Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Grega Gašperšič

Zajem gibanja s senzorjem Kinect

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Matija Marolt

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [**?**]

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

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

Št. naloge: 00030/2013
Datum: 15.04.2013

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

Kandidat: **GREGA GAŠPERŠIČ**

Naslov: **ZAJEM GIBANJA S SENZORJEM KINECT**
USING KINECT FOR MOTION CAPTURE

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V diplomski nalogi raziščite načine za uporabo senzorja Kinect za zajem gibanja. Implementirajte sistem, ki naj podpira zajem gibanja s senzorjem Kinect, vizualizacijo in interpolacijo gibanja ter uvažanje zajetih podatkov v okolji Maya in MotionBuilder za nadaljnjo obdelavo.

Mentor:

doc. dr. Matija Marolt

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 1

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Grega Gašperšič, z vpisno številko **63080439**, sem avtor diplomskega dela z naslovom:

Zajem gibanja s senzorjem Kinect

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matija Marolta
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 4. septembra 2013 Podpis avtorja:

Kazalo

Povzetek

Cilj diplomske naloge je izgradnja nizko proračunskega sistema za zajemanje gibanje. To vključuje predvajanje in interpolacijo gibanja ter tudi uvažanje zajetih podatkov v okolje Maya in MotionBuilder. Da bi lahko to naredili, moramo biti zmožni zajeti podatke o gibanju in jih pretvoriti v obliko, ki je razumljiva tako za človeka kot za računalnik.

Podatki o gibanju so zajeti s pomočjo Microsoftovega senzorja za zaznavanje gibanja Kinect. Sistem temelji na RGB kameri in senzorju 3D globine, sestavljenega iz laserskega projektorja IR in senzorja IR. Za potrebe zamišljenega sistema, morajo biti podatki čim bolj natančni. V ta namen izdelamo tudi vizualizacijo kompleksnega 3D modela, ki služi kot pomoč pri interpretaciji natančnosti zajetih podatkov.

Naslednji korak je pretvarjanje podatkov o gibanje v človeku razumljivo obliko in shranjevanje novih podatkov, možnih za kasnejšo uporabo. Kot del vizualizacije tudi implementiramo ponovno predvajanje in interpolacijo novih podatkov.

Končni korak je implementacija programa za uvažanje podatkov v Autodesk Mayo in pretvorba podatkov v BVH format, ki je primeren za uvažanje v program Autodesk MotionBuilder.

Ključne besede:

zajemanje gibanja, senzor Microsoft Kinect, zajem podatkov, vizualizacija 3D modela, interpolacija podatkov, obdelava podatkov

Abstract

The goal of this thesis is to build a low budget system for motion capture. This includes replaying and interpolation of motion, and importing the captured data into the Maya and MotionBuilder environment. In order to do this, we must be able to capture motion data and convert it into a for that is comprehensible to humans and computers.

Motion data is captured with the help of Microsofts motion detection sensor Kinect. The system is based on a RGB camera and a 3D depth sensor, consisting of a IR laser projector and a IR sensor. For the needs of the projected system, the data must be as accurate as possible. To this end, a visualization of a 3D complex model has been created, which serves as an aid in interpreting the accuracy of the captured data.

The next step is to convert the movement data into human readable form and store the new data, for future use. As part of the visualization we also implement the replaying and interpolation of the new data.

The final step is to implement a program to import data into Autodesk Maya and converting the data into a BVH file format that is suitable for importing into Autodesk MotionBuilder.

Key words:

motion capture, Microsoft Kinect sensor, data capture, 3D model visualization, data interpolation, data processing

Poglavje 1

Uvod

1.1 Motivacija

Zajemanje gibanja, sledenje gibanju ali MOCAP so izrazi, uporabljeni za opis procesa snemanja gibanja enega ali več oseb ali objektov. Na sejah zajemanja gibanja, se gibanje enega ali več osebkov beležijo samo podatki o položaju gibanja, ne pa njegov videz. Ta animacija se pogosto preslika v skelet 3D modela. Uporablja se v vojski, zabavi, športu, medicini ter za potrditev računalniškega vida in v robotiki. Kljub njegovemu širokemu spektru uporabnosti še ne obstaja natančen sistem za zajemanje gibanja, s katerim bi se dalo zajemati, snemati, interpolirati, prikazovati in obdelovati zajete podatke za nizko ceno. Zato bi želeli izdelati takšen sistem, ki ne bi naredil veliko napak.

Za izdelavo takšnega sistema potrebujemo podatke o gibanju. Na tržišču obstaja veliko sistemov za zajemanje gibanja, ki zajemajo gibanje ene ali več oseb. Nekateri delujejo na principu računalniškega vida, ki s pomočjo videokamere zajemajo podatke o značkah na osebku. Drugi za svoje delovanje ne potrebujejo značk. Delujejo na principu algoritmov iz računalniškega vida, ki s pomočjo analize optičnih tokov videokamer prepoznajo človeško obliko, ter jo razbijejo na sestavne dele, primerne za zajemanje gibanja. Za zajemanje gibanja smo izbrali sistem brez značk. Za napravo smo izbrali senzor

Kinect, ki sporoča translacijo posameznih sklepov na človeškem telesu, ki jih moramo pravilno interpretirati, da dobimo nepopačene podatke o poziciji sklepov v 3D prostoru.

1.2 Struktura diplomskega dela

Poleg uvoda in zaključka delo vsebuje še tri poglavja. V prvem delu predstavimo sisteme za zajemanje gibanja in tehnologije, ki jih uporablja senzor Kinect. Osredotočimo se tudi na to, kako pridobimo natančne podatke za nadaljnje delo in kakšna je njihova oblika. Prav tako opišemo, kako smo vizualizirali podatke s pomočjo zunanjih knjižic, ter jih tudi pripravili za kasnejšo obdelavo. V drugem delu predstavimo definicijo interpolacije podatkov. Opišemo tudi implementacijo, ki se uporablja za interpolacijo med gibanjem. V tretjem in zadnjem delu predstavimo dve od mnogih orodij, ki se uporabljata za obdelavo zajetih podatkov. Tukaj opišemo implementacijo programa, ki se uporablja za uvažanje podatkov v orodje, ter pretvobo podatkov iz enega formata v drug. Celotno diplomsko delo zaključimo s prednostmi in omejitvami implementiranega sistema ter opišemo možnosti za nadaljnji razvoj.

Poglavje 2

Zajem in predstavitev podatkov

2.1 Zajem podatkov

Podatke smo zajemali s pomočjo Microsoftovega senzorja za zaznavanje gibanja Kinect, ki deluje na podlagi RGB kamere in senzorja 3D globine. Kinect za vsak sklep sporoča podatke o X, Y, Z koordinatnih oseh in tudi o dodatnenm parametru W, ki pove, kako dobro so bili podatki zajeti. V nadaljevanju bomo uporabljali samo podatke o X, Y in Z koordinatnih oseh, ki se bodo kasneje uporabljale pri snemanju gibanja in izračunavanju ter izrisovanju telesa. Parametra W ne bomo uporabljali, ker bomo dobili popačene podatke, ki pri nadaljnjem delu ne bodo koristili.

Predstavili bomo tudi sisteme za zajemanje gibanja in tehnologije, ki jih uporablja senzor Kinect. Prav tako bomo predstavili, kako podatke z njim zajeti in iz njih izluščiti status zajetega sklepa osebe, ter njegove koordinate.

2.1.1 Sistemi za zajemanje gibanja

Zajemanje gibanja, sledenje gibanju ali MOCAP so izrazi, uporabljeni za opis procesa snemanja gibanja enega ali več oseb ali objektov. Proces snemanja se uporablja se v vojski, filmih, televiziji, virtualnih okoljih, zabavi, športu, robotiki, komunikaciji človekstroj, medicini in za potrditev računalniškega vida [3].

Na sejah zajemanja gibanja, se gibanje enega ali več osebkov beležijo samo podatki o položaju gibanja ne pa njegov videz. Ta animacija se pogosto preslika v skelet 3D modela. Postopek zajemanja gibanja se lahko primerja s tehniko, imenovano rotoscope. To je animacijska tehnika, kjer je bil posnetek projeciran na tako imenovan zaslon, kjer je animator sličico po sličico prerisoval obrise. Tehniko je izumil Max Fleischer, ki jo je uporabljal v svoji seriji Out of the Inkwell.

Sistem za zajemanje gibanja se je začel razvijati kot orodje za analizo fotografij v letih 1970. To se je kasneje razširilo v vse panoge industrije. V današnjem svetu ločimo naslednje načine zajemanja gibanja:

- Elektromagnetno – deluje s pomočjo oddajnikov, ki ustvarijo magnetno polje in senzorjev, ki ugotovijo lokacijo in orientacijo v prostoru. Podatki, ki jih dobimo s pomočjo tega sistema, so zelo natančni, saj je sistem sam neobčutljiv na zakrivanja in omogoča pogled na šest prostorskih stopenj. Zaradi magnetnega polja je domet sistema omejen in prav tako občutljiv na kovinske predmete [2].

- Elektromehanično – temelji na ekso-skeletu s potenciometri. Največja prednost tega sistema je, da proizvede rezultate v realnem času z nizko ceno in visoko natančnostjo, saj ni zakrivanja. Vendar sam sistem nima globalnega položaja. Prav tako pa je zaradi težkega skeleta omejena možnost zajetih gibov [2].

- Žiroskopi – sistem temelji na pridobivanju podatkov s pomočjo žiroskopov v obleki. Podatki, pridobljeni s tem sistemom, so izredno natančni, saj ne more priti do zakrivanja. Vendar je zaradi pomanjkanja globalnega položaja te podatke zelo težko razbrati [1].

- Optično z značkami – trenutno najpopularnejša metoda, deluje s pomočjo obleke, na kateri so narisane ali pripete značke. Te značke so lahko aktivne ali pasivne. Prav tako pa za delovanje tega sistema potrebujemo kamero, ki je kalibrirana tako, da vidi značke. Ker sistem ne potrebuje

težje strojne opreme, je zelo poceni in prilagodljiv. Njegovi slabosti sta omejena ločljivost kamer in omejeno število zajemanja gibanja objektov [1].

- Optično brez značk – deluje s pomočjo natančno kalibriranih kamer. Poleg kalibriranih kamer lahko sistem deluje s pomočjo kalibriranih igralcev. Sistem je izredno prilagodljiv, poceni in tudi zajame deformacije površine. Vendar ima zaradi ločljivosti kamere omejen domet in natančnost [1].Primer prve široke komercialne platforme je Kinect. Podrobnejši opis je v poglavju 2.1.2.

Slika 2.1: Prikazan je igralec Andy Serkis v elektromagnetni obleki [1].

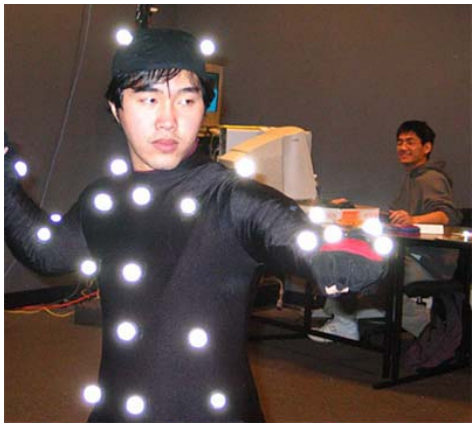
Slika 2.2: Prikazan je eksoskelet s potenciometri [1].

Sistemi za zajemanje gibanja ponujajo številne prednosti od tradicionalnih tehnik računalniške animacije 3D modelov. Rezultate dobimo veliko hitreje kot pri tradicionalnih tehnikah animacije, lahko celo v realnem času. V zabavnih aplikacijah lahko s tem zmanjšamo stroške glavnih okvirjev, na

Slika 2.3: Prikazan je sistem z žiroskopi [1].

čemur temelji animacija. Prav tako je število animacijskih podatkov, ki se lahko proizvedejo v danem času, zelo veliko v primerjavi s tradicionalnimi tehnikami animacije. To prispeva k učinkovitosti stroškov in krajšanju rokov proizvoda. Zaradi tega je tak sistem potencialen za brezplačno programsko opremo in tretjo osebne rešitve. Prav tako se obseg dela ne spreminja glede na kompleksnost ali dolžino delovanja, tako kot pri tradicionalni tehniki. To omogoča opravljanje številnih testov z različnimi slogi, pri čemur smo omejeni samo s talenti igralca. Omogoča tudi enostavno in natančno poustvarjanje zapletenega realističnega gibanja in realne fizične interakcije, kot so sekundarni gibi, masa ter izmenjava sil.

Tako kot vsi sistemi imajo tudi ti določene slabosti. Za pridobivanje in obdelavo podatkov je potrebna posebna strojna in programska oprema. Prav tako so stroški opreme in osebja lahko potencialno previsoki za malo produkcijo. Kljub visoki zanesljivosti opreme se lahko pojavijo težave. Če se pojavijo, je lažje ponovno posneti celotno sceno, kot pa poskušati manipulirati zajete podatke. Le nekaj sistemov omogočajo predvajanje podatkov v realnem času in s tem omogoča možnost ponovnega snemanja. Med zajeman-



Slika 2.4: Prikazano je optično zajemanje gibanja z značkami [12].

Slika 2.5: "Performance Capture from Sparse Multi-view Video", de Aguilar et al., 2008 [1].

jem moramo biti pazljivi na razlike med modelom in človeškim izvajalcem. Če ima računalniški model različne razsežnosti, lahko pride do preseka teles, npr. če ima računalniški model velike roke in človeški izvajalec ni previden s svojimi telesnimi gibi, lahko pride do preseka teles med velikimi rokami in ostalim telesom. Pri snemanju smo prav tako omejeni na tisto, kar se lahko izvede v okviru obsega zajemanja in na gibanje, ki sledi zakonom fizike. Ostalo gibanje je nemogoče zajeti. Kljub veliki količini podatkov je po koncu zajemanja še vedno potrebno uporabiti tradicionalne tehnike animacije, kot so poudarek na predvidevanju in sledenju gibanja, sekundarnega gibanja ali spreminjanja oblike osebka[9].

2.1.2 Tehnologija senzorja Kinect

2.1.2.1 Strojna oprema

Strojni del senzorja Kinecta je prispevalo izraelsko podjetje PrimerSense. Tako imenovana intervalna kamera s pomočjo IR projektorja in kamere ter namenskega procesorja spremlja gibanje v 3D prostoru. To jim omogoča

sistem, imenovan svetlobno kodiranje, ki posnete slike sestavi v 3D sliko.

RGB kamera vsebuje 1.3 megapiksel SOC CMOS Digital Image senzor s 1280 x 1024 aktivnih slikovnih točk. Privzeti podatkovni zajem poteka v resoluciji 640 x 480 slikovnih točk s hitrostjo 24 do 30 Hz, vendar lahko tudi z resolucijo 1280 x 1240 slikovnih točk, pri tem hitrost zajema pade na 12 do 15 Hz. Uporablja se tudi Bayernov RGB filter, kar je polje barvnih filtrov, postavljenih na kvadratno mrežo svetlobnih senzorjev in omogoča vračanje neobdelanih podatkov z resolucijo 1280 x 1024 slikovnih točk. Procesna logika jih nato stisne in pretvori v RGB obliko. S tem nam omogoči prenos podatkov s hitrostjo 30 Hz.

Slika 2.6: Prikazani so vsi senzorji, ki jih vsebuje Kinect.

.

IR senzor globine je sestavljen iz laserskega projektorja IR in senzorja IR, ki sta povezana v stereo načinu. S tem simulirata človekov vid, ki se uporablja za izdelavo 3D slike. Pri tem gre za postavitev projektorja in kamere pod določenim kotom, kjer se s pomočjo triangulacije izračuna oddaljenost objekta od senzorja. Laserski projektor IR svetlobe, ki se nahaja na sprednji strani Kinecta, uporablja tehnologijo strukturirane svetlobe. To pomeni, da ima projecirana svetloba nek vzorec, s čimer si Kinect pomaga pri izračunih

oddaljenosti točk od naprave. Pri izračunih upošteva tudi sosedne točke in njihovo lego v prostoru. Senzor IR svetlobe zajema sliko v črno-beli barvi, z natančnostjo 11 bitov. Zajem slike je možen z naslednji resolucijami:

- 80 x 66 slikovnih točk,
- 320 x 240 slikovnih točk,
- 640 x 480 slikovnih točk.

2.1.2.2 Programska oprema

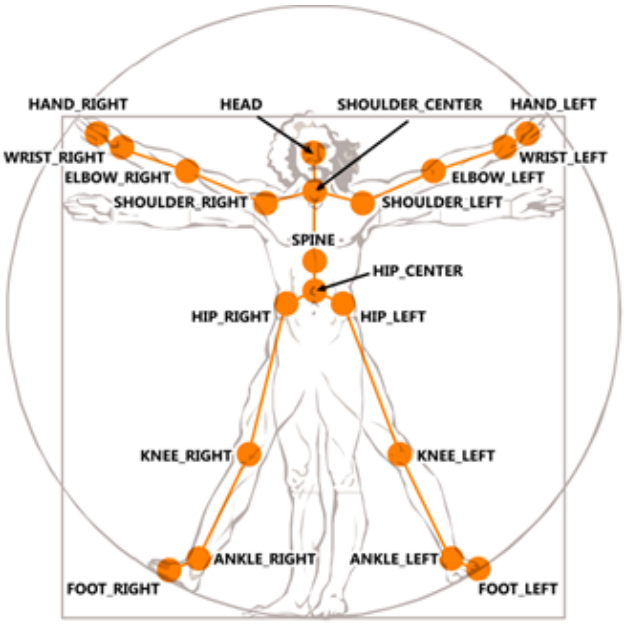
Kinect temelji na tehnologiji programske opreme iz podjetja Rare, ki je hčerinsko podjetje Microsoft Game Studios. V primerjavi z nekaterimi drugimi sistemi, ki imajo značke ali senzorje, ki so direktno pritrjeni na sklepe, se Kinect temu v celoti izogne s pomočjo računalniškega vida. Ti posebni algoritmi so oblikovani tako, da sistem analizira več vhodnih optičnih tokov in prepozna človeško obliko, ter jo razbije na sestavne dele, primerne za sledenje, in jih pošlje računalniku preko USB vhoda. V našem primeru so to človeški sklepi.

Algoritem temelji na lokalni analizi slikovne točke zajete slike. Tradicionalno razpoznavanje vzorcev deluje tako, da določeno odločitveno strukturo naučimo razpoznavanja s pomočjo velikega števila primerov. Da bi to delovalo, bi klasifikatorju sistema morali predstaviti veliko meritev določenih značilnic, s katerimi upamo, da vsebujejo podatke, potrebne za prepoznavo predmeta ali osebe. Značilnice, ki so bile uporabljene v senzorju Kinect, so preproste in temeljijo na formuli 2.1, kjer je (u, v) vektor premika in d(c) globina oziroma oddaljenost od senzorja Kinect za slikovno točko na poziciji x.

$$f = d(x + \frac{u}{d(x)}) - d(x + \frac{v}{d(x)}) \tag{2.1}$$

Enačba 2.1 meri razliko globine od dveh točk odmika do ciljne slikovne točke v (u, v). Edini zaplet je, da se odmik pomanjša z razdaljo. To popravimo z deljenjem z d(x), s tem postane odmik globine neodvisen.

Naslednji korak je usposobiti vrsto razvrščevalca, imenovanega odločitveni gozd, ki je zbirka odločitvenih dreves. Vsako odločitveno drevo je usposobljeno, da prepozna določeno značilnico na označenih globinskih slikah. Odločitvena drevesa so spremenjena, dokler pravilno ne razvrstijo posameznih delov teles na vseh testnih podatkih. Rezultat je klasifikator, ki dodeli verjetnost določene slikovne točke, da se nahaja na določenem delu telesa in izbere področja z največjo verjetnostjo za vsak del telesa [7].



Slika 2.7: Prikazana je hierarhija človekovega telesa, kot jo vidi senzor Kinect.

.

Zaradi same zgradbe klasifikatorja lahko telo predstavimo v obliki hierarhije. Za zajem podatkov je najpomembnejši del telesa koren osebe. Zaradi translacije tega se namreč spremeni pozicija celotnega telesa. Ostali sklepi na telesu so potem odvisni od njega in tudi med seboj. Najvišji v tej hierarhiji je koren v našem primeru sredina kolkov (HipCenter). Temu so podrejeni trije deli spodnji del hrbtenice (SPINE) ter levi (HIP_LEFT) in desni kolk (HIP_RIGHT). Levemu kolku so podrejeni levo koleno (KNEE_LEFT), gleženj (ANKLE_LEFT) in noga (FOOT_LEFT),

desnemu kolku pa desno koleno (KNEE_RIGHT), gleženj (ANKLE_RIGHT) in noga (FOOT_RIGHT). Spodnjemu delu hrbtenice je podrejen zgornji del hrbta (SHOULDER_CENTER), kateremu so podrejeni trije deli, glava (HEAD), leva (SHOULDER_LEFT) in desna rama (SHOULDER_RIGHT). Obema ramenoma so podrejeni ostali sklepi rok. Na sliki 3.1 je predstavljena natančna hierarhija, kakor jo vidi senzor Kinect.

2.1.3 Implementacija zajemanja podatkov

Zajemanje podatkov je napisano v programskem jeziku Microsoft C# s podporo Microsoft XNA Game Studio 4.0. Za povezavo s Kinectom je potrebna tudi knjižica Microsoft Research, ki vsebuje Kinect for Windows SDK beta. Beta SDK nudi prefinjeno opremo knjižic in orodij za pomoč pri razvoju. Primer takega orodja je NUI API, ki zagotavlja sredstva za spreminjanje nastavitev Kinectovih senzorjev in dostop do slikovnih podatkov, zajetih z njim.

Tok podatkov je zajet kot zaporedje sličic. Ob inicializaciji NUI uporabnik opredeli optični tok, ki ga želi uporabiti. Uporabnik ima možnost do naslednjih vrst slikovnih podatkov:

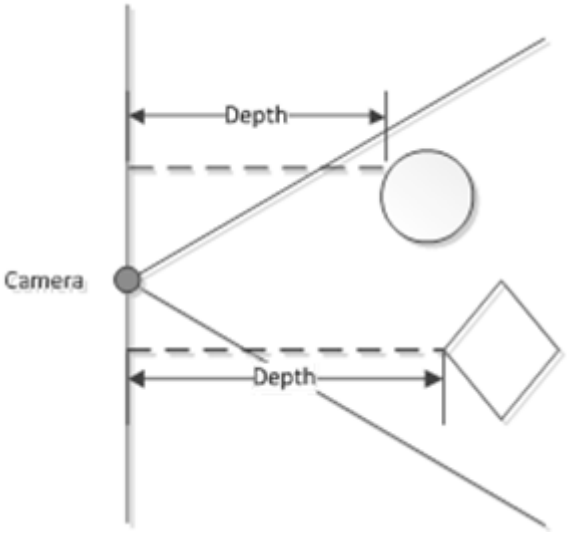
- barvni podatki,
- globinski podatki,
- podatki o segmentacije osebe.

Poleg slikovnih podatkov lahko s pomočjo NUI Skeleton API dobi aktivne informacije o lokaciji do dveh oseb, ki stojijo pred senzorjem Kinect, s podrobnimi informacijami o položaju in usmerjenosti.

V naši implementaciji smo uporabili NUI skeleton API, s pomočjo katerega smo zajeli podatke o gibanju ene osebe, jih vizualizirali in tudi obdelali. Prav tako smo za lažji prikaz uspešnosti zajemanja podatkov o gibanju uporabili globinske podatke.

2.1.3.1 Zajemanje globinskih podatkov

Globinski podatkovni tok nam zagotavlja slike, v katerih je vsaka slikovna točka predstavljena kot razdalja med ravnino kamere do najbližjega objekta v milimetrih, kot je prikazano na sliki 2.8. Podatkovni tokovi so lahko predstavljeni s tremi resolucijami. Naša aplikacija bo uporabljala resolucijo 80 x 60 slikovnih točk, saj bomo informacijo o globinskih podatkih uporabljali samo za predstavitev uspešnosti zajemanja podatkov o skeletu. Prav tako bo uporabljala funkcijo indeksa osebka, s pomočjo katere naša aplikacija zazna obliko in mesto nahajanja določenega osebka.



Slika 2.8: Prikazana je razdalja med ravnino kamere do najbližjega objekta.

Podatki vsake slikovne točke so veliki 16 bitov. Njihova oblika je odvisna od tega, ali je ob zagonu NUI opredeljeno zajemanje samo globine ali globine in indeksa osebkov. Iz tega sledita naslednji dve obliki podatkov:

- Za globinske podatke vsebujejo biti od 0 do 11 za vsako slikovno točko podatke o globini, ostali 4 biti so neizkoriščeni.
- Za globinske podatke in podatke o indeksu osebe vsebujejo biti od 0 do 2 za vsako slikovno točko podatke o indeksu oseb. Kar nam

ponudi možnost dobivanja podatkov o 6 osebah. Preostali biti vsebujejo globinske podatke.

Podatki bodo aplikaciji posredovani preko dogodkov, ki omogočajo večjo prožnost in natančnost. Torej, če so podatki uspešno zajeti, bo predstavitev osebe spremenila barvo. Drugače barva ostane enaka kot ozadje naše aplikacije. Nova barva osebe je odvisna tudi od indeksa osebkov, saj smo za vsak indeks posebej definirali drugačno barvo.

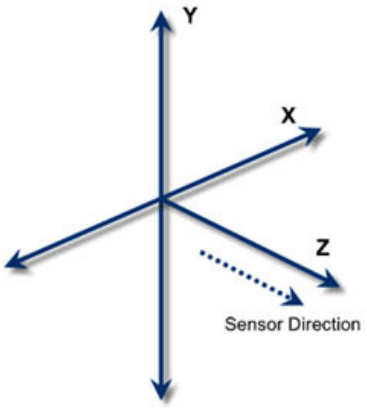
Globinski podatki z vrednostjo 0 pomenijo, da ni nobenih globinskih podatkov na voljo na tem položaju. Bodisi ker so predmeti preveč blizu kamere ali predaleč stran od nje. Zaradi neuporabnosti takšni globinski podatki v naši aplikaciji niso prikazani.

2.1.3.2 Zajemanje podatkov skeleta

Pogon za sledenje skeletov omogoča sledenje za enega ali dve osebi. Prav tako omogoča aktivno in pasivno sledenje skeletu. Aktivno sledenje je omejeno na dve osebi in zagotavlja popolne informacije o skeletu, medtem ko je pasivno sledenje omejeno na šest oseb in zagotavlja samo delne informacije o skeletu. Privzeto je, da se prvima dvema zaznanima skeletoma aktivno sledi, ostalim pa pasivno. Zaradi velikih količin podatkov in lažje razumljivosti smo se odločili za aktivno sledenje samo enega skeleta. S tem smo tudi pridobili na manjši obremenitvi senzorja in tudi večji natančnosti podatkov.

Podatki o skeletih se posredujejo aplikaciji preko dogodkov kot niz točk, ki se imenujejo položaji skeleta. Te sestavljajo skelet, kot je prikazan na sliki 3.1. Pozicije sklepov skeleta so izražene v X, Y in Z koordinatah. Za razliko od koordinat globinskega prostora so te koordinate v metrih. X, Y in Z osi so glavne osi senzorja Kinect. Te sestavljajo desnosučni koordinatni sistem in postavijo senzor v središče, kot je prikazano na sliki 2.9.

Ko senzor Kinect zazna gibanje, pošlje informacije računalniku preko USB vhoda. Ta sproži dogodek. Aplikacija ga zazna in podatke o skeletu začasno shrani. Nato naredi obrat podatkov v Z smeri koordinatnega sistema. Če bi podatke pustili takšne kot so, bi bila naša predstavitev proti nam obrnjena s



Slika 2.9: Prikazan je koordinatni sistem senzorja Kinect.

.

hrbtom. Kar pomeni, da ne bi morali opazovati gibanja oziroma bi morali v vizualizaciji narediti dodatne premike z navidezno kamero, s čimur bi podatke lahko videli. Po obdelavi vseh podatkov se le-ti prenesejo neposredno na 3D predstavitev skeleta. Ta je vizualizirana s pomočjo paketa DigitalRune Engine in podrobneje opisana v poglavju 2.2.3.

2.2 Predstavitev podatkov

Za namen boljše predstavitve smo v sklopu diplomske naloge izdelali programsko orodje MotionCapture za interakcijo in vizualizacijo zajetih podatkov. MotionCapture s Kinectom komunicira preko USB vhoda. Vsebuje možnosti snemanja gibanja v formatu XML, ki ga podpira veliko programov, ki omogočajo predvajanje in spreminjanje 3D grafike. Prav tako omogoča vizualizacijo gibanja v realnem času, predvajanje zajetega gibanja in interpoliranje med že obstoječim gibanjem. Motivacija za izdelavo programskega orodja temelji predvsem na vizualizaciji gibanja, kar nam omogoča veliko boljši pregled velike količine časovno in prostorsko odvisnih podatkov, pridobljenih z zajemanjem gibanja. Vizualni pregled je veliko bolj intuitiven in hkrati omogoča lažjo detekcijo napak ter s tem tudi ponuja možnost, da zavržemo posnetek zaradi nepravilnosti zajetih podatkov.

2.2.1 Uporabljene rešitve

Vizualizacija je napisana v programskem jeziku Microsoft C# s podporo Microsoft XNA Game Studio 4.0. Pri izdelavi vizualizacije smo uporabili knjižice Animation, Animation.Content.Pipeline, Game, Geometry, Mathematics, Mathematics.Content.Pipeline in Physics iz paketa DigitalRune Engine(`http://www.digitalrune.com`). Omogočajo ustvarjanje modelov z visoko stopnjo kompleksnosti, ki vsebujejo sklepe, kosti in jih lahko nadzorujemo z marionetnim podobnim pristopom. S tem se izognemo, da bi med gibanjem modela prišlo do izgube ene prostorske stopnje, imenovane kardanska zapora.

Tako imenovan marionetni pristop deluje po principu, da 3D model spremenimo v lutko iz cunj, na kateri lahko simuliramo pravila fizike. To lutko iz cunj moramo ustrezno kalibrirati, če želimo dobiti lepo vizualizacijo in ji dodati ciljne pozicije na sklepih njenega skeleta. Te pozicije s pomočjo šibkih omejitev premikamo s podatki skeleta, zajetih s Kinectom. Ta pristop je bolj zapleten, saj je treba za vsak 3D model ustvariti primerno tako imenovano lutko iz cunj, za kar je potrebno veliko testiranja z različnimi parametri.

2.2.2 Gradnja modela

V izogibanju težav, kot so izguba prostorske stopnje ali pretvarjanje podatkov o poziciji v rotacijske podatke, ki bi potem direktno manipulirali že obstoječe sklepe modelovega skeleta, uporabimo marionetni pristop. Zaradi tega smo morali naš model ponovno zgraditi. Pri tem smo si bomo pomagali s paketom DigitalRune Engine.

Ker je predstavitev skeleta ena izmed ključnih stvari pri predvajanju gibanja, moramo biti pri vseh naslednjih korakih zelo natančni, saj vsaka nenatančnost pomeni nepravilno gibanje modela.

1. Uvozimo model in določimo toga telesa modela (roke, noge, glava itd.) in njihov odmik. Toga telesa nastavimo tako, da okoli njih definiramo škatle ali kakšen drugi primitivni objekt. Parametre teles in odmikov

nastavljamo, dokler se toga telesa tesno ne prilegajo in smo zadovoljni z rezultati. Na koncu te faze bi morali imeti določena vsa toga telesa našega modela, ki se morajo pravilno premikati z začetno animacijo, ki jo vsak model vsebuje. Primer lahko vidimo na sliki 2.10.

2. Nastavimo maso posameznih togih teles. To je izredno pomembno, ker imajo motorji fizike veliko težavo s prevelikimi razlikami mase. Idealno bi bilo nastaviti parametre tako, da je masa najtežjega dinamičnega objekta desetkrat težja od mase najlažjega dinamičnega objekta. Če ne bomo izrecno nastavili mase togih teles, lahko motor fizike izračuna maso samodejno glede na obliko telesa. To ni optimalno, saj imata dva objekta lahko isto obliko, ampak različne velikosti in s tem tudi različno maso, npr. majhna škatla za roko modela bo veliko lažja od škatle zgornjega dela trupa.

3. V tem koraku bomo dodali sklepe. Sklepe dodajamo med posameznimi togimi telesi v smeri od konca okončin proti sredini kolkov.

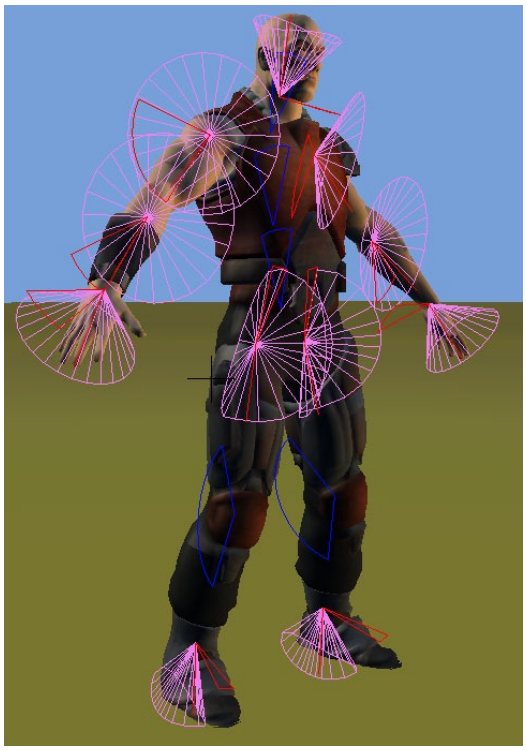
4. Na posameznih sklepih modela dodamo motorje, s pomočjo katerih se bo model premikal. Motorjem moramo nastaviti parameter dušenja in vzmetenja motorja, s pomočjo katerih bo fizična simulacija izračunala položaje gibanja.

5. Sedaj dodamo omejitve na sklepe. Na posameznem sklepu moramo dodati zgornjo in spodnjo mejo rotacije. Prav tako moramo nekaterim sklepom kot sredini hrbtenice, sredini kolkov in kolkoma definirati zgornjo in spodnjo mejo kota, pod katerim se lahko premika v posamezno smer koordinatnega sistema. Če v temu koraku napačno nastavimo parametre, se bo model gibal zelo nerealistično. Ta korak zahteva veliko preizkušanja in potrpljenja. Primer dobro definiranih omejitev lahko vidimo na sliki 2.10.

6. Kot zadnji korak v gradnji v simulaciji onemogočimo zaznavanje trkov med posameznimi togimi telesi našega modela. To naredimo zaradi

tega, da ne bi prišlo do čudnih gibov med trki dveh togih teles. S tem smo gibanje našega modela omejili samo na ključne točke [6].

Slika 2.10: Prikazana so toga telesa, ki smo jih definirali na modelu.



Slika 2.11: Prikazane so omejitve sklepov na modelu.

Z natančno nastavljenimi parametri na naši lutki iz cunj smo dobili 3D model, ki se bo uporabljal za vizualizacijo gibanja. Če je karkoli narobe s temi parametri, bomo dobili popačeno vizualizacijo, ki nam pri nadaljnjem delu ne bo koristila.

2.2.3 Vizualizacija

Naloženo ali zajeto gibanje se prikaže kot 3D model v srednjem delu okna, prikazanega na sliki 2.12. Model je animiran v interaktivnem virtualnem okolju in se giblje v realnem času, bodisi pod nadzorom osebe ali shranjenega

scenarija. Informacije o sklepih dobiva aplikacija preko dogodkov. Te so nato posredovane ciljnim pozicijam na posameznem sklepu, na katere je 3D model vpet. S pomočjo teh ciljnih pozicij se nato izvrši premikanje posameznega dela telesa 3D modela. Razlika med točkami, prejetimi z dogodki in ciljnimi pozicijami, je minimalna. To nakazuje na pravilnost delovanja vizualizacije našega sistema.

Virtualno okolje je sestavljeno iz predstavitve tal in neba. Tla našega okolja so sestavljena iz polinomov in teksture trave, kar pripomore k realizmu. Prav tako podpirajo odkrivanje trkov, s čimer preprečijo modelu, da se ne bi pogreznil vanj in omogočijo boljšo vizualizacijo gibanja.

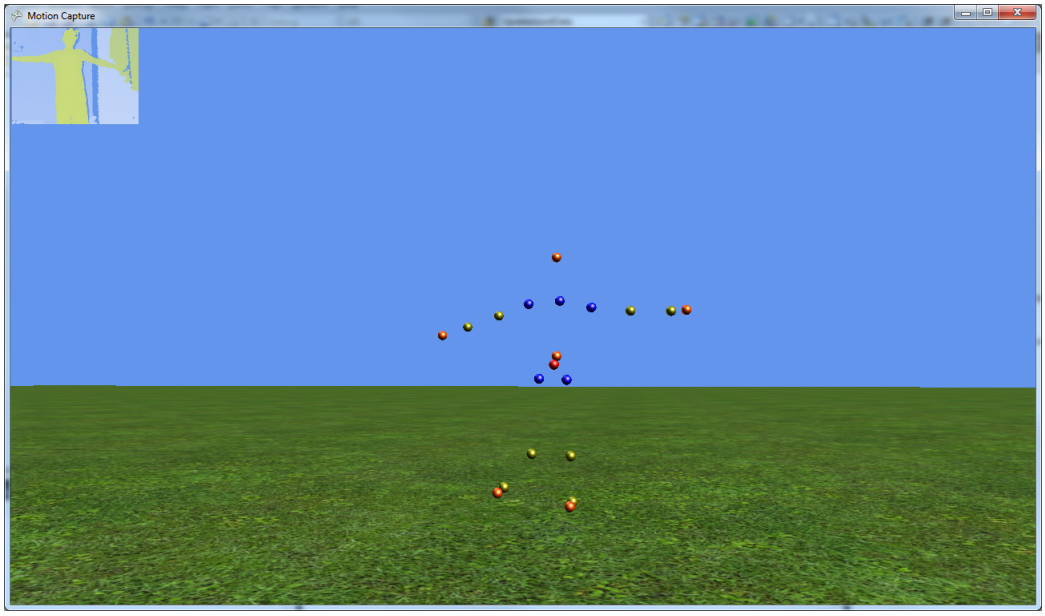
Slika 2.12: Delovno okolje vizualizacijskega programskega orodja, izdelanega kot pripomoček pri zajemanju gibanja.

Poleg 3D modela lahko podatke predstavimo kot množico 3D krogel, kot vidimo na sliki 2.13. Za lažje ločevanje med sklepi so krogle obarvane z naslednjimi barvami:

- rdeča – koren našega skeleta. Ima največji vpliv gibanje 3D modela, saj se zaradi translacije tega namreč spremeni pozicija celotnega telesa.

- modra – otroci, na katere ima koren direkten vpliv. Sami imajo največ vpliva na gibanje posameznih okončin.
- rumena – vsi ostali sklepi v skeletu.
- oranžna - zadnji členi v hierarhiji skeleta, ki imajo na gibanje 3D modela zelo majhen vpliv. Tako je tudi obarvana sredina hrbta, ki v 3D modelu nima nobenega vpliva na gibanje.

Ta predstavitev podatkov nam omogoča lažji pregled pozicije sklepov in vizualizira povezavo med 3D modelom in sklepi, ki iz samih podatkov drugače ne bi bile jasne. Ko dobimo nove informacije o sklepih preko dogodkov, se posamezna krogla s pomočjo translacije premakne v pravilni položaj.



Slika 2.13: Delovno okolje vizualizacijskega programskega orodja s spremenjenim modelom.

Zaradi jasnosti podatkov o uspešnosti zajetega položaja posameznih sklepov niso prikazani v delovnem okolju. Vendar se v levem zgornjem kotu delovnega okolja nad 3D modelom izrisujejo podatki, zajeti iz globinskega toka senzorja Kinect.

Slika 2.14: Prikaz uspešne identifikacije osebe, obarvane z rdečo barvo. Podatki so bili zajeti iz globinskega toka senzorja Kinect. Podrobnejši opis zajemanja podatkov je v poglavju 2.1.3.1.

Izris zajetih podatkov iz globinskega toka lahko prav tako služi za boljšo predstavo o tem, kdaj so bili podatki o osebi zajeti in identificirani, ter kako senzor Kinect vidi dele našega telesa. Prav tako lahko na vizualizaciji vidimo, katere dodatne predmete je senzor Kinect po pomoti vzel kot del našega telesa in jih odstranimo. S tem si zagotovimo veliko boljšo natančnost podatkov, kot če bi imeli vizualiziran samo 3D model. V kolikor pride do izgube ali zamenjave osebe, se spremeni tudi barva obrisa osebe.

2.2.4 Interakcija z aplikacijo MotionCapture

V programskem okolju lahko uporabnik zajame Kinectove vhodne podatke in jih posname na računalnik za kasnejše gledanje. Preden začnemo snemati, predvajati in interpolirati, se moramo postaviti pred senzor Kinect, da zazna naš skelet. Če tega ne storimo, ne moremo začeti nobenega procesa, saj potrebujemo začetno točko o Kinectovem skeletu. S tem pridobimo informacije o sklepih skeleta, ki so nujni za začasno shranjevanje podatkov.

Proces snemanja zaženemo s pritiskom na gumb R. Aplikacija nas nato opozori, da se bo snemanje začelo izvajati čez določen časovni interval, kar

nam omogoči pripravo. Ko je časovni interval enak 0, se začne proces snemanja. Takrat se vsi podatki o sklepih shranjujejo v začasno tabelo, iz katere se potem prenesejo na model. Snemanje zaključimo s ponovnim pritiskom na gumb R. Po končanem snemanju se vsi podatki, shranjeni v tabeli, prenesejo v XML datoteko, podrobneje opisano v poglavju 2.2.5.1

Slika 2.15: Prikazana je sprememba zornega kota kamere v delovnem okolju.

Poleg Kinectovih vhodnih podatkov lahko uporabnik uvede nove vhodne posnetke že zajetega gibanja in opazuje premikanje modela. Naloženo premikanje se ob pritisku na gumb P naloži v začasno tabelo in začne predvajati. Uporabnik lahko ob premikanju opazuje gibe, ki jih je zajel. Ko posnetek pride do zadnjega giba, se začne ponovno predvajati. To lahko ustavimo s ponovnim pritiskom na gumb P.

V primeru, če želi uporabnik videti gibanje iz drugačnega zornega kota, lahko s pomočjo NumPad, PageUp in PageDown gumbov spremeni zorni kot kamere (Slika 2.15). Prav tako lahko s pritiskom na gumb M zamenja model, ter omogoči direkten pogled na podatke o poziciji sklepov, kot je prikazano na sliki 2.12.

Poleg snemanja in predvajanja gibanja imamo tudi možnost interpoliranja gibanja. Uporabnik si lahko ob pritisku na gumb I izbere enega ali več posnetkov, ki jih želi interpolirati. Sledi prikaz novo nastalega gibanja, kjer lahko opazujemo interakcijo interpoliranega gibanja z različnimi posnetki. Interpolacijo lahko uporabnik ustavi s ponovnim pritiskom na gumb I. Podrobnejši opis implementacije je v poglavju 3.2.

Uporabnik lahko tudi že zajeto gibanje pretvori v BVH format (poglavje 2.2.5.2). S pritiskom na gumb C uporabnik zažene proces pretvarjanja. Na zaslonu se pojavi iskalec datotek, s pomočjo katerega lahko izberemo XML datoteko, ki jo želimo pretvoriti. Ko smo zadovoljni z izbiro, pritisnemo gumb Open. Takrat se zažene proces, ki se sprehodi skozi izbrano datoteko. Iz nje izlušči podatke o hierarhiji in položajih sklepov, ki jih pretvori v rotacije in zapiše v datoteko z istim imenom kot začetna datoteka.

2.2.5 Format podatkov

2.2.5.1 XML format podatkov

Extensible Markup Language (XML) je označevalni jezik, ki opredeljuje določeno število pravil za kodiranje dokumentov v obliki, ki je berljiva človeku in računalniku. Namen tega formata v sistemu je shranjevanje zajetih zajemkov. Poleg podatkov o gibanju, uporabniku posreduje podatke o zgradbi skeleta in uspešnosti zajemanja podatkov.

Dokument se začne z verzijo XML-a, čemur sledi ArrayOfSkeletonFrameHistory, kar napoveduje, da bo v nadaljevanju dokumenta predstavljeno določeno število zajemkov, kjer je vsak zajemek označen s SkeletonFrameHistory. V temu delu dokumenta je s Key time označen čas, kdaj se je trenutni zajemek začel. Temu sledi beseda Joints, ki napoveduje, da bo v nadaljevanju dokumenta predstavljena hierarhija človeškega skeleta. Najvišje v tej hierarhiji je ROOT (koren), katerega premikanje vpliva na premikanje vseh ostalih sklepov. V našem primeru je to HipCenter (sredina kolkov). Sledi del, imenovan SerializableJoint, v kateri so vsebovani podatki posameznega

sklepa, kot so ime sklepa, ki je označeno z JointID. Zatem sledi informacija o tem, kako dobro je bila pozicija sklepa zajeta, kar je označeno z besedo TrackingState. Za razliko od ostalih ima ta informacija samo tri stanja:

- **Tracked** – podatki so bili uspešno zajeti.
- **Not tracked** – podatki niso bili zajeti.
- **Inferred** – pri zajemanju podatkov je prišlo do zakrivanja sklepa. Zajeti podatki so samo približki, izračunani na podlagi sosednih sklepov.

Zadnji segment v SerializableJoint je Position, ki določa položaj sklepa v X, Y, Z koordinatnem sistemu.

Za korenom sledijo še vsi ostali sklepi, kot so prikazani na slik 3.1. V ostalem delu dokumenta se za vsak zajeti zajemek ponovi SkeletonFrameHistory. Za lažjo predstavo je v prilogi A primer takšnega dokumenta.

2.2.5.2 BVH format podatkov

Biovision hierarchical data (BVH) format dokumentov je bil razvit v podjetju Biovision. Namen razvoja tovrstnega formata je bila želja po delitvi podatkov zajetega gibanja s strankami. BVH formate je nadomestil prejšnji format podjetja z namenom, da poleg podatkov o gibanju, uporabniku posreduje tudi podatke o zgradbi skeleta [14].

Sestavljen je iz dveh delov. Prvi podaja hierarhijo skeleta in njegovo začetno pozicijo, v drugem delu pa je v obliki sklepov in pozicije korena predstavljeno gibanje.

Dokument se začne z besedo Hierarchy, ki napoveduje, da bo v nadaljevanju dokumenta predstavljena hierarhija človeškega skeleta. Najvišje v tej hierarhiji je ROOT(koren), v našem primeru je to Hip (sredina kolkov). Premikanje korena vpliva na premikanje vseh ostalih sklepov. Za besedo ROOT je napisano ime korena in par zavitih oklepajev, v katerih je napisana preostala hierarhija. Z besedo OFFSET je določen odmik trenutnega sklepa v

X, Y, Z koordinatah od sredine koordinatnega sistema. Zatem sledi informacija o kanalih posameznega sklepa, kar se začne z besedo CHANNELS. Temu sledi število kanalov, zatem pa je napisan seznam z enakim številom oznak, ki določajo tip kanala. Vrstni red oznak določa vrstni red zapisov v drugem delu, kjer je predstavljeno gibanje. Teh kanalov je lahko kolikor želimo, vendar pa je navadno za koren šest kanalov, za ostale sklepe pa trije. V našem primeru si kanali, ki določajo pozicijo, sledijo X, Y, Z, medtem ko je pri rotaciji zaporedje Y, Z, X. Nadaljnja hierarhija se začnejo z besedo JOINT, ki ji sledi ime sklepa. Ostalo je enako kot pri korenu. Zadnji segmenti v hierarhiji se začnejo z End Site, ki mu v odmikih sledijo le še odmiki od središča koordinatnega sistema. V tem segmentu rotacije niso podane.

V drugem delu dokumenta so predstavljeni podatki o gibanju. Segment se začne z besedo MOTION. Temu sledi beseda Frame, za katero je zapisano število zajemkov, ki so v tem dokumentu, nato časovni razmak med dvema zajemkoma in na koncu še za vsak zajemek X,Y, Z pozicija korena v prostoru, katerim sledijo podatki o Y, Z, X rotacijah sklepov, ki so v hierarhiji. Vsak zajemek obsega eno vrstico. Za lažjo predstavo je v prilogi B primer takšnega dokumenta.

Poglavje 3

Interpolacija gibanja

Na matematičnem področju numerične analize je interpolacija metoda gradnje novih podatkovnih točk pri nekem razponu znanih podatkovnih točk. V okviru računalništva se interpolacija nanaša npr. na interpolacijo ključnih položajev v računalniški grafiki, interpolacija gibanja v sistemih za zajemanje gibanja itd. Poznamo veliko različnih interpolacijskih metod, ki se razlikujejo po natančnosti proizvajanja interpoliranih točk, ter časovni in prostorski potratnosti [10]. Najbolj znane so:

1. **Interpolacija najbližjega soseda** – poišče najbližjo vrednost podatkov in dodeli podatkovni točki enako vrednost.

2. **Linearna interpolacija** – linearno interpoliramo med podatkovnimi točkami.

3. **Polinomska interpolacija** – posplošitev linearne interpolacije, kjer interpolant zamenjamo s polinomom višje stopnje.

4. **Interpolacija z zlepki** – interpoliramo s krivuljami višjih redov, npr. Bezierjevimi zlepki.

5. **Interpolacija preko Gaussovih procesov** – uporablja nelinearni Gaussov proces za interpolacijo točk.

Pri izbiri ustreznega algoritma za uporabo na določenem sistemu moramo postaviti naslednja vprašanja:

- Kako natančna je metoda?
- Kako potratna je prostorsko in časovno?
- Kako gladka je interpolacija?
- Koliko podatkovnih točk je potrebnih?

Na podlagi teh vprašanj smo se odločili za linearno interpolacijo. Ta proizvede interpolirane točke dovolj hitro in natančno za sistem v realnem času. Prav tako, v primerjavi z drugimi metodami, ne potrebuje veliko podatkovnih točk, kar zmanjša prostorsko in časovno kompleksnost implementacije.

3.1 Linearna interpolacija

Eden najpreprostejših načinov interpolacije je linearna interpolacija. Pogosto se uporablja za zapolnitev vrzeli v tabeli in tudi v računalniški grafiki, ki se včasih v žargonu imenuje lerp. Operacije so vgrajene v strojno opremo vseh sodobnih računalniških grafičnih procesorjev. Pogosto se uporabljajo kot gradniki za bolj kompleksne operacije, kot npr. bilinearna interpolacija, ki jo lahko dosežemo z dvema lerpoma [11].

Na splošno potrebuje linearna interpolacija dve podatkovni točki (x_0, y_0) in (x_1, y_1). Interpolirana točka (x, y) je potem ravna črta med tema dvema točkama in je podana z enačbo 3.1.

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0} = y_0 + \frac{(x - x_0)y_1 - (x - x_0)y_0}{x_1 - x_0} \tag{3.1}$$

Recimo, da imamo podano funkcijo f(x), ki v točki x generira podatke, zapisane v tabeli 3.1. Razmislimo, da bi s pomočjo linearne interpolacije

ocenjevali f(x) v točki x = 2.5. Ker je 2.5 med 2 in 3, je smiselno, da je f(2.5) na sredini med f(2) = 0.9893 in f(3) = 0.1411, kar daje 0.5252. Iste podatke dobimo z enačbo 3.1 , kar vidimo v 3.2.

$$f(2.5) = 0.9093 + (0.1411 - 0.9093)\frac{2.5 - 2}{3 - 2} = 0.5252 \tag{3.2}$$

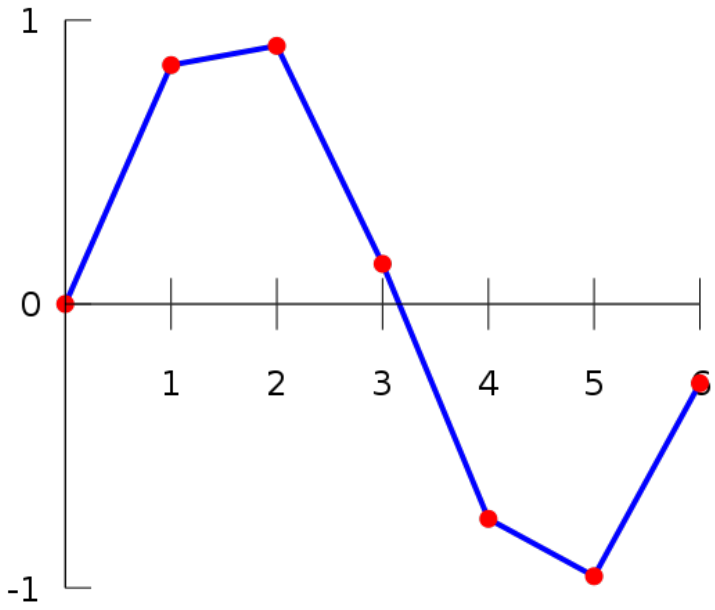
Linearna interpolacija je enostavna in hitra, vendar ni zelo natančna in ni odvedljiva v točki x_k [11]. Naslednja ocena napake pokaže, da linearna interpolacija ni zelo natančna. Označuje funkcijo, ki jo želimo interpolirati z g in domnevamo, da x leži znotraj intervala x_0 in x_1, ter tudi, da je g neskončnokrat odvedljiva. Potem je napaka linearne interpolacije enaka:

$$|f(x) - g(x)| \leq C(x_1 - x_0)^2, C = \frac{1}{8} \max_{y \in [x_0, x_1]} |g''(x)| \tag{3.3}$$

Torej napaka je sorazmerna s kvadratom razdalje med podatkovnimi točkami. Napaka v nekaterih drugih metodah, vključno s polinomsko interpolacijo in interpolacijo z zlepki, je sorazmerna z najvišjim kvadratom razdalje med podatkovnimi točkami. Iz tega tudi sledi, da te metode proizvedejo bolj gladko interpolacijo in v nekaterih primerih služijo za dobro zamenjavo za linearno interpolacijo.

Tabela 3.1: Tabela podatkov za interpoliranje.

x	f(x)
0	0
1	0 .8415
2	0 .9093
3	-0.1411
4	-0.7568
5	-0.9589
6	-0.2794



Slika 3.1: Prikazan je primer linearne interpolacije, kjer so rdeče točke podatki, podani v tabeli 3.1, modre črte pa podatki, interpolirani z metodo linearne interpolacije, podane v enačbi 3.1

.

3.2 Implementacija linearne interpolacije

Implementacija linearne interpolacije omogoča interpoliranje gibanja položajev sklepov skeleta z dveh ali več posnetkov. Po izbiri posnetkov, med katerimi želimo interpolirati podatke, se posnetki prenesejo v tabelo, shranjeno v pomnilniku računalnika in začnejo predvajati. Ko računalnik zazna, da je prišel do zadnjega giba v določenem posnetku, vzame zadnji gib tekočega posnetka in prvi gib naslednjega posnetka ter med njima zažene proces interpolacije.

Proces interpolacije temelji na razstavitvi skeleta zadnjega in prvega giba v tabelo sklepov, kjer so v vsaki tabeli napisani X, Y, Z vektor pozicije sklepa, stanje zajetega giba in časovna enota zajetega gibanja. Interpolacija se nato vrši na vsakem sklepu posebej, kjer interpoliramo podatke med trenutno pozicijo in pozicijo prvega giba naslednjega posnetka, kjer se trenutna pozi-

cija v vsaki iteraciji zamenja z interpolirano pozicijo. Trenutni podatki se pred zamenjavo shranijo v novo tabelo, ki bo uporabljena kot interpolirani posnetek med dvema gibanjema. Ta proces se vrši, dokler ni razdalja med trenutnim posnetkom in prvim posnetkom dovolj majhna za vsak sklep.

Interpolacija med posameznimi pozicijami se vrši s pomočjo ukaza Vector3.Lerp. Ta izvrši interpolacijo med dvema vektorjema po enačbi 3.4 in vrne interpolirani vektor ter prejme naslednje parametre:

- Vector3 value1 – prvi začetni vektor.
- Vector3 value2 – drugi začetni vektor.
- float amount – vrednost med 0 in 1, ki označuje težo drugega vektorja.

Po uspešni izvršeni metodi linearne interpolacije se interpolirani podatki posredujejo našemu 3D modelu, ki gibe vizualizira. Težave lahko nastanejo, če je razdalja med začetnim in končnim gibom prevelika. Takrat se izredno poveča časovna in prostorska potratnost celotne aplikacije. To težavo smo rešili s pomočjo omejevanja gibanja na 1000 slikovnih elementov.

$$value = value1 + (value2 - value1) * amount \tag{3.4}$$

Poglavje 4

Uvoz podatkov

Po uspešnem zajemanju gibanja je pogosto potrebno uporabiti tradicionalne tehnike animacije, kot so poudarek na predvidevanju in sledenju gibanja, sekundarnega gibanja, spreminjanja oblike osebka ali spreminjanja samega osebka. Ponavadi se uporabljajo posebni programi, s katerimi se lahko ustvarja 3D interaktivne aplikacije, računalniške igre, animirane filme, televizijske serije ali vizualne efekte. Za obdelavo podatkov, ki smo jih do sedaj pridobili, bomo uporabili Autodesk Mayo in Autodesk MotionBuilder.

Preden začnemo podatke obdelovati, jih moramo uvoziti. Za to je bilo potrebno napisati program v jeziku Python, imenovan KinectXMLToMaya. Ta bo naše podatke, napisane v XML formatu, uvozil in pretvoril v podatke, primerne za obdelavo v Autodesk Mayi. Motivacija za izdelavo programskega orodja temelji tudi na veliki količini časovno odvisnih tekstovnih podatkov. Vizualni pregled gibanja v Mayi je veliko bolj intuitivno in omogoča lažjo detekcijo ter odpravo napak, ki so se lahko zgodile zaradi prekrivanja določenih površin ali gibanja, ki ga Kinect ni zaznal.

4.1 Autodesk Maya

Autodesk Maya, pogosto skrajšana Maya, je računalniška programska oprema za 3D grafiko, ki deluje na operacijskem sistemu Microsoft Windows, Mac

OS in Linux. Uporablja se za ustvarjanje 3D sredstev za uporabo v filmih, TV serijah, razvoju iger in arhitekturi.

Uporabnik opredeli virtualni delovni prostor (sceno) za izvajanje in urejanje medijev določenega projekta. Maya tudi izpostavlja arhitekturo vozlišča grafov. Zaradi tega vizualna predstavitev scene v celoti temelji na mreži med seboj povezanih vozlišč, odvisnih od informacije vseh drugih. Programska oprema prinaša praktična orodja za pomoč ustvarjanja in vzdrževanje današnje zahtevne 3D animacije, vizualnih učinkov, razvoja iger in postprodukcijskih projektov. Vsebovana orodja:

- **Fluid Effect** – realističen simulator tekočine (velja za dim, ogenj, oblake in eksplozije).
- **Classic Cloth** – samodejno simulira gibljivost realnih oblačil in tkanin.
- **nHair** – omogoča ustvarjanje osupljivih, zelo realističnih las in ostalih elementov, ki temeljijo na dinamiki krivulje.
- **Fur** – omogoča ustvarjanje živalske dlake ali krznu podobne predmete. Simulacija je podobna nHair.
- **Maya Live** – nabor orodij za sledenje CG gibanja.
- **nCloth** – nabor orodij, ki dajo umetnikom nadaljnji nadzor s simulacijami z blagom in ostalimi podobnimi materiali.
- **nParticle** – simulacija kompleksnih 3D učinkov, vključno s tekočinami, oblaki, dimov, spreji in prahom.
- **MatchMover** – omogoča komponiranje CGI elementov z gibalnimi podatki, sestavljenih iz zaporedja videov ali filmov.
- **Composite** – komponiranje filma s pomočjo interaktivnih vozlišč.
- **Camera Sequencer** – omogoča postavitev večih kamer in upravljanje z njimi v določenem zaporedju.

Poleg ostalih orodij, je Maya tudi opremljena s svojim skriptnim jezikom, ki se imenuje Maya Embedded Language (MEL). MEL je uporabljen za prilagoditev osnovnih funkcij programske opreme, saj je veliko orodij in ukazov, ki se uporabljajo v njem, tudi napisanih. Prav tako vsebuje podporo za skriptni jezik Python.

4.2 Autodesk MotionBuilder

Autodesk MotionBuilder je strokovna programska oprema za 3D animacije v realnem času, filmsko proizvodnjo in virtualno kinematografijo. S pomočjo teh funkcionalnosti zelo dobro dopolnjuje Autodesk Mayo in Autodesk 3ds-Max programsko opremo. Prav tako ima podporo FBX, BVH in številnih drugih formatov, kar omogoča izmenjavanje ustvarjenih animacij z drugimi orodji.

Uporablja se za zajemanje podatkov o gibanju, manipulacijo in vizualizacijo podatkov, ki jo naredijo, ne le produktivna animacijsko rešitev, ampak orodje, ki poganja iterativni proces ustvarjanja. Omogoča tudi uporabo tradicionalne tehnike animacije s ključnimi položaji. Zaradi tega je zelo primeren za uporabo v proizvodnji iger (Assasin's Creed, Kilzon2 itd.), filmov(Avatar) in drugih multimedijskih projektov.

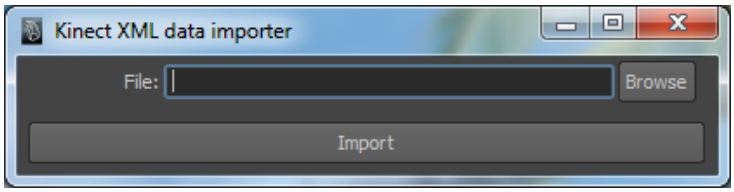
MotionBuilder ponuja animatorjem in tehničnim direktorjem naslednje možnosti:

- prikaz animacije v realnem času in številna animacijska orodja,
- obrazna in skeletna animacija,
- revolucionarno tehnologijo HumanIK oziroma inverzne kinematike,
- enotno nelinearno urejanje virtualnega okolja,
- časovni okvir za urejanje zgodb,
- orodja za prilagajanje in razširitev programske opreme in podporo za skriptni jezik Python ter C++.

Kot eden izmed vodilnih programov v realno časovni CG tehnologiji MotionBuilder omogoča proces virtualne kinematografije. To režiserjem in snemalcem omogoča interaktivno vodenje virtualne kamere z enako prostostjo, kot jo imajo v realnem svetu.

4.3 Uvoz podatkov v Mayo

V okolju Maya lahko uporabnik uvede nove podatke o gibanju s pomočjo programa KinectXMLToMaya in opazuje ter ureja gibanje iz podatkov. Program KinectXMLToMaya lahko zaženemo s pomočjo Script Editorja, ki ga najdemo v meniju Window > General Editors > od Maye. Če ga želimo zagnati, moramo izbrati celotno programsko kodo in pritisniti na Execute gumb.



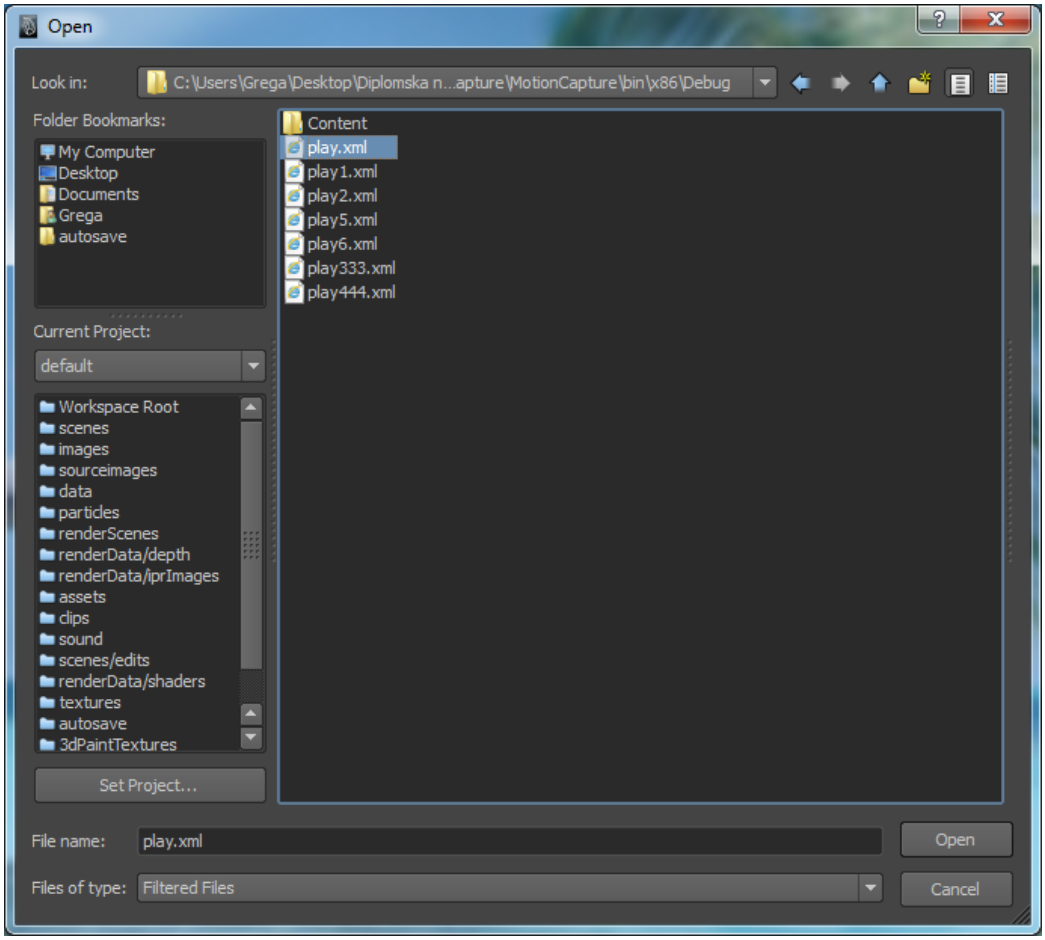
Slika 4.1: Prikazana je program KinectToXMLMaya, izdelan za uvoz podatkov, zajetih s senzorjem Kinect v Mayo.

Ime datoteke, ki jo želimo uvesti, lahko napišemo v polje. Pomagamo si lahko tudi z iskalcem datotek, prikazano na sliki 4.2. Ta je nastavljen tako, da prikaže samo XML datoteke. Odpremo ga s pritiskom na gumb Browse. Z njegovo pomočjo izberemo datoteko, ki jo želimo uvoziti v Mayo. S pritiskom na gumb Open potrdimo izbiro in se vrnemo nazaj na glavni zaslon KinectXMLToMaya. Izbrana datoteka se ob pritisku na gumb Import uvozi v Mayo. Uvoženi podatki so prikazani na sliki 4.3.

Uvažanje podatkov v Mayo poteka s pomočjo funkcije openXML. Ta odpre izbrano datoteko, prebere vse podatke ter iz njih izlušči ime sklepa in X, Y, Z koordinate. Ob prvi pojavitvi SkeletonFrameHistory ustvari modre

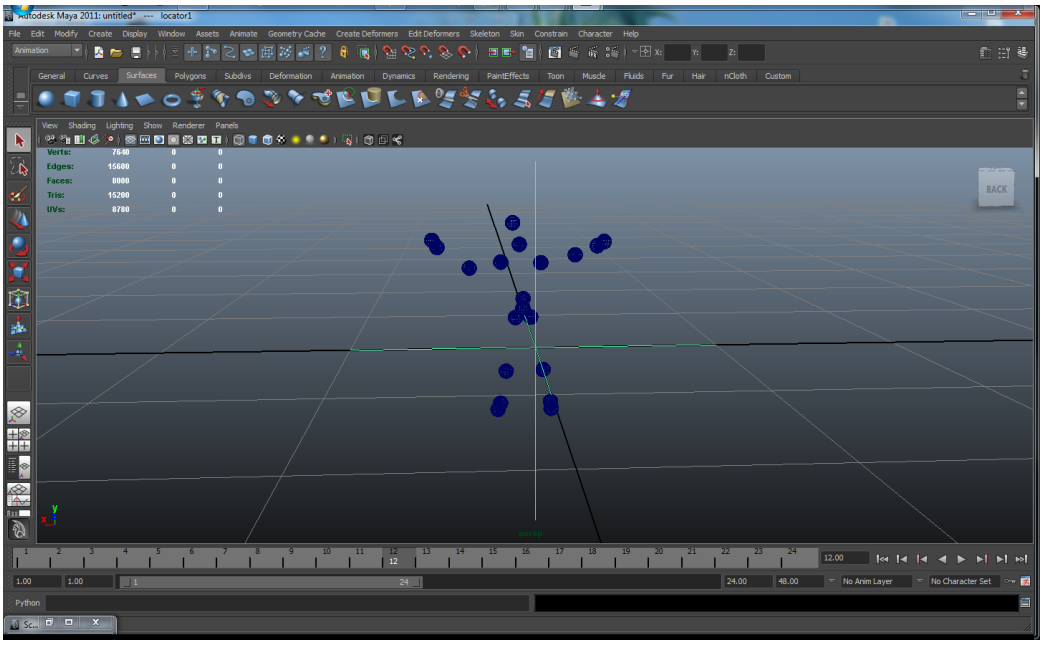
3D krogle. Kasneje samo spreminjamo njihovo pozicijo glede na koordinate, ki jih preberemo iz datoteke.

Če pred branjem pride do napake, nas program o tem obvesti in zaključi uvažanje. Ostale funkcije skrbijo za vizualizacijo programa. Za lažje razumevanje celotnega programa je KinectXMLToMaya v prilogi C.



Slika 4.2: Prikazan je iskalec datotek programa KinectToXMLMaya, izdelan za izbiranje podatkov, zajetih s senzorjem Kinect v Mayo. Nastavljen je tako, da prikazuje samo datoteke, ki so XML formata.

.



Slika 4.3: Prikazani so uvoženi podatki v Mayo. Na sliki so jasno vidne modre krogle, ki ponazarjajo zajete podatke posameznih sklepov kot jih vidi senzor Kinect.

.

4.4 Uvoz podatkov v MotionBuilder

V okolje MotionBuilder lahko uvozimo številne različne formate podatkov zajetih s pomočjo sistemov za zajemanje gibanja. Če želimo podatke zajete z našim sistemo uvoziti, jih moramo prvo pretvoriti v format, ki ga MotionBuilder podpira. To naredimo v aplikaciji MotionCapture s pritiskom na gumb C. Ob pritisku, se pokaže iskalec datotek, s pomočjo katerega izberemo datoteko, ki jo želimo pretvoriti. S pritiskom na gumb Open zaženemo proces pretvarjanja iz XML v BVH format.

Proces pretvarjanja temelji na razstavitivi skeleta na njegovo hierarhijo in na podatke o gibanju. V prvem koraku pretvornik vzame prvi gib v izbrani datoteki ter iz nje razbere hierarhijo skeleta in začetne odmike od izhodišča koordinatnega sistema. Te podatke zapiše v prvi del dokumenta BVH, kot

napisano v poglavju 2.2.5.2. V tabeli 4.4 je prikazana preslikava imen posameznih sklepov iz hierarhiji Kinecta v hierarhijo BVH formata, ki jo MotionBuilder zna prebrati. Če tega ne naredimo, program MotionBuilder naših podatkov ne bi znal uporabiti.

Tabela 4.1: Tabela preslikave imen iz hierarhije Kinecta v hierarhijo, primerno za uporabo v programu MotionBuilder.

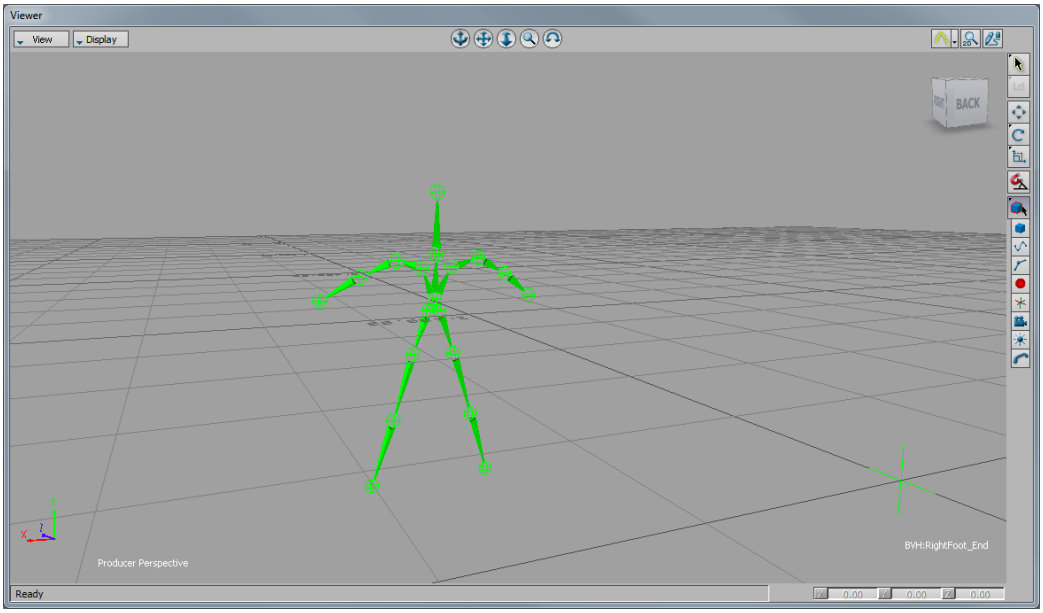
številka sklepa	hierarhija Kinecta	hierarhija MotionBuilderja
0	HipCenter	Hip
1	Spine	Chest
2	ShoulderCenter	Neck
3	Head	EndSite
4	ShoulderLeft	LeftCollar
5	ElbowLeft	LeftUpArm
6	WristeLeft	LeftLowArm
7	HandLeft	EndSite
8	ShoulderRight	RightCollar
9	ElbowRight	RightUpArm
10	WristeRight	RightLowArm
11	HandRight	EndSite
12	HipLeft	LeftUpLeg
13	KneeLeft	LeftLowLeg
14	AnkleLeft	LeftFoot
15	FootLeft	EndSite
16	HipRight	RightUpLeg
17	KneeRight	RightLowLeg
18	AnkleRight	RightFoot
19	FootRight	EndSite

Ko imamo ustvarjeno hierarhijo našega skeleta, moramo ustvariti še drugi del dokumenta BVH, ki bo vseboval podatke o gibanju. Tu se pojavi težava, saj moramo našega podatke o poziciji posameznega sklepa v 3D prostoru

pretvoriti v rotacijo sklepov v odvisnosti od njegovega starša. To naredimo tako, da ustvarimo vektorja U in V, kjer je vektor U razdalja med trenutnim sklepom in njegovim staršem, vektor V pa kot razdalja med trenutnim sklepom in njegovim otrokom. Nato vektorja normaliziramo in izračunamo osi rotacije in kot (enačba 4.1).

$$axis = U \times V; angle = \arccos(U \cdot V); \tag{4.1}$$

Ko imamo osi rotacije in kot, jih pretvorimo v Eulerjeve kote v Y, Z, X koordinatnem sistemu s pomočjo implementirane metode toEuler. Pretvorjene podatke nato zapišemo v datoteko na isti lokaciji in z istim imenom, samo v BVH format. Sedaj lahko podatke uvozimo v MotionBuilderja. To naredimo tako, da v MotionBuilder odpremo meni File > Motion File Import, ki odpre iskalca datotek. V njem izberemo datoteko, ki smo jo pretvorili, in pritisnemo Open. Primer uvoženih podatkov lahko vidimo na sliki 4.4.



Slika 4.4: Prikazani so uvoženi podatki v MotionBuilder.

.

Poglavje 5

Zaključek

Nizkoproračunski sistem za zajemanje gibanja smo uspešno implementirali. Sistem se je do neke mere izkazal za uspešnega, čeprav se tudi po končani implementaciji v nekaterih pogledih ne more meriti z drugimi dražjimi sistemi. Za potrebe razumevanja zajetih podatkov smo tudi implementirali vizualizacijo, ki se je izkazala za močno orodje, saj je privedlo do lažjega razumevanja in berljivosti. Testiranje na množici gibov je prineslo želene rezultate, ki so dokazali, da naš sistem deluje pravilno.

5.1 Prednosti in slabosti

Implementirani sistem za zajemanje gibanja ponuja številne prednosti pred ostalimi sistemi in tudi pred tradicionalnimi računalniškimi animacijami 3D modelov. Bistvena prednost sistema v primerjavi z ostalimi je vizualizacija zajetih podatkov v realnem času, saj le redki sistemi to omogočajo. Vizualizacija omogoča opravljanje testov z različnimi slogi, pri čem smo omejeni samo s talenti osebka. Prav tako omogoča enostavno, natančno poustvaritev zapletenega gibanja in realne fizične interakcije kot sekundarni gibi, masa in izmenjava sil. Edina omejitev vizualizacije je marionetni pristop, saj smo zaradi velikega števila parametrov omejeni na en 3D model. Slabost tega pristopa postane očitna, ko želimo zamenjati 3D model. Takrat bi morali

ponovno zgraditi celotni model in ponovno nastaviti omejitve. Vendar ima veliko prednost pred ostalimi vizualizacijami 3D modelov ko imamo model zgrajen, dobimo zelo realistično gibanje in nimamo ostalih problemov, kot npr. izgubo prostorske stopnje.

Prav tako kot ostali sistemi za zajemanje gibanja ohranja implementirani sistem še eno omejitev. Gibanja, ki ne sledijo zakonom fizike, ni mogoče zajeti in prikazati. Prav tako so rezultati omejeni na tisto, kar se lahko izvede v okviru obsega zajemanja.

Poleg vizualizacije ponuja številne prednosti, kot so snemanje, predvajanje in interpolacijo zajetih podatkov. Zajete podatek smo shranili v XML format, v katerega shranjujemo hierarhijo sklepov in pozicijo posameznega sklepa v 3D prostoru. Implementirali smo tudi pretvornik, ki našemu sistemu omogoča pretvarjanje v bolj standarni BVH format, ki poleg odmikov posameznih sklepov podaja rotacijo sklepov okoli treh osi. To omogoča uvažanje podatkov v Autodesk MotionBuilder ali kakšno drugo programsko orodje, ki ta format podpira. Poleg pretvornika smo implemenirali program v Pythonu, ki podatke, zajete z našim sistemom, uvozi v Autodesk Mayo. Obe implementacije ponujata dodatno možnost obdelave podatkov, ki je pri drugih sistemih za zajemanje gibanja zelo pogrešana.

Trenutni rezultati pokažejo tudi omejitve pri implementaciji interpolacije podatkov. Podatkov, ki se med seboj preveč razlikujejo, sistem ne zna pravilno interpolirati, kar privede do vizualizacije čudnega gibanja. Takrat se izredno poveča časovna in prostorska potratnost celotne aplikacije. To težavo smo rešili s pomočjo omejevanja gibanja na 1000 slikovnih elementov. To nakazuje potrebo po vpeljavi različnih bolj kompleksnih metod interpolacije, ki bodo pokrivale številne različne vrste gibanja.

5.2 Nadaljnje delo

V prihodnosti želimo sprva izboljšati ali pa celo avtomatizirati vizualizacijo 3D modela, kjer bi uporabili marionetni pristop na katerem koli modelu.

Avtomatizacija tega postopka bi omogočila številne prednosti, saj bi lahko poljubno menjali modele, ne da bi morali za vsak model posebej graditi lutko iz cunj, kot je opisano v poglavju 2.2.2.

Možnost za nadaljnje delo vidimo tudi v izboljšanju sistema za interpoliranje gibanja. Z analizo ostalih metod interpolacije bi lahko med seboj primerjali učinkovitost posamezne metode ter s tem ugotovili njihove prednosti in slabosti. Na podlagi rezultatov bi kasneje lahko implementirali najboljšo metodo.

Prav tako vidimo izboljšave v programu za uvažanje podatkov. Namesto uvažanja podatkov v Mayo ali MotionBuilder bi lahko implementirali svoj program oziroma dodatek obstoječemu programu MotionCapture, ki bi nam omogočal spreminjanje in obdelovanje že obstoječih podatkov.

Implementacija in vizualizacija nizkoproračunskega sistema za zajemanje gibanja se je izkazala za uspešno in je po našem mnenju vredna nadaljnjega razvoja.

Slike

Tabele

Dodatek A

Primer XML formata podatkov

```
<?xml version="1.0"?>
<ArrayOfSkeletonFrameHistory
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <SkeletonFrameHistory>
           <KeyTime>00:00:00.0446657</KeyTime>
              <Joints>
                 <SerializableJoint>
                    <JointID>HipCenter</JointID>
                    <TrackingState>Tracked</TrackingState>
                    <Position>
                       <X>-0.000794850464</X>
                       <Y>0.008711346</Y>
                       <Z>1.22558486</Z>
                    </Position>
                 </SerializableJoint>
                 <SerializableJoint>
                    <JointID>Spine</JointID>
                    <TrackingState>Tracked</TrackingState>
                    <Position>
```

```
            <X>-0.00690927869</X>
            <Y>0.0628977</Y>
            <Z>1.27611971</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
        <JointID>ShoulderCenter</JointID>
        <TrackingState>Tracked</TrackingState>
        <Position>
            <X>0.00454859436</X>
            <Y>0.416219234</Y>
            <Z>1.3223573</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
            <JointID>Head</JointID>
        <TrackingState>Inferred</TrackingState>
        <Position>
            <X>0.0281315371</X>
            <Y>0.561480343</Y>
            <Z>1.3244195</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
        <JointID>ShoulderLeft</JointID>
        <TrackingState>Tracked</TrackingState>
        <Position>
            <X>-0.128003627</X>
            <Y>0.298585951</Y>
            <Z>1.27327609</Z>
        </Position>
```

```
</SerializableJoint>
<SerializableJoint>
    <JointID>ElbowLeft</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>-0.246436685</X>
        <Y>0.07642191</Y>
        <Z>1.26857734</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>WristLeft</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>-0.308136463</X>
        <Y>-0.14056845</Y>
        <Z>1.140723</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>HandLeft</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>-0.328094542</X>
        <Y>-0.23214075</Y>
        <Z>1.08478594</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>ShoulderRight</JointID>
    <TrackingState>Tracked</TrackingState>
```

```
    <Position>
        <X>0.14282459</X>
        <Y>0.3069509</Y>
        <Z>1.3229847</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>ElbowRight</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>0.2222976</X>
        <Y>0.071405746</Y>
        <Z>1.3502959</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>WristRight</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>0.2523134</X>
        <Y>-0.1579693</Y>
        <Z>1.23762143</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>HandRight</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>0.2504129</X>
        <Y>-0.232399821</Y>
        <Z>1.20110369</Z>
```

```
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>HipLeft</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>-0.05196163</X>
        <Y>-0.0488381423</Y>
        <Z>1.20311689</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>KneeLeft</JointID>
    <TrackingState>Tracked</TrackingState>
    <Position>
        <X>-0.135067984</X>
        <Y>-0.403920174</Y>
        <Z>1.16340554</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>AnkleLeft</JointID>
    <TrackingState>Inferred</TrackingState>
    <Position>
        <X>-0.180562213</X>
        <Y>-0.6131699</Y>
        <Z>1.12443662</Z>
    </Position>
</SerializableJoint>
<SerializableJoint>
    <JointID>FootLeft</JointID>
```

```
        <TrackingState>Inferred</TrackingState>
        <Position>
            <X>-0.179037645</X>
            <Y>-0.638327539</Y>
            <Z>1.05054474</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
        <JointID>HipRight</JointID>
        <TrackingState>Tracked</TrackingState>
        <Position>
            <X>0.0487977937</X>
            <Y>-0.05334563</Y>
            <Z>1.21765244</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
        <JointID>KneeRight</JointID>
        <TrackingState>Tracked</TrackingState>
        <Position>
            <X>0.113675252</X>
            <Y>-0.415467024</Y>
            <Z>1.187986</Z>
        </Position>
    </SerializableJoint>
    <SerializableJoint>
        <JointID>AnkleRight</JointID>
        <TrackingState>Inferred</TrackingState>
        <Position>
            <X>0.153914645</X>
            <Y>-0.6275282</Y>
```

```
                <Z>1.159193</Z>
            </Position>
        </SerializableJoint>
        <SerializableJoint>
            <JointID>FootRight</JointID>
            <TrackingState>Inferred</TrackingState>
            <Position>
                <X>0.173080936</X>
                <Y>-0.649708748</Y>
                <Z>1.08690667</Z>
            </Position>
        </SerializableJoint>
    </Joints>
</SkeletonFrameHistory>
```

Dodatek B

Primer BVH formata podatkov

```
HIERARCHY
ROOTHips
{
        OFFSET-0.02 -0.22 30.64
        CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
        JOINT Chest
        {
                OFFSET0.15 1.35 -1.26
                CHANNELS 3 Zrotation Xrotation Yrotation
                JOINT Neck
                {
                        OFFSET-0.13 10.19 -2.42
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        End Site
                        {
                                OFFSET -0.72 13.82 -2.47
                        }
                }
                JOINT LeftCollar
                {
```

```
            OFFSET3.18 7.25 -1.19
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftUpArm
            {
                    OFFSET6.14 1.69 -1.07
                    CHANNELS 3 Zrotation Xrotation
                    Yrotation
                    JOINT LeftLowArm
                    {
                            OFFSET7.68 -3.73 2.12
                            CHANNELS 3 Zrotation
                            Xrotation Yrotation
                            End Site
                            {
                                    OFFSET 8.18 -6.02
                                    3.52
                            }
                    }
            }
    }
    JOINT RightCollar
    {
            OFFSET -3.59 7.46 -2.43
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightUpArm
            {
                    OFFSET -5.58 1.57 -3.12
                    CHANNELS 3 Zrotation Xrotation
                    Yrotation
                    JOINT RightLowArm
                    {
```

```
                                        OFFSET -6.33 -4.17 -0.3
                                        End Site
                                        {
                                                OFFSET -6.28 -6.03
                                                0.61
                                        }
                                }
                        }
                }
        }
JOINT LeftUpLeg
{
        OFFSET 1.28 -1.44 0.56
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT LeftLowLeg
        {
                OFFSET 3.36 -10.32 1.55
                CHANNELS 3 Zrotation Xrotation Yrotation
                JOINT LeftFoot
                {
                        OFFSET 4.49 -15.55 2.53
                        CHANNELS 3 Zrotation Xrotation
                        Yrotation
                        End Site
                        {
                                OFFSET 4.46 -16.18 4.38
                        }
                }
        }
}
JOINT RightUpLeg
```

```
        {
                OFFSET -1.24 -1.55 0.2
                CHANNELS 3 Zrotation Xrotation Yrotation
                JOINT RightLowLeg
                {
                        OFFSET -2.86 -10.6 0.94
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT RightFoot
                        {
                                OFFSET -3.87 -15.91 1.66
                                CHANNELS 3 Zrotation Xrotation
                                Yrotation
                                End Site
                                {
                                        OFFSET -4.35 -16.46 3.47
                                }
                        }
                }
        }
}
MOTION
Frames: 3
Frame Time: 0.03333
-0.09 -0.82 122.56 0.08 2.31 0.19 -0.35 2.5 0.01 -1.67 2.85 -0.03 2.03
-2.89 -0.65 -0.2 -2.66 -1.04 2.78 2.17 0.09 -2.98 -2.88 0.61 0.37 -2.7
0.66 -1.5 0.86 1.33 2.28 2.9 -0.46 -0.43 3.11 -0.5 0.07 -2.02 -0.43 -2.95
2.96 0.48 -0.39 -3.03 0.3 -0.44 -2 0.03
-0.1 -0.77 122.56 0.08 2.31 0.18 -0.35 2.51 0 -1.67 2.85 -0.03 2 -2.84 -0.67
0.57 -2.31 -0.9 2.82 2.17 0.08 -2.97 -2.86 0.61 -0.07 -2.68 0.63 -0.77 0.22
1.3 2.28 2.9 -0.47 -0.41 3.11 -0.5 0.07 -2.02 -0.43 -2.96 2.96 0.48 -0.37
-3.03 0.31 -0.45 -2 0.03
```

-0.11 -0.73 122.56 0.08 2.31 0.18 -0.35 2.51 0 -1.67 2.84 -0.02 1.69 -2.38
-1.05 1.27 -1.85 -0.98 2.86 2.19 0.02 -2.86 -2.7 0.61 -1.08 -2.41 0.58 -0.24
-0.16 1.06 2.27 2.9 -0.47 -0.4 3.12 -0.5 0.07 -2.02 -0.43 -2.96 2.96 0.48 -0.35
-3.03 0.31 -0.45 -2 0.03

Dodatek C

Program za uvažanje XML podatkov

```
import maya
import maya.cmds as cmds
import xml
import xml.dom.minidom as xd

TEXT_NODE = 1

importFile = None
window = cmds.window()
cmds.columnLayout()
filePathButton = cmds.textFieldButtonGrp()

defopenXML(path):
    xmldoc = xd.parse(path)

    if xmldoc:
        rootNode = xmldoc.documentElement
        numberOfKeyFrames = 0
```

```
joints = { }

for node in rootNode.childNodes:
    if node.nodeName == "SkeletonFrameHistory"
      numberOfKeyFrames += 1
      if numberOfKeyFrames == 1:
        cmds.spaceLocator()
      cmds.currentTime(numberOfKeyFrames)
      for jointNodes in node.childNodes:
        if jointNodes.nodeName == "Joints":
          for serializableNode in jointNodes.childNodes:
              if serializableNode.nodeType == TEXT_NODE:
                jointID = "
                X = Y = Z = 0
                for n in serializableNode.childNodes:
                    if n.nodeType == TEXT_NODE:
                      if n.nodeName == "JointID":
                        jointID = str(n.toxml()[9:-10])
                      elif n.nodeName == "Position":
                        for pn in n.childNodes:
                            if pn.nodeName == "X":
                              X = float(pn.toxml()[3:-4])
                            if pn.nodeName == "Y":
                              Y = float(pn.toxml()[3:-4])
                            if pn.nodeName == Ž":
                              Z = float(pn.toxml()[3:-4])
                if joints.get(jointID) == None:
                    spheres = cmds.polySphere(radius = 0.05)
                    cmds.setKeyframe(at = 'translateX', v = X)
                    cmds.setKeyframe(at = 'translateY', v = Y)
                    cmds.setKeyframe(at = 'translateZ', v = Z)
```

```
                    joints[jointID] = spheres[0]
                else:
                    spheres = joints[jointID]
                    cmds.select(spheres, r = True)
                    cmds.setKeyframe(at = 'translateX', v = X)
                    cmds.setKeyframe(at = 'translateY', v = Y)
                    cmds.setKeyframe(at = 'translateZ', v = Z)
    else:
        print('Error getting XML document')

def showBrowseWindow():
    global importFile
    importFile = cmds.fileDialog(dm = '*.xml')
    fileSelection(importFile)
    return 1

def fileSelection(fileName):
    cmds.textFieldButtonGrp(filePathButton, edit = True, text = fileName)
    return 1

def startImport(fileName):
    openXML(importFile)
    global window
    cmds.window( window, edit=True, vis = False)
    return 2

def KinectXMLDataImport():
    global window
    global filePathButton
    window = cmds.window(title = "Kinect XML data importer",
    iconName = "2DXMLTrackToMaya", widthHeight=(400, 100) )
```

```
    cmds.setParent('..')
    cmds.columnLayout( columnAttach=('both', 5), rowSpacing = 10,
    columnWidth = 400)
    filePathButton = cmds.textFieldButtonGrp( label = 'File:', fileName =
    ,buttonLabel = 'Browse', columnWidth3 = (75, 200, 75),
    adjustableColumn3 = 2, columnAlign3 = ["right", "left", "center"],
    buttonCommand = showBrowseWindow )
    cmds.button( label = "Import", align ="center", command =
    startImport)
    cmds.showWindow(window)

KinectXMLDataImport()
```

Literatura

[1] Matija Marolt, Animacija pri predmetu RGTI, Avgust, 2012.

[2] Jehee Lee, Motion capture. Dostopno na strani `http://mrl.snu.ac.kr/courses/CourseAnimation/notes/MotionCapture.pdf`.

[3] David Noonan, Peter Mountney, Daniel Elson, Ara Darzi and Guang-Zhong Yang. A Stereoscopic Fibroscope for Camera Motion and 3D Depth Recovery During Minimally Invasive Surgery. In proc ICRA 2009 , pp. 4463-4468

[4] Microsoft Reserch, Kinect for Windows SDK beta, SkeletaViewer Walkthrough: C++ and C#.

[5] Microsoft Reserch, Kinect for Windows SDK beta, Programming Guide, Getting Started with the Kinect for Windows SDK beta from Microsoft Research, 22 Julij, 2011

[6] Digital Rune Engine: How to create ragdols. Dostopno na `http://www.digitalrune.com/Support/Blog/tabid/719/EntryId/128/How-to-Create-Ragdolls.aspx`, Avgust 2011.

[7] Harry Fairhead, Kinect's AI breakthrough explained. Dostopno na strani `http://www.i-programmer.info/news/105-artificial-intelligence/2176-kinects-ai-breakthrough-explained.html`, 30 Marec, 2012

[8] Wikipedia, Kinect. Dostopno na strani http://en.wikipedia.org/wiki/Kinect, Avgust 2012.

[9] Wikipedia, Motion capture. Dostopno na strani http://en.wikipedia.org/wiki/Motion_capture, Avgust 2012.

[10] Wikipedia, Interpolation. Dostopno na strani http://en.wikipedia.org/wiki/Interpolation, Avgust 2012.

[11] Wikipedia, Linear interpolation. Dostopno na strani http://en.wikipedia.org/wiki/Linear_interpolation, Avgust 2012.

[12] Motion capture explained. Dostopno na strani http://computerstories.net/motion-capture-explained/, August 14, 2011.

[13] Kiyoshi Hoshino, Interpolation and extrapolation of motion capture data. Dostopno na strani http://www.miv.t.u-tokyo.ac.jp/pricai02-LAA/papers/hoshino-kiyoshi.pdf, Avgust 2012.

[14] Jeff, Biovision BHV format. Dostopno na stran http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html, 1999.