Tuukka Varjus

Törmäystarkastelu reaaliaikaisissa sovelluksissa

Tietotekniikan kandidaatintutkielma

18. kesäkuuta 2018

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Tuukka Varjus

Yhteystiedot: tuukkavarjus@gmail.com

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Törmäystarkastelu reaaliaikaisissa sovelluksissa

Title in English: Collision detection in real-time applications

Työ: Kandidaatintutkielma

Sivumäärä: 72+0

Tiivistelmä: Tutkielman tarkoituksena on esitellä reaaliaikaisissa sovelluksissa käytössä olevia menetelmiä ja algoritmeja törmäystarkastelun toteuttamiseen. Tutkielmassa käytetty tutkimusmenetelmä on kirjallisuuskatsaus. Lähteinä ovat toimineet niin alan kirjallisuus ja artikkelit, kuin tutkielmassa esitettävien fysiikkamoottoreiden dokumentaatio. Tutkielmassa esiteltävät algoritmit valittiin tunnetuissa fysiikkamoottoreissa esiintyvien algoritmien perusteella. Algoritmien lisäksi tutkielmassa esitellään törmäystarkastelussa esiintyviä termejä, sekä algoritmien matemaattisia taustoja. Tutkielmassa havaittiin törmäystarkastelun olevan melko pitkään tutkittu tietokonegrafiikan osa-alue, jossa tiettyjen algoritmien asema on vakiintunut. Tämän lisäksi tutkielmassa havaittiin törmäystarkastelun olevan edelleen tutkimuksen kohteena.

Avainsanat: Törmäystarkastelu, GJK, SAT, AABB, PhysX, GJK, BVH, Fysiikan mallinnus, Kasipuu

Abstract: The aim of the thesis is to introduce methods and algorithms used for collision detection in real-time applications. The research method used in the thesis is a literature review. Beside literature and articles in the field, the documentation of physics engines presented in the thesis are used as the source. The algorithms presented in the thesis were selected based on the algorithms in well-known physics engines. In addition to algorithms, the thesis aims to present the terms used in collision detecion and the mathematical backgrounds of presented algorithms. In thesis was observed, that collision detection is a relatively long studied field of computer graphics and the usage of specifi algorithms have become established. In addition to this, was observed that

collision detection continues to be the subject of the study.

Keywords: Collision detection, GJK, SAT, AABB, PhysX, GJK, BVH, Physics modelling, Octree

Kuviot

Taulukot

Sisältö

1 Johdanto

Törmäystarkastelu on tärkeä osa monia erilaisia sovelluksia, mukaan lukien videopelit, fysiikkaan perustuvat simulaatiot, animaatiot ja robotiikka. Reaalimaailmassa kiinteät kappaleet eivät uppoudu toisiinsa tai läpäise toisiansa, mutta virtuaalimaailmassa kappaleet koostuvat pisteistä, eivätkä täten omaa reaalimaailman kiinteiden kappaleiden ominaisuuksia. Törmäystarkastelun tehtävänä on saada virtuaalimaailman pisteistä koostuvat kappaleet näyttämään kiinteiltä, määrittelemällä kappaleiden väliset törmäykset kappaleiden toisiinsa vajoamisen estämiseksi. Tähän yleensä ei riitä pelkkä tieto siitä, että törmäävätkö kaksi kappaletta, vaan usein törmäystarkastelun tulee lisäksi tuottaa tarkempaa tietoa törmäyksestä.

Naiivi ratkaisu törmäysten löytämiseen olisi testata kaikki kappaleet ja kappaleiden välillä kaikki pisteet toisiaan vasten. Tällainen brute-force menetelmään perustuva törmäystarkastelu ei useinkaan ole riittävän nopea suoritettavaksi reaaliajassa sovelluksissa, joissa tarkasteltavia törmääjiä on useita tai törmääjät muodostuvat useista tuhansista pisteistä. Jotta saadaan törmäykset ratkottua reaaliajassa, on törmäystarkastelu pilkottu eri vaiheisiin. Yleisimmin törmäystarkastelu on jaettu kahteen tai kolmeen vaiheeseen. Tutkielmassa lähtökohdaksi ollaan valittu kolmessa vaiheessa tapahtuva törmäystarkastelu, mikä on käytössä muun muassa PhysX:ssä (“PhysX: User’s Guide” 2018) ja Havokissa (Gregory 2009, s. 677).

Tutkielmaan ollaan valittu esiteltäviksi algoritmeja, joita käytetään törmäystarkastelun sisältävissä ohjelmistonkehityspakettia, joita puolestaan reaaliaikaisissa sovelluksissa hyödynnetään. Tutkielmassa aihetta on rajattu siten, että tarkastellaan CPU:lla tapahtuvaa suljettujen kappaleiden törmäystarkastelua kolmiulotteisessa avaruudessa. Tutkielman lähteinä ollaan käytetty alan kirjallisuutta ja artikkeleita, sekä tutkielmassa esiteltävien ohjelmistonkehityspakettien dokumentaatioita ja Game Development Conferencen (GDC) esityksiä.

Ensimmäisessä luvussa 2 käydään läpi törmäystarkastelun taustaa ja lähteissä esiintyviä termejä, sekä esitellään tarkemmin tutkielmassa noudatettu törmäystarkastelun liukuhihnamalli. Luvuissa 3, 4, 5 vastaavasti käydään läpi törmäystarkastelun liukuhihnamallin kolme vaihetta yksityiskohtaisemmin, esitellen kussakin vaiheessa yleisesti käytössä olevia algoritmeja. Lopuksi luvussa 6

esitellään johtopäätökset, sekä pohdintaa siitä, mitä rajattiin tutkielman ulkopuolelle.

2 Tausta

Koska törmäystarkastelu on aihepiirinä melko laaja, liittyy siihen monia termejä. Tutkielmassa tullaan tarvitsemaan termejä algebran, geometrian, tietokonegrafiikan ja törmäystarkastelun alueilta. Useille termeille ei ole olemassa vakiintuneita termejä suomen kielessä, joille tutkielmassa pyritään osakseen antamaan suoria käännöksiä ja toisinaan annetaan kuvaannollisempia tarkan käännöksen sijasta. Erityisesti tutkielman aiheessa ollaan suoran käännöksen sijaan päädytty käyttämään aavistuksen vapaampaa käännöstä. Englannin kielessä on aiheelle on vakiintunut termi collision detection, jossa sitä lähteestä riippuen käytetään tarkoittamaan törmäysten havaitsemista tai sitä, mikä tässä tutkielmassa tunnetaan törmäystarkasteluna. Erona näiden kahden termillä välillä tutkielmassa pidetään sitä, että törmäysten havaitseminen palauttaa tiedon siitä, mitkä kappaleet törmäsivät ja törmäystarkastelu puolestaan tuottaa tämän lisäksi tarkempaa tietoa tömäyksistä. Tarkemmin törmäystarkastelun jakautumista törmäysten havaitsemiseen ja sitä seuraavaan vaiheeseen eli kontaktimoniston muodostamiseen (engl. contact manifold generation) selvitetään alaluvussa 2.4.

Lukijan on hyvä huomata, että koska törmäystarkastelun juuret ovat matematiikassa, käytetään tutkielmassa termejä sekä matematiikan, että tietokonegrafiikan puolelta riippuen kontekstista, tarkoittaen kuitenkin oleellisesti samaa. Näitä ovat esimerkiksi leikkaus – törmääminen, kärki – verteksi, affiini muunnos – muunnos, avaruus – ympäristö ja avaruuskappale – kappale. Matematiikassa leikkaukset määritellään pistejoukoille ilmaisemaan pistejoukkojen yhteisiä alkioita ja törmäystarkastelussa käytetään sanaa törmäys, kun kaksi kappaletta ovat penetroituina toisiinsa eli toisin sanoen kahden avaruuskappaleen sisäänsä sulkevien äärellisten pistejoukkojen leikkaus on epätyhjä joukko. Kappaleella puolestaan tarkoitetaan geometrisia muotoja joiden välillä törmäyksiä tarkastellaan. Näitä ovat alaluvussa 2.6 esiteltävät rajaavat tilavuudet, sekä luvussa 2.5 esitettävät törmäyskappaleet. Avaruudella tutkielmassa tarkoitetaan euklidista avaruutta ja termiä ympäristö käytetään tarkoittamaan virtuaalimaailman avaruutta. Edellä mainittujen lisäksi, koska on usein tarpeellista puhua monitahokkaan kärjistä, särmistä ja tahkoista yleisesti, käytetään termiä monitahokkaan ominaisuus (engl. feature) merkitsemään edellä mainittua ‘ominaisuuksien’ joukkoa.

Tutkielmaa varten valittiin kolme fysiikan mallinnus ohjelmistonkehityspakettia: PhysX, Bullet Physics ja Open Dynamics Engine (ODE), joihin on integroituna törmäystarkastelu, sekä törmäystarkastelukirjasto SOLID, joista tutkielmassa käytetään termiä tarkasteltavat kirjastot. Tarkasteltaviin kirjastoihin tehdyt viittaukset perustuvat niiden dokumentaatioon ja lähdekoodiin tutkielman kirjoittamisen hetkellä uusimpiin versioihin. Nämä ovat PhysX 3.4.0, Bullet Physics 2.87, ODE 0.15 ja SOLID 3.5. Tarkemmin tarkasteltavat kirjastot tullaan esittelemään alaluvussa 2.4.

Yksi merkittävä termi, jolle ei tutkielmassa kuitenkaan varattu omaa alalukua on kehyskoherenssi (engl. frame coherence) (G. v. d. Bergen 2004a, s. 54), joka myös tunnetaan myös nimellä ajallinen koherenssi (engl. temporal coherency) (Ericson 2004, s. 18). Kehyskoherenssi tarkoittaa kappaleiden muuntumattomuutta yksittäisten kehysten välillä. Kehyskoherenssi on usein korkea sovelluksissa, joissa kehykseen käytetty aika on lyhyt tai useampi kappale on levossa tai kappaleiden liikkeet ovat hitaita. Korkea kehyskoherenssi mahdollistaa edellisen kehyksen laskennan tuloksien uudelleen käytön nykyisellä kehyksellä. Muita törmäystarkastelun kannalta tärkeimpiä termejä ja matemaattisia taustoja esitellään seuraavissa alaluvuissa. Matemaattisia taustoja tullaan täydentämään tutkielman myöhäisemmässä vaiheessa luvussa 5.1 niiltä osin kuin niitä tullaan tarvitsemaan esiteltävien menetelmien ymmärtämiseksi, mutta ensimmäisten lukujen kannalta vaadittu matemaattinen tausta on koottu tämän luvun alalukuihin.

2.1 Reaaliaikaisuus

Koska tutkielmassa keskitytään reaaliajassa tapahtuvaan törmäystarkasteluun, lienee tarvetta avata reaaliaikaisuuden määritelmää. Käsitteenä reaaliaikaisuus on usein melko löyhä; esimerkiksi elokuvissa yleisesti käytetty kuvataajuus 24 kehystä sekunnissa saattaa nopeaa reagointia vaativassa videopelissä tuntua riittämättömältä. Kuitenkin molempien ajatellaan olevan reaaliaikaisia. Yleistäen voidaan sanoa, että reaaliaikaisuuden määritelmä on riippuvainen tarkasteltavasta sovelluksesta. Jotta esimerkiksi interaktiivinen sovellus voi tuottaa käyttäjälleen todellisen interaktiivisuuden tunteen, vaatii se usein korkeampaa kuvataajuutta kuin esimerkiksi ei-interaktiivinen animaatio. Eräänlainen yleisempi määritelmä reaaliaikaisuudelle annettaan kirjassa (Akenine-Möller, Haines ja Hoffman 2008, s. 1). Määritelmän mukaan 15 kehystä sekunnissa alkaa jo

tuntumaan reaaliaikaiselta ja kuvataajuuden ylittäessä 72 kehystä sekunnissa, ihmissilmä ei juurikaan kykene erottamaan kasvua kuvataajuudessa (Akenine-Möller, Haines ja Hoffman 2008, s. 1).

Tulee kuitenkin huomata, että edellä mainitut kuvataajuudet ovat koko kehyksen tuottamiseen käytettyjä arvoja. Sovellus kuitenkaan harvemmin koostuu pelkästä törmäystarkastelusta, jolloin täytyy joka kehyksellä suorittaa myös muuta laskentaa. Videopeleissä esimerkiksi, jossa on törmäystarkastelun lisäksi monia muita systeemejä päivitettävänä, törmäystarkastelu voi viedä kehykseen käytettävästä ajasta 10 - 30 % (Ericson 2004, s. 17). Esimerkiksi videopelissä, jonka kuvataajuus on 60 kehystä sekunnissa, jolloin yhden kehyksen tuottamiseen käytettävä aika on 16,7 millisekuntia, jää törmäystarkastelulle aikaa noin 2 - 5 millisekuntia (Ericson 2004, s. 17). Tämän tarkemmin reaaliaikaisuuden määritelmää pohtimatta ovat tässä tutkielmassa esiteltyjä algoritmeja ja tekniikoita käytetty interaktiivisissa sovelluksissa, joissa vaatimuksena voidaan yleisesti pitää vähintään 30 kehystä sekunnissa (G. v. d. Bergen 2004a, s. 53; Weller 2013, s. 20). Tutkielman ulkopuolella jäävät metodit, joilla törmäystarkastelu pyritään toteuttamaan haptista palautetta vaativissa sovelluksissa, joissa vaadittava virkistystaajuus realistisen kokemuksen tuottamiseksi on jopa 1000 Hz (Weller 2013, s. 24).

2.2 Konveksisuus

“Yksi tärkeimmistä konsepteista törmäystarkastelussa on ero konveksien ja konkaavien kappaleiden välillä.“ (Gregory 2009, s. 660, suomennos minun). Törmäystarkastelun näkökulmasta konveksisuudelle voidaan määritellä kaksi tärkeää ominaisuutta. Ensimmäinen ominaisuus seuraa geometriasta, jossa konveksi joukko määritellään joukoksi, jonka mistä tahansa kahdesta pisteestä yhdistetty jana ei sisällä pisteitä joukon ulkopuolelta (Schneider 1993, s. 1). Toinen ominaisuus seuraa konveksin funktion ominaisuudesta, jolla lokaali ääriarvo on myös globaali ääriarvo (Schneider 1993, s. 22). Konvekseille avararuuskappaleille pätee edellä mainitut ominaisuudet siten, että mistä tahansa konveksin avaruuskappaleen sisäänsä sulkevan äärellisen pistejoukon kahdesta pisteestä muodostettu jana ei sisällä pisteitä avaruuskappaleen ulkopuolelta, sekä tietyssä suunnassa löydetty lokaali ääriarvo on myös globaali ääriarvo. Konkaaveilla avaruuskappaleilla eivät edellä mainitut ominaisuudet ole päde.

Tämä konveksien ja konkaavien kappaleiden eroavaisuus näyttelee suurta roolia törmäystarkastelussa, sillä edellä mainitut ominaisuudet mahdollistavat yksinkertaisempien ja laskennallisesti tehokkaampien algoritmien käytön (Gregory 2009, s. 661). Ensimmäisenä mainittu ominaisuus muun muassa mahdollistaa törmäystarkastelun kannalta tärkeän teoreeman, erottavan akselin teoreeman, kuten tullaan näkemään luvussa 2.7. Jälkimmäistä ominaisuutta puolestaan hyödynnetään luvussa 5.1 esiteltävässä kantajafunktiossa, kun sitä sovelletaan konvekseille verhoille. Näiden ominaisuuksien tuomasta hyödystä pyritään konkaavit suljetut kappaleet jakamaan konvekseihin osiin ennen ohjelman suoritusta tai ne voidaan sulkea konveksin verhon (engl. convex hull) sisälle. Tarkka matemaattinen määritelmä konveksille verholle on, että pistejoukon S konveksi verho on kaikkien niiden konveksien pistejoukkojen leikkaus, jotka sisältävät pistejoukon S (Rockafellar 1997). Toisin sanoen avaruuskappaleen konveksi verho on pienin konveksi tilavuus, joka sulkee kappaleen kokonaisuudessaan sisäänsä. Yksinkertaisena, hyvänä mielikuvaharjoitteluna riittää tässä vaiheessa ajatella konveksia verhoa peitteenä, joka saadaan kun kappaleen ympärille asetetaan tiukka kuminauha.

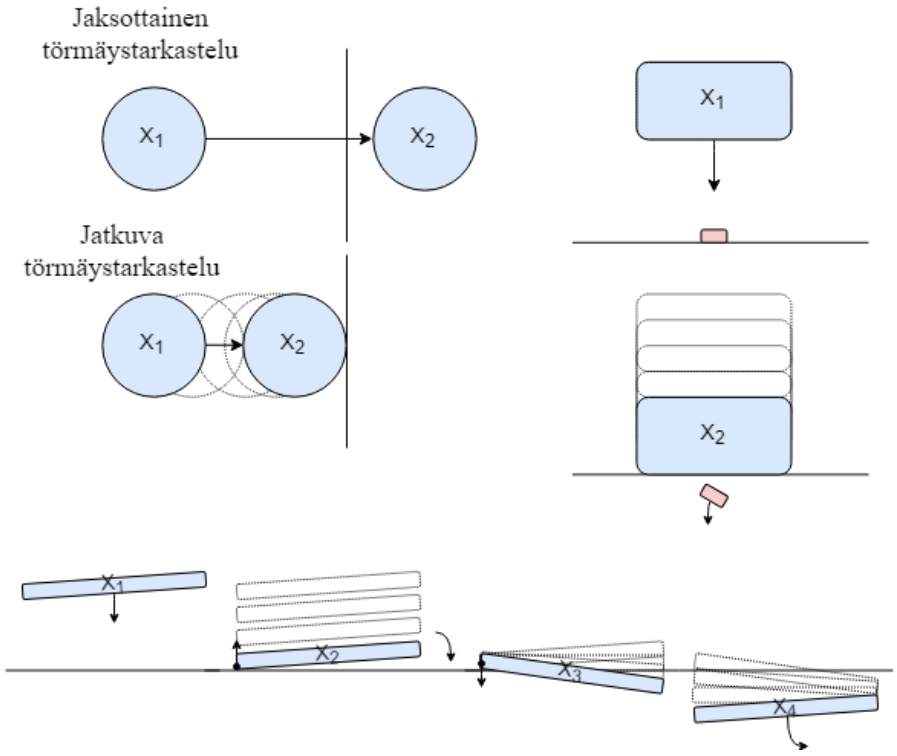
Rajallisesta määrästä kärkiä koostuvaa konveksia verhoa kutsutaan polytoopiksi (engl. polytope) (Schneider 1993, s. 3). Polytoopit ovat geometristen muotojen ‘luokka’, johon luetaan kuuluvaksi konveksit monikulmiot, konveksit monitahokkaat, sekä simpleksit (G. v. d. Bergen 2004a, s. 24). Simpleksit tullaan tarkemmin käsittelemään luvussa 5.1. Yleisemmin rajallisesta määrästä koostuvista kappaleista käytetään nimitystä eksplisiittisesti määriteltävät kappaleet. Eksplisiittisesti määriteltävien kappaleiden vastakohta ovat implisiittisesti määriteltävät kappaleet eli kaarevia pintoja sisältävät kappaleet. Yksinkertainen esimerkki implisiittisesti määriteltävästä kappaleesta on pallo, sillä sen konveksin verhon ei katsota muodostuvan yksittäisistä kärjistä, vaan sen konveksin verhon pisteet ilmaistaan keskipisteen ja säteen avulla. Yleisesti käytetty algoritmi konveksin verhon muodostamiseksi on Barber, Dobkin ja Huhdanpaa (1996) esittelemä algoritmi Quickhull. Algoritmin tarkempi kuvaaminen jätetään tutkielman ulkopuolelle, mutta alkuperäisen artikkelin lisäksi sen hyvin esitteli Gregorius (2014) GDC:ssä. Konkaavien kappaleiden jakautumista konkaavien kappaleiden muodostamiin hierarkkisiin rakenteisiin käsitellään sen sijaan tarkemmin luvussa 4. Tämä on tärkeää sillä luvussa 5 esiteltävät algoritmit, kuten yleisestikin reaaliaikaisessa törmäystarkastelussa käytettävät algoritmit, ovat määritelty ainoastaan

konvekseille kappaleille.

2.3 Jaksottainen ja jatkuva törmäysten havaitseminen

Reaalimaailmassa aika voidaan usein ajatella tapahtuvan jatkuvana. Virtuaalimaailmassa näin ei kuitenkaan ole, vaan aika on jaksottaista, virkistystaajuuden mukaan tapahtuvaa. Tästä syystä myös törmäysten havaitsemisen on loogista tapahtua jaksottaisena eli jokaisella kehyksellä erillisenä. Tällöin kuitenkin ongelmana on, että kehysten välissä tapahtuvia törmäyksiä ei kyetä havaitsemaan. Pahimmillaan tämä johtaa kappaleen läpäisemiseen toisesta kappaleesta eli tunneloitumiseen (engl. tunneling). Gustafsson (2010) esittelee artikkelissaan kolme yleistä fysiikan mallinnuksessa ilmenevää, jaksottaisesta törmäysten havaitsemisesta johtuvaa tunneloitumisen tapausta. Nämä ovat luoti-läpi-paperin ongelma (engl. bullet-through-paper problem), niin kutsuttu puristuminen (engl. sandwitch case), sekä usein pitkällä ja kapealla kappaleella tapahtuva rotaation aiheuttama tunneloituminen. Luoti-läpi-paperin ongelmassa nopeasti liikkuva kappale sijaitsee peräkkäisillä kehyksillä tason eri puolilla, jolloin varsinainen törmäys tapahtuu kehyksien välillä, eikä sitä kyetä havaitsemaan. Luoti-läpi-paperin ongelma on esitetty kuviossa 1 ylhäällä vasemmalla. Puristuminen puolestaan tapahtuu usein kappaleilla, joiden välillä on suuri koko ero. Puristumisessa törmäykset kyetään havaitsemaan, sekä kontaktimonistot muodostamaan, mutta liian myöhään eli syvästi penetroituneilla kappaleilla, jolloin fysiikan mallinnus ei järkevällä tavalla kykene törmäyksiä ratkaisemaan. Tämä on esitetty kuviossa 1 ylhäällä oikealla. Kolmas ongelma, jossa pitkä ja kapea kappale läpäisee tason on yhdistelmä kontaktimoniston muodostamisesta ja seuraavalla kehyksellä tapahtuvasta törmäyksen havaitsemattomuudesta. Tämä johtaa siitä, että kontaktipisteen löytäminen saattaa aiheuttaa voimakaan rotaation kappaleeseen, jolloin kappale tunneloituu vähitellen tason läpi. Tarkemmin tämä on kuvattu kuviossa 1 alhaalla.

Edellä mainittujen ongelmien välttämiseksi voidaan törmäysten havaitseminen suorittaa jatkuvana. Myös termiä dynaaminen törmäysten havaitseminen (engl. dynamic collision detection) (Ericson 2004, s. 214) käytetään tarkoittaen samaa. Edellä esitetyn luoti-läpi-paperin ongelman estäminen käyttämällä jatkuvaa tömäysten havaitsemista on esitetty kuviossa 1 keskellä vasemmalla. Jatkuvan törmäysten havaitsemisen tehtävänä on löytää ensimmäisen törmäyksen ajan-



Kuvio 1. Jaksottaisesta törmäystarkastelusta johtuvat artefaktat esiteltynä kehyksen n ja kappaleen sijainnin X_n mukaisesti.

hetki (engl. time of impact, TOI). TOI:n löytämiseen on olemassa monta metodia, joista yksi paljon käytetty metodi tullaan esittelemään tarkemmin luvussa 5.2.3. Yleisesti voidaan näiden metodien pyrkivän jakamaan kahden kehyksen välinen aikajakso pienempiin aikajaksoihin, mikä johtaa siihen että jatkuva törmäysten havaitseminen on laskennallisesti raskaampaa verrattuna jaksottaiseen törmäysten havaitsemiseen, joten sitä pyritään soveltamaan ainoastaan riskiryhmään kuuluviin kappaleisiin. Riskiryhmään voidaan karkeasti luetella kuuluvaksi muun muassa nopeasti liikkuvat kappaleet, sekä pitkät ja kapeat dynaamiset kappaleet (Catto 2013).

2.4 Törmäystarkastelun arkkitehtuuri

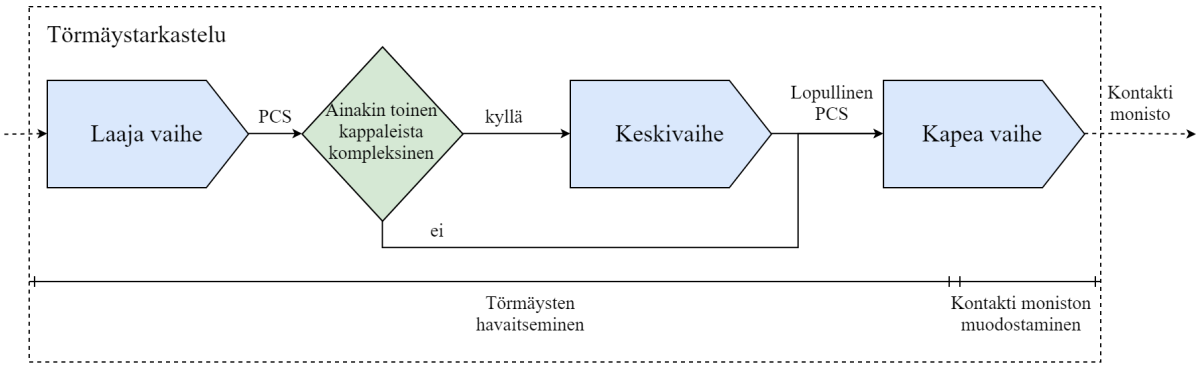
Arkkitehtuurisesti törmäystarkastelu on alemman tason järjestelmä, jonka tarkoituksena on havaita kappaleiden väliset törmäykset, sekä muodostaa kontaktimonistot törmääville kappale pareille. Vaikka törmäystarkastelu voi yleensä toimia itsenäisenä järjestelmänä, on se usein tiukasti

sidottuna fysiikan mallinnukseen (Gregory 2009, s. 46). Fysiikan mallinnuksen yhtenä tehtävänä onkin törmäystarkastelun tuottamien kontaktimonistojen perusteella reagoida törmäyksiin. Törmäystarkastelun ja fysiikan mallinnuksen sidonnaisuudesta johtuu, että virheellinen törmäystarkastelu on usein syynä fysiikkamallinnuksessa ilmeneviin artefakteihin, kuten luvussa 2.3 esiteltiin. Vahvasta sidonnaisuudesta johtuu myös se, että usein törmäystarkastelu on integroituna osaksi fysiikkamoottoria. Muutamia tunnetuimpia törmäystarkastelun sisältäviä fysiikkamoottoreita ovat:

- Havok
- PhysX
- Bullet Physics
- Open Dynamics Engine (ODE)

Havok on eräänlainen kultainen standardi fysiikkamallinnuksessa. Kirjoittamisen hetkellä Microsoftin omistamaa Havokia on käytetty yli 600 videopelissä, sekä muutamissa elokuvissa erikoistehosteiden luomiseen ("Havok: About Havok" 2018). Muihin edellä mainittuihin fysiikkamoottoreihin verrattuna on Havok maksullinen, eikä sen lähdekoodi ja dokumentaatio ole avoimia, jonka takia ei Havokia tulla tässä tutkielmassa sen tarkemmin käsittelemään. Poikkeuksena muutamat Gregory (2009) kirjan kautta tehdyt viittaukset. PhysX puolestaan on nykyisin Nvidian omistama reealiaikainen fysiikkamoottori, joka Havokin ohella on käytössä useissa tunnetuissa kaupallisissa peleissä ("PhysX: Game Titles" 2018). Kirjoittamisen hetkellä fysiikkamoottoria on käytetty yli 500 videopelissä ("Nvidia: PhysX" 2018). Huomattava ero Havokiin on, että PhysX on avointa lähdekoodia ja se on ilmaiseksi käytettäväissä sekä ei-kaupallisten, että kaupallisten sovellusten kehittämiseen ("Nvidia: PhysX" 2018). PhysX:n ohella tunnettuja avoimen lähdekoodin fysiikkamootteita ovat alun perin Erwin Coumansin kehittämä Bullet Physics ja Russell Smithin kehittämä ODE. Näistä Bullet Physics oli alun perin Erwin Coumansin kehittämä ja ODE puolestaan Russell Smithin. Molempia fysiikkamoottoreita on käytetty erinäisissä sovelluksissa kaupallisista videopeleistä robotiikkaa ("Bullet Physics: Homepage" 2018, "ODE: wiki" 2018).

Kuten luvun 2 alussa todettiin, voidaan törmäystarkastelu jakaa törmäysten havaitsemiseen ja

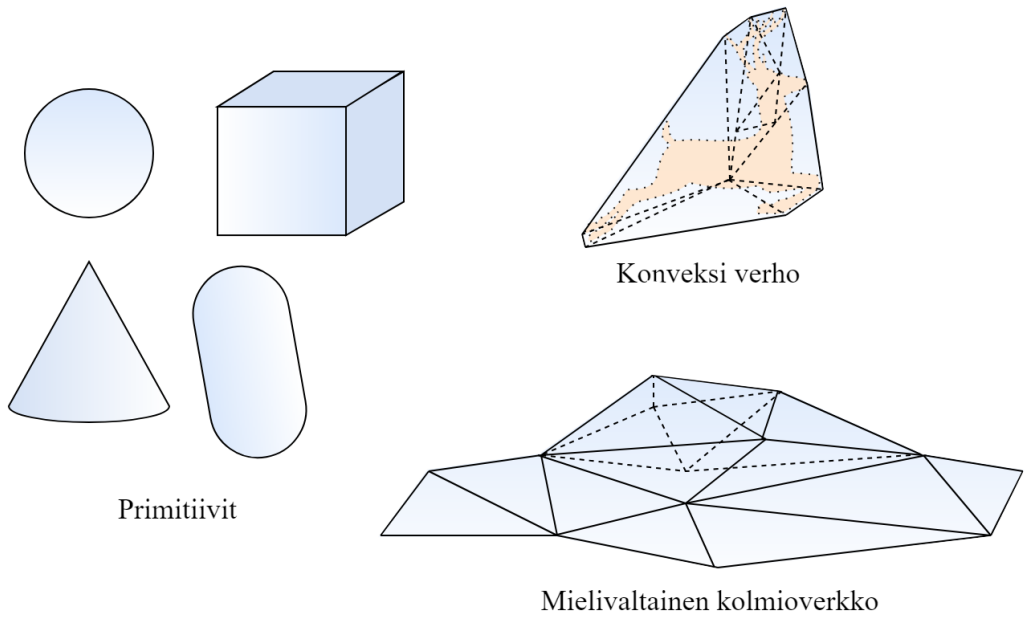


Kuvio 2. Törmäystarkastelun liukuhihnamalli.

kontaktimoniston muodostamiseen. Tämän lisäksi törmäysten havaitseminen jaetaan usein useampaan eri vaiheeseen, jolloin törmäystarkastelun arkkitehtuuria voidaan kokonaisuutena kuvata yksinkertaisesti liukuhihnamallina. Törmäystarkastelun jakaminen itsenäisiin vaiheisiin ei kuitenkaan ole täysin yksiselitteistä. Riippuen lähteestä, jaetaan törmäystarkastelu joko kahteen: laajaan ja kapeaan vaiheeseen (Migdalskiy 2010, s. 63; Ericson 2004, s. 14) tai kolmeen vaiheeseen (Gregory 2009, s. 677; Gustafsson 2010, s. 36), jolloin laajan ja kapean vaiheen välillä suoritetaan keskivaihe. Keskivaihe koostuu monimutkaisemmille kappaleille muodostettujen rajaavien tilavuuksien hierarkioiden törmäysten havaitsemisesta, joka voidaan katsotaan omaksi vaiheekseen (Gustafsson 2010, s. 36, Gregory 2009, s. 677, “PhysX: User’s Guide” 2018) tai se voidaan katsoa sisältyväksi joko laajaan tai kapeaan vaiheeseen (Ericson 2004, s. 284; Weller 2013, s. 14). Tässä tutkielmassa rajaavien tilavuuksien hierarkiat sijoitetaan omaksi vaiheekseen, jolloin liukuhihnamalli koostuu laajasta vaiheesta, keskivaiheesta ja kapeasta vaiheesta. Näistä vaiheista laajassa vaiheessa ja keskivaiheessa muodostetaan potentiaalisesti törmäävien parien lista (engl. potentially colliding set, PCS) ja viimeisessä eli kapeassa vaiheessa suoritetaan tarkempi törmäysten havaitseminen pareittain ja törmääville pareille kontaktimoniston muodostaminen. Tutkielmassa noudatettava liukuhihnamalli esitellään kuviossa 2. Kuviossa kompleksisilla kappaleilla tarkoitetaan, niitä kappaleita jotka esitetään rajaavien tilavuuksien hierarkiana tai koostuvat useammasta törmäyskappaleesta.

2.5 Törmäyskappaleet

Koska useimmissa sovelluksissa renderöitävien kappaleiden muodot ovat liian monimutkaisia, jotta niiden tarkka törmääminen kyettäisiin reaaliajassa ratkaisemaan, käytetään törmäystarkastelussa yleensä yksinkertaistettuja muotoja varsinaisen kappaleen esittämiseen. Tässä tutkielmassa näitä muotoja tullaan kutsumaan törmäyskappaleiksi. Törmäyskappaleita käytetään törmäystarkastelussa lopullisen törmäysten havaitsemiseen ja kontaktimoniston muodostamiseen. Tutkielman käytetyssä liukuhihnamallissa niiden käyttö rajoittuu siis luvussa 5 esiteltävään kapeaan vaiheeseen. Yleensä törmäyskappaleiksi valitaan geometrisia muotoja, jotka riittävän hyvin muistuttavat niitä edustavia renderöitäviä kappaleita, jolloin niillä suoritettava törmäystarkastelu antaa riittävän hyvän approksimaation realistisesta törmäyksestä. Sanavalinnalla riittävän hyvä halutaan painottaa sitä, että reaaliaikaisissa sovelluksissa joudutaan usein tekemään kompromisseja tarkkuuden ja suoritettavan laskennan määrän välillä. Tästä syystä esimerkiksi videopeleissä ohjattavan pelihahmon törmäyskappaleena käytetään usein kapselia, jolloin kapselin ulkopuolelle jäävät raajat saattavat penetroitua seiniin tai muihin virtuaalimaailman kappaleisiin (McShaffry 2014, s. 584). Tämä on melko kaukana reaalimaailman tapahtumista, mutta monessa sovelluksessa tämänkaltainen approksimaatio on riittävän hyvä.



Kuvio 3. Törmäyskappaleita.

Törmäyskappaleista voidaan erikseen erotella primitiivit, jotka ovat yksinkertaisimpia muotoja varsinaisen kappaleen esittämiseen (“PhysX: User’s Guide” 2018, “Bullet Physics: Documentation” 2018). Näihin lukeutuvat muun muassa pallot, suorakulmaiset särmiöt, kapselit, kolmiot ja tasot. Yleensä primitiivien väliseen törmäysten havaitsemiseen voidaan käyttää laskennallisesti kevyempiä yksilöityjä törmäysten havaitsemisen menetelmiä. Tämä ilmenee myös taulukossa 5. Primitiivien lisäksi törmäystarkastelussa käytettyjä muotoja ovat muun muassa luvussa 2.2 esitetyt konveksit verhot, korkeuskentät, useammasta törmäysprimitiivistä koostuvat yhdistelmät ja mielivaltaiset kolmioverkot, jotka myös polykeittoina tunnetaan (engl. poly soup). Mielivaltaisilla kolmioverkoilla tarkoitetaan niitä kolmioista koostuvia kappaleita, joilla ei voida katsoa olevan sisä- tai ulkopuolta. Mielivaltaisilla kolmioverkoilla törmäysten havaitseminen tapahtuu luvussa 4 esitettyjä rajaavien tilavuuksien hierarkioita käyttäen. Taulukkoon 1 on listattu tarkasteltavissa kirjastoissa olevat mahdolliset törmäyskappaleet. Taulukosta on jätetty pois säteet, kolmiot ja

tasot, joiden osalta törmäyksiä on mahdollista tarkastella kaikissa kirjastoissa. Kuviossa 3 esitellään yleisempiä törmäyskappaleita. Kuten kuviossa esitetyistä kappaleista voi huomata, voivat törmäyskappaleet olla implisiittisesti tai eksplisiittisesti määriteltyjä.

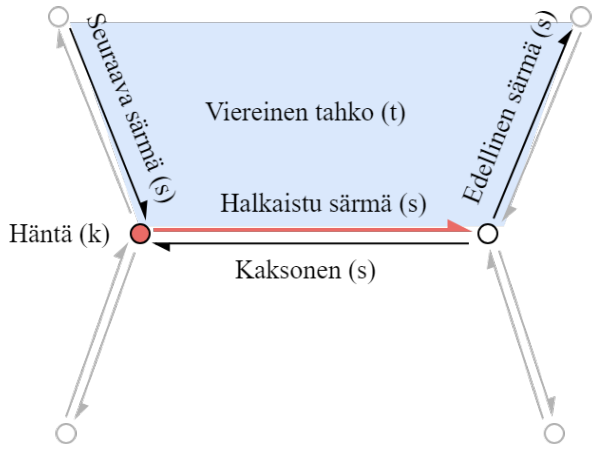
Taulukko 1. Törmäyskappaleet eri kirjastoissa

	PhysX 3.4	Bullet Physics 2.87	ODE 0.15	SOLID 3.5
Laatikko	X	X	X	X
Kartio		X		X
Lieriö		X	X	X
Pallo	X	X	X	X
Kapseli	X	X	X	
Kolmio verkko	X	X	X	X
Koveksi verho	X	X	X	X
Korkeuskartta	X	X	X	

Taulukossa esitettyjen törmäyskappaleiden lisäksi on usein mahdollista käyttää myös niiden yhdistelmiä.

Muodon lisäksi törmäyskappaleet eroavat renderöitävistä kappaleista myös rakenteen osalta. Esimerkiksi suorakulmainen särmiö saadaan keskipisteen, kolmen suuntavektorin ja kolmen särmän puolipituuden avulla määriteltyä. Kapseli puolestaan voidaan tallentaa muistiin janan ja säteen avulla. Merkittävästi rakenteeltaan eroavat myös monitahokkaat, joihin myös konveksi verho lukeutuu. Nämä usein pyritään tallentamaan sellaiseen rakenteeseen, joissa sen ominaisuuksia eli kärkiä, särmiä ja tahkoja pitkin voidaan helposti matkustaa haluttuun suuntaan. Tämä vaatii sen, että jokaisella ominaisuudella on tiedossa viereiset ominaisuudet. Eräs tällainen rakenne on artikkelissa (Muller ja Preparata 1978) esitelty DCEL (double connected edge list), sekä halkaistu särmä (engl. half edge) -datarakenne, jota useassa lähteessä kutsutaan DCEL:ksi (De Berg ym. 2000, s. 29-33). Huomautettakoon kuitenkin, että alkuperäisessä toteutuksessaan DCEL (Muller ja Preparata 1978) rakenteessa särmät kuvataan kaksisuuntaisina, toisin kuin halkaistu särmä -datarakenteessa, jossa jokainen särmä kuvataan kahdella eri suuntiin suunnatuilla halkaistulla särmillä. Näistä yleisemmin käytetty eli halkaistu särmä datarakenne esitellään ku-

viossa 4.



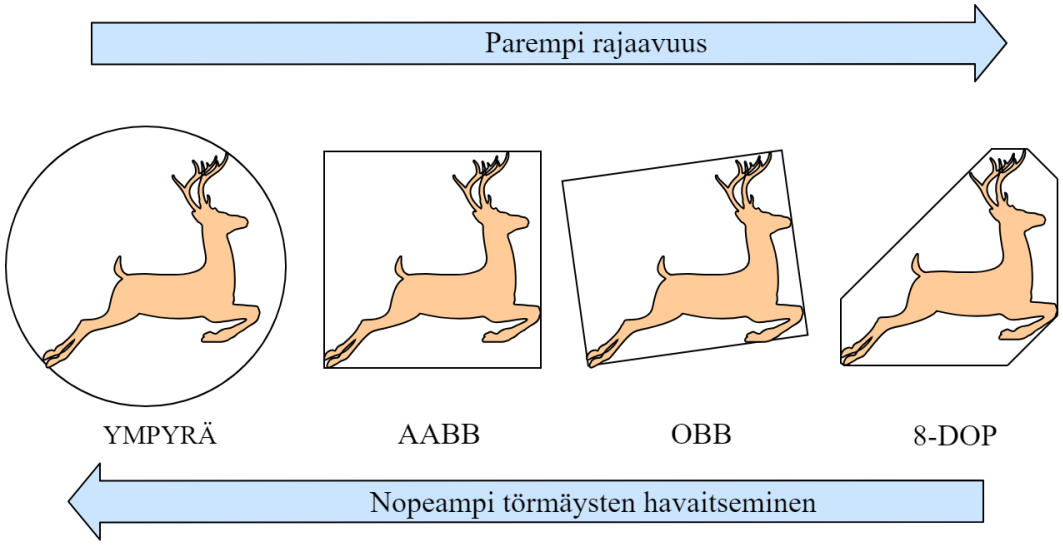
Kuvio 4. Halkaistu särmä datarakenne.

2.6 Rajaavat tilavuudet

Tutkielmassa aikaisemminkin mainittu rajaava tilavuus (engl. bounding volume) on suljettu tilavuus, joka sulkee sisäänsä törmäyskappaleen kokonaisuudessaan. Sen tarkoituksena on toimia yksinkertaisena välineenä potentiaalisesti törmäävien törmäyskappaleiden löytämiseen varhaisessa vaiheessa ja poissulkemaan ne kappaleet tarkemmalta tarkastelulta, joiden ei ole mahdollista törmätä toisiinsa. Tämä perustuu olettamukseen, että jos eri kappaleita kokonaisuudessaan ympäröivät tilavuudet eivät törmää toisiansa, eivät niiden sisältämien kappaleidenkaan ole mahdollista törmätä toisiansa. Kirjassa (Ericson 2004, s. 76) rajaaville tilavuuksille annetaan seuraavat halutut ominaisuudet:

- yksinkertainen ja nopea leikkaavuuden määrittely
- rajaa kappaleen tiukasti
- yksinkertainen ja nopea muodostaa
- yksinkertainen päivittää kappaleen kiertyessä
- käyttää vähän muistia

Valinta eri rajaavien tilavuuksien välillä on aina kompromissi rajaavuuden, muistin käytön ja laskennallisen keveyden välillä. Yleensä yksinkertainen rajaava tilavus ei rajaa kappaletta tiukasti,



Kuvio 5. Rajaavia tilavuuksia.

kun taas hyvin tiukasti kappaleen rajaava tilavuus taas vaatii usein enemmäin tilaa muistista ja laskennallisesti raskaamman törmäyksen määrittelyn. Kuviossa 5 on esitetty yleisimmät rajaavat tilavuudet. Tutkielmassa rajaavista tilavuuksista käsitellään akselin suuntaista rajaavaa laatikkoa (engl. axis aligned bounding box, AABB), joka tulee vastaan laajassa vaiheessa luvussa 3 ja keskivaiheessa luvussa 4. Muiden rajaavien tilavuuksien osalta, aiheesta kiinnostuneen lukijan kannattaa lukea lisää aiheesta kirjasta (Ericson 2004, s. 75-123). Kaikkien rajaavien tilavuuksien osalta on kuitenkin hyvä huomata, että kuten luvussa 2.5 esitetyt törmäyskappaleet, niin myös rajaavat tilavuudet muodostetaan ennen ohjelman ajoa, joten rajaavan tilavuuden muodostamiseen kuluvalla ajalla ei ole vaikutusta varsinaiseen ohjelman suoritukseen.

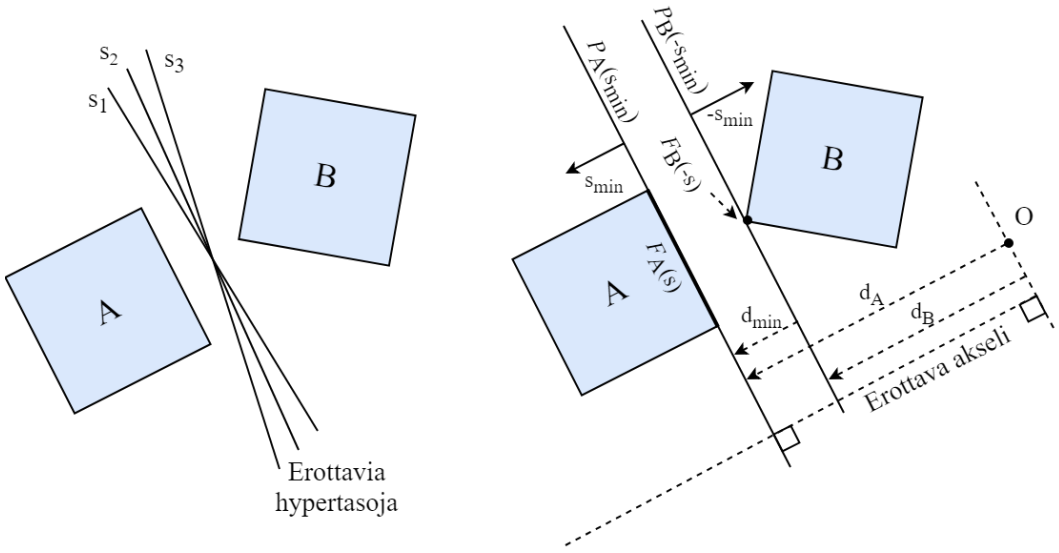
Akselin suuntainen rajaava laatikko eli lyhyemmin AABB on nimensä mukaisesti suorakulmainen särmiö, jonka särmät ovat valitun koordinaatiston akseleiden suuntaiset. Käytännössä tämä tarkoittaa AABB:n särmien suuntaamista maailman koordinaatiston akseleiden suuntaisesti. Kirjassaan G. v. d. Bergen (2004a, s. 72) mainitsee AABB:n olevan käytetyin rajaava tilavuus. Tämä selittyy niiden triviaalilla muodostamisella, vähäisellä muistin tarpeella ja yksinkertaisella törmäysten havaitsemisella. AABB:n muodostaminen tapahtuu läpikäymällä kappaleen pisteet ja löytämällä valittujen akseleiden suunnissa kappaleen ääriarvot. Miten ääriarvot tallennetaan muistiin, on olemassa muutama vaihtoehto, joka määrää myös niiden törmäysten havaitsemi-

seen käytetyt menetelmät. Pienimillään kolmiulotteisessa avaruudessa AABB voidaan mahduttaa kuuteen skalaariin. Tällöin AABB voidaan tallentaa käyttäen keskipistettä ja särmien puolipituuksia tai minimi ja maksimi pisteitä. Keskipisteen ja särmien puolipituuksien avulla määriteltyjen kahden AABB:n A:n ja B:n välinen leikkaavuuden määrittely voidaan suorittaa testaamalla $|a_{keski} - b_{keski}| \leq a_{p.pituus} + b_{p.pituus}$ kaikilla akseleilla. Minimi ja maksimi pisteillä määriteltyjen AABB:n välinen leikkaavuuden määrittely voidaan puolestaan suorittaa testaamalla $a_{min} \leq b_{max}$ ja $a_{max} \leq b_{min}$ kaikilla akselilla.

Jotta kahden AABB:n leikkaavuuden määrittely säilyy yksinkertaisena, tulee AABB päivittää siihen liitetyn kappaleen kiertyessä, silloin kun se ei enää kokonaisuudessaan peitä sisäänsä suljettua törmäyskappaletta. Yleisimmin käytetty tapa AABB:n päivittämiseen on laskea uusi AABB edellisen AABB:n mukaan (Ericson 2004, s. 86). Tällä tavoin muodostetun AABB:n tilavuus tosin on edellisen kehyksen AABB:ta suurempi, jolloin on tärkeää suorittaa päivittäminen alkuperäisen AABB avulla (Ericson 2004, s. 86). Jos puolestaan päivitys suoritettaisiin edellisen kehyksen AABB:n avulla, kasvaisi AABB entisestään jokaisella päivityksellä. Tutkielmassa AABB:ta tullaan tarvitsemaan luvussa 3.2 esitetyssä Sweep-and-Prune algoritmissa, sekä luvussa 4 esitettävässä rajaavien tilavuuksien hierarkiassa.

2.7 Erottavan akselin teoreema

Törmäystarkastelussa usein tulee vastaan käsite erottavan akselin teoreema (engl. separating axis theorem). Teoreema pohjautuu erottavan hypertason teoreemaan (engl. separating hyperplane theorem) (Boyd ja Vandenberghe 2004, s. 46). Hypertaso on varsinaisen avaruuden osajoukko; jos varsinainen avaruus on R^n, niin hypertaso on H^{n-1}. (Rockafellar 1997, s. 4-5). Hypertaso erottaa kaksi konveksia epätyhjää joukkoa toisistaan, jos konveksit joukot sijaitsevat hypertason luomissa eri suljetuissa puoliavaruuksissa (Rockafellar 1997, s. 95). Toisin sanoen, jos kolmiulotteisessa avaruudessa voidaan kahden konveksin avaruuskappaleen väliin sijoittaa ääretön taso, siten että avaruuskappaleet sijaitsevat kokonaisuudessa tämän tason eri puolilla, eivät avaruuskappaleet leikkaa toisiansa. Tällöin kyseistä tasoa kutsutaan erottavaksi tasoksi ja tason normaalia erottavaksi akseliksi. Toisistaan erillään olevilla konvekseilla kappaleilla on aina löydettävissä erottava hypertaso, mutta sama ei päde aina konkaaveille kappaleille (Ericson 2004, s. 64).



Kuvio 6. SAT, jossa erottavat hypertasot s ja kantavat hypertasot P ja kantavat ominaisuudet F.

Toinen läheinen teoreema, mitä törmäystarkastelussa tullaan tarvitsemaan on kantavan hypertason teoreema (engl. supporting hyperplane theorem) (Boyd ja Vandenberghe 2004, s. 51). Hypertaso määritellään epätyhjän konveksin joukon C:n reunapisteen x_0 kantavaksi hypertasoksi, jos se erottaa reunapisteen x_0 ja kaikki $x \in C$, missä $x \neq x_0$ toisistaan. Monikulmiollisten konveksien tapauksessa voi kantavalla hypertasolla sijaita useampia reunapisteitä. Kantavalla hypertasolla sijaitsevia monitahokkaan ominaisuuksia kutsutaan kantaviksi ominaisuuksiksi (engl. supporting feature).

Molempia teoreemia hyödynnetään niin kutsutussa erottavan akselin testissä (engl. separating axis test, SAT). Alun perin erottavia akseleita hyödyntävä menetelmä rajaavien laatikoiden leikkaavuuden määrittämiseen esiteltiin artikkelissa (Gottschalk, Lin ja Manocha 1996). Termi SAT tuotiin esille myöhemmin artikkelissa (G. v. d. Bergen 1997), jolla viitataan Gottschalk, Lin ja Manocha (1996) esittelemään menetelmään. SAT:ssa tarkoitus on löytää konvekseille kappaleille erottavaa tasoa kohtisuorassa oleva erottava akseli, jolle projisoituna kappaleet eivät leikkaa toisiansa. Jos tällainen akseli löytyy, eivät kappaleet törmää toisiinsa. Jos puolestaan erottavaa akselia ei kyetä löytämään, törmäävät kappaleet. Tarkemmin SAT esitellään kuviossa 6 oikealla.

Koska erillään olevien konveksien kappaleiden väliin voidaan muodostaa ääretön määrä erotta-

via tasoja ja täten erottavia akseleita, mikä näkyy myös kuviossa 6 vasemmalla, ei niitä kaikkia ole mahdollista testata. Kahden monitahokkaan mahdolliset erottavat akselit ovat monitahokkaiden tahkojen normaalien suuntaiset akselit, sekä kaikkien monitahokkaiden välisten särmä parien ristitulojen suuntaiset akselit (Gottschalk, Lin ja Manocha 1996). Tällöin kahden vapaasti kiertyneen suorakulmaisen särmiön törmäyksen havaitseminen voidaan suorittaa testaamalla viisitoista mahdollista erottavaa akselia (Gottschalk, Lin ja Manocha 1996). Näistä kuusi testattavaa akselia ovat tahkojen normaalien suuntaisia, sillä suorakulmaisen särmiön tahkojen normaaleista vain kolme ovat keskenään erisuuntaisia. Samoin suorakulmaisen särmiön särmistä vain kolme ovat keskenään erisuuntaisia, jolloin mahdollisia särmä parien ristitulojen suuntaisia testattavia akseleita on yhdeksän.

Yksinkertaisilla geometrisilla kappaleilla, kuten suorakulmaisella särmiöllä edellä kuvattu brute-force menetelmään perustuva SAT saattaa olla riittävän tehokas, mutta koska testattavien särmä parien määrä kasvaa neliöllisesti särmien suhteen, ei SAT sellaisenaan sovellu käytettäväksi reaaliajassa monimutkaisemmille monitahokkaille. Monimutkaisemmille monitahokkaille esitetään luvussa 5.3 optimointimenetelmä särmäparien vähentämiseksi. Suorakulmaisille särmiöille sen sijaan yksinkertainen optimointimenetelmä on testata ensimmäisenä tahkojen normaalien suuntaiset erottavat akselit (G. v. d. Bergen 1997). Tämä perustuu G. v. d. Bergen (1997) suorittamiin empiirisiin kokeisiin, joissa kahdelle toisistaan erillään oleville suorakulmaiselle särmiölle kyettiin löytämään erottava akseli 85% kokeista testaamalla pelkästään tahkojen normaalien suuntaiset erottavat akselit. Menetelmä, jossa testataan pelkästään tahkojen normaalien suuntaiset erottavat akselit, tunnetaan nimellä SAT lite (G. v. d. Bergen 1997).

3 Laajan vaiheen törmäystarkastelu

Laaja vaihe on tutkielmassa noudatettavan liukuhihnamallin ensimmäinen vaihe. Tämän vaiheen tarkoituksena on löytää potentiaalisesti törmäävien parien lista (PCS). Potentiaalisesti törmäävien parien lista muodostetaan poissulkemalla tarkasteltavien kappaleiden joukosta ne parit, joiden ei todeta olevan todennäköistä törmätä toisiinsa. Poissulkeminen pyritään suorittamaan laskennallisesti yksinkertaisilla ja nopeilla testeillä ilman, että törmäyksiä pyritään havaitsemaan sen tarkemmin. Yksi tapa yksinkertaiseen poissulkemiseen on suorittaa törmäysten havaitseminen luvussa 2.6 esitettyjen rajaavien tilavuuksien välillä. Rajaavista tilavuuksista yleisimmin laajassa vaiheessa käytetään AABB:ta (G. v. d. Bergen 2004a, s. 72).

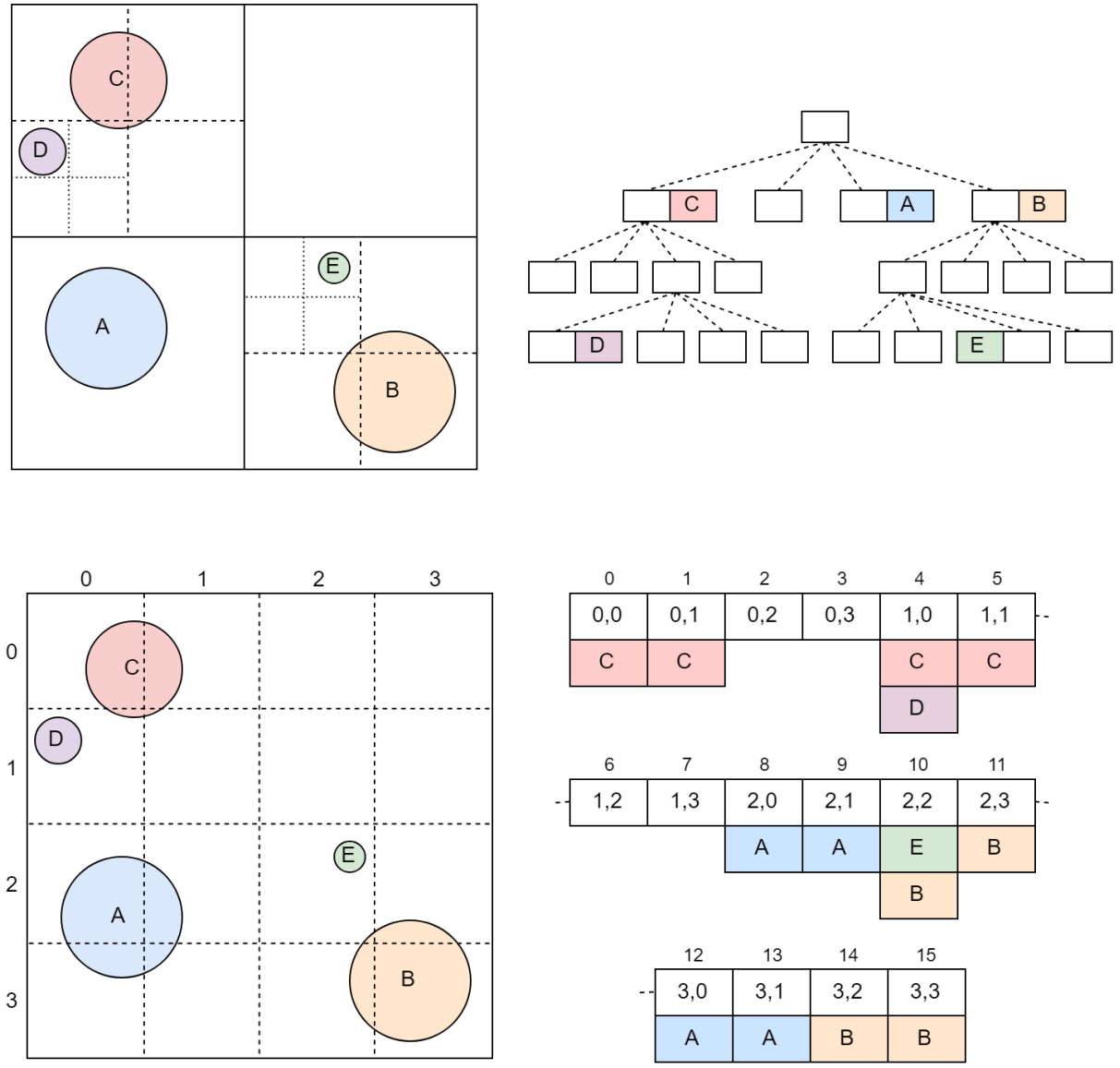
Pelkästään rajaavia tilavuuksia käyttämällä naiivi ratkaisu olisi käydä kaikkien kappaleiden rajaavat tilavuudet keskenään ja löytää ne kappale parit joiden rajaavat tilavuudet leikkaavat. Tällöin naiivin ratkaisun kompleksisuus kappaleiden n suhteen olisi $O(n^2)$. Tämän kaltainen naiivi ratkaisu saattaa toimia ympäristöissä joissa tarkasteltavia kappaleita on vähän, mutta esimerkiksi videopelissä, jossa pelimaailma koostuu useista tuhansista kappaleista, tarvitaan laskennallisesti tehokkaampia menetelmiä, jotta mahdolliset törmäykset kyetään reaaliajassa havaitsemaan. Yleisimmät menetelmät laajassa vaiheessa ovat avaruuden osittamisen metodit ja metodit, joita kirjassa (Weller 2013, s. 13) kutsutaan topologisiksi metodeiksi. Näistä avaruuden osittamisen metodit perustuvat avaruuden jakamiseen pienempiin tarkasteltaviin osiin ja topologiset metodit puolestaan perustuvat kappaleiden suhteellisten etäisyyksien vertailemiseen. Taulukkoon 2 on listattu tarkastelluissa kirjastoissa laajassa vaiheessa esiintyvät metodit, joista Sweep-and-prune (SAP) algoritmi ja sen variantit tullaan esittelemään alaluvuissa 3.2 ja 3.3 ja avaruuden osittamisen metodit, kuten kasipuu luvussa 3.1.

Taulukko 2. Laajan vaiheen algoritmit tarkasteltavissa kirjastoissa

PhysX 3.4	Bullet Physics 2.87	ODE 0.15	SOLID 3.5
Sweep-and-prune (SAP)	SAP	SAP	SAP
Multi box pruning (MBP)	Multi-SAP	Nelipuu	
	Dynaaminen AABB puu	AABB hajautustaulu	

3.1 Avaruuden osittaminen

Avaruuden osittamisen metodit perustuvat avaruuden jakamiseen pienempiin tarkasteltaviin konvekseihin osiin, soluihin (engl. cell). Tällöin potentiaalisesti törmäävien parien lista muodostetaan samaan soluun kuuluvista kappaleista. Kappale katsotaan kuuluvaksi soluun, jos se sisältyy soluun kokonaan tai se leikkaa solun seinämän. Jälkimmäisestä ehdosta seuraa, että sama kappale voi kuulua useampaan soluun. Riippuen metodista, voivat solut sulkea sisäänsä toisia soluja ja täten muodostaa hierarkkisen rakenteen. Esimerkki ei-hierarkkisesta rakenteesta on ruudukko (engl. grid). Hierarkkisia rakenteita puolestaan ovat muun muassa: kasipuu, k-d puu ja BSP puu. Törmäystarkastelun ohella avaruuden osittamisen metodeja hyödynnetään muun muassa renderöinnissä ja säteenseurannassa (Akenine-Möller, Haines ja Hoffman 2008). Avaruuden osittamisen metodeista tässä luvussa tullaan tarkemmin esittelemään ruudukko (engl. grid) ja kasipuu (engl. octree). Kyseiset avaruuden osittamisen metodit on esitetty kuviossa 7. Laajemmin avaruuden osittamisen metodit ovat käsitelty muun muassa kirjassa (Ericson 2004, s. 285-382).



Kuvio 7. Nelipuu ja ruudukko.

Ruudukko on varsin yksinkertainen avaruuden osittamisen rakenne, jossa avaruus jaetaan vierekkäisiin suorakulmaisiin soluihin, ruutuihin. Usein ruudut ovat tasasivuisia eli säännöllisiä, jolloin täsmällisemmin voidaan puhua säännöllisestä ruudukosta (engl. uniform grid). Tämän lisäksi voidaan erotella kaksiulotteinen ruudukko ja sen kolmiulotteinen vastike, josta G. v. d. Bergen (2004a, s. 176) käyttää nimitystä vokseliruudukko (engl. voxel grid). Kolmiulotteinen ympäristö mahdollistaa, sekä kaksiulotteisen ruudukon, että vokseliruudukon käytön. Edellä mainittujen

erotteluiden lisäksi ruudukko voidaan erotella myös sen toteutustavan perusteella. Erilaisia toteutustapoja ovat muun muassa taulukko ja hajautustaulu. Hajautustauluna toteutettuna näkee käytettävän myös nimitystä avaruudellinen hajauttaminen (engl. spatial hashing), jolloin niistä puhutaan erillisenä, ruudukosta eroavana, rakenteena (Weller 2013, s. 27). Avaruudellisessa hajauttamisessa solut hajautetaan hajautusfunktion avulla eri lokeroihin, joissa säilytetään listaa siihen kuuluvista kappaleista. Kuten taulukkona toteutetussa ruudukossa, ovat naapurisolut edelleen helposti löydettävissä laskemalla hajautus viereisille indekseille (G. v. d. Bergen 2013).

Vaikka G. v. d. Bergen (2004a, s. 176) mainitsee kirjassaan ruudukon hyviksi puoliksi pienen muistin tarpeen ja tehokkuuden, ovat ruudukon hyvät puolet suuresti riippuvaisia kappaleiden tasaisesta jakautumisesta ympäristössä, sekä solun ja kappaleiden suhteellisesta koosta. Kappaleiden epätasainen jakautuminen johtaa tiettyjen solujen ruuhkautumiseen ja mahdollisesti toisten solujen jäämiseen tyhjiksi, jolloin ruudukko ei tehokkaasti kykene ympäristöä jakamaan. Kun puolestaan ympäristön kappaleet ovat suuria solujen kokoon nähden, katsotaan useat kappaleet tällöin kuuluvaksi useampaan soluun, joka myös johtaa tehottomuuteen varsinkin, jos nämä suuret kappaleet ovat dynaamisia, jolloin niiden liikkuminen solujen välillä vaatii monia muutoksia soluihin kuuluvien kappaleiden listoihin. Molemmissa edellä mainituissa tapauksissa kyse on oikean solun koon löytämisestä, mikä onkin ruudukon suurin haaste (G. v. d. Bergen 2004a, s. 176, Weller 2013, s. 27). Parhaiten ruudukko soveltuu tasaisen tiheyden omaavien muokkautuvien kappaleiden törmäysten havaitsemiseen, kuten esimerkiksi vaatteiden ja nesteiden (G. v. d. Bergen 2013). Toinen mihin ruudukko soveltuu hyvin käytettäväksi, on luvussa 3.3 esiteltävä niin kutsuttu hybridi metodi.

Sen sijaan että koko avaruus jaetaan säännöllisiin soluihin kuten ruudukossa, pyrkivät hierarkiset rakenteet suorittamaan osittamisen kappaleiden sijaintien mukaisesti. Kasipuussa juurisolmu on kuutio ja osittaminen suoritetaan rekursiivisesti siten, että vanhempisolmu jaetaan kahdeksaan säännölliseen ja yhtä suureen lapsisolmuun. Osittaminen lopetetaan, kun puu saavuttaa sille asetettavan maksimi syvyyden tai uuden solmun määrittelemä solu tavoittaa sille asetetun minimi koon (Ericson 2004, s. 308). Solmujen määrittelemien solujen rajoja leikkaavat kappaleet voidaan sijoittaan useampaan solmuun kuuluvaksi, kuten ruudukossa. Tällöin kasipuun kaikki kappaleet sijoitetaan lehtisolmuihin. Toinen vaihtoehto on säilyttää kappaleita puurakenteen eri

tasoilla. Jälkimmäisen vaihtoehdon heikkoutena on kooltaan pienten kappaleiden mahdollinen joutuminen korkealle hierarkiassa. Tämän estämiseksi voidaan tavallisen kasipuun sijasta käyttää Ulrich (2000) esittelemää löyhää kasipuuta (engl. loose octree). Löyhässä kasipuussa solmujen määräämiä soluja laajennetaan pienellä mitalla, siten että solut leikkaavat toisiansa, joka estää laajennuksen sisään mahtuvien kappaleiden ajautumisen korkealle hierarkiassa.

Tyypillinen tapa muodostaa kasipuu on osoittimien avulla, mutta se voidaan toteuttaa myös ilman osoittimia, kuten esitellään artikkelissa (Gargantini 1982). Näin toteutetusta kasipuusta käytetään nimitystä lineaarinen kasipuu. Lineaarisessa kasipuussa jokainen solmu esitetään sijainti koodin avulla. Yleisimmin Gargantini (1982) käyttämän koodauksen sijasta käytetään Mortonin koodausta, kuten esitellään muun muassa kirjassa (Ericson 2004, s. 314-318). Tällöin kahdeksalle lapsisolmuille annetaan sijainti koodeiksi luvut nollasta seitsemään binäärisenä: 000, 001, 010, 011, 100, 101, 110 ja 111. Jokainen sijainti koodi kertoo, missä suunnassa lapsisolmu sijaitsee. Yhdistelemällä sijaniti koodeja saadaan solmun vastaava taulukon indeksi muodostettua. Mortonin koodin muodostaminen koordinaateista tapahtuu lomittamalla koordinaatit binääri lukuina esitettyinä (Ericson 2004, s. 316). Tarkempi Mortonin koodin muodostamisen kuvaaminen rajataan tämän tutkielman ulkopuolelle. Hyvä lähde eri tapoihin muodostaa Mortonin koodi esitellään sivulla (Anderson 2005).

3.2 Sweep-and-Prune algoritmi

“Kaikki suurimmat fysiikkamoottorit hyödyntävät Sweep-and-Prune -algoritmia (SAP) potentiaalisten törmäysparien löytämiseen laajassa vaiheessa“ (Gregory 2009, s. 677, suomennos minun). Tämä näkyy myös taulukossa 2. Alun perin algoritmin esitteli Baraff (1992, s. 53) nimellä Sort and Sweep, mutta yleisemmin kirjallisuudessa algoritmista näkee käytettävän nimeä Sweep-and-Prune, kuten myös artikkelissa (Cohen ym. 1995), jossa ensimmäisenä kuvataan algoritmin implementointi. Tässä luvussa esitellään perinteinen SAP-algoritmi ja luvussa 3.3 katsotaan muutoksia perinteiseen SAP-algoritmiin.

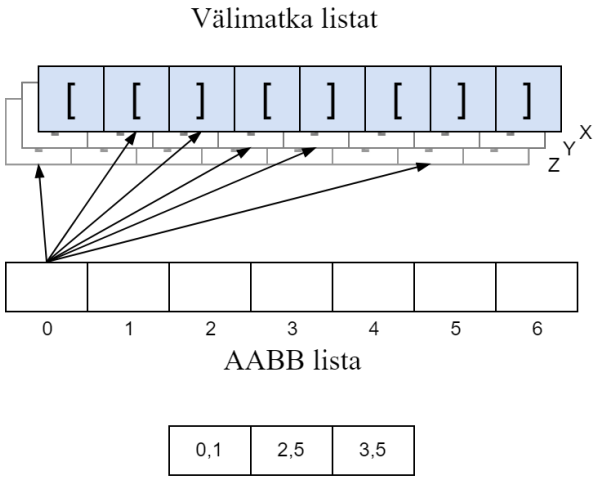
SAP-algoritmi perustuu siihen, että kaksi AABB:ta leikkaavat kolmiulotteisessa avaruudessa toisensa, jos ja vain jos niiden ortogonaaliset projektiot x-, y- ja z-akseleilla leikkaavat. Idea on

sama mitä nähtiin erottavan akselin teoreemassa luvussa 2.7. Tästä moniulotteisen avaruuden tarkastelemisesta useampana yksiulotteisena projektiona Cohen ym. (1995) käyttävät nimitystä avaruuden supistaminen. Kolmiulotteisessa avaruudessa avaruuden supistamiseksi yksiulotteisiin projektioihin tarvitaan kolme AABB:en ääriarvoista koostuvaa järjestettyä listaa. Yksittäisestä järjestetystä listasta käytetään nimitystä välimatkalista (engl. interval list). Perinteisesti listan toteuttamiseen käytetään taulukkoa tai linkitettyä listaa (G. v. d. Bergen 2004a, s. 212; Ericson 2004, s. 330-338). Itsessään SAP-algoritmi jaetaan kolmeen vaiheeseen: AABB:den päivittämiseen, edellä kuvattujen välimatkalistojen lajitteluun, sekä niiden läpikäymiseen eli potentiaalisesti törmäävien kappaleiden määrittämiseen (Cohen ym. 1995). Näistä kaksi jälkimmäistä suoritetaan samanaikaisesti (Cohen ym. 1995).

AABB:en päivittäminen on seurausta siihen liitetyn kappaleen kiertymisestä, kuten esiteltiin luvussa 2.6. Kiertymän lisäksi muut muunnokset kuten translaatio vaikuttavat AABB:n ääriarvojen muuttumiseen. Translaatio ei johda itse AABB:n päivittämiseen, mutta se saattaa johtaa välimatkalistojen epäjärjestykseen, jolloin välimatkalistat täytyy luoda uudelleen. Uudelleen luomisen sijasta, suoritetaan tämä yleensä lajittelemalla. Lajittelualgoritmeista suositaan käytettäväksi lisäyslajittelua (Cohen ym. 1995; Baraff 1992; Tracy, Buss ja Woods 2009), sillä se tuottaa lähes lineaarisen aikavaativuuden lähes lajitellussa listassa. Välimatkalistat oletetaan olevan lähes järjestyksessä, kun kehyskoherenssi on korkea. Lajittelu vaatii AABB:den ääriarvojen vertailun, joka voidaan suoritta joko FPU:lla tai CPU:lla. Kokonaislukujen käyttö antaa nopeamman lukujen vertailun ja vähemmän muistin tarpeen, mutta tuottaa epätarkempia tuloksia. Bullet Physics kirjastossa on SAP-algoritmit sekä kokonaisluvuilla, että 32-bittisillä liukuluvuilla toteutettuna.

Potentiaalisten törmäävien kappaleiden määrittäminen suoritetaan yleensä inkrementaalisesti, kuten suoritetaan myös algoritmin alkuperäisessä (Baraff 1992) toteutuksessa. Inkrementaalisuudella tarkoitetaan sitä, että potentiaalisesti törmäävien parien listaa ei joka kehyksellä luoda tyhjästä, vaan käytetään edellisellä kehyksellä muodostettua listaa, kohdistaen siihen tarpeen mukaan lisäyksiä ja poistoja. Potentiaalisesti törmäävien parien listaan kohdistuvat lisäykset ja poistot johtuvat välimatkalistojen lajittelussa suoritettavista vaihdoista. Kun kahden kappaleen AABB:den ääriarvot vaihtavat välimatkalistalla paikkaa keskenään, voivat kappaleiden projektiot kyseisellä akselilla joko alkaa leikkaamaan tai lakata leikkaamasta. Jos näiden kappaleiden

projektiot kyseisellä akselilla lakkasivat leikkaamasta, poistetaan pari potentiaalisesti törmäävien parien listalta, mikäli se on sinne listattuna. Jos puolestaan kappaleiden projektiot alkoivat leikkaamaan kyseisellä akselilla, täytyy tarkastelu ulottaa myös muille akseleille. Jos kappaleet leikkaavat kaikilla välimatkalistoilla, tulee kyseinen pari lisätä potentiaalisesti törmäävien parien listalle. Edellä esiteltyjen välimatkalistojen ja potentiaalisesti törmäävien parien listan lisäksi SAP:n datarakenteeseen kuuluu lista kaikista AABB:sta (Tracy, Buss ja Woods 2009), jonka avulla välimatkalistoilla sijaitsevat ääriarvot voidaan liittää eri kappaleisiin. SAP-algoritmin datarakenne esitetään kuviossa 8.



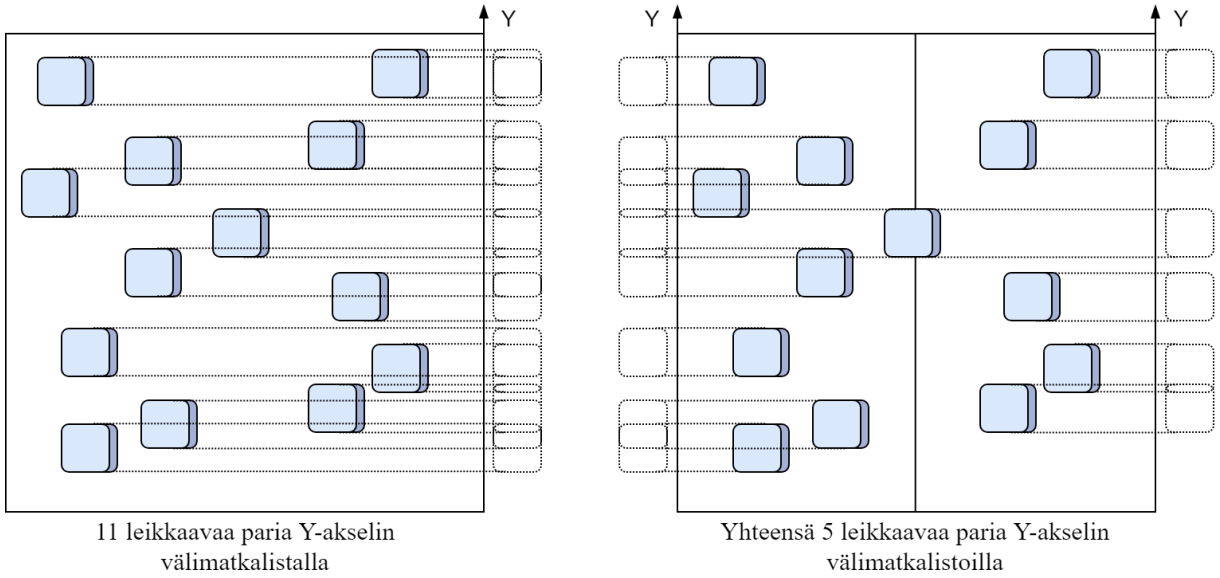
Kuvio 8. SAP-algoritmin rakenne.

3.3 Muutokset SAP-algoritmiin

SAP-algoritmin suuren suosion myötä on alkuperäiseen, niin kutsuttuun perinteiseen SAP-algoritmiin tuotu monia uudistuksia. Artikkelissa (Tracy, Buss ja Woods 2009) esitellään kaksi jo entuudestaan käytössä olevaa muutosta perinteiseen SAP-algoritmiin, sekä yksi uusi muutos. Artikkelissa esitellyt muutokset korjaavat perinteistä SAP-algoritmia vaivaavat kaksi ongelmaa: kappaleiden poiston ja lisäämisen hitaus välimatkalistoilla, sekä vaihtojen aikavaativuuden. Vaihtojen aikavaativuuden pienentämiseen Tracy, Buss ja Woods (2009) esittelevät hybridi metodin, joka yleisemmin tunnetaan nimellä Multi-SAP. Kappaleiden poiston ja lisäämisen nopeuttamiseksi Tracy,

Buss ja Woods (2009) esittelevät jo olemassa olevan metodin, erissä tapahtuva lisääminen ja poistaminen (engl. patch insertion and removal), sekä uuden rakenteellisen muutoksen perinteiseen SAP:in, joka kantaa nimeä segmentoitu välimatkalista (engl. segmented interval list).

“Suurin osa perinteisen SAP:n ajasta kuluu vaihtoihin laajoissa ympäristöissä“ Tracy, Buss ja Woods (2009, suomennos minun). Tarkemmin tämä pätee ympäristöissä, joissa kappaleet ovat tiheästi sijoittuneet ja joista ainakin osa on dynaamisia. Tiheästi sijoittuneiden kappaleiden rasitus voidaan eritoten nähdä korkeana aksiaalisena tiheytenä välimatkalistoilla. Tällöin vaihtoihin kuluva pitkä aika johtaa suuresta määrästä suoritettavia vaihtoja, sillä pienikin translaatio kappaleelle saattaa korkeasti aksiaalisesti tihentyneellä välimatkalistalla vaatia useamman vaihdon välimatkalistoilla. Aksiaalinen tiheys laskee käytettäessä hybridimetodia (Tracy, Buss ja Woods 2009) eli SAP-algoritmin ja avaruuden osittamisen yhdistelmää, jolloin jokainen solu, joita tässä tapauksessa voidaan kutsua myös alueiksi, sisältävät itsenäisen SAP-algoritmin mukaisen datarakenteen. Aksiaalinen tiheyden vertailu perinteisen SAP-algoritmin ja Multi-SAP-algoritmin välillä esitellään kuviossa 9. Tyypillisesti avaruuden osittamisen rakenteista suositaan ruudukkoa (Tracy, Buss ja Woods 2009). Tosin esimerkiksi PhysX:n MVP (multi box pruning) algoritmi antaa käyttäjän määritellä vapaammin AABB:n muotoiset alueet. Yleisesti alueiden nopea paikantaminen tapahtuu ylläpitämällä listaa, johon on tallennettuna kaikkien AABB:n osalta ne alueet tai yksittäinen alue, johon se kuuluu. Tämän listan ja käytettävän avaruuden osittamisen rakenteen muodostamasta kokonaisuudesta Tracy, Buss ja Woods (2009) käyttävät nimitystä ylärakenne (engl. superstructure).

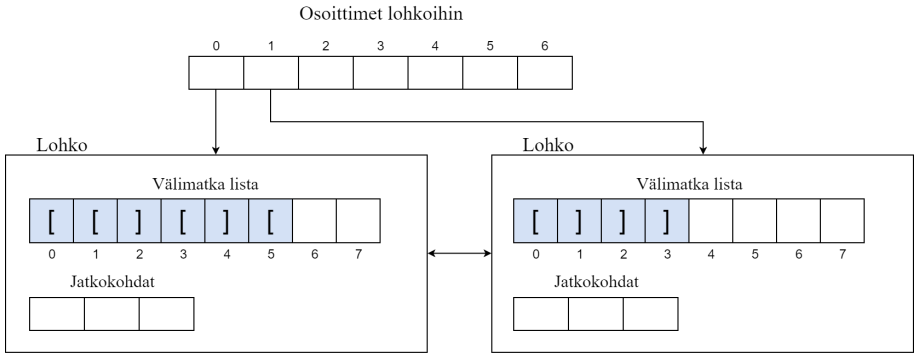


Kuvio 9. Aksiaalinen tiheys perinteisessä SAP-algoritmissa ja Multi-SAP-algoritmissa.

Ylärakenne hallitsee AABB:den sijoittumista soluihin ja niiden liikkumista solujen välillä. Toisin sanoen ylärakenne vastaa kappaleiden lisäämisestä, poistamisesta ja päivittämisestä. Ylärakenteen määrittelemien alueiden rajoja osittain ylittävät kappaleiden AABB:t lisätään kuuluvaksi useamman alueen välimatkalistoille. Lisääminen ja poisto tapahtuvat ylärakenteen kautta kahdessa vaiheessa: ensiksi ylärakenne yhdistää AABB:n oikeaan tai oikeisiin välimatkalistoihin, jonka jälkeen välimatkalistoissa AABB:n ääriarvot lisätään oikeille paikoille tai ne poistetaan oikeista paikoista. Jälkimmäinen vaihe vaatii myös potentiaalisesti törmäävien parien listan tarkastamisen. Tämä on sama menetelmä kuin perinteisessä SAP-algoritmissa, sillä eroavaisuudella, että ensiksi tulee löytää oikeat välimatkalistat, joissa lisättävän tai poistettavan AABB:n ääriarvot sijaitsevat. Samoin ääriarvojen päivitys tapahtuu muuten vastaavasti samoin kuin perinteisessä SAP-algoritmissa, paitsi kappaleen ylittäessä alueen rajat, jolloin ääriarvojen päivitys kohdistuu useamman alueen välimatkalistoihin.

Vaikka Multi-SAP-algoritmi osakseen pienentää kappaleiden poistamiseen ja kappaleiden lisäämiseen kuluvaa aikaa, lisäävät solujen rajoja ylittävät kappaleet suoritettavien lisäyksien ja poistojen lukumäärää. Lisäyksiä ja poistoja voidaan nopeuttaa suorittamalla samalle välimatkalistalle kohdistuvat lisäykset tai poistot erissä (Tracy, Buss ja Woods 2009). Tällöin suoritettavien

lisättävien ja poistettavien kappaleiden vastaavat ääriarvot tallennetaan väliaikaisiin puskureihin siten, että jokaiselle välimatkalistalle on oma puskurinsa. Kun kaikki ääriarvot ovat puskureissa, lajitellaan puskureiden sisältämät ääriarvot suuruusjärjestykseen, jonka jälkeen yhden puskurin eli erän sisältämät ääriarvot voidaan lisätä välimatkalistoille tai poistaa välimatkalistoilta peräkkäin. Pahimmassa tapauksessa tämä vaatii kaikkien välimatkalistojen läpi käymisen kokonaisuudessaan. Koska erissä tapahtuvien lisäyksien ja poistojen tuoma hyöty on riippuvainen yhtä aikaan suoritettavista ääriarvojen lisäyksistä ja poistoista, ei se juurikaan hyödytä Multi-SAP-algoritmin kanssa käytettäväksi ympäristöissä, joissa suuri osa kappaleista on staattisia (Tracy, Buss ja Woods 2009). Tällöin paremmin käytettäväksi soveltuu Tracy, Buss ja Woods (2009) esittelemä segmentoitujen välimatkalistojen käyttö.



Kuvio 10. Lohkotetun välimatkalistan rakenne (Tracy, Buss ja Woods 2009).

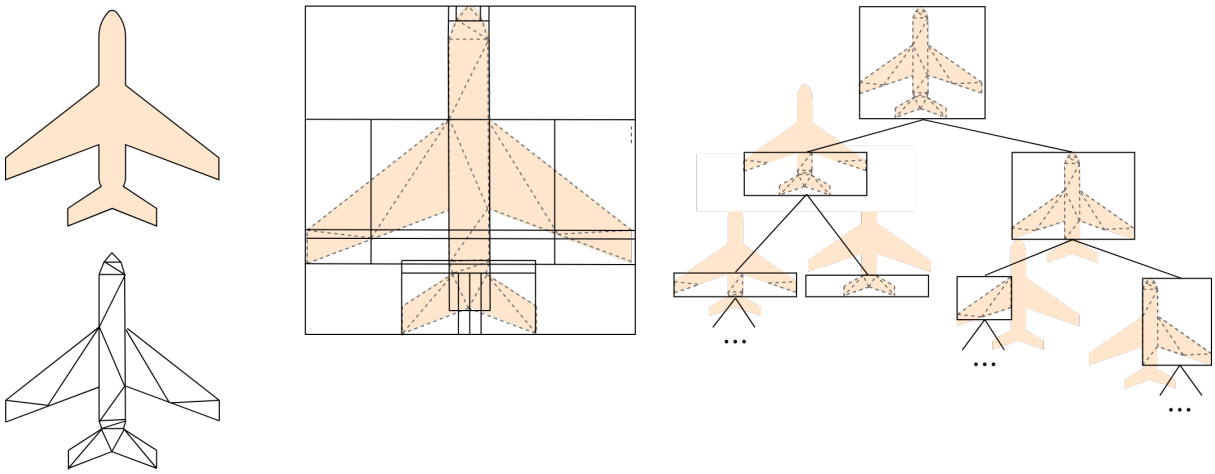
Kuten luvussa 3.2 todettiin, käytetään perinteisen SAP-algoritmin datarakenteessa taulukkoja tai linkitettyjä listoja välimatkalistojen toteuttamiseen. Segmentoidut välimatkalistat datarakenne sen sijaan käyttää välimatkalistana linkitettyä listaa, jonka alkiot ovat pieniä taulukoita (Tracy, Buss ja Woods 2009). Pientä taulukkoa sisältävää linkitetyn listan alkiota kutsutaan lohkoksi (egln. chunk). Tämän lisäksi datarakenteeseen kuuluu järjestetty taulukko, joka sisältää osoittimet kaikkiin lohkoihin (Tracy, Buss ja Woods 2009). Lohkojen sisältämät ääriarvoja sisältävät taulukot eivät välttämättä ole täysiä, jolloin ne mahdollistavat ääriarvojen lisäykset ja poistot, ilman että vaihtoja tarvitsee suorittaa koko listan matkalta. Jokaiseen lohkoon kuuluu myös jatkokohta lista, jossa säilytetään tieto niistä AABB:sta joiden minimiarvo on kyseisessä tai sitä edeltävässä lohkossa, ja joiden maksimiarvo on kyseistä lohkoa seuraavassa lohkossa. Tämä an-

taa paikallista tietoa niistä ääriarvoista, jotka ovat mielivaltaisen kaukana listassa. Jatkokohdat listaa tulee säilyttää ainoastaan yhdelle ulottuvuudelle (Tracy, Buss ja Woods 2009). Tarkemmin segmentoitujen välimatkalistojen datarakenne kuvattu kuviossa 10.

Ääriarvon lisäämisen nopeutuminen johtaa siitä, että lisäys lohkon ääriarvo listalle, joka ei ole täysi, määräytyy suoritettavien vaihtojen yläraja kyseisen ääriarvolistan alkioiden lukumäärän mukaan. Samoin poistoissa ääriarvo listan alkioiden lukumäärä määrittää suoritettavien vaihtojen ylärajan. Enemmän vaivaa puolestaan aiheutuu lisäyksistä, jotka kohdistetaan ääriarvo listalle, joka on täysi. Tällöin suoritetaan lohkon halkaisu (Tracy, Buss ja Woods 2009). Lohkon halkaisussa puolet halkaistavan lohkon ääriarvoista siirretään uuteen lohkoon, joka sijoitetaan halkaistavan lohkon perään. Samalla uusi lohko perii halkaistavan lohkon jatkokohta listan, jonka jälkeen sekä halkaistavan, että uuden lohkon jatkokohta listat tulee päivittää. Myös ääriarvon poistamisen yhteydessä voidaan pirstaloitumisen vähentämiseksi suorittaa kahden lohkon yhdistäminen (Tracy, Buss ja Woods 2009). On myös hyvä huomata, että niiden kappaleiden lisäykset tai poistot, joissa käsitellään eri lohkojen ääriarvo listoja, täytyy jatkokohta lista päivittää vastaavasti (Tracy, Buss ja Woods 2009). Erityisesti nopeasti liikkuvien kappaleiden osalta ääriarvojen vaihtojen aiheuttamat muutokset jatkokohta listoihin tulee suorittaa puskurin kautta. Tällöin molempien ääriarvojen vaihdoista mahdollisesti aiheutuvat muutokset jatkokohtaan listaan tallennetaan puskuriin, jonka mukaan suoritettujen vaihtojen jälkeen päivitetään jatkokohta lista (Tracy, Buss ja Woods 2009).

4 Keskivaiheen törmäystarkastelu

Keskivaiheessa tarkoitus on suorittaa törmäysten havaitseminen niille potentiaalisesti törmääville pareille, joista ainakin toinen koostuu useammasta törmäysprimitiivistä. Törmäyksen havaitsemisella keskivaiheessa pyritään löytämään todelliset potentiaalisesti törmäävät törmäyskappaleparit. Tällöin useammasta törmäysprimitiivistä koostuvasta kappaleesta rajataan pois ne törmäysprimitiivit, joiden ei ole mahdollista törmätä. Keskivaiheessa törmäysten havaitseminen suoritetaan käyttäen rajaavien tilavuuksien hierarkioita (engl. bounding volume hierarchy, BVH). Rajaavien tilavuuksien hierarkia muodostetaan ennen ohjelman ajoa samalla kun mahdollisesti konkaavi kappale jaetaan useampaan törmäysprimitiiviin. BVH:n juurisolmun muodostaa usein laajassa vaiheessa käytetty rajaava tilavuus ja sen lehtisolmut koostuvat törmäysprimitiivejä ympäröivistä rajaavista tilavuuksista. Useimmiten BVH muodostetaan mielivaltaisille kolmioverkolle, jolloin edellä mainitut törmäysprimitiivit ovat kolmiota, mutta on myös mahdollista luoda BVH useammasta monimutkaisemmasta törmäysprimitiivistä, kuten kapseleista. Tällöin puhutaan yhdistelmästä (engl. compound). Tämän lisäksi sitä on mahdollista käyttää laajan vaiheen algoritmina, kuten Bullet Physics fysiikkamoottorissa “Bullet Physics: Documentation” 2018, jolloin BVH muodostetaan kaikkien ympäristön kappaleiden välille. Kuviossa 11 on esitetty alkuperäinen kappale, siitä muodostettu törmäyskappale (kolmioverkko), sekä törmäyskappaleelle muodostettu rajaavien tilavuuksien hierarkia.



Kuvio 11. Rajaavien tilavuuksien hierarkia kolmioverkolle.

Luvussa 3 esitettyjen avaruuden osittamisen metodien ja BVH:n ero on siinä, että avaruuden osittamisessa pääpaino nimenomaan on avaruuden osittamisessa, kun taas rajaavien tilavuuksien hierarkiassa kyse on kappaleen tai kappaleiden tilan osittamisessa. Toisena eroavaisuutena avaruuden osittamisen metodeihin on, että kaksi tai useampaa rajaavaa tilavuutta saattavat leikata toisiansa ja törmäysprimitiivit sijoitetaan jokainen omaan lehtisolmuun (Ericson 2004, s. 235). Ei myöskään ole välttämätöntä, että vanhempisolmun rajaava tilavuus sisältää lapsisolmun rajaavan tilavuuden kokonaisuudessaan. Kuitenkin vanhempisolmun rajaavan tilavuuden täytyy sulkea sisäänsä lapsisolmun sisältämä törmäysprimitiivi tai primitiivit (Ericson 2004, s. 235). Kirjassaan Ericson (2004, s. 237) listaa joitakin BVH:lle toivottuja ominaisuuksia:

- Alipuiden solmujen tulisi olla mahdollisimman lähellä toisiansa.
- Jokaisen solmun hierarkiassa pitäisi olla mahdollisimman pieni.
- Rajaavien tilavuuksien yhteenlaskettu tilavuus tulisi olla mahdollisimman pieni.
- Enemmän huomiota hierarkiaa muodostettaessa tulisi kohdistaa solmuille juuren lähellä.
- Rajaavien tilavuuksien leikkaaminen sisarrussolmujen rajaavien tilavuuksien kanssa tulisi olla minimaalinen.
- Hierarkia tulisi olla tasapainotettu sekä solmu rakenteen, että sisällön kannalta.

Hyvin paljon edellä mainituista ominaisuuksista on kiinni käytetystä rajaavasta tilavuudesta, sekä puun muodostamisesta. BVH:ssa rajaavana tilavuutena voidaan käyttää monia rajaavia tilavuuksia, kuten esimerkiksi AABB:ta, OBB:ta, palloja ja k-DOP:ja (Akenine-Möller, Haines ja Hoffman 2008, s. 647-648). Puun muodostaminen tässä tutkielmassa lyhyesti tullaan esittelemään AABB puulle. AABB puu on käytössä PhysX, Bullet Physics ja ODE fysiikkamoottoreissa. Tämän lisäksi AABB puun variantti BoxTree puolestaan on käytössä SOLID 3.5 kirjastossa. Tarkemmin tarkasteltavien kirjastojen keskivaiheen algoritmit ovat lueteltu taulukossa 3.

Taulukko 3. Keskivaiheen algoritmit tarkasteltavissa kirjastoissa

PhysX 3.4	Bullet Physics 2.87	ODE 0.15	SOLID 3.5
AABB puu (kv)	AABB puu (kv) AABB puu (y)	AABB puu (kv)	BoxTree (kv)

Taulukossa merkitään: kv = kolmioverkko, y = törmäysprimitiivien yhdistelmä.

Käytetyin tapa muodostaa BVH on ylhäältä-alas (Weller 2013, s. 17; G. v. d. Bergen 2013). Samoin yleensä muodostetaan binäärisiä puurakenteita (G. v. d. Bergen 1997). Tällöin binäärisen AABB-puun tapauksessa aluksi rajataan koko kappale AABB:n sisälle. Tämän jälkeen kyseinen AABB jaetaan jotain heuristiikkaa käyttäen. Artikkelissa (G. v. d. Bergen 1997) esitetään jaettavan tason sijoittamista AABB:n pisimmän akselin mediaanin kohdalle. Tällöin AABB jakautuu kahteen yhtä suureen AABB:hen. Näin muodostettuun kahteen AABB:hen G. v. d. Bergen (1997) suorittaa kappaleiden osien sijoittamisen niiden keskipisteiden sijaintien mukaisesti. Jakamista suoritetaan rekursiivisesti, kunnes kaikki törmäysprimitiivit ovat lehtisolmuissa. Kuitenkin voidaan ajautua tilanteeseen, jossa AABB:n pisimmän akselin mediaanin mukaisesti suoritettava jako johtaa siihen, että kaikki törmäysprimitiivit sijoittuvat vain toiseen lapsisolmuista. Tällöin tulee käyttää jotain muuta heuristiikkaa jakavan tason määrittelemiseen, kuten esimerkiksi kappaleiden välisten sijaintien mediaania G. v. d. Bergen (1997).

Samoin kuin AABB täytyy päivittää siihen liitetyn kappaleen kiertyessä, tulee myöskin dynaamiselle kappaleelle muodostettu AABB puu päivittää kappaleen kiertyessä. Toisin kuin AABB puun rakentaminen, päivittäminen aloitetaan lehtisolmuista ja edetään kohti juurta. Tarkemmin tämä suoritetaan yleensä siten, että ensiksi päivitetään vanhempisolmun kaikki lapsisolmut ja vasta sen jälkeen päivitetään vanhempisolmu (G. v. d. Bergen 1997). Kun käytetään taulukkoa solmujen tallentamiseen, voidaan edellä mainittu päivitysjärjestys sisällyttää taulukkorakenteeseen siten, että annetaan jokaiselle lapsisolmulle suurempi taulukon indeksi kuin vanhempisolmulle, jolloin taulukon läpikäynti päinvastaisessa järjestyksessä pitää huolen päivitysten oikeellisesta järjestyksestä (G. v. d. Bergen 1997).

Kuten luvun alussa esiteltiin, suoritetaan törmäysten havaitseminen keskivaiheessa niiden poten-

tiaalisesti törmäävien parien osalta, joista ainakin toinen koostuu useammasta törmäysprimitiivistä. Tällöin törmäysten havaitseminen suoritetaan joko yhden kappaleen ja BVH:n välillä tai kahden BVH:n välillä. Näistä kahden BVH:n törmäysten havaitseminen voi tuottaa seuraavat tulokset (G. v. d. Bergen 2004a, s. 200):

- Kappaleet eivät leikkaa toisiansa.
- Jos molemmat leikkaavat solmut ovat lehtisolmuja, ollaan löydetty yksi potentiaalisesti törmäävä pari.
- Jos toinen on lehtisolmu ja toinen sisäsolmu, tulee lehtisolmua testata kaikkien sisäsolmun lapsisolmujen kanssa.
- Jos molemmat sisäsolmuja, testataan solmu, jolla pienempi rajaava tilavuus, toisen solmun lapsisolmujen kanssa.

Jos yksittäistä törmäyskappaletta ajatellaan lehtisolmuna, voi yksittäisen kappaleen ja BVH:n välinen törmäysten havaitseminen tuottaa edellä mainituista kolme ensimmäistä vaihtoehtoa. Keskivaiheen törmäysten havaitsemisen lopputuloksen perusteella muokataan laajasta vaiheesta saatua potentiaalisesti törmäävien parien listaa, ennen kuin se lähetetään eteenpäin kapeaan vaiheeseen.

5 Kapean vaiheen törmäystarkastelu

Kun keskivaiheesta kappaleiden konvekseista osista ja laajasta vaiheesta konvekseista kappaleista muodostettu potentiaalisesti törmäävien kappaleiden lista valmis, siirrytään tarkastelemaan törmäyksiä pareittain. Tämä tapahtuu luvussa 2 esitettyjen törmäyskappaleiden välillä, kun tähän asti kappaleiden potentiaalinen törmääminen suoritettiin käyttäen rajaavia tilavuuksia. Sen lisäksi, että kapeassa vaiheessa suoritetaan törmäysten havaitseminen, pyritään lisäksi tuottamaan tietoa törmäyskappale parien välisistä suhteista. Kappaleparit, joilla ei tapahdu törmäystä, tarkoittaa tämä mahdollisesti suoritettavan kappaleiden välisen minimaalisen etäisyyden laskemista. Törmäävillä kappale pareille puolestaan tämä tarkoittaa yhden kontaktimoniston muodostamista. Kontaktimonisto koostuu kontaktipisteistä, jotka puolestaan usein koostuvat sijainnista, normaalista ja penetraatiosyvyydestä (Catto 2007; Migdalskiy 2010). Jotta kappaleiden on mahdollista pysyä tasapainossa tulee kolmiulotteisessa avaruudessa kontaktimoniston koostua vähintään neljästä kontaktipisteestä (Gregorius 2015; Catto 2007). Dynaamisille kappaleille kontaktimonisto muodostetaan ensimmäisen törmäyksen aikana (TOI).

Kapean vaiheen osalta tutkielmassa tullaan taulukon 4 perusteella esittelemään tarkemmin simpleksiperustainen GJK-algoritmi alaluvussa 5.2, sekä optimoitu SAT-algoritmi alaluvussa 5.3. Tämän lisäksi tullaan esittelemään kaksi erilaista tapaa muodostaa kontaktimonisto: yhdellä kehyksellä tapahtuva täyden kontaktimoniston (engl. full contact manifold) muodostaminen, sekä inkrementaalinen kontaktimoniston muodostaminen, jolloin jokaisella kehyksellä luodaan yksi kontaktipiste, joka edellisillä kehyksillä muodostettujen kontaktipisteiden kanssa muodostaa kontaktimoniston. Täyden kontaktimoniston luonti esitellään alaluvussa 5.3.2 ja inkrementaalinen alaluvussa 5.2.4. Lisäksi alaluvussa 5.1 tullaan esittelemään algoritmeissa tarvittu matemaattinen tausta, sekä dynaamisille kappaleille TOI:n laskeminen esitellään alaluvussa 5.2.3. Tutkielmasta pois rajataan ylimääräisten kontaktipisteiden karsiminen kontaktimonistosta, joka yleensä tapahtuu fysiikan mallinuksen esikäsittely vaiheessa törmäystarkastelun jälkeen (Moravanszky ja Terdiman 2004). Eräs tapa kontaktipisteiden karsimiseen on esitetty artikkelissa (Moravanszky ja Terdiman 2004). Tutkielmassa esiteltävien algoritmien ohella muita tunnettuja kapean vaiheen algoritmeja, joita ei tässä tutkielmassa tulla käsitellään ovat muun muassa Lin-Canny algorit-

mi (Lin ja Canny 1991), V-Clip (B. Mirtich 1998), CD-Dual (Choi ym. 2005) ja XenoCollide (Snethen 2008).

Taulukko 4. Kapean vaiheen algoritmit tarkasteltavissa kirjastoissa

PhysX 3.4	Bullet Physics 2.87	ODE 0.15	SOLID 3.5
PCM (GJK) SAT	GJK + EPA SAT	Yksilölliset	GJK + EPA

ODE:n lisäksi myös muissa kirjastoissa yksilöllisiä törmäysten havaitsemisen metodeja tiettyjen kappaleiden välillä, kuten Bullet Physics fyssiikkamoottorin osalta esitetään taulukossa 5.

Taulukko 5. Käytetyt kapean vaiheen algoritmit törmäyskappaleiden välillä Bullet Physics 2.87 fysiikkamoottorissa ("Bullet Physics: Documentation" 2018)

	Laatikko	Pallo	Konveksi verho, lieriö, kartio, kapseli
Laatikko	laatikko-laatikko	pallo-laatikko	GJK
Pallo	pallo-laatikko	pallo-pallo	GJK
Konveksi verho, lieriö, kartio, kapseli	GJK	GJK	GJK tai SAT

5.1 Matemaattinen tausta

Kuten aikaisemminkin tutkielmassa todettiin, löytyy törmäystarkastelun juuret matematiikassa ja geometriassa. Kapeassa vaiheessa algoritmien voidaan yleisesti sanoa olevan muiden vaiheiden algoritmeihin verrattuna monimutkaisia ja tutkielmassa esiteltävien algoritmien ymmärtämiseksi tulee esitellä muutama matemaattinen käsite. Nämä käsitteet ovat simpleksit (engl. simplex), Voronoi-alueet, kantajafunktio, Gaussin kuvaus, sekä konfiguraatioavaruuden este (engl. configuration space obstacle, CSO) ja Minkowskin summa. Näistä käsitteistä simpleksi tullaan näkemään GJK-algoritmissa, CSO ja tuki kartoitus GJK ja SAT -algoritmeissa, sekä Gaussin kuvaus SAT-algoritmissa. Käsitteitä, joita tässä ei tulla esittelemään, mutta joita tullaan tarvitsemaan ja

jotka jäävät lukijalle itse ymmärrettäväksi, ovat affiini riippumattomuus, Cramerin sääntö, barysentriset koordinaatit ja konveksi kombinaatio, joka mainittiin jo luvussa 2.2.

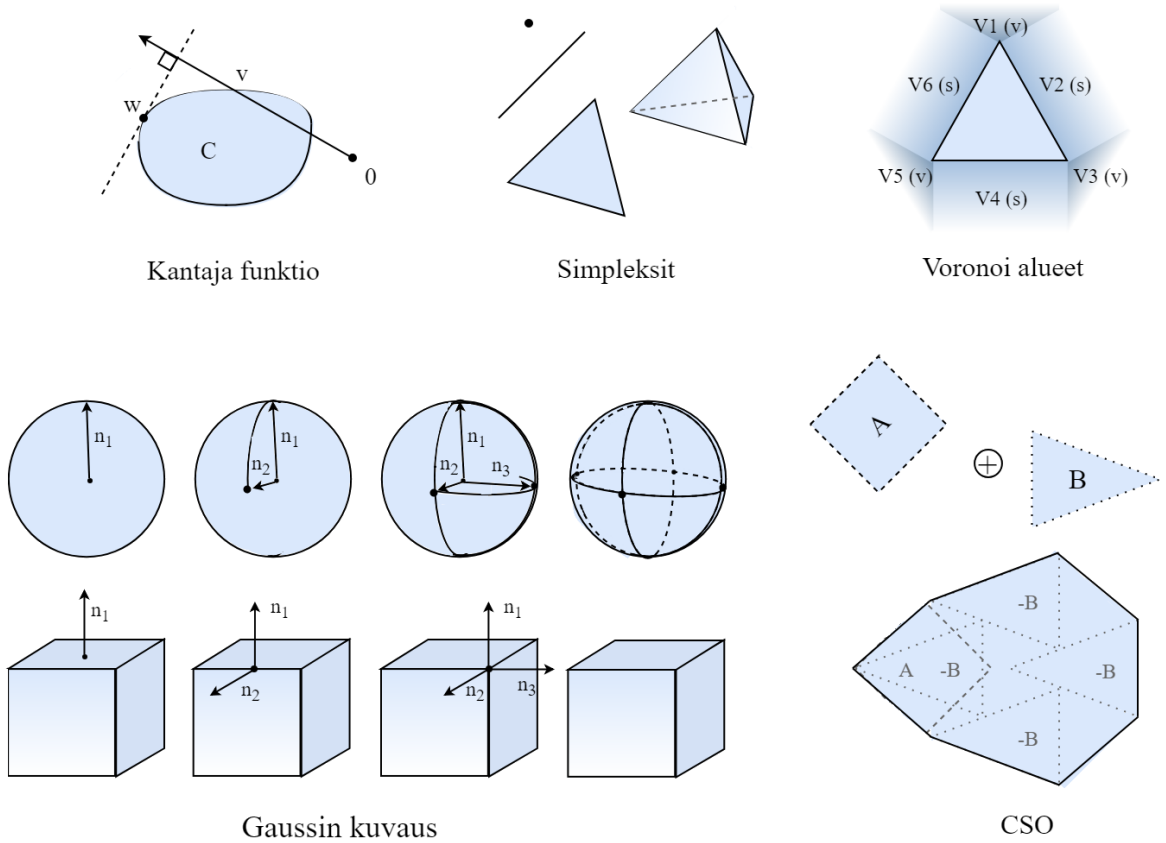
Aiemmin mainittiin GJK-algoritmin olevan simpleksi-perustainen. Simpleksin voidaan määritellä olevan kolmion yleistys mielivaltaisiin ulottuvuuksiin. Tarkempi matemaattinen määritelmä saadaan muodostettua luvussa 2.2 esitellyn polytoopin avulla. Kuten luvussa todettiin, voidaan polytooppi määritellä rajallisesta määrästä kärkiä koostuvaksi konveksiksi monitahokkaaksi. Jos polytoopin kärjet $\{b_0, b_1, ..., b_n\}$ ovat affiinisti riippumattomat, on kyseessä n-ulottuvuuden simpleksi eli n-simpleksi. Koska tutkielmassa tarkastellaan törmäyksiä kolmiulotteisessa avaruudessa, käytetään simpleksejä nollannesta ulottuvuudesta kolmanteen ulottuvuuteen asti. Nämä ovat: piste, suora, kolmio ja tetraedri vastaavassa järjestyksessä. Kuviossa 12 ylhäällä keskellä on esitelty edellä luetellut simpleksit. Kuviossa 12 ylhäällä oikealla on esitetty 2-simpleksin Voronoi-alueet. Kuviossa tumman sinisellä on merkitty kärkien Voronoi-alueet ja vaaleansinisellä puolestaan särmien Voronoi-alueet. Voronoi-alueet ovat avaruuden $\mathbb{R}^n$ osajoukkoja V^n. Voronoi-alueet määräytyvät simpleksin ominaisuuksien normaalien tai niiden viereisten ominaisuuksien normaalien mukaisesti. Esimerkiksi 2-simpleksin särmän määräämä Voronoi-alue määräytyy kyseisen särmän normaalin mukaisesti, kun taas 2-simpleksin kärjen Voronoi-alue määräytyy sen viereisten särmien normaalien mukaisesti.

Kapean vaiheen törmäystarkastelun kannalta pistejoukoille on yksi erityisen tärkeää operaatio: Minkowskin summa ja siitä johdettu operaatio, joka usein tunnetaan nimellä Minkowskin erotus. Minkowskin summa määritellään kahden konveksin joukon $A \oplus B$, avulla muodostetuksi konveksiksi joukoksi C, jolla pätee $C = A \oplus B = \{a + b : a \in A, b \in B\}$, missä $a + b$ on vektoreiden summa. Toisin kuin Minkowskin summa, Minkowskin erotus ei ollut Hermann Minkowskin löytämä ominaisuus pistejoukoille (Schneider 1993, s. 133). Itse termiä Minkowskin erotus näkee ristiriitaisesti käytettävän geometriassa ja törmäystarkastelussa. Geometriassa Minkowskin erotus määritellään Minkowskin summan $A \oplus B$ avulla $A \ominus B = (A^c \oplus B)^c$, missä A^c on joukon A komplementti (Schneider 1993, s. 137). Koska puolestaan muun muassa Ericssonin kirjassa (Ericson 2004, s. 71), kuten myös GJK:n alkuperäisessä artikkelissa (Gilbert, Johnson ja Keerthi 1988) Minkowskin erotus esitetään Minkowskin summan mukaan $A \ominus B = A \oplus (-B)$, pyritään sekaannuksen välttämiseksi tutkielmassa välttämään termiä Minkowskin erotus. Sama havainto

mainitaan myös artikkelin (G. v. d. Bergen 1999) alaviitteessä. Sen sijaan käytetään termiä konfiguraatioavaruuden este (CSO), joka saadaan muodostettua vastaavalla tavalla, eli kun suoritetaan Minkowskin summa kahdelle pistejoukolle A ja $-B$, missä $-B$ saadaan, kun jokainen pistejoukon B piste kerrotaan skalaarilla -1. Avaruuskappaleilla tämä voidaan mieltää yksinkertaisesti, siten että kappaleen A pintaa pyyhkäistään kappaleella $-B$. Tämä on esitelty kuviossa 12 alhaalla oikealla.

Se miksi törmäystarkastelussa ollaan kiinnostuneita Minkowskin summasta ja CSO:sta, johtaa siitä, että sillä on yksi tärkeä ominaisuus: kahden konveksin avaruuskappaleen muodostama CSO sisältää origon jos ja vain jos avaruuskappaleet leikkaavat toisensa. Tällöin kahden avaruuskappaleen leikkaavuuden määrittelemiseen, riittää tutkia sisältyykö origo kappaleiden CSO:hon. Tätä ominaisuutta hyödynnetään luvussa 5.2 esiteltävässä GJK-algoritmissa. Toinen tärkeä ominaisuus mitä tullaan tarvitsemaan on, että CSO:n tahkojen normaalit ovat sen muodostamien kappaleiden ainoat mahdolliset erottavat akselit. Tätä ominaisuutta puolestaan hyödynnetään SAT:n optimoinnissa, kuten luvussa 5.3.1 tullaan näkemään. Tässä vaiheessa on hyvä huomata, että CSO:ta ei yleensä muodosteta kokonaisuudessaan, vaan yleensä ollaan ainoastaan kiinnostuneita tietyssä suunnassa kaukaisimpana sijaitsevasta pisteestä. Tämä saadaan seuraavaksi esiteltävän kantajafunktion avulla.

Kantajafunktio (engl. suport mapping) s on matemaattinen funktio, joka palauttaa konveksin kappaleen A kaukaisimman pisteen suunnassa v. Toisin sanoen $s_A = max\{v \cdot a : a \in A\}$. Kahden konveksin kappaleen A ja B muodostaman CSO:n kaukaisin piste suunnassa v saadaan muotoon $s_{A-B}(v) = s_A(v) - s_B(-v)$. Yleisesti kantajafunktion palauttamaa pistettä kutsutaan kantajapisteeksi (engl. support point). Kattava määrä kantajafunktioita eri primitiiveille ja niiden yhdistelmille on lueteltu artikkelissa (Snethen 2008, s. 168-170), joita ei tässä tutkielmassa tarkemmin käydä läpi. Primitiivien ja niiden yhdistelmien lisäksi kantajafunktio pätee myös mielivaltaisille konvekseille verhoille. Tällöin hyödynnetään niin kutsuttua vuorikiipeily (engl. hill-climbing) metodia, jossa viereisiä ominaisuuksia pitkin kulkemalla voidaan konveksilla kappaleella löytää maksimiarvo tietyssä suunnassa, ilman että kaikkia kärkiä tai särmiä välttämättä tarvitsee käydä lävitse. Kuviossa 12 ylhäällä vasemmalla on esitetty kantaja funtio kaarevalle konveksille verholle.



Kuvio 12. Kantajafunktio, simpleksit S^n, missä $n = 0, ...3$, CSO ja Gaussin kuvaus.

Viimeinen käsite, joka tässä luvussa tullaan esittelemään on Gaussin kuvaus. Gaussin kuvaus on tapa esittää kolmiulotteisen avaruuden pintoja yksikköpallon S^2 muodossa. Kun puhutaan polytoopin ominaisuudesta, jonka vastaava ominaisuus esitetään yksikköpallolla, puhutaan Gaussin kuvauksen kuvantavan polytoopin ominaisuuden yksikköpallossa S^2. Jokainen polytoopin pinnan piste kuvantuu sen normaaliin. Tällöin polytoopin tahko kuvantuu pisteeksi Gaussin kuvauksessa. Polytoopin särmä puolestaan voidaan ilmaista sen viereisten tahkojen normaalien avulla, jolloin särmä kuvantuu kahta normaalia yhdistäväksi kaareksi (engl. great arch) yksikköpallolla. Vastaavalla periaatteella polytoopin kärki kuvantuu sen viereisten tahkojen normaalien rajaamaksi peitteeksi. Kuution eri ominaisuuksien kuvantuminen Gaussin kuvaksessa on esitetty kuviossa 12 alhaalla vasemmalla. Tulee myös huomata että Gaussin kuvauksen yksikköpallo ei ole kään-

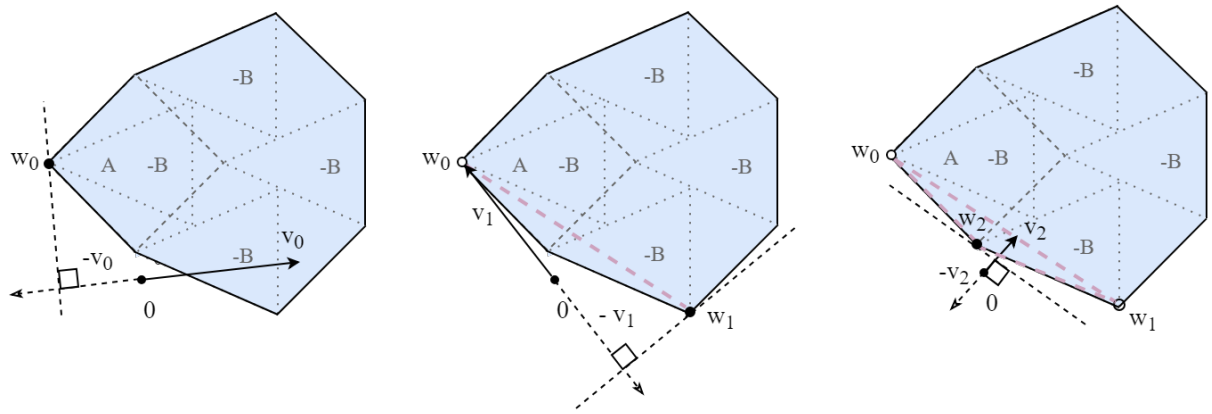
nettävissä yksittäiseksi polytoopiksi, vaan useampi erilainen polytooppi voi kuvantua samanlaiseksi yksikköpalloksi (Migdalskiy 2010). Tutkielmassa Gaussin kuvausta tullaan hyödyntämään mahdollisten erottavien akseleiden karsimiseksi luvussa 5.3.1 ja täyden kontaktimoniston muodostamiseen luvussa 5.3.2.

5.2 Gilbert-Johnson-Keerthi algoritmi

Gilbert-Johnson-Keerthi algoritmi eli lyhyesti GJK-algoritmi on yksi käytetyimpiä algoritmeja törmäysten havaitsemiseen kapeassa vaiheessa, mikä ilmenee myös taulukosta 4. Algoritmin suosio johtuu sen nopeudesta ja monikäyttöisyydestä (Montanari, Petrinic ja Barbieri 2017). Monikäyttöisyys puolestaan johtaa siitä, että geometrian lukeminen tapahtuu pelkästään luvussa 5.1 esitetyn kantajafunktion avulla. Algoritmin iän ja suosion myötä on siitä olemasta monia variantteja. Alun perin algoritmin esitteli Gilbert, Johnson ja Keerthi (1988) artikkelissaan, jolloin algoritmi kykeni laskemaan euklidisen etäisyyden kahden polytoopin välille. Myöhemmin Gilbert ja Foo (1990) esittivät muutoksia algoritmiin, jolla se saatiin yleistettyä kaikille konvekseille kappaleille. Algoritmista nopeamman inkrementaalisen version esittelivät Cameron (1997), joka kantaa nimeä Enhanced-GJK (E-GJK). E-GJK-algoritmilla lähes vakioaikainen suoritusaika, kun kehyskoherenssi on korkea (Cameron 1997). Tämän mahdollistaa vuorikiipeily-metodin käyttö, sekä edellisillä kehyksillä suoritettujen laskentojen uudelleenkäyttö. Edellä mainittujen optimointien lisäksi artikkelissa esitetään approksimaation tuottava menetelmä penetraatiosyvyyden laskemiseksi leikkaavilla kappaleilla. Tarkemmin kuitenkin tässä tutkielmassa tullaan esittelemään G. v. d. Bergen (1999) esittelemä numeerisesti vakaampi ja laskennallisesti nopeampi versio GJK-algoritmista. Mainittakoon myös se, että samassa artikkelissa (G. v. d. Bergen 1999) esitellään myös erottavan akselin testiä hyödyntävä Incremental Separating-axis GJK-algoritmi (ISA-GJK).

GJK-algoritmi on iteratiivinen algoritmi kahden konveksin kappaleen välisen etäisyyden laskemiseen. Algoritmin ajatuksena on muodostaa CSO:n $A - B$:n reunapisteistä simpleksi W siten, että W sisältää origon, jolloin kappaleet A ja B törmäävät. Jos puolestaan voidaan todeta ettei ole mahdollista muodostaa joukkoon $A - B$ sisältyvää origon sisältävää simpleksiä, eivät kappaleet A ja B törmää. Algoritmissa jokaisella iteraatiolla k muodostetaan joukkoon $A - B$ sisäl-

tyvä simpleksi W_k, joka on lähempänä origoa verrattuna edellisen iteraation simpleksiin W_{k-1}. Tarkemmin tämä tapahtuu siten, että aluksi $W_0 = \emptyset$ ja pisteeksi v_0:ksi valitaan mielivaltainen piste siten, että $v_0 \in A - B$:sta. Jokaisella iteraatiolla uusi kantajapiste $w_k = s_{A-B}(-v_k)$ lisätään simpleksiin W_k. Olkoon näin muodostettu uusi simpleksi $Y_k = W_k \cup \{w_k\}$ ja simpleksin Y konveksi verho $conv(Y)$. Samoin jokaisella iteraatiolla etsitään origoa lähimpänä oleva piste v_k siten, että $v_k \in conv(Y)$ ja simplekseksi W valitaan pienin Y:n osajoukko X siten, että $v \in conv(X)$. Algoritmin kolme ensimmäistä iteraatiota on esitelty kuviossa 13.



Kuvio 13. GJK-algoritmin kolme ensimmäistä iteraatiota.

Monitahokkaille GJK-algoritmi päättyy rajallisessa määrässä iteraatiota. Konvekseille kappaleille, jotka eivät kuulu monitahokkaisiin, ei algoritmi välttämättä pääty rajallisessa määrässä iteraatiota. Tällöin GJK päättyy heti kun v_k on riittävän lähellä $A - B$:n origoa lähimpänä olevaa pistettä. Jotta tätä kyetään arvioimaan, tulee origon ja $A - B$:n origoa lähimpänä olevan pisteen välisen etäisyyden yläraja ja alaraja tietää. Ylärajana voidaan käyttää sen hetkisen iteraation approksimaatiota kappaleiden välisestä etäisyydestä $\|v_k\|$ eli pisteen v_k etäisyyttä origosta. Alarajana puolestaan käytetään etumerkillistä etäisyyttä origosta kantaja tasolle pisteeseen w_k, joka on $\frac{v_k \cdot w_k}{\|v_k\|}$ (G. v. d. Bergen 1999). On hyvä huomata, että sekä alaraja ja yläraja molemmat riippuvat $\|v_k\|$:sta, jonka laskenta vaatii neliöjuuren laskennan, joten yleisemmin käytetään neliöllistä etäisyyttä eli kerrotaan molemmat sekä ylä- että alaraja $\|v_k\|$:lla. Näin saadaan pisteen v_k ja $A - B$:n origoa lähimpänä olevan pisteen välisen neliöllisen etäisyyden ylärajaksi $\|v_k\|^2 - v_k \cdot w_k$. Algoritmi päättyy, kun kyseinen neliöllisen etäisyyden yläraja laskee alle neliöön korotetun sallitun

mittapoikkeaman ε_{tol}:n. Mittapoikkeamana yleensä käytetään lukua luvun 10^{-8} ja kone epsilonin välillä, toisin sanoen $10^{-8} \leq \varepsilon_{tol} \leq 10^{-16}$ (Montanari, Petrinic ja Barbieri 2017). Pseudokoodi algoritmille on esitelty listauksessa 1.

Algoritmi 1 GJK-algoritmi (G. v. d. Bergen 2010)

1: $v \leftarrow$ mielivaltainen piste A-B:sta
2: $W \leftarrow \{\}$
3: $w \leftarrow s_{A-B}(-v)$
4: **while** $\|v_k\|^2 - v_k \cdot w_k > \varepsilon^2$ **do**
5: $\quad Y \leftarrow W_k \cup \{w_k\}$
6: $\quad v \leftarrow conv(Y)$:n lähimpänä origoa oleva piste
7: $\quad W \leftarrow$ pienin $X \subseteq Y$, missä $v \in conv(X)$
8: $\quad w \leftarrow s_{A-B}(-v)$
9: **end while**
10: **return** $\|v_k\|$

5.2.1 Etäisyys alialgoritmi

GJK-algoritmi nojautuu suuresti alialgoritmiin joka tunnetaan nimellä Johnssonin algoritmi. Listauksessa 1 origoa lähimpänä olevan simpleksin pisteen v:n ja pienimmän alisimpleksin X:n, jolla pätee $v \in X$, laskeminen tapahtuu Johnssonin alialgoritmissa. Useissa lähteissä mainitaan GJK-algoritmin numeerisesta epävakaudesta, joka johtaa juurensa Johnssonin algoritmissa tapahtuvista pyöristys virheistä. Tarkemmin tämä tapahtuu silloin kun Johnssonin algoritmi etsii lähintä pistettä niin kutsutusta degeneroituneesta simpleksistä, eli simpleksistä jonka kärjet ovat lähes affiinisti riippuvat (Gilbert, Johnson ja Keerthi 1988; G. v. d. Bergen 1999 Montanari, Petrinic ja Barbieri 2017). Alkuperäisessä GJK-algoritmissa (Gilbert, Johnson ja Keerthi 1988) käytetään varmistus proseduuria (engl. backup procedure) v:n ja X:n laskemiseen, kun Johnssonin algoritmi ei degeneroituneelle simpleksille kykene näitä laskemaan. Tarkemmin numeeriset ongelmat tuodaan esille artikkelissa (G. v. d. Bergen 1999). Artikkelissa G. v. d. Bergen (1999) esittelee laskennallisesti raskaan varmistus proseduurin korvaamisen uudella GJK-algoritmin iteraatioiden lopettamisehdolla. Kyseinen suhteelliseen virheeseen perustuva lopettamisehto nähtiin jo

listauksessa 1. Montanari, Petrinic ja Barbieri (2017) osoittavat ettei tämä kuitenkaan täysin kumoa GJK-algoritmin numeerisia epävakauksia ja ehdottavat uutta, numeerisesti vakaampaa ja laskennallisesti tehokkaampaa algoritmia, Signed Volumes (SV) algoritmia Johnssonin algoritmin tilalle. Tässä luvussa tullaan ensin esittelemään Johnssonin algoritmi ja sitten sen korvaava SV algoritmi lyhyesti.

Lyhyesti määriteltynä Johnssonin algoritmi tutkii rekursiivisesti kaikki simpleksin Y alisimpleksit X, kunnes origon sisältämä alisimpleksin määrittelemä Voronoi-alue löytyy, jolloin lasketaan origoa lähimpänä sijaitsevan pisteen $v \in aff(X)$ barysentriset koordinaatit λ_i alisimpleksin kärkien suhteen. Täsmällisemmin Johnssonin algoritmi voidaan määritellä seuraavasti. Olkoon $Y = \{y_1 ..., y_{n+1}\}$ joukko simpleksin kärkiä. Tällöin jokainen simpleksin pinnalla sijaitseva piste x voidaan esittää Y:n sisältämien pisteiden konveksi kombinaationa. Tämä johtaa Carathéodoryn lauseesta (engl. Carathéodory's theorem) (Rockafellar 1997, s. 153), jonka mukaan konveksilla verholla sijaitseva piste $x \in conv(Y) \subset \mathbb{R}^n$ voidaan ilmaista enintään $n+1$ konveksin verhon pisteiden konveksi kombinaationa. Täten barysentrisiä koordinaatteja käyttäen konveksi kombinaation määritelmän mukaisesti piste x voidaan ilmaista muodossa $x = \sum_{i=1}^{n+1} \lambda_i y_i$, missä $\sum_{i=1}^{n+1} \lambda_i = 1$ ja $\lambda_i \geq 0$ (Rockafellar 1997, s. 153-154). Johnssonin algoritmi laskee edellä kuvatun ehdon mukaiset barysentriset koordinaatit pisteelle v , sekä pienimmän alijoukon $X \subseteq Y$, jonka pisteiden konveksi kombinaationa voidaan konveksin verhon origoa lähimpänä oleva piste v ilmaista. Pienin alijoukko $X \subseteq Y$, saadaan kun hylätään kaikki kärjet y_i, joilla pätee $\lambda_i = 0$ (G. v. d. Bergen 2004a, s. 126).

Tällöin joukkoa X voidaan luonnehtia siten, että siihen kuuluvien kärkien barysentriset koordinaatit ovat ei-negatiivisia ja puolestaan niiden kärkien, jotka kuuluvat joukkoon Y, mutta eivät joukkoon X, niin ovat näiden kärkien barysentriset koordinaatit nolla tai pienemmät. Toisin sanoen joukon X tulee täyttää ehdot (Montanari, Petrinic ja Barbieri 2017):

$$\lambda_j > 0 \; ja \; \lambda_i \; \leq 0 \; \forall \; j \in X : i = 1, ..., n+1, i \neq j \tag{5.1}$$

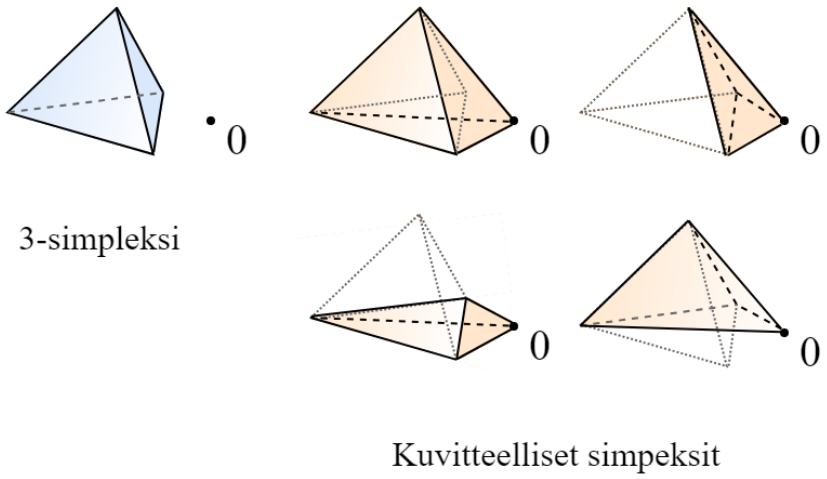
Tämä takaa sen, että origoa lähinpänä oleva piste sijaitsee X:n affiinilla verholla eli $v \in aff(X)$ (G. v. d. Bergen 2004a, s. 126). Geometrisesti tämä tarkoittaa, että vektori v on kohtisuorassa simpleksin ominaisuutta kohtaan (Montanari, Petrinic ja Barbieri 2017). Johnssonin algoritmi

tutkii rekursiivisesti jokaisen simpleksin Voronoi-alueet, kunnes edellä mainitut ehdot täyttävä Voronoi-alue löytyy. Olkoon näin saatu joukko $X = \{x_1, ..., x_r\}$ simpleksin Y alijoukko. Tällöin piste $v \in aff(X)$ on lähimpänä origoa, jos ja vain jos vektori v on kohtisuorassa $aff(X)$ vasten eli $(x_i - x_1) \cdot v = 0$ kaikilla $i = 2, ..., r$ (G. v. d. Bergen 2004a, s. 127). Tällöin barysentriset koordinaatit voidaan ratkaista yhtälöryhmästä $A\lambda = b$:

$$\begin{bmatrix} 1 & ... & 1 \\ (x_2 - x_1) \cdot x_1 & ... & (x_2 - x_1) \cdot x_r \\ ... & ... & ... \\ (x_r - x_1) \cdot x_1 & ... & (x_r - x_1) \cdot x_r \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ ... \\ \lambda_r \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ ... \\ 0 \end{bmatrix} \tag{5.2}$$

Johnssonin algoritmissa yhtälöryhmä ratkaistaan Cramerin sääntöä ja kofaktori kehitelmää (engl. cofactor expansion) käyttämällä. Kofaktori kehitelmälle, mikä myös Laplacen kehitelmänä tunnetaan, voidaan antaa seuraavanlainen määritelmä. Olkoon A $(n \times n)$-matriisi. Tällöin matriisin A kofaktorit ovat $C_{ij} = (-1)^{i+j} A_{ij}$, missä A_{ij} on $((n-1) \times (n-1))$-alimatriisi, joka saadaan poistamalla i:s rivi ja j:s sarake matriisista A (Gruber 2013, s. 20). Koska $\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} A_{ij}$, missä $1 \leq i \leq n$ (Gruber 2013, s. 20), saadaan yhtälö kofaktoreita käyttäen muotoon $\det(A) = \sum_{j=1}^{n} a_{ij} C_{ij}$, josta käytetään nimitystä kofaktori kehitelmä. Edellä määriteltyä kofaktori kehitelmää hyödyntäen saadaan Cramerin sääntöä soveltamalla yhtälöryhmän 5.2 ratkaisu muotoon $\lambda_j = \frac{-1^{1+j} detA_{1j}}{detA}$ (Montanari, Petrinic ja Barbieri 2017). Kun merkitään simpleksin muodostavaa pisteiden joukkoa W, voidaan edellinen ratkaisu johtaa rekursiiviseen muotoon. Rekursiivinen yhtälöryhmän 5.2 ratkaisu on tällöin $\lambda_j = \frac{\Delta_j(W)}{\sum_{j \in k} \Delta_j(W)}$, missä $\Delta_j(W)$ on A:n kofaktori yhdelle mahdollisista $(2^{n+1} - 1)$ simpleksin alisimplekseista (Montanari, Petrinic ja Barbieri 2017). Johnssonin algoritmi laskee $\Delta_j(W)$ kasvattaen W:n kardinaalisuutta (Montanari, Petrinic ja Barbieri 2017). Toisin sanoen Johnssonin algoritmi etsii origoa Voronoi-alueilta alhaalta ylöspäin eli aloittaen varsinaisen simpleksin matalaulotteisimmasta simpleksistä kohti varsinaista simpleksiä.

Kuten luvun alussa todettiin, epäonnistuu edellä kuvattu Johnssonin algoritmi v:n ja X:n laskemisessa, kun X:n kärjet ovat lähes affiinisti riippuvat. Tarkemmin tämä tapahtuu kun $detA$ on suuruudeltaan lähellä laskenta tarkkuutta eli kone epsilonia (Montanari, Petrinic ja Barbieri 2017). Montanari, Petrinic ja Barbieri (2017) suorittamien kokeiden perusteella tulivat he siihen loppu-



Kuvio 14. 3-simpleksin kuvitteelliset simpleksit.

tulokseen etteivät numeeriset epävakaudet johdu degeneroituneesta simpleksistä itsestään, vaan yhtälöryhmään 5.2 sisällytetystä kohtisuoruuden ehdosta 5.1. Tästä syystä Montanari, Petrinic ja Barbieri (2017) ehdottavat algoritmia, joka sen sijaan, että ratkaisee yhtälöryhmän jokaiselle alijoukolle X ja testaa täyttääkö ratkaisu yhtälössä 5.1 esitetyt kohtisuoruuden ehdot, tunnistaa algoritmi uniikin joukon pisteitä, joille kohtisuoruuden ehdot pätevät ja vasta tämän jälkeen ratkaisee yhtälöryhmästä barysentriset koordinaatit. Kyseisestä algoritmista käytetään nimeä SV algoritmi. Artikkelissa (Montanari, Petrinic ja Barbieri 2017) mainitaan metodin vakauden johtuvan siitä, että laskenta suoritetaan sellaisessa aliavaruudessa, jossa barysentristen koordinaattien laskeminen on “turvallisempaa”. Turvallisemmalla tarkoitetaan sitä, että projisointi suoritetaan niille Cartesian tasoille ja akseleille, joilla simpleksin pinta-ala ja pituus eli samoin tilavuusmuodot saavat suurimmat arvonsa. Tällöin oletetaan barysentristen koordinaattien säilyvän muuttumattomia affiinissa muunnoksessa, jolloin ne kyetään laskemaan r-ulotteisessa avaruudessa, missä $r \leq n$, kun simpleksi ja CSO ovat määritelty n ulotteisessa avaruudessa.

SV algoritmi perustuu käsitteeseen, joka kantaa nimeä tilavuusmuoto (engl. volume form). Metodi arvioi varsinaisen simpleksin, sekä siitä johdettujen kuvitteellisten simpleksien (engl. fictitious simplex) tilavuusmuodot $\mu(W)$ (Montanari, Petrinic ja Barbieri 2017). Simpleksin tilavuusmuoto on sen pituuden, pinta-alan tai tilavuuden etumerkillinen arvo (Montanari, Petrinic

ja Barbieri 2017). Kuvitteellinen simpleksi puolestaan on simpleksi, joka saadaan muodostettua korvaamalla yksi varsinaisen simpleksin kärjistä origolla. Kolmiulotteisen simpleksin kuvitteelliset simpleksit on esitetty kuviossa 14. SV algoritmi luottaa yksinkertaiseen matriisiin M, jonka determinantti on verrannollinen simpleksin tilavuusmuotoon eli toisin sanoen $detM = r!\mu(W)$. Montanari, Petrinic ja Barbieri (2017) havainnoivat, että $\mu(W)$ ja $detM$:llä on samat etumerkit, ja koska ollaan pelkästään kiinnostuneita etumerkeistä, ei tilavuusmuotoja tarvitse tarkkaan laskea. Metodi tapahtuu kolmessa vaiheessa:

1. Projisoidaan kaikki simpleksin kärjet Y alempi ulotteiseen avaruuteen.
2. Hylätään kaikki simpleksin kärjet, joita ei tarvita v:n ilmaisemiseen barysentristen koordinaattien avulla.
3. Ratkaistaan $M\lambda = p$ barysentrisille koordinaateilla λ.

Algoritmin toiminnan esittelemiseksi, määritellään funktio *VertaaEtumerkkeja* 5.3, joka palauttaa totuusarvon, sille ovatko kahden luvun etumerkit samat.

$$VertaaEtumerkkeja(a,b) = \begin{cases} 1 & jos \text{ a} > 0 \text{ , b} > 0 \\ 1 & jos \text{ a} < 0 \text{ , b} < 0 \\ 0 & muulloin \end{cases} \tag{5.3}$$

Tämä lisäksi esitellään matriisi M 5.4, joka Carathéodoryn lauseeseen nojalla koostetaan $r+1$ pisteestä, missä r on supistetun avaruuden ulottuvuus ja s:llä merkitään on simpleksin kärkiä ja l:llä simpleksin l:stä koordinaattia.

$$\begin{bmatrix} s_1^l & \dots & s_{r+1}^l \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \dots \\ \lambda_r + 1 \end{bmatrix} = \begin{bmatrix} p^l \\ \dots \\ 1 \end{bmatrix} \tag{5.4}$$

SV algoritmi voidaan jakaa kolmeen alialgoritmiin. Eri alialgoritmeissa käsitellään tietyn ulottuvuuden simpleksiä, kolmiulotteisesta simpleksistä yksiulotteiseen. Kutsutaan näitä alialgoritmeja vastaavasti S3D, S2D ja S1D. Pääalgoritmi rekursiivisesti kutsuu alialgoritmeja ylhäältä

alas, eli Voronoi-alueita tutkitaan moniulotteisemmasta alempi ulotteiseen, mikä on päinvastoin, kuin mitä Johnssonin algoritmissa nähtiin. Samoin toisin kuin Johnssonin algoritmissa, ei kaikkia alisimpleksejä tarvitse käydä läpi, vaan rekursio lopetetaan heti oikean alisimpleksin löydyttyä. Seuraavaksi esitellään lyhyesti alialgoritmit S3D, S2D ja S1D. Tarkemmin ne esitellään artikkelissa (Montanari, Petrinic ja Barbieri 2017), jossa esitellään myös pseudokoodi kyseisille algoritmeille.

S3D algoritmissa suoritetaan aikaisemmin mainituista algoritmin vaiheista vaiheet (2) ja (3). Kärkien hylkäämiseksi lasketaan $detM$, joka kofaktori kehitelmää hyödyntäen saadaan muotoon $\det M = \sum_{j=4 j=1,\ldots,4} (-1)^{i+j} M_{i,j} = C_{4,1}C_{4,2}C_{4,3}C_{4,4}$, missä $M_{i,j}$:llä merkitään matriisin M determinanttia, kun matriisista ensin on poistettu i:nnes rivi ja j:nnes ja sarake. Kofaktorit $C_{4,j}$ ovat verrannollisia simpleksin W:n kuvitteellisten simpleksien tilavuusmuotoihin. Kärkien hylkääminen perustuu simpleksin tilavuusmuodon ja kuvitteellisten simpleksien tilavuusmuotojen etumerkkien vertailemiseen käyttäen *VertaaEtumerkkeja* funktiota 5.3. Jos kaikki etumerkit ovat samoja, sisältyy origo tällöin simpleksiin, jolloin lasketaan barysentriset koordinaatit kyseiselle 3-simpleksille. Barysentriset koordinaatit saadaan yksinkertaisesti kaavalla $\lambda_j = C_4, j/detM$ kaikilla j. Jos taas kaikki etumerkit eivät ole samoja, kutsutaan *VertaaEtumerkkeja* funktiota, siten että $j = \{2,3,4\}$. Tällöin j:nnes kärki voidaan poistaa, jos *VertaaEtumerkkej* funktio palauttaa nolla. Tällöin jatketaan alialgoritmiin S2D. Jos edellä mainittu ehto toteutuu useammalla j:llä, tutkitaan S2D alialgoritmissa rekursiivisesti kaikki kärki j poistamalla saadut alisimpleksit ja lopulta palautetaan näistä se, jolla lähimmän pisteen etäisyys origosta on pienin.

Jos origo ei sisältynyt 3-simpleksiin, tullaan rekursiossa S2D alialgoritmiin. S2D alialgoritmi alkaa suorittamalla vaiheen (1) eli projisoimalla kaikki simpleksin pisteet, sekä origo, jota tarvitaan kuvitteellisten simpleksien määrittelemiseen, alempi ulotteiseen avaruuteen. Tätä varten täytyy ensin projisoida origo O simpleksin affiinille verholle, jolloin saadaan piste p_O. Tämän jälkeen suoritetaan 2-simpleksin ja origon projisointi kaikille kolmelle Cartesian tasolle. Näistä projisoinneista valitaan se, jossa projisoidun simpleksin pinta-ala ja samalla tilavuus muoto ovat suurimmat. Pinta-alojen vertailu suoritetaan matriisin M minoreiden $M_{i,j}$ determinanttien välillä, jossa i on projisoinnissa poistettava koordinaatti ja j puolestaan sen simpleksin kärjen vastaava sarake, joka pudotettiin pois S3D algoritmissa. Merkitään näiden minoreiden determi-

nanteista suurinta μ_max. Tämän jälkeen suoritetaan vaihe (2) vertaamalla μ_max ja kuvitteellisten simpleksien tilavuusmuotojen etumerkkejä käyttäen *VertaaEtumerkkeja* funktiota. Samoin kuin S3D algoritmissa, jos kaikki etumerkit ovat samoja, sisältyy origo kyseiseen 2-simpleksiin, jolloin suoritetaan barysentristen koordinaattien laskenta vastaavalla tavalla kuin S3D algoritmissa. Jos kaikki etumerkit eivät puolestaan ole samoja, jatketaan rekursiossa alialgoritmiin S1D näiden osalta. Tässä vaiheessa, tai tarkemmin jo alialgoritmin alkuvaiheessa käsitellään degeneroituneet eli affiinisti tai lähes affiinisti riippuvat simpleksit siten, että origon projisoinnissa p_O laskemisessa, syntyy NaN arvo, joka syötetään koordinaatin poiston jälkeen *VertaaEtumerkkeja* funktioon, jossa se palauttaa aina nolla. Tällöin edetään S1D alialgoritmiin kaikkien mahdollisten 1-simpleksien osalta.

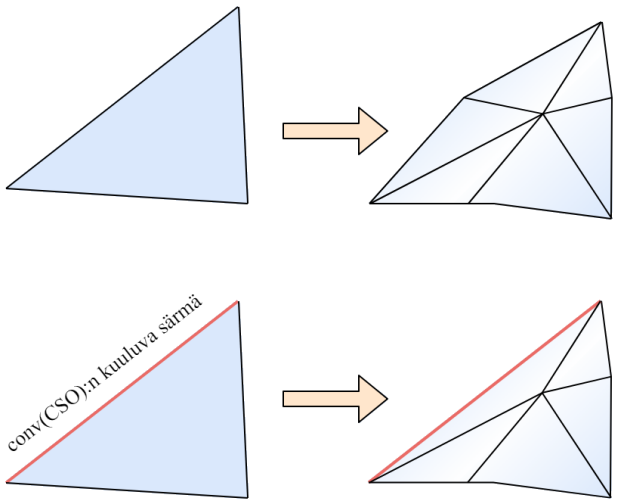
Jos origo ei sisältynyt 2-simpleksiin, tullaan rekursiossa lopulta S1D alialgoritmiin. Samoin kuin S2D algoritmissa, suoritetaan origon projisointi simpleksin affiinille verholle. Alempaan ulottuvuuteen siirtyminen tapahtuu projisoimalla ja etsimällä Cartesian akseli, jolla projisoituna pituus on suurin. Barysentristen koordinaattien laskeminen suoritetaan samoin kuin S2D algoritmissa, mikäli simpleksin tilavuusmuodon ja kuvitteellisten simpleksien tilavuusmuotojen etumerkit ovat samoja. Muulloin palautetaan 0-simpleksi, jolle lähimmän pisteen barysentrinen koordinaatti on yksi. Montanari, Petrinic ja Barbieri (2017) suorittamissa kokeissa vertailtiin SV algoritmin ja Johnssonin algoritmin numeraalista tarkkuutta, sekä kolmen etäisyys algoritmin CPU aikaa. Ensimmäisessä kokeessa, jossa vertailtiin SV algoritmin ja Johnssonin algoritmien numeraalista tarkkuutta, osoittautui SV algoritmin kykenevän laskemaan minimi etäisyyden, joka on suuruudeltaan lähes samaa luokkaa kuin kone epsilon ε. Johnssonin algoritmi puolestaan onnistuu tarkimmillaan minimi etäisyyden laskennassa, kun minimi etäisyys on minimissään luokkaa $\varepsilon^{1/2}$ (Montanari, Petrinic ja Barbieri 2017). CPU-aikaa mittaavassa kokeessa puolestaan, keskenään vertailtiin SV algoritmia, Johnssonin algoritmia varmistus proseduurin kanssa, sekä varmistus proseduuria ilman Johnssonin algoritmia. Kokeista selvisi SV algoritmin suoriutuvan muita algoritmeja noin 15% nopeammin (Montanari, Petrinic ja Barbieri 2017).

5.2.2 Expanding polytope algoritmi

Kuten luvun 5 alussa nähtiin, sisältää kontaktipiste penetraatiosyvyyden. Yksi iteratiivinen algoritmi laskea penetraatiosyvyys konvekseille kappaleille esitellään artikkelissa (G. v. d. Bergen 2001). Algoritmi tunnetaan nimellä Expanding polytope algorithm (EPA). Aikaisemmin metodi penetraatiosyvyyden laskemiseen esiteltiin artikkelissa (Cameron 1997). Cameron (1997) metodista poiketen EPA laskee tarkan penetraatiosyvyyden polytoopeille, sekä ei-polytoopeille penetraatiosyvyyden tietyn toleranssin mukaisesti (G. v. d. Bergen 2004a, s. 148). Samoin kuin Cameron (1997) esittelemässä metodissa, myös EPA käyttää penetraatiosyvyyden laskentaan simpleksiä GJK-algoritmista (G. v. d. Bergen 2004a, s. 147). Sen lisäksi, että EPA hyödyntää origon sisältävää simpleksiä, mikä saadaan GJK-algoritmista, käyttää se myös kantajafunktioita kappaleiden kuvaamiseen samoin kuin GJK-algoritmissa, jolloin se kykenee löytämään penetraatiosyvyyden samoille kappaleille kuin GJK-algoritmi kykenee havaitsemaan törmäyksen. Kuitenkin on hyvä huomata, että EPA ei ole täysin sidottu GJK-algoritmiin, mutta koska sitä yleensä käytetään GJK-algoritmin jatkeena, kuten SOLID 3 -kirjastossa sekä Bullet Physics SDK:ssa, esitellään se tässä tutkielmassa GJK-algoritmin alalukuna.

EPA:ssa penetraatiosyvyys määritellään $A - B$ konveksin verhon pisteeksi joka on lähimpänä origoa. Sananmukaisesti penetraatiosyvyys joka tunnetaan myös nimellä minimal translational distance (MTD) (Migdalskiy 2010) on origosta kyseiseen pisteeseen muodostetun vektorin pituus. Kuten jo todettiin, tarvitsee EPA syötteekseen origon sisältävän simpleksin. Koska GJK-algoritmi voi palauttaa myös tetraedrin sijasta matalamman tason simpleksin: pisteen, janan tai kolmion (G. v. d. Bergen 2004a, s. 159), laajennetaan matalamman tason simpleksiä lisäämällä siihen pisteitä törmäyskappaleen konveksilta verholta. Esimerkiksi, jos GJK-algoritmi palauttaa origon sisältävän kolmion, laajennetaan kolmiota kantajafunktion s avulla siten, että lisätään kolmioon kaksi pistettä: $s_{A-B}(n)$ ja $s_{A-B}(-n)$, missä n on kolmion normaali. Kolmiota alemmat simpleksit ovat poikkeustapauksia, eikä niitä tässä tutkielmassa oteta huomioon. Näiden osalta penetraatiosyvyyden laskeminen esitellään artikkelissa (G. v. d. Bergen 2001). Seuraavaksi esitellään lyhyesti EPA:n toiminta joka toimii lähes vastaavalla tavalla kuin tutkielmassa aiemmin mainittu algoritmi Quickhull (Barber, Dobkin ja Huhdanpaa 1996).

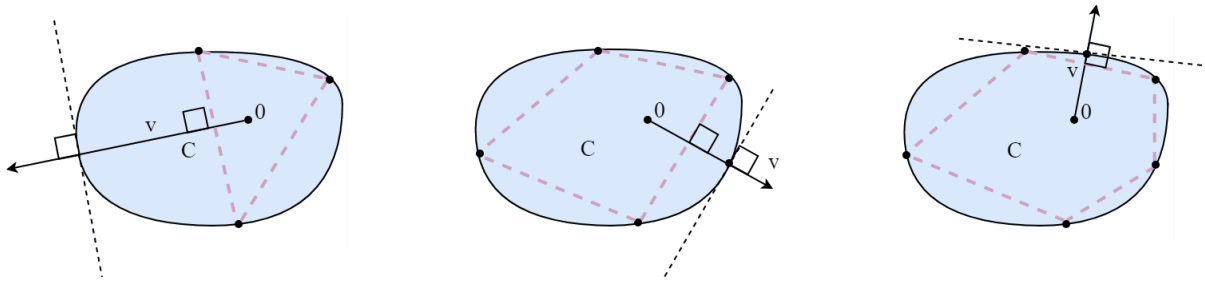
Kun meillä on origon sisältävä tilavuus eli monitahokas, lasketaan jokaisella iteraatiolla lähimpänä origoa piste v, siten että piste v sisältyy laajennettavan monitahokkaan konveksiin verhoon ja tarkemmin se sisältyy origoa lähimpänä olevan tahkon eli kolmion ja samalla simpleksin (G. v. d. Bergen 2001) affiiniin verhoon. Kantajafunktion avulla jokaisella iteraatiolla laajennettavaa monitahokkaaseen lisätään uusi kärki $A-B$:n konveksilta verholta. Kantajafunktion suuntavektorin käytetään origon ja pisteen v muodostamaa vektoria. Uuden kärjen lisääminen monitahokkaaseen vaatii sen tahkon jakamisen, jolla piste v sijaitsee, jakamisen. Tahkon jakaminen tapahtuu laskemalla särmällä e sijaitseva origoa lähimpänä oleva piste v_e jokaiselle tahkon särmälle. Särmät jaetaan lisäämällä jokaiselle särmälle kantajafunktion avulla piste $w_e = s_{A-B}(v_e)$, lukuun ottamatta särmiä, jotka kuuluvat $A-B$:n konveksiin verhoon. Särmä e kuuluu $A-B$:n konveksiin verhoon jos ja vain jos $v_e \cdot w_e = \|v_k\|^2$ (G. v. d. Bergen 2001). Tahkon jakaminen osiin on esittety kuviossa 15.



Kuvio 15. Tahkon jakaminen (G. v. d. Bergen 2001).

Piste v puolestaan voidaan ratkaista seuraavasti. Koska piste v:n tulee sijaita simpleksin affiinilla verholla (G. v. d. Bergen 2001) voidaan kyseisen pisteen barysentriset koordinaatit λ ratkaista

luvussa 5.2.1 esitetyn yhtälöryhmän avulla. Näin muodostetun yhtälöryhmän $A(\lambda_i) = b$ ratkaisu saadaan laskemalla käänteismatriisi A^{-1}, jolloin ratkaisuksi saadaan $\lambda_i = A^{-1}b$ (G. v. d. Bergen 2001). Kuviossa 16 on esitettynä EPA:n kolme ensimmäistä iteraatiota implisiittisesti määrätyllä CSO:lla. Monitahokkailla itaraatioita jatketaan kunnes origoa lähimpänä oleva tahko on myös $A - B$:n ja implisiittisesti määrättävillä muodoilla iterointia jatketaan kunnes virhe nykyisessä penetraatiosyvyyden arviossa v laskee alle annetun toleranssin(G. v. d. Bergen 2001).



Kuvio 16. EPA:n kolme ensimmäistä iteraatiota.

5.2.3 Jatkuva törmäysten havaitseminen GJK

Jatkuvaan törmäystan havaitsemiseen kapeassa vaiheessa on olemassa monia metodeja (Catto 2013). Metodeja yhdistää se, että tarkoitus on löytää ensimmäisen osuman ajankohta. Ensimmäisen osuman ajankohta itsessään on funktion nollakohdan etsimisongelma, kun funktiona on kahden kappaleen välinen etäisyys ajan suhteen (B. V. Mirtich 1996, s. 28-37). Eräs menetelmä TOI:n löytämiselle esitellään väitökirjassa (B. V. Mirtich 1996, s. 28-37). Menetelmä kantaa nimeä Conservative Advancement (CA). CA menetelmässä nollakohtaa lähestytään varovasti pienin aika-askelin positiivisten arvojen puolelta. CA menetelmän ja GJK-algoritmin yhdistelmä metodi esitellään artikkelissa (G. v. d. Bergen 2004b). Yhdistelmäalgoritmissa pienet muutokset GJK-algoritmiin mahdollistavat algoritmin hyödyntämiseen jatkuvassa törmäystarkastelussa. Pseudokoodi algoritmille esitelty listauksessa 2.

Algoritmi 2 GJK-algoritmi: jatkuva törmäystarkastelu (G. v. d. Bergen 2010).

$t \leftarrow 0$
$s \leftarrow 0$
$n \leftarrow 0$
$v \leftarrow$ mielivaltainen piste A-B:sta
$W \leftarrow \{\}$
while $\|v\|^2 > \varepsilon^2$ **do**
    $p \leftarrow s_{A-B}(-v)$
    if $v \cdot p > tv \cdot r$ **then**
        if $v \cdot r > 0$ **then**
            $t \leftarrow \frac{v \cdot p}{v \cdot r}$
            if $t > 0$ **then**
                return false
            end if
            $s \leftarrow tr$
            $W \leftarrow \{\}$
            $n \leftarrow -v$
        else
            return false
        end if
    end if
    $Y \leftarrow W \cup \{p - s\}$
    $v \leftarrow conv(Y)$:n lähimpänä origoa oleva piste
    $W \leftarrow$ pienin $X \subseteq Y$, missä $v \in conv(X)$
end while
return true

Algoritmin perusajatus on ampua kahden kappaleen relatiivisen translaation mukainen äärellinen säde origosta kohti näiden kahden kappaleen muodostamaa CSO:ta. Jokaisella iteraatiolla sädettä lyhennetään sitä mukaan, kun voidaan todistaa ettei lyhennettävä osa säteestä sisällä osumakoh-

taa CSO:n pinnalla. Näin saadaan uusi yläraja kappaleiden väliselle etäisyydelle. GJK-algoritmia ajatellen tämä voidaan nähdä origon siirtämisellä pitkin sädettä kohti CSO:ta. Samoin kuin jokaisella iteraatiolla lyhennetään sädettä, suoritetaan jokaisella iteraatiolla myös GJK-algoritmin iteraatio. Yhdistelmäalgoritmia toistetaan, kunnes sädettä pitkin ollaan edetty riittävän lähelle CSO:ta tai voidaan osoittaa, ettei säde leikkaa CSO:ta.

Täsmällisemmin yhdistelmäalgoritmi voidaan esitellä seuraavasti. Merkitään aikaa $t \in [0,1]$. Tällöin kappaleiden A ja B siirtymä kahden eri ajanhetken t välillä on $A+tc_A$ ja $B+tc_B$, missä c_A ja c_B on kappaleiden A ja B muunnokset vastaavasti. Säteenä käytetään CSO:n $A-B$ muodostavien kappaleiden A:n ja B:n relatiivista muutosta $r = c_B - c_A$. Tällöin ensimmäisen osuman ajankohta on pienin t, jolla pätee $tr \in A-B$. Merkitään osuman ensimmäistä ajankohtaa t_{hit} ja osumakohtaa $t_{hit}r$. Jos origo ei sisälly $A-B$:hen, jolloin $t_{hit} > 0$, niin silloin osumakohta täytyy sijaita $A-B$:n konveksilla verholla. Tällöin osumakohdalle on olemassa aito normaali. Nollasta eroava vektori v ja piste $p \in conv(A-B)$ määrittelevät kantavan tason, jos $v \cdot p = max\{v \cdot x : x \in A-B\}$ (G. v. d. Bergen 2004b). Kuten luvussa 2.7 todettiin, on kantava taso olemassa jokaiselle epätyhjän konveksin joukon reunapisteelle. Kantavalle tasolle, jolla normaali v ja piste $p \in conv(A-B)$, tiedetään että kaikille pisteille $x \in A-B$, joille pätee $v \cdot x > v \cdot p$, eivät sisälly $A-B$:hen (G. v. d. Bergen 2004b). Olkoon $x = tr$ piste säteellä. Tällöin $tv \cdot r > v \cdot p$ osa säteestä ei sisälly $A-B$:hen, eikä se siten voi sisältää osuma kohtaa (G. v. d. Bergen 2004b). Säteen ja kantaja tason leikkauskohta saadaan yhtälöstä $t_{clip} = \frac{v \cdot p}{v \cdot r}$. Tästä seuraa, että jos $v \cdot p > tv \cdot r$ pätee, säde joko leikataan tai hylätään kokonaan. Iteraatioita jatketaan niin kauan edellä mainittu ehto on voimassa tai voidaan todeta ettei säde leikkaa $A-B$:tä, eli $t > 1$ tai ollaan edetty riittävän lähelle $A-B$:ta, jolloin pätee $\|v\|^2 \leq \varepsilon^2$, missä ε on toleranssi osumakohdan absoluuttiselle virheelle.

Lopetusehdon lisäksi toinen muutos, mikä perinteiseen GJK-algoritmiin täytyy tehdä on, että nykyiseen simpleksiin tulee lisätä piste $p-s$ pisteen p sijasta (G. v. d. Bergen 2004b). Jos algoritmi päättyy törmäyksen havaitsemiseen, sisältää tällöin muuttuja t TOI:n, s osumakohdan ja v osumakohdan normaalin. Osuma kohdan normaalina käytetään viimeisimmän kantavan tason normaalia, joka lyhentää säteen. Jos taas havaitaan ettei törmäystä tapahdu, on v erottava akseli, jota voidaan korkean kehyskoherenssin ympäristössä hyödyntää seuraavalla kehyksellä (G. v. d. Bergen 2004b).

Catto (2013) tuo esille CA menetelmiin perustuvien algoritmien, kuten edellä esitetyn yhdistelmäalgoritmin huonon puolen. CA menetelmän perustuvien algoritmien huonona puolena Catto (2013) pitää sitä, että niissä nollakohtaa etsitään ainoastaan positiivisen arvojen puolelta, eikä etäisyyden milloinkaan anneta mennä negatiiviseksi, jolloin ei kyetä hyödyntämään tehokkaampia nollakohdan etsintä menetelmiä (Catto 2013). Joissakin tilanteissa tämä voi ilmetä, siten että yhdistelmäalgoritmi voi vaatia jopa satoja GJK kutsuja TOI:n löytämiseksi yhden kehyksen aikana (Catto 2013). Edellä mainittujen tilanteiden välttämiseksi Catto (2013) esittelee metodin, Bilateral Advancement, joka etsii TOI:ta etäisyys funktion molemmilta puolilta. Nollakohdan löytämiseksi menetelmä yhdistää bisektio- ja Regula falsi menetelmät. Bisektio-menetelmässä tarkastelu puolitetaan, siten että uudeksi tarkasteluväliksi valitaan se, jonka tiedetään sisältävän nollakohdan. Regula falsi -menetelmässä (engl. False position method) puolestaan sovitetaan sekantti kulkemaan tarkasteluvälin pisteiden kautta, jolloin uusi tarkasteluväli määräytyy sekantin ja nollatason leikkauspisteen mukaan, siten että nollakohta sisältyy tarkasteluväliin. Näitä kahta menetelmää iteroidaan kunnes, nollakohta löytyy tietyn toleranssin rajoissa tai voidaan osoittaa, ettei nollakohtaa ole. Algoritmin tarkempi kuvaaminen rajataan tutkielman ulkopuolelle. Tarkemman kuvauksen algoritmin kaksiulotteiseen ympäristöön soveltuvasta versiosta löytää sivulta (Catto 2013).

5.2.4 Inkrementaalinen kontaktimoniston muodostaminen GJK

Kuten luvun 5 alussa todettiin tapahtuu kontaktimoniston muodostaminen GJK-algoritmia hyödyntäen inkrementaalisesti. Tällöin jokaisella kehyksellä tuotetaan yksi kontaktipiste, jolloin täyden kontaktimoniston muodostaminen tapahtuu useammalla kehyksellä. Sen lisäksi, että tämä vaatii uusien kontaktipisteiden määrittelemisen, täytyy aikaisemman määritellyt kontaktipisteen tarkastaa. Myös uudet kontaktipisteet, jotka ovat liian lähellä edellisillä kehyksillä löydettyjä kontaktipisteitä, tulisi hylätä (Catto 2007). Verrattuna luvussa 5.3.2 esitettävään täyden kontaktimoniston muodostamineen, on inkrementaalinen menetelmä näistä huomattavasti yksinkertaisempi ja nopeampi (Gregorius 2015). Hyvien puolien ohella on menetelmällä myös huonot puolensa. Koska joka kehyksellä kyetään luomaan ainoastaan yksi kontaktipiste, ja jos ei aikaisempia kontaktipisteitä ole olemassa, aiheuttaa tämä kappaleen törmätessä fysiikan mallinnuksessa

kappaleen kiertymisen (Gregorius 2015).

Jotta edellä kuvattu artefakta kyetään minimoimaan, jaetaan inkrementaalinen kontaktimoniston muodostaminen kahteen tapaukseen: matalaan ja syvään kontaktiin (G. v. d. Bergen 2010; Catto 2007). Matalasta kontaktista puhutaan silloin kun kappaleet eivät vielä varsinaisesti leikkaa toisiansa, mutta ovat riittävän lähellä, jotta voidaan kappaleiden lähimpien pisteiden avulla kontaktipiste luoda. Mahdollisesti näin luotu tai luodut kontaktipisteet ovat voimassa myös silloin, kun kappaleet penetroituvat eli ovat syvässä kontaktissa, jolloin saattavat ne vaikuttaa fysiikan mallinuksessa ilmentyviin artefakteihin. Jotta voidaan tarkemmin kontakti tulkita joko matalaksi tai syväksi, laajennetaan vähintään toista törmäyskappaletta (G. v. d. Bergen 2010) tai molempia törmäyskappaleita (Gregorius 2015, Catto 2007 tietyllä säteellä. Tässä tutkielmassa algoritmi kuvaillaan siten, että molempia kappaleita A:ta ja B:ta laajennetaan säteillä ρ_A ja ρ_B vastaavasti. Tällä tavalla muodostetut laajennetut kappaleet voidaan kuvitella koostuvan kahdesta kappaleesta, laajennetusta kappaleesta, sekä siihen sisältyvästä sisemmästä kappaleesta eli varsinaisesta törmäyskappaleesta. G. v. d. Bergen (2010) käyttää näistä nimityksiä iho (engl. skin) ja luu (engl. bone) vastaavassa järjestyksessä. Laajennettavana säteenä voidaan esimerkiksi käyttää matkaa, jonka levosta lähtevä kappale liikkuu sovelluksessa käytettävän painovoiman vaikutuksesta yhden kehyksen aikana (G. v. d. Bergen 2010).

Algoritmi 3 GJK-algoritmi yhdistettynä kontaki pisteen muodostamiseen (G. v. d. Bergen 2010)

$v \leftarrow$ mielivaltainen piste A-B:sta
$W \leftarrow \{\}$
$w \leftarrow s_{A-B}(-v)$
while $\|v_k\|^2 - v_k \cdot w_k \leq \varepsilon^2$ **do**
    if $v \cdot w > 0$ **and** $\frac{v_k \cdot w_k}{\|v_k\|}^2 > (\rho_A + \rho_B)^2$ **then** ▷ Laajennetut kappaleet eivät leikkaa
        return false
    end if
    $Y \leftarrow W_k \cup \{w_k\}$
    $v \leftarrow conv(Y)$:n lähimpänä origoa oleva piste
    $W \leftarrow$ pienin $X \subseteq Y$, missä $v \in conv(X)$
    $w \leftarrow s_{A-B}(-v)$
end while
if $\|v_k\|^2 > \varepsilon^2$ **then** ▷ Vain laajennetut kappaleet leikkaavat
    Laske lähimmät varsinaiste törmäyskappaleiden lähimmäiset pisteet p_A ja p_B
    $n \leftarrow \frac{v}{(}\|v_k\|)$
    $p_A \leftarrow p_A - \rho_A n$
    $p_B \leftarrow p_B + \rho_B n$
else ▷ Kappaleet penetroituvat
    Laske kontaktipiste penetraatiosyvyydestä
end if
return true

Matalasta kontaktista on kyse silloin kun laajennetut kappaleet leikkaavat toisiansa, mutta niiden sisältävät varsinaiset törmäyskappaleet eivät leikkaa toisiansa. Syvä kontakti puolestaan on kyseessä, kun varsinaiset törmäyskappaleet leikkaavat toisensa. GJK-algoritmissa jako matalaan ja syvään kontaktiin suoritetaan vertaamalla GJK-algoritmin palauttamaa minimi etäisyyden ylärajan $\|v\|$:n neliötä sallitun mittapoikkeaman ε_{tol}:n neliöön. Mikäli minimi etäisyyden ylärajan neliö on suurempi kuin sallitun mittapoikkeaman neliö, on kyseessä matala kontakti. Muussa tapauksessa on kyse syvästä kontaktista. Huomauttakoot, että jos ei ole tarvetta kappaleiden välistä

etäisyyttä määritellä tarkasti, voidaan GJK-algoritmiin lisätä lopetusehto, että lopetetaan iterointi heti, kun lähimpien pisteiden etäisyyden alarajan $\frac{v_k \cdot w_k}{\|v_k\|}$ neliö laskee alle säteiden summan neliön $(\rho_A + \rho_B)^2$. Pseudokoodi GJK-algroritmin ja inkrementaalisen kontaktimoniston muodostamisen yhdistelmälle esitetty listauksessa 3.

Syvissä kontakteissa joudutaan turvautumaan johonkin toiseen algoritmiin (Gregorius 2015; Catto 2007), kuten luvussa 5.2.2 esitettyyn EPA:an (G. v. d. Bergen 2010). Tällöin MTD:n määrittävä vektori lisätään kontaktimonistoon. Matalilla kontakteilla puolestaan jatketaan varsinaisilla törmäyskappaleilla sijaitsevien lähimpien pisteiden p_A ja p_B määrittelemiseen. Lähimpien pisteiden p_A ja p_B määrittelemiseksi, täytyy simpleksin lisäksi käytettävästä etäisyyden alialgoritmista palauttaa simpleksin affiinin verhon kärkien barysentriset koordinaatit λ_i origoa lähimpänä olevalle pisteelle v, sekä molemmille kappaleille erikseen kantajafunktion $s_{A-B}(v)$ palauttamat pisteet. Tällöin lähimmät pisteet ovat $p_A = \lambda_1 a_1 + ... + \lambda_n an$ ja $p_B = \lambda_1 b_1 + ... + \lambda_n bn$. Tarkemmin tämä saadaan johdettua seuraavasti. Kuten luvussa 5.1 esiteltiin, saadaan kantajafunktio CSO:ssa muodossa $s_{A-B}(v) = s_A(v) - s_B(-v)$, jolloin kaikki simpleksin kärjet $w_i \in W$ ovat vektoreiden $a_i = s_A(v)$ ja $b_i = s_B(-v)$ vektori erotuksia, jollekin suuntavektorille v (G. v. d. Bergen 2010). Ja koska piste v ilmaistaan W:n affiini kombinaationa muodossa: $v = \lambda_1 w_1 + ... + \lambda_n w_n$, niin voidaan lähimmät pisteet muodostaa v:n ilmaisevien barysentristen koordinaattien ja kantajafunktion määräävien pisteiden avulla. Näin muodostetut lähimmät pisteet sijaitsevat varsinaisten törmäyskappaleiden pinnalla, jolloin ne täytyy säteiden ρ_A ja ρ_B, sekä kontaktin normaalin $n = \frac{v}{\|v\|}$ avulla siirtää laajennetun kappaleiden pinnoille.

5.3 SAT kapeassa vaiheessa

Luvussa 2.7 nähtiin brute-force menetelmään perustuva SAT suorakulmaiselle särmiölle. Monimutkaisemmille monitahokkaille brute-force menetelmään perustuva SAT ei ole riittävän nopea, jotta sitä kyettäisiin reaaliaikaisissa sovelluksissa käyttämään. Alaluvussa 5.3.1 esitellään menetelmä SAT:n optimoimiseksi ja alaluvussa esitellään 5.3.2 eräs tapa muodostaa täysi kontaktimonisto.

5.3.1 Optimoitu SAT

Kuten luvussa 2.7 todettiin, kasvaa testattavien särmä parien määrä neliöllisesti särmien lukumäärän mukaan. Tällöin monimutkaisemmilla monitahokkailla särmä parien testaaminen vie suurimman osan SAT:n suoritusajasta, eikä brute-force menetelmään perustuva SAT ole riittävän nopea monimutkaisemille monitahokkaille reaaliaikaiseen törmäystarkasteluun. Erään tavan vähentää testattavien särmä parien lukumäärää, esitteli Migdalskiy (2010) ja Gregorius (2013). Tutkielmassa kyseinen optimointimenetelmä esitetään molempien lähteiden perusteella, samalla katsoen eroavaisuuksia lähteissä käytetyissä termeissä. Särmä parien karsinta suoritetaan hyödyntäen luvussa 5.1 esitettyjä Gaussin kuvausta ja CSO:ta. Lukijan on hyvä huomata, että vaikka tutkielmassa esitellään SAT:n optimointi tässä luvussa, voitaisiin sitä hyödyntää myös liukuhihnamallin aikaisemmissa vaiheissa, sillä perustuen Gregorius (2013) suorittamiin kokeisiin, suoriutuu optimoitu SAT jo yksinkertaisillakin kappaleille brute-force menetelmään perustuvaa SAT:a nopeammin. Monimutkaisemmille monitahokkaille puolestaan ero brute-force menetelmään on useiden kymmenten kertainen (Gregorius 2013)

Särmä parien karsinta perustuu siihen havaintoon, että ainoastaan ne särmä parit tulee testata, jotka CSO:n Gaussin kuvauksessa leikkaavat toisensa. Tällöin särmä parin ristitulona laskettu vektori on CSO:ssa tahkon normaali (Gregorius 2013) eli mahdollinen erottava akseli ja samoin se on molempien särmien yhdensuuntaisten kantavien hypertasojen normaali (Migdalskiy 2010). Vaikka Migdalskiy (2010) ei käytä termiä CSO tai Minkowskin erotus, tuottaa hänen käyttämänsä kahden kappaleen A ja B Gaussin kuvausten $G(A)$ ja $-G(B)$ päällekkäin asettamisen saman Gaussin kuvauksen, kuin Gregorius (2013) käyttämä $G(CSO)$. Kahden Gaussin kuvauksella muodostetun kaaren leikkaavuuden määrittely voidaan jakaa kolmeen tarkastukseen. Olkoon n_{A0} ja n_{A1} kaaren $G(e_A)$ ja n_{B0} ja n_{B1} kaaren $-G(e_B)$ päätepisteet yksikköpallolla. Lisäksi merkitään origoa o. Tällöin kaaret leikkaavat, jos ja vain jos:

- Pisteiden n_{A0}, n_{A1} ja o lävitse kulkeva taso jakaa pisteet n_{B0} ja n_{B1} eri puoliavaruuksiin.
- Pisteiden n_{B0}, n_{B1} ja o lävitse kulkeva taso jakaa pisteet n_{A0} ja n_{A1} eri puoliavaruuksiin.
- Kaaret $G(e_A)$ ja $G(e_B)$ eivät leikkaa toisiansa (Migdalskiy 2010) eli toisin sanoen kaarien $G(e_A)$ ja $-G(e_B)$ tulee sijaita samassa yksikköpallon hemisfäärissä (Gregorius 2013).

Edellä mainitut ehdot täyttyvät vain ja ainoastaan kun origo sisältyy $G(A)$ ja $G(B)$ päätepisteiden muodostamaan tetdaedriin eli 3-simpleksiin (Migdalskiy 2010). Tämä voidaan ratkaista vertailemalla simpleksin kaikkien kärkien etumerkkiä origon saamaan etumerkkiin suhteessa muihin simpleksin kärkiin, kuten tehtiin luvussa 5.2.1 esitellyssä SV algoritmissa. Tosin ilman kuvitteellisten simpleksien ja tilavuusmuodon käsitettä, voidaan tämä ratkaista myös kantavaa hypertasoa käyttämällä. Eli tällöin origon sisältyvyys tetraedriin (3-simpleksi) saadaan määriteltyä muodostamalla kantava hypertaso jokaiselle tetraedrin tahkolle ja tarkastamalla sisältyvätkö muut tetraedrin kärjet ja origo samaan puoliavaruuteen. Näin ollen kahden Gaussin kuvauksen kaaren leikkaavuuden määrittäminen voidaan suorittaa vain muutamaa ristitulo ja pistetulo operaatioita käyttäen. Listauksessa 4 on annettu pseudokoodi kahden Gaussin kuvauksen kaaren leikkaavuuden määrittelemiselle perustuen Gregorius (2013) esitykseen. Mainittakoon, että sama löytyy muun muassa PhysX:ssä toteutettuna.

Algoritmi 4 Algoritmi särmä parin leikkaavuuden määrittelemiseksi Gaussin kuvauksessa (Gregorius 2013)

1: $n_{A0} \leftarrow$ särmän A alku piste
2: $n_{A1} \leftarrow$ särmän A pääte piste
3: $n_{B0} \leftarrow$ särmän B alku piste
4: $n_{B1} \leftarrow$ särmän B pääte piste

5: $xn_A \leftarrow ristitulo(n_{A1}, n_{A0})$
6: $xn_B \leftarrow ristitulo(n_{B1}, n_{B0})$

7: $n_{B0}xn_A \leftarrow pistetulo(n_{B0}, xn_A)$
8: $n_{B1}xn_A \leftarrow pistetulo(n_{B1}, xn_A)$
9: $n_{A0}xn_B \leftarrow pistetulo(n_{A0}, xn_B)$
10: $n_{A1}xn_B \leftarrow pistetulo(n_{A1}, xn_B)$

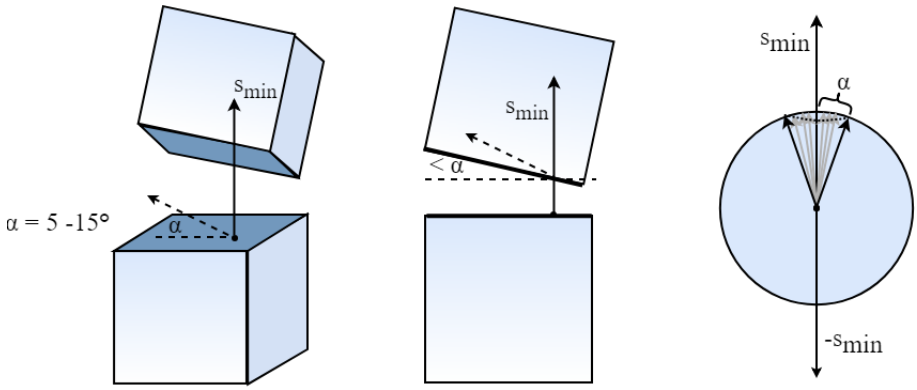
11: **return** $n_{B0}xn_A \cdot n_{B1}xn_A < 0$ **and** $n_{A0}xn_B \cdot n_{A1}xn_B < 0$ **and** $n_{B0}xn_A \cdot n_{A1}xn_B > 0$

5.3.2 Täyden kontaktin moniston muodostaminen SAT

Kuten luvun 5 alussa todettiin, voidaan SAT:n avulla kontaktimonisto muodostaa kokonaisuudessaan yhden kehyksen aikana, jolloin käytetään termiä täyden kontaktimoniston muodostaminen. Tässä luvussa tullaan käymään läpi täyden kontaktimoniston muodostaminen perustuen artikkelissa (Migdalskiy 2010) esitettyyn menetelmään. Vaikka artikkelissa (Migdalskiy 2010), sekä myös tässä tutkielmassa kyseinen menetelmä esitellään SAT:n yhteydessä, ei kyseinen menetelmä ole sidottu SAT:en. Se soveltuu myös muiden törmäysten havaitsemisen algoritmien kanssa käytettynä, sillä huomatauksella, että menetelmä vaatii kontaktin normaalin määrittämisen ja törmäyskappaleiden viereiset ominaisuudet tulee olla nopeasti löydettävissä, jolloin esimerkiksi kuviossa 4 esitelty halkaistu särmä datarakenne soveltuu hyvin käytettäväksi. Kun kontaktin normaalin määrittäminen lasketaan yhdeksi vaiheeksi, voidaan menetelmä jakaa yhteensä kolmeen vaiheeseen:

- Kontaktin normaalin määrittäminen.
- Kontakti alueiden määrittäminen.
- Kontakti alueiden leikkaavuuden määrittäminen.

Kontaktin normaalina voidaan käyttää MTD:tä tai approksimaatiota MTD:stä (Migdalskiy 2010). SAT:ta käyttämällä voidaan kontaktin normaalina s käyttää kuviossa 6 esiteltyä vektoria s_{min}. Penetroituneille kappaleille s määräytyy sen testattavan akselin mukaan, jolle projisoituna saa kappaleiden välinen penetraatiosyvyys pienimmän arvonsa (Catto 2007). Kontaktin normaalin määrittämisen jälkeen tulee kontakti normaalia s kohtisuorassa olevien kappaleiden A:n ja B:n kantavilla tasoilla $P_A(s)$ ja $P_B(-s)$ sijaitsevat ominaisuudet $F_A(s)$ ja ja $F_B(-s)$ määritellä. Koska usein fysiikan mallinnuksen kannalta halutaan löytää mahdollisimmat laajat kontakti alueet, niin tarkemmin pyritään määrittelemään kaikki kantavia tasoja vasten olevat ominaisuudet, toleranssi kulman α:n rajoissa. Tarkemmin α esitellään kuviossa 17 vasemmalla ja keskellä.



Kuvio 17. Toleranssi kulma α.

Artikkelissaan Migdalskiy (2010) jakaa kontakti alueiden määrittämisen kahteen tapaukseen: yksinkertaisten monitahokkaiden, sekä monimutkaisten monitahokkaiden kontakti alueiden määrittämiseen. Yksinkertaisemmille monitahokkaille kuten suorakulmaiselle särmiölle, riittää löytää yksi mahdollisimman moniulotteinen ominaisuus, kuten tahko. Monimutkaisemmille monitahokkaille tulee puolestaan löytää useampi mahdollisimman moniulotteinen ominaisuus, jotta fysiikan mallinnus pystyy järkevästi mallintamaan tilanteet, joissa kappale on levossa. Yksinkertaisen monitahokkaan tapauksessa voidaan suoraan siirtyä kontakti alueiden leikkaavuuden määrittämiseen, jos kantava ominaisuus on tahko. Jos taas kantava ominaisuus on särmä tai kärki, pyritään sitä laajentamaan kulman α rajoissa. Yksinkertaisille monitahokkaille voidaan laajennus suorittaa brute-force menetelmällä. Tällöin tutkitaan kantavan ominaisuuden viereisten ominaisuuksien suuntautumista kantavaan tasoon. Jos viereiset ominaisuudet ovat kulman α rajoissa vasten kantavaa tasoa, lisätään ne kontakti ominaisuuksien joukkoon, josta muodostetaan kontakti alue, kun kaikki edellä mainitun ehdon täyttävät ominaisuudet on löydetty.

Monimutkaisemmilla kappaleilla toleranssi kulman α rajoissa sijaitsevien ominaisuuksien löytämiseen käytettään apuna Gaussin kuvausta. Gaussin kuvauksessa erottava akseli s kuvantuu janaksi, joka läpäisee yksikköpallon. Kun erottavaa akselia laajennetaan toleranssi kulmalla α, muodostuu yksikköpallolla sijaitsevan janan ympärille ympyrän muotoinen pinta $G(s\alpha)$, johon sisältyvät Gaussin kuvauksella muodostetut ominaisuudet ovat toleranssi kulman rajoissa kantavasta tasosta (Migdalskiy 2010). Täsmällisemmin tämä on havainnollistettu kuviossa 17

oikealla. Niiden ominaisuuksien löytämiseksi, jotka Gaussin kuvauksessa sisältyvät joukkoon $G(s\alpha)$:n, esittelee Migdalskiy (2010) kaksi tapaa: täsmällisen kontakti alueen tuottaminen tulvatäyttö (engl. flood-fill) menetelmää käyttäen, sekä approksimaation tuottaminen näytteistämisen (engl. sampling) avulla.

Koska tutkielmassa käsitellään reaaliaikaisia sovelluksia, on laskennallisesti kevyempi (Migdalskiy 2010) näytteistäminen näistä sopivampi vaihtoehto. Täsmällisestä kontakti alueen etsimisestä kiinnostunut lukija voi lukea tarkemmin artikkelista (Migdalskiy 2010, s. 81 -85). Näytteistämisessä $G(s\alpha)$:aa näytteistetään n kertaa säännöllisesti eri kulmista. Esimerkiksi, jos n on kahdeksan, näytteistetään $\frac{360^\circ}{n} = 45^\circ$ asteen välein, jolloin tullaan löytämään kaikki ne $G(s\alpha)$:n sisältämät monitahokkaan kärjet, joiden sisäkulma on maksimissaan 135° astetta (Migdalskiy 2010). Itsessään näytteistäminen tapahtuu yksinkertaisesti kantajafunktion avulla. Kantajafunktion suuntavektorit saadaan yhtälöstä: $n_i = n + (u\cos(\frac{2\pi i}{n}) + v\sin(\frac{2\pi i}{n}))\tan(\alpha)$, missä $i = \{1,2,..,n\}$ (Migdalskiy 2010). Näin muodostetusta n kappaleen monitahokkaan kärkien joukosta, tulee poistaa duplikaatit, sekä toisiaan liian lähellä olevat kärjet (Migdalskiy 2010).

Viimeisessä vaiheessa suoritetaan kahden kappaleen kontakti alueiden leikkaavuuden määrittäminen. Koska kontakti alueiden määrittäminen suoritettiin näytteistämisen avulla, tulee tällöin määritellä enintään n kappaleesta kärkiä koostuvien monikulmioiden leikkaavuus. Kahden monikulmion leikkaavuuden määrittelemiseen on olemassa monia algoritmeja, joista yhden yksinkertaisen mentelmän esittelivät Shamos ja Hoey (1976). Menetelmässä molemmista monikulmiosta muodostetaan kaksi särmistä koostuvaa ketjua ylhäältä alas. Tämän jälkeen kyseiset särmä ketjut käydään iteratiivisesti ylhäältä alas suorittamalla särmien kärkien välillä muodostettujen puolisuunnikkaiden leikkaavuus testejä. Monikulmion puolisuunnikkaan pisteet, jotka sisältyvät myös toisen monikulmion puolisuunnikkaaseen, muodostaa yhden kontaktipisteen kontaktimonistossa. Kun kaikki puolisuunnikkaat ollaan käyty lävitse, on täysi kontaktimonisto valmis.

6 Pohdinta

Tutkielman tavoitteena oli esitellä nykypäivänä reaaliaikaisissa sovelluksissa käytettyjä menetelmiä ja algoritmeja törmäystarkastelun toteuttamiseksi. Tutkielmassa esitettäväksi algoritmeiksi valittiin kolmessa fysiikkamoottorissa, sekä yhdessä törmäystarkastelukirjastossa esiintyvien algoritmien perusteella niissä useimmiten esiintyvät. Esiteltävien algoritmien ohella tutkielmassa käsiteltiin monia törmäystarkastelun ja matematiikan termejä. Koska tutkielmassa käsitelty aihepiiri on kuitenkin melko laaja, rajattiin tutkielmassa aihetta hieman. Tutkielman ulkopuolelle jätettiin muun muassa jatkuvan törmäysten havaitsemisen toteuttaminen laajassa vaiheessa, jonka toteuttamiseen triviaali ratkaisu olisi laajentaa rajaavia tilavuuksien niihin liitettyjen kappaleiden muunnosten mukaisesti. Toinen vaihtoehto olisi käyttää Coming ja Staadt (2006) esittelemää kineettistä SAP-algoritmia, jossa arvioitujen törmäysten ajankohtien mukaan pyritään törmäykset ratkaisemaan oikeassa järjestyksessä ja oikealla ajanhetkellä. Tämän ohella tutkielmasta rajattiin pois yksilölliset leikkaavuuden määrittelemisen menetelmät törmäysprimitiivien välille, sekä GPU:lla tapahtuva törmäysten havaitseminen.

Tutkielmassa havaittiin törmäystarkastelussa käytettyjen algoritmien juurten löytyvän matematiikasta, joista korostuu etenkin konvekseja joukkoja koskevat teoreemat ja käsitteet, kuten esimerkiksi erottavan hypertaso teoreema ja Minkowskin summa. Lisäksi tutkielmassa esitettävien algoritmien pohjalta voidaan havaita törmäystarkastelun olevan kohtalaisen pitkään tutkittu osaalue tietokonegrafiikan historiassa. Samoin voidaan todeta vuosi kymmenten ikäisten algoritmien olevan edelleen suosiossa. Näistä hyviä esimerkkejä ovat alun perin vuonna 1988 esitelty GJK-algoritmi (Gilbert, Johnson ja Keerthi 1988), sekä vuonna 1992 esitelty Sweep-and-Prune algoritmi (Baraff 1992). Kohtalaisen pitkän historian lisäksi voidaan todeta törmäystarkastelun kehittyvän koko ajan ja olevan edelleen tutkimuksen kohteena. Hyvä esimerkki tästä on tutkielmassakin esitetty SV algoritmi (Montanari, Petrinic ja Barbieri 2017). Tulevaisuudessa uudet haasteet tuovat haptiset sovellukset, joissa todellisen interaktiivisuuden tunteen tuottamiseksi, tullaan tarvitsemaan “reaalikaisissa” sovelluksissa käytettyjä törmäysten havaitsemisen algoritmeja moninkertaisesti nopeampia menetelmiä.

Lähteet

Akenine-Möller, T., E. Haines ja N. Hoffman. 2008. *Real-Time Rendering, Third Edition.* CRC Press. ISBN: 9781439865293. https://books.google.fi/books?id=g%5C_PRBQAAQBAJ.

Anderson, Sean Eron. 2005. "Bit twiddling hacks". *URL: http://graphics. stanford. edu/~ seander/bithacks. html.*

Baraff, David. 1992. *Dynamic simulation of non-penetrating rigid bodies.* Tekninen raportti. Cornell University.

Barber, C Bradford, David P Dobkin ja Hannu Huhdanpaa. 1996. "The quickhull algorithm for convex hulls". *ACM Transactions on Mathematical Software (TOMS)* 22 (4): 469–483.

Bergen, G. van den, ja D. Gregorius, toimittaneet. 2010. *Game Physics Pearls.* CRC Press.

Bergen, Gino van den. 1997. "Efficient collision detection of complex deformable models using AABB trees". *Journal of Graphics Tools* 2 (4): 1–13.

———. 1999. "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects". *Journal of Graphics Tools* 4 (2): 7–25.

———. 2001. "Proximity queries and penetration depth computation on 3d game objects". Teoksessa *Game developers conference,* nide 170.

———. 2004a. *Collision Detection in Interactive 3D Environments.* Collision Detection in Interactive 3D Environments, nid. 1. Taylor & Francis. ISBN: 9781558608016. https://books.google.fi/books?id=m6smAQAAMAAJ.

———. 2004b. "Ray Casting against General Convex Objects with Application to Continuous Collision Detection".

———. 2013. "GDC 2013: Spatial Data Structures". Viitattu 2. huhtikuuta 2018. https://www.gdcvault.com/play/1017645/Physics-for-Game-Programmers-Spatial.

Bergen, Gino van den. 2010. “Smooth Mesh Contacts with GJK”. Toimittanut G. van den Bergen ja D. Gregorius: 99–123.

Boyd, Stephen, ja Lieven Vandenberghe. 2004. *Convex Optimization.* New York, NY, USA: Cambridge University Press. ISBN: 0521833787.

“Bullet Physics: Documentation”. 2018. Viitattu 2. huhtikuuta 2018. http://bulletphysics.com/Bullet/BulletFull/index.html.

“Bullet Physics: Homepage”. 2018. Viitattu 2. huhtikuuta 2018. http://bulletphysics.org/wordpress/.

Cameron, Stephen. 1997. “Enhancing GJK: Computing minimum and penetration distances between convex polyhedra”. Teoksessa *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on,* 4:3112–3117. IEEE.

Catto, Erin. 2007. “GDC 2007: Contact Manifolds”. Viitattu 2. huhtikuuta 2018. http://box2d.org/downloads/.

———. 2013. “GDC 2013: Continuous Collision”. Viitattu 2. huhtikuuta 2018. https://archive.org/details/GDC2013Catto.

Choi, Yi-King, Xueqing Li, Wenping Wang ja Stephen Cameron. 2005. “Collision detection of convex polyhedra based on duality transformation”. *The University of Hong Kong.*

Cohen, Jonathan D, Ming C Lin, Dinesh Manocha ja Madhav Ponamgi. 1995. “I-collide: An interactive and exact collision detection system for large-scale environments”. Teoksessa *Proceedings of the 1995 symposium on Interactive 3D graphics,* 189–ff. ACM.

Coming, Daniel S., ja Oliver G. Staadt. 2006. “Kinetic sweep and prune for multi-body continuous motion”. *Computers & Graphics* 30 (3): 439–449. ISSN: 0097-8493. http://www.sciencedirect.com/science/article/pii/S0097849306000677.

De Berg, Mark, Marc Van Kreveld, Mark Overmars ja Otfried Cheong Schwarzkopf. 2000. “Computational geometry”. Teoksessa *Computational geometry.* Springer.

Ericson, C. 2004. *Real-Time Collision Detection.* M038/the Morgan Kaufmann Ser. in Interactive 3D Technology Series. CRC Press. ISBN: 9780080474144. https://books.google.fi/books?id=4wTNBQAAQBAJ.

Gargantini, Irene. 1982. "Linear octtres for fast processing of three-dimensional objects". *Computer Graphics and Image Processing* 19 (1): 88–89.

Gilbert, E. G., D. W. Johnson ja S. S. Keerthi. 1988. "A fast procedure for computing the distance between complex objects in three-dimensional space". *IEEE Journal on Robotics and Automation* 4, numero 2 (huhtikuu): 193–203. ISSN: 0882-4967.

Gilbert, Elmer G, ja C-P Foo. 1990. "Computing the distance between general objects in three-dimensional space". *IEEE Transactions on Robotics and Automation* 6 (1): 53–61.

Gottschalk, S., M. C. Lin ja D. Manocha. 1996. "OBBTree: A Hierarchical Structure for Rapid Interference Detection". Teoksessa *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques,* 171–180. SIGGRAPH '96. New York, NY, USA: ACM. ISBN: 0-89791-746-4.

Gregorius, Dirk. 2013. "GDC 2013: The Separating Axis Test between Convex Polyhedra". Viitattu 2. huhtikuuta 2018. https://www.gdcvault.com/play/1017646/Physics-for-Game-Programmers-The.

———. 2014. "GDC 2014: Implementing Quickhull". Viitattu 2. huhtikuuta 2018. https://archive.org/details/GDC2014Gregorius.

———. 2015. "GDC 2015: Robust Contact Creation for Physics Simulations". Viitattu 2. huhtikuuta 2018. https://www.gdcvault.com/play/1022194/Physics-for-Game-Programmers-Robust.

Gregory, J. 2009. *Game Engine Architecture, Second Edition.* CRC Press. ISBN: 9781466560062. https://books.google.fi/books?id=LlLSBQAAQBAJ.

Gruber, Marvin H. J. 2013. *Matrix Algebra for Linear Models.* Wiley.

Gustafsson, Dennis. 2010. “Understanding Game Physics Artifacts”. Toimittanut G. van den Bergen ja D. Gregorius: 29–44.

“Havok: About Havok”. 2018. Viitattu 2. huhtikuuta 2018. https://www.havok.com/about-havok/.

Lin, M. C., ja J. F. Canny. 1991. “A fast algorithm for incremental distance calculation”. Teoksessa *Proceedings. 1991 IEEE International Conference on Robotics and Automation,* 1008–1014 vol.2. Huhtikuu.

McShaffry, Mike. 2014. *Game coding complete.* Nelson Education.

Migdalskiy, Sergiy. 2010. “SAT In Narrow Phase and Contact-Manifold Generation”. Toimittanut G. van den Bergen ja D. Gregorius: 63–98.

Mirtich, Brian. 1998. “V-Clip: Fast and Robust Polyhedral Collision Detection”. *ACM Trans. Graph.* (New York, NY, USA) 17, numero 3 (heinäkuu): 177–208. ISSN: 0730-0301.

Mirtich, Brian Vincent. 1996. *Impulse-based dynamic simulation of rigid body systems.* Citeseer.

Montanari, Mattia, Nik Petrinic ja Ettore Barbieri. 2017. “Improving the GJK Algorithm for Faster and More Reliable Distance Queries Between Convex Objects”. *ACM Trans. Graph.* (New York, NY, USA) 36, numero 3 (kesäkuu): 30:1–30:17. ISSN: 0730-0301.

Moravanszky, Adam, ja Pierre Terdiman. 2004. “Fast contact reduction for dynamics simulation”: 253–263.

Muller, D.E., ja F.P. Preparata. 1978. “Finding the intersection of two convex polyhedra”. *Theoretical Computer Science* 7 (2): 217–236. ISSN: 0304-3975.

“Nvidia: PhysX”. 2018. Viitattu 2. huhtikuuta 2018. https://developer.nvidia.com/physx-source-github.

“ODE: wiki”. 2018. Viitattu 2. huhtikuuta 2018. https://www.ode-wiki.org/wiki/index.php?title=Main_Page.

“PhysX: Game Titles”. 2018. Viitattu 2. huhtikuuta 2018. https://www.geforce.com/hardware/technology/physx/games.

"PhysX: User's Guide". 2018. Viitattu 2. huhtikuuta 2018. http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Index.html.

Rockafellar, R. Tyrrell. 1997. *Convex Analysis.* Princeton Paperbacks. Princeton University Press. ISBN: 9780691015866. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=969057&site=ehost-live.

Schneider, Rolf. 1993. *Convex bodies : the Brunn-Minkowski theory.* 490. Encyclopedia of mathematics and its applications. Cambridge ; New York: Cambridge University Press. http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=569310.

Shamos, Michael Ian, ja Dan Hoey. 1976. "Geometric intersection problems". Teoksessa *Foundations of Computer Science, 1976., 17th Annual Symposium on,* 208–215. IEEE.

Snethen, Gary. 2008. "XenoCollide: Complex Collision Made Simple". *Game Programming Gems 7:* 165–178.

Tracy, D. J., S. R. Buss ja B. M. Woods. 2009. "Efficient Large-Scale Sweep and Prune Methods with AABB Insertion and Removal". Teoksessa *2009 IEEE Virtual Reality Conference,* 191–198. Maaliskuu.

Ulrich, Thatcher. 2000. "Loose octrees". *Game Programming Gems* 1:444–452.

Weller, René. 2013. *New geometric data structures for collision detection and haptics.* Springer Science & Business Media.