Törmäystarkistus algoritmi toteutettuna kordinaattien mukaan järjestetyillä listoilla.

Simo Siiriä

5. marraskuuta 2004

Sisältö

1 Johdanto

Työn tarkoituksena on tutkia kordinaattiakselien mukaan järjestettyjen listojen mahdollisuuksia rajata törmäystarkistuksien määrää simulaatiossa. Tähän tarkoitukseen rakennettiin yksinkertainen simulaatioympäristö johon menetelmää sovellettiin. Ympäristö toteutettiin java kielellä.

Menetelmän todettiin nopeuttavan huomattavasti simulaatiota verrattuna menetelmään jossa kaikki tötmäysvaihtoehdot tarkistetaan ilman karsintaa. Simulaation luonne vaikutti oleellisesti nopeutuksen tehoon. Ominaisuuksia jotka vaikuttivat tehokkuuteen olivat esimerkiksi alueen partikkelitiheys, partikkelien koko suhteessa alueeseen ja partikkelien kokonaismäärä.

2 Simulaatioympäristö

2.1 Aika

Simulaatio kulkee tasamittaisissa simulaatiojaksoissa. Yhden askeleen kattama aika ilmoitetaan simulaationa sekunteina. Esimerkiksi 0.001 s aika-askelilla, simulaatio, jossa 1 $\frac{m}{s}$ kulkeva kappale liikkuu metrin, vaatii 1000 simulaatio-askelta. Aika-askeleella 0.02 taas ainoastaan 50. Simulaation tarkkuus paranee pienemmällä aika-askeleella. Vastaavasti simulaatio nopeutuu suuremmalla aika askeleella. Askel ei kuitenkaan saisi olla niin suuri, että partikkelit liikkuvat todennäköisesti yhden askeleen aikan matkoja jotka ovat huomattava osa niiden säteestä.

2.2 Alue

Simulaatioympäristö koostuu kaksiulotteisesta neliömäisestä alueesta, joka piirretään kokonaisuudessaan ruudulle. Oletusarvoisesti alueen koko on 500 kertaa 500 m. Simulaation mittasuhteissa yksi pikseli vastaa yhtä metriä. Tämä mittasuhde valittiin jotta simulaatio saataisiin mahdollisimman yksinkertaiseksi, muut mittasuhteet vaatisivat muutamien ylimääräisten kertoimien lisäämistä simulaation.

2.3 Partikkelit

Partikkelit ovat ympyrän muotoisia, ja saman säteisiä. Oletusarvoinen koko on 12 metriä. Partikkelilla on ominaisuuksina säde, paikka(x,y),nopeus(x,y) ja kimmoisuus. Kaikki ominaisuudet säilytetään liukulukuina. Säde ja sijainti ilmoitetaan metreinä(m). origo on alueen vasemmassa yläkulmassa. Nopeudet ilmoitetaan metreinä sekunnissa ($\frac{m}{s}$). Kimmoisuus on lukuarvo joka kuvaa kuinka suuren voiman törmäävät partikelit kohdistavat toisiinsa. Voima lasketaan kaavan 1 mukaisesti, joten kimmoisuuden yksiköksi muodostuu Newtonia neliömetri ($\frac{N}{m^2}$). Piirrettäessä partikkelin väri kuvaa onko se osallisena törmäykseen (punainen) vai ei (sininen).

2.4 Törmäykset

Simulaatiossa on käytännössä kahden tyyppisiä törmäyksiä. Oleellisemmat ovat partikkelien väliset törmäykset. Näiden lisäksi on partikkelien ja alueen seinien välisiä törmäyksiä. Törmäys katsotaan tapahtuneeksi partikkelien välillä, kun niiden keskipisteiden etäisyys on pienempi kun niiden säteiden summa. Partikkelien väliset törmäykset aiheuttavat kullakin ajan-jaksolla tietyn suuruisen, yhtä voimakkaan, ja vastakkaissuuntaisen voiman molempiin törmääviin hiukkasiin. Voiman suunta on aina suoraan poispäin vaikuttavasta kappaleesta. Voiman suuruus lasketaan kaavan 1 mukaisesti.

$$F = K((R_1 + R_2) - Et)^2 \tag{1}$$

Yhtälössä F on Aiheutettu voima, K on partikkelin kimmokerroin, R_1, R_2 törmäävien partikkelien säteet, ja Et niiden keskipisteiden etäisyys toisistaan.

Partikkelin törmääminen seinään lasketaan samalla periaattella. Seiniin ei vaikuta voimia, vaan ainoa vaikuttava voima tulee aina partikkeliin. Sen suunta on suoraan poispäin seinästä. Voima lasketaan kaavan 2 mukaisesti.

$$F = K(-Et_s + R)^2 \tag{2}$$

Missä Et_s on partikkelin keskipisteen etäisyys seinästä (tai kuinka paljon se on seinän takana, jolloin negatiivinen), ja R partikkelin säde.

3 Törmäyksien karsinta-algoritmi

3.1 Raaka-voima vaihtoehto

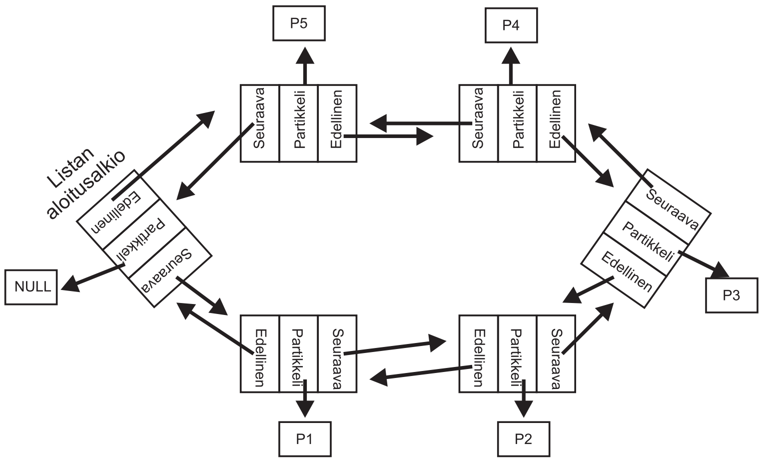
Luonnollisin tapa Tarkastella ympäristössä tapahtuvia törmäyksiä on käydä kaikki mahdollisuudet lävitse. Tämä tarkoittaisi että katsotaan jokaisen hiukaksparin etäisyyttä toisistaan , ja suoritetaan törmäys jos etäisyys on riittävän pieni.

Tällä menetelmällä n:llä kappaleella tarkasteluja tulee $\frac{n^2}{2}$ kappaletta. Tarkastelujen tarve kasvaa siis nopeasti. Lisäksi useimmat tarkasteluista ovat turhia, ja näiden turhien tarkasteluiden läpikäymiseen kuluisi helposti suurin osa simulaatioajasta.

3.2 Listojen rakenne

Listat toteutettiin rengasmaisina linkitettyinä listoina. Listaa varten luodulla oliolla on ominaisuuksinaan pointteri partikkeli olioon, sekä pointterit seuraavaan, ja edelliseen alkioon. Lisäksi alkiolla on tieto siitä kuuluuko se x, vai y suuntaisesti järjestettyyn listaan.

Listan ensimmäinen alkio osoittaa aina tyhjään alkioon. Sen seuraavaan osoittava pointteri osoittaa ensimmäiseen todelliseen alkioon. Ja viimeisen todellisen alkion seuraavaan osoittava pointteri osoittaa aloittavaan tyhjään alkioon(kuva 1). Tämä varmistaa symmetrian alkioiden välillä, koska myös ensimmäisellä ja viimeisellä alkiolla on seuraava ja edellinen alkio.

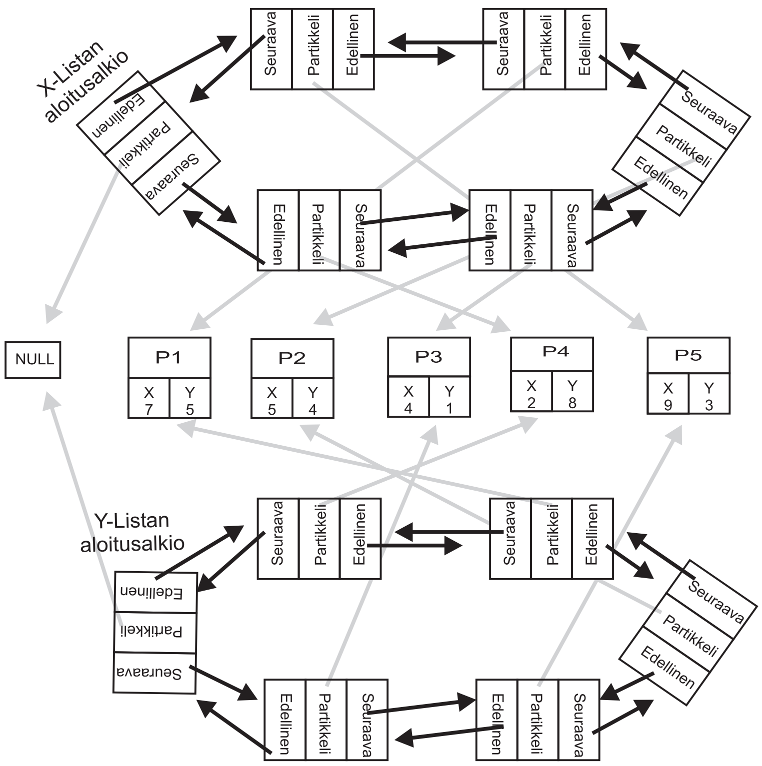


Kuva 1: Rengasmaisen listan rakenne. Vasemmanpuoleisin alkio on listan aloitusalkio. P1-P5 listan sisältämät partikkeli oliot.

3.3 Kordinaattiakselien mukaan järjestetyt listat

Tässä työssä tarkastellaan seuraavanlaisen menetelmän vaikutusta laskentatehoon törmäyksien tarkastelussa: Uutta partikkelia luodessa sijoitetaan osoitin siihen kahteen erilliseen listaan. Sijoitus suoritetaan siten, että toinen listoista pysyy x-kordinaattien mukaisessa nousevassa järjestyksessä.(kuva 2 Toinen lista taas pidetään y-kordinaattien mukaisessa järjestyksessä. Simulaation kuluessa kun partikkeli siirtyy tarkastellaan samalla pitääkö sitä siirtää listoissa eteen- tai taaksepäin jotta listat pysyvät järjestyksessä.

Tarkasteltaessa törmäyksiä, jokainen partikkeli käydään läpi sijoitusjär-



Kuva 2: Kordinaattien mukaan järjestetyt listat. Samat alkiot on järjestetty kahteen listaan eri kriteerien mukaan.

jestyksessä. Törmäyksiä tarkastellessa ei verrata etäisyyttä kaikkiin muihin partikkeleihin, vaan katsotaan kumamstakin listasta seuraavaa ja edellistä alkiota. Olettaen että alkiot ovat olemassa katsotaan onko x-taulukossa olevien x-sijainti niin lähellä että partikkelit voisivat törmätä. Vastaavasti tarkastellaan y-taulukkoa. Käytännössä tämä tarkoittaa etäisyyttä joka vastaa simulaation kahden suurimman säteen summaa. Tässä simulaatiossa säteet ovat identtiset joten $2 * R$ on etäisyys joka asettaa rajan. Molempia listoja käydään läpi alkio kerralaan, kunnes jommastakummasta on löytynyt *molemmista* päistä alkio joka ylittää raja-etäisyyden, tai listan pää. Kun toinen lista täyttää ehdot, tarkastellaan tästä listasta löydettyjen partikkelien mahdolliset törmäykset partikkeliin. Näin vältytään tarkastelemasta partikkeleja jotka ovat listassa tämän alueen ulkopuolella.

3.4 Listojen järjestäminen

Jokainen uusi alkio lisätään oikeaan paikkaansa listassa. Y-listassa tämä tarkoittaa sitä, että listan alusta kuljetaan listaa eteenpäin, kunnes törmätään esnimmäiseen alkioon, jonka partikkelin y-kordinaatti on suurempi kun lisättäbän alkion, tai törmätään listan loppuun. Vastaava vertailu suoritetaan x-listassa.

Koska partikkelit siirtyvät jatkuvasti simulaation kuluessa, tulee listan järjestystä ylläpitää jokaisella simulaatio askeleella. Kutakin partikkelia siirretään ainoastaan kerran jokaisella simulaatioaskeleella. Kullakin partikkelilla on osoitin alkioon joka kuvaa sitä kussakin listassa. Samalla kun partikkeli siirtyy, katsotaan onko sen alkio vielä oikealla paikalla listoissa. Ellei, alkio irroitetaan listasta. Sitten kuljetaan listassa eteen, tai taaksepäin, kunnes löydetään oikea paikka alkiolle, ja liitetään se takaisin listaan. Koska tämä suoritetaan välittömästi kordinaattien muuttumisen jälkeen, ainoastaan tämä alkio voi olla väärässä paikassa. Muutoksen jälkeen lista on siis jälleen järjestyksessä.

4 Tehokkuuden Tutkinta

4.1 Nopeuden määritteleminen

Tutkittaessa koko simulaation nopeutta, ei välttämättä päästä käsiksi itse algoritmin tehonkulutukseen. Ohjelman muut osiot, kuten grafiikan tulostus voivat vääristää tuloksia. Erityisesti näin voi käydä alhaisella partikkelimäärällä, jolloin törmäysten laskenta ei välttämättä vie kuin pineen osan kokonais suoritustehosta. Tämän vuoksi puhdas ruudunpäivityksiä/sekunnissa arviointi ei toimi algoritmin nopeutta testattaessa.

Tässä työssä nopeus on määritelty käynistämällä aikalaskuri aina ennen kuin simulaation hoitava rutiini käynnistetään, ja katkaisemalla laskenta heti kun ohjelma palaa tästä rutiinista. Jokaisella ruudunpäivityksellä on laskettu rutiinissa kuluva aika, ja näiden aikojen keskiarvoa on käytetty vertailulukuna.

4.2 Simulaatioiden vertailukelpoisuus

Jotta nopeusvertailuista saataisiin vertailukelpoisia numeroita, pyrittiin jokainen simulaatio suorittamaan mahdollisimman samalla tavalla. Tämän vuoksi satunnaisluku moottorin siemennumero on jätetty parametriksi jokna käyttäjä voi syöttää. Jokaisessa simulaatiossa simuloitiin myöskin yhtä pitkä simulaatio, samalla askelvälillä.

4.3 Algoritmin oikeellisuuden määritteleminen

Ennen varsinaisia simulaatiota varmistettiin, ettei optimoitu lista-algoritmi karsi liikaa vertailuja. Tämä suoritettiin ajamala identtisestä lähätötilanteesta saman pituisia simulaatiota, ja toteamalla että molemmissa tapauksissa simulaation lopputulos, eli partikkelien sijainnit olivat samanlaiset. Tämä todennettiin vertaamalla lopputuloksen kuvia toisiinsa, ja toteamalla niiden olevan identtisiä.

4.4 Kaoottisuus

Törmäyssimulaatio on hyvin alkuarvo herkkä. Tämän vuoksi pienikin muutos alkuarvoissa tai laskennassa muuttaa lopulta täydellisesti partikkelien loppusijainnin. Simulaatio käsittelee paikkoja, voimia ja nopeuksia liukulukuina, joilla on tietty tarkkuus. Tästä johtuen aina laskettaessa lukuja yhteen niitä joudutaan pyöristämään hieman. Olettaen että täysin samat laskutoimitukset suoritetaan samassa järjestyksessä luvuille, loputulos on kuitenkin aina sama. Jos siis samoilla alkuarvoilla, ja samalla algoritmilla ajetaan kaksi simulaatiota, ne päätyvät täysin samaan lopputulokseen.

Jos kuitenkin samat laskutoimitukset suoritetaan hieman eri järjestyksessä, lopputulokseen voi tulla pientä eroa liukulukujen luonteesta johtuen.

Pitkillä simulaatiolla tämä ilmiö tuli näkyviin. Ero saatiin esiin keksimäärin yli 1000 aika-askeleen jälkeen 200:lla partikkelilla.

Jotta voitiin olla varmoja että ero johtuu liukuluvuista, eikä algoritmista, lisättiin ohjelmaan vielä parametri, jonka ollessa tosi taulukon alkioiden järjestys arvotaan ennen simulaation aloittamista. Tämän ei pitäisi muuttaa varsinaista laskentaa millään muulla tavalla kun vaihtaa järjestys jossa vuorovaikutukset lasketaan. Koska vuorovaikutukset ovat ainoastaan etäisyyksien funktioita, ja vaikuttavat ainoastaan nopeuksiin, jotka eivät vaikuta paikkoihin ennen kun kaikki vaikutukset on laskettu, ei tämä muutos vaikuta lopputulokseen muuten kun liukulukulaskussa tulevien pyöristyshäiriöiden muodossa.

Samalla menetelmällä, mutta arvotulla järjestyksellä syntyi eroa lopputulokseen jotakuinkin yhtä paljon pitkällä ajolla, kuin eri menetelmillä. Ilmeisesti erojen syntyminen johtuu siis lähinnä järjestyksen muuttumisesta ja siten likuluku pyöristysten kasautumisesta.

5 Tulokset

Kaikki ajot joiden avulla vertailu suoritettiin, ajettiin samassa tietokoneessa, Sunin kehittämän Netbeans java-ympäristön applet-viewerin alla. Kaikki ajot suoritettiin sekä järjestettyjen listojen algoritmilla, että raa-alla voimalla toimivalla algoritmilla. Jokaisen simulaation annettiin simuloida törmäyksiä 10 sekunnin ajan, 0.01 aika-askeleella. Jokaisessa simulaatiossa oli siis 1000 askelta. Muut parametrit on esitetty taulukossa 1.

Simulaatiot suoritettiin molemmilla laskenta-algoritmeilla. Käytetyt partikkelimäärät olivat: 10, 20, 40, 100, 200, 400, 1000, 1500, 2000 ja 2500. Saadut tulokset on esitetty taulukossa 2, ja havainnollistettu kuvassa 3. Kuvaajassa simulaatiosta saadut numerot on merkitty tähdillä (*). Pisteet yhdistävä suora on saatu sovittamalla tuloksiin toisen asteen yhtälö. Optimoimat-

Parametri	Arvo
Seed	1
Radius	10
number	10-2500
mode	1,2
koko	600
bouncef	0.1
startspeed	50
timestep	0.01
labels	0
random	0

Taulukko 1: Simulaatioiden parametrit. Parametrien merkitykset selostettu rakemmin liitteessä A

toman simulaation tuloksia vastaava yhtälö oli muotoa

$$Aika = 1.3 * 10^{-4} partikkelit^2 \tag{3}$$

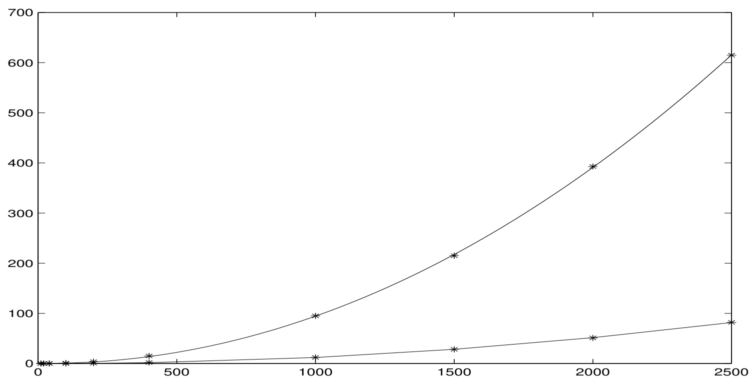
ja optimoidun

$$Aika = 1.3 * 10^{-5} partikkelit^2 \tag{4}$$

LIsäksi kokeiltiin miten laskentavaatimus muuttuu pienentämällä palljoen sädettä puoleen partikkelimäärällä 2000. Tällöin optimoimattoman algoritim tehon vienniksi tuli 386 (suuremmalla säteellä 393) ja optimoidun 25 (suuremmalla säteellä 51).

Partikkeli ($no.$)	Optimoimaton(ms)	Optimoitu(ms)
10	0.03	0.02
20	0.05	0.06
40	0.18	0.05
100	0.9	0.2
200	3.7	0.6
400	15.0	2.0
1000	94.9	12.1
1500	215	28.2
2000	393	51
2500	615	82

Taulukko 2: Simulaatioiden tulokset. Ajat ovat keskimäärin käytetty aika laskentaan per aika-askel.



Kuva 3: Simulaatioiden nopeustulokset. X-akselilla partikkelien määrä, Y-akselilla vastaava millisekuntti määrä. Ylempi kuvaaja vastaa ilman optimointia tehtyjä simulaatiota, alempi optimoinnin kanssa.

6 Johtopäätökset

Taulukko-optimointi vaikuttaa tuntuvasti suorituksen nopeuteen. Tehovaatimus vaikuttaa kuitenkin kasvavan siinäkin nopeudella n^2, vaikkakin paljon pienemmällä kertoimella. Parhaiten optimointi toimii tilanteissa joissa partikkeli tiheys on pieni, joten törmäykset verraten harvinaisia. Karsinta on myöskin tehokkaampaa mitä pienempiä partikkelit ovat suhteessa tilaan jossa ne voivat liikkua.

A Liite: Simulaation parametrit

seed Satunnaisluku siemen.

radius partikkelien säde.

number partikkelien määrä.

mode käytetty algoritmi. 1=raaka-voima, 2=lista-optimointi.

bouncef pallojen kimmoisuus. Suurempi luku=jäykemmät pallot.

startspeed pallojen aloitusnopeus $(\frac{m}{s})$.

timestep aika-askelen pituus.

timelimit kuinka monen sekunnin pituinen simulaatio suoritetaan.

labels piirretäänkö partikkelien järjestysnumerot vai ei.

random Arvotaanko listan järjestystä ennen simulointia vai ei.

koko (Ei varsinaisesti appletin parametri.) Alueen sivun pituus (m).

B Liite: Lähdekoodi