Annemari Auvinen
Milla Törhönen

Kuulasimulaattori

TIE374 Fysikaaliset mallit tietokoneanimaatioissa

Harjoitustyöraportti

28.4.2013

Jyväskylän yliopisto

Tietotekniikan laitos

Sisältö

1 Kuulat ja liikeyhtälöt

Harjoitustyön aiheena oli toteuttaa kuulasimulaattori, jossa voidaan simuloida kuulien käyttäytymistä niiden pudotessa tasoon ja törmätessä toisiinsa. Tässä luvussa kuvataan jäykän kappaleen liikeyhtälöt sovellettuna simulaattorissa käytettyihin kuuliin.

1.1 Kuula

Kuulana käytetään r-säteistä palloa, jonka massa on tasaisesti jakautunut, ja massakeskipiste on pallon keskipiste. Tällaisen kuulan hitaustensori lokaalissa koordinaatistossa on yksinkertaisesti $I_{body} = \frac{2mr^2}{5}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, missä m on kuulan massa ja r säde [4].

1.2 Voimat

Kuulaan vaikuttavana voimana simulaattorissa käytetään painovoimaa $g = [0 \quad 0 \quad -9.80605]$.

1.3 Liikeyhtälöt

Kuulan tilavektori on $Y(t) = \begin{pmatrix} x(t) \\ q(t) \\ P(t) \\ L(t) \end{pmatrix}$,

missä $x(t)$ on kuulan massakeskipisteen sijainti, $q(t)$ kvaterniovektori, $P(t)$ kokonaisliikemäärä. ja $L(t)$ kulmaliikemäärä.

Laskennallisesti näistä saadaan nopeus $v(t) = \frac{P(t)}{m}$, hitaustensori $I(t) = R(t)I_{body}R(t)^T$ ja kulmanopeus $\omega(t) = I(t)^{-1}L(t)$.

Tilavektorin muutos saadaan laskemalla $\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ q(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ 0.5*(\omega(t)*q(t)) \\ F(t) \\ \tau(t) \end{pmatrix}$,

missä *F(t)* on ulkoinen kokonaisvoima ja $\tau(t)$ on ulkoisen voiman aiheuttama vääntömomentti.

1.4 Differentiaaliyhtälöratkaisijat

Simulaattorissa on käytettävissä kolme differentiaaliyhtälöiden ratkaisemiseen tarkoitettua menetelmää: Eulerin menetelmä, keskipistemenetelmä ja 4. kertaluvun Runge-Kutta –menetelmä [8]. Menetelmissä uusi tila lasketaan edellisen tilan avulla alla esitetyillä kaavoilla. Kaavoissa $x_0 = x(t_0)$, h on askelpituus ja funktio $f(x,t) = \frac{d}{dt}Y(t)$.

- Eulerin menetelmä: $x(t_0 + h) = x_0 + hf(x_0, t_0)$.
- Keskipistemenetelmä: $x(t_0 + h) = x_0 + hf(x_0 + \frac{h}{2} f(x_0, t_0), t_0 + \frac{h}{2})$.
- 4. kertaluvun Runge-Kutta:

$$k_1 = hf(x_0, t_0)$$
$$k_2 = hf(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2})$$
$$k_3 = hf(x_0 + \frac{k_2}{2}, t_0 + \frac{h}{2})$$
$$k_4 = hf(x_0 + k_{3,} t_0 + h)$$
$$x(t_0 + h) = x_0 + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

2 Törmäykset

Tässä luvussa esitetään pääpiirteet simulaattorissa käytetystä törmäyksien tarkastelusta ja voiminen laskemisesta. Lähteenä on käytetty Baraffin materiaalia [2], jossa esitettyjä menetelmiä on muokattu kuulasimulaattoriin soveltuviksi.

2.1 Törmäystarkastelu

Kuulasimulaattorissa kuulat voivat törmätä toisiinsa sekä tasoon. Tason sijainti määritellään z-akselille, jolloin törmäystarkastelut siihen helpottuvat. Törmäystarkastelussa tason kanssa käydään jokainen kuula läpi ja tarkastetaan, onko kuulan keskipisteen z-koordinaatti pienempi kuin kuulan säde määritellyn törmäysepsilonin sisällä, jolloin kyseessä on törmäys. Törmäyspisteen x- ja y-koordinaatit saadaan helposti suoraan kuulan keskipisteestä, ja z-koordinaatti on nolla.

Kuulien keskinäinen törmääminen tarkastetaan vastaavalla tavalla, nyt vain tason tilalla on toinen kuula. Kuulat käydään pareittain läpi ja tarkastetaan, onko kuulien keskipisteiden etäisyys pienempi kuin kuulien säteiden summa törmäysepsilonin rajoissa. Kuulien välinen törmäyspiste määritellään kuulien keskipisteiden puoliväliin.

Mikäli törmäys tapahtuu, lasketaan suhteellinen nopeus törmäyspisteessä. Jos nopeus on lähellä nollaa, on kyseessä kontakti, jolloin pitää laskea kappaleisiin vaikuttavat kontaktivoimat. Nopeuden ollessa negatiivinen, kappaleet ovat menossa toistensa sisään ja niille lasketaan törmäysimpulssit.

Tarkan törmäyshetken löytämiseen käytetään rekursiivista aliohjelmaa, joka puolittaa aika-askeleen, ajaa simulaation uudelleen aika-askeleen puoleen väliin ja katsoo onko törmäys tapahtunut ennen puoltaväliä. Jos on, jatketaan kutsumalla aliohjelmaa uudelleen ensimmäisellä puolikkaalla, jos ei, jälkimmäisellä puolikkaalla. Tätä jatketaan kunnes aika-askelen pituus on kutistunut riittävälle tarkkuudelle. Tämän jälkeen suoritetaan törmäyksen vaatimat toimenpiteet törmäyshetkellä, jonka jälkeen jatketaan etsimällä seuraava törmäyshetki.

2.2 Törmäysimpulssit

Törmäyksen aiheuttamien impulssien määrittäminen on toteutettu suoraan Baraffin [2, s. 40-49] materiaalin mukaan. Kuulan törmätessä tasoon tason massan oletetaan olevan äärettömän suuri ja hitaustensorin käänteismatriisin olevan nollamatriisi. Tällöin kuulaan lisätään tason normaalin suuntainen impulssi, joka vaikuttaa suoraan kuulan liikemäärään ja kulmaliikemäärään, sekä niiden kautta kiihtyvyyteen ja kulmakiihtyvyyteen. Kahden kuulan törmätessä toisiinsa lisätään siihen kuulaan, johon normaali osoittaa, normaalin suuntainen impulssi ja siihen kuulaan, johon normaali on määritelty, impulssi vastakkaiseen suuntaan.

2.3 Kontaktivoimat

Kontaktitilanteessa kuulat eivät ole menossa toistensa sisään eivätkä erkaantumassa toisistaan, vaan niiden suhteellinen nopeus törmäyspisteessä on lähellä nollaa. Kontaktivoimia laskiessa tarkoituksena on löytää sellaiset voimat, jotka estävät kuulien vajoamisen toistensa sisään, estävät kuulia eroamasta ja katoavat kuulien erkaantuessa toisistaan [7].

Kontaktivoimien laskemisessa ratkaistava ongelma on muotoa :

$$\begin{aligned} &Af + b \geq 0 \\ &f \geq 0 \qquad\qquad , \\ &f^T(Af + b) = 0 \end{aligned} \tag{1}$$

missä matriisi $A \in R^{n \times n}$ ja vektori $b \in R^n$ ovat tunnettuja ja $f \in R^n$ tuntematon kontaktivoimavektori, joka pitäisi ratkaista, kun havaittuja kontakteja on *n* kappaletta.

Ongelmassa olevat matriisi *A* ja vektori *b* lasketaan simulaattorissa Baraffin [2, s.49-54, 61-67] materiaalien perusteella. Kuten törmäyksessä myös kontaktissa tason massan katsotaan olevan äärettömän suuri ja hitaustensorin käänteismatriisin olevan nollamatriisi. Itse ongelma (1) ratkaistaan käyttämällä Dantzigin algoritmia [3, s.26] LU-hajotelman kanssa [5, s.37].

2.4 Keinotekoinen kitka

Kuulien törmäyksiin on lisätty keinotekoinen kitka pyörimisliikkeen aikaansaamiseksi. Tason kanssa kitka muodostuu suoraan kiihtyvyydestä. Sen projektio tasolle skaalataan keksityllä, kokeilujen kautta sopivaksi havaitulla vakiolla, jonka jälkeen vaikutus asetetaan kuulan kulmaliikemäärään ja sen myötä kulmakiihtyvyyteen. Kahden kuulan välillä käytetään kuulan kiihtyvyyden projektiota kuulien tangenttitasolle. Ongelmaksi muodostuu kulmaliikemäärien suuri kasvu. Kulmaliikemäärän kasvaessa myös kitka kasvaa, ja sen mennessä äärettömään kuulat vain yksinkertaisesti häviävät. Tätä ongelmaa ei sen kummemmin pohdittu, vaan typistettiin kitka tiettyyn mittaan sen kasvaessa suureksi. Kuulien välillä pitäisi ottaa huomioon myös törmäyspisteen kiihtyvyyden vaikutus toiseen kuulaan kohdistuvaan kitkaan. Tätä ei kuitenkaan saatu onnistumaan.

3 Toteutus

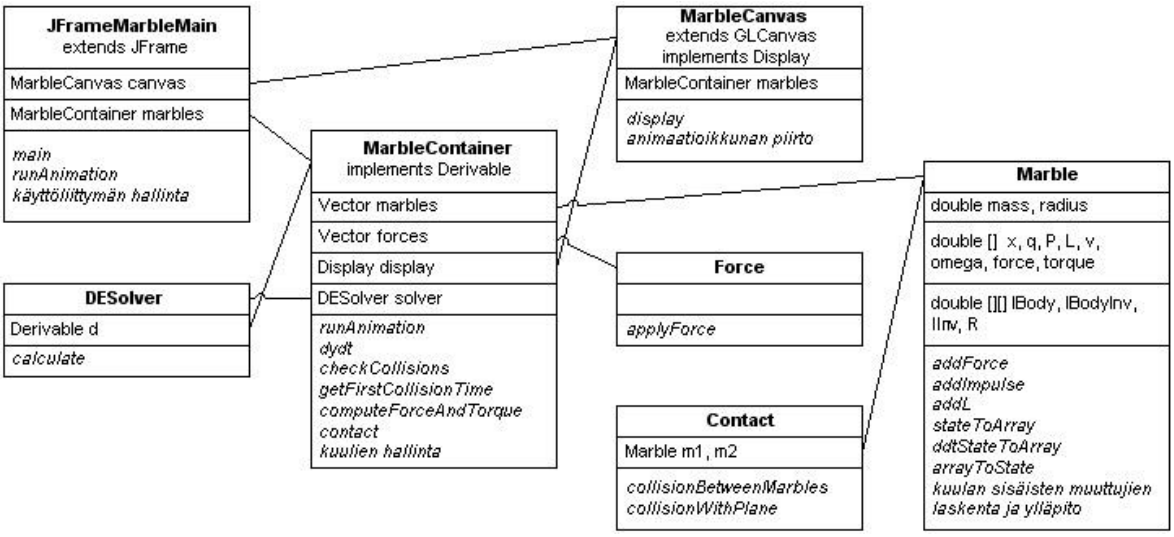
Kuulasimulaattori toteutettiin Javalla ja työkaluna käytettiin Netbeans IDE 3.6 sekä grafiikan piirrossa OpenGL for Java –kirjastoa [6].

3.1 OpenGL for Java

OpenGL for Javasta löytyvät lähes kaikki OpenGL:n funktiot Javalle sovellettuna. Tässä sovelluksessa käytimme piirtämiseen `GLCanvas`-luokasta perittyä `MarbleCanvas`-luokkaa, jossa `display`-metodi on koodattu uudelleen. Koska `MarbleCanvas`-luokalla on viittaus kaikki kuulat sisältävään `MarbleContainer`-luokkaan, se voi pyytää siltä kuulien tiedot yhden kerrallaan. Kuulan sijainti ja asento asetetaan OpenGL funktioilla `glTranslated` ja `glMultMatrixd`, joista `glTranslated` määrittää kuulan sijainnin sen keskipisteen avulla ja `glMultMatrixd` kuulan kierron kvaterniosta q muodostetulla kiertomatriisilla, joka viedään funktiolle 1-ulotteisena taulukkona. Itse kuulan piirto tapahtuu funktiolla `glutWireSphere`.

3.2 Luokkajako

Luokkajako pääpiirteittäin on esitetty kuvassa 1.

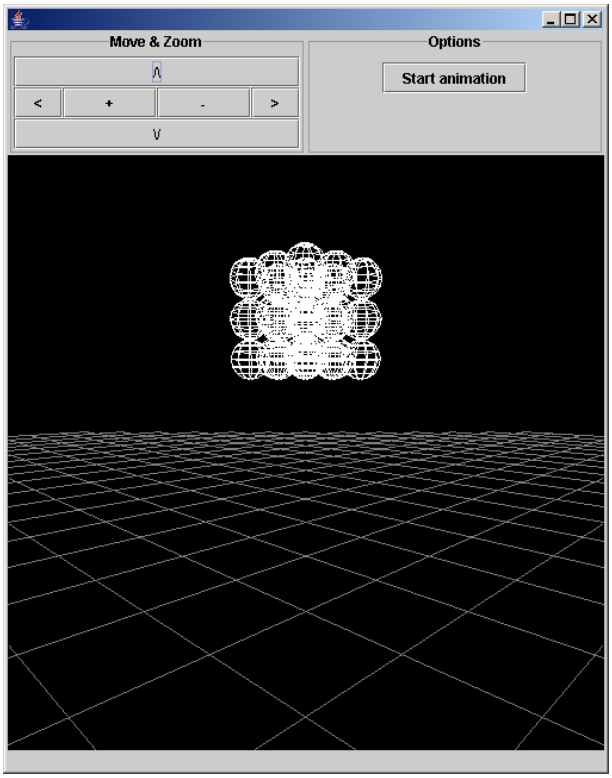


Kuva 1: Kuulasimulaattorin luokkakaavio.

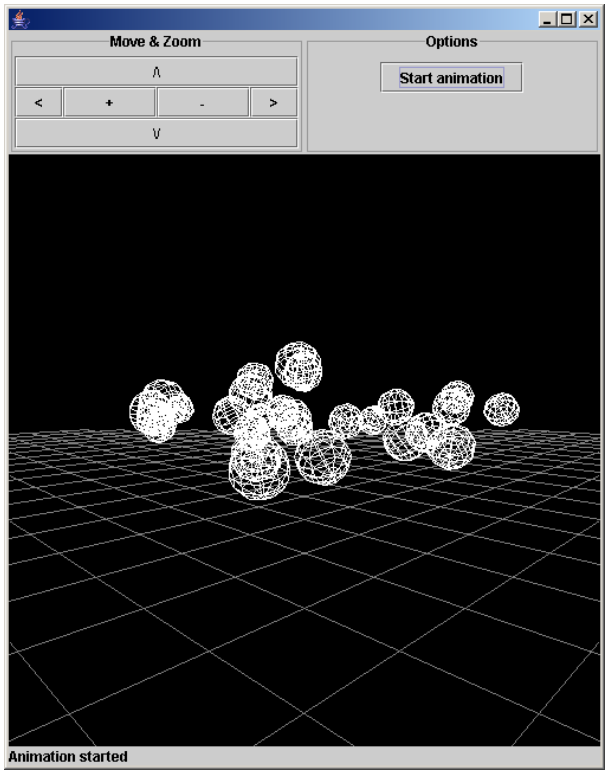
3.3 Toiminta

Sovellus käynnistetään `JFrameMarbleMain`-luokan `main`-metodista. Kuulien muodostaman kuution sivun pituus (kuulien määrä sivulla) on määritettävissä `JFrameMarbleMain`-luokan konstruktorissa, samoin kuulien säde, massa ja kuution etäisyys tasosta.

`Start animation` käynnistää animaation. `Move & Zoom` –ominaisuutta voi käyttää ennen animaation käynnistämistä. Animaation aikana zoomauksen yritys kaappaa kuvan hallinnan animaatiota pyörittävältä säikeeltä, jolloin kuvan päivitys pysähtyy. Kuvassa 2 on esitetty kuulasimulaattorin käyttöliittymä lähtötilanteessa ja kuvassa 3 animaation käynnistämisen jälkeen, kun kuulat ovat alkaneet pudota.



Kuva 2: Lähtötilanne.



Kuva 3: Kuulat kesken animaation ajon, pudotuksen jälkeen.

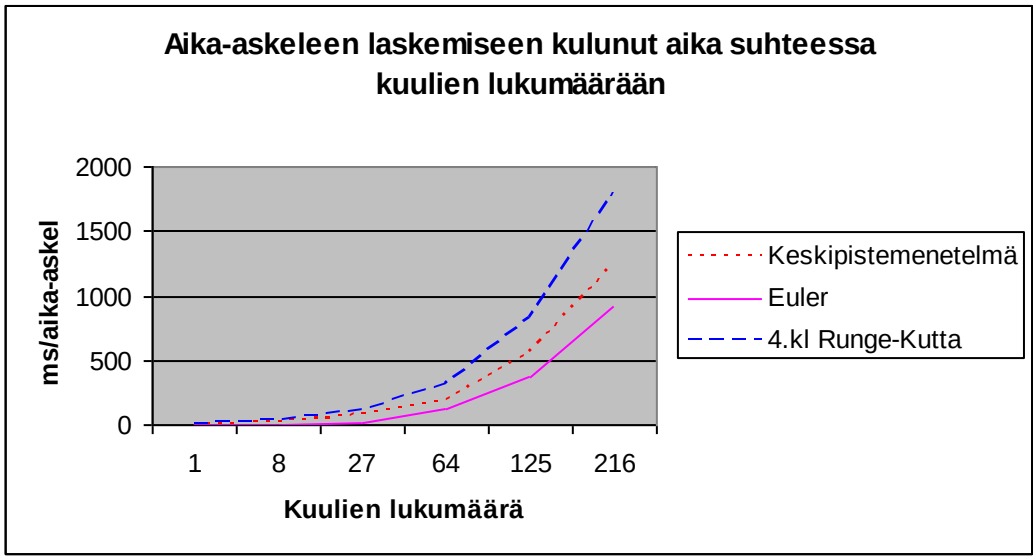
4 Aika-askeleen suoritusajat eri kuulamäärillä ja differentiaaliyhtälöratkaisijoilla

Kuulasimulaattorin suoritusaikoja mitattiin ilman kontaktien laskemisia, koska niitä ei saatu virheettömästi toimimaan. Simulaattorissa mitattiin yhden aika-askeleen suorittamiseen kuluvaa aikaa erilaisilla kuulien lukumäärillä. Kustakin ajosta laskettiin keskiarvo siinä saaduista suoritusajoista. Testiajot suoritettiin kaksi kertaa kaikilla luvussa 1.4 mainituilla differentiaaliyhtälöratkaisijoilla, jolloin voitiin verrata keskenään eri ratkaisijoilla saatuja suoritusaikoja. Koko menetelmän suoritusajaksi on laskettu kahden testiajon tulosten keskiarvo. Suoritusajat ovat taulukoissa millisekunteina. Aika-askel oli kaikissa testeissä 33,33 millisekuntia.

Taulukossa 1 on koottu yhteen eri menetelmien suoritusajat. Yhdellä kuulalla parhaan tuloksen antaa keskipistemenetelmä, mutta kuulien lukumäärän kasvaessa Eulerin menetelmä antaa parhaimmat suoritusajat. Taulukosta voidaan havaita myös, että testeissä kuulien lukumäärän kasvaessa, alkavat eri menetelmien väliset suoritusaikaerot pienentyä. Esimerkiksi 27 kuulalla keskipistemenetelmän suoritusaika on noin viisinkertainen ja 4. kertaluvun Runge-Kutan yli seitsenkertainen verrattuna Eulerin menetelmään, mutta 216 kuulalla suoritusajat ovat alle kaksinkertaiset Eulerin menetelmään verrattuna. Kuvan 4 kuvaajassa on havainnollistettu kuulien lukumäärän vaikutusta suoritusajan kasvuun.

Kuution sivu	**Kuulien lkm**	**Keskipiste**	**Euler**	**4. kl Runge-Kutta**
1	1	3,364583	5,000000	5,000031
2	8	10,000000	4,995117	28,153499
3	27	70,062279	14,117221	106,760596
4	64	188,921712	125,861759	308,680822
5	125	553,016750	373,524471	827,910282
6	216	1245,028068	923,951787	1795,435387

Taulukko 1: Eri menetelmien suoritusajat.



Kuva 4: Aika-askeleen suoritusaika suhteessa kuulien lukumäärän kasvuun eri differentiaaliratkaisumenetelmillä.

Lähteet

[1] Baraff David, *An Introduction to Physically Based Modeling: Rigid Body Simulation 1 – Unconstrained Rigid Body Dynamics*, Siggraph '97 Course notes, 1997.

[2] Baraff David, *An Introduction to Physically Based Modeling: Rigid Body Simulation 2 – Nonpenetration Constraints*, Siggraph '97 Course notes, 1997.

[3] Baraff David, *Fast Contact Force Computation for Nonpenetrating Rigid Bodies*, In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, p. 23-34, 1994.

[4] Lander Jeff, Physics *on the Back of a Cocktail Napkin*, saatavilla WWW-muodossa <URL: http://www.gamasutra.com/features/20000516/lander_pfv.htm>, viitattu 17.11.2004.

[5] Mäkinen Raino A. E., *Numeeriset menetelmät*, Luentomoniste 1, Tietotekniikan laitos, Jyväskylän yliopisto, 1999.

[6] OpenGL for Java, saatavilla WWW-muodossa <URL: http://www.jausoft.com/products/gl4java/gl4java_main.html>, viittattu 17.11.2004.

[7] Vire Markku, *Tietokonepelien fysiikan matemaattinen mallintaminen*, Pro Gradu -tutkielma, Jyväskylän yliopisto, 2003.

[8] Witkin A., Baraff D., *Physically Based Modeling: Principles and Practice, Differential Equation Basics*, Siggraph '97 Course notes, 1997.