Tallinna Ülikool

Informaatika Instituut

Microsoft Kinecti rakenduste loomise õppematerjal, kasutades Kinect for Windows SDK’d

Seminaritöö

Autor: Mait Mikkelsaar

Juhendaja: Jaagup Kippar

Autor: ..... „ ..... „ 2013

Juhendaja: ..... „..... „ 2013

Instituudi direktor: ..... „ ..... „ 2013

Tallinn 2013

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö 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)

Sisukord

Sisukord 3
Sissejuhatus 4
Võõrkeelsete lühendite loetelu 6
1 Kinect seade ja ajalugu 7
1.1 Kinect seade 7
1.2 Ajalugu 8
2 Õppematerjalid 9
3 Microsoft Kinect for Windowsi kasutavad rakendused 12
4 Õppematerjali ülevaade 13
4.1 Ettevalmistused 13
4.2 Rakenduste loomiseks kasutatud programmeerimiskeel 14
4.3 Õppematerjalis kasutatud koodinäidete valik 14
4.4 Õppematerjali analüüs 15
4.5 Õppematerjali kirjeldus 16
5 Õppematerjali testimine 17
Kokkuvõte 18
Kasutatud kirjandus 19
Lisad 20
Küsitlus 20
Küsitluse tagasiside 21
Õppematerjal 22

Sissejuhatus

Töö eesmärgiks on luua õppematerjal Kinectile programmeerimisega alustavale inimesele. Materjal peab kirjeldama seadme olemust ja õpetama näidete abil, kuidas alustada Kinecti kasutavate rakenduste programmeerimist. Õppematerjal peab olema piisavalt hästi lahti seletatud, et seda oleks võimalik iseseisvalt läbida.

Ma valisin selle teema, sest puudub eestikeelne õpetus Kinecti kasutatavate rakenduste loomise jaoks. Kättesaadavad inglisekeelsed õpetused on valdavalt ilma selgitavate tekstideta, mis muudab nende mõistmise keerulisemaks. Mahukamad õpetused on aga liiga keerulised ja algajal on nendes orienteerumine üle jõu käiv.

Suured tarkvara ja riistvara tootvad firmad (Microsoft, Asus) ennustavad, et tuleviku kasutajaliidesed on juhitavad liigutuste abil. Näite abil võib seda mõista järgnevalt: kasutades piltide vaatamise programmi, liigutab kasutaja kätt paremalt vasakule ja ekraanile kuvatakse järgnev pilt, liigutades vasakult paremale, tuleb eelnev pilt. Seminaritöö annab hüppeplatvormi iseseisvaks liigutustepõhiseks tarkvara arendamiseks.

Alustan ülevaatega Kinecti seadmest. Selgitan, mis on *Depth Stream*[1], *Color Stream*[2] ja *Skeleton Stream* [3]ja kuidas neida andmeid kätte saada. Andmete kätte saamist ja rakendamist demonstreerin näidetega. Seminaritöös kasutan Microsofti enda poolt välja antud ametlikku Kinect for Windows SDK 1.8, Kinect for Windows Developer Toolkit ja Microsoft Visual Studio 2012. SDK 1.8 on seminaritöö valmimise ajal kõige uuem versioon. Programmeerimiskeeleks valisin C#, kuna see on kõige populaarsem Kinecti rakenduste kirjutamise keel.

Töö lugejal on soovitatav omada eelnevat programmeerimiskogemust - eelkõige programmeerimiskeeles C#. Lugejatel, kes tahavad näited läbi teha, on vaja ligipääsu Kinect seadmele ja piisava võimekusega arvutit. Arvuti riistvaralised ja tarkvaralised nõuded on välja toodud õppematerjali peatükis number 4, Tarkvara paigaldamine.

[1] Eestikeelne vaste puudub, edaspidi kasutan mõistet sügavuspildi voog..
[2] Eestikeelne vaste puudub, edaspidi videopildi voog.
[3] Eestikeelne vaste puudub, edaspidi skeletiandmete voog.

Töö lõppu on lisatud õppematerjal pealkirjaga „Microsoft Kinecti rakenduste loomise õppematerjal, kasutades Kinect for Windows SDK'd“ (Õppematerjal), juhendi läbinutele antud küsitlus (Küsitlus) ja küsitluse tulemused (Küsitluse tagasiside).

Võõrkeelsete lühendite loetelu

Loetelu on koostatud tuginedes Vallaste sõnastikule (Vallaste, 2013).

SDK – *Software Development Kit*, programmipakett, mis võimaldab programmeerijal luua rakendusi konkreetsele platvormile.

IR – *Infrared*, infrapuna.

RGB – *Red, Green, Blue*, värvusruum videopildi kuvamiseks arvutiekraanil põhivärvuste (roheline, sinine ja punane) kombinatsioonidena.

RGBD - *Red, Green, Blue, Depth*, värvusruum videopildi kuvamiseks arvutiekraanil põhivärvuste (roheline, sinine ja punane) kombinatsioonidena, kus on lisaks juures ka sügavuse andmed.

API – *Application Programming Interface*, rakendusliides, programmiliides, API-liides. Arvuti operatsioonisüsteemiga või rakendusprogrammiga määratud reeglistik, mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teise rakendusprogrammi teenuseid.

YUV – Televisioonis PAL-süsteemi juures kasutatav video värvisignaali kodeerimismudel (värvusruum). Y on heledus, U ja V on värvsussignaalid. YUV-mudelit kasutatakse laialdaselt televisioonis, kus seda nimetatakse komponentvideoks. YUV signaalid luuakse algsetest RGB signaalidest, nii et Y on kõigi kolme komponendi R, G ja B summa, U on punane miinus heledus (R-Y) ja V on sinine miinus heledus (B-Y).

1 Kinect seade ja ajalugu

Peatüki esimeses osas selgitan lühidalt, mis on Kinect ja millest see koosneb. Teises osas kirjutan seadme ajaloost ja mis teeb Kinect'i eriliseks.

1.1 Kinect seade

Kinect for Windows on Microsofti toode, mis võimaldab liikumise tuvastamist. Kinect koosneb mitmest osast: infrapuna projektorist ja –kaamerast, RGB kaamerast, mootorist ja neljast mikrofonist (Joonis 1)(Kinect for Windows Sensor).

Joonis 1. Kinect for Windows seade

Borenstein (2012) kirjeldab IR projektorit ja –kaamerat järgnevalt: IR projektor saadab välja infrapuna täppide võrgustiku mida pildistab IR kaamera. Igas pildi osas, mille Kinecti IR kaamera salvestab, on iga täpp natukene paigast nihkunud võrreldes sellega, mida Kinect eeldab näha. Tulemuseks on, et Kinect suudab muuta selle IR pildil oleva täppidevõrgustiku sügavuse andmeteks, mis salvestab kõigi asjade kaugused, mida seade näeb.

Teiseks kaameraks on värvikaamera. Värvikaamera on väikese võimekusega ja sarnaneb paljudele veebikaameratele.

Värvikaamera toetab järgmisi värvivorminguid (Jana, 2012):

- RGB
  RGB pildivoo jaoks saab valida kahe resolutsiooni vahel: 640 x 480 pikselit, kiirusega 30 kaadrit sekundis ja 1280 x 960 pikseli, kiirusega 12 kaadrit sekundis.
- YUV

YUV pildivoog tuleb ainult ühe resolutsiooniga, milleks on 640 x 480 pikselit, kiirusega 15 kaadrit sekundis.

- *Bayer*[4]

  Bayer pildivoo resolutsioonid on samad, mis on RGB vool.

Kinectil on 4 mikrofoni, mis on jaotatud ümber seadme. Kasutades koos mitut mikrofoni, suudab Kinect lisaks salvestamisele kindlaks teha, kust kohast ruumis heli pärineb (Borenstein, 2012).

Seadmel on olemas mootor, mis võimaldab kaldenurka muuta: -28° kuni +28° (Catuhe, 2012).

Microsoft lisas selle mootori, et võimaldada Kinectil töötada erinevates ruumides. Sõltuvalt ruumi suurusest ja mööbli asukohast võivad inimesed, kes seda kasutavad, seista lähemal või kaugemal Kinectile ja nad võivad olla rohkem või vähem ruumis laiali jaotatud (Borenstein, 2012).

1.2 Ajalugu

Kinecti riistvara arendas PrimeSense – Iisraeli firma, mis ka varemalt valmistas sügavuskaameraid, mis omakorda kasutasid sama baas IR projitseerimise tehnikat. PrimeSense tegi Microsoftiga tihedat koostööd, et toota sügavuskaamera, mis töötaks tarkvara ja algoritmidega, mille Microsoft enda uurimistööga välja töötas (Borenstein, 2012).

Kõige tähtsam selle tarkvara juures on, et see töötles andmeid uuel moel, mis langetas drastiliselt sügavuse eristamise kulusid võrreldes sellel ajal valdavalt kasutatud meetodiga, mida kutsuti "time of flight". See tehnika jälgib aega, mis kulub valguskiirel sensorist väljudes tema tagasi peegeldumiseni. Uue meetodi puhul analüüsis 320x240 piksli suurust sügavuspilti PS1080 kiip, mis automaatselt joondas RGB kaamera informatsiooni infrapuna kaamera omaga – tulemuseks oli suurematele süsteemidele mõeldud RGBD andmed (Webb ja Ashley, 2012).

[4] *Bayer* (kasutatakse ka mõistet *Bayer filter*) – eestikeelne vaste puudub. Värvipildi formaat, mis kasutab punast, rohelist ja sinist värvi. Värvide osakaal värvimustris pole aga võrdne: 50% roheline, 25% punane ja 25% sinine.

2 Õppematerjalid

Selles peatükis annab autor ülevaate olemasolevatest õppematerjalidest. Õppematerjalide kohta toon välja pealkirja ja sisu positiivseid ja negatiivseid külgi.

Beginning Kinect Programming with the Microsoft Kinect SDK (Webb ja Ashley, 2012)

Raamat seletas hästi lahti mis on Kinect ja milline on selle ajalugu. Peatükk nimega Rakenduse alused seletab arusaadavalt lahti kuidas sensori olekute muutustega toime tulla (antud õppematerjal sensori olekuid ei kata).

See teos ei sobi algajale programmeerijale, kuna koodinäited ei ole lahti seletatud nii, kui oleks vaja kasutajale, kes esimest korda läbib antud materjali. Koodi on sellele eelnevas lõigus tutvustatud, kuid pisiasjad on jäänud selgitamata. Näiteks on koodi sees rida (Koodinäide 1), mis on edasijõudnule mõistetav, kuid isik, kes ei ole varem selliste meetoditega kokku puutunud, ei mõista koodirea tähendust. Autori koostatud õppematerjali sihtrühmaks on antud teemaga esmatutvujad ja seega on vaja kommenteerida iga rea otstarvet.

```
this._KinectDevice = KinectSensor.KinectSensors.FirstOrDefault(x => x.Status == KinectStatus.Connected);
```

Koodinäide 1. Koodinäide raamatust Beginning Kinect Programming with the Microsoft Kinect SDK

Kõige rohkem hindab autor selles raamatus peatükki, mis on pühendatud Kinecti arvutustele. Kuna Kinecti rakendused kasutavad kolmemõõtmelist ruumi, siis arvutused erinevad tavalistest C# ja .Net arvutustest. Antud peatükki ei ole algal programmeerijal lihtne mõista, aga kui põhjalikult süveneda Kinecti rakenduste loomisse, siis see ongi rakenduste tuum!

Õppematerjalina oli see hea, kuid miinuseks on lisaks eelnevalt nimetatule, et kasutatav SDK on aegunud. Uuemaid SDK'sid kasutades on mitmed protsessid lihtsamaks tehtud ja mõningad vanemad funktsionaalsused on ära kadunud.

Microsoft Press Start Here Learn the Kinect API (Miles, 2012)

Õppematerjalis kasutatakse vanemat SDK'd. Raamat on hästi peatükkidesse jaotatud ja vajalike lõikude leidmine on lihtne ja kiire. Materjal sisaldab palju informatiivseid pilte nii keskkonna seadistamise kui ka rakenduste näidete kohta.

Autor võttis enda kirjutatud õppematerjalis kasutusele antud raamatu stiili, kus kood on lahti seletatud sellele eelnevas lõigus ja lisaks on koodiread kommenteeritud koodinäite sees (Koodinäide 2).

```
kinectVideo.Source = BitmapSource.Create(
        colorFrame.Width, colorFrame.Height, // image dimensions
        96, 96, // resolution - 96 dpi for video frames
        PixelFormats.Bgr32, // video format
        null, // palette - none
        colorData, // video data
        colorFrame.Width * colorFrame.BytesPerPixel // stride
        );
```

Koodinäide 2. Näide kommenteeritud koodist (Miles, 2012)

Vaadates raamatu näidisrakendusi, võib jääda väärarusaam, et skeletiandmetega rakenduse juures on kõige olulisemad need luud, mis liigeste vahele jäävad. Tegelikkuses on tähtsad just need liigesed ise, sest rakendustes kasutatakse arvutamiseks liigeste koordinaate.

Making things see (Borenstein, 2012)

Autori jaoks oli see kõige huvitavam materjal, kuna see rääkis rohkem sellest, mida saab edasi teha peale põhilise andmete manipulatsiooni omandamist. Miinuseks on aga SimpleOpenNI andmeteekide kasutamine. Erinevuse näiteks võib esile tuua Kinectilt andmete saamise. Kui kasutada Kinect for Windows SDK-d, siis iga valmis kaadri olemasolul käivitub kindel funktsioon (Koodinäide 3). Funktsiooni sisendiks on kaadri muutujad, millest omakorda saab kätte kas siis värvipildi- , sügavuspildi- või skeletivoo andmed.

```
this.sensor.AllFramesReady += this.sensor_AllFramesReady;
```

Koodinäide 3. Sensori kaadrite püüdmise näide koostatud õppematerjalist

SimpleOpenNI-d kasutades on funktsiooni sees vaja uuendada Kinectilt saadud andmeid (Koodinäide 4). Edasi saab sama funktsiooni sees sensori objektilt sügavuspilti küsida.

```
this.sensor.update();
PImage depthImage = Kinect.depthImage();
```

Koodinäide 4. SimpleOpenNI näide (Borenstein, 2012)

Iseenesest ei ole see väga suur miinus, aga see ei lähe kokku antud seminaritöö eesmärgiga luua õppematerjal, kus kasutatakse Kinect for Windows SDK'd Kinecti rakenduste loomiseks

Raamatu 5. peatükki soovitab autor teemast huvitatutele kindlasti põhjalikumalt lugeda. Nimelt räägib see, kuidas teha Kinecti abil mitmemõõtmelist mudelit ja kuidas see mudel hiljem MakerBot 3D printerit kasutades välja printida.

Programming with the Kinect for Windows Software Development Kit (Catuhe, 2012)

Käsitletakse SDK versiooni 1.5, mis on kasutatav ka hilisemate versioonidega. Näited on kenasti lahti seletatud ja arusaadavad. Teostavate toimingute keerukus kasvab sujuvalt ja teemade vahel ei teki suuri hüppeid.

Probleemina näeb autor materjali juures aga seda, et koodinäited ei ole terviklikud. Skeletiandmete voo näitamise juures tuuakse liigese joonistamiseks välja funktsioon Plot, koodinäitele järgnevas peatükis öeldakse, et sellest funktsioonist tuleb juttu hiljem. Plot selgitatakse 4 lehekülge hiljem lahti, aga selle aja peale on juba meelest läinud, kus ja kuidas oli seda enne kasutatud. Selline meetod nõuab lugejal pidevalt tekstis edasi-tagasi liikumist. Kogenud programmeerijale ei ole selline koodinäidete asetus häiriv, aga algajal muudab see õppimise protsessi olulisemalt aeglasemaks.

Kinect for Windows SDK Programming Guide (Jana, 2012)

Kõige parem ja aluseks sobivaim raamat antud loetelust. Sisaldas väga põhjalikku riistvara tutvustust ja kasutusel oli SDK versioon 1.6, mis tähendab, et kood töötab kõigi uuemate SDK versioonidega. Ainsa suure miinusena võib välja tuua selle, et kood läks liiga kiiresti keeruliseks ja, seega algaja võib peatselt raskustesse sattuda.

Raamatu parimaks omaduseks on 11. peatükk nimega „Putting Things Together“, mis ühendab endas erinevaid Kinecti rakenduste edasiarendusi. Huvitavamatest osadest tooks välja Kinecti Windows Azure pilve ühendamise, Kinecti kasutamine koos Windos Phone-ga ja Kinecti ühendamine Netduinoga.

Kokkuvõtteks võib tuua, et paljude loetletud raamatute suureks miinuseks oli vana SDK kasutamine. Uued SDK’d tulevad välja iga poole aasta tagant, seega ongi peaaegu võimatu leida raamatut, kus oleks kasutusel viimased SDK’d. SDK 1.5 põhinevates raamatutes olev kood on kasutatav ka uuemate SDK versioonidega, kuna viimased on tahapoole ühilduvad. Samas saab vanematest raamatutest hea riistvara ja teooria ülevaate.

3 Microsoft Kinect for Windowsi kasutavad rakendused

Selles peatükis toob autor kaks näidet olemasolevatest rakendustest, mis kasutavad Kinect for Windows'i. Näited on võetud erinevatest valdkondadest, vältimaks muljet, nagu Kinectiga saabki luua ainult ühesuguseid rakendusi.

Eyes-Free Yoga

Eyes-Free Yoga on pimedatele mõeldud jooga programm, mis kasutab Microsoft Kinecti. Kõigepealt võtab jooga harrastaja Kinecti ees ette antud joogapoosi asendi sisse. Edasi vaatab programm, et isiku kere asend oleks õige ja vale asendi puhul annab kasutajale häälkäskluse abil teada, kuidas asendit muuta. Alustuseks korrigeerib rakendus pea ja kaela piirkonda ja viimasena vaatab käte ja jalgade asendi üle. Lisaks korrigeerimisele annab programm positiivset tagasisidet, kui asend on õige (Ma, 2013).

Ubi Interactive

Kasutades Ubi Interactive rakendust ja Kinect for Windows seadet, on võimalik muuta iga pind puutetundlikuks ekraaniks. Ubi võimaldab kuvada presentatsioone ja kavandeid pinnale nii, et pildiga saab ka suhelda. Selle tarkvara suurim müügiargument on tema hind. Kui kasutajal on varasemalt juba olemas projektor ja arvuti, siis Kinecti ja vastava tarkvara juurde ostmine tuleb oluliselt odavam selle projektori pildi mõõtmetega puutetundliku ekraani hinnast. Kõige paremaks antud tehnoloogia kasutamise näiteks on arhitektuuriprojektide kavandite esitlemine. Nimelt saab suurele publikule näidata 3D kavandit ja inimesed saavad kavandil ringi liikuda ning vaadata seda erinevatest külgedest. Olenevalt kavandist võib olla ka võimalus esemete nihutamiseks ja tõstmiseks.

4 Õppematerjali ülevaade

Peatüki esimeses osas antakse ülevaade käesoleva seminaritöö raames koostatud õppematerjali ettevalmistustes. Peatüki teises osas toon välja valitud programmeerimiskeele ja selle keele valiku põhjused. Kolmandas osas kirjeldan õppematerjalis kasutatud SDK eelistuse. Peatüki neljandas osa sisaldab õppematerjali analüüsi. Viiendas osas on õppematerjali kirjeldus.

4.1 Ettevalmistused

Enne õppematerjali koostamist tuli leida vastused mitmele küsimusele:

- Miks on vaja luua Kinecti rakenduste loomise õppematerjal?

  Esimene SDK versioon, Kinect for Windows SDK 1.0 oli väga populaarne ja sellele kirjutati palju õpetusi. Järgmine versioon, numbriga 1.5 muutus esimesest versioonist väga palju ja vanemad õpetused, mis on kirjutatud 1.0-le, ei tööta uuemate SDK versioonidega. Kõige rohkem muutus skeletiandmete voo kättesaamist, mis muudeti lihtsamaks. Leidsin, et on vaja head õpetust, mis näitaks, kuidas saada seadmelt andmeid kätte, kasutades uuemaid SDK-sid. Selline eestikeelne õpetus puudub.

- Kes on sihtgrupp ja millised on nende vajalikud teadmised?

  Sihtgrupiks on isikud, kes on huvitatud liigutustepõhiste programmide loomisest. Õppematerjali saab kasutada ka loengutes, kus käsitletakse C# programmeerimiskeelt või kus on teemaks interaktiivsed programmid.

- Mida peaks materjal katma?

  Õppematerjal peab sisaldama: rakenduse projekti loomist, kuna kõik ei ole Visual Studioga tuttavad; andmevoogude kättesaamist; andmete kasutamist lihtsamates programmides; erinevate andmevoogude kasutamist ühes rakenduses; lisaülesandeid, mis aitaksid kinnitada õpitud teadmisi ja oleksid piisavalt rasked, et vajavad sügavamat asjasse süvenemist.

4.2 Rakenduste loomiseks kasutatud programmeerimiskeel

Kasutades Kinect for Windows SDK-d saab Kinecti programme luua mitmes erinevas programmeerimiskeeles: C#, C++, Visual Basic ja HTML5 koos JavaScriptiga. Kui vaadata olemasolevaid õppematerjale, mis on veebis kui ka raamatu kujul, siis enim levinud keel Kinecti rakenduste jaoks on C#, mida autor kasutab ka enda töös. Kõige uuema Kinect for Windows versiooniga, versioon 1.8, tuli võimalus kasutada Kinecti HTML5 ja Javascriptiga – kolmanda osapoole teekidega sai seda juba varem teha. Rakendades mitteametlikke, kolmandate osapoolte, arendustarkvara, on võimalik ka Flashi ActionScripti ja Javat kasutada, aga neid tarkvarasid antud seminaritöö ei käsitle.

4.3 Õppematerjalis kasutatud koodinäidete valik

Autor lähtus koodi kirjutamisel selle lihtsusest. Lihtsuse all on mõeldud, et kood peab olema võimalikult lühidalt kirja pandud, valitud peab olema kõige lühem tee, mis viib tulemuseni ja teemat mõistvale programmeerijale peab kood olema mõistetav ka kommentaarideta.

Näitena võib välja tuua Kinect sensori eluea haldamise. Õppematerjalis ei ole kirjeldatud seadme eluea haldamist KinectSensorChooser klassiga vaid on võetud esimene sensor ja alustatakse selle kasutamist (Koodinäide 5). Kasutades aga KinectSensorChooser klassi on võimalik käsitleda olukordi, mis tekivad, kui sensor puudub, on vooluvõrgust välja tõmmatud või kui sensor tõmmatakse välja programmi töötamise ajal (Kinect for Windows SDK).

```
KinectSensor sensor = KinectSensor.KinectSensors[0];
```

Koodinäide 5. Sensori määramine

Programm võib kasutada ka mitut Kinect seadet, kas siis samaaegselt või erinevatel aegadel. Mitme seadme haldus toimub kasutades viimati nimetatud klassi.

Koodinäidetes on kasutatud põhialuseid, kuidas saada andmed kätte ja neid töödelda, sellest tulenevalt ei ole kasutatud erinevaid liideseid. Näiteks on olemas KinectBackgroundRemoval API, kuid juhendi ülesandes eeldatakse, et kasutaja saab sama efekti kasutades sügavusvoogu. KinectBackgroundRemoval võimaldab tausta eemaldamist – õppematerjali peatükis eemaldatakse nende punktide andmed, mis on kaugemal kui määratud punkt. Tulemus on sama, aga autor leiab, et rakenduste edaspidiseks arenduseks on olulisemad baasteadmised ja seega keskendub neile.

4.4 Õppematerjali analüüs

Õppematerjalis on kasutusel Microsofti enda poolt välja antud ametlik tarkvara, millega töötamiseks on Kinect for Windows mõeldud. Arenduskeskkonnaks on Microsoft Visual Studio, kasutada saab järgnevaid keskkonna versioone: 2011, 2012 ja 2013. Töös on kasutusel arenduskomplekti versioon 1.8. Koodid, mis on kirjutatud versioonile numbriga 1.5 või uuemale, töötavad ka kasutades SDK 1.8.

Materjali läbimiseks on soovitatavad algteadmised programmeerimises. Töös kasutatav programmeerimiskeel on küll C#, aga õppematerjalis ei ole kasutusel midagi, mis oleks väga iseloomulik just C# keelele, seega piisab ka eelnevast kokkupuutest teiste keeltega. Arenduskeskkonna eelnev tundmine ei ole vajalik, kuna põhilised toimingud, mis on keskkonnale spetsiifilised, on välja toodud koos illustreerivate piltidega.

Juhendi koostamisel jälgis autor, et kõik tegevused oleksid võimalikult lihtsad ja hästi lahti seletatud. Õppematerjali läbimisel lähevad tegevused ja ülesanded järjest raskemaks seega ei ole mõistlik isikul, kes antud tehnoloogiaga esmakordselt kokku puutub, jätta peatükke vahele. Autor eeldas, et kõik materjali läbijad ei pruugi üheselt mõista kuidas on koodi osad kokku pandud ja seega lisas lisade alla iga rakenduse koodi. Tervikliku koodi olemasolu töö lõpus tagab selle, et juhendi läbimine ei jääks pooleli kui materjali läbija teeb enda rakenduse koodis vea. Lisades olev kood annab ka võimaluse tagasi pöörduda töötava koodi juurde olukorras, kus õppematerjali läbinu arendadab näidetest edasi enda rakendust ja tekkinud vea tõttu soovib alustada projektiga uuesti.

Töös olevad peatükid on püütud järjestada loogiliselt nii, et iga järgnev rakendus on edasiarendus või täiendus eelmises peatükis loodule. Õppematerjali eeldatav läbimise aeg on 3 akadeemilist tundi. Kui lahendada ka peatükkide lõpus olevad ülesanded, siis eeldatavalt lisandub töö läbimise ajale 1 akadeemiline tund.

Materjali läbimise tulemuseks on Kinecti kasutavad rakendused, mis saavad seadmelt video-, sügavus- või skeletiandmete voo. Samuti omandab kasutaja algteadmisi nende voogude kinnipüüdmise ja kasutamise kohta. Juhendi läbimine ei anna põhjalikke teadmisi seadme kasutamise võimalustest, vaid võimaluse saada lihtne ja arusaadav algus Kinecti võimalustega tutvumiseks.

4.5 Õppematerjali kirjeldus

Õppematerjal annab baasteadmised, mis on vajalikud, et alustada Kinect for Windows seadmele rakenduste loomist. Kirjeldatud on videopildi, sügavuse voo ja skeletiandmete voo saamine ja rakendamine programmis.

Esimene peatükk annab ülevaate seadme osadest, selle tööpõhimõtetest ja ajaloost.

Teine peatükk kirjeldab arvuti süsteemi nõudeid, mis on vajalikud tarkvara paigaldamiseks. Järgmiseks on õpetus, kuidas paigaldada arvutisse vajalikud programmid.

Kolmandas peatükis on kirjeldatud rakenduste loomist. Peatükk koosneb kaheksast alapeatükist, kus iga alapeatükk kirjeldab mingi programmi koostamist:

1. Esimene osa on projekti seadistamine, mis on iga järgneva programmi baasiks.
2. Teine osa on kõige lihtsama Kinecti kasutava rakenduse koostamine – konsoolirakenduse koostamine.
3. Kolmas osa on videopildi näitamine.
4. Neljas osa on sügavuse voo rakendamine, kasutades erinevate sügavuste näitamiseks halli erinevaid toone.
5. Viies osa on skeletiandmete voo kättesaamine. Selle voo kasutamine rakenduses on kõige raskem.
6. Kuues osa kirjeldab kuidas ühendada videopilt ja sügavusvoo andmed ning kuidas kasutada neid koos kasutada rakenduses.
7. Seitsmendas osas ühendatakse skeletiandmete voog videopildiga. Selle peatüki lõpus on kõige raskemad ülesanded, mis nõuavad põhjalikumat süvenemist.

Lisades on välja toodud iga rakenduse juurde kuuluv terviklik kood, mis abistab, kui mingi punkti juures peaks hätta jääma. Selle järgi on ka hea enda koodi kontrollida.

5 Õppematerjali testimine

Õppematerjali testijateks valisin välja Tallinna Ülikooli informaatika eriala 3 üliõpilast. Testimisel osalevad tudengid valisin välja mugavusvalimit kasutades. Mugavusvalimi kasutamine oli tingitud kahest asjaolust: esiteks oli vaja õppematerjalis kirjeldatud kriteeriumitele vastavat arvutit ja teiseks pidi omama 3 akadeemilist tundi materjali läbitöötamiseks. Testijate arvu piiras ka Kinect for Windows seadme piiratud kättesaadavus.

Testimise protsess alguses andsin materjali läbijale kätte Kinect for Windows seadme ja arvuti koos õppematerjaliga. Testija läbis õppematerjali iseseisvalt. Tagasiside saamiseks kasutasin kirjalikku küsimustikku (Joonis 2) ja vabas vormis intervjuud. Küsitluse tulemused on lisades (Tabel 1 ja Tabel 2).

Küsitluse vastajatest puudus kõigil varasem kokkupuude Kinectile rakenduste loomisega, ühel isikul puudus varasem kokkupuude töös kasutatud C# programmeerimiskeelega. Testgrupp arvas üksmeelselt, et materjal on iseseisvalt läbitav, mis oli ka õppematerjali loomise üks kriteeriumeid. Töös kasutatud koodi peeti arusaadavalt lahtiseletatuks ja puudusid segaseks jäävad mõisted. Üks vastaja kirjutas küsitluses, et materjalis ei olnud piisavalt ekraanipilte. Hilisemas intervjuus aga ei osanud ta välja tuua ühtegi kohta, kuhu oleks võinud ekraanipildi juurde lisada ja seega muutis enda arvamust.

Töö parendamiseks soovitati lisada rakenduse käivitamise ekraanipilt ja teha suuremaks mõningad ekraanipildid, kuna need olid raskesti loetavad. Kümne palli skaalal, kus 0 on, et üldse ei olnud arusaadav ja 10, et oli täiesti arusaadav, hinnati õppematerjal arusaadavust keskmiselt 8,7 palliga, mis on väga hea tulemus. Vastajad kiitsid lisades olevaid terviklikke koode, mis enamikes nende läbitud õppematerjalides on puudu olnud.

Kokkuvõttes jäid testijad õppematerjaliga rahule ja vastasid, et nad soovitaksid seda ka teistele. Peale testgrupilt vastuste saamist, viisin õppematerjali soovitatud muudatused sisse.

Kokkuvõte

Seminatitöö eesmärgiks oli koostada eestikeelne Microsoft Kinect for Windows rakenduste loomise õppematerjal, mis oleks algajale kergesti arusaadav. Autor uuris olemasolevaid raamatuid, veebimaterjale ja Kinect for Windows Toolkitiga kaasa tulevaid koodinäiteid ja koostas nende põhjal õppematerjali.

Õppematerjali testisid kolm Tallinna Ülikooli informaatika eriala tudengit. Testimise eesmärgiks oli välja selgitada, kas õppematerjal on mõistetav, iseseisvalt läbitav, ja välja tuua materjali parendamise võimalused. Testimise tulemused näitasid, et õppematerjal on arusaadav ja sisaldab piisavalt selgitusi. Testijad käisid välja ka mõned ideed, kuidas juhendit paremaks muuta ja vastavad muudatused said ka autori poolt materjali sisse viidud. Testimise tulemustest saab järeldada, et seminaritöö eesmärk sai täidetud.

Teema edasiarendamiseks saab võtta juurde helivoo ja kirjeldada igat voogu ja selle kasutamist täpsemalt. Lisada võib ka Kinecti rakenduste programmeerimise parimad praktikad ja välja tuua rakenduste kasutajaliidese loomise juhised. Teine edasiarendamise suund oleks võtta juurde kolmandate osapoolte arendustarkvarad, üks näide oleks OpneNI, ja võrrelda nende kasutamisvõimalusi.

Kasutatud kirjandus

Vallaste. Külastatud 14. detsembril, 2013, aadressil http://www.vallaste.ee

Ma, M. (2013). *Yoga accessible for the blind with new Microsoft Kinect-based program*. Külastatud 19. detsembril , 2013, aadressil http://www.washington.edu/news/2013/10/17/yoga-accessible-for-the-blind-with-new-microsoft-kinect-based-program/.

Kinect for Windows Team. (2013). *Turn any surface into a touch screen with Ubi Interactive and Kinect for Windows.* Külastatud 19. detsembril, 2013, aadressil http://blogs.msdn.com/b/kinectforwindows/archive/2013/08/13/turn-any-surface-into-a-touch-screen-with-ubi-interactive-and-kinect-for-windows.aspx

Kinect for Windows Sensor. Külastatud 14. detsembril, 2013, aadressil http://msdn.microsoft.com/en-us/library/hh855355.aspx.

Webb, J., Ashley J. (2012). *Beginning Kinect Programming with the Microsoft Kinect SDK*. New York: Apress.

Borenstein, G. (2012). *Making Things See*. Sebastopol: Maker Media

Catuhe, D. (2012). *Programming with the Kinect for Windows Software Development Kit*. Washington: Microsoft Press.

Miles, R. (2012). *Microsoft Press Start Here Learn the Kinect API*. Washington: Microsoft Press.

Jana, A. (2012). *Kinect for Windows SDK Programming Guide*. Birmingham: Packt Publishing.

Kinect for Windows SDK. Külastatud 14. detsembril, 2013, aadressil http://msdn.microsoft.com/en-us/library/jj663803.aspx.

Lisad

Küsitlus

Microsoft Kinecti rakenduste loomise õppematerjali testijatelt tagasiside saamiseks loodud küsitlus. Küsitlus asub veebiaadressil: https://docs.google.com/forms/d/1P16Kj1FQj8HePAHc8ekZDD53a6VoJDsV_Qn3l61u45w/viewform

Kinect'i õppematerjali tagasiside

* Kohustuslik

Sinu nimi? *

Mitmendal kursusel olete? *

Kas on läbitud kursus .NET raamistik või on muu varasem kokkupuude C#-ga? *
- Jah
- Ei

Kas on varasem kokkupuude Kinect'iga? *
- Jah
- Ei

Kui on varasem kokkupuude Kinect'iga, siis milline?

Kas õppematerjal oli arusaadav? *

0 1 2 3 4 5 6 7 8 9 10

Üldse mitte arusaadav Täiesti arusaadav

Kas kood oli arusaadavalt lahti seletatud? *
(kui ei, siis mis ei olnud)

Kas mõni mõiste jäi segaseks? *
(kui jah, siis milline)

Kas ekraanipilte oli piisavalt? *
- Jah
- Ei

Kas õppematerjal on iseseisvalt läbitav või on vaja juhendajat kõrvale? *
- Iseseisvalt läbitav
- Juhendaja on vajalik

Kas soovitaksid seda õppematerjali teistele? *
- Jah
- Ei

Kas soovid midagi lisada või täpsustada õppematerjali kohta?
(siia kirjuta vabas vormis)

Saada ära

Ärge saatke paroole kunagi Google'i vormide kaudu.

100%: ongi valmis.

Joonis 2. Tagasiside küsitlus

Küsitluse tagasiside

	Ajatempel	Mitmendal kursusel olete?	Kas on läbitud kursus .NET raamistik või on muu varasem kokkupuude C#-ga?	Kas on varasem kokkupuude Kinect'iga?	Kas õppematerjal oli arusaadav?	Kas kood oli arusaadavalt lahti seletatud?
Vastaja 1	12/12/2013 19:47:16	3	Ei	Ei	9	jah
Vastaja 2	12/13/2013 11:37:34	3	Jah	Ei	9	jah
Vastaja 3	12/13/2013 13:59:17	2	Jah	Jah	8	jah

Tabel 1. Tagasiside esimene osa

	Kas mõni mõiste jäi segaseks?	Kas ekraanipilte oli piisavalt?	Kas õppematerjal on iseseisvalt läbitav või on vaja juhendajat kõrvale?	Kas soovitaksid seda õppematerjali teistele?	Kas soovid midagi lisada või täpsustada õppematerjali kohta?	Kui on varasem kokkupuude Kinect'iga, siis milline?
Vastaja 1	ei jäänud	Ei	Iseseisvalt läbitav	Jah	Võiks rohkem pilte ja ikoone olla	EI ole isegi kordagi mänginud
Vastaja 2	ei	Jah	Iseseisvalt läbitav	Jah		
Vastaja 3	ei	Jah	Iseseisvalt läbitav	Jah	Mõned pildid olid võib-olla liiga väikesed ning ei saanud aru, mida seal kirjutatud oli, seletused olid arusaadavad ja lihtsad lugeda. Hea oli veel see, et kui mingis osas hätta jäin, sai lõpus olevast koodist abi.	Programmeerimiskogemust ei ole, ainult mänginud mänge xbox + kinect

Tabel 2. Tagasiside teine osa

Õppematerjal

Tallinna Ülikool

Informaatika Instituut

Microsoft Kinecti rakenduste loomise õppematerjal, kasutades Kinect for Windows SDK'd

Autor: Mait Mikkelsaar

Tallinn 2013

Sisukord

Sisukord 2
1 Võõrkeelsete sõnade ja lühendite loetelu 4
2 Sissejuhatus 5
3 Kinect seade ja ajalugu 6
3.1 Kinect seadme tutvustus 6
3.1.1 IR kaamera ja – projektor 6
3.1.2 Värvikaamera 6
3.1.3 Mikrofonid 7
3.1.4 Mootor 7
3.2 Ajalugu 7
4 Tarkvara paigaldamine 8
4.1 Süsteeminõuded: 8
4.2 SDK v1.8 paigaldamise õpetus: 9
4.3 Microsoft Visual Studio paigaldamine 9
5 Rakenduste programmeerimine 10
5.1 Projekti seadistamine 10
5.2 Konsooli rakendus 13
5.2.1 Ülesandeid 15
5.3 Videopildi rakendus 16
5.4 Sügavuspildi rakendus 19
5.4.1 Ülesandeid 22
5.5 Skeleti rakendus 23
5.5.1 Ülesandeid 29
5.6 Videopildi ja sügavusandmete voo ühendamine 30
5.6.1 Ülesandeid 33

5.7 Videopildi ja skeletiandmete voo ühendamine 35
5.7.1 Ülesandeid 36
5.8 Kinecti nurga liigutamine 37
5.8.1 Ülesandeid 37
6 Kokkuvõte 38
7 Kasutatud kirjandus 39
8 Lisad 40
8.1 Konsoolirakenduse kood 40
8.2 Videopildirakenduse kood 41
8.3 Sügavuspildirakenduse kood 43
8.4 Skeletirakenduse kood 46
8.5 Joonistamise rakenduse kood 51
8.6 Videopildi ja skeletiandmete voo ühendamise kood 55

1 Võõrkeelsete sõnade ja lühendite loetelu

Loetelu on koostatud tuginedes Vallaste sõnastikule (Vallaste, 2013).

SDK – *Software Development Kit*, programmipakett, mis võimaldab programmeerijal luua rakendusi konkreetsele platvormile.

IR – *Infrared*, infrapuna.

RGB – *Red, Green, Blue*, värvusruum videopildi kuvamiseks arvutiekraanil põhivärvuste (roheline, sinine ja punane) kombinatsioonidena.

RGBD – *Red, Green, Blue, Depth*, värvusruum videopildi kuvamiseks arvutiekraanil põhivärvuste (roheline, sinine ja punane) kombinatsioonidena, kus on lisaks juures ka sügavuse andmed.

Draiver – välisseadet opsüsteemiga liidestav juhtimisprogramm.

Environment variable – keskkonnamuutuja. Opsüsteemi, veebiserveri või muu juhtprogrammi poolt värskendatav andmeüksus.

Installer – installeerija. Rakendus, mis kopeerib tarkvara kasutaja arvutisse, seab selle üles ja kohandab konkreetse kasutaja vajadustele.

Bitmap image – rasterkujutis, rasterpilt.

Event – sündmus. Programmi poolt avastatav tegevus või toiming.

Polling – pollimine, saatele kutsumine. Elektroonilises sides nimetatakse pollimiseks teiste programmide või seadmete pidevat kontrollimist ühe programmi poolt, et näha, mis olekus need on.

Dpi - täppi tolli kohta.

2 Sissejuhatus

Käesolev õppematerjal on mõeldud inimestele kellel on huvi liigutusi tuvastavate rakenduste ja Kinect seadme vastu. Materjal on mõeldud iseseisvaks läbitöötamiseks ja eeldatav aeg, ilma lisaülesanneteta, on 3 akadeemilist tundi.

Õppematerjalist aru saamine eeldab programmeerimise algteadmisi, soovitatavalt keeles C#, kuna õppematerjal on kirjutatud C#is. Valisin keeleks C# kuna see on kõige populaarsem keel, milles Kinectile rakendusi koostatakse. Programmeerimise keskkonnaks on Microsoft Visual Studio. Õppematerjal on koostatud Visual Studio 2012 Professionali peal, aga arenduseks sobivad uuemad ja vanemad Visual Studio versioonid. Näidisrakenduste jooksutamiseks on vajalik Kinect for Windows seade või Xbox Kinect koos ühendatava kaabliga, millel on USB ots arvutiga ühendamiseks ja toide seadmele.

Teemat ajendas käsitlema isiklik huvi Kinecti vastu ja töötavate näidete puudus. Nimelt on enamik õpetusi kirjutatud vanematele Kinect for Windows SDK versioonidele ja need ei tööta, kui on SDK versioon 1.5 või uuem. Koodinäidete aluseks võtsin Kinect for Windows Development Toolkiti näidete koodid.

Õppematerjal näitab kuidas saada kätte erinevad vood ja kuidas neid rakendada. Lühidalt on ka kirjeldatud seadme mootori võimalusi. Materjal ei käsitle seadme audiovoo kätte saamist ja töötlemist.

Lisade alla on pandud käsitletud programmide põhi failide terviklikud koodid. Juhul kui materjalisiseselt jääb koodi asukoht või rakendamine segaseks, saab selle üle vaadata lisade alt.

3 Kinect seade ja ajalugu

Peatüki esimeses osas selgitan, mis on Kinect ja millest see koosneb. Teises osas kirjutan seadme ajaloost ja mis teeb Kinecti eriliseks.

3.1 Kinect seadme tutvustus

Kinect on Microsofti toode, mis võimaldab tuvastada liikumist. Kinect koosneb infrapuna projektorist ja –kaamerast, värvikaamerast ja neljast mikrofonist.

3.1.1 IR kaamera ja – projektor

IR projektor saadab välja infrapuna täppide võrgustiku mida pildistab IR kaamera. Tehases, kus Kinect tehti, kalibreeriti iga Kinect nii, et see teab, kus iga täpp, mida projitseeritakse, täpselt asub, kui seda projitseeritakse vastu ühetasast seina kindla kauguse tagant. Iga objekt, mis on lähemal kui Kinecti kalibreeritud kaugus, lükkab need täpid paigast kindlasse suunda ja iga objekti puhul, mis on kaugemal kui see vahemaa, lükatakse need täpid teises suunas. Kuna Kinect on kalibreeritud teadma kõikide täppide originaalkaugust, saab Kinect kasutada nende nihet, et leida distants objektini antud vaates. Igas pildi osas, mille Kinecti IR kaamera salvestab, on iga täpp natukene paigast nihkunud võrreldes sellega, mida Kinect eeldab näha. Tulemuseks on, et Kinect suudab muuta selle IR pildil oleva täppidevõrgustiku sügavuse andmeteks, mis salvestab kõigi asjade kaugused, mida seade näeb (Borenstein, 2012).

3.1.2 Värvikaamera

Teiseks kaameraks Kinecti küljes on värvikaamera. See sarnaneb paljudele veebikaameratele ja väiksematele digikaameratele. Sellel on suhteliselt madal resolutsioon – 640 x 480 pikselit. Selle kaameraga saab salvestada ka kõrgema resolutsioonilist pilti (1280x1024 pikselit), aga see põhjustab järeleandmise kaadrisageduses, nimelt 10 kaadrit sekundis tavapärase 30 kaadrit sekundis asemel, mis on standardiks 640x480 resolutsiooni puhul (Borenstein, 2012).

Iseenesest ei ole see värvikaamera huvitav, aga kuna seade on lisatud Kinecti külge kindlale kohale IR kaamera suhtes, siis suudab Kinect joondada värvikaamera pildi ja sügavuspildi, mis on tehtud IR kaameraga. See võimaldab lisada ja eemaldada kindlal kaugusel asuvad objektid koos realistlike värvidega teda ümbritsevast keskkonnast (Borenstein, 2012).

3.1.3 Mikrofonid

Ümber Kinecti on jaotatud neli mikrofoni. Ühest mikrofonist oleks piisanud, et lindistada lihtsalt heli. Kasutades koos mitut mikrofoni, suudab Kinect lisaks salvestamisele kindlaks teha, kust kohast ruumis see pärineb. Näiteks kui mitu mängijat ütlevad käsklusi, et mängu juhtida, siis Kinect saab kindlaks teha milliselt mängijalt milline käsk tuli (Borenstein, 2012).

Käesolevas õppematerjalis ei käsitleta mikrofonide võimalusi.

3.1.4 Mootor

Kinectil on lisaks võime ka liikuda. Tema plastikaluses on väike mootor ja mitmed käigud. Seda mootorit keerates saab Kinect kallutada enda andureid üles ja alla. Mootori liikumisulatus on piiratud 30 kraadiga. Microsoft lisas selle mootori, et võimaldada Kinectil töötada erinevates ruumides. Sõltuvalt ruumi suurusest ja mööbli asukohast võivad inimesed, kes seda kasutavad, seista lähemal või kaugemal Kinectile ja nad võivad olla rohkem või vähem ruumis laiali jaotatud. Mootor võimaldab Kinectil suunata ennast parimale kohale, et jäädvustada inimesi, kes seda kasutavad (Borenstein, 2012).

3.2 Ajalugu

Kinecti riistvara arendas PrimeSense – Iisraeli firma, mis ka varemalt valmistas sügavuskaameraid, mis omakorda kasutasid sama, baas IR projetseerimise tehnikat. PrimeSense tegi Microsoftiga tihedat koostööd, et toota sügavuskaamera, mis töötaks tarkvara ja algoritmidega, mille Microsoft enda uurimistööga välja töötas (Borenstein, 2012).

Kõige tähtsam antud tarkvara juures on, et see töötles andmeid uuel moel, mis langetas drastiliselt sügavuse eristamise kulusid võrreldes sellel ajal valdavalt kasutatud meetodiga, mida kutsuti "*time of flight*". See tehnika jälgib aega, mis kulub valguskiirel sensorist väljudes tema tagasi peegeldumiseni. Uue meetodi puhul analüüsis 320x240 piksli suurust sügavuspilti PS1080 kiip, mis automaatselt joondas RGB kaamera informatsiooni infrapuna kaamera omaga – tulemuseks oli suurematele süsteemidele mõeldud RGBD andmed (Webb ja Ashley, 2012).

4 Tarkvara paigaldamine

Selleks, et arvuti saaks suhelda Kinecti seadmega, on vaja paigaldada vastav tarkvara - Kinect for Windows SDK. Kinect for Windows SDK koosneb erinevatest infoobjektide kogudest - teekidest, mis võimaldavad meil programmeerida Kinecti sensorit kasutavaid rakendusi erinevatele Microsofti arendusplatvormidele. SDK abil saame me programmeerida WPF, WinForms, XNA rakendusi ja natukene vaeva nähes on võimalik teha ka brauseripõhiseid rakendusi, mis jooksevad Windowsi operatsioonisüsteemil. Kummaline on aga, et Kinect for Windows SDK'ga ei ole võimalik luua Xbox'i mänge, samas aga saavad arendajad kasutada SDK'd Xbox'i Kinectiga. Kinecti lähirežiimi kasutamiseks, on meil aga vaja ametlikku Kinect for Windows riistvara (Webb ja Ashley, 2012).

SDK sisaldab *draiver*eid et kasutada Kinecti arvutitega, millel jookseb kas Windows 7, Windows 8 või Windows Embedded Standard 7 operatsioonisüsteem.

Süsteemi nõuded ja SDK paigaldamise õpetused on olemas veebiaadressil http://www.microsoft.com/en-us/download/details.aspx?id=40278, aga ma kirjutan nad õppematerjali välja juhuks, kui leht ei ole kättesaadav. Paigaldamise juhend ei ole suuresti muutunud viimase kolme versiooni vältel, seega on suur tõenäosus, et see töötab ka järgmise versiooni korral.

4.1 Süsteeminõuded:

Riistvara:

- 32 biti (x86) või 64 bitine protsessor
- Dual-core 2.66-GHz või kiirem protsessor
- Püsiv USB 2.0 siin (ainult Kinect seadmele kasutamiseks)
- 2 GB RAM (soovitatavalt 4 GB)
- Kinect for Windows

Tarkvara:

- Microsoft Visual Studio 2010, Visual Studio 2012 või Visual Studio 2013 (sobib ka Express versioon)
- .NET Framework 4.0 või 4.5

- Kõne toega Kinect for Windows r0akenduste arendamiseks on vajalik paigaldada Microsoft Speech Platform SDKv11

4.2 SDK v1.8 paigaldamise õpetus:

1. Veendu, et Kinecti sensor ei oleks ühendatud ühessegi arvuti USB porti.
2. Kui arvutile on paigaldatud vanem Kinect for Windows versioon (v1.0 või v1.5-1.7), tuleb sulgeda kõik avatud näited ja jätkata sammuga number 5. Kinect for Windows v1.8 uuendab varasemat versiooni.
3. Eemaldada kõik muud *draiver*id mis on paigaldatud Kinecti sensori jaoks.
4. Kui arvutile on paigaldatud Microsoft Server Speech Platform 10.2, eemaldage Microsoft Server Speech Platform Runtime ja SDK komponendid - nii x86 kui ka x64 bitise versiooni, ja Microsoft Server Speech Recognition Language - Kinect Language Pack.
5. Sulgege Visual Studio (juhul kui käib). See tuleb sulgeda enne SDK paigaldamist ja peale seda uuesti käivitada, et saaks kasutada *environment variable*'sid, mida SDK vajab.
6. Allalaadimise kohas (http://www.microsoft.com/en-us/download/details.aspx?id=40278), teha klikk DOWNLOAD nupul.
7. (Käivitada *installer* ja järgida ekraanile kuvatavaid juhuseid.)
8. Kui SDK on paigaldamise edukalt lõpetanud, ühendada Kinect sensor välisesse vooluallikasse ja siis ühendada sensor arvuti USB porti. Draiverid laetakse automaatselt.
9. Kinecti sensor peaks nüüd korrektselt töötama.
10. Kui soovite peale õppematerjali läbimist Kinectile edasi arendada, tõmmake endale ka Kinect for Windows Developer Toolkit, milles sisalduvad erinevad lähtekoodi näited, tööriistad ja teised väärtuslikud arendusvahendid, mis teevad Kinecti rakenduste arendamise lihtsamaks.

4.3 Microsoft Visual Studio paigaldamine

Visual Studio on alla laetav aadressilt http://www.visualstudio.com/en-us/downloads/. Näidisrakenduste programmeerimise jaoks sobivad nii Professional kui ka Express versioonid.

Kui *installer* on alla laetud, käivitage see ja järgige ekraanile kuvatavaid juhiseid.

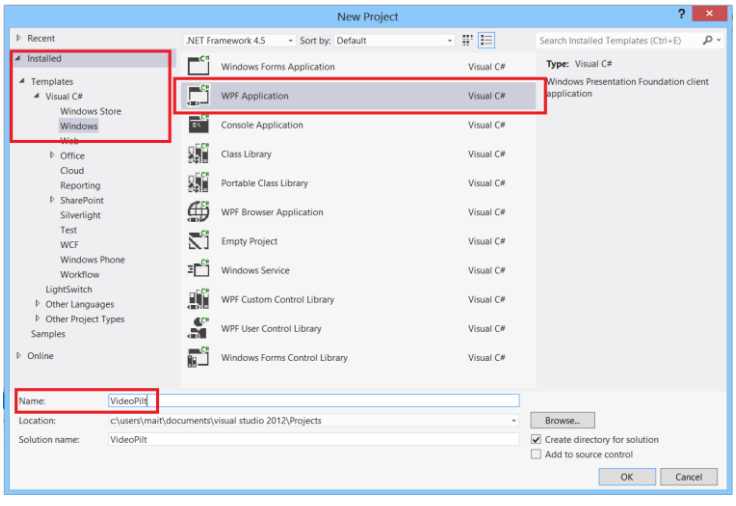
5 Rakenduste programmeerimine

Kinectilt saame kätte kolme tüüpi voogedastusi, mida hakkame rakendustes kasutama – *Color Image Stream*[5], *Depth Stream*[6] ja *Skeleton Stream*[7]. *Color Image Stream* on tavaline videovoog, nagu videokaamerast tulev pilt. Sügavuspildi voog annab andmeid selle kohta, kui kaugel on objektid kaamerast. Sügavuspildi voogu kuvades on näha halli pilti, kus hall toon näitab kaugust. Skeletiandmete voog on neist kolmest kõige keerukam, samas ka kõige huvitavam. Sügavuse infot, töödeldakse erinevate algoritmidega ja seeläbi saadakse skeletiandmete voog. Nende andmete abil on võimalik inimeste skeleti jälgimine ruumis ehk *Skeleton Tracking*. Edasi näitan, kuidas need vood kätte saada ja kuidas neid kasutada saab.

5.1 Projekti seadistamine

Järgnevalt kirjeldan, kuidas üles seada projekti, et alustada rakenduse programmeerimist kasutades Kinecti seadet.

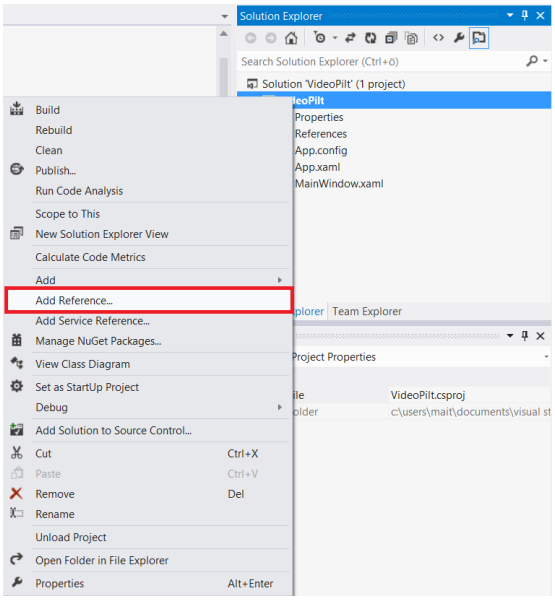
Kõigepealt tuleb käivitada Visual Studio. Järgnevalt valige menüüst File > New > Project. Teile avaneb uus aken, kust saate valida millist projekti soovite luua. Avanenud aknas on vasakul pool menüü, kus tehke järgnevad valikud: Installed > Templates > Visual C# > Windows. Akna keskel on projektide šabloonid, millest tuleb valida WPF Application. Akna all servas tuleb anda projektile nimi (Joonis 3).



Joonis 3. Projekti šablooni valimine

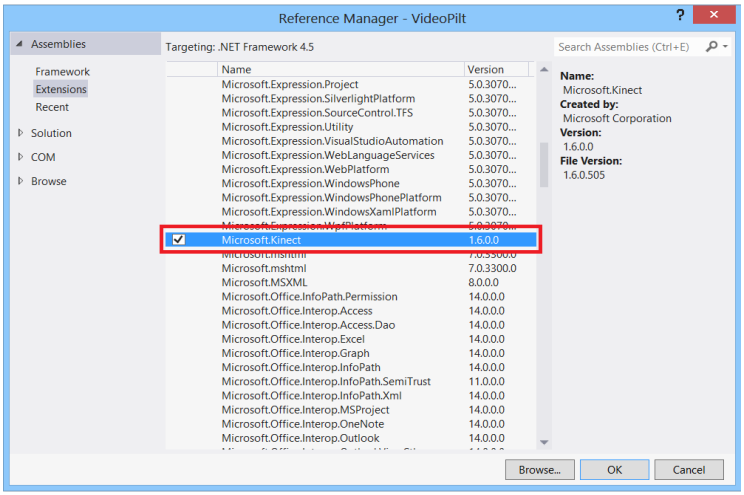
[5] Eestikeelne vaste puudub, edaspidi kasutan mõistet videopildi voog.
[6] Eestikeelne vaste puudub, edaspidi sügavuspildi voog.
[7] Eestikeelne vaste puudub, edaspidi skeletiandmete voog.

Projekt on loodud ja nüüd tuleb see siduda Kinecti andmeteegiga. Selleks tuleb vasakus aknas oleval loodud projektil teha hiire parema klahviga klikk ja avanenud menüüst valida Add Reference (Joonis 4).



Joonis 4. Viite lisamise akna avamine

Selle peale avaneb uus aken, mille vasakul olevast menüüst tuleb valida Extensions. Seejärel ilmuvad keskele erinevad laiendused, mida saab projektile lisada. Nendest laiendustest tuleb valida Microsoft.Kinect (Joonis 5).



Joonis 5. Microsoft Kinect laienduse lisamine

Hetkel veel ei ole võimalik seda teeki kasutada, kuna projektis ei ole öeldud, et seda kasutatakse. Seega tuleb lisada MainWindow.xaml.cs faili juurde järgnev rida – using Microsoft.Kinect (Koodinäide 6).

```
using Microsoft.Kinect;
```

Koodinäide 6. Koodisisene viide Kinect seadme andmeteegile

Kõnealune viide tuleb lisada sinna lõppu, kus on ka projekti loomisel tekitatud viited, mis kõik algavad väljendiga using.

Nüüd on projekti juurde lisatud Kinecti teek. Edasi tuleb määrata, millised funktsioonid programmi käivitumisel käivituvad, ja loodud rakenduse aken avaneb või sulgub.

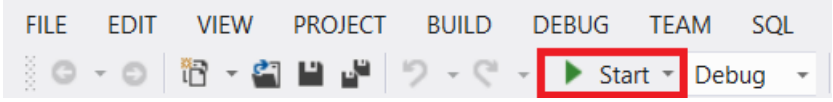
MainWindow.xaml failis tuleb teha graafilises osas klikk MainWindow peal, et muuta aken aktiivseks. Seda tehes ilmuvad alla paremasse nurka Properties paneeli erinevad andmed akna kohta. Paneeli üleval paremas nurgas on ikoon, millel on välgunool.

Ikoonile vajutades ilmuvad paneeli erinevad sündmusetöötlejad. Vasakul on kirjas sündmus ja paremale poole saab kirjutada mis nimega funktsioon käivitub.

Sealt tuleb üles otsida **Loaded** sündmus ja sellele lisada funktsiooni nimi, mis käivitub, kui programmi aken on avanenud (Joonis 6). Vajutades enterit lisab see aknale sündmuse ja lisab ka MainWindow.xaml.cs faili loodud tühja funktsiooni. Tegevust tuleb korrata ja lisada funktsioon mis käivitub, kui aken sulgeb (**Closing**).

Joonis 6. Loaded ja Closing funktsioonide lisamine

Rakenduse käivitamiseks on mitmeid erinevaid viise, aga kõige lihtsamad on menüüribalt Start valiku vajutamine (Joonis 7) või klahvikombinatsiooni Ctrl ja F5 üheaegne vajutamine. Rakendust saab käivitada ka lihtsalt F5 vajutades, aga selliselt programmi käivitamine ei võimalda rakenduse töös leiduvaid vigu tuvastada.



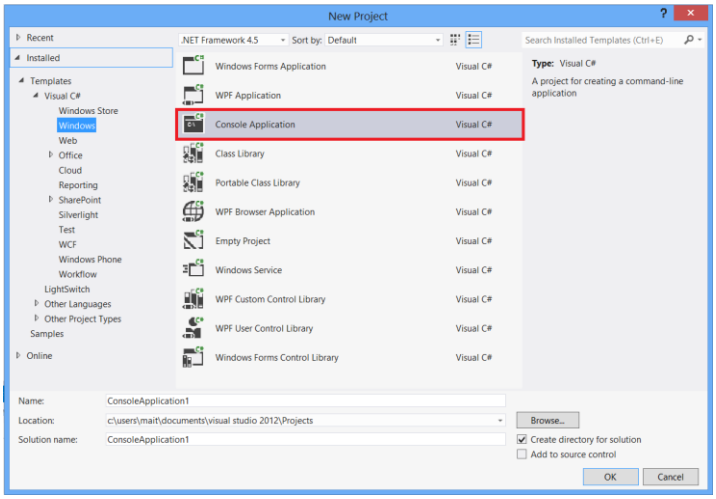
Joonis 7. Programmi käivitamine

Selliselt tuleb valmis seada enda projektid, et edasi saaks programmeerima hakata. Järgnevates näidetes ma ei hakka iga kord projekti seadistust õpetama vaid jätkan selle koha pealt.

5.2 Konsooli rakendus

Esimese projektina teen kõige lihtsama programmi, kus saab kasutada Kinecti. Projekti valmis seadmise juures saab kasutada eelnevat peatükki – „Projekti seadistamine“.

Erinevuseks on see, et me ei kasuta WPF Application šablooni vaid Console Application (Joonis 8). Konsoolirakenduses ei ole programmiakent seega ei pea projekti lisama akna avamisel ja sulgemisel käivituvaid funktsioone.



Joonis 8. Console Application šabloon

Edasi tuleb lisada viide Kinecti andmeteegile, mis on kirjeldatud Projekti üles seadmise peatükis (Projekti seadistamine).

Konsoolirakenduse põhi on valmis ja nüüd saab hakata koodi kirjutama. Kõigepealt tuleb Main’is defineerida sensor. Kasutan kõige lihtsamat võimalust, mis võtab esimese sensori, mille leiab, ja alustan kaadrite püüdmist (Koodinäide 7).

```
//määran sensoriks esimese sensori
KinectSensor sensor = KinectSensor.KinectSensors[0];
//algatan sügavuspildi voo kogumise
sensor.DepthStream.Enable();
```

```
//kogun kaadreid ja iga kaadri peale käivitan funktsiooni sensor_DepthFrameReady
sensor.DepthFrameReady += sensor_DepthFrameReady;
//käivitan sensori
sensor.Start();
// kui on vajutatud tühikut, lõpetab rakendus töö
while (Console.ReadKey().Key != ConsoleKey.Spacebar) { }
```

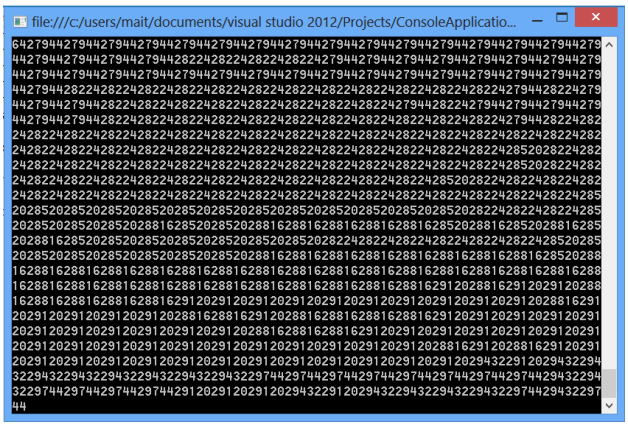
Koodinäide 7. Sensori defineerimine ja kaadrite püüdmine

Peale Main'i lõppu tuleb teha uus funktsioon nimega sensor_DepthFrameReady (Koodinäide 8). Funktsiooni sisenditeks on objekt mis saadab ja kaadri muutujad. Funktsiooni sees näitame, et kasutame sügavuspildi kaadri muutujat ja salvestame selle muutujasse. Kui see kaader on tühi, siis katkestab funktsiooni töö. Kui aga ei ole tühi siis kirjutame kaadri konsooli.

```
static void sensor_DepthFrameReady(object saatja, DepthImageFrameReadyEventArgs e){
	using (var sygavusKaader = e.OpenDepthImageFrame()){
	if (sygavusKaader == null){return;}
		//Tekitame muutuja, mis on sama pikk, kui kaadrist tulevate pikselitejada pikkus
		short[] bitid = new short[sygavusKaader.PixelDataLength];
		//Kopeerime kaadri pikselitejada äsja tekitatud muutjasse
		sygavusKaader.CopyPixelDataTo(bitid);
		//iga bit'i selles uues muutujas kirjutame konsooli välja
		foreach (var bit in bitid)
		Console.Write(bit);
	}
}
```

Koodinäide 8. Sügavuskaadri info konsooli kirjutamine

Koodi käivitamisel näeb rakendus välja midagi sellist:



Joonis 9. Konsooli rakendus

Terviklik kood on õppematerjali lisades (Konsoolirakenduse kood).

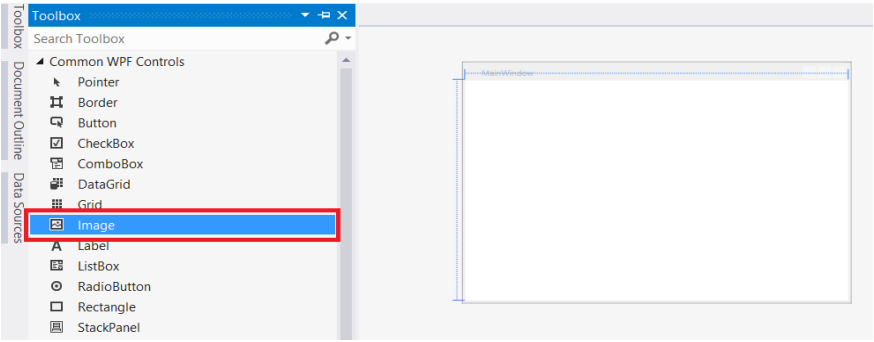
5.2.1 Ülesandeid

- Proovi näide tööle saada.
- Muuda bittide värv roheliseks, et tekiks sarnane efekt nagu on filmis Matrix. Vihjeks niipalju, et muuta tuleb konsooli värvi.

5.3 Videopildi rakendus

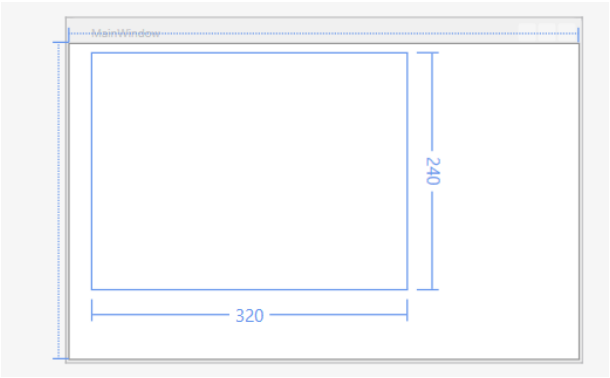
Ma alustan kõige lihtsama voo, videopildi voo edastamise näitega. Videopildi rakenduse loomist jätkan ma kohast, kus lõppes projekti seadistamine varasemas peatükis (Projekti seadistamine).

Avades fail MainWindow.xaml, saame lisada kasti, kuhu hakkab rakenduses tulema videopilt. Sellejaoks tuleb vasakul paanil nimega Toolbox valida Common WPF Controls ja selle alt Image (Joonis 10).



Joonis 10. Videopildi kasti valimine

Edasi tuleb keskel olevale lõuendile tuleb joonistada, kus see pilt asuma hakkab (Joonis 11).



Joonis 11. Videopildi kasti lisamine

Videoakna kood ilmub ka xaml faili. Edasi tuleb paremalt Properties aknast anda video aknale nimi (Joonis 12).

Joonis 12. Videoakna kood xaml failis

Nime kirjutamise lõppedes enterit vajutades tekib ka see xaml faili juurde.

Selles rakenduses on kirjas teine võimalus kuidas koodis sensor kirja panna. Enne MainWindow funktsiooni defineerime ära Kinecti (Koodinäide 9).

```
private KinectSensor sensor;
```

Koodinäide 9. Sensori defineerimine

Projekti loomise juures sai meil loodud kaks funktsiooni: laadimine ja sulgemine. Laadimise funktsioon käivitub akna avanemisel ja seega sinna paneme ka koodi, mis käivitab Kinectist videopildi andmete saamise (Koodinäide 10).

```
//määran sensoriks esimese sensori
this.sensor = KinectSensor.KinectSensors[0];
// algatan videopildi voo kogumise
this.sensor.ColorStream.Enable();
//kogun kaadreid ja iga kaadri peale käivitan funktsiooni sensor_ColorFrameReady
this.sensor.ColorFrameReady += sensor_ColorFrameReady;
//käivitan sensori
this.sensor.Start();
```

Koodinäide 10. Videopildi rakenduse laadimise funktsiooni sisu

Korrektne on, et kui rakendus sulgeda, siis peatatakse ka sensor. Lisame sulgemine funktsiooni sensori peatamise (Koodinäide 11).

```
this.sensor.Stop();
```

Koodinäide 11. Sensori andmetekogumise peatamine

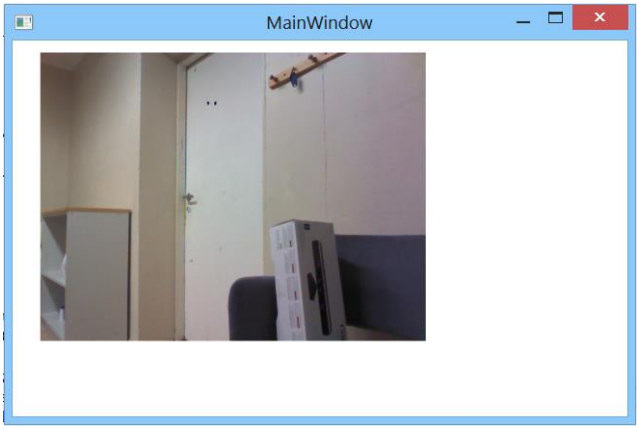
Lisame ka juurde funktsiooni sensor_ColorFrameReady, mis tegutseb edasi Kinectilt saadud kaadritega (Koodinäide 12). Funktsiooni sees näitame, et kasutame videopildi kaadreid. Videopildi kaadrite puudumisel lõpetab funktsioon töö. Kui aga kaader ei ole tühi, siis kirjutame kaadrist tulevad pikselid pildi sisse, mille tekitasime varasemalt. Kuna pilt on pikselite formaadis BGR32, siis iga pikselitejadas oleva pikseli kohta on meil sinine, roheline, punane ja tühi piksel – seega tuleb pikselitejada laius korrutada neljaga.

```
void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e){
      using (ColorImageFrame v2rvipildiKaader = e.OpenColorImageFrame()){
            if (v2rvipildiKaader == null) {return;}
            //tekitame uue bait tüüpi muutuja, millele anname pikkuseks kaadrist
            tulnud pikselite pikkuse
            byte[] pikselid = new byte[v2rvipildiKaader.PixelDataLength];
```

```
            //kopeerime kaadri pikselitejada muutujasse pikselid

            v2rvipildiKaader.CopyPixelDataTo(pikselid);

            int sammupikkus = v2rvipildiKaader.Width * 4;

            //Määran pildi allika, mille jaoks loon bitmap'i. Bitmap'i
            parameetriteks on: pildi laius, kõrgus, Bitmap'i palett, pikselite
            formaat, , pikselid, sammupikkus

            pilt1.Source = BitmapSource.Create(v2rvipildiKaader.Width,
            v2rvipildiKaader.Height, 96, 96, PixelFormats.Bgr32, null, pikselid,
            sammupikkus);

        }

}
```

Koodinäide 12. Videopildi rakenduse värvipildi kaadri töötlemine

Töötav programm näeb välja midagi sarnast:



Joonis 13. Videopilt

Programmi tervikkood asub lisades (Videopildirakenduse kood).

5.4 Sügavuspildi rakendus

Kinect kasutab IR projektorit ja kaamerat, et tekitada sügavuspilt seadme ees olevast vaatest. Erinevalt tavalistest piltidest, kus iga piksel salvestab valguse värvi, siis iga sügavuspildi piksel salvestab objekti kauguse Kinectist selle vaate kohani. Kui vaadata neid sügavuspilte, siis nad näevad välja kui kummaliselt moonutatud must-valged pildid. Pildid näevad imelikud välja, kuna pildi iga osa värv ei näita kui ere objekt on vaid kui kaugel see on. Mida heledamad on kohad pildil, seda lähemal need objektid on. Kui kirjutada iga pikseli eredust uuriv andmevoo töötlemise programm, saame me leida iga Kinecti ees oleva objekti kauguse (Webb ja Ashley, 2012).

Sügavuskaamera pildilt võib leida ka segadusse ajavat infot. Näiteks võivad väga lähedal asuvad objektid näida mustad, mis tähendaks nagu nad asuksid kõige kaugemal. Põhjuseks on, et Kinecti sügavuskaamera hakkab andmeid saama objektidest, mis on vähemalt miinimumkaugusel. Objektid mis on sellest lähemal, on mustad. Pildil võib olla ka objekti ümber imelik must ala ehk vari. Varju tekkimise põhjuseks on, et objekti servadest ei peegelda täpid andurisse korralikult tagasi ja seega ei loe Kinect nende kaugusi välja. Sellise andmete kao korral jätab seade sellele kohale mustad alad. Kui sügavuspildil on peegel või mõni muu peegeldav pind, siis ka nendelt tulev info võib olla segadusse ajav. Nimelt leiab Kinect mitte peegli kauguse vaid peeglis olevate objektide kauguse. Peeglis olevad objektid on erinevatel kaugustel, seega on ka sügavuspildis peegli ühtlase halli tooni asemel objektide kauguse näitamiseks erinevad hallid (kauguse) toonid. Isegi kui objekt on Kinecti seadmele lähemal kui peegel, siis peeglist vaadatuna näib nagu see oleks kaugemal. Põhjus on tarkvaras, mis võtab projitseeritud täpi läbitud teekonna objektini. Seda võib vaadelda ka nii, nagu peegeldatud objekt oleks peegli taga niipalju kui on peegli ja objekti vahel vahemaad (Webb ja Ashley, 2012).

Järgmiseks vaatame, kuidas saada kätte sügavuspildi voog. Sarnaselt videopildile, kuvan rakenduses sügavuspildi välja.

Esimese asjana tuleb projekt seadistada (Projekti seadistamine).

Edasi tuleb sarnaselt videopildi rakendusele tekitada aken kuhu saab hiljem kuvada sügavuspildi (Joonis 12).

Enne MainWindow funktsiooni defineerime ära Kinecti sensori, ülekirjutatava bitmapi, sügavusepikselite massiivi ja värvipikselid.(Koodinäide 13).

```
private KinectSensor sensor;

private WriteableBitmap v2rviBitmap;

private DepthImagePixel[]sygavusPikselid;

private byte[] v2rviPikselid;
```

Koodinäide 13. Sügavuse rakenduse muutujate deklareerimine

Laadimise funktsiooni lisame koodi, mis näitab millist sensorit me hakkame rakenduses kasutama (Koodinäide 14). Kui programm on sensori üles leidnud, siis saame hakata andmeid küsima. Algatan sügavuspildi voo kogumise ja määran tuleva voo resolutsiooni ja kaadrisageduse. Edasi loome Bitmap'i ja määrame selle rakenduses välja näidatava pildi allikaks. Viimase asjana enne sensori käivitamist määrame funktsiooni, mis käivitub kaadrite lisandumisel.

```
if (null != this.sensor){//siin on kood, mis käivitub kui sensor on leitud

      this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);

      // muutuja, kuhu eraldame ruumi tulevatele sügavuse pikselitele

      this.sygavusPikselid=new
      DepthImagePixel[this.sensor.DepthStream.FramePixelDataLength];

      // muutuja, kuhu eraldame ruumi loodavatele värvi pikselitele

      this.v2rviPikselid=new   byte[this.sensor.DepthStream.FramePixelDataLength   *
      sizeof(int)];

      //Bitmap'i parameetriteks on pildi laius, kõrgus, dpi x, dpi y, pikselite
      formaat, .

      this.v2rviBitmap=new      WriteableBitmap(this.sensor.DepthStream.FrameWidth,
      this.sensor.DepthStream.FrameHeight, 96.0, 96.0, PixelFormats.Bgr32, null);

      this.pilt1.Source = this.v2rviBitmap;

      //kogun   kaadreid   ja   iga   kaadri   peale   käivitan   funktsiooni   this.
      SensorDepthFrameReady

      this.sensor.DepthFrameReady += this.SensorDepthFrameReady;

      //käivitan sensori

      this.sensor.Start();

}
```

Koodinäide 14. Sügavuspildi rakenduse laadimise funktsiooni sisu

Järgmiseks kirjutame funktsiooni SensorDepthFrameReady, mis käivitub siis kui tuleb uus kaader (Koodinäide 15). Funktsiooni sisenditeks on objekt mis saadab ja kaadri muutujad.

Funktsiooni sees näitame, et kasutame sügavuspildi kaadri muutujat ja salvestame selle muutujasse. Kui kaader ei ole tühi, töötleme andmeid ja kirjutame pikselid bitmap'i.

```
private void SensorDepthFrameReady(object saatja, DepthImageFrameReadyEventArgs e){
	using (DepthImageFrame sygavusKaader = e.OpenDepthImageFrame()){
		if (sygavusKaader != null){
			// kirjutame pikselite andmed ajutisse massiiv
			sygavusKaader.CopyDepthImagePixelDataTo(this.sygavusPikselid);
			//tekitan muutujad kaadri maksimum- ja miinimum sügavuseks
			int minSygavus = sygavusKaader.MinDepth;
			int maxSygavus = sygavusKaader.MaxDepth;
			//muudame sügavuse RGB'ks
			int v2rviPikseliIndeks = 0;
			for (int i = 0; i < this. sygavusPikselid.Length; ++i)
			{
				//pikseli sygavus
				short sygavus = sygavusPikselid[i].Depth;
				//Väärtused mis jäävad miinimumist ja maksimumist
				väljapoole muudetakse 0ks ehk mustaks

				byte intensiivsus = (byte)(sygavus >= minSygavus &&
				sygavus <= maxSygavus ? sygavus: 0);
				//sinine byte
				this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
				//roheline byte
				this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
				//punane byte
				this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
				//kuna me näitame BGR pilti, siis viimane bait on
				kasutamata 32 bitist. Viimast kasutaks alpha korral BGRA
				korral.
				 ++ v2rviPikseliIndeks;
			}
			//kirjutame pikselite info bitmap'i. Parameetriteks on .....
			this.v2rviBitmap.WritePixels(
```

```
                                new     Int32Rect(0,     0,     this.v2rviBitmap.PixelWidth,
                                this.v2rviBitmap.PixelHeight),

                                this.v2rviPikselid,

                                this.v2rviBitmap.PixelWidth * sizeof(int), 0

                                );

                        }

                }

        }
```

Koodinäide 15. Sügavuskaadri töötlemine

Programmi käivitamisel näeb aken välja midagi sarnast:

Joonis 14. Sügavuspilt

Terviklik kood on õppematerjali lisades (Sügavuspildirakenduse kood).

5.4.1 Ülesandeid

- Muuda koodi nii, et pikselid, mis on kindlast vahemaast tagapool, muudetakse mustaks. Näiteks, kui kasutaja on seadmest 2 meetri kaugusel, siis kõik mis temast tahapoole jääb on must.
- Muuda koodi nii, et kõik mis on kindlast vahemaast eespool, muudetakse mustaks. Näiteks, kui kasutaja on 2 meetri kaugusel, siis temast eespool olevaid esemeid ei kuvataks.

5.5 Skeleti rakendus

Rakenduses kasutan Skeletonization-tarkvara, mis järeldab kasutaja skeleti asukoha (täpsemalt kasutaja liigeste ja neid ühendavate luude asukoha) Kinectilt saadud info põhjal (Borenstein, 2012).

Skeleton tracking süsteemid analüüsivad sügavuspilti rakendades keerulisi algoritme, mis kasutavad maatriksi teisendusi, masinõpet ja teisi vahendeid, et arvutada skeleti punkte (Webb ja Ashley, 2012).

Skeleti andmed tulevad Skeletiandmete voost. Andmevoo andmetele saab ligi kas läbi *event*'ide või *polling*'ut kasutades (Webb ja Ashley, 2012). Õppematerjalis keskendun ma *event*ide kasutamisele.

Objektil KinectSensor on *event* nimega SkeletonFrameReady, mis käivitub iga kord, kui uued skeletiandmed on saadaval. Skeletiandmeid on võimalik saada ka AllFramesReady *event*'ist. Iga kaader SkeletonStream'ist esitab skeleti objektide kogumikku. Iga selline objekt sisaldab andmeid, mis kirjeldavad sketleti ja liigeste asukohta. Igal liigesel on oma identiteet (pea, õlad, küünarnukid, jne.) ja 3D vektor (Webb ja Ashley, 2012).

Selles peatükis valmib rakendus, mis kuvab kasutaja liigeste punktid ja neid ühendavad jooned ekraanile.

Esimese asjana tuleb projekt seadistada (Projekti seadistamine).

Tekitame MainWindow.xaml faili akna kuhu hakkame hiljem joonistama kasutaja skeletti. Pildile tuleb anda ka nimi, et selle poole saaks pöörduda. Enne MainWindow funktsiooni defineerime mõned muutujad (Koodinäide 16).

```
//Joonistamise ala laius
private const float esitusLaius= 640.0f;
//Joonistamise ala kõrgus
private const float esitusKorgus= 480.0f;
//Liigeste suurus
private const double liigeseSuurus= 3;
//Liigese, mille kohta on andmed olemas, värv
private readonly Brush jalgitavaLiigeseVarv= new SolidColorBrush(Color.FromArgb(255, 68, 192, 68));
```

```
// Liigese, mille kohta ei saa andmeid kätte ja algoritm oletab asukoha, värv

private readonly Brush jareldatavaLiigeseVarv= Brushes.Yellow;

//Liigeste vaheliste joonte, niiöelda luude, mis oletatakse liigeste asukohtade
järgi, värv

private readonly Pen jareldatavaLuuVarv= new Pen(Brushes.Gray, 1);

private KinectSensor sensor;

//Skeleti joonistuse grupp

private DrawingGroup joonistusGrupp;

//Pildi allikas

private DrawingImage pildiAllikas;
```

Koodinäide 16. Skeleti rakenduse muutujate defineerimine

Laadimisne funktsioonis määrame pildi allikaks joonistuse grupi, mille sisse hakkame lisama vajalikke elemente. Võimaldame ka skeletiandmete voo püüdmise (Koodinäide 17). Kui kaader on kätte saadud, käivitub funktsioon sensor_AllFramesReady, kus töödeldakse kaadri andmeid.

```
private void laadimine(object sender, RoutedEventArgs e)
{
       this.sensor = KinectSensor.KinectSensors[0];
        this.joonistusGrupp = new DrawingGroup();
        this.pildiAllikas = new DrawingImage(this.joonistusGrupp);
        pilt1.Source = this.pildiAllikas;

        if (null != this.sensor)
        {
             this.sensor.SkeletonStream.Enable();
             this.sensor.AllFramesReady += this.sensor_AllFramesReady;
             this.sensor.Start();
        }
}
```

Koodinäide 17. Skeleti rakenduse laadimise funktsioon

Kinecti tarkvara võimaldab meil jälgida mitut kasutajat korraga seega esimese asjana sensor_AllFramesReady funktsioonis tekitame skelettide massiivi.

Skeleti kaadri saabumisel tekitame andmetest skeleti (mitme kasutaja korral skeletid) ja lisame selle skelettide massiivi (Koodinäide 18).

```
Skeleton[] skeletid = new Skeleton[0];

using (SkeletonFrame skeletiKaader = e.OpenSkeletonFrame())

{

	if (skeletiKaader != null)

		{

			skeletid = new Skeleton[skeletiKaader.SkeletonArrayLength];

		  skeletiKaader.CopySkeletonDataTo(skeletid);

	}

}
```

Koodinäide 18. Skeleti rakenduses massiivi skeletid täitmine skelettidega

Järgmisena kasutame joonistuse gruppi, kuhu tekitame skeletid massiivis olevad luustikud (Koodinäide 19).

```
using (DrawingContext joonistuseKontekst = this.joonistusGrupp.Open())

{

	joonistuseKontekst.DrawRectangle(Brushes.Black, null, new Rect(0.0, 0.0,
	esitusLaius, esitusKorgus));


	if (skeletid.Length != 0)

		{

			foreach (Skeleton skel in skeletid)

		  {

			this.JoonistaLuudJaLiigesed(skel, joonistuseKontekst);

			}

	 }

	// Ei lase joonistada väljapoole joonistuse gruppi

	this.joonistusGrupp.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0,
	esitusLaius, esitusKorgus));

}
```

Koodinäide 19. Skeleti rakenduses skelettide joonistamine joonistuse gruppi

Iga skeleti massiivis joonistame välja kasutades funktsiooni JoonistaLuudJaLiigesed (Koodinäide 20). Kõik luud joonistame välja funktsiooniga JoonistaLuu, mis ühendab omavahel sisendina sisse antud liigesed. Kui kõik luud on joonistatud, joonistame ka kõik liigesed. Liigesed mille kohta on Kinectil andmed, joonistab ühte värvi. Need liigesed, mille kohta puudub info, nende asukoha oletab Skeletonization tarkvara ja need värvitakse teist värvi.

```
private void JoonistaLuudJaLiigesed(Skeleton skelett, DrawingContext
joonistuseKontekst)

{

	// pea

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.Head,
	JointType.ShoulderCenter);

	// õlavööde

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
	JointType.ShoulderLeft);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
	JointType.ShoulderRight);

	// selgroog

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
	JointType.Spine);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.Spine,
JointType.HipCenter);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipCenter,
JointType.HipLeft);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipCenter,
JointType.HipRight);

	// vasak käsi

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderLeft,
	JointType.ElbowLeft);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ElbowLeft,
	JointType.WristLeft);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.WristLeft,
JointType.HandLeft);

	// parem käsi

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderRight,
	JointType.ElbowRight);

	this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ElbowRight,
	JointType.WristRight);
```

```
        this.JoonistaLuu(skelett,      joonistuseKontekst,      JointType.WristRight,
        JointType.HandRight);

        // vasak jalg

        this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.HipLeft,
JointType.KneeLeft);

        this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.KneeLeft,
JointType.AnkleLeft);

        this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.AnkleLeft,
JointType.FootLeft);

        // parem jalg

        this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.HipRight,
JointType.KneeRight);

        this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.KneeRight,
        JointType.AnkleRight);

        this.JoonistaLuu(skelett,      joonistuseKontekst,      JointType.AnkleRight,
        JointType.FootRight);


        foreach (Joint liiges in skelett.Joints)

                {

                        Brush liigeseVarv = null;

                //liigese andmed on olemas

                        if (liiges.TrackingState == JointTrackingState.Tracked)

                        {

                                liigeseVarv = this.jalgitavaLiigeseVarv;

                }

                //liigese andmed ei ole olemas

                        else if (liiges.TrackingState == JointTrackingState.Inferred)

                        {

                                liigeseVarv = this.jareldatavaLiigeseVarv;

                }

                        if (liigeseVarv != null)

                {

                        //liigese joonistamine

                        joonistuseKontekst.DrawEllipse(liigeseVarv,                null,
                        this.SkeletiPunktEkraanile(liiges.Position),      liigeseSuurus,
                        liigeseSuurus);
```

```
            }

        }

}
```

Koodinäide 20. Skeleti rakenduse JoonistaLuudJaLiigesed funktsioon

Luu joonistamine on iseenesest lihtne – joonega tuleb ühendada kaks liigest (Koodinäide 21).

```
private void JoonistaLuu(Skeleton skelett, DrawingContext joonistuseKontekst,
JointType liigeseTyyp1, JointType liigeseTyyp2)

{

        Joint liiges1 = skelett.Joints[liigeseTyyp1];

              Joint liiges2 = skelett.Joints[liigeseTyyp2];

        Pen luuVarv = this.jareldatavaLuuVarv;

        joonistuseKontekst.DrawLine(luuVarv,
        this.SkeletiPunktEkraanile(liiges1.Position),
        this.SkeletiPunktEkraanile(liiges2.Position));

}
```

Koodinäide 21. Skeleti rakendamise JoonistaLuu funktsioon

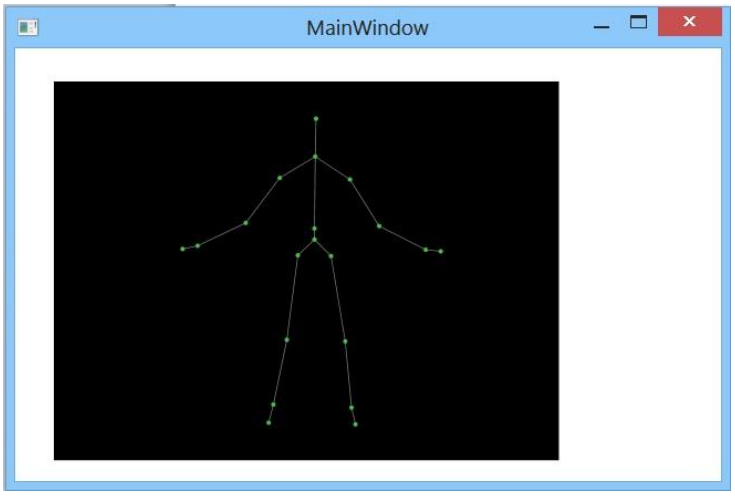
Mõlemas, nii luude kui liigeste joonistamiste juures, olen ma kasutanud funktsiooni SkeletiPunktEkraanile, mille ülesandeks on viia skeleti punkt sügavuspildi punktiks, et seda saaks välja kuvada (Koodinäide 22).

```
private Point SkeletiPunktEkraanile(SkeletonPoint skeletiPunkt)

{

        DepthImagePoint                          sygavusPunkt                          =
        this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skeletiPunkt,
        DepthImageFormat.Resolution640x480Fps30);

        return new Point(sygavusPunkt.X, sygavusPunkt.Y);

}
```

Koodinäide 22. Skeleti rakenduse funktsioon SkeletiPunktEkraanile

Funktsioon sulgemine näeb välja samasugune, nagu eelmistes rakendustes, kus me lõpetasime sensori töö.

Programmi käivitamisel näeb rakendus välja selline:



Joonis 15. Skeletivoo andmete pilt

Selle rakenduse failis MainWindow.xaml.cs olev kood on õppematerjali lisades (Skeletirakenduse kood).

5.5.1 Ülesandeid

- Muuda koodi nii, et jälgitavad liigesed oleksid punast värvi.
- Muuda luude jämedust nii, et need oleksid sama jämedad kui liigesed.
- Kui antud programmis muutuvad need liigesed, mille kohta infot ei saa kätte, kollaseks, siis tee nii, et liigeseid ei näita kui andmed puuduvad.

5.6 Videopildi ja sügavusandmete voo ühendamine

Selles peatükis ühendame videopildi ja sügavuspildi voo. Tulemuseks on joonistamise programm, kus Kinecktist saadud videopildi peale saab joonistada.

Kõigepealt seadistame projekti nagu eelnevates peatükkides (Projekti seadistamine).

Enne MainWindow funktsiooni defineerime mõned muutujad (Koodinäide 23).

```
private KinectSensor sensor;

//sügavuspildi pikselid

private DepthImagePixel[] sygavusPikselid;

//Kuna me teeme videopildi peale teise, läbipaistvate pikselitega pildi, kuhu saab joonistada, siis tuleb ka uuele pildile anda mõõdud

const int varvimisePildiLaius = 640;

const int varvimisePildiKorgus = 480;

//Uue pildi suurus baitides

const int varvimisePildiBaitideSuurus = varvimisePildiKorgus * varvimisePildiLaius *4;

//Uue pildi baidid

byte[] varvimisePildiBaidid = new byte[varvimisePildiBaitideSuurus];

//Määrame vahemiku milles olles joonistatakse pildile

int varvimiseMaxKaugus = 1500;

int varvimiseMinKaugus = 1000;
```

Koodinäide 23. Joonistamise rakenduse muutujate defineerimine

Joonistamise programmi tööpõhimõte on, et kui kehaosa satub eelnevalt defineeritud vahemikku, siis joonistatakse pildi peale sinise pooleldi läbipaistva värviga.

Kuna me seekord soovime saada kahte erinevat voogu, siis tuleb ka mõlemate voogude püüdmine võimaldada (Koodinäide 24). Anname ka teada millises formaadis me soovime andmeid kätte saada. Sügavuspildi pikselite massiivi pikkuseks saab sügavusvoo kaadri pikselite andmete pikkus. Enne sensori käivitamist määrame funktsiooni, mis käivitub kui kaadrite info on olemas.

```
private void laadimine(object sender, RoutedEventArgs e)

{

	this.sensor = KinectSensor.KinectSensors[0];
```

```
        if (null != this.sensor)
        {
            this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30
            );
            this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fp
            s30);
            this.sygavusPikselid                    =                    new
            DepthImagePixel[this.sensor.DepthStream.FramePixelDataLength];
            this.sensor.AllFramesReady += this.SensorAllFramesReady;
            this.sensor.Start();
        }
}
```

Koodinäide 24. Joonistamise rakenduse laadimise funktsioon

SensorAllFramesReady funktsioonis kasutame videopildi- (Koodinäide 25) ja sügavuskaadreid (Koodinäide 26). Sügavuskaadri andmed salvestame muutujasse nimega sygavusKaader. Enne kaadri andmete kasutamist tuleb kindlaks teha, et see ei ole tühi. Sügavuskaadrist salvestame pikselite info ajutisse massiivi nimega sygavusPikselid. Kuna me kasutame rakenduses värvilist pilti videopildi peal, siis tuleb muuta sügavuse info RGB infoks. Käime tsükliga läbi kõik pikselid sygavusPikselid massiivis ja kontrollime kas saadud sügavus jääb eelnevalt defineeritud vahemikku. Need pikselid mis on selles vahemikus, värvime funktsioonis varviPikselid pildil siniseks(Koodinäide 27). Funktsiooni varviPikselid sisenditeks on: pikseli asukoht, sinise värvi väärtus, rohelise värvi väärtus, punase värvi väärtus ja läbipaistvus.

```
using (DepthImageFrame sygavusKaader = e.OpenDepthImageFrame())
 {
        if (sygavusKaader!= null)
        {
            sygavusKaader.CopyDepthImagePixelDataTo(this.sygavusPikselid);
            // muudame sügavuse andmed RGB andmeteks
            for (int i = 0; i < this.sygavusPikselid.Length; ++i)
            {
                // pikseli kaugus
                short sygavus = sygavusPikselid[i].Depth;
```

```
                    if (varvimiseMinKaugus < sygavus && sygavus <
                    varvimiseMaxKaugus)
                    {
                        // värvimise funktsioon
                    varviPikselid(i * 4, 255, 0, 0, 100);
                    }
                }
            }
}
```

Koodinäide 25. Joonistamise rakenduse sügavuskaadrite kasutamine

Kontrollime ka videopildikaadri olemasolu. Videopildi näitamise jaoks määrame edusammu vastavalt videopildi kaadri laiusele. Tekitame baitide massivi nimega pikselid, kuhu kopeerime videopildi pikselite andmed.

```
using (ColorImageFrame v2rvipildiKaader = e.OpenColorImageFrame())
{
if (v2rvipildiKaader == null)
{
    return;
}

    int sammupikkus = v2rvipildiKaader.Width * 4;
    byte[] pikselid = new byte[v2rvipildiKaader.PixelDataLength];
    v2rvipildiKaader.CopyPixelDataTo(pikselid);
    pilt2.Source = BitmapSource.Create(v2rvipildiKaader.Width,
v2rvipildiKaader.Height,
        96, 96, PixelFormats.Bgr32, null, pikselid, sammupikkus);
    pilt1.Source = BitmapSource.Create(v2rvipildiKaader.Width,
v2rvipildiKaader.Height,
        96, 96, PixelFormats.Bgra32, null, varvimisePildiBaidid, sammupikkus);
}
```

Koodinäide 26. Joonistamise rakenduse videopildi kaadrite kasutamine

```
private void varviPikselid(int asukoht, byte b, byte g, byte r, byte a)
{
```

```
        // joonistatava pildi sinise pikseli väärtus
        varvimisePildiBaidid[asukoht] = b;
        // järgmise pikseli asukoht
        asukoht++;
        // joonistatava pildi rohelise pikseli väärtus
        varvimisePildiBaidid[asukoht] = g;
        asukoht++;
        // joonistatava pildi punase pikseli väärtus
        varvimisePildiBaidid[asukoht] = r;
        asukoht++;
        // joonistatava pildi läbipaistvuse pikseli väärtus
        varvimisePildiBaidid[asukoht] = a;
}
```

Koodinäide 27. Joonistamise rakenduse värvitava pildi pikselite väärtuste andmise funktsioon

Käivitades rakenduse, tekib aken, kus on näha videopilt. Kui kasutaja või ese satub värvimisvahemikku, muutuvad pildil pikselid selle koha pealt siniseks(Joonis 16. Joonistamise rakendus). Rakenduse terviklik kood on lisades (Joonistamise rakenduse kood).

Joonis 16. Joonistamise rakendus

5.6.1 Ülesandeid

- Muuda joonistuse värvi.
- Tee nii, et joonistus ei oleks läbipaistev.

- Lisa pildi alla 3 nuppu, pane nende nimesiltideks: punane, kollane, roheline. Nupule vajutades muutub värv vastavalt nupu sildile. Vajutades rohelisele nupule saab edasi joonistada rohelise värviga.

5.7 Videopildi ja skeletiandmete voo ühendamine

Skeletiandmete voo oleme juba mustale taustale kuvanud, seega on mõistlik jätk kuvada skelett videopildi peale, et näha kas Kinect ka tegelikult inimese ära tunneb. Tulemuse saavutamiseks on vaja ühendada videopildi ja skeleti rakendus. Aluseks võtame skeleti rakenduse koodi ja lisame sinna videopildi koodi.

Projekti MainWindow.xaml failis tuleb Image elemendi juurde teha teine samasugune element ja panna sellele teine nimi (Koodinäide 28).

```
<Grid>

      <Image      x:Name="pilt2"      HorizontalAlignment="Left"      Height="480"
      Margin="28,25,0,0" VerticalAlignment="Top" Width="640"/>

      <Image      x:Name="pilt1"      HorizontalAlignment="Left"      Height="480"
      Margin="28,25,0,0" VerticalAlignment="Top" Width="640"/>

</Grid>
```

Koodinäide 28. Videopildi ja skeletiandmete voo ühendamise projekti Image elemendid

Selle jaoks, et seade hakkaks videopilti püüdma, on vaja võimaldada lisaks skeletiandmete voole ka videopildi voo püüdmine (Koodinäide 29).

```
this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);

this.sensor.SkeletonStream.Enable();
```

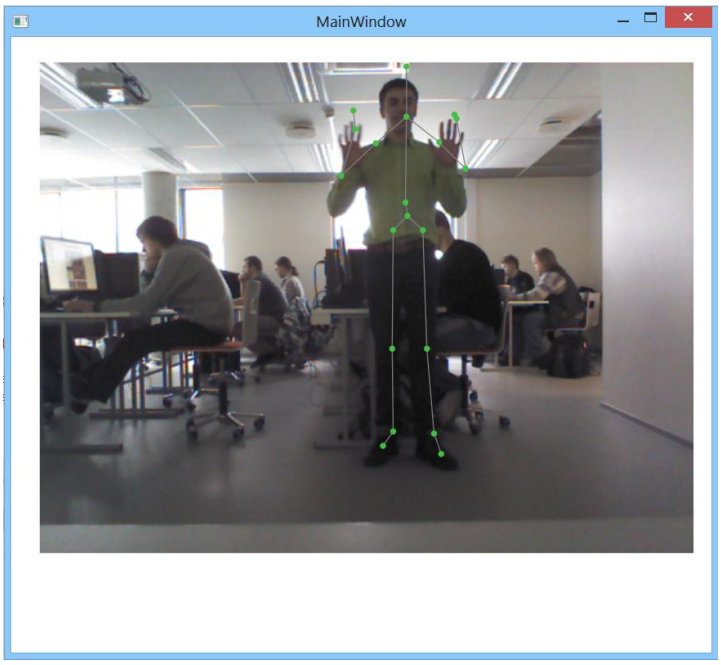
Koodinäide 29. Skeletivoo ja videovoo püüdmise võimaldamine

Projektis tegeles skeletikaadrite andmetega funktsioon sensor_AllFramesReady ja seal töötleme ka videopildi kaadreid. Videopildi kaadreid tööleme samamoodi nagu videopildi rakenduses sensor_ColorFrameReady funktsiooni sees (Koodinäide 12). Pildi nimeks aga ei ole enam pilt1 vaid antud kontekstis on pilt2. Tulemuseks on meil kaks pilti – videopilt ja selle peal olev musta taustaga skeleti pilt. Viimase asjana eemaldame skeletipildilt musta tausta. Musta tausta läbipaistvaks muutmiseks ei ole muud vaja teha, kui joonistamise kontekstis asendada must läbipaistvusega (Koodinäide 30).

```
joonistuseKontekst.DrawRectangle(Brushes.Transparent,  null,  new  Rect(0.0,  0.0,
esitusLaius, esitusKorgus));
```

Koodinäide 30. Joonistuse konteksti värvuse läbipaistvaks muutmine

Projekti käivitamisel tuleb sarnane pilt:



Joonis 17. Videopildi ja skeletiandmete ühendamise rakenduse tulemus

Rakenduse faili MainWindow.xaml.cs kood on õppematerjali lisades (Videopildi ja skeletiandmete voo ühendamise kood).

5.7.1 Ülesandeid

- Kuva videopildi peale ainult labakäte liigesed.
- Moodusta videopildile teine skelett, mis on dubleeritud esimesest ja asub nihkega vasakul.
- Muuda dubleeritud skelett esimese peegelpildiks.

5.8 Kinecti nurga liigutamine

Lisaks erinevate voogude püüdmisele on Kinectil võime ennast ise mootori abil liigutada. Õppematerjali alguses on lühidalt kirjeldatud Kinecti mootorit. Kuna Kinect on võimeline ise ennast liigutama, on võimalik kirjutada rakendusi, mis on suurema või väiksema vaateväljaga kui on kaamerate oma. Sellist vaatevälja muutust on otstarbekas kasutada siis, kui kasutaja peab palju liikuma kas lähemale või kaugemale.

Sensori nurga muutmine on väga lihtne. Nurga väärtus on kirjas sensori muutujas ElevationAngle. Näiteks kui tahame kaameraid suunata 3 kraadi võrra ülespoole, tuleb lihtsalt senisele nurgale liita 3 kraadi juurde (Koodinäide 31).

```
this.sensor.ElevationAngle = this.sensor.ElevationAngle - 3;
```

Koodinäide 31. Sensori nurga muutmine

5.8.1 Ülesandeid

- Võta ette videopildi rakendus. Lisa sinna kaks nuppu ja nimeta need Üles ja Alla nuppudeks. Tee nii, et nupule vajutades liigub kaamera vastavalt kas üles või alla.
- Võta ette skeletiandmete voo ja videopildi ühisrakendus ja muuda koodi nii, et kui mingi liiges liigub kõrgemale või madalamale kui on videopildi piirid, siis Kinect liigutab ennast kas üles või alla.

6 Kokkuvõte

Õppematerjalis saime kätte video-, sügavus- ja skeletiandmete voo. Uurides seadme ja SDK rohkemaid võimalusi on võimalik luua väga keerukaid süsteeme. Tavaliselt ei ole ükski Kinecti rakendus nii lihtsakoeline kui antud näidetes, aga nende eesmärgiks oli anda lihtne ülevaade kuidas andmeid kätte saada ja neid kasutada.

7 Kasutatud kirjandus

Vallaste. Külastatud 11. detsembril, 2013, aadressil http://www.vallaste.ee

http://www.microsoft.com/en-us/download/details.aspx?id=40278

Webb, J., Ashley J. (2012). *Beginning Kinect Programming with the Microsoft Kinect SDK*. New York: Apress.

Borenstein, G. (2012). *Making Things See*. Sebastopol: Maker Media

Catuhe, D. (2012). *Programming with the Kinect for Windows Software Development Kit*. Washington: Microsoft Press.

Miles, R. (2012). *Microsoft Press Start Here Learn the Kinect API*. Washington: Microsoft Press.

Windows. (2013). Kinect for Windows SDK (Versioon 1.8) [Tarkvara]. Külastatud 11. detsembril, 2013, aadressil http://www.microsoft.com/en-us/download/details.aspx?id=40278

Windows. (2013). Microsoft Visual Studio [Tarkvara]. Külastatud 11. detsembril, 2013, aadressil http://www.visualstudio.com/en-us/downloads/

8 Lisad

8.1 Konsoolirakenduse kood

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.Kinect;


namespace KonsooliRakendus

{

    class Program

    {

        static void Main(string[] args)

        {

            KinectSensor sensor = KinectSensor.KinectSensors[0];


            sensor.DepthStream.Enable();

            sensor.DepthFrameReady += sensor_DepthFrameReady;


            sensor.Start();

            while (Console.ReadKey().Key != ConsoleKey.Spacebar) { }

        }

        static          void          sensor_DepthFrameReady(object          saatja,
DepthImageFrameReadyEventArgs e)

        {

            using (var sygavusKaader = e.OpenDepthImageFrame())

            {

                if (sygavusKaader == null) {return;}

                short[] bitid = new short[sygavusKaader.PixelDataLength];

                sygavusKaader.CopyPixelDataTo(bitid);
```

```
                foreach (var bit in bitid)
                    Console.Write(bit);
            }
        }
    }
}
```

8.2 Videopildirakenduse kood

MainWindow.xaml:

```
<Window x:Class="VideoPilt.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow"     Height="350"     Width="525"     Closing="sulgemine"
Loaded="laadimine">
    <Grid>
       <Image      x:Name="pilt1"      HorizontalAlignment="Left"      Height="240"
       Margin="23,10,0,0" VerticalAlignment="Top" Width="320"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;

namespace VideoPilt
{
    public partial class MainWindow : Window
    {
        private KinectSensor sensor;

        public MainWindow()
        {
            InitializeComponent();
        }

        private void sulgemine(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            this.sensor.Stop();
        }

        private void laadimine(object sender, RoutedEventArgs e)
        {
            this.sensor = KinectSensor.KinectSensors[0];

            this.sensor.ColorStream.Enable();
            this.sensor.ColorFrameReady += sensor_ColorFrameReady;

            this.sensor.Start();
        }

        void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
```

```
        {
            using (ColorImageFrame v2rvipildiKaader = e.OpenColorImageFrame())
            {
                if (v2rvipildiKaader == null)
                {
                    return;
                }
                byte[] pikselid = new byte[v2rvipildiKaader.PixelDataLength];
                v2rvipildiKaader.CopyPixelDataTo(pikselid);

                int sammupikkus = v2rvipildiKaader.Width * 4;
                pilt1.Source = BitmapSource.Create(v2rvipildiKaader.Width,
                v2rvipildiKaader.Height, 96, 96, PixelFormats.Bgr32, null, pikselid,
                sammupikkus);
            }
        }
    }
}
```

8.3 Sügavuspildirakenduse kood

MainWindow.xaml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;

namespace SygavusPilt
{
    public partial class MainWindow : Window
    {
        private KinectSensor sensor;
        private WriteableBitmap v2rviBitmap;
        private DepthImagePixel[]sygavusPikselid;
        private byte[] v2rviPikselid;

        public MainWindow()
        {
           InitializeComponent();
        }

        private void laadimine(object sender, RoutedEventArgs e)
        {
             this.sensor = KinectSensor.KinectSensors[0];
             if (null != this.sensor)
             {
                   this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x48
             0Fps30);
                   this.              sygavusPikselid              =              new
                   DepthImagePixel[this.sensor.DepthStream.FramePixelDataLength];
                   this.v2rviPikselid                        =                      new
                   byte[this.sensor.DepthStream.FramePixelDataLength                *
                   sizeof(int)];
                   this.v2rviBitmap                        =                      new
                   WriteableBitmap(this.sensor.DepthStream.FrameWidth,
                   this.sensor.DepthStream.FrameHeight,         96.0,         96.0,
                   PixelFormats.Bgr32, null);
```

```
                this.pilt1.Source = this.v2rviBitmap;
                this.sensor.DepthFrameReady+= this.SensorDepthFrameReady;
                this.sensor.Start();
            }
        }

    private void sulgemine(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
        }

        private void SensorDepthFrameReady(object sender,
DepthImageFrameReadyEventArgs e)
        {
            using (DepthImageFrame sygavusKaader = e.OpenDepthImageFrame())
            {
            if (sygavusKaader!= null)
              {
                sygavusKaader.CopyDepthImagePixelDataTo(this. sygavusPikselid);
                 int minSygavus = sygavusKaader.MinDepth;
                 int maxSygavus = sygavusKaader.MaxDepth;
                 int v2rviPikseliIndeks = 0;
                 for (int i = 0; i <   this. sygavusPikselid.Length; ++i)
                       {
                      short sygavus = sygavusPikselid [i].Depth;
                      byte intensiivsus = (byte)(sygavus >= minSygavus &&
                      sygavus <= maxSygavus ? sygavus: 0);
                      this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
                      this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
                      this.v2rviPikselid [v2rviPikseliIndeks++] = intensiivsus;
```

```
                        ++ v2rviPikseliIndeks;

                         }

                    this.v2rviBitmap.WritePixels(      new      Int32Rect(0,      0,
                    this.v2rviBitmap.PixelWidth, this.v2rviBitmap.PixelHeight),

                      this.v2rviPikselid,

                      this.v2rviBitmap.PixelWidth * sizeof(int),0);

                }

            }

        }

    }

}
```

8.4 Skeletirakenduse kood

MainWindow.xaml.cs:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Microsoft.Kinect;


namespace SkeletiPilt

{
```

```
public partial class MainWindow : Window
{
    private const float esitusLaius = 640.0f;
    private const float esitusKorgus = 480.0f;
    private const double liigeseSuurus = 3;
    private readonly Brush jalgitavaLiigeseVarv = new
   SolidColorBrush(Color.FromArgb(255, 68, 192, 68));
    private readonly Brush jareldatavaLiigeseVarv = Brushes.Yellow;
    private readonly Pen jareldatavaLuuVarv = new Pen(Brushes.Gray, 1);
    private KinectSensor sensor;
    private DrawingGroup joonistusGrupp;
    private DrawingImage pildiAllikas;

    public MainWindow()
    {
        InitializeComponent();
    }
    private void laadimine(object sender, RoutedEventArgs e)
    {
        this.sensor = KinectSensor.KinectSensors[0];
        this.joonistusGrupp = new DrawingGroup();

        this.pildiAllikas = new DrawingImage(this.joonistusGrupp);
        pilt1.Source = this.pildiAllikas;

        if (null != this.sensor)
        {
            this.sensor.SkeletonStream.Enable();
            this.sensor.AllFramesReady += this.sensor_AllFramesReady;

            this.sensor.Start();
        }
```

```
        }

        private void sensor_AllFramesReady(object sender, AllFramesReadyEventArgs
e)
        {
            Skeleton[] skeletid = new Skeleton[0];
            using (SkeletonFrame skeletiKaader = e.OpenSkeletonFrame())
            {
                if (skeletiKaader != null)
                {
                    skeletid = new Skeleton[skeletiKaader.SkeletonArrayLength];
                    skeletiKaader.CopySkeletonDataTo(skeletid);
                }
            }

            using (DrawingContext joonistuseKontekst = this.joonistusGrupp.Open())
            {
              joonistuseKontekst.DrawRectangle(Brushes.Black, null, new Rect(0.0,
              0.0, esitusLaius, esitusKorgus));

               if (skeletid.Length != 0)
               {
                    foreach (Skeleton skel in skeletid)
                    {
                        this.JoonistaLuudJaLiigesed(skel, joonistuseKontekst);
                    }
               }
              this.joonistusGrupp.ClipGeometry = new RectangleGeometry(new
              Rect(0.0, 0.0, esitusLaius, esitusKorgus));
            }
        }

        private void JoonistaLuudJaLiigesed(Skeleton skelett, DrawingContext
        joonistuseKontekst)
```

```
{
    this.JoonistaLuu(skelett,       joonistuseKontekst,       JointType.Head,
    JointType.ShoulderCenter);

    this.JoonistaLuu(skelett,  joonistuseKontekst,  JointType.ShoulderCenter,
    JointType.ShoulderLeft);

    this.JoonistaLuu(skelett,  joonistuseKontekst,  JointType.ShoulderCenter,
    JointType.ShoulderRight);

    this.JoonistaLuu(skelett,  joonistuseKontekst,  JointType.ShoulderCenter,
    JointType.Spine);

    this.JoonistaLuu(skelett,      joonistuseKontekst,      JointType.Spine,
    JointType.HipCenter);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.HipCenter,
    JointType.HipLeft);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.HipCenter,
    JointType.HipRight);


    this.JoonistaLuu(skelett,   joonistuseKontekst,   JointType.ShoulderLeft,
    JointType.ElbowLeft);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.ElbowLeft,
    JointType.WristLeft);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.WristLeft,
    JointType.HandLeft);


    this.JoonistaLuu(skelett,  joonistuseKontekst,  JointType.ShoulderRight,
    JointType.ElbowRight);

    this.JoonistaLuu(skelett,   joonistuseKontekst,   JointType.ElbowRight,
    JointType.WristRight);

    this.JoonistaLuu(skelett,   joonistuseKontekst,   JointType.WristRight,
    JointType.HandRight);


    this.JoonistaLuu(skelett,     joonistuseKontekst,     JointType.HipLeft,
    JointType.KneeLeft);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.KneeLeft,
    JointType.AnkleLeft);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.AnkleLeft,
    JointType.FootLeft);


    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.HipRight,
    JointType.KneeRight);

    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.KneeRight,
    JointType.AnkleRight);
```

```
    this.JoonistaLuu(skelett,    joonistuseKontekst,    JointType.AnkleRight,
    JointType.FootRight);


      foreach (Joint liiges in skelett.Joints)

      {

          Brush liigeseVarv = null;

          if (liiges.TrackingState == JointTrackingState.Tracked)

          {

              liigeseVarv = this.jalgitavaLiigeseVarv;

          }

          else if (liiges.TrackingState == JointTrackingState.Inferred)

          {

              liigeseVarv = this.jareldatavaLiigeseVarv;

          }

          if (liigeseVarv != null)

          {

            joonistuseKontekst.DrawEllipse(liigeseVarv,                 null,
            this.SkeletiPunktEkraanile(liiges.Position),       liigeseSuurus,
            liigeseSuurus);

          }

      }


  }


private void JoonistaLuu(Skeleton skelett, DrawingContext joonistuseKontekst,
JointType liigeseTyyp1, JointType liigeseTyyp2)

{

      Joint liiges1 = skelett.Joints[liigeseTyyp1];

      Joint liiges2 = skelett.Joints[liigeseTyyp2];


      Pen luuVarv = this.jareldatavaLuuVarv;

    joonistuseKontekst.DrawLine(luuVarv,
    this.SkeletiPunktEkraanile(liiges1.Position),
    this.SkeletiPunktEkraanile(liiges2.Position));

  }
```

```
        private Point SkeletiPunktEkraanile(SkeletonPoint skeletiPunkt)
        {
           DepthImagePoint                          sygavusPunkt                          =
           this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skeletiPunkt,
           DepthImageFormat.Resolution640x480Fps30);
            return new Point(sygavusPunkt.X, sygavusPunkt.Y);
        }

        private void sulgemine(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
        }
    }
}
```

8.5 Joonistamise rakenduse kood

MainWindow.xaml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;

namespace SygavusJoonistus
{
    public partial class MainWindow : Window
    {
       private KinectSensor sensor;
       private DepthImagePixel[] sygavusPikselid;
       const int varvimisePildiLaius = 640;
       const int varvimisePildiKorgus = 480;
       const int varvimisePildiBaitideSuurus = varvimisePildiKorgus *
       varvimisePildiLaius * 4;
       byte[] varvimisePildiBaidid = new byte[varvimisePildiBaitideSuurus];
       int varvimiseMaxKaugus = 1500;
       int varvimiseMinKaugus = 1000;

        public MainWindow()
        {
            InitializeComponent();
        }

        private void laadimine (object sender, RoutedEventArgs e)
        {
            this.sensor = KinectSensor.KinectSensors[0];
            if (null != this.sensor)
            {
                    this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x48
                    0Fps30);

                    this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640
                    x480Fps30);
```

```
            this.sygavusPikselid                    =                    new
            DepthImagePixel[this.sensor.DepthStream.FramePixelDataLength];

            this.sensor.AllFramesReady += this.SensorAllFramesReady;


            this.sensor.Start();

    }

}


private void sulgemine(object sender, System.ComponentModel.CancelEventArgs
e)

{

    if (null != this.sensor)

    {

        this.sensor.Stop();

    }

}


private void SensorAllFramesReady(object sender, AllFramesReadyEventArgs e)

{

    using (DepthImageFrame sygavusKaader = e.OpenDepthImageFrame())

    {

        if (sygavusKaader != null)

        {

            sygavusKaader.CopyDepthImagePixelDataTo(this.sygavusPikselid);

            for (int i = 0; i < this.sygavusPikselid.Length; ++i)

            {

                short sygavus = sygavusPikselid[i].Depth;

             if    (varvimiseMinKaugus    <    sygavus    &&    sygavus    <
             varvimiseMaxKaugus)

                {

                    varviPikselid(i * 4, 255, 0, 0, 100);

                }

            }

        }
```

```
            }

            using (ColorImageFrame v2rvipildiKaader = e.OpenColorImageFrame())

            {

                if (v2rvipildiKaader == null)

                {

                    return;

                }


                int sammupikkus = v2rvipildiKaader.Width * 4;

                byte[]pikselid = new byte[v2rvipildiKaader.PixelDataLength];

                v2rvipildiKaader.CopyPixelDataTo(pikselid);


                pilt2.Source      =      BitmapSource.Create(v2rvipildiKaader.Width,
                v2rvipildiKaader.Height,    96,    96,    PixelFormats.Bgr32,    null,
                pikselid, sammupikkus);

                pilt1.Source      =      BitmapSource.Create(v2rvipildiKaader.Width,
                v2rvipildiKaader.Height,    96,    96,    PixelFormats.Bgra32,    null,
                varvimisePildiBaidid, sammupikkus);

            }

        }


        private void varviPikselid(int asukoht, byte b, byte g, byte r, byte a)

        {

            varvimisePildiBaidid[asukoht] = b;

            asukoht++;

            varvimisePildiBaidid[asukoht] = g;

            asukoht++;

            varvimisePildiBaidid[asukoht] = r;

            asukoht++;

            varvimisePildiBaidid[asukoht] = a;

        }

    }

}
```

8.6 Videopildi ja skeletiandmete voo ühendamise kood

MainWindow.xaml.cs:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Microsoft.Kinect;


namespace SkeletoniPilt

{

    public partial class MainWindow : Window

    {

        private const float esitusLaius = 640.0f;

        private const float esitusKorgus = 480.0f;

        private const double liigeseSuurus = 3;

       private       readonly       Brush       jalgitavaLiigeseVarv       =       new
       SolidColorBrush(Color.FromArgb(255, 68, 192, 68));

        private readonly Brush jareldatavaLiigeseVarv = Brushes.Yellow;

        private readonly Pen jareldatavaLuuVarv = new Pen(Brushes.Gray, 1);

        private KinectSensor sensor;

        private DrawingGroup joonistusGrupp;

        private DrawingImage pildiAllikas;
```

```
        public MainWindow()
        {
            InitializeComponent();
        }

        private void laadimine (object sender, RoutedEventArgs e)
        {
            this.sensor = KinectSensor.KinectSensors[0];
            this.joonistusGrupp = new DrawingGroup();

            this.pildiAllikas = new DrawingImage(this.joonistusGrupp);
            pilt1.Source = this.pildiAllikas;

            if (null != this.sensor)
            {
this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
                this.sensor.SkeletonStream.Enable();
                this.sensor.AllFramesReady += this.sensor_AllFramesReady;

                this.sensor.Start();
            }
        }

        private void sensor_AllFramesReady(object sender, AllFramesReadyEventArgs
e)
        {
            Skeleton[] skeletid = new Skeleton[0];
            using (SkeletonFrame skeletiKaader = e.OpenSkeletonFrame())
            {
                if (skeletiKaader != null)
```

```
    {
        skeletid = new Skeleton[skeletiKaader.SkeletonArrayLength];
        skeletiKaader.CopySkeletonDataTo(skeletid);
    }
}

using (DrawingContext joonistuseKontekst = this.joonistusGrupp.Open())
{
  joonistuseKontekst.DrawRectangle(Brushes.Transparent,    null,    new
  Rect(0.0, 0.0, esitusLaius, esitusKorgus));

    if (skeletid.Length != 0)
    {
        foreach (Skeleton skel in skeletid)
        {
            this.JoonistaLuudJaLiigesed(skel, joonistuseKontekst);
        }
    }
  this.joonistusGrupp.ClipGeometry    =    new    RectangleGeometry(new
  Rect(0.0, 0.0, esitusLaius, esitusKorgus));
}
using (ColorImageFrame v2rvipildiKaader = e.OpenColorImageFrame())
{
    if (v2rvipildiKaader != null)
    {
      byte[] pikselid = new byte[v2rvipildiKaader.PixelDataLength];
      v2rvipildiKaader.CopyPixelDataTo(pikselid);

      int sammupikkus = v2rvipildiKaader.Width * 4;
      pilt2.Source    =    BitmapSource.Create(v2rvipildiKaader.Width,
      v2rvipildiKaader.Height,96,    96,    PixelFormats.Bgr32,    null,
      pikselid, sammupikkus);
    }
}
```

```
}

private void JoonistaLuudJaLiigesed(Skeleton skelett, DrawingContext
joonistuseKontekst)

{

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.Head,
    JointType.ShoulderCenter);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
    JointType.ShoulderLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
    JointType.ShoulderRight);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderCenter,
    JointType.Spine);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.Spine,
    JointType.HipCenter);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipCenter,
    JointType.HipLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipCenter,
    JointType.HipRight);


    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderLeft,
    JointType.ElbowLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ElbowLeft,
    JointType.WristLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.WristLeft,
    JointType.HandLeft);


    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ShoulderRight,
    JointType.ElbowRight);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.ElbowRight,
    JointType.WristRight);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.WristRight,
    JointType.HandRight);


    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipLeft,
    JointType.KneeLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.KneeLeft,
    JointType.AnkleLeft);

    this.JoonistaLuu(skelett, joonistuseKontekst, JointType.AnkleLeft,
    JointType.FootLeft);
```

```
        this.JoonistaLuu(skelett, joonistuseKontekst, JointType.HipRight,
        JointType.KneeRight);

        this.JoonistaLuu(skelett, joonistuseKontekst, JointType.KneeRight,
        JointType.AnkleRight);

        this.JoonistaLuu(skelett, joonistuseKontekst, JointType.AnkleRight,
        JointType.FootRight);


         foreach (Joint liiges in skelett.Joints)

         {

             Brush liigeseVarv = null;

             if (liiges.TrackingState == JointTrackingState.Tracked)

             {

                 liigeseVarv = this.jalgitavaLiigeseVarv;

             }

             else if (liiges.TrackingState == JointTrackingState.Inferred)

             {

                 liigeseVarv = this.jareldatavaLiigeseVarv;

             }

             if (liigeseVarv != null)

             {

               joonistuseKontekst.DrawEllipse(liigeseVarv, null,
               this.SkeletiPunktEkraanile(liiges.Position), liigeseSuurus,
               liigeseSuurus);

             }

         }

     }


    private void JoonistaLuu(Skeleton skelett, DrawingContext joonistuseKontekst,
    JointType liigeseTyyp1, JointType liigeseTyyp2)

     {

         Joint liiges1 = skelett.Joints[liigeseTyyp1];

         Joint liiges2 = skelett.Joints[liigeseTyyp2];


         Pen luuVarv = this.jareldatavaLuuVarv;
```

```
            joonistuseKontekst.DrawLine(luuVarv,
            this.SkeletiPunktEkraanile(liiges1.Position),
            this.SkeletiPunktEkraanile(liiges2.Position));

        }



        private Point SkeletiPunktEkraanile(SkeletonPoint skeletiPunkt)

        {

            DepthImagePoint                          sygavusPunkt                          =
            this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skeletiPunkt,
            DepthImageFormat.Resolution640x480Fps30);

             return new Point(sygavusPunkt.X, sygavusPunkt.Y);

        }

        private void sulgemine(object sender, System.ComponentModel.CancelEventArgs
e)

        {

            if (null != this.sensor)

            {

                this.sensor.Stop();

            }

        }

    }

}
```