TARTU ÜLIKOOL
LOODUS- JA TÄPPISTEADUSTE VALDKOND
Arvutiteaduse instituut
Informaatika õppekava

Laimis Dalke

Tarkvara audio taasesituseks piltkoodist

Bakalaureusetöö (9 EAP)

Juhendaja: Prof. Eero Vainikko

Tartu 2016

Tarkvara audio taasesituseks piltkoodist

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks on luua tarkvara, mis sooritab pildituvastust, heli parsimist QR-koodist, kodeerimist, dekodeerimist ja taasesitust. Antakse ülevaade helist digitaalses formaadis, selle omapäradest ning erinevates kasutusvaldkondades esinenud implementatsioonidest. Kirjeldatakse tehnoloogiaid, millega saab tänapäeval kõige paremini käesolevat programmi luua ning tuuakse välja neist kõige paremad. Samuti kirjeldatakse tarkvara loomise tööprotsessi, sellega seonduvaid töövahendeid ning kogu tarkvara arendusteed. Lõpptulemuseks on tarkvara, mis suudab pildituvastuse põhjal tagasi mängida heli ja seda protsessi ka vastupidi läbi viia.

Võtmesõnad:
Tarkvara, ribakood, heli, kompresseerimine, kodeerimine
CERCS: P175 informaatika, süsteemiteooria

Audio playback software from barcode

Abstract:

The purpose of this Bachelors thesis is to develop a software that is capable of image recognition, audio parsing from a QR Code, audio encoding, decoding and playback. An overview is given of what digital audio is and what its peculiarities and use cases in different real world scenarios are. We also bring forth the idea of this software, best technologies available today to carry out the given task and explain the chosen methods and why they were used. Additionally, we analyze each step in the development of this software, the reasoning behind those steps and the technologies used to execute our tasks. The final result is a program that can parse an image, decode its audio contents and playback them. This process can also be reversed.

Võtmesõnad:
Software, barcode, sound, compression, encoding
CERCS: P175 informatics, system theory

Sisukord

1. Sissejuhatus

Tänapäeval on väga populaarseks kujunenud QR-kood [1, 2] ehk ruutkood, mis on Jaapanis loodud kodeeritud kahemõõtmeline maatrikskood ning võimaldab skaneerida infot mobiiltelefoni, kus mobiilirakendus selle dekodeerib. Selle eesmärgiks on panna pikad ning raskesti meeldejäävad tekstilõigud ja veebiaadressid kodeeritud kujul kirja nii, et selle informatsiooni omandamine oleks kiire ja vaevutu.

Sellel tehnoloogial on aga mõningaid negatiivseid külgi. Nimelt kõige esimeseks probleemiks on see, et keegi ei tea, mis antud ruutkoodil täpselt kirjas võib olla – kas veebiaadress, mingi tekstilõik või hoopis mingi muu tundmatu hulk andmeid. Teiseks probleemiks ongi tundmatutele veebiaadressitele viitamine. Ei saa kunagi kindel olla, kas skaneeritud ruutkood viib kahjulikule aadressile või mitte. Üldiseks ohuks on alati ka tundmatute andmete dekodeerimine. Kolmandaks probleemiks osutub aga internetiühendus. Nimelt on enamus ruutkoode just veebiaadressid, mis vajavad avanemiseks internetiühendust kasutaja nutitelefonis. Paljudel inimestel tänapäeval pole võimalust pääseda ligi internetiühendusele mobiiltelefonis. Kui internetiühendus ka on, siis tuleb see telefonis aktiivseks seada, oodata selle edukat ühendumist ning alles siis ruutkoodi pildistada. Neljandaks probleemiks on ooteaeg. Nimelt peale skaneerimist tuleb veel oodata kuni dekodeeritud veebileht ennast ära laeb ja alles siis saab informatsioonile ligi. Need negatiivsed tunnused raiskavad palju aega ning on just vastupidise tulemusega probleemidele, mida ruutkood lahendada üritab.

Minu idee antud bakalaureusetööks algaski sellest, et küsisin endalt, miks on ruutkoodides ainult veebiaadressid. Nii tekkis mõte panna neisse hoopis midagi huvitavamat, nagu helisid. Väljakutse on aga kuidas salvestada helifaili ribakoodi, kui kõikide ribakoodide ainukeseks eesmärgiks on numbri ja tähtede jada salvestamine. Nimelt on antud bakalaureusetöö eesmärgiks luua programmikood, mis suudab sisseantud pildifailist heli tuvastada ja seda kohe ka tagasi mängida või salvestada. Valmis programm saab olema sellisel kujul, et teda on võimalik ka kergelt telefonirakenduseks muuta. Siinjuures saab kirjeldada ka sünteetilist protsessi algusest lõpuni, eeldades, et programm on nutitelefonis ja töötavas korras. Töö algab sellest, et pildistatakse koodiriba, mis sisaldab kodeeritud helifaili. Selle skaneerimisel alustab toimingut pilduvastus, mis tõlgib pildi peal oleva koodi arvutile arusaadavaks baidivooks. Seejärel dekodeeritakse antud baidivoog ja saadakse kokkusurutud heli [3, 13]. Seda heli nüüd saabki juba ka kasutajale esitada. See protsess on ka pööratav. Esimeseks sammuks siinpuhul on nutitelefoniga heli salvestamine. Seejärel surub tarkvara antud heli kokku väiksemasse formaati ja kodeerib [20] selle omakorda pildi peale printimiseks valmis baidivooks. Antud baidivoog konverteeritaksegi pildiformaati ning siis saabki juba pilti välja printida.

Selle rakenduse positiivseteks külgedeks ongi kõigi hetkel olevate ruutkoodide negatiivsed küljed. Nimelt pole selle kasutamiseks vaja internetiühendust, tööprotsess on kiire ja väljundi saab kasutaja kohe ka kätte.

Põhiliseks motivatsiooniks selle programmi loomiseks on unikaalsus. Tänapäeval pole taolise idee rakendusi pea kuskilt leida. Õnnestus leida ainult kaks samalaadset lahendust – Venemaal loodud tarkvara Phonopaper [5] ja SoundPaper [12] - kuid nende populaarsus on marginaalne. Samuti jääb arusaamatuks, millisel moel nad oma rakenduse eesmärgid saavutavad, sest täpsustavaid spetsifikatsioone ei leidu ja demonstreerivad videolõigud jäävad väga pealiskaudseteks. Minu lahenduse eesmärgiks on samaväärse rakenduse loomise protsessi kirjeldada, seda analüüsida ja dokumenteerida. Arvan, et antud ülesanne on piisavalt huvitav ning omalaadne, mis on väga heaks ajendiks töötava lõpptulemuse saavutamiseks.

Taolisele tarkvarale võib leiduda ka palju praktilisi rakendusi. Näiteks saab integreerida selliseid skaneeritavaid heliribasid muusikaga seonduvatesse ajakirjadesse ja intervjuude artiklitesse ajalehtedes. Samuti oleks sihtgrupiks ka loodusajakirjad, kus heliribad võivad sisaldada loomade või looduse hääli. Idee võib osutuda kasulikuks ka pimedatele ja vaegnägemisega isikutele. Näiteks ravimite pakendid, mis juba kasutatavad pimedate kirja. Viimase põhilise rakendusena võiksid olla ka turismikohad. Pannes selliseid heliribasid kuulsate monumentide või vaatamisväärsuste juurde saab näiliselt imiteerida giidide tööd.

Loodetavaks lõpptulemuseks on programmi lähtekood keeles Java [21], mida on lihtne portida erinevatesse seadmetesse. Põhiliseks siht platvormiks oleks loomulikult nutitelefonid.

Antud tarkvara loomise kitsendusteks on põhiliselt kodeerimise ja kokkupakkimise algoritmid. Nimelt on arvutiteaduses selge piir kokkupakkimise algoritmide võimekusel, millest alates andmeid väiksemaks muuta ei saa või pole mõistlik. Andmete kokkusurumine on tavaliselt tagajärgedega operatsioon, mis tavaliselt ei suudagi kõiki soovitud tulemusi saavutada. Bakalaureusetöö eesmärk on analüüsida ning leida parimad algoritmid ja tehnoloogiad antud tarkvara loomiseks.

2 Taust

Antud peatükk annab ülevaate olemasolevatest töödest ja tehnoloogiatest.

2.1 Sarnased olemasolevad lahendused

Hetkel on sarnaste olemasolevate lahenduste valik väga kitsas. Praegu leidub kaks lahendust taolisele probleemile: PhonoPaper ja SoundPaper. Need programmid suudavadki helifaili muuta pildiks, seda salvestada ja seejärel ka seda taasesitada. PhonoPaper'i omapäraks on see, et nende algoritm ei kaasa pea mingit andmete kompresseerimist ega vähendamist ribakoodi kujul. Tegemist on üldiste helilainete visualiseerimise- ja lugemisega, millele eelneb palju helifaili ümberpakendamist. Selle tõttu on ka heli kvaliteet märgatavalt kehv. SoundPaper'i puhul on tegemist ribakoodiga. Samalaadne lõpptulemus on ka minu programmil, kuid kahjuks pole SoundPaper'i ribakoodi täpsustavaid spetsifikatsioone kuskilt leida. Nende kodulehelt leiduvate piltide põhjal võib aga eeldada, et kasutatavaks ribakoodi tüübiks on midagi sarnast DataMatrix'i [19] ja PDF-417-ga [18].

Põhiline erinevus minu enda lahendusest tulenebki andmete kodeerimisest. Nimelt ei kaasa PhonoPaper'i algoritm pea mingit andmete kompresseerimist ega vähendamist ribakoodi kujul. SoundPaper'i puhul on aga raske öelda, kui efektiivne nende lõpptulemus on, sest reaalset lahendust ja demonstratsiooni ei suutnud ma kuskilt leida.

2.2 Seotud uurimistööd või projektid

Samalaadse tarkvara loomisega otseselt seonduvaid projekte ja uurimustöid kahjuks pole. Sellest hoolimata eksisteerivad uurimustööd tehnoloogiate üle, mida ma oma programmi loomisel kasutan. Nendest põhiliseimad hõlmavad andmete salvestamist suure tihendusega ribakoodidesse.

Üheks neist on Xiaofei Fengi artikkel [6] värviliste ribakoodide kasutatavusest ja nende tihendusvõimsusest. Samuti kirjeldatakse värviliste, kõrge tihendustega ribakoodidide võimsust ka Hewlett-Packardi meeskonna poolt kirjutatud artiklis „Effect of copying and restoration on color barcode payload density” [7]. Nende artiklite põhimõteteks on rakendada nii-öelda kolmandat dimensiooni andmete salvestamiseks ribakoodidesse. Siiani on ribakoodid olnud nii ühe kui kahe dimensioonilised, mis tähendab, et andmeid salvestatakse nii ridadesse kui ka tulpadesse suhteliselt. Kaasates siia ka värvilised mustrid, saab piltlikult öeldes neid ribakoode klassifitseerida kui kolme dimensiooniliste koodidena, sest tihendus kasvab tulpade ja ridade väliselt.

Hao Wang ja Yanming Zou poolt kirjutatud artiklis [8] räägitakse uuest, nutitelefonidele optimiseeritud ribakoodi võimalikust lahendusest. Kuna telefonide kaamerad tänapäeval pole veel fotoaparaatidega võrreldavad, siis tuleb rakendada spetsiifilisi meetodeid piltkoodi formuleerimiseks, võttes arvesse andmete hulga ja antud seadmetes protsessorite võimsusi.

Orhan Bulan, Vishal Monga ja Gaurav Sharma uurimuses [9] analüüsitakse ribakoodi mustrite kompaktsust. Iga ribakoodi probleemiks on kindla hulga informatsiooni salvestamine mingi ühikulisesse pindalasse, kuid olles piiratud ainult must-valge värvivalikuga, ei saa siit edasi enam kaugele jõuda. Probleemi lahendusena implementeeritakse kahe-tooniline värvivalik ja punktsüsteem, mis erinevalt ruutsüsteemist, suudab tänu kahe-toonilisele rakendusele salvestada rohkem informatsiooni.

Kuna heliformaate on tohutult, siis tarkvaras kasutatavaks heliks valisin Vorbis OGG heli formaadi. Antud teemal on kaks täpsustavad uurimustööd ja artiklit. „The Ogg Encapsulation Format Version 0” [10] Räägib OGG formaadi omapäradest. Räägitakse selle formaadi kujunemisest, kasutatavusest ja põhiideest. „Ogg Vorbis and MP3” [11] artiklis võrreldakse MP3 ja OGG formaate. Teades, et mõlemal formaadil on nii tugevusi kui ka nõrkusi, kirjeldatakse neid täpsemalt ja võrreldakse nende mõju heli salvestamisel ja esitamisel.

2.3 Tehnoloogiad

Heli taasesitluse tarkvara koosneb põhiliselt kolmest komponendist.

- Heli kokkupakendamine väiksemasse formaati
- Kokkupakitud heli kodeerimine ja ettevalmistamine ribakoodi panekuks
- Ettevalmistatud andmetest ribakoodi genereerimine

Järgnevas käsitleme neid kõiki lähemalt.

2.3.1 Heli Kokkupakkimine

Heli kokkupakkimiseks on tohutul hulgal formaate. Alustades AAC, AU, MP4 formaatidest ja lõpetades OGG, WAV ja WMA formaatidega. Olemasolevate tehnoloogiate valik, mis võimaldaks helifaile kokku pakkida Java programmeerimiskeeles on väiksem. Nimelt on tasuta kättesaadava teegina olemas ainult üks võimekas teek, milleks on JAVE [14].

Valitud tehnoloogia

Heli kokkupakendamise jaoks tuli valida selline lahendus, mille järel on helifaili suurus kõige

väiksem. Praegu on sellise eesmärgi lahendamiseks kaks põhilist formaati: MP3 ja OGG. Testimise tulemusena on näha, et kui pakkida kokku 200 kilobaidi suurune WAV formaadis fail MP3 formaati, siis tulemuseks saame 46 kilobaidi suuruse faili. OGG formaat tagastab aga 9.4 kilobaidi suuruse faili. On selgesti näha, et OGG formaadist tuletatav fail võimaldab suuremat kokkupakkimise indeksit saavutada, mis osutuski piisavaks põhjuseks selle tehnoloogia valimiseks. Heliformaatide pakendamiseks siinpuhul kasutangi eelnevalt mainitud Java JAVE teeki, mis on avatud lähtekoodiga ja tasuta kättesaadav GPL litsentsi alusel.

2.3.2 Heli kodeerimine

Andmete kodeerimiseks on lahendusi päris mitmeid. Tuleb aga tähele panna, et praeguse peatüki juures eeldame, et helifail on juba kokku pakitud ning tegemist pole enam kokkupakkimisega. Siinpuhul on tegemist hoopis andmete teisele kujule viimisega. Nimelt me kodeerime andmed sobivale kujule, mis võimaldab meil imiteerida tavapärast tekstisisendit. Seda selle tõttu, et helifailis pole sõnesid ega arve. Helifail on lihtsalt üks abstraktne andmete kogum. Selle etapi eesmärk on ette valmistada sisendandmed järgmise etapi jaoks ning garanteerida edukas sisend ribakoodi genereerivasse meetodisse. Olemasolevateks tehnoloogiateks on siin Base16, Base32 ja Base64 [16]. Samuti ka Ascii86, BasE91 ja palju teisi.

Valitud tehnoloogia

Heli kodeerimine on protsess, kus me muudame sisendfaili baidid ribakoodi algoritmile loetavaks tekstihulgaks. Selleks on mitmeid kodeeringuid, mis tagavad kadudeta edasi-tagasi kodeerimise. Põhiliseks eesmärgiks on jällegi valida kõige efektiivsem kodeering, mis toob tagasi kõige vähem baite. Järgnevas tabelis näeme, mis omadustega on tänapäeval enamkasutatavad kadudeta kodeeringud:

Kodeering	Tähestik	Tähe:baidi suhe	Äärestus
base16	0-9 A-F	2.00	N/A
base32	A-Z 2-7	1.60	=
base32Hex	0-9 A-V	1.60	=
base64	A-Z a-z 0-9 + /	1.33	=
base64Url	A-Z a-z 0-9 - _	1.33	=

Tabel 1: Baitide kodeerimine tekstkujule

Kuna base64 kodeering tagab kõige parema tähe-baidi suhte, siis otsustangi valida selle. Base64Url kodeering on parim veebirakendustes, kus mängivad tähtsat rolli mõtte- ja alakriipsud ning mille

arvelt saab ruumi veelgi kokku hoida. Meie rakendus on aga teiste nõuetega.

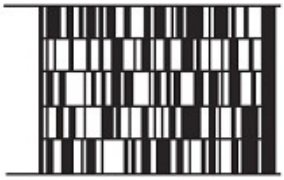
2.3.3 Ribakoodi genereerimine

Ribakoodi genereerimiseks on mitmeid tehnoloogiaid. Kuna olulist rolli mängib ribakoodi mahutavus, siis peame piirduma kahedimensiooniliste tüüpidega. Tulemuseks saame 6 tänapäeval enamlevinud ribakoodi:

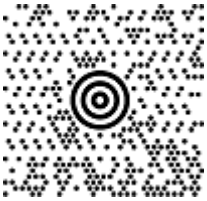
- PDF-417 – piiratud kõrguse ja laiuse suhe, suure tihendusega
- DataMatrix – suure tihendusega, eeliseks on väike pindala
- Maxicode – piiratud füüsilise suurusega, seega ka võrreldavalt väiksema mahuga
- QR Code – tänapäeval enamlevinud ribakood, suure tiheduse ja töökindlusega
- Code 49 – väiksemamahuline kahemõõtmeline ribakood
- 16K – visuaalselt sarnane Code 49 ribakoodiga, mahutavus veelgi väiksem

Pilt 1: DataMatrix ribakoodi näitepilt

Pilt 2: QR Code ribakoodi näitepilt



Pilt 4: Code 49 ribakoodi näitepilt



Pilt 3: MaxiCode ribakoodi näitepilt

Pilt 5: PDF-417 ribakoodi näitepilt

Ribakoodide genereerimiseks leiduvad järgnevad Java teegid: Barcode4J, Barbecue, ZXING. Nad kõik võimaldavad ribakoodide genereerimist ja leiduvad sisuka dokumentatsiooniga.

Valitud tehnoloogia

Ribakoodi genereerimiseks kasutan ZXING [17] avatud lähtekoodiga teeki. Teek on avalikult kättesaadav Apache 2004 v2.0 litsentsi alusel ning võimaldab ribakoode nii kirjutada kui ka lugeda. Kõige parem omadus antud teegi puhul on see, et ta võimaldab töötada mitmete erinevate ribakoodi formaatidega, mis garanteerib selle, et vajadusel saab ribakoodi tüüpi muuta ilma, et tuleks midagi uuesti oma programmi kaasata. Oma rakenduse ribakoodi tüübina kasutan tavalist QR ribakoodi formaati. Otsustasin selle kasuks, sest selline tehnoloogia võimaldab salvestada kõige rohkem andmeid paberil suhteliselt vajamineva paberi pindalaga. Andmete tihedus QR-koodis on keskmiselt 32.361 bitti ruutsentimeeteri kohta [2].

3 Lahenduskäik

Nagu eelnevalt juba mainitud, koosneb tarkvara kolmest osast: heli kokkupakkimine, kodeerimine ja ribakoodi genereerimine. Igal punktil esineb ka pööratav osapool, mis hõlmab pildifailist helifaili genereerimise. Erandina on heli kokkupakkimise osa, kus helifaili on võimatu tagasi originaalsesse formaati lahti pakkida. See tähendab seda, et kui sisendfailiks on MP3 fail ning meie programm moodustab sellest OGG formaadis faili, siis tagasi me MP3 helifaili ei saa.

Lõpptulemusena saadud pildifail koosneb mitmetest väiksematest QR Code ribakoodidest. Nende järjestamine toimub vasakult paremale ning ülevalt alla. Samas järjestuses toimub ka ribakoodide lugemine.

Programmis on ridadel 46 ja 47 eeldefineeritud kaks staatilist väärtust: *chunkSizeinBytes* ning *chunkDimensions.* Esimene neist näitab palju informatsiooni me baitides ühele alam-koodile sisestame. Teine väärtus määrab alam-ribakoodi laiuse ja kõrguse. See hetkelolev lahendus sellele, kuidas me lõpptulemusena saadud pildifaili alam-ribakoode paneme ja neid sealt loeme.

3.1 Ribakoodi genereerimine

Ribakoodi genereerimisel on sisendfailiks helifail. Helifail võib olla pea ükskõik mis kujul, sest kasutame väga võimekat JAVE teeki.

3.1.1 Heli kokkupakkimine

Ribakoodi genereerimisel on heli kokkupakkimine meie esimene etapp. Sisendiks on siin helifail. Operatsiooni algatab meetod *generateImage(String filePath, String outputImage).* Heli konverteerimine ehk väiksemaks formaadiks muutmine algab kohe esimesel real selles samas meetodis:

```
81.  byte[] audioBytes = convertFile(filePath);
```

Siin käivitame omakorda meetodi *convertFile(String filePath),* mis võtab sisendiks töödeldava helifaili ja tagastab OGG formaadis oleva helifaili baidivoo. Heli konverteerimine OGG formaati, meetodis *convertFile,* toimub JAVE teegi abil järgnevalt:

```
114. EncodingAttributes att = new EncodingAttributes();
115. att.setFormat("ogg");
116. AudioAttributes aAtt = new AudioAttributes();
117. aAtt.setCodec("vorbis");
118. att.setAudioAttributes(aAtt);
119. new Encoder().encode(new File(filePath), tempFile, att);
```

Siin tekitame uue faili *tempFile*-ina, millest me nüüd saame baidivoo tagastada kergema protseduuriga.
Järgnevalt lähme tagasi reale 82 ja käivitame nüüd teise kokkupakkimise algoritmi:

```
82. byte[] compressedAudioBytes = compressBytes(audioBytes);
```

See meetod kompresseerib sisseantud helifaili baidivoo ZIP-laadse algoritmiga. Tegemist on Java-sisese paketiga GZIP [4], mis suudab sisendandmeid kokku pakkida omakorda kuni 30%:

```
215. ByteArrayOutputStream bos = new ByteArrayOutputStream();
216. GZIPOutputStream gzip = new GZIPOutputStream(bos);
217. gzip.write(data);
218. gzip.close();
219. byte[] compressed = bos.toByteArray();
220. bos.close();
```

Nüüd on meil helifail optimaalselt kokkupakitud ning võib edasi minna kodeerimise juurde.

3.1.2 Heli kodeerimine

Kodeerimise protsessi eesmärk on valmistada ette sisendina antud baidivoog sõnele viimiseks. Esialgu aga kodeerimisest. Seda teeme real 83, kohe peale kokkupakitud baidivoo tagastust:

```
83. byte[] base64Encoded = Base64.getEncoder().encode(compressedAudioBytes);
```

Järgmisena moodustame sõne Java sisseehitatud *String* objekti klassiga:

```
85. String input = new String(base64Encoded);
```

Base64 kodeeringu kasutuse põhjuseks on see, et kui jätta vahele rida 83 ja selle asemel anda kohe real 85 *String* klassi sisendiks kokkupakitud baidivoog, siis saadud sõne tuleb kadudega kui ta tagasi baidivooks teha. Nimelt ei vasta antud baidid paljudes asukohtades sellel, mida me algul

ribakoodi panna üritasime.

Kodeerimise tulemusena on meil nüüd inimesele arusaamatu sõne, kuid ribakoodi genereerimisele täielikult kõlblik sisend.

3.1.3 Ribakoodi genereerimine

Nüüd võime alustada ribakoodi genereerimisega. Selle etapi esimeseks osaks on algul sisendist moodustada mitu väiksemat ribakoodi. Seda teeme real 98:

```
98. ArrayList<BufferedImage> outputImages = createImages(input, chunkSizeinBytes, cycles, hintMap, chunkDimensions);
```

createImages meetodil on viis parameetrit. Esimeseks on sisendsõne ehk meie kodeeritud helifaili baidivoog. Teiseks parameetriks on ühe ribakoodi mahutavus baitides. Kolmandaks parameetriks on eelnevalt, real 88, leitud arv, mis näitab, mitu väiksemat ribakoodi tuleb genereerida. Neljas parameeter hoiab endas Vea taluvuse indeksit, mis meil on hetkel seatud madala (L) peale. Viimaseks parameetriks on väiksema ribakoodi laius. Meetod tagastab massiivi koos kõigide alam-piltidega.

Meetodis *createImages* kõige tähtsamaks osaks ongi ribakoodi genereerimine. Seda saavutame alam-meetodiga *createQRCode*:

```
182. images.add(createQRCode(qrCodeData, hintMap, chunkDimensions));
```

Kasutame teeki ZXING, mis hõlbustab kogu tööprotsessi tunduvalt. Meetodi *createQRCode* sisu on vastav:

```
255. BitMatrix matrix = new MultiFormatWriter().encode(qrCodeData, BarcodeFormat.QR_CODE, chunkDimensions, chunkDimensions, hintMap);
256. BufferedImage image = MatrixToImageWriter.toBufferedImage(matrix);
257. return image;
```

Tagastatavaks objektiks ongi üks ribakoodi pilt. Nüüd tuleb meil aga need kõik pildid kokku panna ja nad salvestada.

Piltide kokkuliitmiseks kasutame Javasse sisseehitatud operatsioone. Neid rakendame meetodis *joinBufferedImages*, mida me kutsume välja real 99:

```
99. BufferedImage joinedImage = joinBufferedImages(outputImages, rows, columns, chunkDimensions);
```

Ülesande implementatsioon on järgnev:

```
286. int height = chunkDimensions*rows;
287. int wid = chunkDimensions*columns;
288. BufferedImage newImage = new BufferedImage(wid,height,
     BufferedImage.TYPE_INT_ARGB);
289. Graphics2D g2 = newImage.createGraphics();
290. int c = 0;
291. for(int i = 0; i < rows && c < images.size(); i++){
292.       for(int j = 0; j < columns && c < images.size(); j++){
293.             g2.drawImage(images.get(c++), null, chunkDimensions*j,
              chunkDimensions*i);
294.       }
295. }
296. g2.dispose();
297. return newImage;
```

Nüüd jääbki üle ainult fail salvestada ja meie helifaili ribakoodi genereerimine on lõppenud.

```
100. ImageIO.write(joinedImage, "png", new File(imagePath));
```

Väljundfail tekib *imagePath* muutja väärtusega määratud asukohta.

3.2 Helifaili genereerimine ribakoodist

Helifaili genereerimine toimub pöördprotsessi põhiselt. Ainukeseks erinevuseks ongi see, et pöördprotsess lõpeb zip formaadi lahtipakkimisega, mille tulemusena tekib meile OGG formaadis fail. Sisendformaadis olevat faili me tagasi ei saa tekitada.

3.2.1 Ribakoodist andmete tagastamine

Helifaili genereerimise peameetodiks on *generateAudio(String imagePath, String filePath)*, kus *imagePath* on sisendiks antud pildifail ja *filePath* on helifaili loomiseks mõeldud asukoht. *generateAudio* esimeseks operatsiooniks on sisendpildifaili lahtiharutamine väiksemateks ribakoodideks. See operatsioon kävitub real 60:

```
60. ArrayList<BufferedImage> inputImages = splitJoined(imagePath, chunkDimensions);
```

Selle meetodi sisu katab täpselt samu Java-siseseid pakette mida me väiksemate ribakoodide liitmisel kasutasime:

```
196. BufferedImage image = ImageIO.read(new FileInputStream(filePath));
197. ArrayList<BufferedImage> images = new ArrayList<BufferedImage>();
198. int chunkHeight = image.getHeight()/chunkDimensions;
199. int chunkWidth = image.getWidth()/chunkDimensions;
200. for(int i = 0; i < chunkHeight; i++){
201.       for(int j = 0; j < chunkWidth; j++){
202.             images.add(image.getSubimage(j*chunkDimensions, i*chunkDimensions,
                            chunkDimensions, chunkDimensions));
203.       }
204. }
```

Olulist rolli mängib muutuja *chunkDimensions*, mis seab alam-piltide mõõtmed. Edasiste arenduste käigus tuleks see lahendus ümber ehitada suhteliseks, kus me ei kasutaks enam kindlalt määratud suurusi, vaid lõikame sisendfaili mitmeks väiksemaks osaks, leides eelnevalt, mitmest väiksemast pildist meie sisendfail koosneb. Operatsiooni lõpus tagastame massiivi väiksemate ribakoodidega millest on nüüd võimalik andmeid ükshaaval välja tõmmata. Seda teeme meetodiga *getDataFromImages*:

```
61. String outputString = getDataFromImages(inputImages, hintMap);
```

getDataFromImages kaasab peamise operatsioonina meetodit *readQRCode* real 149:

```
149. String part = readQRCode(inputImages.get(i), hintMap);
```

Selle eesmärgiks on ZXING teegiga ribakood tuvastada ja sellest informatsioon hankida:

```
271. BufferedImageLuminanceSource t = new BufferedImageLuminanceSource(image);
272. BinaryBitmap binaryBitmap = new BinaryBitmap(new HybridBinarizer(t));
273. Result qrCodeResult = new MultiFormatReader().decode(binaryBitmap, hintMap);
274. return qrCodeResult.getText();
```

Tulemuseks on sõne, mis koosneb sisseantud pildifaili sisust. Järgnevalt peame me selle lahti kodeerima ja seejärel ka lahti pakkima ning salvestama.

3.2.2 Andmete dekodeerimine

Andmete dekodeerimine on lahtipakkimisele eelnev protsess. Siin tuleb meil käesolev sõne muuta baidivooks, millest Java GZIPi pakett aru saab. Selleks jooksutame real 62 oleva koodi:

```
62. byte[] outputBytes = Base64.getDecoder().decode(outputString.getBytes());
```

Paneme tähele, et ribakoodist saadud andmete muutujast *outputString* tuleb eelnevalt töötlemata baidivoog kätte saada. Seda teeme lihtsa Java *String.getBytes()* implementatsiooniga.

3.2.3 Andmete lahtipakkimine

Nüüd kus meil on kokkupakitud baidivoog olemas, peame me ta ka lahti pakkima.

```
63. byte[] decompressed = decompressBytes(outputBytes);
```

decompressBytes meetod hõlmab endas jällegi Java-sisese GZIP paketi implementatsiooni. Siin peame andmed lahti pakkima ja need tagastama faili salvestamiseks.

```
231. ByteArrayInputStream bis = new ByteArrayInputStream(compressed);
232. GZIPInputStream gis = new GZIPInputStream(bis);
233. byte[] buffer = new byte[1024];
234. ByteArrayOutputStream out = new ByteArrayOutputStream();
235. int len;
236. while ((len = gis.read(buffer)) > 0) {
237.       out.write(buffer, 0, len);
238. }
239. gis.close();
240. out.close();
241. return out.toByteArray();
```

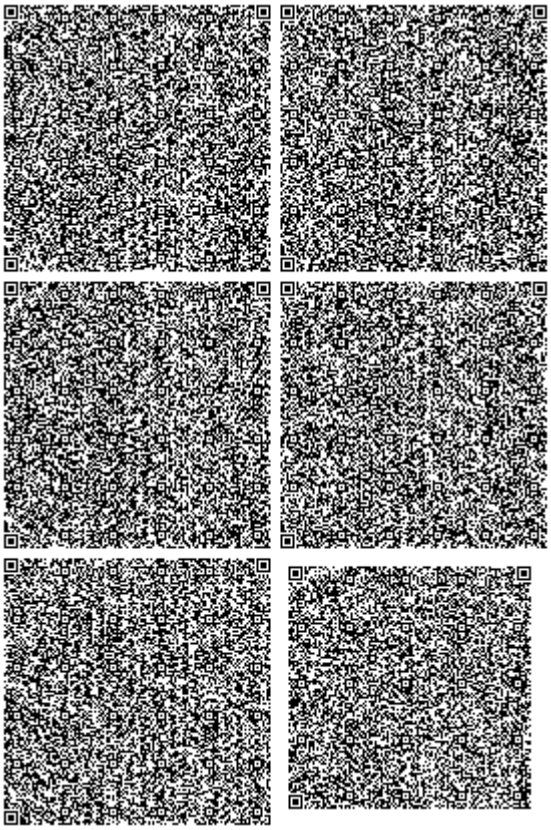
Nüüd jääb üle helifail salvestada ja sellega on ribakoodist heli genereerimine lõppenud.

```
64. FileOutputStream outAudioFile = new FileOutputStream(filePath);
65. outAudioFile.write(decompressed);
66. outAudioFile.close();
```

Helifail tekib muutujaga *filePath* määratud asukohta.

3 Testimine ja programmi tulemuste analüüs

Eduka testi sooritamiseks pidi programm sisendfaili põhjal edukalt ribakoodi genereerima ja seda ka lugema. Seda ta ka väga edukalt tegi. Ribakoodi ennast näeb Pildil 6. Testitava arvuti protsessoriks on Intel 2600k 3.4 GHz ja mäluks 1333 MHz DDR3 muutmälu.



Pilt 6: Programmi väljund ribakoodi genereerimisel

Test 1

Käesoleva ribakoodi sisendiks oli helifail järgnevate omadustega

- WAV formaat
- 202,796 baiti
- 1411. biti sagedusega
- 1.13 sekundit kestev salvestus

Väljundiks on Pildil 6 esinev ribakood omaudustega:

- PNG formaat
- 22,273 baiti
- kõrgus 414 pikslit
- laius 414 pikslit
- biti sügavus 32

Keskmiselt kulus antud ribakoodi genereerimiseks **630 millisekundit**. Selle lugemiseks kulub keskmiselt **240 millisekundit**.

Test 2

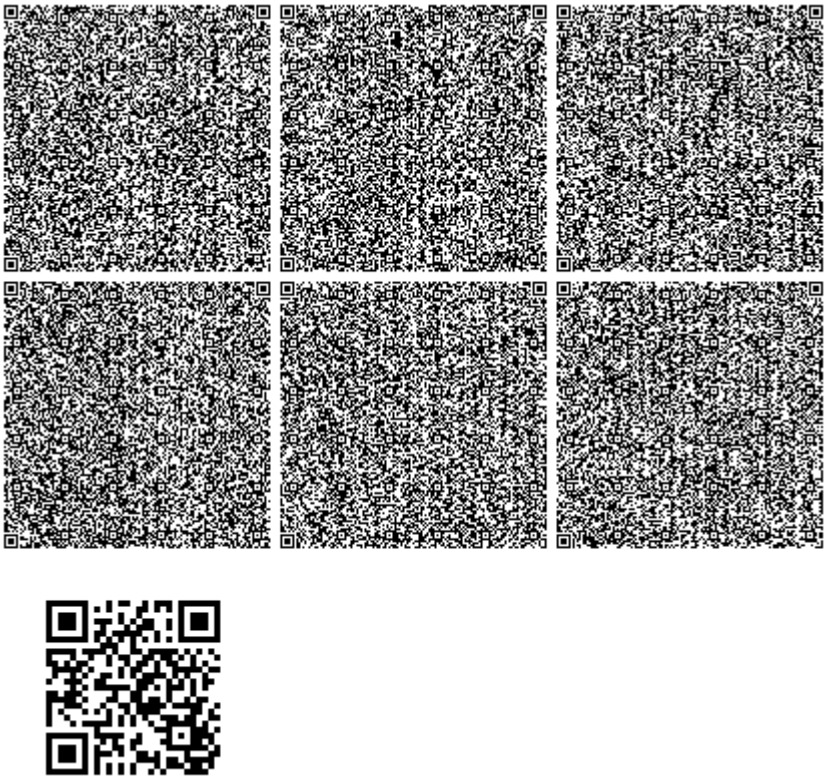
Teiseks testiks oli järgnev helifail:

- WAV formaat
- 253,996 baiti
- biti sagedus 1411
- 1.17 sekundit kestev salvestus

Väljundiks on Pildil 7 esinev ribakood omadustega:

- PNG formaat
- 24,030 baiti
- kõrgus 414 pikslit
- laius 414 pikslit
- biti sügavus 32

Keskmiselt kulus antud ribakoodi genereerimiseks **641 millisekundit**. Selle lugemiseks kulub keskmiselt **232 millisekundit**.



Pilt 7: Teise testi tulemusena esinev ribakood

Test 3

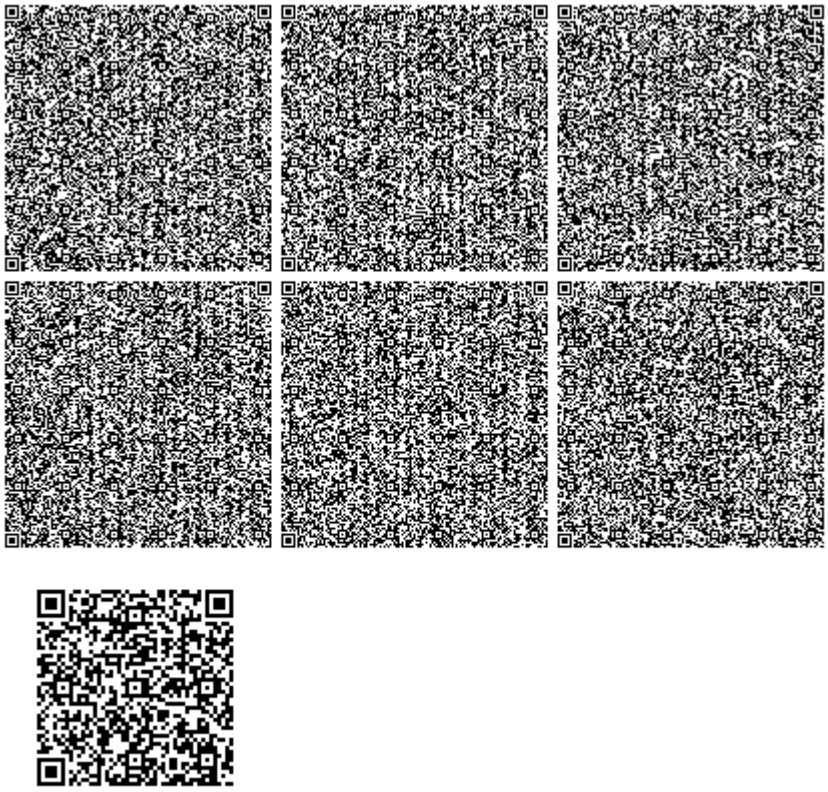
Kolmandal testil kasutasin samasugut helifaili nagu testis 2, kuid seekord MP3 formaadis helifaili:

- MP3 formaat
- 58,743 baiti
- biti sagedus 320
- 1.17 sekundit kestev salvestus

Väljundiks on Pildil 8 esinev ribakood omaudustega:

- PNG formaat
- 24,528 baiti
- kõrgus 414 pikslit
- laius 414 pikslit
- biti sügavus 32

Keskmiselt kulus antud ribakoodi genereerimiseks **664 millisekundit**. Selle lugemiseks kulub keskmiselt **250 millisekundit**. Väljundiks on vastav ribakood:



Pilt 8: MP3 formaadis oleva helifaili väljund

4 Kokkuvõte

Lõpptulemusest on näha, et etteseatud eesmärk sai saavutatud. Programmile saab anda sisendfaili ning temast ribakoodide pilt genereerida. Samuti saab ka sellest pildist helifaili tagastada.

Kuna kasutatav ribakoodi algoritm, QR Code, ei ole üldse mõeldud helide salvestamiseks, siis tulemuseks saadud ribakoodi suurus on päris suur. Näiteks kui meie eesmärk oleks 5 sekundi pikkuse heli salvestamine paberile, siis selleks oleks vaja ühte tervet paberilehte.

Kui võrrelda saadud tulemust olemasolevate lahendustega, siis heli kvaliteet on siin kindlaks eeliseks. Näiteks PhonoPaper'i näitevideote helikvaliteet on märgatavalt kehvem, vaatamata sellele, et heli pikkus on mitmeid kordi suurem.

Üheks huvitavaks aspektiks pildifaili genereerimise algoritmi juures on see, et olenemata helifaili suurusest, suudab programm ikkagi tulemuspildi genereerida. See tähendab, et väljundina, sõltuvalt väga suurest sisendist, võib tekkida lõpmata pikk ribakoodide fail. Teisisõnu - piirangut pildifaili genereerimisel pole.

Kuna programm eeldab hetkel, et pildifailist heli genereerimisel on sisendfail kindla laiuse ja kõrguse suhtega, tuleks järgmise arendusprotsessiga luua pildituvastaja, mis leiab, mitmest väiksemast ribakoodist pildifail koosneb. Selle eesmärgi saavutamiseks peaks ka pildifaili genereerimise protsessi natuke muutma, kus genereeritava pildi peale lisatakse sobivad laiuse ja kõrguse märgendid.

Antud programmile sobiks aga kõige enam värviliste ribakoodide implementatsioon [13], sest nende mahutavus on kuni neli korda suurem. Kahjuks pole aga hetkel sellised lahendused avalikult kättesaadavad. HCCB ehk Kõrge Tihendusega Värviline Ribakood on hetkel Microsofti poolt arendamisel.

Käesolevat programmi saab nüüd edasi viia pea kõikjale kus Java virtuaalmasin töötada suudab. Põhiliseks sihtplatvormiks oleksid aga nutitelefonid. Samuti on antud programmi lähtekood vägagi võimalusterohkem edasiste arenduste jaoks. Programmis on efektiivsed kodeerimise ja, kokkupakkimise meetodid ning palju muud.

Viited

1. QR Code specifications http://www.qrcode.com/en/about/standards.html [Võrgumaterjal] [Viidatud: 1. Mai 2016].
2. 2D COLOR BARCODES FOR MOBILE PHONES http://www.tmrfindia.org/ijcsa/v8i19.pdf [Viidatud: 1. Mai 2016].
3. Vorbis I specification http://www.xiph.org/vorbis/doc/Vorbis_I_spec.html [Võrgumaterjal] [Tsiteeritud: 03. jaanuar 2016] .
4. GZIP https://docs.oracle.com/javase/7/docs/api/java/util/zip/GZIPInputStream.html [Viidatud: 1. Mai 2016].
5. Phonopaper http://www.warmplace.ru/soft/phonopaper/ [Võrgumaterjal] [Viidatud: 13. Märts 2016].
6. Design and realization of 2D color barcode with high compression ratio http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5540872&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5540872 [Võrgumaterjal] [Viidatud: 13. Märts 2016].
7. Effect of copying and restoration on color barcode payload density http://dl.acm.org/citation.cfm?id=1600222 [Viidatud: 1. Mai 2016].
8. 2D Bar Codes Reading: Solutions for Camera Phones http://waset.org/publications/5194/2d-bar-codes-reading-solutions-for-camera-phones [Võrgumaterjal] [Viidatud: 13. Märts 2016].
9. High capacity color barcodes using dot orientation and color separability http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1335172 [Võrgumaterjal] [Viidatud: 13. Märts 2016].
10. The Ogg Encapsulation Format Version 0 https://tools.ietf.org/html/rfc3533 [Võrgumaterjal] [Viidatud: 13. Märts 2016].
11. Ogg Vorbis and MP3 https://webdocs.cs.ualberta.ca/~c603/latex/LaTeX_docs/article2/oggVorbis.pdf [Võrgumaterjal] [Viidatud: 13. Märts 2016].
12. SoundPaper http://www.soundpaper.com/ [Võrgumaterjal] [Viidatud: 11. Aprill 2016].
13. About High Capacity Color Barcode Technology http://research.microsoft.com/en-us/projects/hccb/about.aspx [Viidatud: 1. Mai 2016].
14. JAVE http://www.sauronsoftware.it/projects/jave/ [Võrgumaterjal] [Viidatud: 11. Aprill 2016].
15. ZXING https://github.com/zxing/zxing [Võrgumaterjal] [Viidatud: 11. Aprill 2016].

16. Base16, Base32, Base64 encoding specification https://tools.ietf.org/html/rfc4648 [Võrgumaterjal] [Viidatud: 1. Mai 2016].
17. Java http://www.oracle.com/technetwork/java/index.html [Võrgumaterjal] [Viidatud: 1. Mai 2016].
18. PDF-417 specification http://www.morovia.com/manuals/PDF417-Font-ware-Writer-SDK-4/chapter.overview.php [Võrgumaterjal] [Viidatud: 4. Mai 2016].
19. DataMatrix Guideline http://www.gs1.org/docs/barcodes/GS1_DataMatrix_Guideline.pdf [Võrgumaterjal] [Viidatud: 5. Mai 2016] .

Lisad

I Programmi lähtekood

Programmi lähtekood on pakendatud RAR arhiivi ning on üles riputatud koos selle tööga.

II Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks.

Mina, **Laimis Dalke**,

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

   Tarkvara audio taasesituseks piltkoodist,

   mille juhendaja on Eero Vainikko,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

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.

Tartus, 11.05.2016