Sound with Web Audio API. Loetud 27 jaanuar 2016 aadressil http://middleearmedia.com/play-sound-web-audio-api/

Metivier, O. Web Audio API Audio Buffer. Loetud 27 jaanuar 2016 aadressil http://middleearmedia.com/web-audio-api-audio-buffer/

Metivier, O. Web Audio API Basics. Loetud 13 jaanuar 2016 aadressil http://middleearmedia.com/web-audio-api-basics/

Metivier, O. Web Audio API Oscillators. Loetud 13 jaanuar 2016 aadressil http://middleearmedia.com/web-audio-api-oscillators/

Oliver, D.(1999). Sams Teach Yourself HTML in 24 Hours, 4th Edition. Carmel: SAMS Publishing

Smus, B. Web Audio API. Loetud 13 jaanuar 2016 aadressil http://chimera.labs.oreilly.com/books/1234000001552/ch01.html

Smus, B.(2013) Getting Started with Web Audio API. Loetud 13 jaanuar 2016 aadressil http://www.html5rocks.com/en/tutorials/webaudio/intro/?redirect_from_locale=es#toc-abstract

W3C. Audio Working Group. Loetud 16 veebruar 2016 aadressil https://www.w3.org/2011/audio/#entry-9048

LISAD

Lisad

Lisades on välja toodud koodid, mis autor lõi, et oma veebilehtede näited tööle saada.

Lisa 1 Fikseeritud sagedusega sagedusgeneraator

```
window.onload=function(){

  var audioContext,
      oscillator,
      gain,
      startButton = document.querySelector('.start'),
      stopButton = document.querySelector('.stop');

  go();

  startButton.onclick = start;
  stopButton.onclick = stop;

  function go() {
      audioContext = new(window.AudioContext)();
      gain = audioContext.createGain();
      oscillator = audioContext.createOscillator();
      oscillator.type = 'sine';
      oscillator.frequency.value = 200;
      oscillator.connect(gain);
      oscillator.start();
  }

  function start() {
      UI('start');
      gain.connect(audioContext.destination);
  }

  function stop() {
      UI('stop');
      gain.disconnect(audioContext.destination);
  }
```

```
  function UI(state) {
      switch (state) {
          case 'start':
              startButton.disabled = true;
              stopButton.disabled = false;
              break;
          case 'stop':
              startButton.disabled = false;
              stopButton.disabled = true;
              break;
      }
  }
}
```

Lisa 2 Fikseeritud sagedusega neli erinevat helilainet

```
window.onload=function(){

  var audioContext,
      osc,
      gain,
      startButton = document.querySelector('.start'),
      stopButton = document.querySelector('.stop'),
      waveButtons = document.querySelectorAll('.wave button');



  go();

  startButton.onclick = start;
  stopButton.onclick = stop;

  addEventListenerBySelector('.wave button', 'click', function (event) {
    var type = event.target.dataset.waveform;
    changeType(type);
  });

  function go() {
      audioContext = new(window.AudioContext)();
      gain = audioContext.createGain();
      osc = audioContext.createOscillator();
```

```
        osc.type = 'sine';
        osc.frequency.value = 200;
        osc.connect(gain);
        osc.start(0);
    }

    function start() {
        UI('start');
        gain.connect(audioContext.destination);
    }

    function stop() {
        UI('stop');
        gain.disconnect(audioContext.destination);
    }

    function changeType(type) {
      osc.type = type;
    }

    function addEventListenerBySelector(selector, event, fn) {
      var list = document.querySelectorAll(selector);
      for (var i = 0, len = list.length; i < len; i++) {
          list[i].addEventListener(event, fn, false);
      }
    }

    function UI(state) {
        switch (state) {
            case 'start':
                startButton.disabled = true;
                stopButton.disabled = false;
                break;
            case 'stop':
                startButton.disabled = false;
                stopButton.disabled = true;
                break;
        }
    }
}
```

Lisa 3 Muudetava sagedusega ja helitugevusega sagedusgeneraator

```
window.onload=function(){

  var audioContext,
      osc,
      gain,
      startButton = document.querySelector('.start'),
      stopButton = document.querySelector('.stop'),
      waveButtons = document.querySelectorAll('.wave button'),
      gainSlider = document.querySelector('.gain-slider'),
      frequencySlider = document.querySelector('.frequency-slider');

  go();

  startButton.onclick = start;
  stopButton.onclick = stop;

  addEventListenerBySelector('.wave button', 'click', function (event) {
    var type = event.target.dataset.waveform;
    changeType(type);
  });
  gainSlider.oninput = function () {
    changeGain(gainSlider.value);
  }

  frequencySlider.oninput = function () {
    changeFrequency(frequencySlider.value);
  }

  function go() {
      audioContext = new(window.AudioContext)();
      gain = audioContext.createGain();
      osc = audioContext.createOscillator();
      osc.type = 'sine';
      gain.gain.value = gainSlider.value;
      osc.frequency.value = frequencySlider.value;
      osc.connect(gain);
```

```
    osc.start();
}

function start() {
    UI('start');
    gain.connect(audioContext.destination);
}

function stop() {
    UI('stop');
    gain.disconnect(audioContext.destination);
}

function changeType(type) {
  osc.type = type;
}

function changeFrequency(frequency) {
  osc.frequency.value = frequency;
}

function changeGain(volume) {
  gain.gain.value = volume;

}

function addEventListenerBySelector(selector, event, fn) {
  var list = document.querySelectorAll(selector);
  for (var i = 0, len = list.length; i < len; i++) {
      list[i].addEventListener(event, fn, false);
  }
}



function UI(state) {
    switch (state) {
        case 'start':
            startButton.disabled = true;
            stopButton.disabled = false;
```

```
                break;
            case 'stop':
                startButton.disabled = false;
                stopButton.disabled = true;
                break;
        }
    }
}
```

Lisa 4 Audio puhver

```
window.onload=function(){

  var audioContext = new (window.AudioContext)(),
      source,
      start = document.querySelector('.start'),
      stop = document.querySelector('.stop');


  function data() {
    source = audioContext.createBufferSource();
    request = new XMLHttpRequest();
    request.open('GET', 'Audio2.mp3', true);
    request.responseType = 'arraybuffer';
    request.onload = function() {
      var audioData = request.response;
      audioContext.decodeAudioData(audioData, function(buffer) {
          audioBuffer = buffer;
          source.buffer = audioBuffer;
          source.connect(audioContext.destination);
          source.loop = true;
        })
    }
    request.send();
  }
  start.onclick = function() {
    data();
    source.start(0);
    start.setAttribute('disabled', 'disabled');
  }
```

```
  stop.onclick = function() {
    source.stop(0);
    start.removeAttribute('disabled');
  }
}
```

Lisa 5 Audio puhver koos helitugevuse ja kiiruse muutmisega

```
window.onload=function(){

  var audioContex = new (window.AudioContext)(),
    source,
    gain,
    start = document.querySelector('.start'),
    stop = document.querySelector('.stop'),
    playbackControl = document.querySelector('.playback-speed-control'),
    gainSlider = document.querySelector('.gain-slider');

  function data() {
    source = audioContex.createBufferSource();
    gain = audioContex.createGain();

    request = new XMLHttpRequest();
    request.open('GET', 'Audio2.mp3', true);
    request.responseType = 'arraybuffer';
    request.onload = function() {
      var audioData = request.response;
      audioContex.decodeAudioData(audioData, function(buffer) {
      audioBuffer = buffer;
      source.buffer = audioBuffer;
      source.connect(gain);
      gain.connect(audioContex.destination);
      gain.gain.value = gainSlider.value;
      source.playbackRate.value = playbackControl.value;
      source.loop = true;
      })
    }
```

```
    request.send();
  }
  start.onclick = function() {
    data();
    source.start(0);
    start.setAttribute('disabled', 'disabled');
  }
  stop.onclick = function() {
    source.stop(0);
    start.removeAttribute('disabled');
  }
  playbackControl.oninput = function() {
    source.playbackRate.value = playbackControl.value;
  }
  gainSlider.oninput = function(){
    gain.gain.value = gainSlider.value;
  }
}
```

Lisa 6 Audio puhvri heli läbi filtri

```
window.onload=function(){

  var audioContext = new (window.AudioContext)(),
    source,
    gain,
    start = document.querySelector('.start'),
    stop = document.querySelector('.stop'),
    playbackControl = document.querySelector('.playback-speed-control'),
    gainSlider = document.querySelector('.gain-slider'),
    low_pass_filter = document.querySelector('.low-pass-filter'),
    filterSlider = document.querySelector('.filter-frequency-control');

  function data() {
    source = audioContext.createBufferSource();
    gain = audioContext.createGain();

    low_pass_filter = audioContext.createBiquadFilter();
    low_pass_filter.type = "lowpass";
    low_pass_filter.frequency.value = 1000;
```

```
    request = new XMLHttpRequest();
    request.open('GET', 'Audio2.mp3', true);
    request.responseType = 'arraybuffer';
    request.onload = function() {
      var audioData = request.response;
      audioContext.decodeAudioData(audioData, function(buffer) {
      audioBuffer = buffer;
      source.buffer = audioBuffer;
      source.connect(gain);
      gain.connect(low_pass_filter);
      low_pass_filter.connect(audioContext.destination);
      gain.gain.value = gainSlider.value;
      source.playbackRate.value = playbackControl.value;
      source.loop = true;
      })
    }
    request.send();
  }
  start.onclick = function() {
    data();
    source.start(0);
    start.setAttribute('disabled', 'disabled');
  }
  stop.onclick = function() {
    source.stop(0);
    start.removeAttribute('disabled')
  }
  playbackControl.oninput = function() {
    source.playbackRate.value = playbackControl.value;
  }
  gainSlider.oninput = function(){
    gain.gain.value = gainSlider.value;
  }
  filterSlider.oninput = function(){
    low_pass_filter.frequency.value = filterSlider.value;
  }
}
```

Lisa7 Kolm erinevat heli kolmest puhvrist

```
window.onload=function(){

  var audioContext = new (window.AudioContext)(),
      source1,
      source2,
      source3,
      start_one = document.querySelector('.start-one'),
      stop_one = document.querySelector('.stop-one');
      start_two = document.querySelector('.start-two'),
      stop_two = document.querySelector('.stop-two');
      start_three = document.querySelector('.start-three'),
      stop_three = document.querySelector('.stop-three'),
      gainSlider = document.querySelector('.gain-slider'),
      gain = audioContext.createGain();

  function data1() {
    source1 = audioContext.createBufferSource();
    request = new XMLHttpRequest();
    request.open('GET', 'Audio1.mp3', true);
    request.responseType = 'arraybuffer';
    request.onload = function() {
      var audioData1 = request.response;
      audioContext.decodeAudioData(audioData1, function(buffer) {
          audioBuffer = buffer;
          source1.buffer = audioBuffer;
          source1.loop = true;
          source1.connect(gain);
          gain.connect(audioContext.destination);
          gain.gain.value = gainSlider.value;
        })
    }
    request.send();
  }
  function data2() {
    source2 = audioContext.createBufferSource();
    request = new XMLHttpRequest();
    request.open('GET', 'Audio2.mp3', true);
    request.responseType = 'arraybuffer';
```

```
  request.onload = function() {
    var audioData2 = request.response;
    audioContext.decodeAudioData(audioData2, function(buffer) {
        audioBuffer = buffer;
        source2.buffer = audioBuffer;
        source2.loop = true;
        source2.connect(gain);
        gain.connect(audioContext.destination);
        gain.gain.value = gainSlider.value;
      })
  }
  request.send();
}
function data3() {
  source3 = audioContext.createBufferSource();
  request = new XMLHttpRequest();
  request.open('GET', 'Audio3.mp3', true);
  request.responseType = 'arraybuffer';
  request.onload = function() {
    var audioData3 = request.response;
    audioContext.decodeAudioData(audioData3, function(buffer) {
        audioBuffer = buffer;
        source3.buffer = audioBuffer;
        source3.loop = true;
        source3.connect(gain);
        gain.connect(audioContext.destination);
        gain.gain.value = gainSlider.value;
      })
  }
  request.send();
}
function changeGain(volume) {
  gain.gain.value = volume;

}
gainSlider.oninput = function () {
  changeGain(gainSlider.value);
}
start_one.onclick = function() {
  data1();
```

```
    source1.start(0);
    start_one.setAttribute('disabled', 'disabled');
  }

  stop_one.onclick = function() {
    source1.stop(0);
    start_one.removeAttribute('disabled');
  }
  start_two.onclick = function() {
    data2();
    source2.start(0);
    start_two.setAttribute('disabled', 'disabled');
  }

  stop_two.onclick = function() {
    source2.stop(0);
    start_two.removeAttribute('disabled');
  }
  start_three.onclick = function() {
    data3();
    source3.start(0);
    start_three.setAttribute('disabled', 'disabled');
  }

  stop_three.onclick = function() {
    source3.stop(0);
    start_three.removeAttribute('disabled');
  }
}
```

Lisa 8 Heli otse failist

```
window.onload=function(){
  var audioContext = new (window.AudioContext)(),
      audio = document.querySelector('audio'),
      source = audioContext.createMediaElementSource(audio),
      gain = audioContext.createGain();

source.connect(gain);
gain.connect(audioContext.destination);
```

```
}
```

Lisa 9 Mikrofoni signaali kasutamine

```
window.onload=function(){

  navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia;

  var audio = document.getElementById("audio");
  var source = window.URL ? window.URL.createObjectURL : function(stream) {return stream;};
  var audioContext = window.AudioContext;

  document.getElementById('play').addEventListener('click', function() {
    navigator.getUserMedia({
      audio: true
    },
    function(stream) {

      audioStream = stream;
      audio.src = source(stream);
      audio.play();
    },
    function(error) {

      console.log("Error");
    });
  });
  document.getElementById('stop').addEventListener('click', function() {

    audio.pause();
    audioStream.stop();
  });
}
```

Lisa 10 Analüüsimisvahend

```
  var audioContext,
```

```
    audioBuffer,
    sourceNode,
    analyserNode,
    javascriptNode,
    data = null,
    playing = false,
    amplitude,
    contex,
    canvasWidth = 512,
    canvasHeight = 256,
    audio = "Audio.mp3";

window.requestAnimationFrame = (function(){
  return  window.requestAnimationFrame ||
          function(callback, element){
            window.setTimeout(callback, 1000 / 60);
          };
})();
$(document).ready(function() {
    contex = $("#canvas").get()[0].getContext("2d");
        audioContext = new AudioContext();
    $("#start").on('click', function(e) {

        AudioNodes();
        script.onaudioprocess = function () {
            // get the Time Domain data for this sample
            analyser.getByteTimeDomainData(amplitude);

            if (playing == true) {
                requestAnimationFrame(drawTime);
            }
        }
        if(data == null) {
            load(audio);
        } else {
            play(data);
        }
    });
    $("#stop").on('click', function(e) {
        source.stop(0);
```

```
            playing = false;
        });
    });
    function AudioNodes() {
        source     = audioContext.createBufferSource();
        analyser   = audioContext.createAnalyser();
        script = audioContext.createScriptProcessor(1024, 1, 1);

        amplitude = new Uint8Array(analyser.frequencyBinCount);

        source.connect(audioContext.destination);
        source.connect(analyser);
        analyser.connect(script);
        script.connect(audioContext.destination);
    }
    function load(url) {
        var request = new XMLHttpRequest();
        request.open('GET', url, true);
        request.responseType = 'arraybuffer';
        request.onload = function () {
            audioContext.decodeAudioData(request.response,  function  (buffer)
{
                data = buffer;
                play(data);
            });
        }
        request.send();
    }
    function play(buffer) {
        source.buffer = buffer;
        source.start(0);
        source.loop = true;
        playing = true;
    }
    function drawTime() {
        clear();
        for (var i = 0; i < amplitude.length; i++) {
            var value = amplitude[i] / 256;
            var y = canvasHeight - (canvasHeight * value) - 1;
            contex.fillStyle = 'RED';
```

```
        contex.fillRect(i, y, 1, 1);
    }
}
function clear() {
    contex.clearRect(0, 0, canvasWidth, canvasHeight);
}
```