TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Magnus Mihkel Peterson 123759 IABB

RIISTVARALISE KIIRENDUSEGA KASUTAJALIIDESE LOOMISE TEEK

bakalaureusetöö

Juhendaja: Martin Rebane
MSc
lektor

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Töö eesmärk on luua C++ programmeerimiskeeles kasutajaliidese teek, mis toetaks riistvaralist kiirendust.

Põhilised probleemid millega tegelesin selles töös on mitmekihilise vaate loomine ning vaatel olevate objektide interaktiivseks muutmine. Samuti oli probleemiks programmi töökiirus ning stabiilsus.

Selle töö lõpptulemuseks on teek, millega on võimalik hõlpsalt luua kasutajaliideseid ning millesse on sisse kirjutatud sisendite käitlemine, heli taasesitus, enimlevinumad kasutajaliidese objektid ning nende transformeerimise loogika.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 7 peatükki ja 20 joonist.

Abstract

The aim of the thesis is to create a library for the of graphical user interfaces in C++ that uses hardware acceleration.

Main problems I dealt with were creating a multi-layered view and making displayed objects interactive. There was also a problem with the program speed and stability.

The result of this thesis is a library that can be used to easily create graphical user interfaces. The library will feature input handling, sound playback, common graphical user interface objects and their transformation logic.

The thesis is in Estonian and contains 30 pages of text, 7 chapters and 20 figures.

Lühendite ja mõistete sõnastik

Verteks	***Vertex*** Punkt ruumis.
Primitiiv	***Primitive*** Verteksite kogu, mis moodustavad üksiku kolmemõõtmelise olemi.
Vertikaalne sünkroniseemine	***Vertical synchronization*** Kaadrisageduse sünkroniseerimine monitori värskendussagedusega.
Kutsegraaf	***Call graph*** Kutsegraaf on suunatud graaf, mis tähistab arvutiprogrammi alamrutiinide omavaheliste väljakutsumiste seoseid.
FPS/Kaadrisagedus	***Frames per second*** Kaadrisagedus on kuvatavate kaadrite (videot või animatsiooni moodustavate üksikpiltide) arv sekundis. [1]

Jooniste nimekiri

Sisukord

1. Sissejuhatus

Kasutajaliidesed on programmide üks olulisemaid aspekte, sest lõppude lõpuks on tarkvaraarendus lõppkasutajate jaoks. Stiil, keerukus, kasutusmugavus, töökindlus ning järjepidevus kujundavad kasutajate arvamuse terve rakenduse kohta.

1.1 Taust ja probleem

Kasutades enimlevinuid kasutajaliidese teeke, on disainerite ja front end arendajate töö piiratud valitud teegi võimalustega või ressursside kasutusega. Keerukad kasutajaliidesed kasutavad liigselt protsessori jõudlust ning ei pruugi piisavalt kiirelt toimida, et programm toimiks nõnda, nagu looja seda ette nägi. Kui kasutajaliidesele panna palju animeeritud kujundeid, ei jooksuta isegi uuemad arvutid programmi sujuvalt, sest enamik kasutajaliidese teeke ei kasuta riistvaralist kiirendust. Probleemi paremaks mõistmiseks võib võtta näiteks heli taasesitamise programmi, mis visualiseerib esitatud heli näiteks tuhande püstkriipsu liigutamisega või suuruse muutmisega, määrates igale kriipsule helisageduse vahemik, mida hakkab kriips representeerima. Taoline programm ei ole kasutatav, sest protsessor ei suuda nii nõnda palju kujutisi animeerida piisavalt kiiresti, et tulemus oleks sujuv.

Selle probleemi lahendamiseks on appi võetud Direct3D 9 teek, millega on võimalik graafikaprotsessori jõudlust ära kasutada ning kuvada väga kiirelt ka kõige keerukamad kasutajaliidesed.

Kuigi on olemas palju erinevaid kasutajaliidese teeke nagu näiteks C++ keeles kirjutatud wxWidgets [2], .NET platvormi Windows Forms [3] ja Java Swing [4], kuid ükski neist ei kasuta visualiseerimiseks graafikaprotsessorit.

1.2 Ülesande püstitus

Töö eesmärk on luua kasutajaliidese teek, milles on juba olemas enimkasutatavad objektid, mida saab kergelt ekraanile kuvada. Teegi abil tehtav kasutajaliides peab kasutama riistvaralist kiirendust ning olema kiirem kui mõni teine riistvaralise kiirenduseta kasutajaliidese teek.

1.3 Metoodika

Teegi ning selle põhjal tehtud kasutajaliidese kirjutamiseks kasutatakse C++ programmeerimiskeelt ja DirectX 9.0c teekide kogu.

Teegi töökiiruse võrdlemiseks kirjutatakse teegi abil lihtne kasutajaliides, mida võrreldakse samuti C++ programmeerimiskeeles kirjutatud kasutajaliidesega, mis kasutab aga wxWidgets teeki.

1.4 Ülevaade tööst

Teegi lähtekood asub aadressil http://www.github.com/zhardas/engine/

Peatükis 2 kirjeldan ühe keerukama kuvatava objekti ülesehitust, tuues välja sellega seotud kutsegraafid ning selgitan selle pärilust. Selle peatüki esimeses alampunktis kirjeldan ka kuidas on selliste objektidega seotud virtuaalseid meetodeid täiendatud objektide vajaduste täitmiseks. Sellele järgnevas alampunktis on aga selgitatud sündmuste käsitlemist.

Peatükis 3 selgitan teegi elutsüklit, mis on jagatud kaheks suuremaks osaks: uuendamine ja visualiseerimine. Samuti on lühidalt kirjeldatud sellega seotud vertikaalset sünkroniseerimist, mis mõjutab programmi kaadrisagedust.

Peatükis 4 on lahti mõtestatud graafikaprotsessori abil pildi kuvamine ning sellega seotud osad. Kuna sellega seonduva osa maht on tegelikult äärmiselt suur, on kirjeldatud ainult antud töös peamiselt kasutatud osad, millest peaks piisama töö toimimise mõistmiseks.

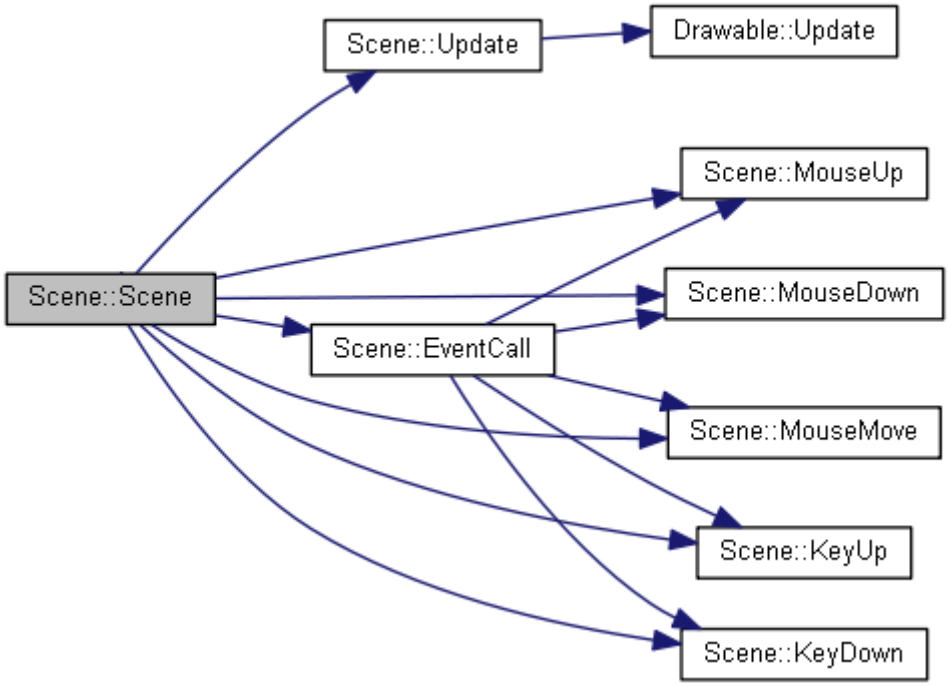
Töö peatükis 5 kirjeldan tehtud töö ja wxWidgets teegi põhjal tehtud kasutajaliideseid ning nende võrdluse tingimusi ja tulemusi.

Peatükk 6 tegeleb arendusvaatega, milles tuuakse välja teegi mõningad võimalikud edasiarendused, mis muudaksid lõpptulemust veelgi kiiremaks ja/või mugavamaks.

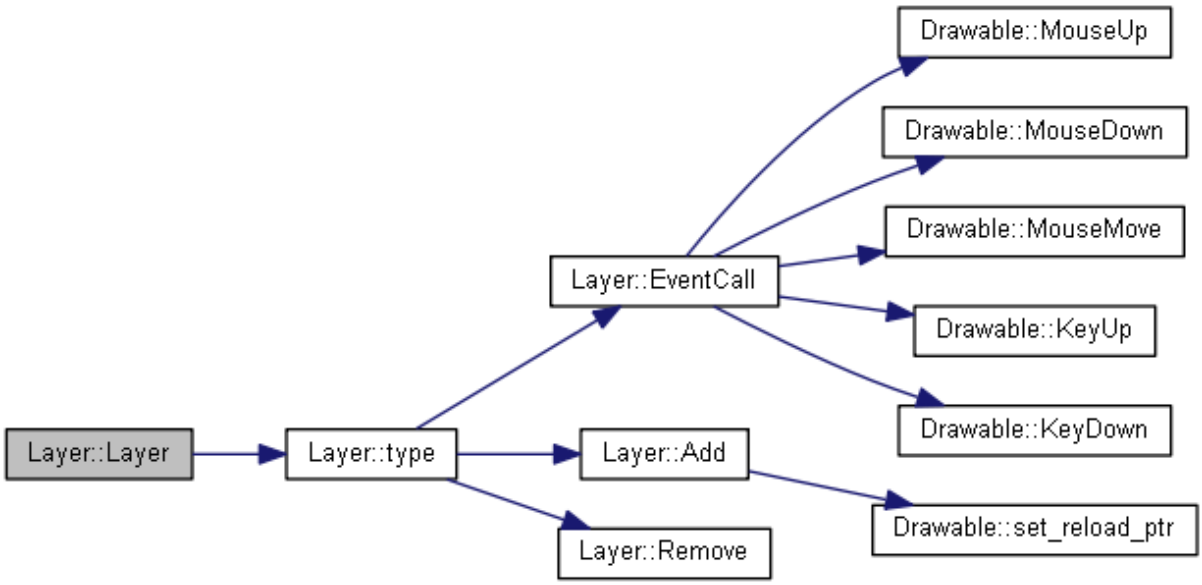
2. Kuvatava objekti ülesehitus

Teegi kuvatava objekti ülesehituse selgitamiseks võtame näiteks tekstivälja klassi *TextBox*, kuna see on üks keerukamaid kuvatavaid objekte ning sisaldab kõiki tähtsamaid osi. Tekstivälja muudab keerukas see, et see koosneb mitmest kuvatavast objektist: tekstivälja tagataust ning sellele kuvatav tekst ning nende kuvamisprotsess on vägagi erinev. Lisaks sellele on vaja tekstiväljal kirjutamiseks klaviatuuriga seotud sündmuste loogikat.

Kõigepealt on määratud *Game* objektile kuvatav *Scene* objekt (Joonis 1), mis sisaldab loendit *Layer* (Joonis 2) objektidest.

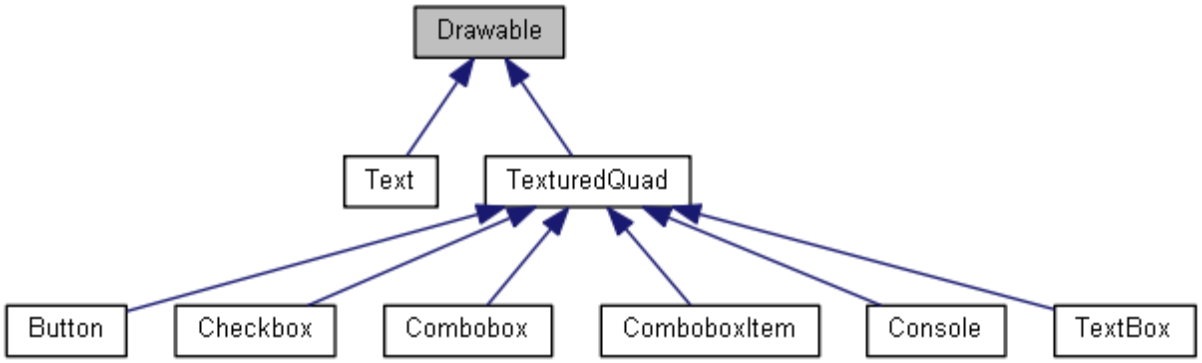


Joonis 1 - *Scene* klassi kutsegraaf



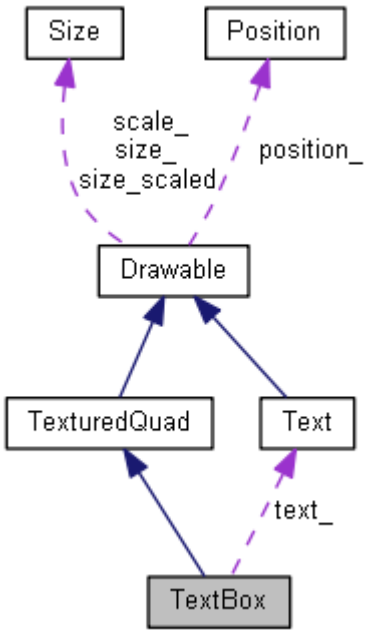
Joonis 2 - *Layer* klassi kutsegraaf

Layer objektid sisaldavad omakorda loendit *Drawable* superklassi põhjal tehtud objekte, millest üks on ka hetkel näiteks võetud *TextBox* (Joonis 3).



Joonis 3 - *Drawable* klassi pärilus

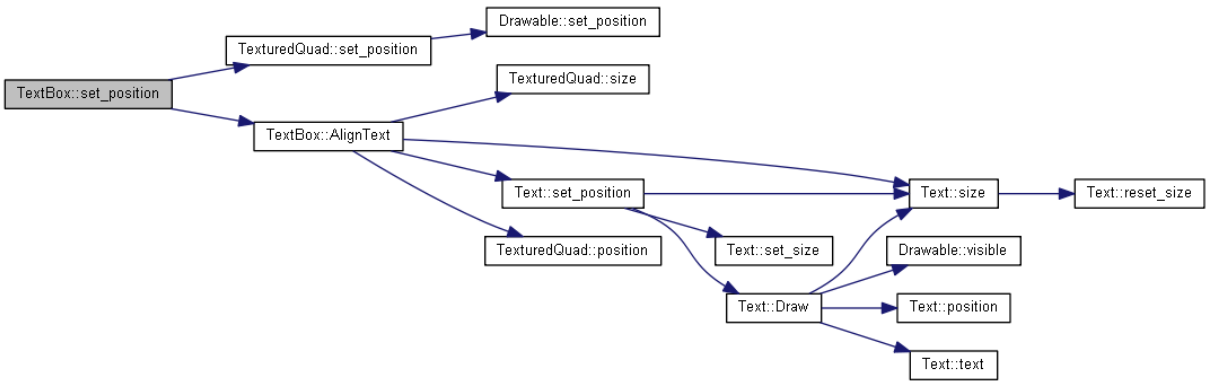
Klass *TextBox* on klassi *TexturedQuad* alamklass, mis on omakorda klassi *Drawable* alamklass (Joonis 4).



Joonis 4 - *TextBox* klassi hierarhia

2.1 Asukoht ja suurus

Peamiselt on vaja sellistel alamklassidel täiendada suuruse ja asukoha saamise ning seadmisega seotud virtuaalseid meetodeid ning seda eriti siis, kui klass sisaldab ka teist sarnast alamklassi. Antud juhul sisaldab *TextBox* ka *Text* klassi põhjal tehtud objekti. Kui tekstivälja esmalt tekitatakse või kui selle suurust või asukohta muudetakse, on vaja ka selles sisalduvaid teisi kuvatavaid objekte vastavalt liigutada ja/või asukohta muuta. Kuna asukoha ja suuruse muutmine toimuvad vägagi sarnaselt, võtame ainult asukoha muutmise näite. Esmalt muudetakse objekti enda (ehk antud olukorras tekstilahtri tagatausta) asukoht ning seejärel paigutatakse tekstiobjekt vastavalt tagatausta uuele asukohale ja suurusele (Joonis 5).



Joonis 5 - klassi *TextBox* asukoha muutmise kutsegraaf

2.2 Sündmused

Loodud teek toetab mitut hiirega ja klaviatuuriga seotud sündmust. Sündmused saavad alguse *InputManager* klassist, mis loeb Windowsi sõnumeid, et tuvastada hiire või klaviatuuri tegevusi. Kui leiti vastav sõnum sündmuse alustamiseks, kutsutakse välja vastav sündmus hetkel aktiivne olevas stseenis (*Scene*). Kui võtta näiteks hiire nupu allavajutus, kutsutakse *Scene* klassis välja meetod *MouseDown* (Joonis 1). Meetodis käiakse tsükliga läbi kõik stseenil olevad kihid (*Layer*) ning kutsutakse *Layer* klassis välja sama meetod *EventCall*, mis käib tsükliga läbi kõik kihil olevad *Drawable* objektid ning kutsub ka neil välja sama nimega meetodi, mis kutsuti ka *Scene* klassis (Joonis 2). *Drawable* klass sisaldab loendit iga erinevat tüüpi sündmuse jaoks. Loendites saab hoida funktsiooni viiteid, mis kutsutakse välja sündmuse aktiveerimisel. Selline süsteem võimaldab kirjutada anonüümseid funktsioone [5], ning võimaldab kergelt lisada uusi funktsioone keset programmi tööd.

Alljärgnev on *TextBox* klassis klaviatuuri nupuvajutuse sündmuse funktsiooni lisamine.

```
events_key_down_.push_back([this](const uint8_t &key) {
  if (!is_active_ || !is_editable)return false;
  if (key == VK_BACK) {
    if (text_->text().length() > 0) {
      text_->set_text(text_->text().substr(0, text_->text().length() - 1));
      AlignText();
      return true;
    }
  } else {
    std::stringstream ss;
    ss << text_->text() << key;
    text_->set_text(ss.str());
    AlignText();
    return true;
  }
  return false;
});
```

Funktsioon saab parameetriks täisarvu, mille väärtus võib olla 0 kuni 255 ning see tähistab vajutatava nupu koodi. Kui tekstilahter ei ole aktiivne või muudetav, lõpetab funktsioon oma tegevuse. Vastasel korral jätkab funktsioon tööd ning kontrollib sisestatud nuppu. Kui tegemist on klahviga *Backspace*, kustutatakse tekstivälja tekstist viimane täht ära. Vastasel korral teisendatakse parameetriks antud täisarv tekstikujule ning lisatakse see olemasolevale tekstile. Mõlemal juhul paigutatakse teksti positsioon uuesti uue tekstipikkusega arvestamiseks. Kui funktsioon on töö teinud, tagastatakse tõeväärtus. Kui tagastatakse tõene väärtus, ei käsitleta peale funktsiooni tööd enam seda sama sündmust edasi. Väär väärtuse korral aga kutsutakse välja ülejäänud sündmusega seotud funktsioonid. Taoline lahendus on mõeldud seepärast, et

kui tekib olukord, kus kuvatakse mitu asja üksteise peale, et nendest kõige peamine ainult reageeriks vastavale sündmusele. Samuti hoiab see ära üleliigse töö tegemise.

3. Elutsükkel

Elutsükkel koosneb kahest peamisest osast: uuendamine ja visualiseerimine. Kuna teek kasutab vertikaalset sünkroniseerimist rebimise efekti ärahoidmiseks [6] (Joonis 6), toimuvad visualiseerimine ja uuendamine vahetult kordamööda, kuigi elutsüklisse on kirjutatud ka vastav loogika vertikaalse sünkroniseerimiseta toimimiseks. Kahe programmi võrdluse ajaks on aga vertikaalne sünkroniseerimine välja lülitatud, et oleks võimalik kuvada maksimaalne kaadrisagedus.

Joonis 6 - rebimise efekt [6]

3.1 Uuendamine

Uuendamise käigus uuendatakse hetkel aktiivne olev stseen. Stseeni uuendamise käigus käiakse tsüklina läbi kõik stseenil olevad kihid ning nendel olevad kuvatavad objektid, mis seejärel uuendatakse. Kui objekt sisaldab teisi kuvatavaid objekte, käiakse need samuti läbi ning uuendatakse ja nõnda edasi kuni kõik üksteise sees olevad kuvatavad objektid on uuendatud. Kui objekt sisaldas ka teisi kuvatavaid objekte, kontrollitakse, et kas on vaja mõni objekt eemaldada ning seejärel see eemaldatakse kogust. Sama toimub ka stseeni uuendamisel, kui kontrollitakse kas kihtidel olevaid objekte on vaja eemaldada. Taoline objektide eemaldamise süsteem on mõeldud seepärast, et ei tekiks olukorda, kus on vaja kasutada objekti, mis on juba eemaldatud. Kuna kuvatavate objektide uuendamine toimub virtuaalse funktsioonina, on seda võimalik täiendada ning kirjutada keerukamat koodi. Näiteks, kui tegemist on tekstuuriga kuvatava objektiga, mis laiendab tavalist kuvatavat objekti, siis uuendamise funktsioon ka animeerib objekti, kui see on seatud animeeritavaks.

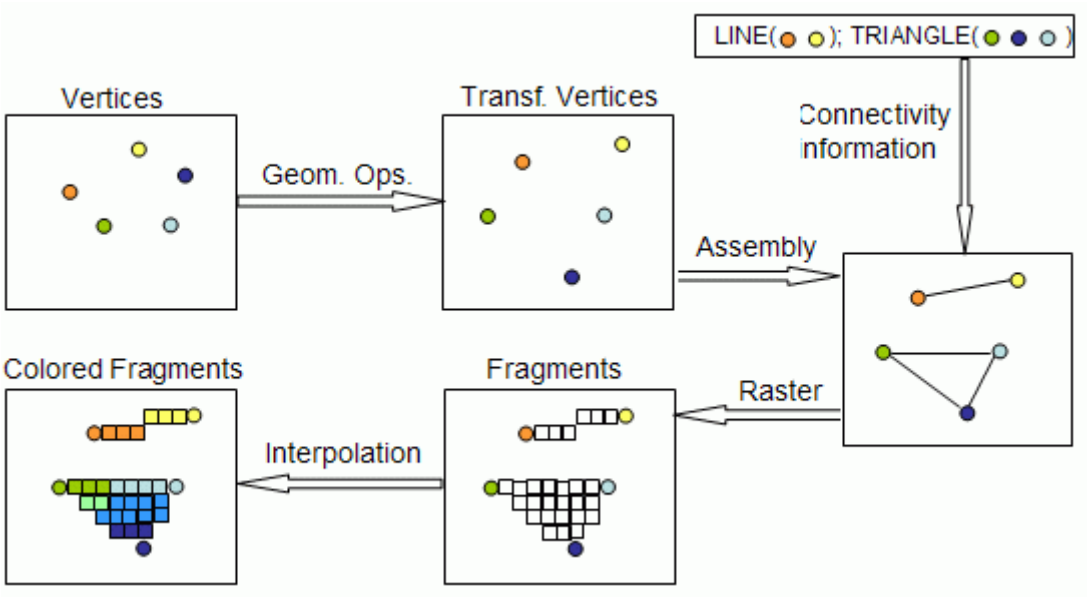
Kui teek on seadistatud kasutama lisamooduleid nagu näiteks heli taasesitus ning võrgundus, siis kutsutakse välja vastavad uuendusmeetodid.

3.2 Visualiseerimine

Uuendamise järel toimub visualiseerimine. Esmalt puhastatakse ekraan eelnevalt kuvatud objektidest ning seadistatakse uuesti vaate ja projektsiooni transformatsioon juhuks kui see on uuendamise käigus muutunud. Seejärel käiakse tsükliga läbi kõik stseenil olevad kihid. Kui tegemist on dünaamilise kihiga ehk tegemist on pidevalt muutuva verteksite puhvriga, luuakse uus puhver ning selleks kutsutakse välja kihil olevate objektide vastav virtuaalne meetod. Kui puhver on valmis, seadistatakse ruumi transformatsioon iga objekti korral ning kutsutakse välja objekti kuvamismeetod, mis kasutab loodud puhvrit primitiivide loomiseks, kutsudes välja teegi visualiseerimisklassi funktsioone.

4. Pildi kuvamine

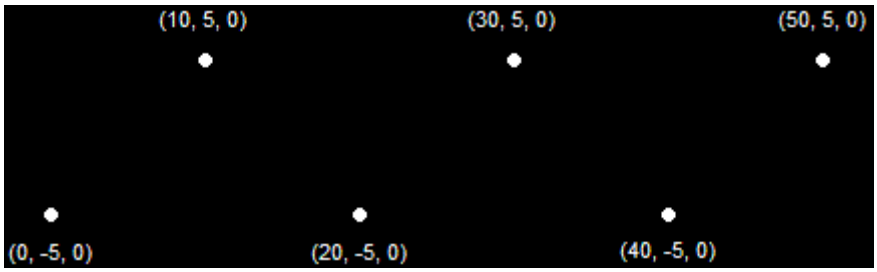
Pildi kuvamine toimub mitmes sammus. Esiteks on vaja graafikakaardi mällu salvestada iga kuvatava primitiivi verteksite info. Verteksid on kirjeldatud kolmemõõtmelise koordinaadiga, millele lisaks võib olla juures näiteks punkti värvus ja primitiivile asetatud tekstuuri koordinaadid. Info salvestatakse üldjuhul suures hulgas korraga graafikakaardi mällu ning nendest moodustatakse salvestamise järjekorras või indeksitega, kui mõnda punkti kasutatakse mitmekordselt, primitiivid. Primitiivideks on üksik verteks või mitmest verteksist ühendatud tervik, mis asetseb kolmemõõtmelises keskkonnas. Enne primitiivide loomist peab aga ära määrama transformatsioonimaatriksid, mille abil on võimalik verteksite asukohta nihutada, pöörata ning muuta omavahelist kaugust. Samuti saab maatriksitega ära määrata vaatenurga ning vaatevälja. Kui ruum, vaade ja projektsioon on ära määratud, tuleb enne primitiivi loomist määrata ära kasutatav tekstuur, kui verteksitele on määratud tekstuuri koordinaadid. Primitiivide loomise käigus toimub rasteriseerimine, mille käigus transformeeritakse kolmemõõtmeline ruum vastavalt vaatenurgale kahemõõtmeliseks pikslite info koguks. Pikslid värvitakse vastavalt punktidele määratud värvile ja/või tekstuurile, kui see info on punktide struktuuris olemas ning seejärel kuvatakse see ekraanile. Protsess on kuvatud ka alljärgneval joonisel (Joonis 7). [7]



Joonis 7 – visualiseerimise protsess [7]

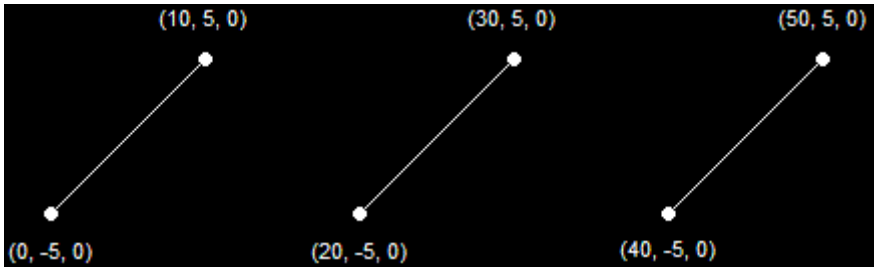
4.1 Primitiivid

Primitiivide moodustamiseks on kuus viisi. Esimene viis on moodustada punktide jada. Punktid ei ole omavahel ühendatud ning iga eraldiseisev punkt on primitiiv. Alljärgneval joonisel (Joonis 8) on seega kujutatud 6 primitiivi.

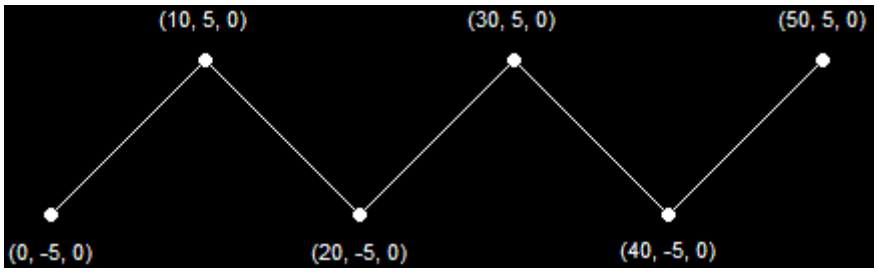


Joonis 8 - punktide nimekiri [8]

Järgnevad kaks võimalust on joonte jada ning joone riba. Nende erinevus seisneb selles, et jada korral tekitakse iga kahe punkti järel täiesti eraldiseisev joon, kuid riba korral tekitatakse iga järjestikuses oleva punkti vahele joon. Igat joont, olgu see teiste joontega ühenduses või mitte, loetakse eraldiseisvaks primitiiviks. Seega on alljärgnevatel joonistel kujutatud kolm primitiivi (Joonis 9) ja viis primitiivi (Joonis 10).



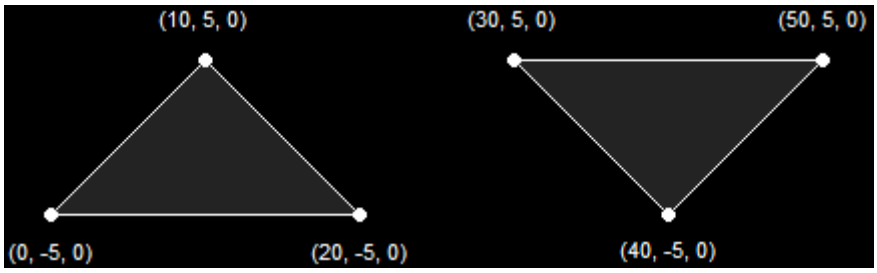
Joonis 9 - joonte nimekiri [8]



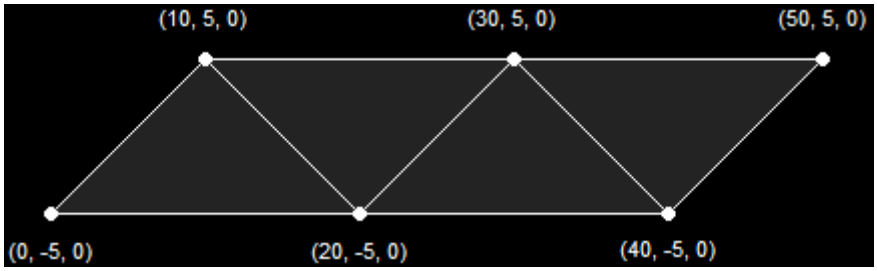
Joonis 10 - joonte riba [8]

Viimased kolm meetodit on kolmnurkade loomiseks. Nendeks on kolmnurkade jada, kolmnurkade riba ning kolmnurkade lehvik. Nagu joonte puhulgi, on kolmnurkade jada korral kõik loodud kolmnurgad eraldiseisvad ning kolmnurkade riba korral omavahel ühenduses. Kolmnurkade lehviku korral on samuti loodud kolmnurgad omavahel ühenduses, kuid erinevalt kolmnurkade ribast, on kolmnurgad ühendatud kõige esimese punkti ja viimati loodud

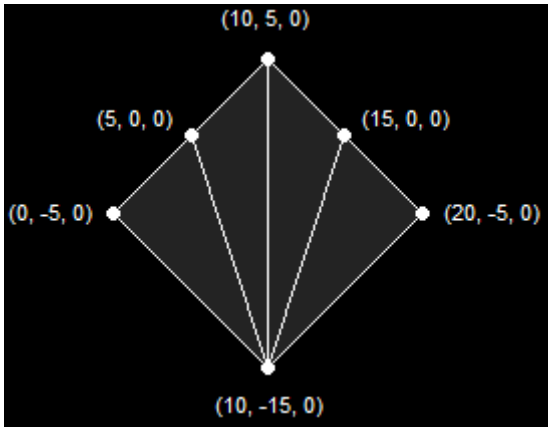
kolmnurga vaba punktiga. Sarnaselt joontega, loetakse igat kolmnurka eraldi primitiiviks, olenemata sellest, kas see on mõne teise kolmnurgaga ühenduses või mitte. Seega alljärgnevatel joonistel (Joonis 11, Joonis 12 ja Joonis 13) on vastavalt 2, 4 ja 4 primitiivi kujutatud.



Joonis 11 - kolmnurkade jada [8]



Joonis 12 - kolmnurkade riba [8]



Joonis 13 - kolmnurkade lehvik [8]

Käesolevas projektis on kasutusel ainult kolmnurkade riba, sest projektis on vaja kuvada ainult nelinurkseid objekte ning kolmnurkade riba korral on vaja määrata kõige vähem vertekseid, mis tähendab et see on nelinurksete objektide kuvamiseks kõige efektiivsem ja mugavam.

4.2 Transformeerimine

Verteksite transformeerimine koosneb kolmest osast: ruumi, vaate ja projektsiooni transformeerimine. Transformeerimiseks vajalike maatriksite loomiseks on DirectX 3D teegis

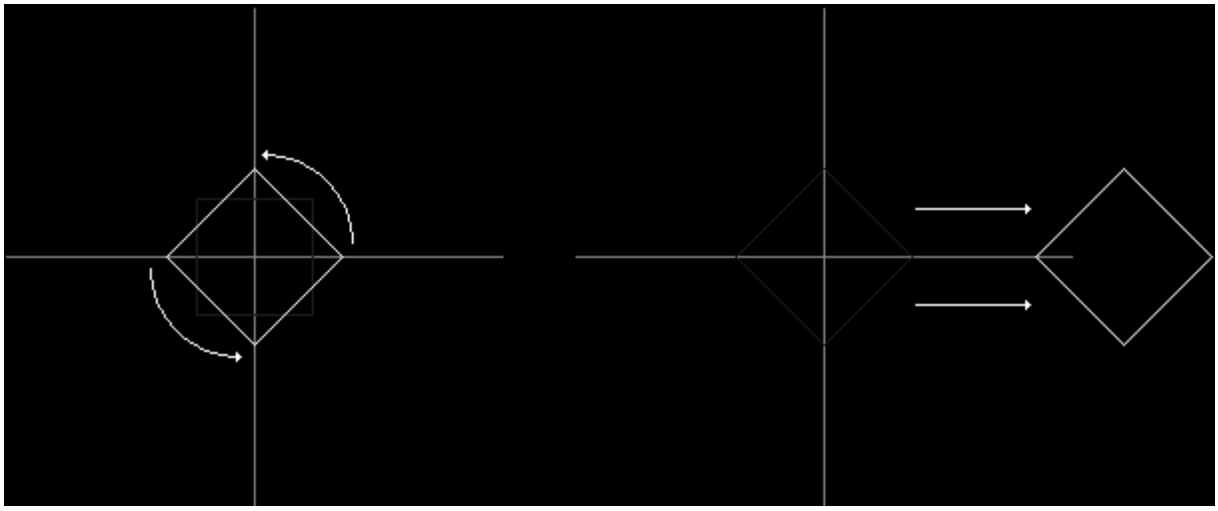
olemas vastavad meetodid, kuid neid on võimalik ka käsitsi luua kui selleks peaks vajadus olema. [9]

4.2.1 Ruumi transformatsioon

Ruumi transformeerimise käigus luuakse transformeerimismaatriks, millega korrutatakse sellele järgnevalt loodud primitiivide sees olevad verteksid. Ruumi transformeerimine koosneb kolmest osast: nihutamine, pööramine ning skaleerimine.

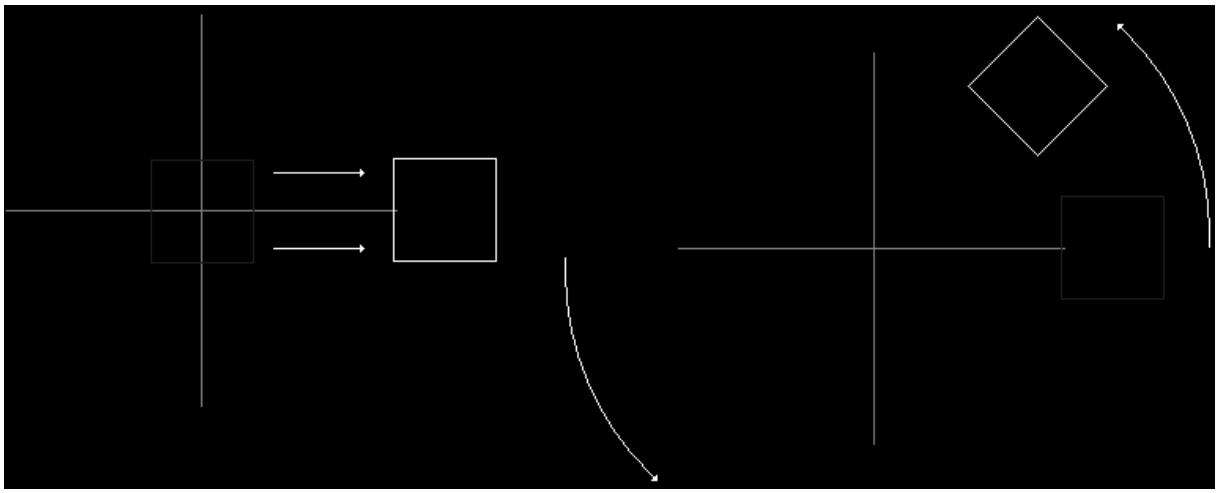
Nihutamine seisneb selles, et kõiki primitiivis olevaid vertekseid liigutatakse samas suunas sama palju ühikuid. Pööramise käigus liigutatakse primitiivis olevaid vertekseid mööda pöörlemistelje kujuteldavat ringjoont. Skaleerida saab kolmes mõõtmes eraldi kordajaga. [8]

Kuna ruumi transformeerimiseks saab määrata ainult ühe maatriksi ning korraga nihutades, pöörates ning skaleerides tekib kolm erinevat maatriksit, võetakse kasutusele maatriksite omadused ning need korrutatakse omavahel läbi. Selle tulemusena tekkinud maatriks sisaldab kõigi tegevuste andmeid ning määratakse ruumi transformatsioonimaatriksiks. Maatriksite korrutamisega tuleb aga arvesse võtta korrutamise järjekord. Selle selgitamiseks võtame näiteks ruudu pööramise ja nihutamise. Kui me korrutame pööramismaatriksi nihutamismaatriksiga, pööratakse ruut kohapeal ning seejärel nihutatakse seda määratud suunas (Joonis 14). [8]



Joonis 14 - ruudu pööramine ja nihutamine [8]

Kui aga täpselt samasuguseid tegevusi teha vastupidises järjekorras, nihutatakse ruut määratud suunas ning pööratakse objekti hoopis ümber tema esialgse pöörlemistelje (Joonis 15). [8]

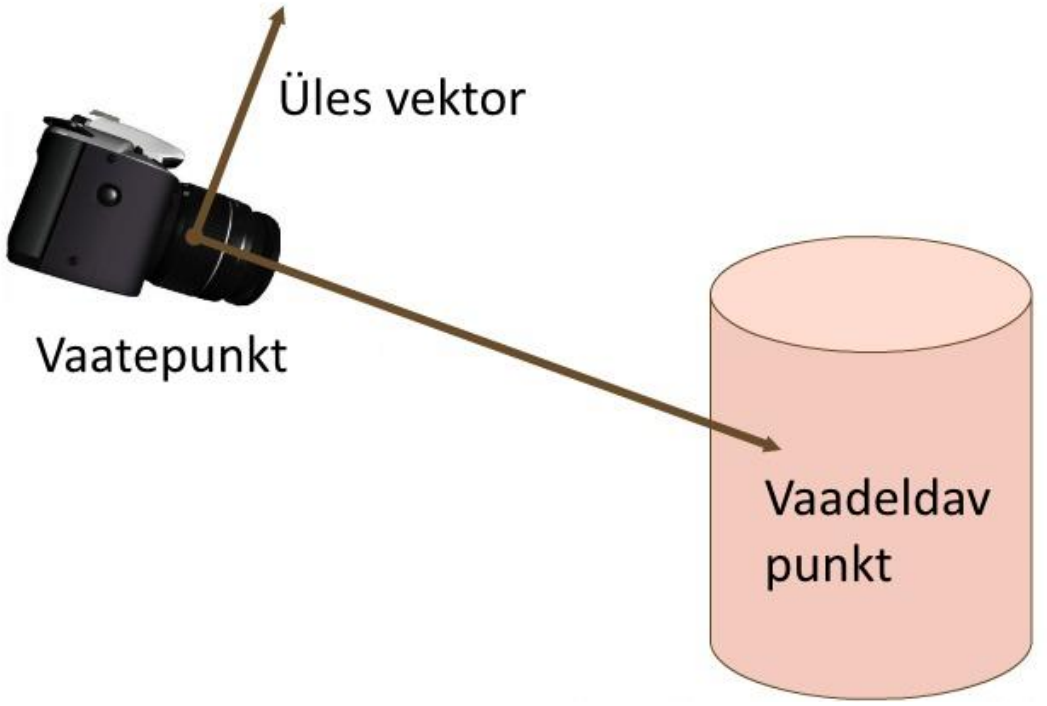


Joonis 15 - ruudu nihutamine ja pööramine [8]

Käesolevas projektis on transformeerimise järjekorraks kõigepealt skaleerimine, pööramine ning seejärel nihutamine.

4.2.2 Vaate transformatsioon

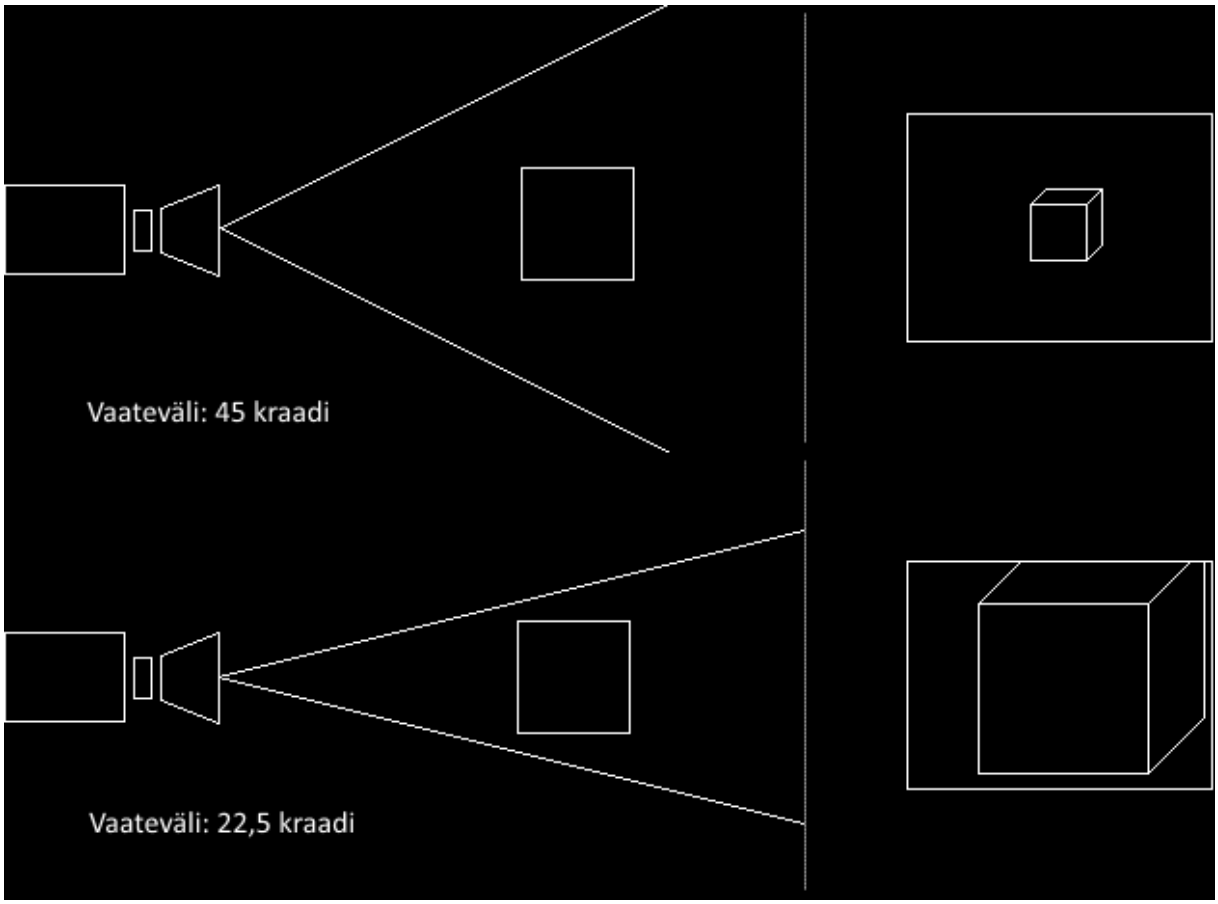
Vaate transformatsiooni võib pidada vägagi sarnaseks virtuaalse kaamera seadistamisega. See määrab ära kuskohas asub n-ö kaamera kolmemõõtmelises ruumis, kuhu poole see suunatud on ning kuidas on kaamera pööratud, määrates ära selle ülespoole suunatud vektori. [8]



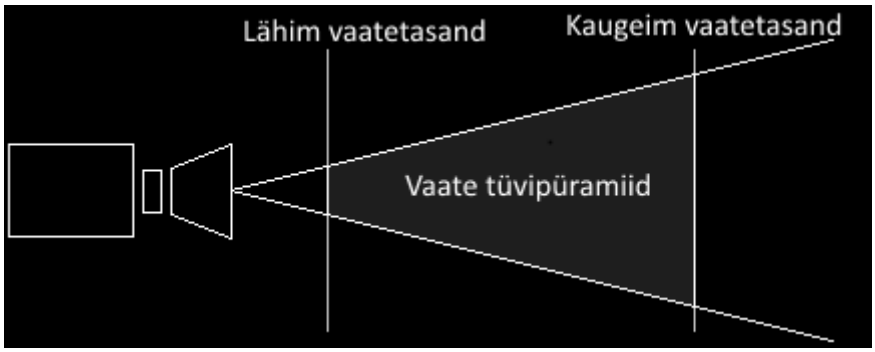
Joonis 16 - vaate transformeerimine kaamera näitel [10]

4.2.3 Projektsiooni transformatsioon

Kui vaate transformatsiooni võib pidada sarnaseks virtuaalse kaameraga, siis projektsiooni transformatsioon oleks selle kaamera läätse seadistamine. Transformatsiooniga määratakse ära kaamera vaateväli, kasutades selleks vaatevälja nurka (Joonis 17), kuvasuhet ning lähimat ja kaugeimat vaatetasandit. Kuvasuhe on ekraani laiuse ja kõrguse jagatis, et määrata ära piksli suurus, et kuvatav pilt näeks võimalikult täpne. Lähim ja kaugeim kuvatav kaugus tekitavad koos vaatevälja nurga ja kaamera asendiga tüvipüramiidi, millest väljaspool olevaid objekte ei kuvata (Joonis 18). [8]



Joonis 17 - vaateväli [8]



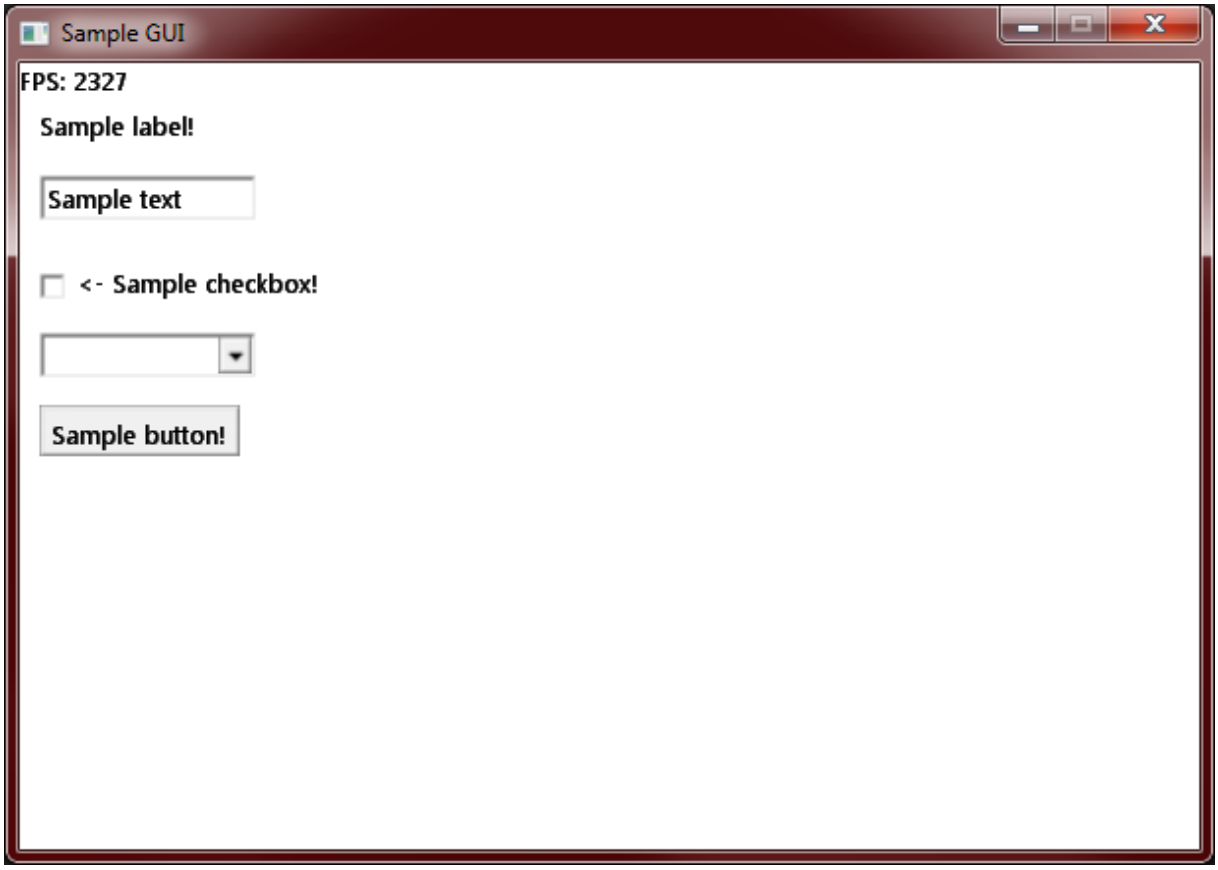
Joonis 18 - vaate tüvipüramiid [8]

Antud projektis on seadistatud vaade ja projektsioon nõnda, et nähtav ala oleks täpselt samade mõõtmetega kui selle projekti käigus valminud teegi abil tehtud programmi aken. Seega ükski kuvatud objekt ei deformeeru vaatest ja projektsioonist tingitult vaid kuvatakse vastavalt ruumi transformatsioonile. Vaatetasandite kaugus on väga väike, sest puudub vajadus objekte kuvada erinevatel kaugustel, kuna see muudaks kasutajaliidese loomise liigselt keerukaks ning see ei paku erilist eelist.

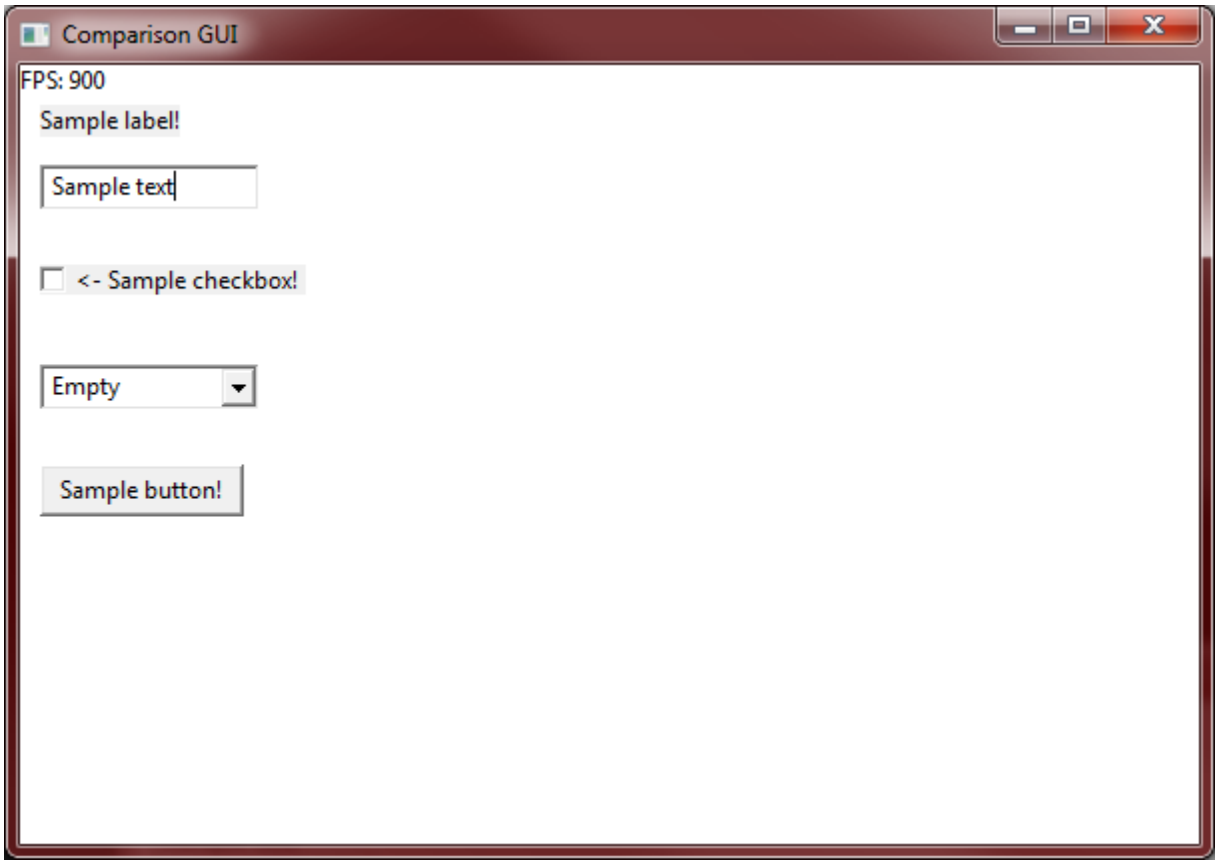
5. Tulemuse võrdlus

Projekti käigus tehtud teegi abil on loodud suhteliselt lihtne programm (Joonis 19), mis on loodud võimalikult sarnaselt wxWidgets teegi abil loodud teise programmiga (Joonis 20). Mõlema programmi lähtekood on saadaval aadressil: https://github.com/Zhardas/gui.

Teekide efektiivsuse leidmiseks on leitud nende abil tehtud programmide kaadrisagedus ehk FPS. FPS näitab kui palju kordi on programmi uuesti kuvatud ühe sekundi jooksul. Selle leidmine toimub elutsüklis, milles peale igat visualiseerimist suurendatakse lugerit, mille arv kuvatakse ja seejärel nullitakse iga sekundi järel. Mida efektiivsem on teek, seda suurem arv kordi suudetakse vormi uuesti kuvada. Mõõtmiseks piisab lihtsast vormist, peaasi, et mõlemad vormid oleksid võimalikult sarnased.



Joonis 19 - riistvaralise kiirendusega kasutajaliides



Joonis 20 - kasutajaliides (wxWidgets)

Mõlemal programmil on kuvatud tekstiobjekt, tekstilahter, märkekast koos tekstiga, liitkast ning tekstiga nupp. Samuti on nende kahe programmi võrdluseks kuvatud üles-vasakule nurka programmi kaadrisagedus.

Joonistelt on võimalik näha, et antud töö käigus loodud teegiga tehtud kasutajaliidese kaadrisagedus on 2327 kaadrit sekundis ning wxWidgets teegiga tehtud kasutajaliides ainult 900 kaadrit sekundis. See tähendab, et riistvaralisega kiirendusega kasutajaliidese ühe kaadri kuvamiseks kulub ligikaudu 0,00043 sekundit, kusjuures wxWidgetsi abil tehtu ühe kaadri jaoks kulub 0,001 sekundit. Programmid jooksutati masinal, millel oli AMD Phenom II X6 1100T Black Edition Thuban 3.3GHz kuuetuumaline protsessor ning Gigabyte GTX 780 WindForce OC 3 GB videokaart. Operatsioonisüsteemiks oli 64 bitine Windows 7 Ultimate. Testist võib järeldada, et antud masina peal on riistvaralise kiirendusega loodud kasutajaliides üle 2,3 korra kiirem.

6. Arendusvaade

Käesolev osa kirjeldab programmi võimalikke edasiarendusi.

6.1 Skriptimine

Hetkel võimaldab teek ainult püsiprogrammeerides määrata ära kuvatavad objektid, nende asukohad ja sündmused. Lisades teeki aga skriptimiskeele (nt. Lua, Javascript või Python) toetuse, oleks võimalik muuta kuvatavat infot ilma programmi uuesti kompileerimata. See muudaks arendusprotsessi kiiremaks ning lihtsustaks programmi uuendamist.

6.2 Disainiprogramm

Lisades eelnevalt mainitud skriptimiskeele toetuse, on võimalik kirjutada programm skriptifaili loomiseks ja muutmiseks. Programmi abil saaks valmisobjekte lohistada vaatele ning määrata ära nende parameetrid. See kiirendaks arendusprotsessi ning disaineril oleks pidev ülevaade, milline tulemus välja näeb.

6.3 Mitmeplatvormiline teek

Kuna C++ programmeerimiskeeles on võimalik kirjutada koodi, mis töötaks mitmel platvormil, on antud projektid platvorm kinni ainult kasutatud teekide tõttu. DirectX 9.0c teekide kogu vajab toimimiseks Windowsi platvormi. Et toetada rohkem platvorme, oleks vaja lisada juurde või asendada DirectX teegid mõne teise sarnase eesmärgiga teekidega. Riistvaralise kiirendusega annab kuvada ka kasutades OpenGL ja Vulkan teeke, mis toimivad nii Windowsi, Linuxi kui ka Mac platvormidel.

6.4 Vulkan

Vulkan on uue põlvkonna graafika ja arvutus rakendusliides, mis pakub kõrge efektiivsusega, mitmeplatvormilise ligipääsu kaasaegsetele graafikaprotsessoritele, mida kasutatakse mitmesugustel seadetel alates arvutitest kuni konsoolide ja mobiiltelefonideni välja. [11]

Vulkani rakendusliides võimaldab paremini graafikaprotsessori ressursse hallata (mäluhaldus ja puhvrid), kasutab protsessori kõiki tuumi tänu lõimtöötlusele. [12]

7. Kokkuvõte

Käesoleva töö põhieesmärgiks on luua tavalistest kasutajaliidese loomise teekidest kiirem alternatiiv, kasutades selleks riistvaralist kiirendust. Sellel teegil on enimkasutatud kasutajaliidese objektide klassid, mida saab kergelt kuvada.

Töö käigus sai loodud teek, mis kasutab DirectX 9.0c teekide kogu graafikaprotsessori utiliseerimiseks.

Järgnevalt on loetletud käesoleva lõputöö tulemused ehk loodud teegi funktsionaalsus:

- Graafikaprotsessori utiliseerimine;
- Enimkasutatud kasutajaliidese objektide kuvamine;
- Vertikaalne sünkroniseerimine.

Lisaks tulemustele, pakub töö ka võimalikud edasiarendused:

- Skriptimiskeele toetus arendusprotsessi kiirendamiseks;
- Disainimisprogrammi loomine, et kasutajaliidest saaks luua visuaalselt objekte pukseerides;
- Muuta kasutajaliidese teek mitmeplatvormiliseks, et seda saaks ka kasutada Linuxi ning Maci operatsioonisüsteemidega masinate peal;
- Vulkan rakendusliidese kasutamine, et saada veelgi parem ligipääs graafikaprotsessorile ning kiirem tulemus.

Töö tulemust võrreldi ka wxWidgets teegiga, luues mõlemaga võimalikult sarnane kasutajaliides ning võrreldes nende kaadrisagedust. Tulemusena oli loodud teek antud olukorras üle 2,3 korra kiirem konkurendist, mistõttu võib pidada kiirema kasutajaliidese teegi loomise eesmärki täidetuks.

Summary

The main objective of this thesis is to create a better alternative to common graphical user interface creation libraries by making use of hardware acceleration. This library has common user interface classes that can easily be displayed.

During the process a library was created that uses DirectX 9.0c libraries to utilize graphics processor.

The following is the resultant library functionality:

- Utilizing graphics processor;
- Displaying commonly used graphical user interface objects;
- Vertical synchronization.

In addition, potential further developments are also brought out:

- Scripting language support to accelerate the development process;
- Designer program that can be used to create interface scripts visually by drag and drop elements;
- Add cross-platform support so that the resulting program would also work on Linux and Mac platforms.
- Use Vulkan API instead of DirectX libraries to get a better and faster access to graphics processor.

The results of the thesis were also compared with the library wxWidgets by creating a similar user interface with both libraries and comparing their frame rate. The result was that the created library was more than 2.3 times faster than its competitor in that situation. Therefore, the objective of creating a faster interface can be considered to be fulfilled.

Kasutatud kirjandus

[1] „Kaadrisagedus - Vikipeedia, vaba entsüklopeedia,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Kaadrisagedus. [Kasutatud 23 mai 2016].

[2] „Overview - wxWidgets,“ [Võrgumaterjal]. Available: https://www.wxwidgets.org/about/. [Kasutatud 19 mai 2016].

[3] „Windows Forms Overview,“ Microsoft, [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/8bxxy49h(v=vs.110).aspx. [Kasutatud mai 19 2016].

[4] „Swing (Java) - Wikipedia, the free encyclopedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Swing_(Java). [Kasutatud 19 mai 2016].

[5] „Anonüümne funktsioon - Vikipeedia, vaba entsüklopeedia,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Anon%C3%BC%C3%BCmne_funktsioon. [Kasutatud 21 mai 2016].

[6] „Intel Z77 Express Chipset Review. Page 4 - X-bit labs,“ 4 august 2012. [Võrgumaterjal]. Available: http://www.xbitlabs.com/articles/mainboards/display/intel-z77_4.html. [Kasutatud 20 mai 2016].

[7] „Pipeline Overview >> Lighthouse3d.com,“ [Võrgumaterjal]. Available: http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/pipeline-overview/. [Kasutatud 20 mai 2016].

[8] „DirectXTutorial,“ [Võrgumaterjal]. Available: http://www.directxtutorial.com/Lesson.aspx?lessonid=9-4-3. [Kasutatud 26 aprill 2016].

[9] „Transforms (Direct3D 9) (Windows),“ [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/bb206269(v=vs.85).aspx. [Kasutatud 7 mai 2016].

[10] „Libgdx series: lens paper matrix transformation,“ 9 aprill 2014. [Võrgumaterjal]. Available: http://www.programering.com/a/MjNzEjMwATM.html. [Kasutatud 20 mai 2016].

[11] „Vulkan - Industry Forged,“ Khronos, [Võrgumaterjal]. Available: https://www.khronos.org/vulkan/. [Kasutatud 21 mai 2016].

[12] M. Schott ja L. M. Bishop, „Nvidia Gameworks - GDC2016,“ 2016. [Võrgumaterjal]. Available: http://developer.download.nvidia.com/gameworks/events/GDC2016/mschott_lbishop_gl_vulkan.pdf. [Kasutatud 21 mai 2016].

[13] „e-Teatmik: IT ja sidetehnika seletav sõnaraamat,“ [Võrgumaterjal]. Available: http://www.vallaste.ee/index.asp. [Kasutatud 20 mai 2016].

Lisa 1

Teegi lähtekood asub aadressil http://www.github.com/zhardas/engine/ ning selle põhjal tehtud programm, mida kasutati ka teekide võrdluseks asub aadressil https://github.com/zhardas/gui/. Samal aadressil leidub ka võrreldava kasutajaliidese koodi, avades selleks kausta „comparison_gui“. Teegi lähtekoodi aadressilt leiab ka teegi kasutusjuhendi.