NIVELIKKÄIDEN MEKANISMIEN TOTEUTUS TIETOKONEANIMAATIOSSA

Marko Aho

Helsinki 28.4.2000

PRO GRADU Tutkielma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET

Tiedekunta/Osasto	Laitos – Institution
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä(t)– Författare
Marko Aho

Työn nimi – Arbetets titel
Nivelikkäiden mekanismien toteutus tietokoneanimaatiossa

Oppiaine – Läroämne
Tietojenkäsittelytiede

Työn laji – Arbetets art	Aika – Datum	Sivumäärä – Sidoantal
Pro Gradu tutkielma	28.4.2000	41

Tiivistelmä – Referat

Tutkielmassa esitellään nivelikkäiden mekanismien teoria ja toteutus kolmiulotteisessa tietokoneanimaatiossa. Käänteiskinematiikan ratkaisuna esitetään yksinkertainen rekurssiomenetelmä, Jacobin matriisiin perustuva iteratiivinen menetelmä sekä ratkaiseminen minimointiongelmana. Tuloksena on suunnitelma reaaliaikaisesti laskettavasta nivelikkäiden mekanismien animointijärjestelmästä, jota voisi käyttää interaktiiviseen asennon määrittelyyn. Järjestelmä käyttää suoraa kinematiikkaa ja käänteiskinematiikkaa nivelikkään mekanismin reaaliaikaiseen ohjaamiseen. Työn kokeellisessa osuudessa on tehty nivelikkäiden mekanismien reaaliaikaisen ohjausjärjestelmän prototyyppi.

CR-luokitus: I.3.7 - Three dimensional graphics and realism - animation

Avainsanat – Nyckelord
reaaliaikainen interaktiivinen kolmiulotteinen animaatio, nivelikkäät mekanismit, käänteiskinematiikka

Säilytyspaikka – Förvaringställe
Tietojenkäsittelytieteen laitoksen kirjasto, sarjanumero C-2000-

Muita tietoja – Övriga uppgifter
CD-ROM-levy

Sisältö

Liitteet

Esipuhe

Tämä kirjoitus on tuotettu Helsingin yliopistossa Tietojenkäsittelytieteen laitoksella Pro Gradu työnä lukukausilla 1996-2000. Haluan kiittää kaikkia saamastani tuesta, erityisesti ohjaajiani apulaisprofessori Matti Mäkelää ja lehtori Harri Lainetta.

Marko Aho

1. Johdanto

Interaktiivinen kolmiulotteinen tietokoneanimaatio asettaa käytettävälle laitteistolle, algoritmeille ja malleille suuria vaatimuksia. Koska animaatio on interaktiivista, se pitää laskea reaaliajassa. Tällöin yleensä rajoitetaan käytettävien mallien realistisuutta. Yksinkertaisimmillaan joudutaan tyytymään piste- tai viivajoukkoihin.

Nivelikkäät mekanismit ovat yksi tapa muodostaa mielenkiintoisia liikkuvia kappaleita interaktiiviseen animaatioon. Esimerkiksi ihminen voidaan esittää nivelikkäänä mekanismina. Pelkistettynä tukirankana esitetty ihmishahmo saadaan vastaamaan hyvin käyttäjän antamiin käskyihin. Animaatio nopeasti reagoivalla tukirangalla tuntuu realistisemmalta kuin jos hahmo näyttää ihmiseltä, mutta se vastaa käyttäjän toimiin vain muutaman kerran sekunnissa.

Kehittyneissä laitteistoissa, joissa animaation laskenta on erotettu kuvien muodostamisesta nämä tukirangat voidaan päällystää kolmiulotteisilla kappaleilla, tekstuureilla, jopa elastisella nahalla. Tietokoneanimaatio-ohjelmistot ovat sisältäneet nämä ominaisuudet jo vuosia. Näissä ohjelmistoissa animaatio tuotetaan valmistamalla kuvat eräajopohjaisesti ja vain animaation ohjaus on interaktiivista. Toisaalta esimerkiksi kotitietokoneiden pelit pystyvät käyttämään ennalta tuotettuja liikkumistapoja hahmojen interaktiiviseen liikuttamiseen. Nivelikäs hahmo voi liikkua kolmiulotteisessa ympäristössä vaikkapa kävellen, juosten tai lentäen - usein pelaajaa kohden [Car96].

1.1 Tutkimusongelma

Kirjallisessa osassa on tutkittu, miten nivelikkäitä mekanismeja saadaan toteutettua tietokoneanimaatioon. Vertailukohdaksi on otettu kolmiulotteisen animaation tekemiseen tarkoitettu ohjelmisto, jossa on nivelikkäät mekanismit. On tuotettu suunnitelma, miten nivelikkäitä mekanismeja voidaan toteuttaa interaktiiviseen kolmiulotteiseen tietokoneanimaatioon tai virtuaalitodellisuusympäristöön.

Kokeellisessa osassa on tuotettu interaktiivisen nivelikkään mekanismin prototyyppi. Mekanismeja voi rakentaa osoittamalla lapsi-vanhempi-suhde ja mekanismi sallii haarautumisen eli hierarkiset rakenteet. Mekanismia voi ohjata suoraan niveliä ohjaamalla eli suoralla kinematiikalla sekä epäsuorasti tavoiteohjaamalla eli käänteiskinematiikalla.

Mekanismilla on jokin alkuasento. Alkuasentona käytetään joko mekanismin nykyistä asentoa tai sitä muutetaan suoralla kinematiikalla. Mekanismi halutaan saada johonkin toiseen asentoon - loppuasentoon. Mekanismista osoitetaan paikallaan pysyvä kohta ja tavoitteeseen kiinnitetty kohta, mihin ohjaus kohdistetaan. Asennon ohjauksessa käytetään sellaisia komentoja kuin “mene tuonne”, “katso tuohon suuntaan” tai “katso tuohon pisteeseen”. Ohjelman on tuotettava lopputulos mahdollisimman nopeasti siten, että interaktiivisuus säilytetään. Loppuasentoa ei voida kuitenkaan aina saavuttaa, jolloin tyydytään paikalliseen minimiin. Kun loppuasento on saavutettu, tavoitetta voidaan animoida. Esimerkiksi tavoitteena olevaa paikkaa voidaan muuttaa hieman, kuitenkin siten, että uusi paikka on lähellä vanhaa paikkaa. Ohjelma laskee uuden asennon ja tästä syntyy animaation toinen kuva. Kokeiden tuloksena saadaan vastaus kysymyksiin: “Voiko käänteiskinemaattista ohjausta käyttää reaaliaikaisesti? Jos voi, miten monimutkaista mekanismia voidaan ohjata?”

Alkuasennon ja loppuasennon välinen siirtymä voidaan tallettaa skriptiksi, ja toistaa nopeasti. Toiston aikana ei käytetä käänteiskinematiikkaa, vaan pelkkiä kulmanopeuksia tai kulmainterpolaatioita. Tällä tavalla voidaan määritellä esimerkiksi kävelysykli, jota toistetaan nopeasti uudelleen ja uudelleen.

1.2 Rajoitukset

Työssä keskitytään animaatioon eli liikkeen tuottamiseen. Kuvien tuottamista eli renderöintiä ei käsitellä. Animoinnissa ei tarkastella mekanismien massoja eikä tällaisen järjestelmän dynamiikkaa. Kappaleiden pintamalleja ei oteta huomioon, eikä niiden välisiä törmäyksiä. Kokeellisessa osassa animoitavat kappaleet esitetään vain tukirankoina, joiden päälle voidaan kuitenkin myöhemmissä toteutuksissa ripustaa pintamalli. Muita kuin edellä mainittuja ohjausmenetelmiä ei tutkita kokeellisessa osassa.

1.3 Toteutusympäristö

Toteutus on tehty C++-ohjelmointikielellä OpenGL-kirjaston avulla. Toteutus on siirrettävissä eri käyttöjärjestelmien välillä.

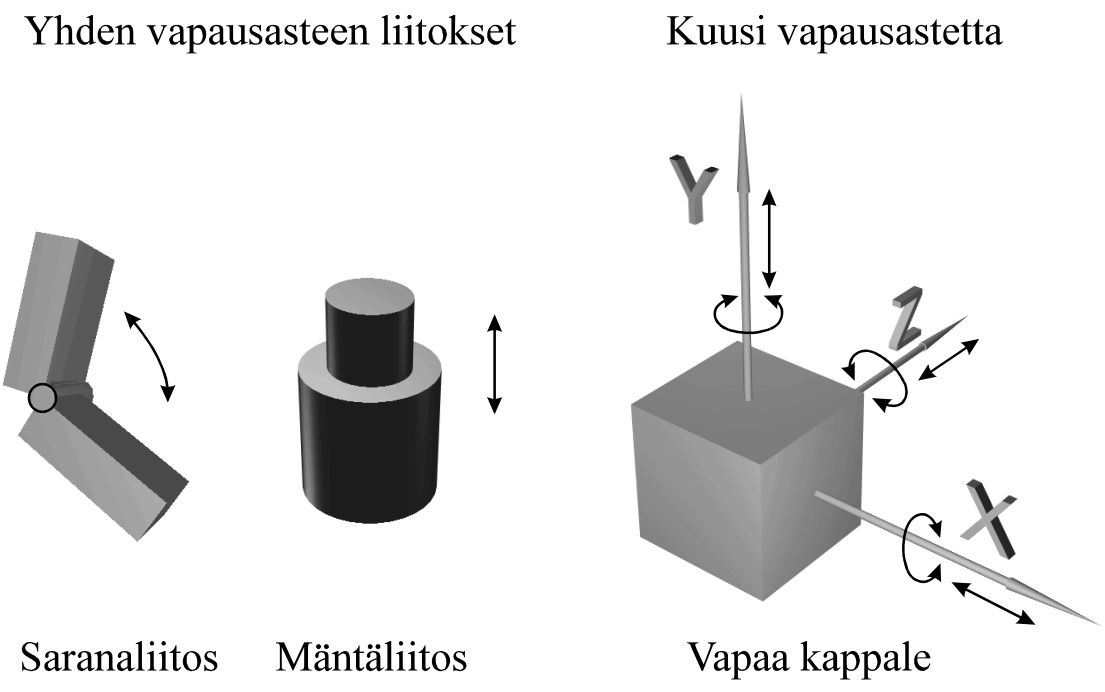
2. Käsitteet

2.1 Kinematiikka

Tietokoneanimaation ohjaaminen voidaan jakaa kolmeen ryhmään: avainkuva-animaatio, proseduraalinen animaatio ja dynaaminen simulointi. Avainkuva-animaatiossa ohjaaja määrittelee mallien asennot tietyissä avainkuvissa, ja tietokone interpoloi väliin tarvittavat kuvat [IsC87]. Proseduraalisessa animaatiossa käsitellään kappaleen paikkaa ja asentoa avaruudessa algoritmisesti. Kappaleille voidaan asettaa myös nopeus ja kiihtyvyys. Näin määritellään kappaleiden liike kinematiikassa. Vastaavasti dynaamisessa simuloinnissa määritellään kappaleiden liike kappaleisiin kohdistuvilla voimilla [Bad87].

2.2 Vapausasteet

Vapausasteet (Degrees of freedom, DOF) ilmoittavat, miten monta muuttujaa tarvitaan ilmaisemaan kappaleen paikka ja asento yksikäsitteisesti avaruudessa. Jos kappaleella on yksi vapausaste, se voi joko pyöriä jonkin akselin ympäri tai se voi liikkua akselin suuntaisesti. Kahdella vapausasteella se voi joko pyöriä kahden akselin ympäri, liikkua tasossa tai pyöriä akselin ympäri ja liikkua sen suuntaisesti. Vastaavasti vapausasteiden kasvaessa päästään kolmiulotteisessa avaruudessa aina kuuteen vapausasteeseen (Kuva 2.1), jolloin kappale voi sekä liikkua kaikkien kolmen akselinsa suhteen, että kiertyä niiden ympäri [IsC87].



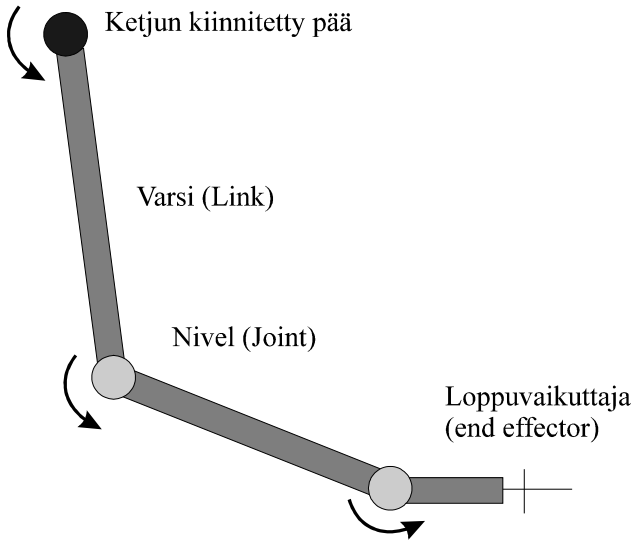
Kuva 2.1 Vapausasteita

2.3 Nivelikkäät mekanismit

Nivelikäs mekanismi (articulated figure) on rakenne, joka koostuu sarjasta jäykkiä varsia (links), jotka kytkeytyvät toisiinsa nivelin (joints). Tietokoneanimaatiossa rajoitutaan usein vain kiertoliikkeeseen pystyviin niveliin. Yhdellä nivelellä on tällöin yhdestä kolmeen vapausastetta. Nivel voidaan toteuttaa myös siten, että samassa kohdassa on kolme niveltä, joilla on vain yksi vapausaste kullakin [Kob82]. Robotiikassa on käytössä myös prismaattiset “nivelet” - mäntäliitokset [WaW92]. Käänteiskinematiikan käyttöä robotiikassa on selostettu muun muassa Chinin tutkielmassa [Chi96].

2.4 Ketju ja loppuvaikuttaja

Nivelin toisiinsa yhteenliitetyt jäykät kappaleet muodostavat avoimen kinemaattisen ketjun [KoB82]. Ketjun toinen pää (fixed end, proximal end) kytketään paikoilleen - usein maailmankoordinaatistoon. Ketjun vapaata päätä (free end, distal end) kutsutaan loppuvaikuttajaksi (end effector). Loppuvaikuttajan paikka ja asento määräytyy kaikkien ketjun kappaleiden asentojen perusteella. Loppuvaikuttajan vapausaste saadaan laskemalla yhteen kaikkien ketjun nivelten vapausasteet [WaW92]. Kuvassa Kuva 2.2 olevassa käsivarressa loppuvaikuttajalla on kolme vapausastetta.



Kuva 2.2 Robottikäsivarsi, jossa on kolme vapausastetta

2.5 Ohjaaminen

Nivelikäs mekanismi voidaan ohjata haluttuun asentoon joko suoraan nivelten asentoihin vaikuttamalla tai epäsuorasti määrittämällä loppuvaikuttajalle haluttu paikka tai asento. Suoraan ohjaamiseen tarkoitetut asentokäskyt voidaan antaa käyttöliittymästä

lukuarvoina tai osoitinlaitteella ja käskyt voidaan tallettaa myöhempää toistoa varten skripteihin. Vaihtoehtoisesti käskyt voidaan laskea proseduraalisesti kuten esimerkiksi partikkelianimaatiossa. Epäsuoraa ohjaamista sanotaan tavoiteohjatuksi asennon määritykseksi tai tavoiteohjatuksi animoinniksi [KoB82]. Tavoiteohjauksessa ohjaajan tulee voida antaa myös useita yhtäaikaisia tavoitteita eli kinemaattisia rajoitteita (kinematic constraints).

Tavoitteena on saada ohjaajan käyttöön yhä abstraktimpia ohjausmenetelmiä. Pisimmälle vietyinä ohjausmenetelmät perustuvat käyttäytymismalleihin [BlG95], jotka piilottavat alla olevat nivelikkäät mekanismit korkean tason käyttöliittymän alle. Tällöin nivelikkäiden mekanismien suoran ohjauksen tarve saada saadaan minimoitua.

2.6 Suora kinematiikka

Suoran ohjauksen menetelmää kutsutaan suoraksi kinematiikaksi (forward kinematics). Ketjun jokaiselle nivelelle annetaan jokin asento. Jokaisen ketjussa olevan kappaleen liike ja asento määräytyy suhteessa kappaleen vanhemman koordinaatistoon. Jos vanhempi pyörii origonsa ympäri myös kaikki vanhemman lapsikappaleet pyörivät vanhemman origon ympärillä.

Ohjaamiseen menetelmä on työläs. Halutunlaisen asennon löytäminen voi osoittautua hankalaksi - kun alaraajan on saanut oikeaan asentoon, ohjaaja huomaakin, että se ei aivan yllä kohteeseen, joutuu liikuttamaan yläraajaa ja taas säätämään alaraajaa. Tällä menetelmällä ei myöskään pystytä antamaan rajoitteita kappaleiden liikeradoille: kun lasketaan jalan askellusta avainkuvien välillä, on melko todennäköistä, että jossain ruudussa jalka läpäisee lattian [GiM85].

2.7 Käänteiskinematiikka

Käänteiskinematiikka (inverse kinematics, IK) ratkaisee tavoiteohjatun tehtävän nivelikkäälle mekanismille. Loppuvaikuttajalle annetaan jokin paikka tai asento mihin loppuvaikuttajan tulee pyrkiä. Kun tavoite on annettu, lasketaan ketjun nivelille algoritmisesti sellaiset asennot, jotka mahdollistavat tavoitteen saavuttamisen [WaW92]. Koska ketjun toinen pää on kuitenkin kiinnitetty, tavoitteiseen ei ole aina mahdollista päästä. Jos tavoitetta ei pystytä saavuttamaan, joudutaan tyytymään johonkin ketjun asentoon, joka on mahdollisimman lähellä annettua tavoitetta. Tämä asento voi olla paikallinen minimi, koska globaalin minimin etsiminen vie useimmiten liian kauan aikaa [ZhB94].

2.8 Animointi

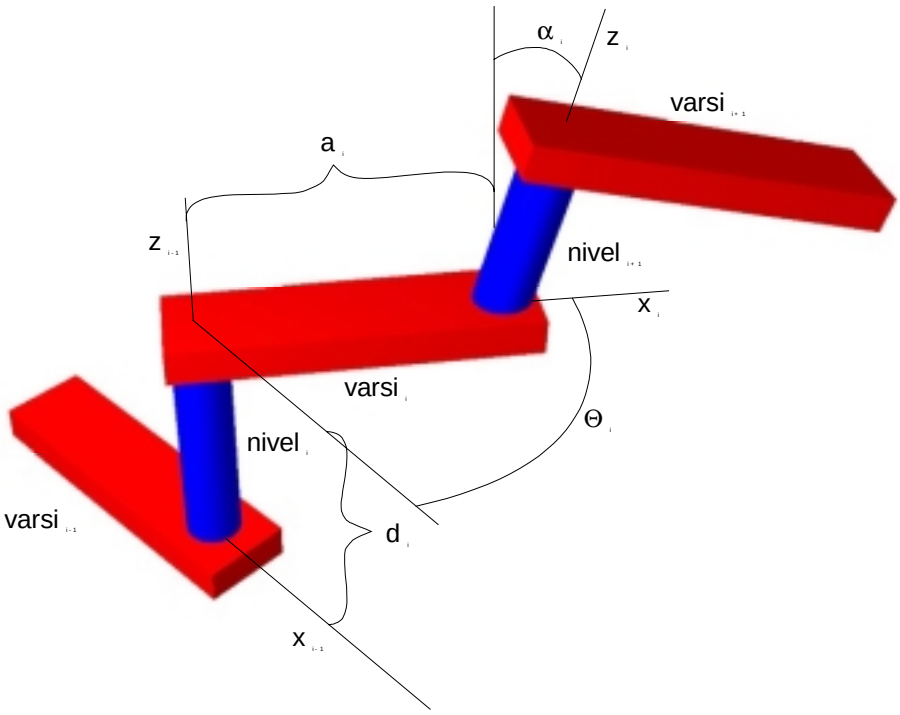
Käänteiskinematiikassa algoritmin tuottama nivelikkään mekanismin loppuasento on ainoa, mitä animaatiossa voidaan käyttää. Asennon etsimisen välivaiheita ei tulisi näyttää. Animaatio saadaan syntymään siten, että jokaisessa aikayksikössä tavoitetta muutetaan hieman ja käänteiskinematiikka-algoritmi laskee uuden asennon. Kun tavoitteen muutos on tarpeeksi pieni aikayksikköön nähden, on lopputulos sulavaa animaatiota. Hyvän animaation aikaansaamiseksi on tärkeää osata valita oikea osa mekanismista loppuvaikuttajaksi [PhB91]. Valintaa voidaan muuttaa animaation aikana. Samoin on tärkeää miten tavoite asetetaan ja miten sitä animoidaan. Mikäli nivelketju jää paikalliseen minimiin nivelten rajoitteiden vuoksi, jää rajoitteiden kiertäminen ohjaajan vastuulle. Yleensä tarvitaan useita yhtäaikaisia rajoitteita realistisen animaation aikaansaamiseksi [BPW93].

3. Nivelmekanismit

3.1 Nivelten esitystavat

3.1.1 Denavit-Hartenberg -määrittely

Denavit-Hartenberg-notaatio (DH) on varhaisimpia kinemaattisia määritelmä nivelikkäille mekanismeille. Se kehitettiin jo 1950-luvulla [WaW92] ja on ollut käytössä etenkin robotiikassa [ZhB94]. DH-notaatiossa käytetään neljää parametriä määrittelemään kappaleen asento vierekkäisten koordinaattijärjestelmien välillä. Kuvassa Kuva 3.1 varren pituutta merkitään *a*:lla, nivelen kiertymää α:lla, nivelen pituutta *d*:llä sekä nivelen asentoa eli kahden varren välistä kulmaa θ:lla [GiM85].



Kuva 3.1 DH-notaatio [GiM85]

DH-määrittelystä saadaan kahden vierekkäisen koordinaattikehyksen i ja i-1 välinen orientaatiomatriisi:

$$ {}^{i-1}T_i = \begin{bmatrix} \vec{n}\,\vec{o}\,\vec{a}\,\vec{p} \\ 0\,0\,0\,1 \end{bmatrix}, \tag{1} $$

missä

$p_x = a_i \cos \theta_i$, $p_y = a_i \sin \theta_i$, $p_z = a_i d_i$,

$n_x = \cos \theta_i$, $n_y = \sin \theta_i$, $n_z = 0$,

$o_x = - \cos \alpha_i \sin \theta_i$, $o_y = \cos \alpha_i \cos \theta_i$, $o_z = \sin \alpha_i$,

$a_x = \sin \alpha_i \sin \theta_i$, $a_y = - \sin \alpha_i \cos \theta_i$ ja $a_x = \cos \alpha_i$ [GiM85].

Kahden minkä tahansa kappaleen i ja j välinen asentomatriisi saadaan laskemalla toistuvasti ketjun vierekkäisten nivelten asentoja [GiM85]:

$$ {}^{i}T_j = {}^{i}T_{i+1} \, {}^{i+1}T_{i+2} \ldots {}^{j-1}T_j. \quad (2) $$

Nollakehys on maailmankoordinaatisto, joten minkä tahansa kappaleen asento saadaan tällä tavalla ilmaistuksi maailmankoordinaatistossa. Myös vapausasteet saadaan määrittelystä tarvittaessa [GiM85].

3.1.2 Akseli-paikka-määrittely

Sims ja Zeltzer kehittivät akseli-paikka-määrittelyn, mikä soveltuu myös haarautuviin nivelhierarkioihin. Määrittelyssä merkitään isäntäkappaleen koordinaatistossa nivelen paikka, nivelen akselin orientaatio, ja nivelen kiertymäkulma. Lisäksi tarvitaan osoittimia. Isäntäkappaleessa on osoitin nivelessä olevaan kappaleeseen (tai useampaan) ja lapsikappaleessa on osoitin isäntään [WaW92].

3.1.3 Nivelten esittäminen tietuetasolla

Badler, Manoochecri ja Walters [BMW87] esittävät nivelten toteutuksen tietorakenteena. Tietorakenne on kuvassa Kuva 3.2. Nivelhierarkiassa on rankasolmu, johon liittyy maailmassa näkyvä varsi, ja nivel jolla solmu liittyy isäntäänsä.

Nivelellä on paikka, origo, joka ilmaistaan isännän koordinaatistossa, sekä nivelen orientaatio, jolla ilmaistaan kappaleiden keskinäinen asento. Sama asia ilmaistaan myös paikallisessa transformaatiomatriisissa isännän koordinaatistosta varren koordinaatistoon. Lisäksi on globaali matriisi, johon talletetaan muunnos maailman koordinaatistosta rankasolmuun. Nivelhierarkian juurisolmun isäntä on maailman koordinaatisto, joten sen paikallinen ja globaali muunnosmatriisi ovat samat. Origoa liikuttelemalla voidaan toteuttaa myös prismaattisia liitoksia.

Varsi koostuu näkyvästä kappaleesta, nimestä ja kosketuspisteestä. Näkyvä kappale on varren pintamalli, joka ilmaistaan varren omassa koordinaatistossa. Kosketuspiste on

paikka varren omassa koordinaatistossa, johon voidaan kohdistaa tavoitteet. Tämä paikka on yleensä varren päässä oleva kohta, jossa nivel sijaitsee. Rankasolmussa oleva kosketuspiste on sama piste ilmaistuna maailmankoordinaateissa.

Maalilla on paikka sekä paino. Paikka on se tavoitepiste maailmankoordinaateissa, johon kosketuspiste yritetään saada. Painolla voidaan säädellä miten tärkeää tavoitteen saavuttaminen on muihin nivelhierarkiassa oleviin tavoitteisiin nähden. Nollapainoinen tavoite jätetään kokonaan huomiotta. On huomattava, että maalilla ei ole tavoitteenaan kappaleen asentoa. Tämä puute on korjattu Badlerin myöhäisemmissä tutkimuksissa [ZhB94].

```
RANKASOLMU {
      RANKASOLMU  *vanhempi;
      int         lastenlkm;
      RANKASOLMU  *lapset[];
      VARSI       varsi;
      NIVEL       nivel;
      MAALI       maali;
      VEKTORI     kosketuspiste;    // Maailman koordinaatistossa
}
NIVEL {
      VEKTORI     paikka;           // Isännän koordinaatistossa
      ORIENTAATIO orientaatio;
      MATRIISI    paikallinen;      // Muunnos isäntäkoordinaatistosta
      MATRIISI    globaali;         // Maailman koordinaatistosta
}
VARSI {
      KAPPALE     pintakappale;
      char        nimi[];
      VEKTORI     kosketuspiste;
}
ORIENTAATIO {
      float       puolitaso, poikkeama, kierto;
}
VEKTORI {
      float       x, y, z;
}
MATRIISI{
      float       elementti[4][4];
}
MAALI {
      VEKTORI     paikka;
      float       paino;
}
```

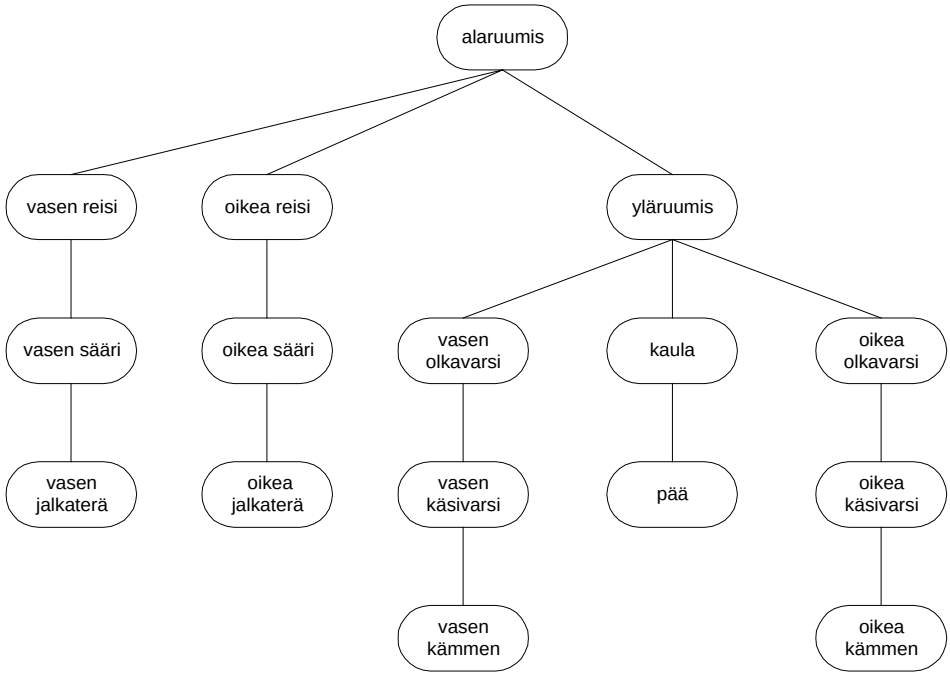
Kuva 3.2 Tietorakenne nivelikkäälle mekanismille [BMW87]

3.2 Nivelhierarkia

Nivelet muodostavat rankaan nivelhierarkian. Hierarkialla on yksi juurisolmu (root node) ja tämän solmun alla olevat solmut ovat juurisolmun perillisiä. Käänteistä kinematiikkaa käytetään ratkaisemaan hierarkian sisällä olevia yksittäisiä ketjuja. Nämä ketjut pitää voida osoittaa hierarkiasta. Aina ei välttämättä haluta mukaan koko ketjua

juuresta lehtisolmuun. Käänteiskinematiikkaa voidaan käyttää kahden minkä tahansa solmun välille valitsemalla tukisolmu (base node) ja loppuvaikuttaja. Loppuvaikuttajan on oltava hierarkiassa suoraan alenevassa polvessa tukisolmun suhteen. Käänteiskinematiikkamoottorin tehtävänä on ratkaista välissä olevien solmujen paikka ja asento. Väliin jääneitä solmuja kutsutaan tyhjiksi solmuiksi (empty node) [WaW92]. Kuvan Kuva 3.3 hierarkiassa juurisolmu on alaruumis, tukisolmuksi voidaan valita vaikkapa oikea käsivarsi ja loppuvaikuttajaksi oikea kämmen. Oikea käsivarsi olisi tällöin tyhjä solmu.

Ketjun juurisolmua pitää voida vaihtaa kesken animaation. Tällä saavutetaan esimerkiksi tilanne, jossa paino siirtyy jalalta toiselle kävelysyklin aikana [PhB91]. Samaan rankaan voidaan asettaa useita tuki-loppuvaikuttaja solmupareja. Mikäli ne ovat päällekkäisiä, niiden vaikutusta lopputulokseen voidaan säätää priorisoinnilla. Korkeammalla prioriteetilla olevat ketjut arvioidaan ensiksi ja alemman prioriteetin ketjut täyttävät tyhjiä solmuja [WaW92]. Mikäli tavoitetta ei voida saavuttaa, korkeamman prioriteetin loppuvaikuttajat pääsevät lähemmäksi tavoitettaan kuin



Kuva 3.3 Kädellisen nivelhierarkia

alemmalla prioriteetilla olevat [BMW87].

3.3 Prototyypissä käytettävä tietorakenne

Nivelhierarkia tarvitsee toimiakseen välttämättä paikan ja asennon maailmassa sekä hierarkiaa varten lapsilistan ja osoittimen vanhempaan. Paikka ja asento esitetään globaaleissa koordinaateissa ja asento talletetaan matriisina. Lisäksi maalin kiinnittämistä varten tarvitaan kosketuspiste loppuvaikuttajan koordinaatistossa. Koska mikä tahansa hierarkian osa voi olla loppuvaikuttaja, talletetaan jokaiselle kappaleelle kosketuspiste.

Loppuvaikuttajan kosketuspiste tulee olla ilmaistavissa myös globaaleissa koordinaateissa. Tämä saadaan kertomalla kosketuspiste kappaleen orientaatiomatriisilla.

Nivelen asento voidaan tarvita suhteessa vanhempaan. Tämä saadaan kahden asennon välisenä erotuksena. Matriisi M_k ilmoittaa kappaleen asennon ja M_v kappaleen vanhemman asennon. Näiden välinen erotus on:

$$M_{delta} = M_{\mathrm{v}}^{\mathrm{T}} M_{\mathrm{k}}, \tag{3}$$

mikä voidaan laskea kaavasta (12).

4. Nivelten ohjausmenetelmät

4.1 Suora kinematiikka

4.1.1 Ongelma

Kolmiulotteisessa avaruudessa olevan jäykän kappaleen paikkaa merkitään vektorilla (x, y, z) ja asentoa vektorilla (ϕ, θ, ψ) [KoB82]. Nämä muodostavat kappaleen tilavektorin $\boldsymbol{\theta} = (x, y, z, \phi, \theta, \psi)$. Yleisesti animoitavan ketjun tilavektori on $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$. Tilavektorin ulottuvuus on sama kuin ketjun vapausasteiden lukumäärä. Suoran kinematiikan ongelmana on ratkaista X, kun $\boldsymbol{\theta}$ on annettu eli

$$X = f(\theta). \tag{4}$$

Loppuvaikuttajan tila X määräytyy epäsuorasti kaikkien ketjun nivelten asentojen kautta [WaW92].

4.1.2 Laskenta

Suoran kinematiikan laskenta on suhteellisen yksinkertaista ja laskennallisesti nopeata. Laskenta redusoituu pisteen pyörittämiseen kolmiulotteisessa avaruudessa vapaavalintaisen akselin ympäri [HeB94]. Vanhempien liike periytyy lapsille. Vaikka suora kinematiikka soveltuu huonosti interaktiiviseen ohjaukseen, se tarjoaa kuitenkin täyden vapauden muokata nivelikästä mekanismia.

Ohjaustapana suora kinematiikka on paljon voimakkaampi kuin pelkkien skriptien käyttämisen asennon määrittelyssä. Avainkuva-animaatiossa käytetään suoraa kinematiikkaa laskemaan avainkuvien väliset asennot interpoloimalla nivelten asentoja. Skripteillä voidaan ohjata nivelmekanismi esimerkiksi toistamaan kävelysykliä.

4.1.3 Esimerkki

Kuvassa 4.1 on esimerkki suorasta kinematiikasta. Kuvassa on kolmen kuvan sarja, missä kädet heiluvat puolelta toiselle. Kumpikin käsi on koottu kolmesta osasta: olkavarresta, käsivarresta ja kämmenestä. Käsivarsi on kiinnitetty olkavarteen ja kämmen käsivarteen. Kaikissa kappaleissa origo vastaa kiinnityspistettä vanhempaan. Kiinnityspiste on määritelty siihen kohtaan, missä ihmisellä on nivel. Kun olkavartta pyöritetään z-akselin (lukijan katseen suuntainen akseli) ympäri, muu käsi seuraa mukana.

Kuva 4.1 Animaatio toteutettuna suoralla kinematiikalla

4.1.4 Kierto vapaavalintaisen akselin ympäri

Asetetaan kolmiulotteiseen avaruuteen akseli *a*, joka kulkee origon kautta. Akselin *a* ympäri tapahtuva kierto kulman θ verran voidaan ilmaista ortogonaalisella matriisilla

$$R(\theta) = R(u)^{-1} R_z(\theta) R(u). \tag{5}$$

R(u) on ortogonaalinen orientaatiomatriisi, jolla akselin *a* suuntainen yksikkövektori *u* kierretään globaalien koordinaattiakselien suuntaisiksi. Vektori *u* on omassa koordinaatistossaan z-akselilla ja kiertämisen jälkeen u asettuu samansuuntaiseksi kuin globaali z-akseli [HeB94]. Normaalisti ohjelmisto säilyttää kappaleen asennon sellaisenaan orientaatiomatriisina *R(u)*, joten se on saatavissa suoraan nivelestä. $R(u)^{-1}$ on tämän matriisin käänteismatriisi, mikä ortogonaalisten matriisien tapauksessa on transpoosi $R(u)^T$ [HeB94]:

$$R(u)^{-1} = R(u)^T. \tag{6}$$

Matriisi

$$R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{7}$$

ilmaisee kulman θ suuruisen kierron z-akselin ympäri.

Aina matriisia *R(u)* ei kuitenkaan ole sellaisenaan saatavissa tai se halutaan muodostaa parametreja antamalla. Yleisesti asento voidaan antaa joko Eulerin kulmina, suunta- ja ylösvektorina tai kvaterniona [WaW92]. Näistä kaikista voidaan muodostaa yllä mainittu orientaatiomatriisi.

Mikäli akseli *a* on ilmaistu Eulerin kulmina ψ,θ,ϕ, eli kiertoina z, y ja x -akselien ympäri (roll, pitch, yaw) [Sho85], niin

$$R(u)=\begin{bmatrix}\cos\theta\cos\phi & \cos\theta\sin\phi & -\sin\theta\\ \sin\psi\sin\theta\cos\phi-\cos\psi\sin\phi & \sin\psi\sin\theta\sin\phi+\cos\psi\cos\phi & \cos\theta\sin\psi\\ \cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi & \cos\psi\sin\theta\sin\phi-\sin\psi\cos\phi & \cos\theta\cos\psi\end{bmatrix}. \tag{8}$$

Jos taas asento on ilmaistu akselin *a* suuntaisena (yksikkö)vektorina u, niin

$$R(u)=\begin{bmatrix}u'_{x1} & u'_{x2} & u'_{x3}\\ u'_{y1} & u'_{y2} & u'_{y3}\\ u'_{z1} & u'_{z2} & u'_{z3}\end{bmatrix}, \tag{9}$$

missä

$$\begin{aligned} u'_z &= u \\ u'_y &= \frac{u\times u_x}{|u\times u_x|} \\ u'_x &= u'_y\times u'_z \end{aligned} \tag{10}$$

ja $u'_x=(u'_{x1}, u'_{x2}, u'_{x3})$, $u'_y=(u'_{y1}, u'_{y2}, u'_{y3})$ ja $u'_z=(u'_{z1}, u'_{z2}, u'_{z3})$ [HeB94].

Kolmas tapa olisi ilmaista asento käyttämällä kvaternioita (quaternion) ja muuttaa kvaternio orientaatiomatriisiksi [Sho85]. Kvaterniot ovat ylivoimaisia interpoloitaessa asentoja avainkuvatekniikassa. Ne myös välttävät Eulerin kulmilla esiintyvän Gimbal lukon [WaW92].

4.1.5 Pisteen kiertäminen akselin ympäri

Kierretään pistettä P orientaatiomatriisilla M_{delta} pisteen O suhteen. Pisteet P ja O on ilmaistu globaaleissa koordinaateissa. Muodostetaan vektori $v = P - O$, mikä ilmaisee siirtymän pisteestä O pisteeseen P. Vektorin v ilmaisema siirtymä kierretään orientaatiomatriisin M_{delta} verran eli kerrotaan vektori v oikealta orientaatiomatriisilla M_{delta}. Lisätään tämä tulovektori pisteeseen O ja näin saadaan pisteen P uusi paikka. Kaavana:

$$P' = O + (P - O) M_{delta}. \tag{11}$$

Tässä piste P voidaan ajatella kappaleen origoksi, joka siirtyy esivanhemman origon O ympäri orientaatiomuutoksen M_{delta} verran, kun esivanhempi on kiertynyt tämän verran.

Kappaleen orientaatio muuttuu myös matriisin M_{delta} ilmoittaman muutoksen verran. Tämä saadaan kertomalla kappaleen orientaatiomatriisi M_k oikealta matriisilla M_{delta}:

$$M'_k = M_k M_{delta}. \tag{12}$$

Kiertymämatriisi M_{delta} ja esivanhemman origo O välitetään edelleen kaikille kappaleen lapsille, jotka puolestaan kiertävät paikkaansa ja asentoaan kaavojen (11) ja (12) ilmaisemilla tavalla.

4.2 Käänteiskinematiikka

4.2.1 Ongelma

Käänteiskinematiikan ongelmana on ratkaista θ, kun X on annettu [WaW92], toisin sanoen:

$$\theta = f^{-1} (X). \tag{13}$$

Kun loppuvaikuttajan tila tunnetaan, pitää selvittää kaikki ketjun nivelten paikat ja asennot. Kun tilavektorin ulottuvuus kasvaa, järjestelmä muuttuu alimääritetyksi. Tällöin siinä on redundanttisuutta eli sama lopputilan asento voidaan saavuttaa useilla nivelten asennoilla. Tilannetta voidaan kuitenkin helpottaa asettamalla nivelille liikerajoitteita (constraints). Esimerkiksi kyynärnivel taipuu vain asentoihin, joissa sen kulma on välillä 20º - 180º. Järjestelmän tulee tuottaa vastaukseksi vain yksi nivelten asento. Yksi tyypillisimmistä rajoitteista on, että nivelen loppuasennon tulee olla mahdollisimman lähellä vanhaa asentoa. Tällä pyritään minimoimaan siirtymiseen tarvittava energia.

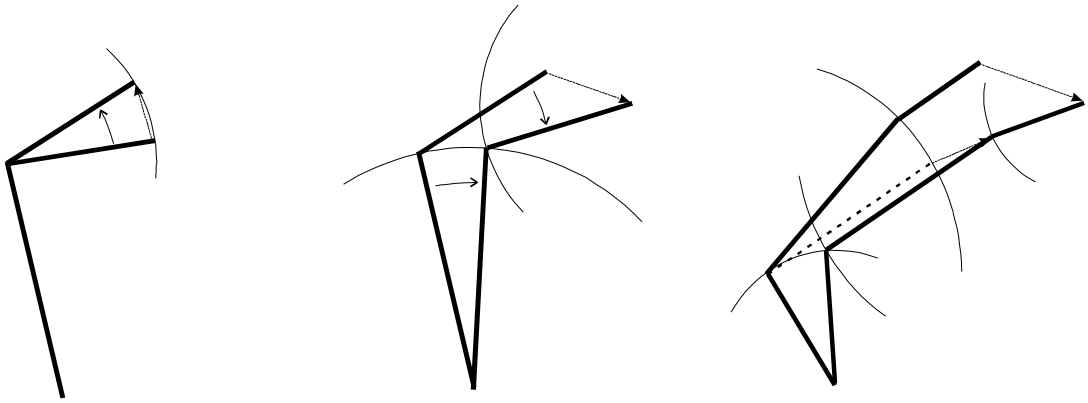
4.2.2 Käänteiskinematiikan yleiset ratkaisumahdollisuudet

Käänteiskinematiikan ongelman ratkaisu on epälineaaristen yhtälöryhmien ratkaisua. Ongelma voidaan ratkaista joko suljetussa muodossa, iteroimalla käyttäen Jacobin matriisin pseudokäänteismatriisia, minimointiongelmana tai deformoitavana kappaleena [ZhB94].

4.3 Käänteiskinematiikan ratkaisu rekursiivisesti

Kirjallisuudessa viitataan yksinkertaisen rekursiivisen ratkaisumahdollisuuden olemassaoloon. Badler ja Manoochechri ja Walters [BMW87] mainitsevat käyttävänsä kolmiepäyhtälöä ja ahnetta algoritmia, mutta muuta tietoa ei anneta. Ratkaisun sanotaan käyttäytyvän huonosti nivelten liikerajoitteiden kanssa ja siitä onkin luovuttu myöhäisemmissä töissä voimakkaampien algoritmien hyväksi [ZhB94].

Kehitin itse algoritmin rekursiiviseen ratkaisemiseen. Rekursiivisen ratkaisun idea on esitetty kuvassa 4.2. Maalina käytetään vain paikkaa. Mikäli ensimmäinen varsi ketjun päässä voi yltää maaliin, kierretään vain ensimmäistä niveltä. Mikäli vartta pitää myös liikuttaa, liikutetaan vartta (ja nivelen paikkaa). Lasketaan varrelle ja sen edeltäjälle kohtauspaikka. Mikäli paikka on olemassa, liikutetaan varsi kohtauspaikkaan ja asetetaan se katsomaan maaliin sekä kierretään varren edeltäjää katsomaan kohtauspaikkaan. Kuvassa tämä on tapaus, jossa toinen nivel yltää maaliin. Mikäli kohtauspaikkaa ei ole, annetaan tarvittava liike edelleen rekursiivisesti maalina varren edeltäjälle. Näin jatketaan aina ketjun juureen asti. Mikäli juurtakin jouduttaisiin vielä siirtämään, ei maali ole tavoitettavissa ja rekursiossa paluuarvona palautetaan tieto epäonnistumisesta. Tällöin jo toteutetut varren liikuttamiset pitää purkaa.



Ensimmäinen nivel yltää maaliin | Toinen nivel yltää maaliin | Kolmas nivel yltää maaliin

Kuva 4.2 Rekursiivinen ratkaisu

4.3.1 Algoritmi

Algoritmi puoliohjelmana:

```
Liikuta_rekursiivisesti (Paikka maali, Indeksi i) {
Jos (i > 0) {
      nivel := nivelketju[i]
      Aseta_katsomaan_kohti(nivel, maali)
      etäisyys_maaliin := maali – nivel.kosketuspiste
      Jos (etäisyys_maaliin < epsilon) {
            // Maali löydetty
            Palauta tosi
      } muuten {
            Liikuta(nivel, etäisyys_maaliin) // Nivel liikkuu maaliin
            kohtaavat := Laske_kohtauspaikka(
                  nivelketju[i-1], nivel, kohtauspaikka)
            Jos (kohtaavat) {
                  Liikuta(nivel, kohtauspaikka)
                  Aseta_katsomaan_kohti(nivel, maali)
                  Aseta_katsomaan_kohti(nivelketju[i-1],
                        kohtauspaikka)
                  Palauta tosi
            } muuten {
                  Jos (Liikuta_rekursiivisesti(
                              kohtauspaikka, i-1) ) {
                        Liikuta(nivel, kohtauspaikka)
                        Aseta_katsomaan_kohti(nivel, maali)
                        Palauta tosi
                  } muuten {
                        // Maali ei tavoitettavissa
                        // palauta nivel alkuperäiseen paikkaansa
                        Liikuta(nivel, -etäisyys_maaliin)
                        Palauta epätosi
                  }
            }
      }
} muuten
      Palauta epätosi
}
```

Algoritmissa on tosiarvoinen funktio `Laske_kohtauspaikka`, joka laskee kahden nivelen (tai oikeastaan varren) välisen kohtauspaikan. Kohtauspaikka on kolmiulotteisessa avaruudessa kahden pallon leikkauksella. Kohtauspaikka voidaan laskea esimerkiksi iteroimalla. Iteraatiossa muutetaan nivelten asentoja katsomaan vuoron perään toistensa joko paikkaan (`nivel`) tai kosketuspisteeseen (`nivel-1`). Laskettu kohtauspaikka palautetaan muuttujassa `kohtauspaikka`. Mikäli nivelet eivät voi kohdata, kohtauspaikkaan on kuitenkin laskettu paikka, jolla nivel olisi lähinnä edeltäjäänsä.

Funktio `Aseta_katsomaan_kohti` asettaa nivelen katsomaan kohti annettua maalia. Funktio on yleinen ”Look at”-algoritmi [HeB94], jonka periaate on esitetty kaavoissa 9 ja 10. Rekursiivinen algoritmi saa lähtöparametreikseen alkuperäisen maalin paikan ja nivelketjun pään eli loppuvaikuttajan.

4.4 Käänteiskinematiikan ratkaisu Jacobin matriisilla

4.4.1 Jacobin matriisi

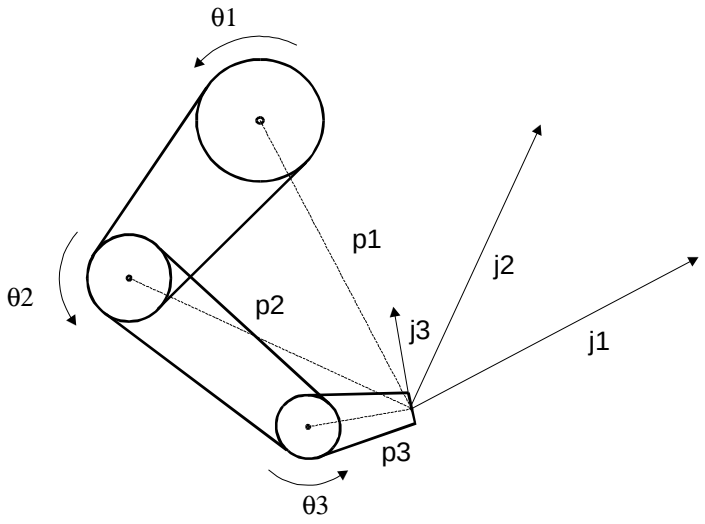
Loppuvaikuttajan nopeuden X suhde nivelten kulmanopeuksiin θ on

$$\dot{X} = J\dot{\theta}, \tag{14}$$

missä J on nimeltään Jacobin matriisi [Mac90]. Matriisi voidaan esittää sarakevektoreina, joka on kolminivelisessä kaksiulotteisessa (Kuva 4.3) tapauksessa:

$$J = [\, j_1\ j_2\ j_3].$$

Tässä sarake j_i esittää nivelen i tuottamaa liikevaikutusta loppuvaikuttajaan. Vaikutus on verrannollinen nivelen kulmanopeuteen θ_i yksikköympyrällä. Vaikutuksen suuruus saadaan laskettua asettamalla vektori p_i nivelen i pyörimisakselista loppuvaikuttajaan. j_i ja p_i ovat samansuuruisia mutta kohtisuorassa toisiaan vastaan. Tämä suhde laajennetaan kolmiulotteiseksi kertomalla p_i ristitulolla nivelen i pyörimisakseliin asetetulla yksikkövektorilla [Mac90]. Kuten huomataan, J muuttuu koko ajan. Se pitää muodostaa uudelleen jokaisessa simulaatioaskeleessa.



Kuva 4.3 Jacobin matriisin sarakkeiden fyysinen tulkinta [Mac90]

4.4.2 Jacobin matriisin kääntäminen

Yhtälö $\theta = f^{-1}(X)$ saadaan periaatteessa ratkaistua muodostamalla Jacobin matriisin käänteismatriisi. Nivelten asennot nivelhierarkiassa saadaan loppuvaikuttajan liikkeestä yhtälöllä:

$$d\theta = J^{-1}(dX)\,. \tag{15}$$

Koska niveliä voi olla mielivaltainen määrä ja jokaisella nivelellä on esimerkiksi kuusi vapausastetta, niin Jacobin matriisi ei yleensä ole neliömatriisi. Ja vain neliömatriisilla on käänteismatriisi. Yhtälölle (15) saadaan kuitenkin ratkaisu käyttämällä pseudokäänteismatriisia [GiM85].

Merkitään Jacobin matriisin J pseudokäänteismatriisia J^+:lla. Tällä on seuraavanlaiset ominaisuudet:

$$\begin{aligned} &J\,J^+\,J = J \\ &J^+\,J\,J^+ = J^+ \\ &(J^+\,J)^T = J^+\,J \\ &(J\,J^+)^T = J\,J^+\,. \end{aligned} \tag{16}$$

Pseudokäänteismatriisi tuottaa pienimmän neliösumman suoran ratkaisun (least squares minimun norm solution) yhtälölle (15). Siten se tuottaa ratkaisun myös yli- että alimääritetyissä tapauksissa.

Pseudokäänteismatriisi voidaan laskea täyttä astetta r oleville m * n -matriiseille kaavalla:

$$A^+ = \begin{cases} (A^T A)^{-1} A^T, \text{ kun } m > n = r, \text{ ja} \\ A^T (A\,A^T)^{-1}, \text{ kun } r = m < n. \end{cases} \tag{17}$$

Yleensä yhtälö (15) kuitenkin ratkaistaan käyttämällä Gaussin eliminointia vaihtokaaviolla, joten varsinaista pseudokäänteismatriisia ei tarvitse muodostaa [Väl87].

4.5 Ongelmat käänteiskinematiikassa

Käänteiskinematiikassa ongelmina ovat redundanttiset asennot [WaW92] ja yleisesti laskennan raskaus. Ongelmana on myös Jacobin matriisissa esiintyvät singulariteetit, jotka johtuvat pahoista asennoista [Mac90] sekä se, että Jacobin matriisi ei yleensä ole neliömatriisi, jolloin ratkaisuun joudutaan käyttämään pseudokäänteismatriisia [GiM85]. Lisäksi numeeriset virheet vaivaavat yleensä kaikkea realistiseen simulointiin pyrkiviä ohjelmia.

4.5.1 Numeeriset virheet

Simulaation laskemisessa pitää aina varautua numeeriseen epätarkkuuteen. Prototyypissä tästä on huolehdittu kahdella tavalla: pitämällä riittävän lähellä olevia lukuja samoina vertailun avulla (etenkin nollaan verrattaessa) sekä huolehtimalla kappaleiden asentomatriisien ortonormaaliudesta [Gems].

Pseudokäänteismatriisi tuottaa vain paikallisia muunnoksia. Sen laskeminen liian suurella maalin etäisyydellä tuottaa varmasti liian suuria muutoksia niveliin. Tätä varten tavoitetta pitää lähestyä iteratiivisesti [WaW92].

4.5.2 Laskennan raskaus

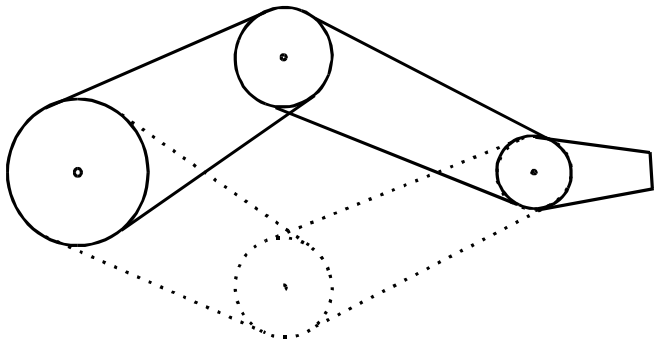
Jacobin matriisin muodostaminen jokaisella simulaatioaskeleella ja matriisin kääntäminen tuottaa laskentaan reaaliaikaisuutta rajoittavaa raskautta. Pseudokäänteismatriisin tuottaminen on vaativuudeltaan luokkaa $O(n^2)$, kun n on vapausasteiden määrä [PZB90]. Laskenta tulee entistä raskaammaksi, mikäli tavoitteeseen joudutaan iteroimaan.

4.5.3 Matriisi ei ole neliömatriisi

Nivelketjut ovat yleensä pitkiä ja loppuvaikuttajan vapausaste on suurempi kuin kuusi. Tällöin J ei ole neliömäinen eikä siten käännettävissä. Matriisin kääntämiseen voidaan käyttää pseudokäänteismatriisia [GiM85].

4.5.4 Redundanttiset asennot

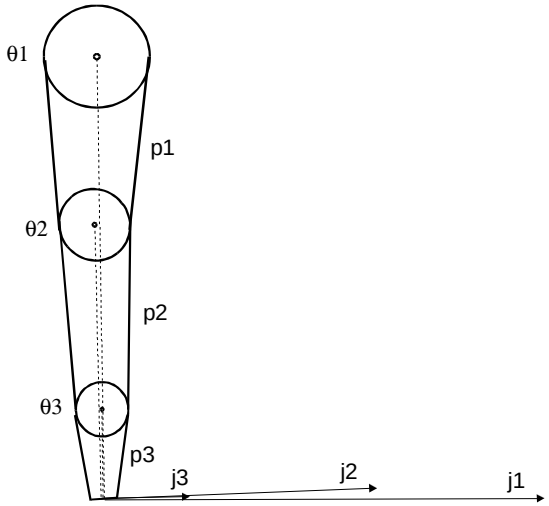
Redundanttinen asento (Kuva 4.4) syntyy, kun nivelketju voi yltää lopputulokseen kahdella tai useammalla nivelten asennolla. Näin yleensä onkin. Redundanttisuus johtaa siihen, että J on alimääritetty. Redundanttisuutta voi vähentää asettamalla nivelille jäykkyyksiä [GiM85] tai mukavuus-rajoitteita (comfort) [BPW93]. Nivel pyrkii valitsemaan sen asennon, jossa se on lepoasennossa.



Kuva 4.4 Redundanttinen asento

4.5.5 Singulariteetti

Mikäli raaja venytetään suoraksi, sen nivelet osuvat samalle linjalle (kuva 4.5). Tällöin nivelten liike ei ole enää toisistaan riippumattomia. Nivelet ovat tulleet lineaarisesti riippuvaisiksi eli J on singulaarinen [Mac90]. Tällöin menetetään yksi tai useampi vapausaste – raaja ei voi liikkua ketjun kiinnityskohtaa lähemmäksi tai poispäin. Vapausasteita menetetään myös silloin, kun mitkä tahansa ketjun osat sattuvat samalle linjalle toistensa kanssa.

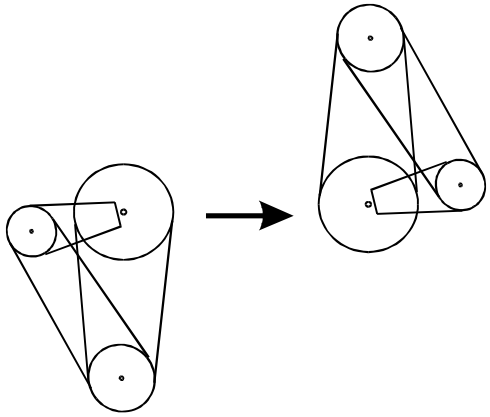


Kuva 4.5 Raaja lähestyy singulaarista asentoa [Mac90]

Singulariteetin voi havaita siitä, että J:n determinantti on 0 (mikäli J on neliömäinen ylipäänsäkään). Singulariteetin voi välttää poikkeuttamalla raajan asentoa hieman täydestä ojennuksesta [WaW92]. Toisaalta pseudokäänteismatriisi tuottaa ratkaisun myös singulaarisille matriiseille. Maciejevski käyttää singulaarisen arvon osiinjakamisen menetelmää (singular value decomposition) singulariteetin välttämiseksi [Mac90].

4.5.6 Pahat asennot

Pahalla asennolla (ill conditioning) tarkoitetaan sellaista asentoa (Kuva 4.6), joka tuottaa suuria liikkeitä nivelketjuun, kun loppuvaikuttaja liikkuu vain hieman [Mac90]. Pahat asennot aiheutuvat mekanismeista itsestään, eikä niitä voi helposti välttää. Pahimmassa tapauksessa nivelketju oskilloi kahden ratkaisun välillä. Yksi keino välttää niitä, on minimoimalla siirtymiseen tarvittava energia [ZhB94].



Kuva 4.6 Paha asento

4.6 Käänteiskinematiikan toteutus Jacobin matriisilla

4.6.1 Jacobin matriisin rakentaminen

Nivelikkään mekanismin n+1:stä ei-prismaattisesta varresta {{0}, {1},…, {n}} koostuvan loppuvaikuttajan nopeus V_{n0} ja kulmanopeus Ω_{n0} {0}-kehyksestä katsottuna liittyvät nivelten paikallisiin kulmanopeuksiin suhteessa vanhempiin Jacobin matriisin kautta [WaW92]:

$$\begin{bmatrix} V_{n0} \\ \Omega_{n0} \end{bmatrix} = \begin{bmatrix} b_{x1}, b_{y1}, b_{z1}, ..., b_{xn-1}, b_{yn-1}, b_{zn-1} \\ a_{x1}, a_{y1}, a_{z1}, ..., a_{xn-1}, a_{yn-1}, a_{zn-1} \end{bmatrix} \begin{bmatrix} \theta_{x1} \\ \theta_{x1} \\ \theta_{x1} \\ \vdots \\ \theta_{xn-1} \\ \theta_{xn-1} \\ \theta_{xn-1} \end{bmatrix} \tag{18}$$

Tässä (a_{xi}, a_{yi}, a_{zi}) ovat ketjun i:nnen kappaleen x, y, z –akselit muunnettuina {0}-kehykseen eli maailmankoordinaatistoon. Ne saadaan suoraan kappaleen i asentomatriisin kolmesta ensimmäisestä rivistä. Lasketaan vektori $P = P_n - P_I$, missä P_n on ketjun loppuvaikuttaja ja P_i on kappaleen i paikka maailmankoordinaatistossa. Lasketaan edelleen kukin vektori *b* kertomalla *a* ristitulolla *P*:n kanssa:

$$\begin{aligned} b_{xi} &= a_{xi} \times (P_n - P_i) \\ b_{yi} &= a_{yi} \times (P_n - P_i) \\ b_{zi} &= a_{zi} \times (P_n - P_i) \end{aligned} \tag{19}$$

Tässä *a*- ja *b*-vektorien muodostama matriisi on Jacobin matriisi [WaW92], kuten yhtälö (16) ilmaisee. Sen muoto on 6 x (3*n), kun n on ketjun nivelten lukumäärä.

4.6.2 Algoritmi

Lasketaan maalin ja loppuvaikuttajan etäisyyden ja asennon erotus ja derivoidaan ne saadaksemme nopeuden ja kulmanopeuden. Muodostetaan Jacobin matriisi kuten yllä kuvattiin. Lasketaan Jacobin matriisin pseudokäänteismatriisi kaavan (17) mukaan. Nivelten kulmanopeudet θ saadaan kaavan (15) mukaan. Nopeudet integroidaan [Gim85], eli nivelen uusi kiertokulma saadaan kaavasta $\varphi = \varphi_0 + \omega_0 t$, missä φ_0 on alkuperäinen kiertokulma, ω_0 on kulmanopeus (eli θ) ja t on simulaatioaskeleen aika [Sep95]. Koko nivelketjulle saadaan uudet asennot kuten kappaleessa 4.1.5 selitetään. Laskeminen voidaan tehdä myös pelkillä paikan ja asennon erotuksilla, jolloin

vastauksena saadaan nivelten asennon eroja. Vastaavasti laskenta voidaan tehdä myös korkeamman kertaluvun derivaatoilla kuten kiihtyvyydellä [Mac90].

Ratkaisualgoritmi C++-syntaksia mukaellen puoliohjelmana:

```
solve_goal_once(goal) {
int n = chain.size();
VectorN goal_delta[6];
VectorN joints_delta[n*3];
MatrixMN jacobian[6,n*3];
// Calculate goal and end effector difference
vec3 position_delta = goal.get_position() –
      chain[n-1].get_touchpoint_in_world();
mat3 orient_delta = (chain[n-1].get_orientation()).transpose() *
      goal.get_orientation();
vec3 orientation_delta = matrix_to_euler_angles(orientation_delta);
goal_delta = VectorN(position_delta, orientation_delta);
Pn = chain[n-1]->get_touchpoint_in_world();
// Construct Jacobian
for (int i = 0; i < n; i++) {
   vec3 axi, ayi, azi, bxi, byi, bzi;
      mat3 matrix = chain[i]->get_orientation();
   ax = matrix.row(1);
   ay = matrix.row(2);
   az = matrix.row(3);
      vec3 P = chain[i]->get_position();
   bx = ax ^ (Pn - P); // Crossproduct of ax and (Pn - P)
   by = ay ^ (Pn - P);
   bz = az ^ (Pn - P);
   jacobian.column[i*3] = VectorN(bx,ax);
   jacobian.column[i*3+1] = VectorN(by,ay);
   jacobian.column[i*3+2] = VectorN(bz,az);
}
// Invert Jacobian with pseudoinverse and calculate joint angle deltas
joints_delta = jacobian.inverse() * goal_delta;
// Apply joint angles to chain
for (int i = 0; i < n; i++) {
      vec3 euler_angles = joints_delta[i*3];
   mat3 or_delta = euler_angles_to_matrix(euler_angles);
   chain[i]->add_orientation(or_delta);
}
}
```

4.6.3 Iterointi

Ratkaisun laskeminen on virhealtista ja paikallista. Mikäli maali on liikkunut liian pitkän matkan, nivelten tuottama asento ei osu maaliin. Tämän takia maalin etsintä joudutaan tekemään iteratiivisesti. Mikäli asento ei ole riittävän lähellä maalia, asennon muutos hylätään, maali puolitetaan ja yritetään osua tähän uuteen puolimaaliin. Mikäli osutaan, jatketaan puolimaalista alkuperäiseen maaliin tai jaetaan maali edelleen rekursiivisesti puoliksi [WaW92].

Iteraatioalgoritmi C++-syntaksia mukaellen puoliohjelmana:

```
iterate(goal){
int iterations = 1, sub_iterations = 0;
current_goal = final_goal = goal;
save_old_joints();
previous_tracking_error_to_final_goal =
      calculate_tracking_error(final_goal);
while (iterations <= iteration_limit) {
      set_goal_delta(current_goal);
      solve_goal_once(current_goal);
      tracking_error = calculate_tracking_error(current_goal);
      if (tracking_error < tracking_error_epsilon) {
      // See, if we reached the final goal
            tracking_error = calculate_tracking_error(final_goal);
            if (tracking_error < tracking_error_epsilon) {
                  // Achieved final goal.
            return;
      }
      // Still needs iterations
      if (tracking_error <= previous_tracking_errorto_final_goal) {
        // Iteration neared to final goal.
        // Continue iteration from this point to final goal
        previous_tracking_error_to_final_goal = tracking_error;
        iterations++;
        sub_iterations = 0;
        save_old_joints();
        current_goal = final_goal;
      } else {
         // Iteration did not near to final goal.
         // Retract joints and halve goal.
         sub_iterations++;
         retract_joints();
         halve_goal(current_goal);
         if (sub_iterations > sub_iteration_limit) {
            // Subiteration limit exceeded. No point to continue...
            iterations += iteration_limit;
            sub_iterations = 0;
         }
      }
   }
   else {
      // Did not reach goal. Retract and halve goal.
      sub_iterations++;
      retract_joints();
      halve_goal(current_goal);
      if (sub_iterations > sub_iteration_limit * iteration_limit) {
         // Iteration limits exceeded. No point to continue
         iterations++;
         sub_iterations = 0;
      }
   }
}
// Iteration limit exceeded, goal not reached. Might retract joints.
}
```

4.7 Käänteiskinematiikka minimointiongelmana[1]

Zhao ja Badler esittävät käänteiskinematiikan minimointiongelmana ja ratkaisevat ongelman epälineaarisella ohjelmoinnilla [ZhB94]. Heidän tekniikkansa on käytössä ihmishahmojen interaktiivisessa manipuloinnissa Jack-ohjelmistossa [Jack]. Järjestelmässä maalia ja loppuvaikuttajaa käsitellään erillisinä toisistaan. Rajoitteita voi olla useita yhtä aikaa ja ne ratkaistaan painojensa mukaisissa suhteissa. Nivelillä voi olla liikerajoitteita ja jäykkyys.

4.7.1 Loppuvaikuttajan kuvaaminen

Loppuvaikuttajan tila kuvataan yhdeksänulotteisena (9D) vektorina **e**, mikä koostuu paikka-, suunta- ja ylöspäinvektoreista. Periaatteessa riittää tallettaa paikka ja suuntakulmat, kuten kappaleessa 4.1 on esitetty. Loppuvaikuttaja voidaan asettaa tavoitteiden kuten paikan, suunnan tai kiertokulman tai näiden yhdistelmien alaisiksi. Loppuvaikuttajan liike on samalla tavalla nivelistä riippuvaista kuin kappaleessa 4.4.1. Riippuvuutta kuvataan Jacobin matriisilla

$$\frac{\partial \mathrm{e}}{\partial \theta} = \left(\frac{\partial e}{\partial \theta_1} \frac{\partial e}{\partial \theta_2} \cdots \frac{\partial e}{\partial \theta_n} \right)$$

Merkitään **v** = loppuvaikuttajan (yksikkö)suuntavektori, **r** = loppuvaikuttajan paikka, **u** = nivelen kiertymäakseli. Koska ihmisellä on vain kiertoliikkeeseen kykeneviä niveliä, niin järjestelmä ei käsittele mäntämäisiä niveliä. Ne on kuitenkin tarvittaessa kiertoniveliä helpompi toteuttaa.

4.7.2 Maalin kuvaaminen

Maali kuvataan potentiaalifunktiona, jonka lähtöalue pitää olla sama kuin loppuvaikuttajan maalialue. Potentiaalifunktio mittaa loppuvaikuttajan etäisyyttä (paikka tai asento) maalista ei-negatiivisena reaalilukuna. Merkitään potentiaalifunktiota P(**x**):llä. Funktion gradienttia merkitään $\nabla_{\mathbf{x}}\mathrm{P}(\mathbf{x})$:llä ja se muodostetaan vektorin **x** osittaisderivaatoista. Gradientti on useamman muuttujan skalaarifunktion muutoksen suuruutta ilmaiseva vektorifunktio. Jos funktio f on differentioituva, niin f kasvaa nopeimmin gradientin osoittamaan suuntaan [Gro95].

[1] Kappale 4.7 perustuu Zhaon ja Badlerin lähteeseen [ZhB94] milloin ei muuta ilmaista.

4.7.3 Maalityypit

Maaleina esitetään paikka, asento, paikka ja asento, tähtäys, viiva, taso ja puoliavaruus. Paikkamaali määritellään paikkana P kolmiulotteisessa avaruudessa. Sen lähtöalue on R^3. Potentiaalifunktio on $P(\mathbf{r}) = (\mathbf{p} - \mathbf{r})^2$ ja gradientti $\nabla_{\mathbf{r}} P(\mathbf{r}) = 2(\mathbf{p} - \mathbf{r})$.

Asentomaali on pari ortonormaaleja vektoreita $\{\mathbf{x}_g, \mathbf{y}_g\}$, missä alaviite g viittaa maaliin (goal). Lähtöalue on tällöin $S^2 * S^2$, missä S^2 on 3D yksikkövektori. Potentiaalifunktio on $P(x_e, y_e) = c^2_{dx} (x_g - x_e)^2 + c^2_{dy} (y_g - y_e)^2$, missä alaviite e viittaa loppuvaikuttajaan (end effector). Tässä c_d on skaalaustekijä, jolla yksi pituusyksikkö saadaan vastaamaan d-asteista kulmaa. Merkitään $c_d = 360/(2\pi d)$. Gradientti on $\nabla_{\mathbf{xe}} P(x_e, y_e) = 2c^2_{dx} (x_e - x_g)$ ja $\nabla_{\mathbf{ye}} P(x_e, y_e) = 2c^2_{dy} (y_e - y_g)$. Asentomaalissa voidaan jättää toinen maali (suunta- tai ylöspäinvektori) rajoittamattomaksi asettamalla vastaava c_d nollaksi.

Edelliset maalit voidaan yhdistää, jolloin niiden vaikutus potentiaalifunktioon painotetaan. Vastaavalla tavalla kuin edellä määritellään muutkin maalit. Tähtäysmaalissa esimerkiksi ihminen saadaan katsomaan johonkin pisteeseen. Viivamaalissa loppuvaikuttajan pitää osua jollekin viivalle, tasomaalissa tasolle. Puoliavaruutta voidaan käyttää esimerkiksi törmäysten välttämiseen, rajoittamalla loppuvaikuttaja vaikkapa seinän toiselle puolelle.

4.7.4 Tavoitefunktio

Loppuvaikuttajan etäisyyttä maaliin nykyisillä nivelten asennoilla θ kutsutaan tavoitefunktioksi ja se merkitään $G(\theta) = P(\mathbf{e}(\theta))$. Haluttaisiin ratkaista yhtälö $G(\theta) = 0$. Tämä ei ole kuitenkaan aina mahdollista, koska maali ei ole välttämättä saavutettavissa. Yritetään minimoida $G(\theta)$. Suurella osalla nivelistä on liikerajoitteita – pienin ja suurin kulma, johon nivel voi taipua. Nämä rajoitteet ilmaistaan lineaarisina epäyhtälöinä. Esitetään ongelma epälineaarisena ohjelmointiongelmana, jossa on lineaarisia rajoitteita muuttujina:

$$\begin{cases} \text{minimoi} & G(\theta), \\ \text{kun} & a^T_i\theta = b_i,\ i = 1,2,\ldots,l \\ & a^T_i\theta \le b_i,\ i = l+1,l+2,\ldots,k, \end{cases} \tag{20}$$

missä a_i, i=1,2,…,k ovat sarakevektoreita, joiden ulottuvuus on sama kuin θ:jen. Yhtälöt mahdollistavat lineaarisia suhteita nivelten välillä, kuten olkapäässä. Epäyhtälöt mahdollistavat pienimmän ja suurimman kulman esittämisen nivelille.

4.7.5 Ratkaisun laskenta

Epälineaarisen ohjelmoinnin avulla löydetään useimmiten paikallinen minimi. Tämä on yleensä riittävä. Mikäli asento ei ole tyydyttävä, jokin toinen paikallinen minimi voidaan saavuttaa toisella yrittämällä alkuasentoa hieman muuttamalla. Paikalliseen minimiin on tyytyminen myös siksi, että ohjelma pysyisi interaktiivisena.

Ongelma ratkaistaan Davidonin muuttuvan metriikan menetelmällä. Menetelmässä käytetään BFGS (Broyden, Fletcher, Goldfarb, Shanno) toisen asteen ylläpitosääntöä. Muuttujien lineaariset rajoitteet käsitellään Rosenin projektiomenetelmällä [BPW93]. Laskemisessa käytetään hyväksi potentiaalifunktiota sekä sen gradienttia. Laskenta on lineaarisesti riippuvaista nivelten lukumäärästä ja nivelketjulle asetettujen maalien lukumäärästä. Ratkaisu löytyy iteratiivisesti. Jokaisella iteraatiolla minimiä etsitään tietystä suunnasta. Gradientti otetaan huomioon useammalta iteraariokierrokselta. Menetelmä lähestyy aina monotonisesti minimiä. Menetelmä on globaalisti konvergoiva eli se löytää aina jonkin (paikallisen) ratkaisun oli lähtöpiste mikä tahansa.

4.7.6 Useat yhtäaikaiset rajoitteet

Yleensä käänteiskinematiikka on ratkaistu evoluutiomaisesti alkuasennosta lähtien. Tämä algoritmi löytää vastauksen koko hakuavaruudesta. Tämän vuoksi tarvitaan useita yhtäaikaisia rajoitteita. Muuten esimerkiksi kyynärpää voi heittelehtiä villisti sivulle, kun kämmentä liikutetaan. Tavoitefunktio G(θ) on ei-negatiivinen, joten useat yhtäaikaiset rajoitteet voidaan ratkaista minimoimalla kaikkien maalien

$$G^{kaikki}(\theta) = \sum_{i=1}^{m} w_i G_i(\theta), \tag{21}$$

tavoitefunktioiden summa

missä m on rajoitteiden lukumäärä, i viittaa rajoitteeseen ja w_i on negatiivinen painokerroin. $G^{kaikki}(\theta)$ korvaa G(θ):n kaavassa 19.

5. Nivelikkäät mekanismit animaatio-ohjelmistossa

5.1 TrueSpace

Truespace on Caligari-yhtiön [Cal97] kaupallisesti saatavilla oleva kolmiulotteinen animaatio-ohjelmisto Microsoft Windows-ympäristöön. Siinä on animaation ohjaamiseen toteutettu muun muassa käänteiskinematiikka ja suora dynamiikka. Ohjelmassa on intuitiivinen käyttöliittymä ja ohjelman käyttö on kohtuullisen helppo oppia verrattuna useisiin ammattilaisten käyttöön tarkoitettuihin ohjelmistoihin kuten 3D Studio Max tai Alias/Wavefront. Ohjelmasta jaetaan Internetissä tutustumisversiota, jonka perusteella on seuraavassa esitetty nivelikkään mekanismin käyttö ja toteutus ohjelmassa. Esittelyohjelma on ominaisuuksiltaan kuten kaupallinen tuote. Ainoastaan animaatiomaailman tallentaminen on estetty ja renderöitävien animaatioiden kuvakoko on rajattu.

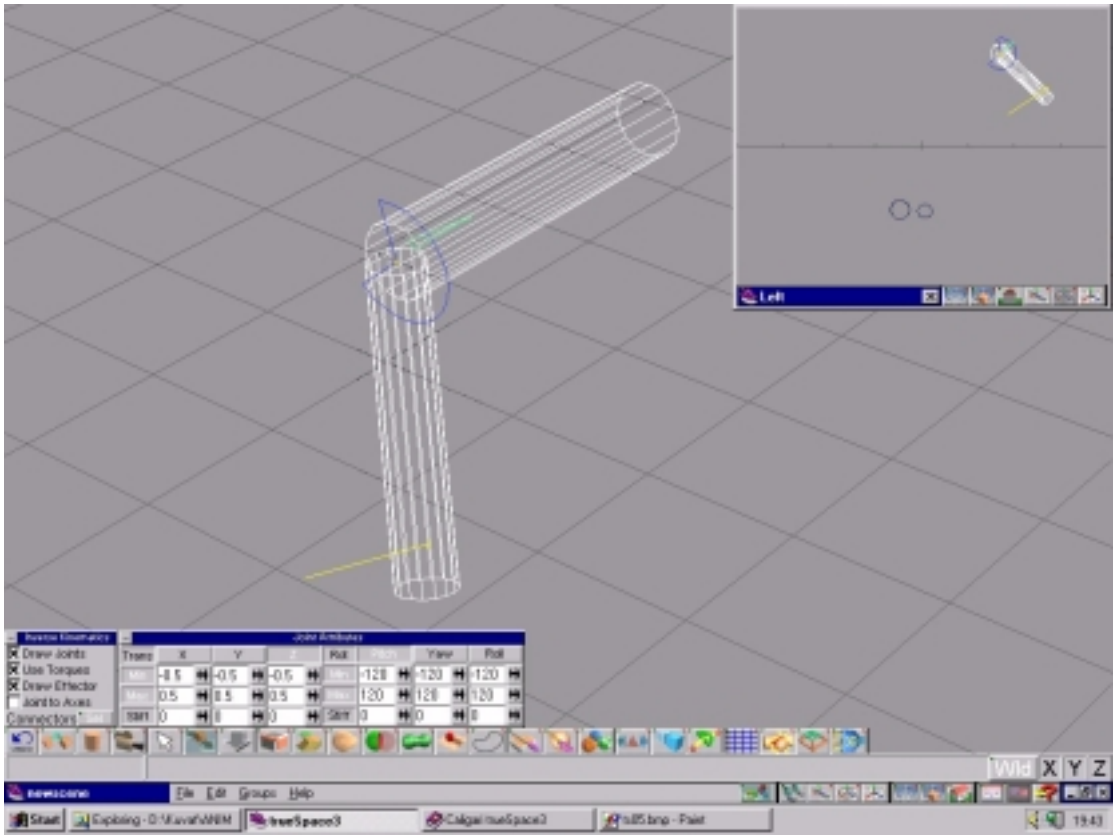
5.2 Nivelikkäiden mekanismien tekeminen

5.2.1 Nivelen lisäys ja asennon muutos

Ohjelmassa voi muodostaa nivelikkäitä mekanismeja. Osoitetaan ensin lapsikappale, valitaan nivelen tyyppi ja osoitetaan isäntäkappale. Uusi nivel muodostetaan näiden kappaleiden yhtymäkohtaan. Kuvassa 5.1 on ohjelman käyttöliittymä sekä yksi nivelikäs kappale. Nivelikästä kappaletta voi ohjata tarttumalla lapsikappaleeseen ja vetämällä sitä. Ohjaus toteutuu kappaleeseen kohdistuvana voimavektorina, joka ilmaistaan graafisesti ruudulla keltaisena viivana.

5.2.2 Nivelhierarkia

Kappaleita lisäämällä voi muodostaa ketjun tai nivelhierarkian. Hierarkiassa on jokin kappale, joka on kiinnitetty maailmaan. Kiinnitetyksi kappaleeksi voi valita minkä tahansa kappaleen hierarkiasta ja sitä voi vaihtaa. Aliketjun saa tällöin tarkkaan kontrolliin ilman, että koko hierarkian asento muuttuisi. Monimutkaisia kappaleita saa rakennettua myös siten, että samassa kappaleessa on useita kinemaattisia osia ja siten useita kiinnitettyjä kappaleita. Ketjua voi ohjata tarttumalla mihin tahansa kappaleeseen ketjussa ja vetämällä kappaletta. Nivelketjun välissä olevien kappaleiden asento lasketaan käänteiskinematiikalla.



Kuva 5.1 Truespace käyttöliittymä ja nivel

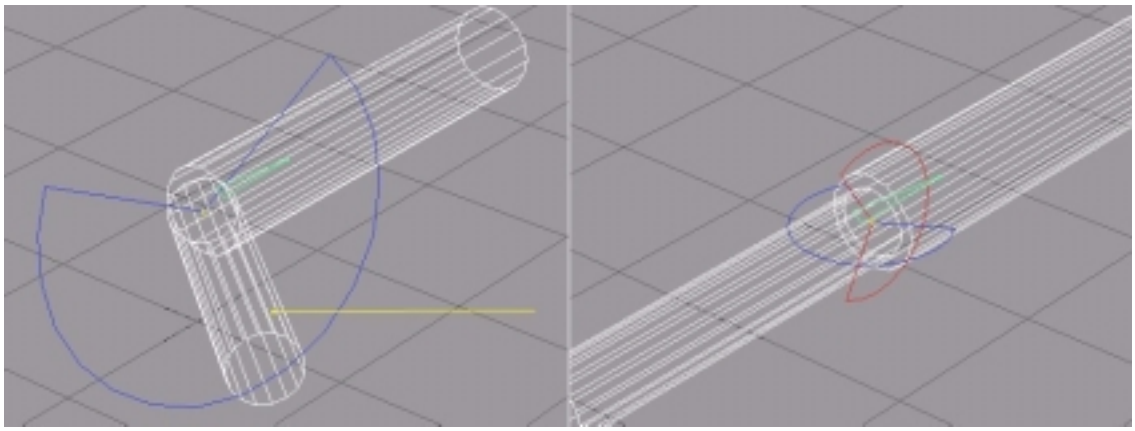
5.2.3 Nivelten ominaisuudet

Erilaisia niveltyyppejä ovat 0d - kiinteä liitos, 1d mäntäliitos, 1d saranaliitos, näiden yhdistelmä, 2d pallonivel, 2d tasoliitos sekä vapaasti määriteltävä liitos. Nivelen paikkaa ja ominaisuuksia voi muuttaa manipuloimalla suoraan niveltä. Nivelen ominaisuudet kuvataan ruudulla graafisesti ja niitä voi muuttaa suoraan hiirellä käsittelemällä tai dialogista. Kuvan 5.2 vasemman puoleisessa kuvassa on nivelen jäykkyyttä kasvatettu. Myös nivelen liikerataa voi rajoittaa. Saranaliitos voi olla välillä ±360° ja mäntäliitos ±10 yksikköä suhteessa nivellinkkiin. Nivelestä kappaleen keskipisteeseen on linkki, joka on nivelen pyörimisakseli (roll). Muut akselit ovat tätä kohtisuorassa kuvan 5.2 oikealla puolella olevassa pallonivelessä.

5.2.4 Animointi

Ohjelmassa animointi toteutetaan avainkuvatekniikalla. Kappaleelle asetetaan käänteiskinematiikalla jokin alkuasento johonkin tiettyyn animaation kellonaikaan ja tämä asento nauhoitetaan. Samoin asetetaan loppuasento. Animaatio voidaan esikatsella ruudulla ennen renderöintiä. Esikatselu voi olla toteutettu joko viivagrafiikalla tai Gouraud-varjostuksella. Animaation välikuvissa nivelhierarkian asento lasketaan

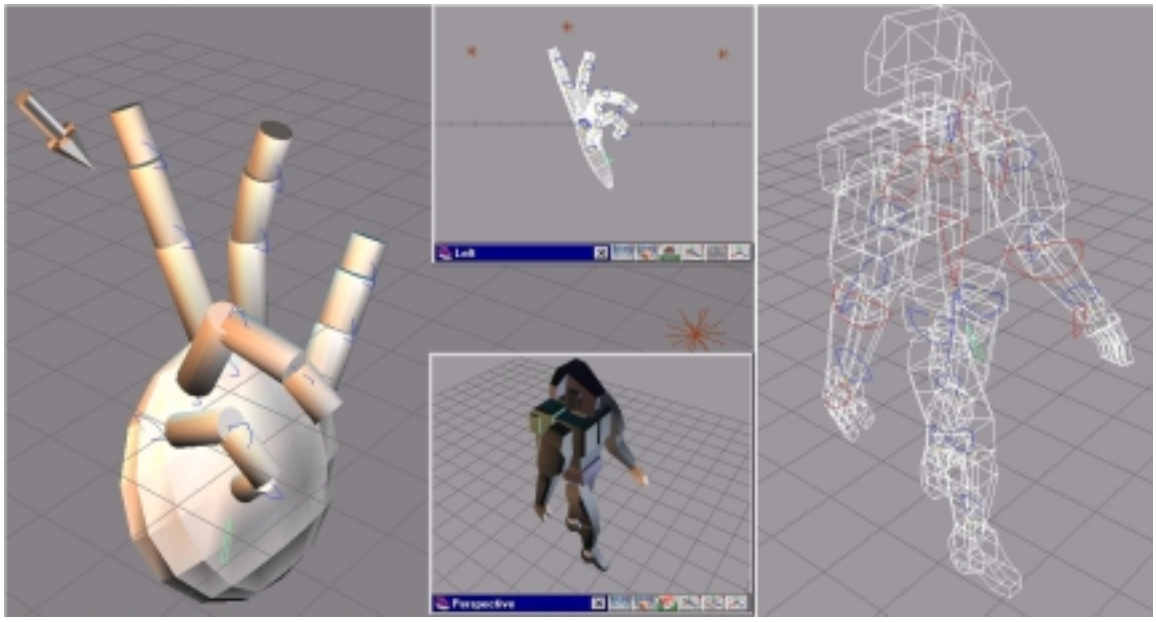
interpoloimalla nivelten asento. Kappaleiden manipulointi ruudulla on sinänsä interaktiivista kolmiulotteista animaatiota ja tällöin nivelhierarkian asento lasketaan jatkuvasti käänteiskinematiikalla. Manipulointi on erittäin interaktiivista, eikä ohjelmassa saa tehtyä niin monimutkaista nivelikästä kappaletta, etteikö sitä pystyttäisi päivittämään ruudulle reaaliajassa.



Kuva 5.2 Nivelen manipulointi ja pallonivel

5.3 Käyttötapausesimerkki

Liitteessä 1 on toteutettu animaatio kädestä, joka puristuu nyrkkiin ja avautuu. Animaatio on talletettu avi-muotoisena oheiselle CD:lle. Animaatio on toteutettu ohjelman mukana tulleella valmiilla kappaleella. Käsi on alkuasennossaan kuvassa 5.3 vasemmalla puolella. Samassa kuvassa on oikealla puolella robotti esimerkkinä monimutkaisemmasta nivelhierarkiasta.



Kuva 5.3 Käsi- ja robottikappaleet

6. Sovelluskohteet

6.1 Ihmisen animointi

Käänteiskinematiikkaa on menestyksellisesti käytetty ihmisen animointiin. Yksi kehittyneimmistä järjestelmistä on Norman I. Badlerin Pennsylvanian yliopistossa ohjaaman ryhmän kehittämä Jack-animointijärjestelmä [Jack]. Järjestelmä kykenee erittäin realistiseen ihmisen reaaliaikaiseen simulointiin. Käytetyssä mallissa on muun muassa ihmisestä mallinnettuja niveliä 88 kappaletta, lihaksista mallinnettu voimantuotto, törmäyksen tunnistus ja välttäminen. Malli osaa kävellä ja pitää yllä tasapainoa, kurottaa ja pitää kiinni. Järjestelmä on täysin ohjelmoitavissa korkean tason ohjelmointikielillä. Jack on käytössä muun muassa Yhdysvaltojen armeijalla simulointi-tehtävissä ja sitä voi käyttää vaikkapa ergonomisen työympäristön tutkimiseen.

Phillips, Zhao ja Badler [PZB90] ratkaisevat useita yhtäaikaisia kinemaattisia rajoitteita. Phillips ja Badler [PhB91] esittävät reaaliaikaisen interaktiivisen tasapainon säilyttävän hahmon. Badler, Hollick ja Granieri [BHG95] käyttävät käänteiskinematiikkaa virtuaali-todellisuushahmon esittämiseen, kun ainoat syötteet käyttäjältä on enää pään asento, kämmenien asento ja lantion asento. Käänteiskinematiikalla lasketaan reaaliaikaisesti koko muun vartalon asento virtuaalitodellisuudessa.

Ihmisen tai eläimen animoinnissa on otettava erityisesti huomioon luuston rajoitukset nivelten liikeratoihin. Rajoituksia käsitellään tällöin omina kinemaattisina rajoituksina. Toisaalta esimerkiksi nivelten jäykkyys auttaa redundanttisten vapausasteiden ongelmassa [ZhB94]. Lisäksi on huomattava, että selkäranka ei ole kiinteä kappale vaan muovautuu huomattavastikin asennon mukaan [MoB91]. Kattavin selvitys ihmisen animoinnissa kohdattavista haasteista ja ratkaisuista on kirjassa Badlerin, Phillipsin ja Webberin kirjassa Simulating Humans [BPW93].

6.2 Kävely

Kävelysyklin määrittely on yksi yleisimpiä sovelluskohteita käänteiskinematiikalle. Yksi animointijärjestelmä, jossa kävely on toteutettu on PODA-systeemi [GiM85 ja Gir87]. Girard ja Maciejevski [GiM85] esittelevät kävelyyn tarvittavia käsitteitä, käyttävät splinejä jalan loppuvaikuttajan animointiin ja ratkaisevat jalan nivelten asennot käänteiskinematiikalla. Girard [Gir87] suunnittelee interaktiivisesti useampijalkaisten hahmojen askellusta. Watt ja Watt [WaW92] esittelevät kävelyesimerkin, joka käyttää suoraa kinematiikkaa.

Interaktiivisen manipuloinnin ja animoinnin lisäksi käänteiskinematiikalla on myös muita sovelluskohteita. Käänteiskinematiikkaa käytetään saavutettavissa olevan työtilan määrittämiseen tai törmäyksen välttämiseen [ZhB94].

7. Kokeellinen osuus

7.1 Prototyyppi

7.1.1 Prototyypin perusta

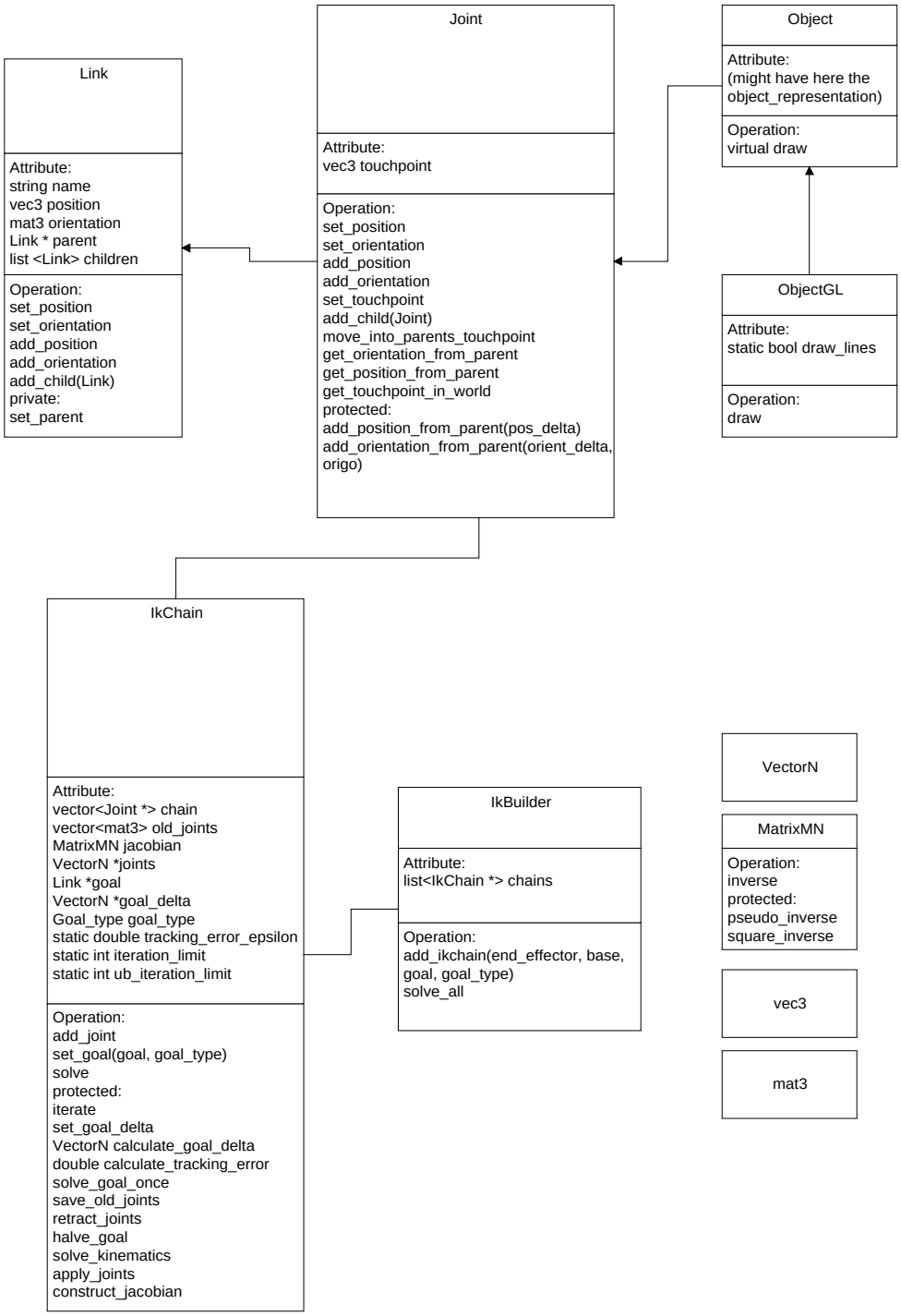
Teorioiden havainnollistamiseksi on työssä rakennettu prototyyppi nivelikkäistä mekanismeista. Ohjelma on tehty C++-ohjelmointikielellä [Str97] oliosuuntautuneesti [Rum91]. Ohjelman luokkakaavioon (kuva 7.1) on kuvattu tärkeimmät toteutetut luokat ja niiden keskeiset ominaisuudet ja menetelmät. Prototyypin luokat kommentteineen sekä testiympäristön käyttöliittymä on toteutettu englannin kielellä.

Ohjelma käyttää valmista alikirjastoa kolmiulotteisten perusprimitiivien esittämiseen. Tällaisia ovat muun muassa kolmialkioinen vektori (vec3) paikan esittämiseen ja 3*3 matriisi (mat3) asennon esittämiseen. Alikirjasto on otettu käyttöön Graphics Gems-kirjasarjasta [Gems] ja siihen on tehty itse pieniä muutoksia.

Työssä on toteutettu itse luokat vapaaulotteisten vektorien (VectorN) ja matriisien (MatrixMN) esittämiseen. Etenkin tässä, mutta muuallakin työssä, on nojauduttu C++ Standard Template Library (STL) [STL] alikirjastoon muun muassa vektorien ja listojen esittämisessä. Vapaaulotteisen neliömatriisin käänteismatriisin laskemista varten on toteutettu Gaussin eliminointi vaihtokaaviolla algoritmi kirjasta Numerical Recipies in C [PFT92]. Lisäksi on toteutettu asennon määrittelyssä helpottavia apufunktioita kvaternioita varten kirjasta Advanced Animation and Rendering Techniques [WaW92] ja Computer Graphics [HeB94].

7.1.2 Nivelten esittämiseen tarkoitetut luokat

Nivelikkäät mekanismit koostuvat perusluokista varsi (Link) ja nivel (Joint), joka periytyy varresta. Nämä luokat muodostavat rangan (skeleton) perustan, jota voidaan animoida. Lisäksi on luokka kappale (Object), joka periytyy nivel-luokasta. Tähän luokkaan voitaisiin ajatella liitettäväksi kappaleen ulkomuodon esitys (”liha”). Toteutuksessa tämä luokka on kuitenkin virtuaalinen. Nivelketju-olio (IkChain) muodostetaan kappaleista ajonaikaisesti ketjunrakentaja-olion (IkBuilder) avulla. Nivelketju osaa ratkaista itse sille annetun maali. Se sisältää keskeisimmän osan käänteiskinematiikan toteutuksesta. Kaikki tähän mennessä luetellut luokat ovat riippuvaisia ainoastaan ohjelman mukana seuraavista lähdekoodeista, standardi C++:sta sekä STL-kirjastosta.



Kuva 7.1 Luokkakaavio

7.1.3 Testiympäristön toteutus

Nivelikkäiden mekanismien havainnollistamiseksi on toteutettu kolmiulotteinen testiympäristö. Testiympäristö käyttää OpenGL-ohjelmointirajapintaa grafiikan piirtämiseen [OpenGL]. Käyttöliittymärutiinit hoidetaan OpenGL AUX-kirjaston avulla.

Näillä ohjelmointikirjastoilla testiympäristön pystytys on helppoa ja ohjelmasta tulee useisiin käyttöjärjestelmiin siirrettävä.

Testiohjelmisto on käännetty Borland C++-kääntäjällä Microsoft Windows NT 4.0 käyttöjärjestelmässä. Pelkän kääntäjän ilman kehitysympäristöä saa nykyisin vapaasti Internetistä [Borland]. Käännetty testiohjelmisto toimii tarvittavien dynaamisten linkkikirjastojen (DLL) avulla missä tahansa 32-bittisessä Windows-käyttöjärjestelmässä. Ohjelmisto on myös siirretty SGI Unix-käyttöjärjestelmän alle, missä se toimi yhtä hyvin.

Liitteenä olevalla CD-ROM-levyllä on testiympäristön ajokelpoinen Windows-versio oheistiedostoineen. Lisäksi mukana on lähdekoodi sekä projektitiedostot Borland C++ kääntäjää varten ja Makefile-tiedosto Unix-käyttöjärjestelmää varten.

7.1.4 OpenGL:n osuus

Testiympäristössä on myös luokka ObjectGL, joka tukeutuu OpenGL-funktioihin piirtääkseen itsensä. Piirtämisessä käytetään yksinkertaista viivapiirtoa tai näennäistä kartiota rangan havainnollistamiseksi. Testien pääohjelma sisältää nivelketjujen rakentamisen, testitapaukset ja käyttöliittymän. Nämä osat ovat ainoita, jotka ovat riippuvaisia OpenGL-rajapinnasta.

7.2 Kokeet

Testeissä on keskitytty käänteiskinematiikan testaamiseen. Suora kinematiikka sisältyy implisiittisesti toimivaan käänteiskinematiikkaan, koska nivelten asennot muutetaan suoralla kinematiikalla. Testeissä on muodostettu nivelketju, jossa on kahdesta viiteen niveltä. Lisäksi on maali, johon ketjun loppuvaikuttajan pitää pyrkiä. Tavoitteeksi voidaan asettaa maalin paikka, asento tai molemmat. Maali voi liikkua, kiertyä tai tehdä molempia yhtäaikaa.

Testiympäristössä (kuva 7.2) nivelketjun juuri sijaitsee maailman origossa. Ketjua ympäröi viivoilla piirretty kuutio, jonka kolmelle särmälle projisoidaan maalin ja loppuvaikuttajan paikka. Testiympäristö käyttää oikeakätistä koordinaatistoa. Koordinaatiston akselit on piirretty myös kuvaan siten että RGB-värit vastaavat

Kuva 7.2 Testiympäristö

akseleita XYZ.

Testit ajetaan oletusasetuksin, paitsi että ohjelman tulostustoiminnon astetta (debug) on nostettu yhdellä. Tällöin ohjelma tulostaa jokaisella maalin liikutusaskeleella (yksi/ruutu), joutuuko se käyttämään iteraatiota ja saavutetaanko maali. Oletuksena on käyttää Jacobin matriisin iteratiivista ratkaisutapaa ja iteraatioiden maksimimäärä on 16. Maaliin osutaan, kun loppuvaikuttajan etäisyys maalista on pienempi kuin 0,5.

7.2.1 Testit 1-5

Testit 1-5 ovat oikeastaan kaksiulotteisia testitapauksia. Niissä maali ja kaikki nivelet sijaitsevat samalla ortogonaalisella tasolla. Ensimmäisessä testissä (kuva 7.2) on kolmen nivelen ketju. Ketju sijaitsee XY-tasolla. Maali liikkuu ympyräradalla samalla tasolla origon ympäri. Maalina käytetään paikkamaalia. Maalin paikka on aluksi hieman

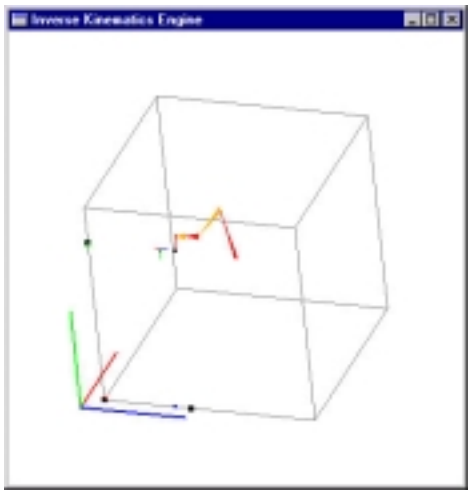
erillään loppuvaikuttajasta. Nivelketju saavuttaa maalin heti kahdella iteraatiolla ja seuraa sen jälkeen maalia ilman iteraation tarvetta heti ensimmäisellä ratkaisukerralla.

Testissä 2-4 on viiden nivelen ketju. Toinen testi on muuten samanlainen kuin ensimmäinen. Tämä testi osoittaa myös käänteiskinematiikan toimivan. Kolmas testi poikkeaa toisesta vain siten, että maalina on myös loppuvaikuttajan asento ja maalin asentoa muutetaan vähitellen. Neljäs ja viides testi ovat muuten samanlaisia kuin toinen testi, paitsi että neljännessä testissä ketju ja maali ovat XZ-tasossa ja viidennessä testissä ne ovat YZ-tasossa. Kaikki nämä testit tuottavat oikean lopputuloksen ilman iteraatioiden tarvetta ja testit toimivat nopeasti ja tehokkaasti.

7.2.2 Testit 6-10

Testeissä 6-10 maalit sijaitsevat kolmiulotteisessa avaruudessa, joten niveletkin joutuvat tekemään kiertoliikkeitä kaikkien kolmen akselinsa ympäri. Näissä testeissä käytetään viisinivelistä ketjua. Testissä kuusi on maalina vain maalin paikka. Maali lähtee kiertämään akselia (1, 1, 0) myötäpäivään hyvin pienin askelin (0,1°). Aluksi ketju pysyy hyvin maalissa, eikä iteraatiota tarvita. Kuitenkin jo parinkymmenen asteen liikkumisen jälkeen ei maalia saavuteta iteroimallakaan – ei vaikka iteraatiomäärää suurentaisi huomattavastikin. Nivelketju ei tunnu pystyvän liikkumaan kolmessa ulottuvuudessa maaliin - maali jatkaa menojaan ja ketju jähmettyy paikalleen. Kyseessä ei kuitenkaan ole singulariteetti tai se, ettei maali ole tavoitettavissa. Syytä epäonnistumiseen ei ole löydetty ohjelmasta tai lähteistä.

Seitsemäs testi (kuva 7.3) on muuten sama kuin kuudes, mutta maalina on myös asento jota muutetaan saman verran kuin paikkaa. Tämä testi tuottaa epäonnistumisen vielä aikaisemmin kuin kuudes testi. Yhdeksäs testi on sama kuin kuudes, mutta maali liikkuu suuremmin askelin.



Kuva 7.3 3D testi epäonnistuu maalin tavoittelussa

Kahdeksannessa testissä maalina on maalin paikka ja asento, mutta tällä kertaa pelkästään maalin asento muuttuu. Tässä testissä nivelketju ei pysy maalin perässä kovin kauaa. Lopulta ketjun pää alkaa loitontua maalista, lähentyen silloin tällöin takaisin maalia. Kymmenennessä testissä maali liikkuu vain yhdellä puolella origoa XZ-tasoa ajatellen. Maalina on paikka ja asento ja kumpaakin muutetaan. Ketju ei pysy maalin perässä myöskään tässä testissä.

7.2.3 Testit 11-12 ja 13

Testeissä 11 ja 12 maalina on vain maalin paikka. Maali liikkuu kohti jotain ennalta arvottua pistettä suoralla radalla. Maalin liikettä voidaan verrata hiirellä ohjaamiseen tai siihen, että maalin pitäisi liikkua alkupaikasta johonkin loppupaikkaan ja ketjun animoitua mukana. Testissä 11 käytetään kolmen nivelen ketjua. Ketju ei aina suoriudu tehtävästä, mutta toisinaan ketju pysyy hyvin maalin mukana. Testi 12 on samanlainen, mutta viiden nivelen ketjulle.

Testi 13 on kahden nivelen testi. Tällöin ketjulla on kuusi vapausastetta eli Jacobin matriisi on neliömäinen. Jacobin matriisi on kuitenkin singulaarinen eikä nykyinen ohjelmisto käytä pseudokäänteismatriisia tämän tehtävän ratkaisuun.

7.2.4 Testit yksinkertaisella rekursioratkaisulla

Testit voidaan ratkaista myös käyttämällä yksinkertaista rekursiomenetelmää. Rekursiomenetelmä ratkaisee ongelman vain maalin paikan suhteen. Lisäksi Aseta_katsomaan_kohti-algoritmin nykytoteutus ohjelmassa vaatii, että kappaleen kosketuspisteen on sijaittava z-akselilla. Näin on vain testeissä 5 ja 10-13. Näissä kaikissa menetelmä tuottaa oikean lopputuloksen testeille. Silloin tällöin ketjun jotkin nivelet kuitenkin liikkuvat suuren määrän toiseen redundanttiseen asentoon, jolloin ketjun liike näyttää häiritsevän nykivältä.

Testissä 12 nähdään hyvin, miten menetelmä tuottaa erilaisen ratkaisun Jacobin matriisiin verrattuna. Siinä missä Jacobin matriisi –menetelmä liikuttaa yleensä kaikkia ketjun niveliä yhtä aikaa, rekursiomenetelmä liikuttaa yleensä vain ketjun päässä lähtien niveliä yhtä kerrallaan. Kun loppuvaikuttaja ei enää yllä tavoitteeseen, vastuu siirtyy sitä edeltävälle ja niin edelleen. Lopputulos muistuttaa renkaista tehtyä ketjua, jota vedetään toisesta päästä. Tähän voisi nivelten jäykkyys tuoda parannuksen.

Testiohjelmassa on myös mahdollisuus käyttää molempia ratkaisumenetelmiä yhtä aikaa. Tällöin rekursiomenetelmään turvaudutaan vasta jos iteraatiomenetelmä ei tuota tulosta. Lisäksi rekursiomenetelmää koskevat edellä mainitut rajoitukset.

7.3 Johtopäätökset

Jacobin matriisia käyttävä ratkaisumenetelmä toimii erinomaisesti kaksiulotteisissa tapauksissa. Kolmiulotteisissa tapauksissakin toteutettu menetelmä tuottaa toisinaan oikean ratkaisun, mutta yleensä ketju ei pysty seuraamaan maalia. Yksinkertainen rekursiivinen ratkaisumenetelmä tuottaa erikoisehtojensa alla lähes poikkeuksetta oikean lopputuloksen. Menetelmän löytämä asento on kuitenkin vähemmän luonnollinen kuin Jacobin matriisin menetelmällä löydetty. Molemmat ovat riittävän nopeita menetelmiä interaktiiviseen asennon määrittelyyn. Niveliä on harvoin tarve lisätä ketjuun niin monta, että interaktiivisuus olisi uhattuna.

Kuten esimerkkinä käytetty kaupallinen ohjelma todentaa, käänteiskinematiikka on käyttökelpoista ja toimivaa interaktiivisessa asennon määrittelyssä. Tieteellisissä piireissä käänteiskinematiikkaa tunnutaan pidettävän ratkaistuna ongelmana. Pariin viimeiseen vuoteen alan saralla ei ole ilmestynyt uusia merkittäviä tieteellisiä artikkeleita. Ongelmat ovat lähinnä toteutuksessa.

8. Jatkotutkimuskohteita

Yleensä käänteiskinematiikka ei sinällään riitä nivelikkäiden mekanismien ohjaukseen. Tavoitteena on yhä autonomisempia virtuaalisia olioita. Olioille on voitava asettaa dynaamisen mallinnuksen ominaisuuksia kuten massa ja vääntö [BPW93]. Oliota on voitava ohjata käyttäytymissääntöihin perustuen. Maalin animointi perustuu heuristiikkaan ja tekoälyn merkitys korostuu abstraktimmalle tasolle noustaessa. Suurin osa reaaliaikaisista toteutuksista esimerkiksi peleissä perustuu edelleenkin väliasentojen tallettamiseen ja suoraan kinematiikkaan näiden asentojen välillä. Tällöin haasteena on sulavat siirtymät näiden asentojen välillä.

9. Yhteenveto

Nivelikkäiden mekanismien määrittely ja toteutus on kohtuullisen helppoa, kun animointiin käytetään suoraa kinematiikkaa. Ongelmana on käänteiskinematiikka. Siihen ei ole löydetty yhtä tehokasta - tai helppoa - ratkaisua. Jokainen sovellusalue tarvitsee oman erityisen ratkaisunsa. Käänteiskinematiikan toteutus on hyvin vaativaa. Käänteiskinematiikka on viime vuosina tullut interaktiiviseksi ja animaatio-ohjelmistojen perusominaisuudeksi. Sitä käytetään harvoin reaaliaikaisesti esimerkiksi peleissä, koska se ei aina tuota haluttua ratkaisua ja laskemiseen käytettävä kapasiteetti on kuitenkin rajallista. Edelleen suurin osa reaaliaikaisista toteutuksista turvautuu asentojen väliseen interpolointiin ja suoraan kinematiikkaan. Näin saavutetaankin usein riittävä realismi.

Lähteet

Bad87 Badler, N. I., Articulated figure animation. IEEE Computer Graphics and Applications 7, 6 (1987), 10-11.

BHG95 Badler, N. I., Hollick, M. J., Granieri, J. P., Real-Time Control of a Virtual Human Using Minimal Sensors. The Journal of Virtual Reality and Teleoperators 2, 1 (1995), 82-86.

BlG95 Blumberg, B. M., Galyean, T. A., Multi-level Direction of Autonomous Creatures for Real-time Virtual Environments. Proc. Computer Graphics , (1995), 47-54.

BMW87 Badler, N. I., Manoochehri, K. H., Walters, G., Articulated figure positioning by multiple constraints. IEEE Computer Graphics and Applications 7, 6 (1987), 28-38.

Borland -, Borland C++ kääntäjä. http://www.borland.com, 2000.

BPW93 Badler, N. I., Phillips, C. B., Webber, B. L., Simulating Humans. Oxford University Press. New York, 1993.

Cal97 Caligari, TrueSpace 3D animaatio-ohjelmisto. http://www.caligari.com, 1997.

Car96 Carmack, J. et al., Quake, versio 1.06. http://www.idsoftware.com,

Chi96 Chin, K. W., Honours research proposal on animating human motion using an inverse kinematic engine. http://www.cs.curtin.edu.au/~chinkw/html/proposal/proposal.html,

Gems A. Glassner, P. Heckbert, Graphics Gems I-IV, ftp:princeton.edu pub/GraphicsGems. , 1994.

GiM85 Girard, M., Maciejewski, A. A., Computational Modeling for the Computer Animation of Legged Figures. Proc. Computer Graphics 19, 3 (1985), 263-270.

Gir87 Girard, M., Interactive design of 3D computer animated legged animal motion. IEEE Computer Graphics and Applications 7, 6 (1987), 39-51.

Gro95 Grossman, S. I., Multivariable Calculus, Linear Algebra and Differential Equations, third edition. Saunders College Publishing. Orlando, 1995.

HeB94 Hearn, D., Baker, M. P., Computer Graphics, second edition. Prentice Hall, Inc. New Jersey, 1994.

IsC87 Isaacs, P. M., Cohen, M. F., Controlling Dynamic Simulation with Kinematic Constraints, Behaviour Functions and Inverse Dynamics. Proc. Computer Graphics 21, 4 (1987), 215-224.

Jack -, Jack - The Human Modeling and Simulation System. http://www.cis.upenn.edu/~hms/jack.html, 1997.

KoB82 Korein, J., Badler, N. I., Techniques for goal directed motion of articulated structures. IEEE Computer Graphics and Applications 2, 9 (1982), 71-81.

Mac90 Maciejewski, A. A., Dealing with the ill-conditioned equations of motion for articulated figures. IEEE Computer Graphics and Applications 10, 3 (1990), 63-71.

MoB91 Monheit, G., Badler, N. I., A kinematic model of the human spine and torso. IEEE Computer Graphics and Applications 11, 2 (1991), 29-38.

OpenGL -, OpenGL - avoin grafiikan ohjelmointirajapinta. http://www.opengl.org, 2000.

PFT92 Press, W. H., Flannery, B. P., Teukolsky, S. A., Numerical Recipies in C: The Art of Scientific Computing. Cambridge University Press. New York, 1992.

PhB91 Phillips, C. B, Badler, N. I., Interactive Behaviors for Bibedal Articulated Figures. Proc. Computer Graphics 25, 4 (1991), 359-362.

PZB90 Phillips, C. B., Zhao, J., Badler, N. I., Interactive real-time articulated figure manipulation using multiple kinematic constraints. Proc. Computer Graphics 24, 2 (1990), 245-250.

Rum91 Rumbaugh, J. et al., Object Oriented Modeling and Design. Prentice-Hall, Inc. New Jersey, 1991.

Sep95 Seppänen, R. et al., MAOL-taulukot. Otava. Helsinki, 1995.

Sho85 Shoemake, K., Animating Rotation with Quaternion Curves. Proc. Computer Graphics 19, 3 (1985), 245-254.

STL ISO/ANSI, C++ Standard Template Library. http://www.roguewave.com, 1995.

Str97 Stroustrup, B., The C++ Programming Language, Third Edition. Addison Wesley. Reading, Massachusetts, 1997.

WaW92 Watt, A., Watt, M., Advanced animation and rendering techniques: theory and practice. Addison-Wesley. New York, 1992.

Väl87 Väliaho, H., Lineaarialgebra. Yliopistopaino. Helsinki, 1987.