Haaga-Helia
ammattikorkeakoulu Oy

Shadows Collide - 2D-törmäyksentunnistusohjelma

Olli Lappalainen

Opinnäytetyö
Tietojenkäsittelyn koulutus
2019

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Tekijä
Olli Lappalainen

Koulutusohjelma
Tietojenkäsittely

Raportin/Opinnäytetyön nimi	**Sivu- ja liitesivumäärä**
Shadows Collide – 2D-törmäyksentunnistusohjelma	29 + 2

Shadows Collide on 2D-törmäyksentunnistusohjelma, joka kehitettiin toimeksiantona Shadow Gameworks -pelistudiolle. Ohjelman tarkoituksena on olla perusta yrityksen ensimmäisen tietokonepelin fysiikkamoottorille. Ohjelma sisältää komentorivikäyttöliittymän, jolla käyttäjä voi kokeilla vektoreina, eli x ja y, listana syötettyjen monikulmioiden keskinäistä törmäämistä. Törmäämisellä tarkoitetaan kahden monikulmion kohtaamista 2D-avaruudessa.

Ohjelma sisältää kolme erilaista törmäyksentunnistusalgoritmia. AABB-algoritmi vaakasuorille nelikulmioille, ympyräalgoritmi ympyröille ja OBB-algoritmi muille koverille monikulmioille ja edellä mainittujen yhdistelmille. OBB-algoritmin keskeisessä roolissa käytetään SAT-algoritmia. SAT on erottavan akselin lauseen lyhenne, eli Separating Axis Theorem. Lauseen mukaan kaksi kuperaa monikulmiota eivät leikkaa, jos niiden välistä löytyy akseli, joka ei leikkaa kumpaakaan monikulmiota. Kuperalla monikulmiolla tarkoitetaan sulkeutuvaa ja vähintään kolme pistettä sisältävää monikulmiota, joka ei leikkaa itseään ja jonka jokainen sisäkulma on maksimissaan 180-astetta.

Shadows Collide kehitettiin Agile-projektina suunnittelu- ja kehitysvaiheissa. Suunnitteluvaiheessa luotiin matemaattinen selvitys tarvittaville funktioille ja konsepteille. Ohjelman rakenne suunniteltiin, josta syntyi vuo- ja luokkakaaviot. Kaavioiden perusteella GitHub-palvelun projektinhallintaan luotiin tehtäväkortit. Kehitysvaiheessa ohjelma ohjelmoitiin GitHub PMS:n tehtäväkorttien perusteella. Ohjelmoinnin aikana syntyneet lisätarpeet lisättiin tehtäväkortteina projektinhallintaan, jotta ohjelman kehitystä oli mahdollista seurata. Shadows Collide on kehitetty C++:lla ja kehitysohjelmana käytettiin Microsoft Visual Studio 2019 Community IDE:tä.

Hyvästä suunnittelusta huolimatta ohjelman kehityksessä ilmeni odottamattomia haasteita, kuten monissa, ellei kaikissa sovelluskehitysprojekteissa. Matemaattisten funktioiden ohjelmointi algoritmiksi osoittautui yleisimmäksi haasteeksi. Toiseksi yleisimmät ongelmat olivat ohjelmiston loogisia osia, jotka olivat jääneet suunnittelusta sivuun ja esiintyivät vasta ohjelmoidessa. Ongelmista huolimatta Shadows Collide valmistui suunnitellussa aikataulussa tyydyttävin tuloksin ja täyttäen toiminnalliset vaatimukset.

Jatkokehityksenä Shadows Collide optimoidaan vähemmän muistia käyttäväksi. Ohjelman lisäksi ohjelmoidaan renderöintimoottori Vulkan tai DirectX 3D-grafiikkarajapinnoilla, jonka jälkeen fysiikkasimulaatioiden kehitys on mahdollista. Pitkäaikaisena tavoitteena on luoda pelimoottori, jota käytetään Shadow Gameworksin ensimmäisessä tietokonepelissä.

Asiasanat
Törmäyksentunnistus, tietokonepelit, C++, peliohjelmointi, tasogeometria

Sisällys

1 Johdanto ... 1
1.1 Shadows Collide ... 1
1.2 Projektin tavoite ... 2
1.3 Projektin rajaus ... 2
1.4 Keskeiset käsitteet ... 3
2 Törmäyksentunnistamisen menetelmät, matematiikka ja algoritmit ... 6
2.1 AABB – Axis Aligned Bounding Box ... 6
2.2 Ympyrä ... 7
2.2.1 Ympyrän keskipisteen laskeminen ... 7
2.2.2 Ympyröiden törmäyksentunnistaminen matemaattisesti ... 10
2.3 OBB – Oriented Bounding Box ... 11
2.3.1 OBB – OBB törmäyksentunnistus ja Separating Axis Theorem (SAT) ... 12
2.3.2 Projektion laskeminen ... 14
2.4 Kupera monikulmio ... 15
2.4.1 Monikulmion kuperuuden tunnistaminen ... 16
2.4.2 Kuperan monikulmion törmäyksentunnistus ... 17
2.5 Tila-avaruuden jakaminen ... 17
3 Kehityssuunnitelma ... 19
3.1 Suunnittelu ... 19
3.2 Kehitys ... 19
3.3 Teknologiat ja työkalut ... 20
4 Shadows Collide arkkitehtuuri ... 21
4.1 Ohjelman elinkaari ... 21
4.2 Ohjelman luokat ... 22
5 Pohdinta ... 25
Lähteet ... 28
Liitteet ... 30
Liite 1. Vuokaavio ... 30
Liite 2. Luokkakaavio ... 31

1 Johdanto

Shadows Collide on 2D-törmäyksentunnistusohjelma Shadow Gameworksin tulevaa tietokonepeliä varten. Shadows Gameworks on 2019 perustettu omarahoitteinen tietokonepeliyritys, joka työstää ensimmäistä tietokonepeliään. Tavoitteena on kehittää interaktiivisia kokemuksia pelaajille rakkaudesta lajiin. Siten myös antaa oma panos peliteollisuuden nykytilan parantamiselle, joka keskittyy valitettavasti yhä enemmän rahoittajiin eikä pelaajiin.

Törmäyksentunnistamista käytetään mm. fysiikkasimulaattoreissa, monissa videopeleissä ja graafisissa käyttöliittymissä. Esimerkiksi videopelissä hahmon pysyminen maalla, taistelupelissä miekan osuminen kohteeseen ja jalkapallopelissä pallon potkaiseminen käyttävät törmäyksen tunnistamista. Käyttöliittymissä tunnistetaan käyttäjän hiiren liike napin päällä törmäyksentunnistamisella. (Thompson 2019a.)

Ilman törmäyksentunnistamista kaikki videopelit paitsi tekstipohjaiset, olisivat pelikelvottomia. Ammuntapelissä yksikään luoti ei osuisi kohteeseen ja hahmot leijuisivat ohjelman sammumiseen asti äärettömässä 2D- tai 3D-avaruudessa reagoimatta ympäristöön. Mikään fysiikkasimulaatio tai interaktiivinen graafinen käyttöliittymä ei reagoisi käyttäjän tekemisiin.

Törmäyksentunnistamisen avulla voidaan määrittää mitä tapahtuu, kun ohjelmassa esiintyvät geometriat leikkaavat toisiaan (Grant 2005, 139). Törmäyksentunnistamisella ylläpidetään illuusiota virtuaalimaailman fyysisestä ympäristöstä (Ericson 2005a, 1).

1.1 Shadows Collide

Shadows Collide luo perustan laajamittaiselle törmäyksentunnistusjärjestelmälle, jota voidaan tulevaisuudessa käyttää tietokonepelin fysiikkamoottorissa. Ohjelma ei kuitenkaan sisällä visualisointia geometrioista, koska se ei ole renderöintimoottori. Käyttöliittymänä toimii yksinkertainen komentorivikäyttöliittymä, johon käyttäjä syöttää vektorien x- ja y-koordinaatit. Vastauksena ohjelma ilmoittaa käyttäjälle törmäyksen lopputuloksen boolean-arvona ja geometrioiden etäisyyden toisistaan koordinaatistolla.

Ohjelma tunnistaa käyttäjän syöttämästä vektorilistasta muodon tyypin ja päättää siten tarvittavan törmäyksentunnistusoperaation tyypin. Käyttäjä voi syöttää vain kuperia monikulmioita. Jos ohjelma toteaa monikulmion olevan kovera, ilmoittaa se virheestä ja pyytää

kuperaa monikulmiota. Käyttäjän on syötettävä vähintään kolme vektoria, eli kolmio. Shadows Collide ei tässä opinnäytetyössä sisällä suorien tai yksittäisten pisteiden törmäyksentunnistusta.

1.2 Projektin tavoite

Projektin tavoitteena on luoda perusta 2D-fysiikkamoottorille tietokonepeliä varten. Olen myös tarkoituksella valinnut olla käyttämättä valmiita vastaavia ohjelmia kehittääkseni omaa tietotaitoani tietokonepelien kehittämisessä.

Lopputuloksena ohjelman myötä syntyy myös rajapinta, jolla ohjelmaa voidaan kutsua esimerkiksi fysiikkamoottorissa. Shadows Collide tulee siis itsessään olemaan ohjelmistokomponentti. Tekemisen aikana syntyvät myös onnistuneen lopputuloksen takaamiseksi tarvittavat matemaattiset- ja ohjelmistoarkkitehtuuriset kaaviot.

Lopulta olen itse saavuttanut korkeamman käytännön ja teorian osaamisen tason pelifysiikoiden yhden tärkeimmän osa-alueen toiminnasta ja rakentamisesta. Lisäksi olen saanut tarvittavaa pohjaa oman pelimoottorin rakentamiselle tai hyvät lähtökohdat jonkun toisen moottorin muokkaamiselle omaan käyttöön sopivaksi.

1.3 Projektin rajaus

Ohjelma sisältää törmäyksentunnistuksen kaikille kuperille monikulmioille, mutta ei koverille muodoille. Ohjelma palauttaa käyttäjälle virheen, jos syötetty lista vektoreita muodostaa koveran monikulmion.

Muodon tulkinta

- Suoraksi tulkitaan vektorilista, jossa on kaksi vektoria.
- Suorakulmaiseksi nelikulmioksi tulkitaan vektorilista, jossa on neljä vektoria ja jokaisen suoran kulma seuraavaan suoraan on 90 asetta.
- Kolmioksi tai monikulmioksi tulkitaan vektorilista, joissa on kolme tai yli neljä vektoria.
- Ympyräksi tulkitaan kaikki yli kahdeksan vektoria sisältävät vektorilistat.
- Kuperaksi muodoksi tulkitaan vektorilista, jonka kaikki pisteet suhteessa yhteen monikulmion suoraan kerrallaan ovat samalla puolella suoraa.

Algoritmin valinta

- AABB: Jos molemmat monikulmiot ovat suorakulmaisia nelikulmioita ja jokaisen suoran kulmakerroin on nolla.
- Ympyrä: Jos molemmat muodot ovat ympyröitä.
- OBB: Jos vektorilistassa on kaksi eri tyyppistä muotoa tai neliö, jonka suoran kulmakerroin ei ole nolla.

Ohjelmaan ei implementoida monikulmioiden graafista visualisointia, koska oman renderöintimoottorin luominen olisi turhan työlästä, jopa laajempi kuin projekti muutoin. Ohjelman algoritmien toimivuus todennetaan luomalla monikulmioita, jotka tiedettävästi leikkaavat tai eivät ja ovat kuperia tai koveria. Näiden vektorit syötetään ohjelmalle yksikkötestinä, jotta voidaan todentaa algoritmien luotettavuus. Monikulmioita voi luoda esimerkiksi sivulla https://www.geogebra.org.

1.4 Keskeiset käsitteet

Akseli
Koordinaatiston x-, y- tai z-ulottuvuuksia kuvaava suora. Eli leveys-, korkeus- tai syvyysulottuvuus.

Vektori
Piste koordinaatistolla, jonka sijainti merkitään x-, y- ja z-arvoilla. 2D-avaruudessa pisteellä on vain korkeus- ja leveysarvo.

Suora
Viiva, jonka alku ja loppu merkitään kahdella vektorilla.

Normaali
Suoran s normaali n on suora, joka kulkee kohtisuoraan, eli 90 asteen kulmassa suoran s poikki. Esimerkiksi ristissä poikkiviiva on pystyviivan normaali.

Säde
Ympyrän keskipisteestä reunapisteeseen piirretty suora.

Projektio
Projektio on pisteen kohtisuora sijainti suoralla, joka ei kulje pisteen sijainnin kautta. Toisin sanoen pisteen p projektio saadaan muodostamalla suoralle s normaali n, joka kulkee pisteen p kautta. Pisteen p projektio suoralla s on tällöin suoran s ja normaalin n leikkauspiste.

Monikulmio
Monikulmio on geometria, joka sisältää n -määrän vektoreita. Monikulmion sivu on suljettu, eli viimeisen vektorin sijainti on ensimmäisen vektorin kanssa sama. Monikulmioita ovat esim. kolmiot, nelikulmiot, viisikulmiot ja satunnaiset monikulmiot.

Nelikulmio

Monikulmio, joka muodostuu neljästä vektorista.

Suorakulmio

Nelikulmio, jonka kaikki kulmat ovat suoriakulmia.

Neliö

Nelikulmio, jonka kaikki sivut ovat tasapitkät ja jokainen kulma on suorakulma, eli 90 astetta.

Lävistäjä

Monikulmion lävitse kulkeva suora.

Algoritmi

Kuvaus prosessin vaiheista, jonka tarkoitus on saavuttaa tietty tavoite. Järjestyksessä oleva lista yksiselitteisistä ohjeista kuinka edetä, jotta tietty lopputulos saavutetaan.

Bounding box eli raja-arvolaatikko

Törmäyksentunnistuksessa käytettävä yksinkertainen monikulmio, jolla ympäröidään monimutkaisempi monikulmio algoritmien keventämiseksi.

Agile

Tarkoittaa ketterää ohjelmistokehitystä ja on yleisnimitys ketterille ohjelmistokehityksen menetelmille. Agilen tunnuspiirteitä ovat iteraatiot kehityksessä, joita toistetaan mahdollistaakseen työskentelyn mukautuvuus projektin tilanteesta riippuen.

Versionhallinta

Järjestelmä, jonne voidaan tallentaa ohjelmistokoodia sen kehityksen eri vaiheissa. Näin voidaan ylläpitää ohjelmiston versiointia ja turvata esimerkiksi toimivaan versioon palaamisen mahdollisuus viallisen päivityksen jälkeen.

PMS (Project Management System)

Yhdistelmä projektin hallintaan käytettäviä menetelmiä. Ohjelmistoprojekteissa PMS:llä tarkoitetaan ohjelmaa, jonne projektia voidaan esimerkiksi dokumentoida, jakaa tehtäviin, jakaa tehtäviä projektiryhmän jäsenille, raportoida ongelmia ja seurata projektin edistymistä. Käytettyjä projektinhallintajärjestelmiä ovat mm. Atlassianin kehittämät Jira ja Trello.

Tekstieditori

Ohjelmakoodin hallintaan ja tuottamiseen tarkoitettu kirjoitusohjelma. Tunnettuja tekstieditoreita ovat esimerkiksi Notepad, Notepad++, Microsoft Visual Studio Code, Sublime Text, Atom, Vim ja Emacs.

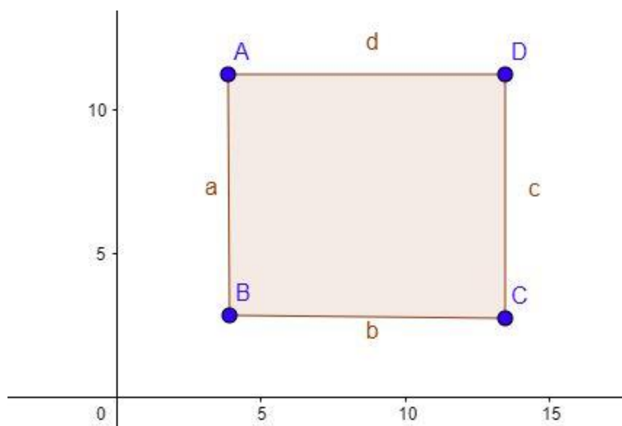
2 Törmäyksentunnistamisen menetelmät, matematiikka ja algoritmit

Törmäyksentunnistaminen on käytännössä x-, y-, ja z-avaruudessa sijaitsevien pisteiden sijaintien vertailua toisiinsa. Esimerkiksi pelissä tietokoneen näytöllä näkyvä laatikko muodostuu kulmapisteistä, joilla on näytöllä x- ja y-sijainti. Jos kyseessä on kolmiulotteinen kuva, on sillä myös z-sijainti, eli syvyysarvo. Näiden pisteiden sijainteja vertailemalla voidaan todeta, jos muodot leikkaavat toisiaan. Pisteiden vertailu onnistuu yksinkertaisella tasogeometrialla ja soveltamalla sitä erilaisiin törmäyksentunnistamisen algoritmeihin ja menetelmiin.

Törmäyksentunnistamiseen käytetään usein yksinkertaisempia monikulmioita monimutkaisen muodon sijaan. Esimerkiksi 2D-pelissä vihollisen ympärillä on suorakulmio tai ympyrä, jonka kohtaaminen toisen vastaavanlaisen muodon kanssa on ohjelmallisesti helpompi ja kevyempi tunnistaa. Näitä muotoja kutsutaan nimellä ”bounding box” eli raja-arvolaatikko. Yleisesti käytettyjä törmäyksiä ovat ympyrä – ympyrä, nelikulmio – ympyrä, nelikulmio – nelikulmio ja monikulmio - monikulmio. (MDN 2019.)

2.1 AABB – Axis Aligned Bounding Box

Axis aligned bounding box (AABB) -koe on erittäin yksinkertainen menetelmä tunnistaa törmäys kahden pelin perus x- ja y-akseliin nähden kallistumattoman suorakulmion törmäys. Yksinkertaisuutensa takia AABB törmäyksen tunnistaminen on laskennallisesti kevyttä. (Toptal 2019.)



Kuva 1. Axis Aligned Bounding Box

AABB törmäyksessä nelikulmioiden akselit ovat samat kuin pelin perusakseli, joten vertailussa voidaan verrata suorakulmioiden kulmien leikkaamista perusakselilla. Jos kulmat risteävät korkeus- ja leveysakseleilla, suorakulmiot törmäävät. (Learn OpenGL 2019.)

Törmäyksentunnistukseen tarvitaan yhteenlaskua ja vertailua pienemmyys- ja suuremmuusmerkeillä. Törmäys voidaan todentaa seuraavasti:

Suorakulmio 1 pienin x-arvo: N1Xmin
Suorakulmio 1 pienin y-arvo: N1Ymin
Suorakulmio 1 leveys: N1l
Suorakulmio 1 korkeus: k
Suorakulmio 2 pienin x-arvo: N1Xmin
Suorakulmio 2 pienin y-arvo: N1Ymin

Suorakulmiot N1 ja N2 leikkaavat, jos kaikki seuraavat väittämät ovat tosia.

- N1Xmin + N1l > N2Xmin
- N1Ymin + N1k > N2Ymin
- N1Xmin < N2Xmin + N2l
- N1Ymin < N2Ymin + N2k

2.2 Ympyrä

Ympyrän törmäys toisen ympyrän kanssa ei ole laskennallisesti yhtä tehokasta kuin AABB – AABB törmäyksen tunnistaminen, koska menetelmässä tarvitaan neliöjuurta, joka vaatii enemmän muistia kuin yksinkertaiset yhteenlaskut ja vertailut. Törmäyksen tunnistaminen on kuitenkin suhteellisen yksinkertainen operaatio.

Algoritmi tehdään laskemalla molemmista ympyröistä säteet, sekä keskipisteiden etäisyys toisistaan. Ympyrät leikkaavat, jos keskipisteiden välinen etäisyys on pienempi kuin säteiden pituuksien summa. (Thompson 2019b.)

2.2.1 Ympyrän keskipisteen laskeminen

On todennäköistä, että ympyrästä tunnetaan vain kehän pisteiden sijainnit. Törmäyksentunnistamiseen tarvitaan kuitenkin myös keskipiste. Ympyrän keskipiste voidaan laskea, kun tiedetään kolme pistettä ympyrän kehältä.

Käyttämällä kolmea pistettä voidaan luoda kaksi suoraa keskimmäisestä pisteestä ensimmäiseen ja viimeiseen pisteeseen. Ympyrän keskipiste on suorien keskinormaalien leikkauspiste. (Bourke, P. 1992.)

Pisteistä P_1, P_2 ja P_3 suorien yhtälöt ovat:

$$y_a = m_a(x - x_1) + y_1 \text{ ja } y_b = m_b(x - x_2) + y_2$$

Yhtälöissä m on suoran kulmakerroin, jotka voidaan laskea seuraavasti:

$$m_a = \frac{y_2 - y_1}{x_2 - x_1} \text{ ja } m_b = \frac{y_3 - y_2}{x_3 - x_2}$$

Yhtälöiden y_a ja y_b keskinormaalien yhtälöt ovat:

$$y'_a = -\frac{1}{m_a}\left(x - \frac{x_1 + x_2}{2}\right) + \frac{y_1 + y_2}{2} \text{ ja } y'_b = -\frac{1}{m_b}\left(x - \frac{x_1 + x_2}{2}\right) + \frac{y_1 + y_2}{2}$$

Suorien yhtälöissä kulmakerroin lasketaan vielä erikseen $-\frac{1}{m_a}$, koska suoran normaalin kulmakerroin on suoran kulmakertoimen vastaluku. Normaalin yhtälössä x- ja y-arvot jaetaan kahdella, koska halutaan saada normaalien sijainti suoran keskipisteellä.

Keskipisteen x-arvon voi laskea seuraavalla kaavalla:

$$x = \frac{m_a m_b (y_1 - y_3) + m_b (x_1 + x_2) - m_a (x_2 + x_3)}{2 * (m_b - m_a)}$$

Keskipisteen y-arvo selviää sijoittamalla edellä mainitun laskun x-arvo y'_a tai y'_b yhtälön x-arvoksi. (Bourke, P. 1992.)

Esimerkki: Olkoon $P_1(9.91, 11.71)$, $P_2(12, 14.48)$ ja $P_3(15.28,13.34)$

Kulmakertoimet:

$$m_a = \frac{14{,}48 - 11{,}71}{12{,}14 - 9{,}91}$$

$$m_a = \frac{2{,}77}{2{,}23}$$

$$m_a = \sim 1{,}2$$

$$m_b = \frac{13{,}24 - 14{,}48}{15{,}28 - 12}$$

$$m_b = \frac{-1{,}24}{3{,}28}$$

$$m_b = \sim -0{,}4$$

Suorien yhtälöt:

$$y_a = 1{,}2 * (x - 9{,}91) + 11{,}71$$

$$y_b = -0{,}4 * (x - 12) + 14{,}48$$

Keskinormaalien yhtälöt:

$$y'_a = -\frac{1}{1{,}2}(x - \frac{9{,}91 + 12}{2}) + \frac{11{,}71 + 14{,}48}{2}$$

$$y'_b = -\frac{1}{-0{,}4}(x - \frac{9{,}91 + 12}{2}) + \frac{11{,}71 + 14{,}48}{2}$$

Ympyrän keskipisteen x-arvo:

$$x = \frac{1{,}2 * -0{,}4 * (11{,}71 - 13{,}34) \pm 0{,}4 * (9{,}91 + 12) - 1{,}2 * (12 + 15{,}28)}{2 * (-0{,}4 - 1{,}2)}$$

$$x = \frac{1{,}2 * -0{,}4 * -1{,}63 + -0{,}4 * 21{,}91 - 1{,}2 * 27{,}28}{2 * -1{,}6}$$

$$x = \frac{0{,}78 + -8{,}76 - 32{,}74}{-3{,}2}$$

$$x = \frac{-40{,}72}{-3{,}2}$$

$$x = \sim 12{,}73$$

Keskipisteen y-arvo:

$$y = -\frac{1}{1{,}2}(12{,}73 - \frac{9{,}91 + 12}{2}) + \frac{11{,}71 + 14{,}48}{2}$$

$$y = -\frac{1}{1{,}2}(12{,}73 - \frac{21{,}91}{2}) + \frac{11{,}71 + 14{,}48}{2}$$

$$y = -\frac{1}{1{,}2}(12{,}73 - 10{,}96) + \frac{11{,}71 + 14{,}48}{2}$$

$$y = -\frac{1}{1{,}2} * 1{,}77 + \frac{11{,}71 + 14{,}48}{2}$$

$$y = -\frac{1}{1{,}2} * 1{,}77 + \frac{26{,}19}{2}$$

$$y = -0{,}83 * 1{,}77 + 13{,}10$$

$$y = -1{,}47 + 13{,}10$$

$$y = 11{,}63$$

Keskipisteen sijainti on (12.73, 11.63).

2.2.2 Ympyröiden törmäyksentunnistaminen matemaattisesti

Säteenpituuden ja keskipisteiden etäisyyden saa laskemalla pisteiden välisen suoran pituuden.

Laskutoimitus ilmaistaan kaavalla (OPH 2019.):

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esimerkiksi olkoon keskipiste A(2, 3) ja keskipiste B(5, 8).

$$|AB| = \sqrt{(5 - 2)^2 + (8 - 3)^2}$$
$$|AB| = \sqrt{3^2 + 5^2}$$
$$|AB| = \sqrt{9 + 25}$$
$$|AB| = \sqrt{34}$$
$$|AB| = \sim 5{,}8$$

Keskipisteiden AB etäisyys on ~5,8.

Vastaavasti voidaan laskea ympyröiden säteet.

Olkoon ympyrä Y_1 keskipiste A(2, 3) ja ympyrän kehältä piste B(2, 5). Ympyrän säde r_1 lasketaan:

$$r = \sqrt{(2 - 2)^2 + (5 - 3)^2}$$
$$r = \sqrt{0^2 + 2^2}$$
$$r = \sqrt{0 + 4}$$
$$r = \sqrt{4}$$
$$r = 2$$

Olkoon ympyrä Y_2 keskipiste A(5, 8) ja ympyrän kehältä piste B(6, 10). Ympyrän säde r_2 lasketaan:

$$r = \sqrt{(6 - 5)^2 + (10 - 8)^2}$$
$$r = \sqrt{1^2 + 2^2}$$
$$r = \sqrt{1 + 4}$$
$$r = \sqrt{5}$$

$$r = \sim 2{,}2$$

Lasketaan säteiden r_1 ja r_2 summa

$$|r_1 r_2| = 2 + 2{,}2$$

$$|r_1 r_2| = 4{,}2$$

Verrataan säteiden summaa keskipisteiden etäisyyteen ja saadaan törmäyksen c-arvo.

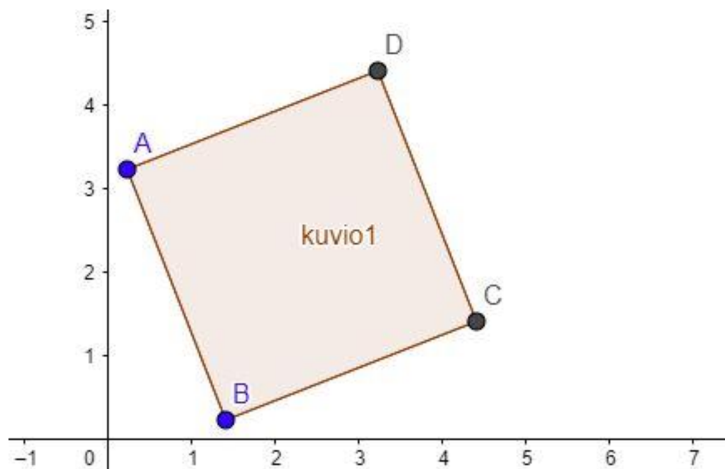
$c = 5{,}8 > 4{,}2$

$c = \text{tosi}$

Ympyröiden keskipisteiden etäisyys on pidempi kuin ympyröiden säteiden summa, joten ympyrät eivät leikkaa.

2.3 OBB – Oriented Bounding Box

OBB on AABB:n kaltainen suorakulmio, mutta toisin kuin AABB:n sivut ovat saman suuntaisia pelin perusakselin kanssa, OBB on kallistunut johonkin suuntaan. (Ericson 2005b, 101.)



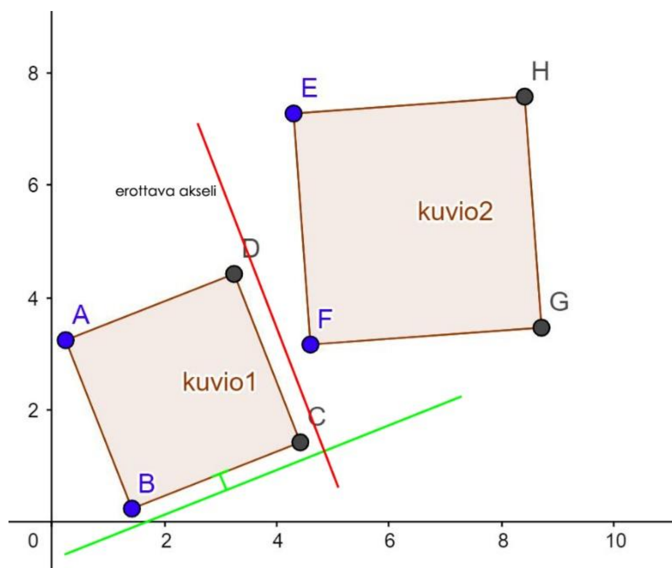
Kuva 2. Oriented Bounding Box.

Iso ero AABB:hen verrattuna on, että OBB ei muutu, vaikka sen sisällä oleva muoto kallistuisi, koska OBB kallistuu sen mukana. OBB pysyy siis jatkuvasti saman kokoisena. Esimerkiksi jos videopelissä pelattava hahmo on ympäröity AABB:llä. Hahmon seistessä AABB olisi korkea suorakulmio. Hahmon kaatuessa AABB muuttaisi kokoaan koko kaatumisen ajan kasvamalla leveyttä ja madaltumalla. Jos käytettäisiin OBB:tä, hahmon kaatuessa OBB kallistuisi hahmon mukana eikä muuttaisi muotoaan. OBB:llä on siis mahdollista luoda niin sanotusti tiukempi raja-arvolaatikko. (Madhav 2018a.)

OBB:n ongelma on kuitenkin, että se tarvitsee paljon muistia. Yleensä OBB ilmaistaan ohjelmoinnissa keskipisteellä, kaltevuusmatriisilla ja kolmella puolimittaisella särmällä kuvaamaan x-, y- ja z-pituuksia. Kaksiulotteisessa ohjelmassa särmiä olisi vain kaksi kuvaamassa x- ja y-pituuksia. Muitakin vaihtoehtoja on, esimerkiksi suorakulmio voidaan esittää tasoina tai vektoreina. Ensimmäisenä mainittu on näistä kuitenkin kaikkein vähiten muistia vievä vaihtoehto ja siksi käytetyin. OBB:stä saisi kevyemmän, jos kaltevuusmatriisin sijasta käyttäisi kvaternioita tai Eulerin lauseeseen perustuvia arvoja. Ne kuitenkin tulisi törmäyksentunnistamista varten muuttaa takaisin kaltevuusmatriisiksi. (Ericson 2005b, 101.)

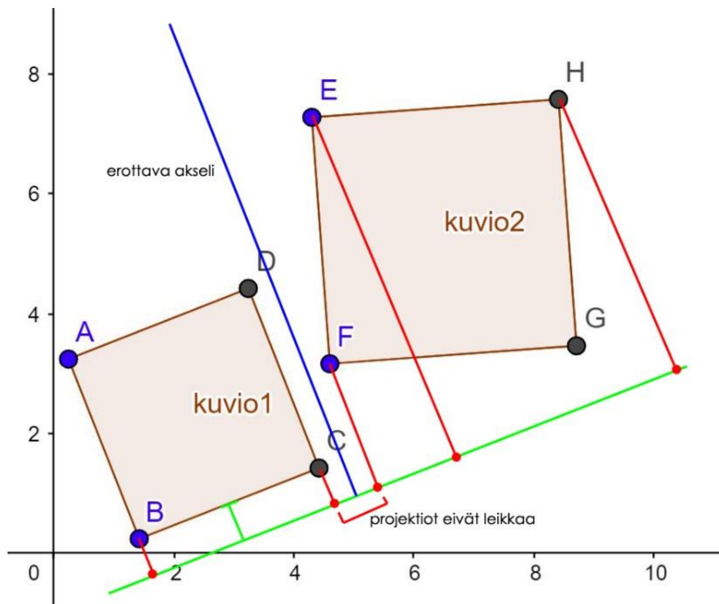
2.3.1 OBB – OBB törmäyksentunnistus ja Separating Axis Theorem (SAT)

Kahden kallistuneen monikulmion törmäyksen voi tehokkaasti kokeilla käyttämällä erottavan akselin lausetta, engl. *Separating Axis Theorem*, jonka lyhenne on SAT. Lauseen mukaan kuperat monikulmiot eivät leikkaa, jos niiden välissä on akseli, jota kumpikaan monikulmioista ei leikkaa, eli erottava akseli. Monikulmioiden tulee olla kuperia.



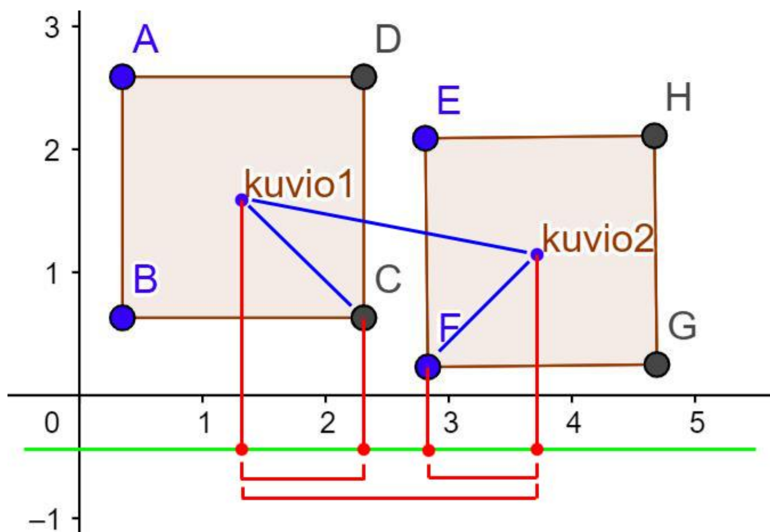
Kuva 3. Separating Axis

Erottava akseli etsitään monikulmioiden kulmapisteiden projektioilla monikulmion sivujen kanssa samansuuntaisille suorille. Monikulmiot eivät leikkaa, jos kaikista kulmapisteistä voidaan tehdä projektio jollekin monikulmion sivun kanssa samansuuntaiselle suoralle siten, että monikulmion n_1 kulmapisteiden projektiot eivät leikkaa monikulmion n_2 projektioita. (Szauer 2017a.)



Kuva 4. SAT projektiot

SAT-koetta voidaan huomattavasti nopeuttaa symmetrisillä monikulmioilla. Silloin niille voidaan laskea keskipiste, jolloin tarvittavien koelaskujen määrää voidaan huomattavasti vähentää. Symmetrisissä monikulmioissa voidaan laskea monikulmion sivun kanssa yhdensuuntaisella akselilla molempien vertailtavien monikulmioiden keskipisteiden etäisyys ja etäisyydet keskipisteestä toista kuviota lähimpään kulmapisteeseen. Jos monikulmioiden keskipisteen ja kulmapisteen etäisyyksien summa on pienempi kuin keskipisteiden etäisyys, monikulmiot eivät leikkaa. Kuitenkin jos SAT-koe tehdään epäsymmetriselle monikulmiolle, tulee kokeessa laskea projektiot kaikista kulmapisteistä kaikille sivujen kanssa yhdensuuntaisille suorille, kunnes erottava akseli löytyy. (Ericson 2005c, 158.)



Kuva 5. Symmetriset SAT projektiot

Kuvaesimerkissä 5 kuvataan, kuinka symmetristen monikulmioiden keskipiste - kulmapiste ja keskipiste - keskipiste -projektioita vertaamalla voidaan todeta, että monikulmiot eivät leikkaa toisiaan, koska keskipiste - kulmapiste -projektioiden etäisyyksien summa on pienempi kuin keskipiste - keskipiste -projektioiden etäisyys.

2.3.2 Projektion laskeminen

Monikulmion n_1 pisteiden p_1 ja p_2 kautta kulkevan suoran $|AB|$ yhtälö:

$$|AB| = m_a(x - x_1) + y_1$$

Suoran kulmakerroin m_a lasketaan seuraavasti:

$$m_a = \frac{y_2 - y_1}{x_2 - x_1}$$

Projektio saadaan laskemalla projektoitavan pisteen p_3 kautta kulkevalle suoralle yhtälö, jonka kulmakerroin on suoran $|AB|$ kulmakertoimen vastaluku.

$$|AB|_n = -\frac{1}{m_a}(x - x_3) + y_3$$

Suorien $|AB|$ ja $|AB|_n$ leikkauspiste on pisteen p_3 projektio.

$$m_a(x - x_1) + y_1 = -\frac{1}{m_a}(x - x_3) + y_3$$

Esimerkki:

Olkoon monikulmion n pisteet $p_1(3,7)$ ja $p_2(5,8)$ ja niiden läpi kulkeva suora $|AB|$. Olkoon projektoitava piste $p_3(4,4)$.

Suoran kulmakerroin:

$$m_a = \frac{8 - 7}{5 - 3}$$

$$m_a = \frac{1}{2}$$

$$m_a = 0{,}5$$

Suoranyhtälö:

$$|AB| = 0{,}5(x - 3) + 7$$

Suoran $|AB|$ normaalin $|AB|_n$ yhtälö on:

$$|AB|_n = -\frac{1}{0{,}5}(x-4) + 4$$

Pisteen p_3 projektio p_n suoralle $|AB|_n$ on $|AB|$ leikkauspiste suoran$|AB|_n$ kanssa:

$$0{,}5(x-3) + 7 = -\frac{1}{0{,}5}(x-4) + 4$$

$$0{,}5x - 1{,}5 + 7 = -2x + 8 + 4$$

$$0{,}5x + 2x = 8 + 4 + 1{,}5 - 7$$

$$2{,}5x = 13{,}5 - 7$$

$$2{,}5x = 6{,}5$$

$$x = 2{,}6$$

Y-arvon laskeminen tapahtuu sijoittamalla x-arvo toiseen alkuperäisistä yhtälöistä.

$$y = 0{,}5(2{,}6 - 3) + 7$$

$$y = 0{,}5(-0{,}4) + 7$$

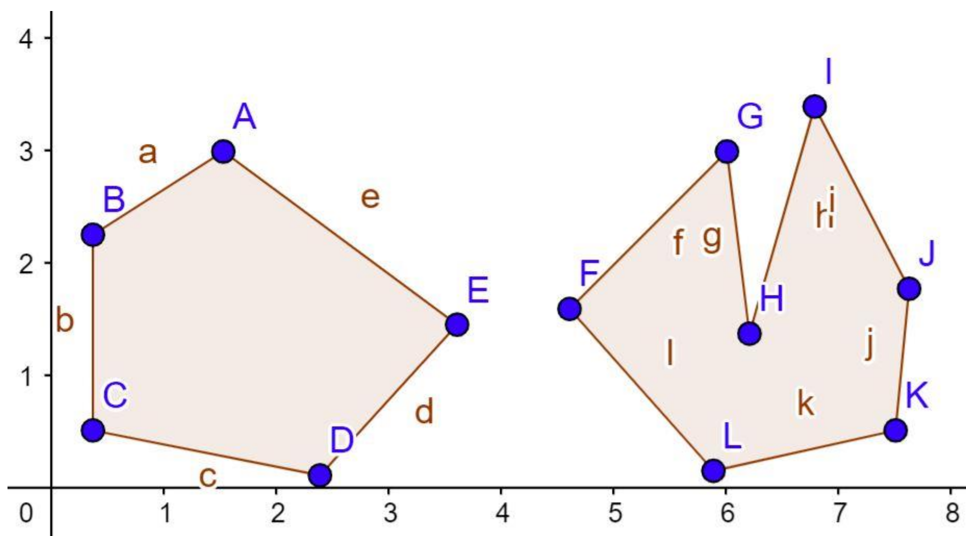
$$y = -0{,}2 + 7$$

$$y = 6{,}8$$

Projektio p_n on $(2.6, 6.8)$.

2.4 Kupera monikulmio

Vaikka suorakulmiot ja ympyrät ovat usein riittäviä törmäyksentunnistukseen, välillä tarvitaan monimutkaisempia muotoja. Nelikulmion sijasta voidaan käyttää monikulmioita, joissa ei ole rajoitusta kulmien määrässä. Kuitenkin mitä enemmän kulmia, sitä enemmän muistia törmäyksentunnistaminen vie.



Kuva 6. Kupera ja kovera monikulmio

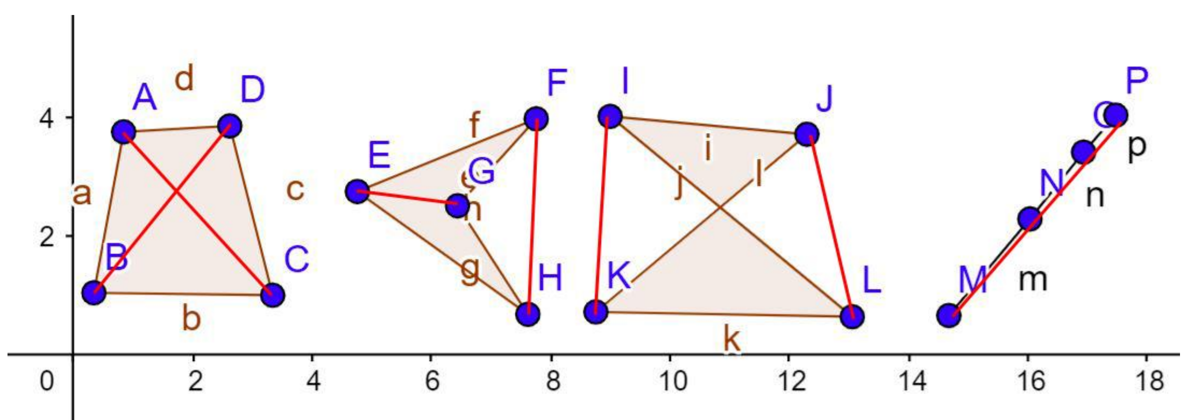
Kuperalla monikulmiolla tarkoitetaan monikulmiota, jonka jokainen sisäkulma on alle 180 astetta. (Madhav 2018b.)

Monikulmio esitetään vektoreina, jotka ovat järjestyksessä myötä- tai vastapäivään. Vektoreita tulee olla vähintään kolme ja kaikki ovat yhdistetty seuraavaan vektoriin. Yksittäinen vektori voi olla kupera, mutta monikulmio, jossa on vain kuperia vektoreita, ei välttämättä ole kupera. Esimerkiksi monikulmiosta tulee kovera, jos siinä on neljä peräkkäistä 90 asteen kulmaa ja sen sivut ovat eri pituisia. Silloin kaikki vektorit ovat kuperia, mutta monikulmio on kovera. Vektori on kovera, jos vektoriin yhdistyvien molempien sivujen sisäkulma on yli 180 astetta. Monikulmio poikkeuksetta kovera, jos siinä on kovera vektori. Kolmio ja suorakulmio ovat ainoita monikulmioita, jotka ovat poikkeuksetta kuperia. (Ericson 2005d, 56-58.)

2.4.1 Monikulmion kuperuuden tunnistaminen

Törmäyksentunnistus koverilla monikulmioilla, on huomattavasti raskaampaa ja monimutkaisempaa kuin kuperien monikulmioiden. Tietokoneohjelma ei kuitenkaan suoraan tiedä, onko kyseessä kupera vai kovera monikulmio. Silloin monikulmion tyyppi täytyy kyetä tunnistamaan.

Nelikulmioille varma koe kuperuuden tunnistamiseksi voidaan tehdä lävistäjillä. Nelikulmio on kupera, jos vastakkaiset vektorit voidaan yhdistää lävistäjällä, joka on kokonaan nelikulmion sisällä.



Kuva 7. Nelikulmion kuperuuden selvittäminen (mukaillen Ericson 2005e.)

Kuvan 7 nelikulmiossa ABCD lävistäjät muodostavat kolmiot ABD, BCD, ACB ja ACD. Voidaan huomata, että esimerkiksi ABD-kolmion kärkipiste A on BD-jänteestä vastakkaisella puolella verrattuna BCD-kolmion C-kärkipisteeseen. Sama voidaan todeta ACB- ja

ACD-kolmioista. Voidaan todeta, että nelikulmio ABCD on kupera. Sama ei kuitenkaan käy toteen nelikulmioissa EFGH, IJKL tai MNOP. Matemaattisesti tämä voidaan esittää nelikulmiolle ABCD ehdoilla, jos: $(BD\ x\ BA) * (DB\ x\ BC) < 0$ ja $(AC\ x\ AD) * (AC\ x\ AB) < 0$, nelikulmio on kupera. (Ericson 2005e, 59-60.)

Yleisempi keino erottaa kupera ja kovera monikulmio, on tarkastaa monikulmion jokaisen vektorin sijainti suhteessa johonkin sivuun. Kaikenlaiset monikulmiot voidaan todeta kuperiksi, jos jokaisen sivun kohdalla voidaan todeta, että kaikki vektorit sijaitsevat samalla puolella sivua. Kuperan monikulmion voi myös tunnistaa vertailemalla sisäkulmia toisiinsa.

Kupera monikulmio voidaan myös tunnistaa tarkastamalla jokaisen sisäkulman aste. Useimmissa tapauksissa monikulmio voidaan todeta kuperaksi, jos jokainen sen kulmista on enintään 180 astetta. Algoritmi ei kuitenkaan tunnista itseään leikkaavia monikulmioita ja esimerkiksi väittäisi pentagrammia kuperaksi monikulmioksi.

Myös kulman suunnasta on mahdollista päätellä monikulmion kuperuus. Useimmissa tapauksissa monikulmio voidaan todeta kuperaksi, jos kahta peräkkäistä kulmaa verrataan toisiinsa ja todetaan, että ne kääntyvät samaan suuntaan. (Ericson 2005f, 61.)

2.4.2 Kuperan monikulmion törmäyksentunnistus

Satunnaisten monikulmioiden keskinäisen leikkauksen voi tunnistaa käyttämällä erottavan akselin lausetta. Molempien monikulmioiden tulee läpäistä kuperuuskoe ennen SAT-koetta. (Szauer 2017a.)

Törmäyksentunnistamisen voi myös suorittaa kevyemmin käyttämällä GJK-algoritmia. GJK-algoritmi on nimetty kehittäjiensä nimien mukaan, eli E. G. Gilbert, D. W. Johnson ja S. S. Keerthi. GJK-algoritmi hyödyntää geometrista Minkowskin summa -menetelmää laskettuna erotuksella summan sijaan. Menetelmässä lasketaan monikulmion B pisteiden erotus monikulmion A pisteisiin. Minkowskin summan tulos kahdesta kuperasta monikulmiosta on kupera monikulmio itsessään. Monikulmiot leikkaavat toisiaan, jos origo, eli perusakselin nollapiste, sijaitsee Minkowskin summan sisäpuolella. (Gregory 2017a.)

2.5 Tila-avaruuden jakaminen

Tila-avaruutta jakamalla törmäyksentunnistusta voidaan huomattavasti tehostaa vähentämällä operaatioiden määrää. Oletetaan, että pelimaailmassa on 500 monikulmiota. Olisi todella raskasta ajaa 500^{500} SAT-koetta jokaisen monikulmion mahdollisen törmäyksen tunnistamiseksi. Törmäyksen voisi jättää tunnistamatta, jos voidaan helposti ja vähällä

suorituskyvyllä määrittää, että monikulmio A sijaitsee aivan toisella puolella tila-avaruutta kuin monikulmio B.

Törmäyksentunnistaminen voidaan jakaa laajaan, väli- ja kapeaan vaiheeseen. Laajassa vaiheessa luodaan suuria AABB:itä, joilla tunnistetaan mahdolliset ryhmät, joiden alueilla törmäyksiä tapahtuu. Välivaiheessa luodaan tarkempia raja-arvohierarkioita, kuten OBB- ja AABB-puita. Kapeassa vaiheessa tehdään raskaammat ja tarkkuutta vaativat törmäyksentunnistusoperaatiot yksittäisille raja-arvolaatikkopareille. (Gregory 2017b.)

Erilaisia raja-arvolaatikko -hierarkioita voidaan luoda esimerkiksi puuhierarkioilla. OBB:t voidaan yhdistää toisiinsa luomalla kahden OBB:n ympärille yksi isompi OBB, joka taas yhdistetään kolmannen OBB:n kanssa. Vastaavasti samanlainen operaatio voidaan suorittaa AABB-olioille. Näin törmäyksentunnistus voidaan eristää vain tietyille alueille tila-avaruudessa ja suorittaa törmäyksentunnistus-operaatioita vain tunnistettavasti toistensa kanssa tekemisissä oleville olioille. (Ericson 2005g.)

AABB-törmäyksiä pystytään tehokkaasti vähentämään Sweep and prune -algoritmilla. Koska AABB - AABB -törmäys tapahtuu vain, jos nelikulmioiden x- ja y-arvot risteävät. Algoritmilla tarkastetaan ensin vain toinen akseleista ja suoritetaan törmäyksentunnistus-operaatiot raja-arvolaatikoille, jotka leikkaavat valitulla akselilla. Sitten sama operaatio suoritetaan toiselle akselille. Näin saadaan tarvittavien operaatioiden määrä vähennettyä mahdollisesti puoleen verrattuna siihen, että suoritettaisiin törmäyksentunnistus jokaiselle mahdolliselle AABB-parille. Tätä algoritmia käytetään muun muassa törmäyksentunnistamisen laajassa vaiheessa. (Madhav 2018c.)

3 Kehityssuunnitelma

Shadows Collide kehitetään Agile-menetelmällä. Projektinhallintaan käytetään GitHub-palvelun Kanban-taulua. Projekti kehitetään kahdessa erillisessä vaiheessa, jotka ovat suunnittelu- ja kehitysvaiheet.

3.1 Suunnittelu

Projektin ensimmäisessä vaiheessa ohjelmisto suunnitellaan. Suunnittelun tuloksena tuotetaan vuo- ja luokkakaaviot, jotka lisätään tämän raportin loppuun liitteinä. Vaiheessa luodaan myös matemaattinen suunnitelma ja tehtäväkortit projektinhallintaan.

Vuokaavio sisältää kuvauksen ohjelman aloituksesta, jolloin käyttäjä syöttää vektorilistan ohjelmalle aina lopetukseen asti, jolloin käyttäjä saa palautusarvon törmäyksentunnistusoperaatiosta. Kaaviossa kuvataan myös geometriatyypin tulkitseminen ja törmäyksentunnistusoperaation valitseminen.

Luokkakaaviossa kuvataan ohjelman pääluokka, joka sisältää käyttöliittymän päätoiminnallisuudet, kuten datan sisäänoton ja päätöksen siitä, minkä tyyppinen monikulmio on kyseessä. Kaavioon tehdään myös törmäyksentunnistuskoeluokkien kuvaukset ja erityyppisien hyväksyttyjen geometrioiden luokkakuvaukset.

Tässä vaiheessa suoritetaan myös selvitys matemaattisille toimenpiteille. Tietoperustan perusteella suunnitellaan matemaattiset kaavat, joita ohjelman kehityksessä tarvitaan. Tästä ei synny erillistä jaettavaa dokumenttia. Matemaattiset operaatiot löytyvät ohjelman lähdekoodista.

Vaiheen lopuksi GitHubin projektinhallintaan luodaan tehtäväkortit suunnitteluvaiheessa syntyneiden dokumenttien perusteella. Tästä eteenpäin voidaan valvoa ohjelmoinnin edistymistä.

3.2 Kehitys

Toisessa vaiheessa suoritetaan itse ohjelmointi. Ohjelmointi suoritetaan projektinhallintaan luotujen tehtävien perusteella. Jos kehityksen aikana huomataan lisätarpeita, lisätään niistä kortit tehtävänhallintaan, jotta kaikki projektissa tehdyt tehtävät olisivat näkyvillä.

3.3 Teknologiat ja työkalut

Ohjelman kehityskielenä on C++. Kielen valinta perustuu sen yleisyyteen tietokonepelien ohjelmoinnissa sekä sen tehokkaaseen suorituskykyyn.

Kielen etuina on, että se on olio-ohjelmointikieli, jolloin voidaan tuottaa paremmin DRY-ohjelmointia. DRY tarkoittaa ”*don’t repeat yourself*”. Ohjelmoinnista voidaan tehdä selkeämpää ja helpommin ylläpidettävää. C++:lla on myös laaja valikoima kirjastoja. Sen suosion ja iän myötä kielelle on kehitetty useita kirjastoja, jotka auttavat tehokkaassa ohjelmoinnissa. C++ tukee osoittimia, jotka mahdollistavat paremman muistinhallinnan. C++ -koodi käännetään ennen ohjelman ajamista matalan tason koodiksi. Hyvin tehokkaan muistinhallinnan ja matalan tason kääntämisen takia C++ on erittäin nopea. (Hackr.io 2019.)

Tietokonepelit tarvitsevat salamannopeaa suorituskykyä. Monet törmäyksentunnistusoperaatiot, fysiikkasimulaatiot sekä grafiikan renderöinti täytyy tapahtua millisekunneissa. Tietokonepelin kuvan päivitystaajuuden olla vähintään 60Hz, eli kuvan tulee päivittyä 60 kertaa sekunnissa. Tämä tarkoittaa siis, että suuri määrä laskentaa tulee tapahtua alle 16,6 millisekunnissa, jotta tietokonepelin kuva päivittyisi sujuvasti 60 kertaa sekunnissa. Monet pelaajat odottavat peleiltä jopa 180 hertsin päivitystaajuutta, joka jättää laskennalle aikaa vain 5,5 millisekuntia. C++:n mahdollisuudet muistin optimointiin tekevät kielestä hyvän työkalun tietokonepelien kehittämiseen. Vaikka nykypäivänä on tarjolla monta muuta hyvää kieltä pelinkehitykseen esimerkiksi C#, Java, Kotlin, Rust ja JavaScript, ovat teolliset 3D-pelimoottorit kirjoitettu pääosin C:llä ja C++:lla. Siksi jokaisen vakavasti tietokonepelejä kehittävien ohjelmoijien olisi hyvä osata C++ ohjelmointia. (Gregory 2017c.) C++ kielenkehittäjä Bjarne Stroustrup on koonnut listan moderneista peleistä ja pelimoottoreista, jotka ovat ohjelmoitu C++:lla. Listaa voi tarkastella osoitteessa: http://www.stroustrup.com/GamesListOfCpp.pdf. Huomattavaa listassa on, että siitä löytyvät nykypäivän käytetyimmät pelimoottorit ja myydyimmät pelit. Listan sisällöllä voidaan vain painottaa pelikehittäjän tarvetta osata C++ ohjelmointia.

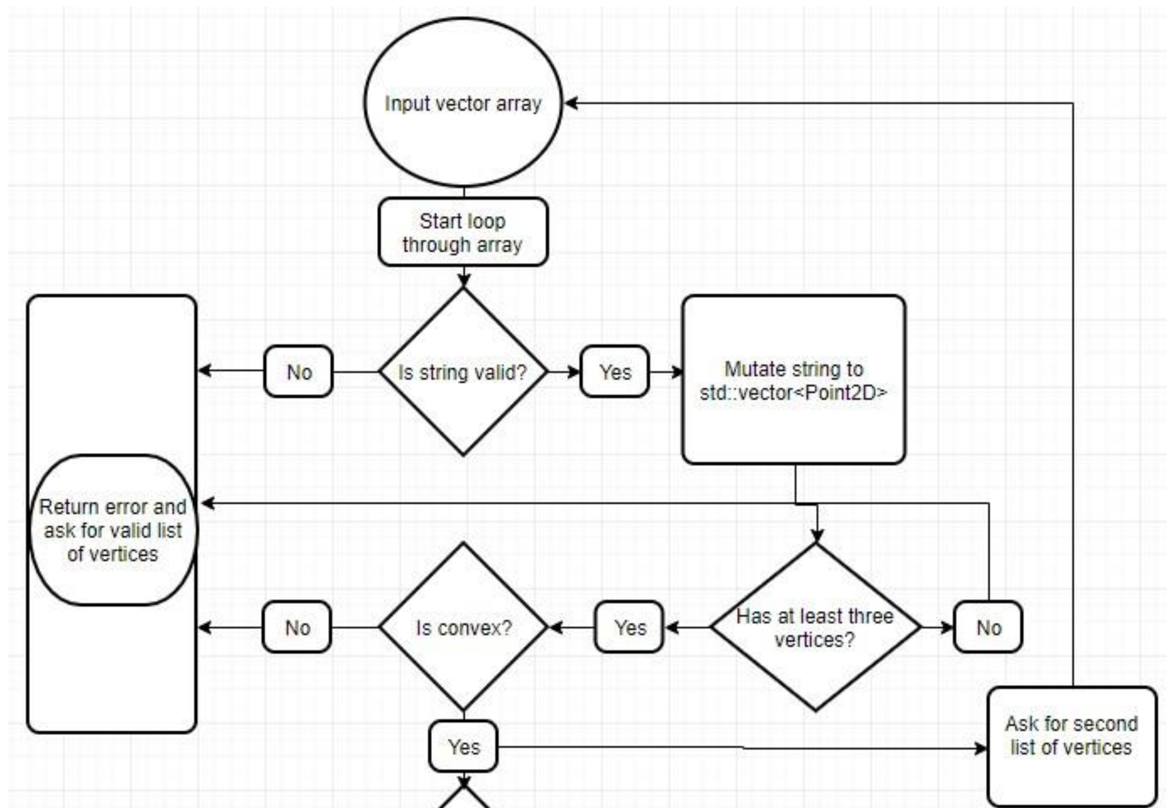
Käytän tekstieditorina Microsoft Visual Studio 2017 IDE:tä. Visual Studion C++ virheenkorjaus- ja monitorointityökalut ovat todella edistyneitä ja käytännöllisiä, jotka auttavat tehokkaassa ohjelmoinnissa. Visual Studio on myös ollut vuosia päätyökaluni ohjelmoinnissa, joten sitä ei ole syytä vaihtaa. Lähdekoodi säilytetään Microsoftin GitHub-versionhallinnassa.

4 Shadows Collide arkkitehtuuri

4.1 Ohjelman elinkaari

Liitteessä 1, eli vuokaaviossa esitetään ohjelman elinkaari. Ohjelman käynnistyessä käyttäjälle esitetään ohjeet vektorilistan oikeanlaista formaattia varten ja pyydetään syöttämään validi lista vektoreita. Vektorilistoja pyydetään käyttäjältä yhteensä kaksi kappaletta. Molemmilla kerroilla vektorilista validoidaan ennen ohjelman jatkumista.

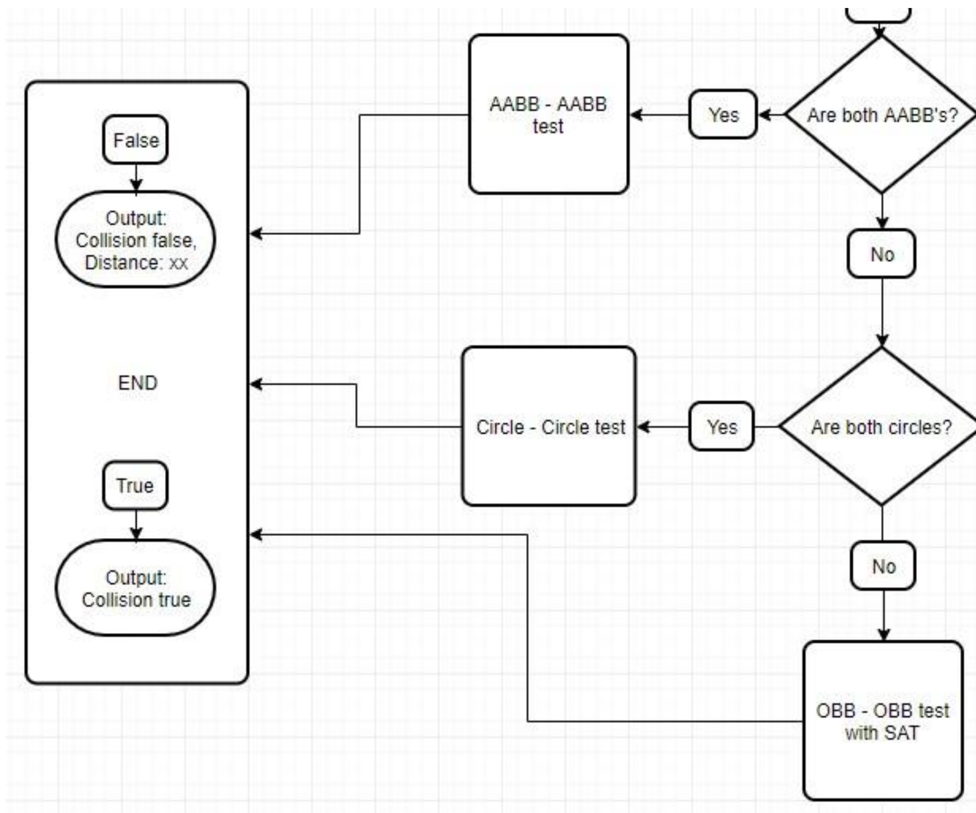
Validoinnissa tarkastetaan käyttäjän syöttämän vektorilistan formaatti ja vektorien määrä. Vektoreille tehdään kuperuustesti, jonka jälkeen listalle ajetaan monikulmion tyypin tunnistustestit.



Kuva 8. Vektorilistan syöttäminen ja validointi

Jos molemmat monikulmiot ovat AABB:ita, tehdään AABB – AABB törmäystesti. AABB-algoritmi käyttää vähiten muistia ja suoritetaan siksi ensimmäisenä. Ympyräntunnistustesti tehdään, jos toinen monikulmioista ei ole AABB. Ympyrä – ympyrä törmäystesti tehdään, kun molemmat monikulmiot ovat ympyröitä. Jos vaatimukset AABB – AABB tai ympyrä – ympyrä -testeille eivät täyty, suoritetaan OBB – OBB törmäystesti, joka käyttää SAT-algoritmia. OBB – OBB törmäystesti on viimeisenä, koska se on ohjelman vakiotörmäystesti. SAT-algoritmilla voidaan tunnistaa kaikkien kuperien monikulmioiden törmäykset, mutta

SAT-algoritmi kahdelle AABB:lle olisi vain muistin hukkaan heittämistä. Siksi SAT-testi jätetään viimeiseksi vaihtoehdoksi, jotta ohjelmassa voidaan varmistaa mahdollisimman tehokas suorituskyky.

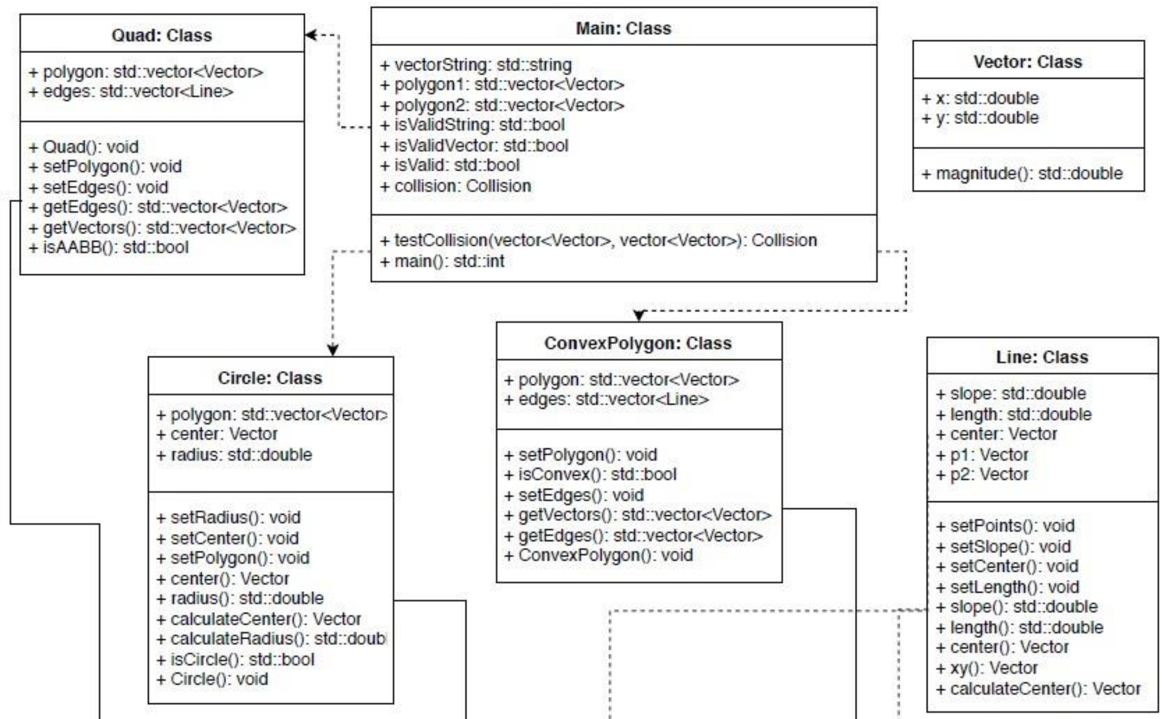


Kuva 9. Monikulmion tunnistaminen, törmäystestit ja ohjelman lopettaminen

Törmäystestien jälkeen käyttäjälle palautetaan testin tulos boolean-arvona. Törmäyksen lopputuloksen lisäksi käyttäjälle kerrotaan monikulmioiden etäisyys toisistaan. Ohjelma päättyy ja ilmoittaa käyttäjälle, että ikkunan voi sulkea CTRL+C komennolla.

4.2 Ohjelman luokat

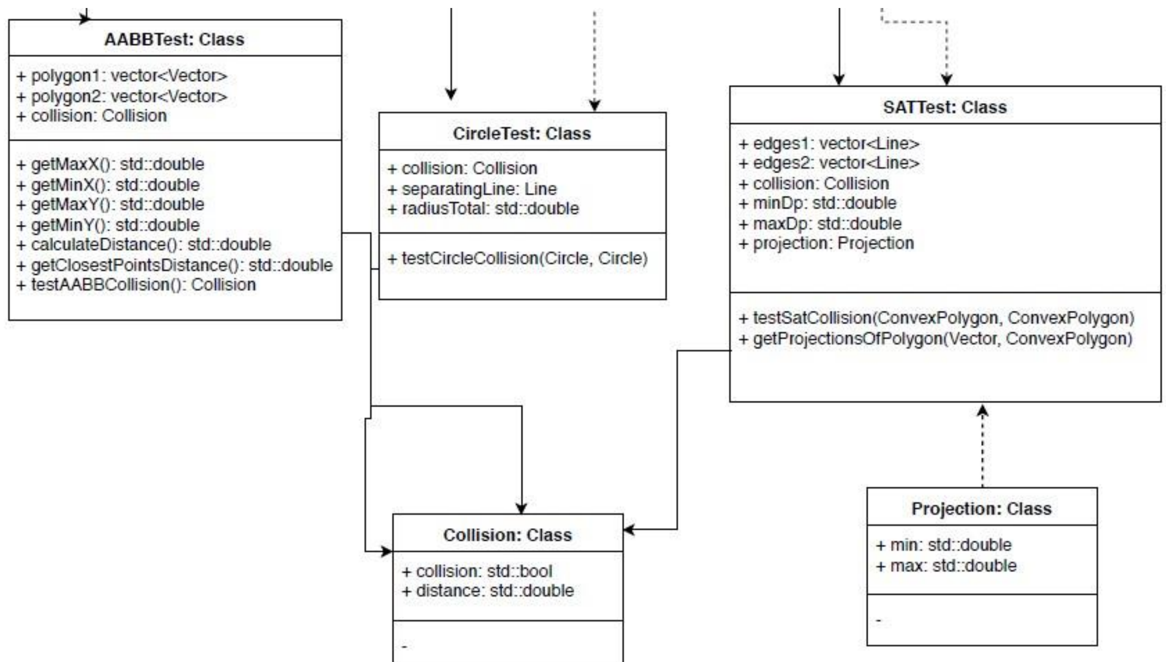
Liitteessä 2 esitetään ohjelman luokkakaavio. Ohjelman luokat voidaan rajata pääluokkaan, geometrisiin luokkiin, törmäystesteihin ja apuluokkiin. Luokkien muuttujat ovat tyypeiltään C++ standard-kirjaston datatyyppejä tai ohjelmaan luotujen luokkien olioita. Luokkakaaviossa standard-kirjaston muuttujat ovat merkitty std::-etuliitteellä.



Kuva 10. Ohjelman pääluokka ja geometriset luokat

Ohjelman pääluokka sisältää main-funktion ja törmäystestien kutsumisen, eli ohjelman pääasiallisen toimintalogiikan. Main-funktiossa luodaan ohjelman käyttöliittymä ja suoritetaan syötteen validointi. Validointiin käytetään while-silmukkaa, joka toistuu, kunnes käyttäjä on syöttänyt validin vektorilistan. Törmäystesti päätetään if-ehdoilla kutsumalla esimerkiksi Quad-luokan isAABB-funktiota. Funktiolle syötetään parametrinä vektorilista ja funktio palauttaa tosi tai epätosi arvon. Vektorilistasta luodaan edellä mainitussa esimerkissä Quad-olio, kun isAABB-funktio palauttaa molemmilla vektorilistoilla positiivisen boolean-arvon.

Geometrisiä luokkia ovat Quad, Circle, ConvexPolygon, Vector ja Line. Jokainen luokista sisältää geometrialle oleellisten arvojen asettamisen kuten monikulmion pisteet, keskipiste, säde, kulmakerroin, suoran pituus ja sivut. Geometrialuokista Quad, Circle ja ConvexPolygon sisältävät funktiot monikulmion tyypin tunnistamiselle. Luokkakaavion selkeyden vuoksi Vector-luokan relaatioita ei ole erikseen eritelty kaavion jokaiselle luokalle, koska Vector-luokka on ohjelmassa hyvin yleisesti käytetty geometrinen luokka.



Kuva 11. Törmäystesti- ja apuluokat

Törmäystestiluokat sisältävät algoritmille olennaiset muuttujat ja funktiot, kuten vektoreiden maksimi- ja minimisijaintien ja projektioiden laskemisen. SAT-algoritmi käyttää Projection-apuluokkaa projektioiden hallintaa varten. Törmäystestiluokkien testCollision-funktiot ottavat vastaan parametreina kaksi testille tarkoitetun tyyppistä oliota. Esimerkiksi SAT-Test-luokan testSATCollision-funktio ottaa kaksi muuttujaa, jotka ovat ConvexPolygon-luokan olioita. Testiluokat palauttavat Collision-apuluokan olion, joka sisältää törmäyksen lopputuloksen ja etäisyyden.

Shadows Collide lähdekoodi on saatavilla osoitteesta: https://github.com/ollilappalainen/shadows-collide.

5 Pohdinta

Projektin tavoitteena oli luoda ohjelma, joka sisältää algoritmit erilaisten monikulmioiden törmäyksen tunnistamiseen ja luo perustan pelimoottorin fysiikkamoottorille. Ohjelmaa oli tarkoitus voida kokeilla komentorivikäyttöliittymän kautta ja saada tieto syötettyjen monikulmioiden törmäyksestä boolean-arvona, sekä syötettyjen monikulmioiden etäisyys koordinaatistolla. Projektisuunnitelmassa aikataulutettiin ohjelmointi alkamaan viikolla 14. Käytin aikaa ohjelmointiin kuukauden ja ohjelma saavutti asetetut tavoitteet aikataulussa kuukautta myöhemmin. Kuten kaikissa sovelluskehitysprojekteissa, tässäkin projektissa ilmeni ennalta suunnittelemattomia ongelmia hyvästä suunnittelusta huolimatta. Suurimmat ongelmat ilmenivät matemaattisten funktioiden kääntämisessä algoritmeiksi.

Olin tehnyt hyvinkin kattavan matemaattisen selvityksen tarvittavista funktioista ja teoriasta, jonka oli tarkoitus olla riittävä. Suunnittelun puutteet kuitenkin ilmenivät esimerkiksi ohjelmoidessani SAT- algoritmia. Esiselvityksessä olin selvittänyt, että lauseen mukainen erottava akseli on olemassa, jos jonkin monikulmion sivun kanssa yhdensuuntaiselle suoralle tehdyt projektiot monikulmioiden vektoreista eivät risteä. Minulla oli tiedossa, kuinka pisteen kohtisuora projektio suoralle tehdään ja sen kääntäminen algoritmiksi ei ollut ongelma. Projektioiden jälkeen huomasin, että käytössä oli projektioarvot, mutta etukäteen oli jäänyt suunnittelematta, kuinka tunnistaa väli kahden monikulmion projektioissa. Ongelma pysäytti sovelluksen kehityksen muutamaksi päiväksi. Luin paljon lähdeaineistoa ja tutkin tarjolla olevia avoimia lähdekoodeja. Jokaisella tietenkin oli oma lähestymistapansa, kuten minullakin, koska en ollut seurannut mitään ohjetta, vaan pyrkinyt ratkaisemaan ongelmat itse ja tehnyt ohjelmasta omanlaisen. Lopulta ratkaisu ongelmaan oli projektion laskeminen akselin nollapisteen ja vektorin sijainnin pistetulolla. Tällöin tuloksena oli yksi verrannollinen arvo, eikä x- ja y-koordinaatti, jota on vaikeampi vertailla orientoituneella suoralla. Pistetulo tarkoittaa kahden vektorin välistä tuloa. Tulo on lukuarvo, ja on sitä suurempi, mitä kauempana toinen vektori toisesta on. Pistetulosta otin talteen monikulmioiden pienimmät ja suurimmat arvot ja vertailin AABB-algoritmin tapaan näiden leikkaamista.

Monikulmion tunnistaminen ympyräksi ilmeni myös yllättävän ongelmalliseksi. Alkuperäinen ajatus oli laskea sivujen kulmat ja pituudet. Monikulmio olisi ympyrä, jos sivujen kulmat ja pituudet olisivat aina samat. Projektin rajauksissa on määritelty vain, että kaikki kahdeksan tai enemmän vektoreita sisältävät monikulmiot tulkitaan ympyröiksi. Rajaus ei vastaa todellisuuden tarpeita, koska monikulmio voi olla täysin sattumanvarainen monikulmio tai soikea. Järkeviä vaihtoehtoja ratkaisuksi ongelmaan oli laskea ympyrän säde jokaisesta reunapisteestä tai jokaisen vektorin kulman summa ja tutkia olisiko summa 360-astetta. Valitsin ympyrän säteiden laskemisen, koska se tarvitsee vähemmän muistia kuin

kulmien laskeminen. Säteitä laskiessa prosessin voi lopettaa välittömästi, jos säde on eri kuin edellinen laskettu säde. Säteiden arvoissa ilmeni myös muutaman sadasosan heittoja, jotka johtivat kymmenesosiin pyöristettyinä yhden desimaalin heittoon. Nämä heitot johtuivat pisteiden sijaintien epätarkkuudesta. Esimerkiksi double-tyyppinen luku asetettuna 3,65 arvolla tallentuu arvona 3,649999999. Myös käyttämäni kokeiluvektorit eivät olleet tuhannesosan tarkkoja. Välttääkseni virheellisen ympyrän tulkinnan tai tarpeettoman tarkan tulkinnan, lisäsin yhden desimaalin virherajan säteiden pituuksien vertailuun. Eli ohjelma hyväksyy arvot 0,4; 0,5 ja 0,6; jos edellinen säde on ollut esimerkiksi 0,5.

Kuperan monikulmion tunnistamisessa vertailin yhtä vektoria kerrallaan johonkin monikulmion sivuista. Eli käytössä oli kolme vektoria, joista voi muodostaa kaksi suoraa. Kertomalla ristiin näiden kahden suoran vektorit ja vähentämällä ne toisistaan saadaan lukuarvo, joka on suoraan verrannollinen siihen, onko laskussa käytetty kolmas vektori suoran negatiivisella vai positiivisella puolella. Kyseessä on monikulmio, jos kaikissa vertailuissa kaikki pisteet olivat joko 0 ja <0 tai 0 ja >0.

Nelikulmion tunnistaminen onnistui suunnitellulla tavalla. Nelikulmio tunnistetaan laskemalla jokaisen sivun pituus. Jos pituudet ovat samat, jatketaan ohjelmassa kulmakertoimien laskemiseen. Monikulmio on saman suuntainen perusakselin kanssa, jos kulmakertoimet ovat 0.

Törmäyksentunnistusalgoritmin valinta ei tuottanut ongelmia, kun kaikille monikulmioille oli tehty algoritmit tyypin tunnistamiseksi. Käyttäjän syöttämälle vektorilistalle ajetaan ensin nelikulmion ja ympyrän tunnistusalgoritmit. Kuperuuden tunnistusta ei ajeta enää uudestaan, koska vektorit validoitiin jo syöttämisen yhteydessä. Jos syötetty monikulmio ei ole nelikulmio eikä ympyrä, käytetään vakio törmäyksentunnistusalgoritmina SAT-algoritmia.

Ohjelmoin jokaiselle monikulmiotyypille ja törmäyksentunnistusalgoritmille oman luokan, jota voidaan kutsua sisällyttämällä ohjelmaan halutun luokan header-tiedosto. Syntynyttä rajapintaa voi tulevaisuudessa vielä optimoida luomalla ohjelmalle oman nimiavaruuden. Siten ohjelman rajapinta olisi yksinkertaisemmin käytössä nimiavaruuden kautta.

Visual Studio osoittautui tehokkaaksi työkaluksi C++ -ohjelmoinnissa. Esimerkiksi SAT-algoritmia en olisi saanut kahdessa päivässä toimimaan ilman Visual Studion virheenkorjaustyökaluja. Ongelmana oli, että sain projektioista vääriä arvoja toistuvista korjausyrityksistä huolimatta. Ongelma ratkesi virheenkorjaustyökalujen ansiosta ja onnistuin jäljittämään ongelman lähteeksi Line-luokan, jossa olin unohtanut kokonaan asettaa annetut

pisteet suoran arvoksi. Siksi jokainen Line-olio sai pisteiden arvoksi maksimi double-arvon ja projektiot olivat vääriä.

Ohjelmaan jäi tiedettyjä puutteita, kuten parempi muistin optimointi. Tällä hetkellä monikulmion tyypin tunnistaminen ja törmäystestin ajaminen vie noin 2,2 millisekuntia, mutta optimoinnilla siitä voisi saada vielä paljon tehokkaamman. Ohjelmassa on jonkun verran turhia arvojen kopioimisia, jotka voi korjata käyttämällä viiteparametreja ja osoittimia. Ohjelman signaaleja ei myöskään käsitellä asianmukaisesti.

Käyttäjälle ohjelman lopettaminen näyttää toimivan oikein ja kuten pitää, eli ohjelma sammuu CTRL+C -komennolla. Todellisuudessa tätä signaalia ei tällä hetkellä ohjelmassa käsitellä. CTRL+C -lopetussignaali aiheuttaa käyttäjälle näkymättömän virheen käsittelemättömästä poikkeuksesta ja lopettaa ohjelman. AABB-törmäyksentunnistusalgoritmi ei tunnista saman arvoisia vektoreita törmäyksenä. Tämä tietenkin on suunnittelijan tulkinnasta riippuvaista, että halutaanko saman arvoiset pisteet tulkita törmäävinä pisteinä. Ympyrä- ja SAT-algoritmit kuitenkin pitävät saman arvoisia vektoreita törmäävinä, joten yhdenmukaisuuden takia AABB-algoritmi tulee korjata vastaavanlaiseksi. Ympyrä- ja AABB-algoritmien palauttama etäisyys ei sisällä negatiivista arvoa. Etäisyyden voisi korjata näyttämään vain nollaa suuremmat etäisyydet. Eli törmäyksen tapahtuessa etäisyys olisi aina 0. En ole keksinyt negatiiviselle etäisyydelle mitään tarpeellista käyttötarkoitusta. Ohjelma on tällä hetkellä käännettävissä vain Windowsille Visual Studiolla, koska ohjelman C++ kääntäjä on Visual Studion oma. Siksi ohjelmaan voisi lisätä CMake-tiedoston, jotta koodista tulisi toimiva myös Linux- ja Mac-käyttöjärjestelmille ja muille kehitystyökaluille, jotka tukevat CMake-kääntäjää.

Julkaisin linkin lähdekoodiin Reddit-keskustelupalstan cpp_questions -keskutelualueella ja pyysin kokeneempia ohjelmoijia arvioimaan koodini. Sain monilta todella hyvää palautetta ja tarpeellisia korjausehdotuksia koodin sekä ohjelman tehokkuuden parantamiseksi. Jatkokehityksenä lisään korjausehdotukset GitHub-projektin ongelmiin, jotta voin kehittää ohjelmaa niiden mukaisesti. Aloitan myös renderöintimoottorin kehittämisen OpenGL-, Vulkan- tai DirectX -3D-grafiikkarajapinnalla. Renderöintimoottorin myötä ohjelman parametrit oletettavasti muuttuvat riippuen grafiikkarajapinnan tarvitsemista parametreista. Lopullinen tavoite on ohjelmoida kokonainen toimiva tietokonepeli. Projektin aikana koin itse kehittyväni huomattavasti ohjelmoijana. Lähtökohtana oli lähes täysi tietämättömyys aiheesta ja C++ -kielestä. Onnistuin kuitenkin omaksumaan tyydyttävästi uuden konseptin, kielen ja matemaattisen ohjelmoinnin sekä luomaan toimivan lopputuloksen. Koen myös olevani valmiimpi jatkokehittämään Shadows Collidesta täysimittaisen fysiikkamoottorin.

Lähteet

Bourke, P. 1992. Circles and spheres. Luettavissa: http://paulbourke.net/geometry/circlesphere/. Luettu: 7.4.2019

Ericson C. 2005a. Real-Time Collision Detection, s. 1. Elsevier Inc.

Ericson C. 2005b. Real-Time Collision Detection, s. 101. Elsevier Inc.

Ericson C. 2005c. Real-Time Collision Detection, s. 158. Elsevier Inc.

Ericson C. 2005d. Real-Time Collision Detection, s. 56-58. Elsevier Inc.

Ericson C. 2005e. Real-Time Collision Detection, s. 59-60. Elsevier Inc.

Ericson C. 2005f. Real-Time Collision Detection, s. 61. Elsevier Inc.

Ericson C. 2005g. Real-Time Collision Detection, s. 261-263. Elsevier Inc.

Grant P. 2005. Physics for Game Programmers, s. 139. aPress.

Gregory J. 2017a. Game Engine Architecture, Second Edition, 12.3.5.5 Detecting Convex Collisions: The GJK Algorithm. Oreilly.

Gregory J. 2017b. Game Engine Architecture, Second Edition, 12.3.6.2 Spatial Partitioning. Oreilly.

Gregory J. 2017c. Game Engine Architecture, Second Edition, Preface to the first edition. Oreilly.

Hackr.io 2019. C++ Language: Features, Uses, Applications & Advantages. Luettavissa: https://hackr.io/blog/features-uses-applications-of-c-plus-plus-language. Luettu: 25.4.2019.

Learn OpenGL 2019. Collision Detection. Luettavissa: https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection. Luettu: 10.9.2018.

Madhav S. 2018a. Game Programming in C++: Creating 3D Games, First Edition. Chapter 10: Collision Detection, Oriented Bounding Boxes. Oreilly.

Madhav S. 2018b. Game Programming in C++: Creating 3D Games, First Edition. Chapter 10: Collision Detection, Convex Polygons. Oreilly.

Madhav S. 2018c. Game Programming in C++: Creating 3D Games, First Edition. Chapter 10: Collision Detection, Testing Box Collisions in PhysWorld. Oreilly.

MDN 2019. 2D collision detection. Luettavissa: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection. Luettu: 30.3.2019

OPH 2019. Janan pituus koordinaatistossa. Luettavissa: http://www02.oph.fi/etalukio/opiskelumodulit/manmath/koordina/janapit/janapit.html. Luettu: 20.4.2019

Szauer G. 2017a. Game Physics Cookbook. 5. 2D Collisions, Separating Axis Theorem. Oreilly.

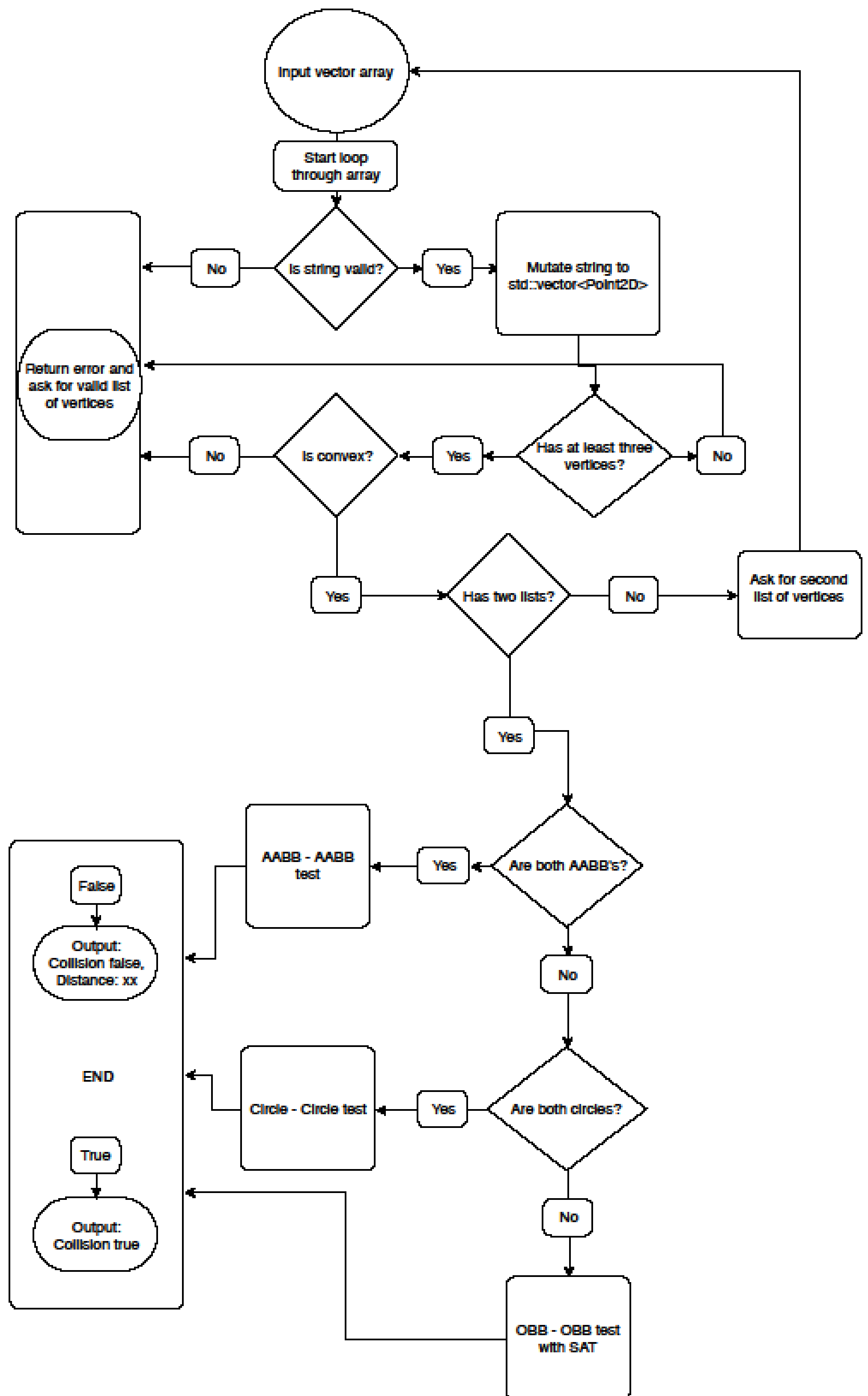
Thompson, J. 2019a. Collision Detection, Introduction. Luettavissa: http://www.jeffreythompson.org/collision-detection/index.php. Luettu: 30.3.2019

Thompson, J. 2019b. Collision Detection, Circle/Circle. Luettavissa: http://www.jeffreythompson.org/collision-detection/circle-circle.php. Luettu: 20.4.2019

Toptal 2019. Video Game Physics Tutorial - Part II: Collision Detection for Solid Objects. Luet-tavissa: https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects. Luettu: 20.9.2018

Liitteet

Liite 1. Vuokaavio



Liite 2. Luokkakaavio