Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

Simulaatio pelimoottoreissa

Kandidaatintyö

29. huhtikuuta 2012

Dominik Mäkinen

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Dominik Mäkinen
Työn nimi:	Simulaatio pelimoottoreissa
Päiväys:	29. huhtikuuta 2012
Sivumäärä:	23
Pääaine:	Ohjelmistotekniikka
Koodi:	T3001
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	TkT Vesa Hirvisalo (Tietotekniikan laitos)

Simulaatiolla pelimoottoreissa tarkoitetaan reaalimaailmaan sijoittuvien ilmiöiden, esineiden ja olentojen fysikaalisten vuorovaikutuksien mallintamista. Tämä mallinnus perustuu yksinkertaistettuun Newtonilaiseen mekaniikkaan ja on hyvin usein toteutettu pelimoottoriin kuuluvassa fysiikkamoottorissa.

Tämän kandidaatintyö selvittää, miten reaalimaailman simuloinnin kolme pääkomponenttia, törmäyksien havaitseminen ja hallinta, jäykkien kappaleiden dynamiikka ja ei-jäykkien kappaleiden dynamiikka toimivat peleissä ja pelimoottoreissa. Työn painopiste on erityisesti reaaliaikaisten pelien kuten esimerkiksi kolmiulotteisten ammuskelupelien fysikaalisen dynamiikan simuloinnissa.

Käytännön esimerkkinä simulaation toiminnasta hyödynnetään Bulletia. Bullet on avoimen lähdekoodin fysiikkamoottori, joka huolehtii peleissä tapahtuvista reaalimaailman simuloinneista. Tärkeimmät komponentit Bulletissa ovat törmäysten havaitseminen, jäykän kappaleen dynamiikka ja ei-jäykän kappaleen dynamiikka. Simulaatio toteutetaan Bulletissa hyödyntämällä näitä komponentteja Bulletin simulaatioaskeleessa. Tähän simulaatioaskeleeseen kuuluvat painovoiman vaikutuksen lisääminen kappaleisiin, lavea -ja tarkkavaiheinen törmäysten havaitseminen ja kappaleiden paikan päivittäminen.

Avainsanat:	simulaatio, simulointi, pelimoottori, peli, törmäykset, jäykkä kappale, dynamiikka, fysiikkamoottori, Bullet
Kieli:	Suomi

Sisältö

1 Johdanto

Simulaation osuus tietokonepeleissä ja pelimoottoreissa on kasvanut huomattavasti viimeisen kahden vuosikymmenen aikana. Simulaatiolla peleissä tarkoitetaan jonkin asian tai ilmiön esittämistä ja mallintamista mahdollisimman realistisesti tietokoneen tai pelikonsolin avulla. Vuoropohjaisissa strategiapeleissä kuten esimerkiksi Civilization-sarjassa simulaatio käsittää eri kansojen tai eri kaupunkien kehittymisen tai niiden välisten suhteiden mallintamista. Reaaliaikaisissa peleissä kuten esimerkiksi nykyaikaisissa ensimmäisen persoonan ammuskelupeleissä (Crysis) simuloinnilla tarkoitetaan mahdollisimman tarkkaa reaalimaailman fysikaalista mallintamista. Pääpainona on etenkin mekaniikka, jossa Newtonin lait ovat vahvassa roolissa. Tässä työssä termit simulaatio ja simulointi ovat synonyymejä.

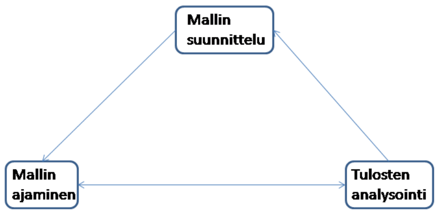
Tämän kandidaatintyö tutkii simulaation toimintaa pelimoottoreissa. Työssä keskitytään etenkin reaaliaikaisissa peleissä tapahtuvaan simulaatioon. Painopisteenä on varsinkin dynamiikan mallinnus eli se, miten eri objektit reagoivat keskenään esimerkiksi törmäystilanteessa ja, miten näitä törmäystilanteita ylipäätään havaitaan. Työssä ei siis tarkastella esimerkiksi vuoropohjaisissa peleissä tapahtuvaa simulaatiota. Työssä ei myös syvennytä fysiikan teorioihin ja kaavoihin. Vaikka aiheena onkin simulaatio pelimoottoreissa, simulaation toteutus ja varsinkin reaalimaailman simulointi on usein tehty pelimoottoriin kuuluvassa fysiikkamoottorissa. Työ on luonteeltaan kirjallisuuskatsaus.

Luvussa 2 esitellään kaksi yleisintä simulointimallia: jatkuvatoiminen simulointi ja diskreetti tapahtumasimulointi. Luvussa 3 esitellään puolestaan tyypillinen pelimoottorin rakenne. Luku 4 keskittyykin itse aiheeseen. Tämä luku on jaettu kolmeen osaan: törmäysten havaitsemiseen ja hallintaan, jäykkien kappaleiden dynamiikkaan ja ei-jäykkien kappaleiden dynamiikkaan. Nämä kolme osaa ovat hyvin keskeisessä roolissa reaalimaailman simuloinnissa. Luvussa 5 esitellään konkreettinen esimerkki, josta ilmenevät luvun 4 asiat ja teoriat käytännössä. Valitsin esimerkiksi Bulletin, joka on useissa kaupallisissakin projekteissa käytetty avoimeen lähdekoodiin perustuva fysiikkamoottori.

2 Simulointimallit

Tässä luvussa määritetään ensiksi, mitä simulaatio on. Tämän jälkeen keskitytään kahteen kenties yleisimpään simulointimalliin diskreettiin tapahtumasimulointiin sekä jatkuvatoimiseen simulointiin. Nämä simulointimallit ovat ikään kuin kehys, jolle itse pelimoottoreissa tapahtuva simulointi rakentuu.

Tietokonesimulointi on reaalimaailmaan sijoittuvan järjestelmän mallin suunnittelua, suorittamista ja lopputulosten analysointia. Ensiksi fyysistä objektia simuloitaessa tarvitaan matemaattinen malli. Toiseksi tietokoneohjelman avulla päivitetään tämän mallin tilaa tietyin aika-askelin. Lopuksi mallin käyttäytymistä analysoidaan ja muutetaan mallia paremmaksi tai suoritetaan mallia uudenlaisilla muuttujilla. Kuvassa 1 esiintyy nämä kolme simulointivaihetta ja niiden väliset suhteet. [8]



Kuva 1: Simulointivaiheet.

Tietokonesimuloinnilla pyritään täten matemaattisen mallin avulla selvittämään, kuinka tällainen reaalimaailmaan sijoittuva objekti käyttäytyy erilaisissa tilanteissa. Tietokonesimulointi on sopiva tilanteisiin, joissa alkuperäisen järjestelmän muuttaminen tai testaaminen on hyvin vaikeaa, mahdotonta tai kallista. Tarkempana määritelmänä simulaatiolle voidaan pitää seuraavaa: reaalimaailman prosessin tai laitteiston operaatioiden mallintaminen hyvin tehtyjen loogisten, tilastollisten tai matemaattisten oletuksien avulla.

Tietokonesimuloinnit, tai simuloinnit ylipäätään, voidaan jakaa ainakin neljään eri ryhmään: jatkuvatoiminen, Monte Carlo, diskreetti tapahtuma - ja toimijapohjainen simulointi. Käsittelen näistä seuraavaksi kahta yleisintä eli jatkuvatoimista ja diskreettiä tapahtumapohjaista simulointia. [14]

2.1 Jatkuvatoiminen simulointi

Jatkuvatoiminen simulointi mallintaa järjestelmää esittävien yhtälöiden joukkoa. Tämänkaltainen simulointi voi koostua esimerkiksi fysiikkaa mallintavista differentiaaliyhtälöistä, joita päivitetään lyhyen aikajakson välein. Esimerkiksi auton jousitusta voidaan mallintaa jatkuvatoimisen simuloinnin avulla. [14]

Jatkuvatoimista järjestelmää voidaan kutsua myös dynaamiseksi järjestelmäksi. Koska tietokone toimii pohjimmiltaan diskreettien muuttujien ja operaatioiden avulla, on jatkuvat funktiot muunnettava diskreeteiksi esimerkiksi approksimoimalla. [12]

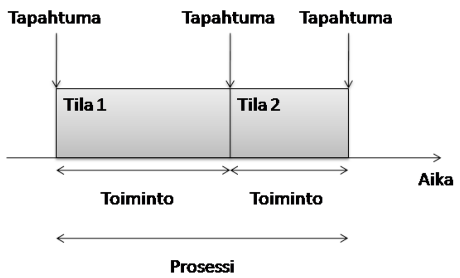
Esimerkkinä jatkuvatoimisesta simuloinnista voidaan myös pitää vesisäiliötä, jossa on sisään- ja ulostuloputki. Vettä siis virtaa vesisäiliöön ja vesisäiliöstä pois tietyn jatkuvan funktion (usein ajan suhteen) mukaan. [17]

2.2 Diskreetti tapahtumasimulointi

Diskreetissä tapahtumasimuloinnissa järjestelmän tila muuttuu vain ennalta määrättyjen tapahtumien seurauksena. Aika siis etenee diskreetissä tapahtumasimuloinnissa nimensä mukaisesti epäjatkuvasti, hyppäyksittäin. Pankkiautomaatille saapuvien asiakkaiden jonotusjärjestelmä on esimerkki diskreetistä tapahtumasimuloinnista: järjestelmässä on neljä mahdollista tilaa, asiakkaan saapuminen, palveluun jonottaminen, palvelun käyttäminen ja poistuminen. Kun tila muuttuu järjestelmässä, tässä tapauksessa esimerkiksi palvelun jonottamisesta palvelun käyttämiseen, se tapahtuu välittömästi. [14]

Diskreettiä tapahtumasimulointia voidaan havainnollistaa myös vesisäiliö esimerkin avulla. Nyt vettä ei virtaakaan säiliöön tai säiliöstä pois, vaan veden määrää lisätään tai vähennetään säiliöautojen avulla. Aina kun säiliöauto tuo vettä, se liittyy tapahtumajonoon. Kun on tämän auton vuoro, se täyttää säiliön ja poistuu paikalta. Tilanmuutokset tapahtuvat tässäkin välittömästi. [17]

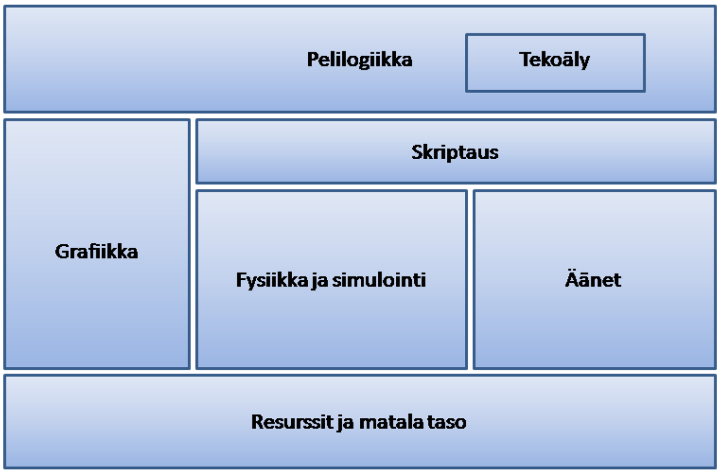
Diskreettejä tapahtumasimulointeja voidaan tarkastella kolmesta eri näkökulmasta: varsinaisesta tapahtumapohjaisesta, toimintopohjaisesta ja prosessipohjaisesta näkökulmasta. Tapahtumapohjainen näkökulma on se, millaiseksi diskreetti tapahtumasimulaatio yleensä mielletään, eli tähän lähestymistapaan kuuluu joukko tapahtumia, jotka muuttavat välittömästi mallin tilaa. Toimintopohjaisessa näkökulmassa mallin tilaa muuttavat jatkuvatoimiset mutta keskenään diskreetit toiminnot. Toiminnolla on lähtötila, suoritettava toiminto ja lopputila. Prosessipohjainen näkökulma koostuu keskenään diskreettien prosessien joukosta, joka muuttaa mallin tilaa. Kuvassa 2 esitetään, kuinka nämä kolme näkökulmaa suhteutuvat toisiinsa. Näitä näkökulmia voidaan siis myös pitää diskreetin tapahtumasimuloinnin hienojakoisuusluokkina tai abstraktiotasojakoina. [10]



Kuva 2: Diskreetin tapahtumasimuloinnin näkökulmat.

3 Pelimoottorit

Pelimoottorit koostuvat useista erinäisistä pienemmistä moottoreista. Kuvassa 3 esiintyy tyypillinen kolmiulotteisen ensimmäisen persoonan ammuskelupelin (engl. 3D first person shooter, 3D FPS) pelimoottorin arkkitehtuuri (huomattavasti yksityiskohtaisempi versio lähteessä [9]). Esittelen seuraavissa alaluvuissa viisi mielestäni tärkeintä osamoottoria. Vaikka esimerkiksi ensimmäisen alaluvun nimi on grafiikka, sillä tarkoitetaan nimenomaan grafiikasta huolehtivaa osamoottoria (grafiikkamoottori).



Kuva 3: FPS-tyylisen (engl. first person shooter) pelin pelimoottorin arkkitehtuuri.

3.1 Grafiikka

Kolmiulotteisten pelien moottorissa grafiikasta huolehtii 3D-moottori toiselta nimeltään yksinkertaisesti grafiikkamoottori. 3D-moottorin tehtäviin kuuluvat kaikki matalan tason tehtävät grafiikassa, kuten näytönohjaimen kanssa keskustelu, mallien transformaatio ja renderöinti. [21] Toisin sanoen grafiikkamoottori huolehtii pelimaailmassa näkyvien objektien piirtämisestä näytölle. 3D-moottori toteutetaan renderöijän avulla, joka käyttää hyödyksi tiettyä ohjelmointirajapintaa, joista tunnetuimmat lienevät Direct3D ja OpenGL. [6]

On olemassa kahdenlaisia grafiikan liukuhihnoja: kiinteätoimintoisia (engl. fixed-function) ja ohjelmoitavia liukuhihnoja. Kiinteätoimintoisia liukuhihnoja ei pysty muokkaamaan, kun taas ohjelmoitavia voi. [19] Ohjelmoitavat liukuhihnat toteutetaan varjostimien avulla. Varjostimia on kahta eri tyyppiä: verteksi -ja pikselivarjostimet. Verteksivarjostimilla voidaan hallita piirtämistä nimensä mukaisesti verteksien ominaisuuksilla, kuten paikalla, normaalilla ja väreillä. Pikselivarjostimilla puolestaan hallitaan piirtämistä kuvaan pohjautuvilla ominaisuuksilla, kuten verteksien paikoilla ja tekstuurien koordinaateilla.

Pelimaailmaan (tässä engl. scene) kuuluvat kaikki pelissä esiintyvät objektit. Objektien välistä suhdetta kutsutaan pelimaailmagraafiksi. Jotta renderöijä saataisiin tehokkaammaksi, täytyy kontrolloida piirrettäväksi tarkoitettujen objektien määrää. Tämä toteutetaan pelimaailmagraafihallinnan avulla. [6]

Jokaiselle pelimaailmassa näkyvälle objektille täytyy määrittää ainakin seuraavat kolme komponenttia. Ensimmäinen komponentti on objektin geometria eli se, minkä muotoinen tietty kappale on. Toinen komponentti on materiaali eli se, miten tietty kappale heijastaa valoa ja kuinka läpinäkyvä se on. Kolmas komponentti on tekstuurit eli se, miltä tietty objekti näyttää. Tekstuuri on ikään kuin paperi, joka kiedotaan tietyn kappaleen päälle. Pelimaailmalle täytyy puolestaan määrittää ainakin seuraavat kaksi komponenttia. Ensimmäinen komponentti on valaistus eli se, onko pelimaailmassa esimerkiksi ympäristövalaistusta ja kuinka paljon, tai mitkä ovat pelimaailman valolähteet. Toinen komponentti on kamera, eli määritetään, mikä on se alue, joka näytetään ja millainen on kameran linssi. Jos esimerkiksi kyseessä on ensimmäisen persoonan ammuskelupeli, kamera on sijoitettu pelaajan silmien kohdalle. [15]

Muutama seikka vielä itse varsinaisesta kuvan piirtämisestä. Kuva piirretään näytölle hyödyntäen esimerkiksi rasterisaatiota. Rasterisaatiolla tarkoitetaan kolmiulotteisen maailman muuttamista kaksiulotteiseksi. Jokainen kappale pelimaailmassa koostuu tietystä määrästä monikulmioita. Monikulmiot puolestaan koostuvat kolmioista. Nämä kolmiot lähetetään näytönohjaimelle, joka rasterisoi ne kaksiulotteiseksi. [19]

3.2 Fysiikka

Fysiikkamoottoria tarvitaan, jos peliin halutaan uskottavasti ja realistisesti käyttäytyviä objekteja. Fysiikkaa mallinnettaessa täytyy ottaa huomioon ainakin seuraavia asioita. Ensimmäiseksi täytyy tietää, miten jokin tietty kolmiulotteinen kappale on mallinnettu sekä pelimaailma, missä tämä malli esiintyy. Seuraavana ja ehkä tärkeimpänä kohtana on huolehtia objektien mahdollisimman realistisesta liikkumisesta tai siirtymisestä, sekä mahdollisesta pyörimisestä tietyn akselin suhteen. Usein peleissä on myös isoja liikkuvia tai liikkumattomia jäykkiä kappaleita kuten kivet tai puut, sekä hyvin muovautuvia tai joustavia ei-jäykkiä kappaleita kuten vaatteet. Täten fysiikkamoottorin olisi myös hyvä ottaa huomioon sekä jäykkien että joustavien kappaleiden dynamiikka. Reaalimaailman kappaleisiin vaikuttaa usein monia eri voimia esimerkiksi gravitaatio tai tukivoima. Liikkuviin objekteihin vaikuttaa myös kitka, ilmanvastus tai, jos liikutaan vedessä, vedenvastus. Näiden eri voimien vaikutus on tärkeää ottaa huomioon fysiikkamoottoria suunnitellessa. Toimiva fysiikkamoottori huolehtii myös törmäyksistä toisin sanoen, miten eri kappaleet käyttäytyvät törmätessään.

Fysiikkamoottorissa tärkeimmät matematiikan aihealueet ovat kolmion geometria, vektorilaskenta, matriisilaskenta ja differentiaaliyhtälöt. Kolmea ensimmäistä kuitenkin tarvitaan enemmän grafiikassa. [4] Differentiaaliyhtälöt fysiikkamallinnuksessa käsitellään ei-lineaarisina yhtälöinä. Näitä yhtälöitä kutsutaan myös nimellä tavalliset differentiaaliyhtälöt (engl. ODE). Yhtälöt ratkaistaan käyttämällä erinäisiä numeerisia menetelmiä. Käytetyimmät menetelmät ovat Eulerin menetelmä, keskipistemenetelmä ja neljännen asteen Runge-Kutta menetelmä. [6]

3.3 Ääni

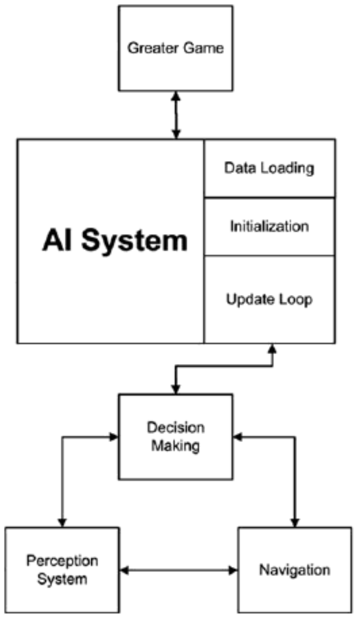
Yleensä äänien tallennus, toistaminen ja muokkaaminen sekä näistä huolehtiva äänimoottori jäävät vähemmälle huomiolle pelimoottoreissa. 1990-luvun peleissä äänien toistaminen oli hyvin yksinkertaista ja mitään erityistä moottoria tälle ei tarvittu. Nykyään pelien äänet ovat kuitenkin huomattavasti monimutkaisempia ja moninaisempia, joten tarve erilliselle äänimoottorille on kasvanut. Tämän moottorin tehtävänä on huolehtia esimerkiksi kolmiulotteisten äänien toistamisesta, ympäristöäänistä ja MP3-tiedostojen toistamisesta. Äänimoottorit, kuten grafiikkamoottorit, hyödyntävät tiettyä ohjelmointirajapintaa, joista yleisimmät ovat DirectX Audio ja OpenAL. [13]

Äänet peleissä voivat olla yksinkertaisesti taustamusiikkia, puhetta tai erinäisiä efektejä kuten räjähdysääniä, mitkä ovat peleissä mukana vain tunnelman lisäämiseksi. Toisaalta esimerkiksi pelissä Thief: Deadly Shadows äänet ovat integroitu fysiikan ja tekoälyn kanssa, jolloin tietyn äänen kuuluminen laukaisee jonkun tapahtuman. [15]

3.4 Tekoäly

Tekoäly peleissä mielletään usein yksinkertaisesti pelattavuudeksi tai esimerkiksi algoritmeiksi, jotka huolehtivat törmäysten hallinnasta. Parempi selitys tekoälylle, kuitenkin on seuraavanlainen. Tekoäly peleissä on koodia, joka huolehtii tietokoneen "pelaamien" hahmojen, esimerkiksi vihollisten, päätöksenteosta, jonka päämääränä on olla oleellinen, tehokas ja hyödyllinen. Täten tulee illuusio ajattelevasta hahmosta.

Tekoälymoottori koostuu usein vähintään seuraavanlaisista osista: päätöksenteko, havaitseminen ja navigointi. Ensimmäinen on tekoälymoottorin ydin. Sen tehtävänä on löytää järkevä tai järkevin ratkaisu annetusta, pelimaailmaa koskevasta, informaatiosta. Havaitsemisella tarkoitetaan asioita pelissä, mihin tekoälyn halutaan reagoivan esimerkiksi pelaajan paikka. Navigointia tarvitaan tekoälyn ohjaaman hahmon siirtämiseksi paikasta A paikkaan B. Navigointi jaetaan usein vielä reitinetsimiseksi ja esteiden välttelyksi. Kuvassa 4 esiintyvät nämä eri osat ja niiden suhteet. Ylempi osa (AI System kuvassa) toimii datan välittäjänä itse tekoälymoottorille (alimmat kolme suorakulmiota). [18]



Kuva 4: Tekoälymoottorin toiminta. Lähde: [18]

3.5 Skriptaus

Skriptauksella hallitaan tietyn ohjelman esimerkiksi pelin käyttäytymistä, ja skriptitiedostot sijaitsevat erillään varsinaisesta lähdekooditiedostosta. Skriptauksella luodaan peleihin sisältöä esimerkiksi, millaisia hahmoja pelissä on, ja miten ne käyttäytyvät. Mielestäni skriptauksen ja pelimoottorin suhdetta voi kuvata esimerkiksi seuraavin elokuvatermein: käsikirjoittaja ja ohjaaja ovat niin sanottuja elokuvan skriptaajia, näyttelijät ja tarina skriptejä ja kamerat, lavasteet ja puvut itse pelimoottori. Pelimoottorit kannattaakin suunnitella sellaisiksi, että ne sallivat skriptauksen. Tätä kutsutaan tieto-ohjautuvaksi (engl. data-driven) suunnitteluksi.

Skriptauksella voidaan saavuttaa muutamia hyötyjä. Esimerkiksi resurssitehokkuus on yksi näistä. Sillä tarkoitetaan sitä, että vältetään muistin tuhlaamista. Toinen etu on pelin suunnittelun nopeutuminen, kun tarvitsee vain muokata skriptitiedostoa, eikä kääntää koko ohjelmaa uudelleen.

Vaikka skriptausta voi tehdä graafisella editorilla, monipuolisimman hyödyn kuitenkin skriptauksesta saa, kun käyttää jotain skriptauskieltä. Tunnetuin kieli pelimoottoreissa lienee Lua. Muita kieliä ovat esimerkiksi Python ja BASIC. [15]

4 Reaalimaailman simulointi pelimoottoreissa

Reaalimaailman simuloinnilla on tärkeä rooli reaaliaikaisissa peleissä, jotka ovat esimerkkejä reaaliaikaisesta järjestelmästä, kuten esimerkiksi FPS-peleissä (Crysis), autokilpailupeleissä (GTR), reaaliaikaisissa strategiapeleissä (Command and Conquer) ja useimmissa uusissa roolipeleissä (TES V: Skyrim). Näissä pelitapahtumat sijoittuvat usein vuorovaikutteiseen tapahtumaympäristöön, jolloin pelimaailman realistinen olemus on tärkeää pelinautinnon takaamiseksi. Tämä nautinto saadaan aikaiseksi mahdollisimman tarkalla reaalimaailman simuloinnilla, joka esimerkiksi määrittää, miten objektit liikkuvat ja käyttäytyvät fysikaalisesti pelimaailmassa. Toisaalta pitää ottaa huomioon myös tehokkuusseikat, koska pelit ovat reaaliaikaisia järjestelmiä. Täten simulointi ei saa olla liian tarkka eikä liikaa laskentatehoja vaativa. Yleensä pätee periaate: jos se näyttää oikealta, se on oikea [4]. Esittelen seuraavaksi kolme tärkeintä reaalimaailman simuloinnin osaa: törmäysten havaitseminen ja hallinta, jäykkien kappaleiden dynamiikka ja ei-jäykkien kappaleiden dynamiikka.

4.1 Törmäysten havaitseminen ja hallinta

Törmäyksellä tarkoitetaan luonnollisesti kahden tai useamman kappaleen osumista tai koskettamista toisiinsa. Pelimoottoreissa ja simuloinnissa törmäysten hallinta koostuu kahdesta osasta: törmäysten havaitsemisesta ja törmäyksiin reagoimisesta [4]. Peleissä eri objektien törmäyksiä on ollut jo Pongista ja Space Invaderista lähtien.

Objektit peleissä voidaan jakaa liikkuviin ja paikalla oleviin. Tämän perusteella myös voidaan jakaa törmäyksen havaitsemiseen liittyvät kyselyt (engl. query). Näitä kyselyitä on kahdenlaisia. Ensimmäisessä kyselyssä täytyy selvittää ylipäätään, ovatko kappaleet kosketuksissa toisiinsa. Kahden liikkumattoman välillä tämä on suoraviivaista mutta tapauksessa, jossa vähintään toinen liikkuu, pitää tutkia, tapahtuuko törmäys tietyllä aikavälillä. Toisessa törmäysten havaitsemiseen liittyvässä kyselyssä etsitään toisiinsa koskettavien kappaleiden välinen leikkausjoukko eli siis ne pisteet, joissa törmäys tapahtuu. Tämän selvittäminen on sekä liikkuvien että liikkumattomien kappaleiden välillä samankaltaista. Nämä kyselyt voidaan selvittää edelleen kahdella eri tavalla: leikkauksiin tai etäisyyksiin pohjautuvalla tavalla. [6] Näiden tarkka kuvaus löytyy äsken mainitusta lähteestä.

Törmäysten etsimiseksi kappaleelle täytyy määrittää jokin rajaava tai reunustava geometrinen objekti (engl. bounding). Kolme yleisintä geometrista objektia ovat pallo, (ympyräpohjainen) lieriö ja särmiö (tai laatikko). Tällainen objekti on siis pienin mahdollinen objekti, joka mahtuu pelissä esiintyvän tietyn kappaleen ympärille. Törmäysten selvittämiseksi pallojen tapauksessa täytyy määrittää pallojen keskipisteiden välinen etäisyys ja se, onko kummankaan pallon säde suurempi kuin toisen pallon säde vähennettynä etäisyydestä. Jos on, kyseessä on törmäys. Lieriöiden tapauksessa täytyy ottaa huomioon sekä pohjan säde että lieriön korkeus. Rajaavia särmiöitä on kahdenlaisia: akseliin kiinnitettyjä särmiöitä (engl. axis-aligned bounding box) ja suuntautuneita särmiöitä (engl. oriented box). Särmiöiden tapauksessa riittää tutkia, onko jokin särmiön kulmista toisen rajaavan kappaleen sisällä.

Jos peliobjekti on monimutkainen ja halutaan lisätä tarkkuutta, kannattaa peliobjektin rajaava kappale jakaa osiin. Ihmisen muotoisen kappaleen tapauksessa tämä tarkoittaisi sitä, että esimerkiksi päälle, keskivartalolle ja raajoille olisi omat rajaavat kappaleet. Tehokkuussyistä voidaan myös säilyttää alkuperäinen jakamaton rajaava kappale, jolloin mahdollisen törmäyksen tutkimiseen riittää katsoa vain näitä jakamattomia rajaavia kappaleita. [4] Nämä siis pätevät konvekseille eli kuperille kappaleille. Konkaavit eli koverat kappaleet pitää ensin jakaa niin moneksi kuperaksi kappaleeksi, että kappale ei ole enää kovera. Tämä tehostaa törmäyksiin liittyvää laskemista huomattavasti. [1]

Törmäyksiin reagoimisen simuloinnissa hyödynnetään liikemäärää $p = mv$ ja liikemäärän säilymislakia $p_1 + p_2 = u_1 + u_2$, jossa p:t ovat kappaleiden liikemäärät ennen törmäystä ja u:t sen jälkeen. Lisäksi tarvitaan vielä energian säilymislaki $K_1 + K_2 = L_1 + L_2$, jossa K:t ovat kappaleiden liike-energiat ennen törmäystä ja L:t sen jälkeen. Näiden avulla saadaan kahden törmäävän kappaleen tapauksessa kaavat $w_1 = v_1 + (w_{1\mathrm{p}} - v_{1\mathrm{p}})n$ ja $w_2 = v_2 + (w_{2\mathrm{p}} - v_{2\mathrm{p}})n$, jotka ovat lopulliset nopeusvektorit törmäyksen jälkeen (johto: [4]). w:t ovat siis nopeusvektorit törmäyksen jälkeen, v:t ovat nopeusvektorit ennen törmäystä, w:t ja v:t alaindeksillä p ovat ulottuvuuteen yksi projisioituja vektoreita ja n projisioinnissa tarvittava yksikkövektori. Usein tarvitaan ottaa huomioon myös pyörimisliike. Koska tämän tutkimuksen aiheena ei ole varsinaisesti fysiikka, löytyy tämä kaava ja sen johto lähteestä [4].

Törmäysten reagointiin liittyy myös usein impulssien laskeminen törmääville kappaleille. Törmäyksen jälkeen sekä kappaleiden suhteelliset lineaariset nopeudet että kulmanopeudet muuttuvat. [16]

Törmäyskyselyt voidaan jakaa myös diskreetteihin ja jatkuviin, mihin pätevät samat ideat kuin diskreeteissä tapahtumasimuloinneissa ja jatkuvissa simuloinneissa. Diskreetit törmäyskyselyt tapahtuvat välittömässä ajassa. Ne voivat olla yksinkertaisia leikkaustestejä eli siis tutkitaan, leikkaavatko kappaleet toisensa. Käytetyin algoritmi diskreeteissä törmäyskyselyissä lienee GJK-algoritmi (eräs esitys lähteessä [2]). Se ottaa huomioon leikkaustestien lisäksi myös kappaleiden väliset etäisyydet, jolloin saadaan selville törmäävien kappaleiden lähimmät pisteet. Diskreetit törmäyskyselyt ovat usein riittäviä esimerkiksi peleissä mutta, jos halutaan lisää tarkkuutta, tarvitaan jatkuvia törmäyskyselyjä. Tämä epätarkkuus johtuu ajallisesta laskostumisesta eli siitä, että jos kappale liikkuu liian nopeasti, niin se saattaa liikkua toisen kappaleen läpi. Tällaista ongelmaa ei ole jatkuvissa törmäyskyselyissä. Ideana näissä on siis ottaa huomioon kappaleiden liikerata tietyllä aikavälillä ja näin selvittää aika, jolloin törmäys tapahtuu. Törmäyksen havaitseminen toteutetaan siis kappaleen oletetulla liikeradalla eli jatkuvat törmäyskyselyt ovat ikään kuin askeleen edellä. [1]

4.2 Jäykkien kappaleiden dynamiikka

Lähes kaikki objektit pelimaailmassa ovat jäykkiä tai ei-jäykkiä kappaleita. Jäykkiä kappaleita ovat esimerkiksi usein itse pelaaja, maastossa kasvava puu tai ajoneuvo. Jäykkiä kappaleita yhdistävät seuraavat ominaisuudet: paikka, nopeus ja vaikuttavat voimat sekä koko ja suunta, jos kappale ei ole pistemäinen. Vakiintunut esitystapa jäykälle kappaleelle on monitahokas. Matemaattisesta syistä simuloinnin helpottamiseksi valitaan jäykän kappaleen origoksi massakeskipiste. [6]

Jäykät kappaleet säilyttävät aina muotonsa. Vaikka yksikään reaalimaailman objekti ei ole puhtaasti jäykkä kappale, niin simuloinnissa tiettyjen kappaleiden olettaminen jäykäksi on usein tarpeen. Esimerkiksi tennispallo muuttaa muotoaan litteäksi, kun sitä lyö mailalla mutta simuloinnissa tennispalloa voi kuvata kuitenkin jäykkänä kappaleena. Jäykkiä kappaleita voidaan ajatella myös joukkona tietyn massaisina hiukkasina. Näiden hiukkasten väliset etäisyydet toisistaan pysyvät koko ajan vakiona, jolloin kappale ei muuta muotoaan ja on jäykkä. Tärkeää täten on laskea kyseessä olevien hiukkasten yhteinen massakeskipiste. Täten kappaleeseen vaikuttava voima voidaan selvittää käyttämällä kaavaa $F = M\frac{\mathrm{d}^2X_{\mathrm{cm}}}{\mathrm{d}t^2}$, jossa X_{cm} on kappaleen massakeskipisteen paikka. [4]

Jäykkien kappaleiden dynamiikan selvittämiseksi voidaan käyttää ainakin viittä eri simulointiparadigmaa: rajoitepohjaista, impulssipohjaista, rangaistuspohjaista ja hybridipohjaista paradigmaa sekä törmäyksen synkronointimenetelmää. Keskityn seuraavaksi rajoitepohjaiseen ja impulssipohjaiseen paradigmaan. [7]

Yleisesti ottaen liikeyhtälöinä jäykille kappaleille pätee kaava $F = ma$ liikkeelle ja pyörimiselle kaava $\tau = J\omega + J \times \omega$, jossa τ on kappaleeseen vaikuttava momentti, J hitausmomentti ja ω kulmanopeus [20].

Rajoitepohjaisessa paradigmassa tarkoitus on laskea rajoitteiden aiheuttamat voimat kappaleille. Rajoite tarkoittaa tässä tapauksessa estettä, jonka läpi kappale ei pääse. Esimerkiksi pallon koskettaessa seinää seinä toimii rajoitteena pallolle. Tarkoituksena on ensiksi selvittää esimerkiksi kosketustilanteessa kappaleiden väliset suhteelliset nopeudet kosketuspisteen tai kosketuspisteiden suhteen. Suhteelliselle nopeudelle saadaan kaava $s = Ru$, jossa R on s:n suuntainen rajoite ja u lineaarisen ja kulmanopeuden yhdistävä vektori.

Rajoitepohjainen simulointi ei salli sitä, että jäykkien kappaleiden koskettaessa, ne lävistäisivät toisiaan. Tämä toteutuu, kun $Ru \geq 0$. Kun määritetään vielä kappaleiden massat, massan jakaantumisen ja hitausmomentit huomioon ottava massamatriisi ja kappaleiden väliset kosketuspisteet huomioon ottava rajoitematriisi, saadaan rajoiteyhtälö selville. [7] Itse kaava ja sen johto löytyvät äsken mainitusta lähteestä.

Rajoitepohjaiseen simulointiin liittyvät myös vahvasti lineaariset vastavuoroiset ongelmat, LVP (engl. linear complementary problems, LCP). Käytännössä siis rajoitepohjainen simulointi on näiden ongelmien ratkaisemista. LVP on kuitenkin tämän työn laajuuden ulkopuolella mutta lisätietoa niistä saa lähteestä [1].

Impulssipohjainen paradigma perustuu sellaiseen olettamukseen, että kaikki kosketukset kappaleiden välillä ovat sarja törmäyksiä. Jokainen törmäys aiheuttaa törmääville kappaleille pienen impulssin. Tässä paradigmassa käsitettä rajoite ei ole olemassa. Jatkuvat kosketukset ovat myös sarja törmäyksiä esimerkiksi tapauksessa, jossa kirja on pöydän päällä.

Impulssipohjaisen simuloinnin tai simulointiparadigman toiminta koostuu pääosin seuraavista kolmesta vaiheesta. Ensiksi täytyy määrittää maksimi aikaväli, jossa järjestelmä on yhtenäinen siten, että yhtäkään kahden tai useamman kappaleen kosketusta tai törmäystä ei jää huomaamatta. Tärkeintä on tarkistaa kriittinen pari siis pari, jonka törmäyksen tapahtumiseen kuluu vähiten aikaa. Toiseksi järjestelmää päivitetään ajassa ja samalla yhdistetään eri kappaleiden liikeyhtälöt. Kolmanneksi tarkistetaan, onko kriittisen parin välillä tapahtunut törmäystä. Jos on, lasketaan ja asetetaan impulssit tämän parin välille.

Koska impulssipohjaisessa simuloinnissa ei esiinny rajoitteita, kuten rajoitepohjaisessa simuloinnissa, niin jokainen kappale liikkuu itsenäisesti. Täten laskennan rinnakkaistaminen eri säikeille tai prosessoreille on verrattain helppoa. [16]

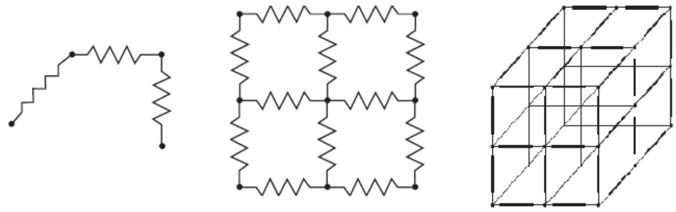
4.3 Ei-jäykkien kappaleiden dynamiikka

Ei-jäykkiä kappaleita (engl. soft body, deformable object) ovat sellaiset esineet kuten köydet, hiukset, vaatekappaleet tai hyytelöt. Ei-jäykät kappaleet voidaan jakaa kolmeen eri ulottuvuusluokkaan. Ensimmäiseen luokkaan, kaarimassaisiin, kuuluvat yksiulotteiset kappaleet kuten köydet tai hiukset. Toinen luokka, pintamassaiset, koostuu kaksiulotteisista kappaleista kuten vaatteet tai veden pinta. Kolmanteen luokkaan, tilavuusmassaisiin, kuuluvat esimerkiksi hyytelömäiset rakenteet. [6]

Pelimoottorien alkuaikoina ei-jäykät kappaleet jätettiin yleensä huomiotta tai niitä simuloitiin jäykkinä kappaleina. Esimerkiksi pelihahmojen hiukset olivat usein vain liikkumaton, kiinteä osa hahmossa. Kuitenkin nykyään lähes kaikissa peli -tai fysiikkamoottoreissa on tuki suhteellisen realistisille ei-jäykille kappaleille. [4]

On monia tapoja simuloida ei-jäykkiä kappaleita. Yksi näistä on rajallisen osan menetelmä (engl. finite element method). Siinä kappale tai oikeastaan kappaleen matemaattisen mallin määrittelyjoukko jaetaan pienempiin osiin, joilla jokaisella on oma itseään kuvaava funktio. Toinen menetelmä on avaruudellisesti yhdistetyt hiukkasjärjestelmät, jossa ei-jäykkä kappale koostuu hiukkasista, jotka pyrkivät pitämään tietyn etäisyyden . Kolmas tapa, massa-jousi-järjestelmät, on sopivin reaaliaikaisille sovelluksille kuten peleille. [11]

Jousilla siis on peleissä paljon tärkeämpikin rooli kuin olla vain pelaajan käyttämänä hyppykeppinä tai osana auton alustaa. Massa-jousi järjestelmässä ei-jäykät kappaleet koostuvat verkosta, jossa solmuina ovat massapisteet ja kaarina ovat jouset. Tämä järjestelmä pohjautuu fysiikasta tuttuun Hooken lakiin $F = -kx$ tai paranneltuna versiona, joka ottaa huomioon vaimennuksen: $F = -kx - bv$. Kuvassa 5 on havainnollistettu massa-jousi järjestelmän eri tapaukset. [4]



Kuva 5: Vasemmalla kaarimassainen, keskellä pintamassainen ja oikealla tilavuusmassainen massa-jousi järjestelmä.

Esimerkkinä massa-jousi-systeemistä voidaan pitää vaatekappaleen simulointia. Tässä vaatekappale koostuu joukosta hiukkasia, joilla on massa mutta ei tilavuutta. Näiden hiukkasten välillä on massattomia jousia. Näitä jousia on kolmea eri tyyppiä. Rakenteelliset jouset määräävät vaatekappaleen venyvyyden. Leikkausjouset estävät vaatekappaletta repeytymästä kahtia. Taivutusjouset taas estävät vaatekappaleen liiallisen luhistumisen. [11]

5 Simuloinnin toteutus Bullet-moottorilla

Esittelen seuraavaksi, miten edellisen luvun käsitteet ja teoriat ilmenevät käytännön esimerkissä. Bullet on paljon muun muassa peleissä tai pelimoottoreissa, animaattoreissa ja ylipäätään simuloinnissa käytetty avoimen lähdekoodin fysiikkamoottori. Esimerkiksi Rockstarin pelimoottori RAGE hyödyntää Bulletia kolmannen osapuolen väliohjelmistona. Tässä pelimoottorissa ja muissakin Bullet toimii itsenäisenä osana tarjoten palvelun mahdollisimman realistiselle fysiikan simuloinnille. Muun muassa pelit kuten Grand Theft Auto IV ja Red Dead Redemption hyödyntävät RAGE:a ja Bulletia. Käytän tässä koko luvussa lähdettä [5] ellen toisin mainitse.

5.1 Arkkitehtuuri

Bulletin arkkitehtuuri koostuu kuudesta eri komponentista: matala taso, törmäyksen havaitseminen, jäykän kappaleen dynamiikka, ei-jäykän kappaleen dynamiikka, monisäikeisyys ja liitännäiset. Kuvassa 6 on graafinen esitys Bulletin arkkitehtuurista. Tämä arkkitehtuuri toimii siten, että jokainen kuvassa ylempänä esiintyvää komponenttia käyttävä kappale käyttää myös jokaista alempaa komponenttia. Esimerkiksi ei-jäykän kappaleen komponenttia hyödyntävä kappale käyttää myös jäykän kappaleen komponenttia, törmäyksen havaitsemisen komponenttia ja matalan tason komponenttia.

Kuva 6: Bulletin arkkitehtuuri.

Matalalla tasolla tarkoitetaan tässä yhteydessä lineaarialgebrasta, muistin varaamisesta ja tietorakenteista (säiliö) huolehtivasta komponentista. Käytännössä tässä komponentissa määritellään sellaiset lineaarialgebran asiat kuten skalaarit, vektorit ja matriisit, joita ylemmän tason komponentit hyödyntävät. Matalan tason komponentissa myös huolehditaan muistin varaamisesta kappaleille sekä muistin tasauksesta (engl. align) 16:lle tavulle pelikonsoleita varten. Matalan tason komponentti tarjoaa myös yhteensopivuussyistä oman tietorakenneluokan kappaleiden säilyttämiselle.

Bulletin törmäysten havaitsemisen komponentti liittää kappaleeseen törmäysmuodon ja tarjoaa tehokkaita algoritmeja kuten GJK törmäysten havaitsemiseksi. Jäykän kappaleen dynamiikan komponentti huolehtii kappaleen liikkeestä ja liikeradoista. Se lisää kappaleille massan, nopeuden, hitauden ja rajoitteet sekä siihen vaikuttavat voimat. Bulletin ei-jäykkien kappaleiden komponentti tarjoaa simulointimahdollisuutta sellaisille esineille kuten köysille, vaatekappaleille ja geelimäisille kappaleille. Bulletissa on myös tuki monisäikeisyydelle ja erinäisille mallinnuslisäosille.

5.2 Simulaatiomekanismi

Simulaatioaskel Bulletissa tapahtuu seuraavin vaihein. Ensiksi on dynamiikan edelleen lähettäminen (engl. forwarding), johon kuuluvat painovoiman vaikutuksen lisääminen kappaleeseen ja muutosten (engl. transform) ennustaminen. Nämä hyödyntävät tietoa nopeuden muutoksesta. Tämän jälkeen tehdään lavea (engl. broadphase) -ja tarkkavaiheinen (engl. narrowphase) törmäysten havaitseminen. Toinen voidaan kytkeä myös pois päältä. Laveavaiheiseen törmäysten havaitsemiseen kuuluvat akseliin kiinnitettyjen särmiöiden laskeminen ja mahdollisten toisiaan koskettavien kappaleiden etsiminen. Tarkkavaiheisessa törmäysten havaitsemisessa lasketaan toisiaan koskettavien kappaleiden törmäyspisteet. Tähän tarvitaan löytää sopiva algoritmi seuraavassa kappaleessa esitettävän törmäysmatriisin avulla. Lopuksi palataan dynamiikan edelleen lähettämiseen, jossa tällä kertaa selvitetään kappaleeseen vaikuttavat rajoitteet ja kappaleen mahdollinen uusi paikka. Tämä tapahtumaketju tai simulaatioaskel on siis luonteeltaan diskreetti ja se tapahtuu tietyllä aikavälillä, joka Bulletissa on joko vaihtuva tai vakio 1/60 sekuntia.

Sopivan algoritmin etsimiseksi Bullet hyödyntää törmäysmatriisia. Törmäysmatriisi on neliömatriisi ja sen sarakkeina/riveinä ovat eri primitiivimuodot, kolmioverkko ja yhdistelmämuoto (lisää näistä seuraavassa aliluvussa). Alkioiden arvona on tietty algoritmi, joka esimerkiksi kuperien kappaleiden tapauksessa on GJK-algoritmi. Tämä algoritmi lopulta määrää mitkä kahden kappaleen väliset pisteet törmäävät. Tämä algoritmi myös rekisteröidään suoritukseenlaittajalle (engl. dispatcher).

5.3 Kappaleiden mallinnus

Kappaleiden mallinnus Bulletissa lähtee liikkeelle aina sopivan törmäysmuodon valitsemisella. Sopivan törmäysmuodon tai toiselta nimeltään aiemmin esiintynyt kappaleen rajaavan geometrisen objektin etsimiseksi voidaan soveltaa seuraavaa tapaa. Ensiksi täytyy tietää, onko kappale liikkumaton tai liikkuva. Liikkumattoman kappaleen tapauksessa törmäysmuodoksi voidaan valita korkeuskenttä (maastolle) tai kolmioista koostuva verkko (kolmioverkko) (engl. triangle mesh). Yksinkertaisille liikkuville kappaleille törmäysmuodoksi voidaan asettaa kappaleesta approksimoitu primitiivimuoto. Bulletissa on seuraavat primitiivimuodot: laatikko, pallo, kapseli, lieriö ja kartio. Monimutkaisemmille kappaleille törmäysmuodoksi voidaan approksimoida kolmioverkosta muodostettu konveksiverho (engl. convex hull). Konveksiverhoon kuuluvat kaikki tiettyjen pisteiden, tässä tapauksessa kolmioiden, muodostaman kuperan kappaleen sisällä olevat pisteet. Jos tarkkuutta halutaan lisätä, liikkuvan kappaleen törmäysmuodoksi voidaan asettaa myös yhdistelmämuoto, joka koostuu useasta eri törmäysmuodoista. Käyttäjä voi myös määrittää täysin oman törmäysmuodon.

Bulletissa jäykät kappaleet jaetaan kolmeen luokkaan: dynaamisiin, staattisiin ja kinemaattisiin. Dynaamiset kappaleet ovat kappaleita, jotka liikkuvat esimerkiksi ammukset. Dynaamisilla kappaleilla on positiivinen massa ja kappaleen tilaa päivitetään joka simulointiaskeleella. Staattiset kappaleilla ei ole massaa ja ne saavat käsitellä törmäyksiä mutta eivät liikkua. Seinät ja kenties maasto on esimerkkejä staattisista jäykistä kappaleista. Kinemaattiset kappaleet ovat myös nollamassaisia mutta käyttäjä (pelikehittäjä) voi liikuttaa niitä esimerkiksi animaatioissa.

Kaikki jäykän kappaleen dynamiikan operaatiot esimerkiksi massan ja voimien lisäys tehdään kappaleen massakeskipisteelle. Massakeskipiste myös määrittää kappaleen inertiaalikoordinaatiston siis sen, miten kappale reagoi erisuuruisiin voimiin. Tälle jäykän kappaleen mallille ei kuitenkaan saisi tehdä operaatioita kuten skaalaus tai leikkaus. Nämä operaatiot tulisi tehdä kappaleen törmäysmuodolle. Jäykän kappaleen maailmankoordinaattien muunnos siis kappaleen paikallisten koordinaattien muuntaminen pelimaailman koordinaatistoon tapahtuu kappaleen massakeskipisteen mukaan. Muunnoksessa jäykkää kappaletta pidetään siis pistemäisenä objektina.

Bullet käyttää kappaleiden paikan päivittämiseen MotionStatea. MotionState huolehtii myös kappaleen maailmankoordinaattien muunnoksen renderöijän käyttämään koordinaatistoon.

Bulletissa on viidentyyppisiä rajoitteita, joita voidaan käyttää kappaleiden liikeratojen rajaamiseksi toisin sanoen vähentää kappaleen mahdollisia vapausasteita (kolme lineaariselle liikkeelle ja kolme pyörimisliikkeelle). Ensimmäinen on piste-piste rajoite, jossa kappaleet ovat toisissa kiinni yhdessä pisteessä. Esimerkiksi köysi, joka on kiinni katossa yhdessä pisteessä, toteuttaa piste-piste rajoitteen. Toinen rajoite on saranarajoite, joka rajoittaa kappaleen liikeradan vain yhteen kulmavapausasteeseen. Esimerkkinä voidaan pitää oven saranoita. Kolmas rajoite on liukurirajoite, jonka vaikutuksen alaisena kappaleet voivat liikkua vain yhdessä vapausasteessa. Neljäs rajoite on esimerkiksi räsynukkemallinnuksessa käytettävä kartiorajoite. Lisäksi on vielä viides rajoite, joka on yleinen kuuden vapausasteen rajoite. Siinä käyttäjä voi itse valita rajoitettavat vapausasteet.

Bulletissa ei-jäykkä kappale on kolmioverkko, jossa jokainen solmu on dynaaminen itsenäinen osa eli jokaiseen solmuun vaikuttaa omat voimat ja niissä voi olla omat massat. Toisin kuin jäykillä kappaleilla ei-jäykillä kappaleilla ei ole vain yhtä pistettä, johon kohdistuu maailmankoordinaattien muunnos, vaan sillä on useita pisteitä (kolmioverkon solmut) eli ei-jäykissä kappaleissa ei lasketa massakeskipistettä. Jokaiseen ei-jäykän kappaleen muodostavan kolmioverkon verteksiin voidaan myös laittaa tietty rajoite tai kiinnittää jokin tai jotkut ei-jäykän kappaleen verteksit jäykkään kappaleeseen.

5.4 Bullet fysiikkamoottorina

Kaiken kaikkiaan Bullet on varsin tarkka fysiikkamoottori pelimoottoreihin. Usein reaaliaikaiset pelit ovat nopeatempoisia, joten tarkkuutta vaaditaan suurilla päivitysnopeuksilla. Bullet onkin tarkimmillaan suurilla päivitysnopeuksilla [3], joten se on erinomainen valinta peleille. Tarkkuutta Bulletissa lisäävät varsinkin törmäysten havaitsemisessa käytettävät konveksiverhot. Niiden avulla pystytään luomaan hyvinkin yksityiskohtaisia törmäysmuotoja.

Varsinaisesti mitään suurempia kehityskohteita Bulletissa ei juurikaan ole. Tietenkin yksittäisissä tapauksissa kuten esimerkiksi lähteessä [3] esiintyvässä ympyröiden pinonta testissä, jonka kaikki lähteessä esiintyvät fysiikkamoottorit epäonnistuivat, on parantamisen varaa. Ylipäätään tarkkuutta voidaan aina lisätä uusien versioiden myötä mutta Bullet on kuitenkin hyvin onnistunut fysiikkamoottori.

6 Yhteenveto

Simuloinnilla on siis keskeinen rooli reaaliaikaisissa peleissä. Simulointi on kehittynyt merkittävästi viime vuosikymmenien aikana aina yksinkertaisista törmäyskyselyistä hyödyntävästä SpaceInvaderista lähes realistisesti maailmaa mallintavaan Crysikseen. Nykyään lähes kaikki laitteet pöytätietokoneista kännyköihin pystyvät suorittamaan pelimoottoreissa tapahtuvaa simulointeja, jolloin hyvin yksinkertaisissa ja pienissäkin peleissä pystytään jo simuloimaan realistisesti käyttäytyviä objekteja.

Vaikka simulointi pelimoottoreissa on teoreettisesti jatkuvatoimista, niin käytännössä se on diskreetti tai diskretisoitu. Tämä johtuu siitä yksinkertaisesta syystä, että kaikki tietokoneella tehtävä laskenta on aina diskreettiä kellosyklein tapahtuvaa. Simulaatiolla pelimoottoreissa tarkoitetaan usein reaalimaailman simulointia, joka voidaan jakaa karkeasti kolmeen osa-alueeseen: törmäysten havainnointiin ja hallintaan, jäykkien kappaleiden dynamiikkaan ja ei-jäykkien kappaleiden dynamiikkaan. Näiden osa-alueiden toiminta käy hyvin ilmi Bullet-nimisestä käytännön fysiikkamoottorista.

Kaiken kaikkiaan reaalimaailman simuloinnilla tulee olemaan kasvava osa pelimoottoreissa. Kenties simulointia voitaisiin hyödyntää tai upottaa sisään muihinkin pelimoottorin osamoottoreihin kuin vain fysiikkamoottoriin. Esimerkiksi simuloinnissa tarvittavaa laskentaa voitaisiin yksinkertaistaa, jos vaikka peliobjektin fysikaaliset ominaisuudet olisivat valmiiksi laskettu objektin mallia tehdessä. Siis jos esimerkiksi tietyllä mallintekoeditorilla tehtäisiin pallo, olisi jo tässä vaiheessa tiedossa pallon fysikaaliset ominaisuudet eikä niitä tarvitsisi laskea aina uudestaan simuloinnin yhteydessä. Toisaalta simulointia voitaisiin tehostaa myös liittämällä siinä tarvittava laskenta suoraan omaan laitteistotason komponenttiin. Näin oli itseasiassa tehtykin AGEIA PhysX fysiikkalaajennuskortin avulla. Bulletissa valitettavasti näin ei ole, vaan laskenta suoritetaan normaalilla suorittimella. Olisikin mielenkiintoista nähdä sellaisten fysiikkalaajennuskorttien ilmestymisen ja yleistymisen, jotka pystyvät hyödyntämään mitä tahansa fysiikkamoottoria.

Tarkkuutta fysiikkamoottoreihin ja simulointiin saataisiin lisää, jos simuloitavat kappaleet sinänsä koostuisivat atomeista tai edelleen kvarkeista ja elektroneista. Kappaleiden rajaavat törmäysmuodot ja dynamiikan laskenta pätisivätkin vain näille alkeishiukkasille.

Kaikki näistä koostuvat kappaleet siis käytännössä kaikki pelimaailman objektit vuorovaikuttaisivat toistensa kanssa alkeishiukkasten perusteella. Esimerkiksi tennispalloa simuloitaessa sen käyttäytyminen ei määräytyisi pallosta itsestään, vaan siitä koostuvista atomeista. Suhteellisuusteoriankin huomioon ottamista voitaisiin myös harkita. Tällainen alkeishiukkassimulointi olisi luonnollisesti äärimmäisen raskasta mutta laskentateho kasvaa jatkuvasti ja esimerkiksi kvanttitietokoneiden avulla tällainen simulointi saattaisi olla jonakin päivänä mahdollista.

Lähteet

[1] J. Bender, K. Erleben, J. Trinkle ja E. Coumans. Interactive simulation of rigid body dynamics in computer graphics. *EUROGRAPHICS 2012 State of the Art Reports*, sivut 28–30. Eurographics Association, 2012. Saatavissa `http://i31www.ira.uka.de/~jbender/Papers/STAR.pdf`. Viitattu 10.4.2012.

[2] W. Bittle. Gjk (gilbert–johnson–keerthi), 2010. Saatavissa `http://www.codezealot.org/archives/88`. Viitattu 25.4.2012.

[3] A. Boeing ja T. Bräunl. Evaluation of real-time physics simulation systems. *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia GRAPHITE 07*, 1:8, 2007, sivu 8. doi: 10.1145/1321261.1321312. URL `http://portal.acm.org/citation.cfm?doid=1321261.1321312`.

[4] D. Conger. *Physics Modeling for Game Programmers*. Course Technolgy, Boston, MA, 2004, sivut 3–8, 173–190, 201–204, 235–246, 297–303. ISBN 9781592000937. Saatavissa `http://site.ebrary.com/lib/aalto/docDetail.action?docID=10065752`. Viitattu 15.3.2012.

[5] E. Coumans. *Bullet 2.80 Physics SDK Manual*, 2012, sivut 11–12, 17–30, 32–33, 42.

[6] D. H. Eberly. *3D Game Engine Architecture*. Morgan Kaufmann, San Francisco, CA, 2005, sivut 149–150, 487–491, 565–572, 580, 592. ISBN 0-12-229064-X.

[7] K. Erleben. *Stable, Robust, and Versatile Multibody Dynamics Animation*. Väitöskirja, University of Copenhagen, Kööpenhamina, 2005, sivut 20–23. Saatavissa `http://image.diku.dk/kenny/download/erleben.05.thesis.pdf`. Viitattu 11.4.2012.

[8] P.a. Fishwick. Computer simulation. *Potentials, IEEE*, 15:24–27, 1996, sivut 24–27. ISSN 0278-6648. doi: 10.1109/45.481372. Saatavissa `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=481372`. Viitattu 10.2.2012.

[9] J. Gregory. Runtime game engine architecture, 2009. Saatavissa `http://www.gameenginebook.com/Fig-RuntimeArch.jpg`. Viitattu 25.4.2012.

[10] K. Helsgaun. Discrete event simulation in java, 2000, sivut 3–6. Saatavissa `http://www.akira.ruc.dk/~keld/research/javasimulation/JAVASIMULATION-1.1/docs/Report.pdf`. Viitattu 10.2.2012.

[11] D. Hyunh. Cloth simulation using hardware tessellation. Opinnäytetyö tieteen maisteriksi, Rochester Institute of Technology, Rochester, New York, 2011, sivut 4–5. Saatavissa http://search.proquest.com/docview/861741907/fulltextPDF?source=fedsrch&accountid=27468#. Viitattu 12.4.2012.

[12] B.P. Kwok. Continuous simulation. *SIGSIM Simul. Dig.*, 11:55–57, 1979, sivut 55–57. ISSN 0163-6103. doi: 10.1145/1102838.1102847. Saatavissa http://doi.acm.org/10.1145/1102838.1102847. Viitattu 10.2.2012.

[13] M. McCuskey. *Beginning Game Audio Programming.* Course Technolgy, Boston, MA, 2008, sivu xxvii. ISBN 9781592000296. Saatavissa http://site.ebrary.com/lib/aalto/docDetail.action?docID=10064354. Viitattu 15.3.2012.

[14] R McHaney. *Understanding Computer Simulation.* Roger McHaney sekä Ventus Publishing ApS, 2009, sivut 9–10, 15, 20. ISBN 978-87-7681-505-9. Saatavissa http://www.scribd.com/doc/47243583/5/Continuous-Simulation. Viitattu 10.2.2012.

[15] M. McShaffry. *Game Coding Complete.* Charles River Media, Boston, MA, kolmas painos, 2009, sivut 306–309, 354, 408–409. ISBN 9781584506805. Saatavissa http://site.ebrary.com/lib/aalto/docDetail.action?docID=10314625. Viitattu 15.3.2012.

[16] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* Väitöskirja, University of California, Berkeley, 1996, sivut 9–15, 54. Saatavissa http://www.kuffner.org/james/software/dynamics/mirtich/mirtichThesis.pdf. Viitattu 10.4.2012.

[17] P.F. Roth. Discrete, continuous, and combined simulation. *Proceedings of the 20th conference on Winter simulation*, WSC '88, sivut 56–60, New York, NY, USA, 1988. ACM. ISBN 0-911801-42-1. doi: 10.1145/318123.318154. Saatavissa http://dl.acm.org/citation.cfm?id=318371.318376&coll=DL&dl=ACM&CFID=79677687&CFTOKEN=49290220. Viitattu 10.2.2012.

[18] B. Schwab. *AI Game Engine Programming.* Course Technolgy, Boston, MA, toinen painos, 2008, sivut 2–5, 31–32, 41. ISBN 9781584505723. Saatavissa http://site.ebrary.com/lib/aalto/docDetail.action?docID=10365197. Viitattu 15.3.2012.

[19] A. Sherrod. *Game Graphics Programming.* Course Technolgy, Boston, MA, 2008, sivut 9, 11. ISBN 9781584505167. Saatavissa http://site.ebrary.com/lib/aalto/docDetail.action?docID=10251034. Viitattu 15.3.2012.

[20] J. Tsuda. Practical rigid body physics for games. *ACM SIGGRAPH ASIA 2009 Courses*, SIGGRAPH ASIA '09, sivut 1–83, New York, NY, USA, 2009.

ACM. doi: 10.1145/1665817.1665831. Saatavissa http://doi.acm.org/10.1145/1665817.1665831. Viitattu 11.4.2012.

[21] S. Zerbst ja O. Düvel. *3D Game Engine Programming.* Course Technology Crisp, Boston, MA, 2004, sivu 4. ISBN 9781592003518. Saatavissa http://site.ebrary.com/lib/aalto/docDetail.action?docID=10063412. Viitattu 15.3.2012.