

Patrik Tirkkonen

3D-LIDARIN SIMULOINTI UNITY3D-YMPÄRISTÖSSÄ

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Marraskuu 2021

TIIVISTELMÄ

Patrik Tirkkonen: 3D-LIDARin simulointi Unity3D-ympäristössä

Kandidaatintyö

Tampereen yliopisto

Automaatiotekniikan tutkinto-ohjelma

Marraskuu 2021

Työssä kehitetään kolmiulotteista maailmaa skannaava Light Detection and Ranging, eli LIDAR-järjestelmän simulaatio Unity3D-ympäristössä. Työ jakautuu kahteen osaan. Ensin käsitellään teoriaosuus LIDARin sekä Unity3D:n oleellisista ominaisuuksista liittyen tähän työhön. Teoriaosuuden jälkeen esitellään kehitetty kokonaisuus.

LIDAR on etäisyydenmittausjärjestelmä, joka perustuu valon kulkuajan mittaamiseen. LIDAR lähettää laserlähteestään valosäteen, joka heijastuu takaisin osuessaan esteeseen. Esteen etäisyys lähteestä lasketaan mittaamalla valon kuluttama aika matkaan. Tätä kutsutaan valon kulkuajaksi. LIDARilla on erilaisia käyttökohteita missä sitä käytetään, mutta tässä työssä keskitytään vain etäisyyden mittaamiseen ja kohteen paikallistamiseen sivusuunnassa lähietäisyydeltä.

Unity3D on Unityn ympäristö, jossa käyttäjä kehittää kolmiulotteista näkymää. Unity3D yhdistetään yleensä pelien kehittämiseen, mutta sitä käytetään myös videoiden ja simulaatioiden luomiseen. Unity3D:n kanssa käytetään Unityn fysiikkamoottoria. Fysiikkamoottorilla tässä työssä toteutetaan Raycast-osumien sijainti peliobjektien osumalaatikossa sekä objektien liikehdintä. Raycastin tehtävä tässä työssä on simuloida lasersädettä.

Simulaatio koostuu yksinkertaisesta näkymästä, johon lisätään esivalmiiksi luodut LIDAR-peliobjektit. LIDAR skannaa ympärillä olevaa näkymää, ja tallentaa objekteihin osuvista Raycasteista tarpeellisen tiedon tietorakenteeseen sekä pistepilvitiedostoon. Pistepilvitiedostoon tallennetaan osumien x-, y- ja z-koordinaatit LIDAR-peliobjektin paikallisessa koordinaatistossa. Tämän tiedoston sisältö syötetään 3D-mallinnusohjelmaan, jossa muodostetaan kuvaa pisteiden sijainneista.

LIDARin skannaustapa mitata ympärillä olevan maailman kohteita vaihtelee. Yksi tapa toteuttaa LIDAR on skannaava mekaaninen LIDAR. Tämä skannaa ympäristöä pyörien ja vaihtuen atsimuuttikulman astetta. Mekaanisilla LIDAReilla on yleensä täysi 360 asteen näkökenttä laitteen ympäriltä. Toinen LIDAR-järjestelmätyyppi on puolijohde-LIDAR. Niiden näkökenttä ei ole yhtä laaja kuin mekaanisella LIDARilla, mutta niissä ei ole isoja liikkuvia komponentteja.

Toteutettu simulaatio sisältää kaksi erilaista LIDAR-järjestelmää: skannaava LIDAR ja Flash LIDAR. Ensimmäinen peliobjekti skannaa ympärillä olevaa maailmaa paloittain skannaavan mekaanisen LIDARin toimintatapaa mukaillen, jonka atsimuuttikulmaa päivitetään joka funktio-

kutsu. Korkeus mitataan kerroksittain yhden funktiokutsun aikana sijoittamalla Raycastit osoittamaan tiettyjen asteiden välein pystysuunnassa. Toinen toteutettu peliobjekti on Flash LIDAR -peliobjekti. Se ottaa joka funktiokutsulla kuvan isosta alueesta suoraan LIDAR-peliobjektin edestä. Kehitettyjen peliobjektien parametrien arvoja on mahdollista päivittää kesken ajon, ja saaduista koordinaattipisteistä voi muodostaa totuudenmukaista ja luotettavaa pistepilvikuvaa.

Avainsanat: LIDAR, Unity3D, Raycast

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

ALKUSANAT

Unityn käytön opettelu on ollut jo hetken aikaa kiinnostuksen kohteena henkilökohtaiseen käyttöön. Tämän työn aihe mahdollisti erittäin mielenkiintoisen alun tälle.

Haluaisin kiittää ohjaajaa Jukka Yrjänäistä hyvästä ja mielenkiintoisesta kandityön aiheesta, sekä tuesta ja avusta mitä hän antoi tämän työn aikana sen loppuun saamiseksi. Kiitokset myös perheelleni tuesta ja kannustuksesta mitä he ovat antaneet työn aikana.

Tampereella 10.11.2021

Patrik Tirkkonen

SISÄLLYSLUETTELO

1.	JOHDANTO	1
2.	LIDAR	2
2.1	Toimintaperiaate.....	2
2.2	Time-of-Flight	5
2.3	Kaupallisesti saatavat LIDARit.....	7
3.	UNITY3D-YMPÄRISTÖ	9
3.1	Unityn käyttökohteet	9
3.2	Unity-ympäristö.....	10
3.3	Kappaleen transformaatio Unityssä	11
3.4	Unityn pelisilmukka	12
4.	SIMULAATION TOTEUTUS.....	15
4.1	Skannaavan LIDARin toteutus.....	16
4.2	Flash-LIDAR.....	18
4.3	Simulaation ulostulo	20
4.4	Yksikkötestaus.....	21
5.	SIMULOINNIN TULOKSET	23
6.	YHTEENVETO.....	30
	LÄHTEET	32
	LIITTEET	34

LYHENTEET JA MERKINNÄT

2D	Kaksiulotteinen
3D	Kolmiulotteinen
c	Valonnopeus tyhjiössä, vakioarvo 299 792 458 m/s
C#	Ohjelmointikieli (C-sharp)
CW	engl. Continuous-Wave, Jatkuva-aaltainen
Flash LIDAR	LIDAR-järjestelmä, joka kuvaa laajaa näkökenttää skannaamisen sijasta
FoV	engl. Field-of-View, näkökenttä
Hz	Hertsi, taajuuden yksikkö
Laser	engl. Light Amplification by Stimulated Emission of Radiation, laite, joka tuottaa koherenttia valoa
LIDAR	engl. Light Detection and Ranging, järjestelmä, joka mittaa etäisyyksiä valon kulkuajan avulla
MASER	engl. Microwave Amplification by Stimulated Emission of Radiation, laite, joka luo koherenttia sähkömagneettista aaltoa
MEMS	engl. Microelectromechanical systems, mikrosysteemi
OPA	engl. Optical Phased Array, optinen vaiheryhmä
PB	engl. Pulse-Based, pulsseihin perustuva
RGB	Red-Green-Blue arvot.
ROS	Robot Operating System, työkalu robottiapplikaatioiden luomiseen
t	Kulunut aika sekunteina
ToF	engl. Time-of-Flight, signaalin kulku aika
W	Watti, tehon yksikkö

1. JOHDANTO

Vuonna 1958 fyysikot Arthur Leonard Schawlow ja Charles Hard Townes esittivät yhdessä tehdyn tutkielman ”Infrared and Optical Masers” [1]. Maser (engl. Microwave Amplification by Stimulated Emission of Radiation), joka tuottaa mikroaalloja korkeampi-taajuuksista säteilyä, kutsutaan termillä *Laser* (engl. Light Amplification by Stimulated Emission of Radiation) [2]. Laser on tärkeä osa tämän työn aihetta, Light Detection and Ranging eli LIDAR-järjestelmää, jota simuloidaan tässä työssä Unity3D-ympäristössä.

LIDAReita käytetään useaan eri tarkoitukseen. Yleisimpiä ovat itseajavat autot, robotiikka, ilmakuvakartoitus sekä ilmakehän ominaisuuksien mittaukset [3]. LIDARien avulla tutkijat kykenevät tutkimaan ympäristöjä tarkasti ja joustavasti [4]. Tässä työssä keskitytään robotiikassa ja itseajavissa autoissa käytettäviin sivusuuntaan osoittaviin LIDAR-järjestelmiin.

Unity3D on Unityn ympäristö luoda kolmiulotteista sisältöä. Se on pelimoottori kaiken tasoille ohjelmoijille. Sitä käytetään videopelien tekemisen lisäksi simulaatioiden toteuttamiseen, jos esimerkiksi oikean maailman laitteita ei ole saatavilla tai sen toteuttaminen on vaarallista [5]. Unity3D on aloittelijaystävällinen pelimoottori, sillä sen käyttöliittymää on helppo käyttää, sekä sen dokumentaatio on helposti saatavilla [6].

Tässä työssä käsitellään perustietoa LIDARista, Unity3D:n käytöstä sekä siitä, miten Unity3D-pelimoottoria voidaan käyttää reaali maailman LIDAR-järjestelmän simuloinnissa. Työn tarkoituksena on selvittää, miten ideaalitulanteen LIDAR-järjestelmän simulointi tapahtuu hyödyntäen pelimoottorin ominaisuuksia. Toteutamme kaksi LIDAR-peliobjektia. Ensimmäinen on ympäri paloittain skannaava LIDAR ja toinen on koko näkökentästä yhtäaikaaisesti kuvan ottava Flash LIDAR.

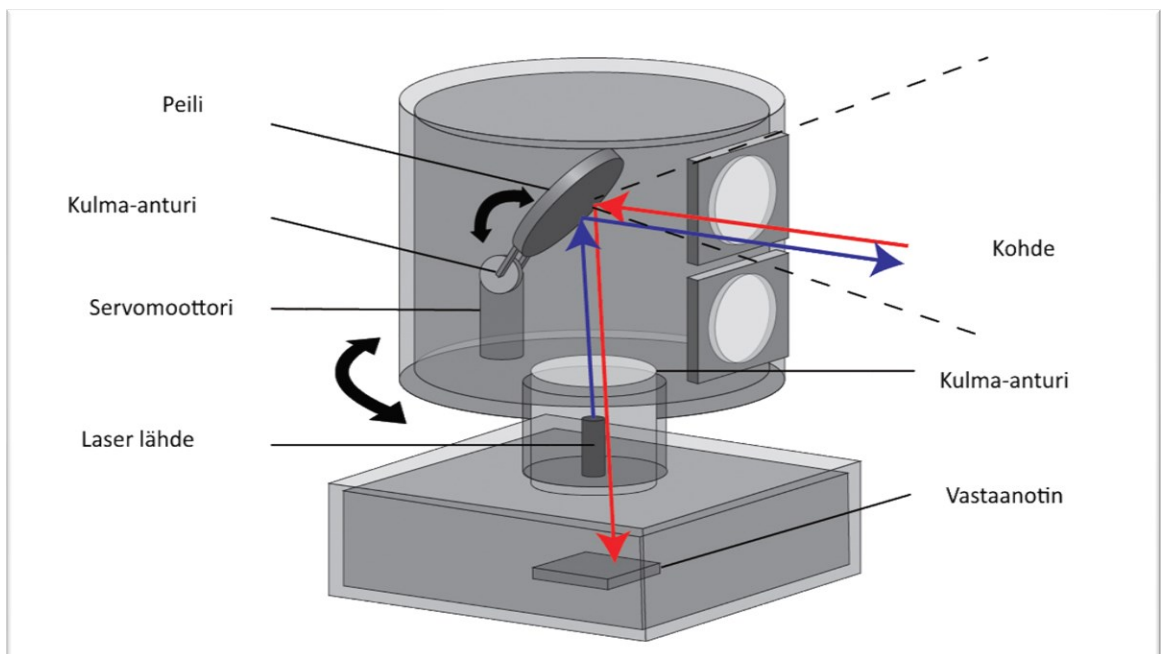
Luvussa 2 käsitellään LIDARin toimintaperiaatetta. Siinä selitetään, mikä LIDAR on ja miten se toimii, sekä esitetään kaksi ympäri skannaavaa LIDARia. Unityä ja sen tärkeimpiä ominaisuuksia liittyen tähän työhön käsitellään luvussa 3. Itse kirjoitettu logiikka, sekä Unityn tuottama ohjelmistokoodi kuvataan luvussa 4. Ohjelman rakenne kuvataan myös tässä luvussa. Luvussa 5 tarkastellaan toteutettujen LIDAR-peliobjektien toimintaa paikallaan sekä liikkeessä. Lopuksi viimeisessä luvussa on yhteenveto, jossa pohditaan saatuja tuloksia ja sitä, miten tätä työtä voisi laajentaa.

2. LIDAR

LIDAR on lasersädettä käyttävä tekniikka, jolla mitataan etäisyyksiä. Kohteen etäisyyksien mittaamisen lisäksi on mahdollista saada muutakin informaatiota, kuten esimerkiksi tuulen nopeuden tai ilmakehän lämpötilan [7]. LIDAR pystyy mittaamaan suoraan ylös- tai alaspäin, mutta näillä on eri käyttökohteet. Tässä työssä keskitytään sivullepäin mittaavaan LIDAR-järjestelmään, minkä käyttötarkoitus on tunnistaa lähellä olevat kohteet.

2.1 Toimintaperiaate

LIDAR-järjestelmässä on useita eri osia, jotka toimivat yhdessä muodostaen kokonaisuuden. Tässä työssä pääosissa ovat laserlähde ja laserin vastaanotin. Kuvassa 1 näytetään, mitä osia ympäri skannaavassa LIDAR-järjestelmässä on.



Kuva 1: Ympäri skannaavan mekaanisen LIDARin osia, muokattu lähteestä [8].

Laserlähde on laite, joka emittoi koherenttia valoa. Valo on koherenttia silloin, kun valonsäteiden aallonpituudet ovat samoja, eli valo on monokromaattinen sekä värähtely tapahtuu samaan suuntaan ja samalla taajuudella. Lasersäteet pysyvät tiheästi yhdessä pitkänkin matkan ajan, eli laserilla on mahdollista osoittaa tarkasti kaukanakin oleviin kohteisiin. [9]

Lasersäteitä lähettävät laitteet luokitellaan tiettyyn laserluokkaan sen mukaan, miten vaarallista niiden lähettämä säteily on. Laserluokkaan 1 kuuluvat laserlaitteet ovat käytännössä harmittomia ihmisille, kun taas laserluokkaan 4 kuuluvat laitteet ovat vaarallisia ja pystyvät helposti aiheuttamaan silmä- tai ihovammoja. [10]

LIDARin laservalon aallonpituus on noin $1,5 \mu\text{m}$ riittävän alhaisella teholla, mikä on luokiteltavissa ihmisen silmille turvalliseksi [11]. Tämä säde on vaarallista, jos siihen altistuu pidemmäksi aikaa. LIDARin lähettämien lyhyiden pulssien ajan säde on kuitenkin turvallista. Velodyne on LIDAREiden valmistukseen keskittyvä yritys, joka valmistaa vain 905 nm:n aallonpituisia lasereita lähettäviä LIDAREita vedoten parempaan kosteuden läpäisykykyyn, pienempään virrankulutukseen sekä komponenttien saatavuuteen [12].

Laservalo, jota LIDAR emittoi, voi olla diskreetistä pulsseina lähtevää tai jatkuva-aikaista. Riippuu LIDARin käyttötarpeesta kumpaa tapaa käytetään. Tästä kerrotaan lisää aluvussa 2.2.

LIDAR-järjestelmän vastaanotin ottaa vastaan esteistä heijastuneet lasersäteet. Vastaanotin muuttaa saadun datansa suhteessa omaan sijaintiinsa pallokoordinaateiksi eli säde on r , korkeus on ω ja atsimuutti on α [13]. Pallokoordinaateista muuttaminen karteesisen kolmiulotteiseen koordinaattijärjestelmään onnistuu tietokoneen avulla reaaliaikaisesti kaavoilla (1), (2) ja (3)

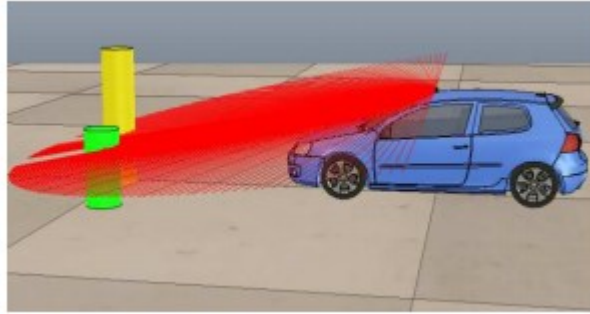
$$X = r \cos(\omega) \sin(\alpha) \quad (1)$$

$$Y = r \cos(\omega) \cos(\alpha) \quad (2)$$

$$Z = r \sin(\omega). \quad (3)$$

Kaavoissa r on etäisyys, ω on kulma vertikaalisessa kulmassa ja α on atsimuuttikulma eli kulma horisontaalisessa suunnassa. X , Y ja Z ovat osuapisteiden sijainti laskettuna karteesisessä koordinaatistossa. LIDARin tapauksessa täytyy huomioida, että LIDAR sijaitsee kolmiulotteisen avaruuden origossa eli pisteissä $(0,0,0)$.

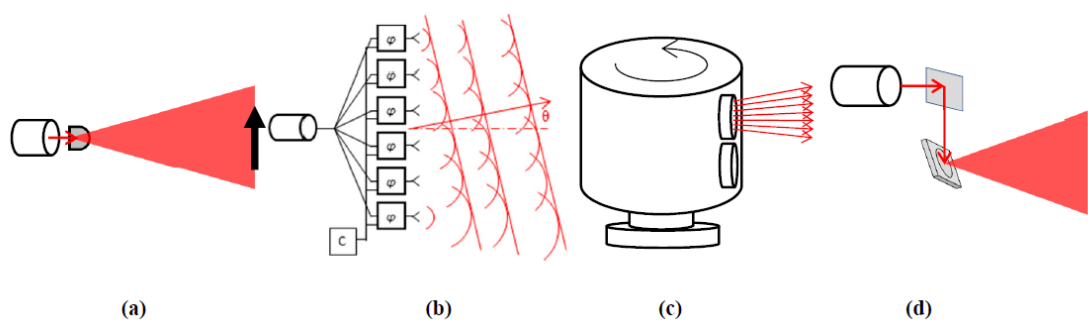
2D-LIDAR mittaa etäisyyksiä kahdessa ulottuvuudessa x - ja y -akseleilla. Sensorin asentoa muuttamalla 2D-LIDAR pystyy ottamaan korkeuden mukaan tekemällä mittauksia ala- ja yläviistoon mahdollistaen käytön esimerkiksi itseajavissa autoissa [14]. Osoituskulman vaihtelulla tarkastellaan matalalla olevia esteitä ilman jatkuvaa kolmiulotteista kuvaa. Kuvassa 2 näytetään visualisointia 2D-LIDARin näkökentästä. Punainen väri havainnollistaa LIDARin näkökenttää.



Kuva 2: Alaviistoon osoittava 2D-LIDAR tunnistaa matalat esteet [14].

2D-LIDARilla pystytään muodostamaan kolmiulotteista dataa vaihtelemalla tämän osoituskulmaa, mutta tämä ei ole tehokas valinta. 3D-LIDAR toimii kuten 2D-LIDAR, mutta laserlähteestä lähtee useampi lasersäde muodostaen usean kerroksen ottaen huomioon myös korkeuden eli z-koordinaatiston, ja laite pyörii yhtäaikaaisesti muodostaen kolmiulotteista kuvaa. 3D-LIDAR pystyy tehokkaammin tunnistamaan korkeussuunnan eri kohdissa olevat esteet verrattuna 2D-LIDARIin jättäen vähemmän sokeita kohtia mittaus tuloksiin. 3D-LIDAR soveltuu parhaiten tarkkuutta vaativiin mittauksiin sekä kartoitustehäviin [15].

LIDAReita on kahdenlaisia. On mekaanisia, jotka ovat kooltaan isoja ja joilla on laaja 360°:n näkökenttä horisontaalisessa suunnassa. Mekaanisten LIDAReiden kanssa on *puolijohde-LIDAR* (engl. Solid-state LIDAR). Puolijohde-LIDARissa ei ole isoja liikkuvia mekaanisia osia, vaan lasersäteen suunnan ohjaaminen onnistuu ilman mekaanisia laitteita, kuten moottoria [3]. Valonsädettä ohjataan puolijohde-LIDARissa kolmella eri menetelmällä. Näitä menetelmiä ovat mikrosysteemit (engl. Microelectromechanical systems, MEMS), optiset vaiheryhymät (engl. Optical Phased Array, OPA) ja Flash LIDAR, jonka toimintaperiaate on lähellä tavallista digitaalista kameraa. Flash LIDAR ottaa yhtäaikaisesti kuvan isolta näkökentältä [16,17]. Kuvassa 3 on näiden toimintatapaa havainnollistettu visuaalisesti.



Kuva 3: (a) Flash LIDAR (b) OPA LIDAR (c) LIDAR pyörivällä skannauksella (d) MEMS LIDAR, muokattu lähteestä [18].

Riippumatta siitä, mitä LIDARia käyttää, lopputuloksena saadaan mittaustuloksia osu-
makohteesta. Näillä osumapisteillä pystytään luomaan pistepilveä (engl. point cloud) tie-
tokoneen avulla. Pistepilvi on kokoelma pisteitä avaruudessa. Jokaisella pisteellä on
koordinaatit karteesisessa koordinaatistossa. LIDARin tapauksessa pistepilvi muodos-
tuu lähetetyn lasersäteen osumakohdasta esteeseen. Tämä muodostaa pisteitä kohteen
reunasta.

2.2 Time-of-Flight

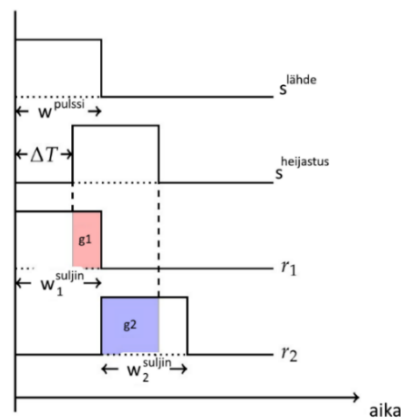
Mittaus, jossa objekti matkaa jonkin etäisyyden ja siitä mitataan siihen kulunut aika, kut-
sutaan *Time-of-Flightiksi* (ToF) eli kulkuajaksi. Mainittu objekti voi olla yksittäinen hiuk-
kanen tai aaltoa, joka lähtee laitteesta.

Nimensä mukaisesti LIDAR perustuu valon säteilyyn. LIDAR toimii siten, että laite lähet-
tää laservaloa, joka kohteeseen osuessaan heijastuu takaisin lähtölaitteeseen. Tästä mi-
tataan valon matkaan kulunut aika, jonka avulla saadaan kohteen ja LIDARin välinen
etäisyys d selville kaavalla (4)

$$d = \frac{1}{2}ct. \quad (4)$$

Tässä t on valon kuluttama aika törmätä kohteeseen ja heijastua sieltä takaisin. Valon-
nopeuden arvo on c . Tapa, miten t mitataan, vaihtelee mittaustyylin mukaan: säteileekö
laite valoa pulssimaisesti vai jatkuvana aaltona [19].

Pulssitaista kulkuajatekniikkaa (engl. Pulse-Based Time-of-Flight, PB-ToF) käyttävä
kamera mittaa etäisyyttä emittoimalla valoa pulsseina. Valopulssin osuessa esteeseen
tämä heijastuu takaisin ja kameran anturit vastaanottavat kyseisen pulssin. [19] Kuvassa
4 esitetään kahden sulkijan sisältävän PB-ToF-kameran toimintaperiaate.



Kuva 4: Pulssipohjaisen kameran ajoitukset, muokattu lähteestä [19].

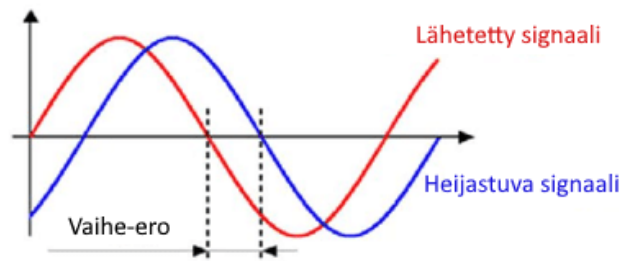
Kuvassa 4 aloitetaan ajanhetkestä 0, milloin lähtölaite emittoi signaalia $s^{\text{lähde}}$ pulssin w^{pulssi} ajan. Yhtäaikaisesti ensimmäinen suljin r_1 avautuu w_1^{suljin} ajaksi. Heijastuksesta anturiin saapuvan signaalin $s^{\text{heijastus}}$ pulssi on yhtä suuri kuin laitteesta emittoitu pulssi, mutta siinä huomioidaan etäisyyden tuottama aikaviive t . Kamerassa on aina vain toinen suljin auki. Heti kun ensimmäinen suljin r_1 sulkeutuu, avautuu toinen suljin r_2 ajanhetken w_2^{suljin} ajaksi. Näin laitteesta lähtenyt pulssi jakautuu kahdelle sulkijalle.

Nyt saadaan selville etäisyyden tuottama aikaviive t . Välivaiheet löytyvät lähteestä [19]. Sijoittamalla t yhtälöön (4) tulee etäisyyden d arvoksi yhtälö (5)

$$d = \frac{1}{2} c \omega \frac{g_2}{g_1 + g_2}, \quad (5)$$

missä c on valonnopeus, ω on lähetetyn valopulssin kesto ja g_2 ja g_1 ovat vastaanottiin saapunut signaali sulkimien ollessa auki.

Jatkuva-aaltoista kulkuakatekniikkaa (engl. Continuous-Wave Time-of-Flight, CW-ToF) käyttävä kamera lähettää jatkuvaa sinimuotoisella aallolla moduloitua signaalia. Esteeseen osuessaan aalto heijastuu takaisin ja vastaanotin vastaanottaa tämän signaalin. Tässä metodissa mitataan laitteesta lähetetyn ja esteestä heijastuvan signaalin välistä vaihe-eroa [20]. Kuvassa 5 näytetään tämä visuaalisesti.



Kuva 5: CW-ToF perustuu vaihe-eron laskemiseen, muokattu lähteestä [21].

Kun tiedetään, millä taajuudella signaalia moduloidaan sekä signaalien välinen vaihe-ero, pystytään ratkaisemaan etäisyys kaavalla (4) sijoittamalla aika t . Sijoitus on tapahtunut kaavassa (6)

$$d = \frac{c}{(4\pi f)} \phi. \quad (6)$$

Tarkemmat välivaiheet löytyvät lähteestä [22]. Kaavassa (6) c on valonnopeus, f on moduloidun signaalin taajuus ja ϕ on lähtevän ja vastaanotetun valosignaalin vaihe-ero. Näillä saadaan laskettua etäisyys syötelaiteen ja esteen välillä.

2.3 Kaupallisesti saatavat LIDARit

Tässä aluvussa esitellään kaksi esimerkkiä LIDAR-sensoreista, Velodyne Puck ja RoboSense Bpearl. Nämä LIDARit laitetaan tukevasti pintaan kiinni, josta ne skannaavat 3D-ympäristöä pyörien. LIDARit sopivat hyvin esimerkiksi auton katolle aseteltavaksi. Kuvassa 6 näytetään, miltä Velodyne Puck ja RoboSense Bpearl näyttävät. Taulukossa 1 esitetään tietoja näistä kahdesta LIDAR-sensorista.



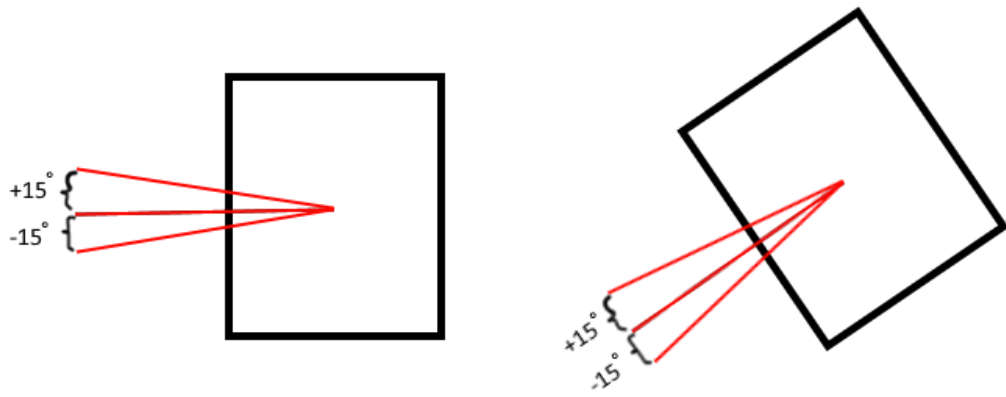
Kuva 6: Vasemmalla on Velodyne Puck [23] ja oikealla on RoboSense Bpearl [24].

Taulukko 1: Velodyne Puckin ja RoboSense Bpearlin vertailudata. Puckin data on lähteestä [13] ja Bpearlin data on lähteestä [24].

	Velodyne Puck	RoboSense Bpearl
Kanavien lukumäärä	16	32
Vertikaalinen FoV	30°	90°
Horisontaalinen FoV	360°	360°
Kulmaresoluutio vertikaalinen	2,0°	2,81°
Kulmaresoluutio horisontaalinen	0,1° – 0,4°	0,2° tai 0,4°
Kiertonopeus	5 Hz – 20 Hz (300–1 200 RPM)	10 Hz tai 20 Hz (600 tai 1 200 RPM)
Mittausetäisyys	100 m	100 m
Mittauspisteitä sekunnissa	288 000	576 000

Molemmat esitetyt LIDARit ovat mekaanisia LIDAReita. Ne pyörivät ympäri 360° muodostaen 3D-kuvaa. Kanavien lukumäärä ilmaisee lähetettyjen lasersäteiden määrää.

Vertikaalinen FoV (engl. Field of view) on se näkökenttä, mitä LIDAR näkee pystysuunnassa ja kanavien välit on asetettu vertikaalisen kulmaresoluution välein. Esimerkiksi Velodyne Puck näkee -15° ja $+15^\circ$ välillä, eli yhteensä 30° . Kuva 7 havainnollistaa tätä.



Kuva 7: Näkökenttä sivusta katsottuna. Vaikka laitetta kallistaa, se kattaa yhtä suuren näkökentän pystysuunnassa laitteen suhteen.

Laitteen näkökentän osoittamissuuntaa voi vaihtaa kallistamalla laitetta. Kulmaresoluutio horisontaalisessa suunnassa (engl. Angular resolution) on laitteen kyky tunnistaa pieniä kohteita tarkasti. Mitä pienempi kulmaresoluutio, sitä paremmin laite tunnistaa pieniä kohteita lähettämällä säteensä tiheämmin. Tämä luku on riippuvainen LIDARin kiertonopeudesta (engl. Rotation rate).

Mittausetäisyys on valmistajan esittämä lukema, miten kaukana olevia kohteita tämä laite pystyy hyvissä olosuhteissa mittaamaan. Tämä maksimietäisyys kuitenkin vaihtelee riippuen sääolosuhteista, miten hyvin valo heijastuu kohteesta, sekä muista ennakoimattomista tapahtumista [24].

3. UNITY3D-YMPÄRISTÖ

Unity3D on Unity Technologiesin kehittämä pelimoottori saatavilla Windows-, Mac- ja Linux-käyttöjärjestelmän omaaville tietokoneille [25]. Unityn avulla pystytään kehittämään 2D- ja 3D-ympäristöjä helposti. On olemassa sekä ilmaisversio, että maksullinen versio Unitystä. Ilmaisversio on tarkoitettu kehittäjille, opiskelijoille ja pienille yrityksille, keillä on tuloja alle \$100 000 vuodessa liitettävissä Unityn käyttöön. [26]

Unity3D:ssä on myös hyvin dokumentoitu fysiikkamoottori [27]. Fysiikkamoottori on tietokoneohjelmisto, mitä käytetään ilmiöiden simuloimiseen, missä kappaleiden fysikaaliset ominaisuudet otetaan huomioon. [28]

Tässä työssä fysiikkamoottoria tarvitaan LIDARin lasersäteiden simuloimiseen, sen osu-
makohdan tarkastamiseen sekä objektin räsynukkemaisen (engl. Ragdoll) liikuttamiseen. Unityssä on valmiina hyvin dokumentoidut luokat ja metodit tähän tarkoitukseen. Tämän työn tekemisessä on käytetty Unity 2020.3.0f1 ilmaista versiota. Jatkossa Unityllä tarkoitetaan Unity3D:tä.

3.1 Unityn käyttökohteet

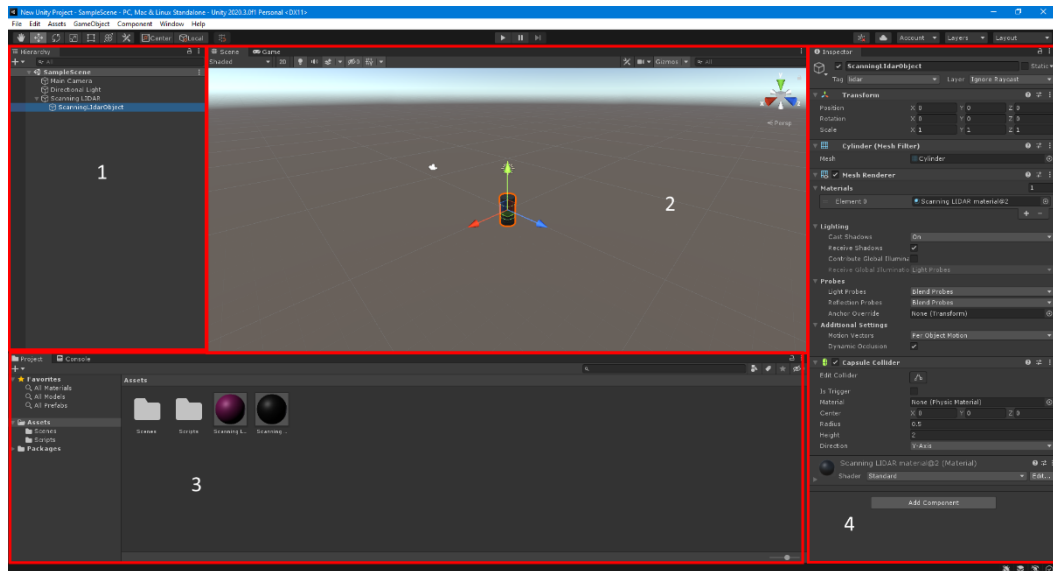
Unityn suurin käyttötarkoitus sen käyttäjillä on pelien kehittäminen. Näitä pelejä on mahdollista tehdä usealle eri alustalle, kuten eri konsoleille, puhelimille, tietokoneen käyttöjärjestelmille, verkkoselaimiin (WebGL) ja myös älytelevisioille. [29]

Unityä käytetään pelien kehittämisen lisäksi myös useassa muussa käyttökohteessa. Mahdollisuus käyttää Unityssä kehitettyjä sovelluksia useassa eri laitteessa, kuten esimerkiksi kosketusnäytöllä ja tietokoneella mahdollistavat usean eri käyttötavan.

Unityä käytetään myös erilaisten simulaatioiden toteuttamiseen. Oikean maailman ilmiöitä on joko vaarallista tehdä, se on liian kallista, tai prototyypin rakentaminen vie paljon aikaa. Esimerkiksi robotiikassa oikean robotin rakentaminen on aikaa vievää ja kallista, jolloin simulaatiolla pystytään ehkäisemään mahdolliset ongelmat ennen varsinaisen robotin rakentamista [5].

3.2 Unity-ympäristö

Unity-editorin käyttöliittymä on suunniteltu käyttäjälle mahdollisimman helpoksi käyttää, ja käyttäjä pystyy muokkaamaan käyttöliittymää oman makunsa mukaiseksi. Kuvassa 8 on näyttökuva Unity-editorista projektin alussa, johon lisätty korostettuna oleva peliobjekti. Tärkeimmät alueet aidattu punaisella suorakulmiolla.



Kuva 8: Kuva Unity3D:n käyttöliittymästä. Tärkeimmät alueet aidattu punaisella ja numeroituna valkoisella.

Alue 1 on hierarkiaikkuna. Siinä listataan pelinäkömään (engl. Scene) kuuluvat peliobjektit, ja niiden säilössä olevat tiedostot. Hierarkiaikkunassa voidaan liittää jokin peliobjekti liitokseen toisen peliobjektin alle raahaamalla se hiirellä. Kuvassa ”Scanning LIDAR” objekti on tyhjä säiliö, minkä alle asetetaan komponentit. ”ScanningLidarObject” on kyseiselle peliobjektille annettu sylinterimäinen muoto, jolle on annettu väritys. Tämä nähdään alueessa 2. Alue 2 on Unity-editorin ikkuna tarkastella, miltä pelinäkömä näyttää ilman, että kokoaa koko projektia. Tämän voi vaihtaa *Game viewiin* tai *Scene viewiin* riippuen tarpeesta. Game view simuloi pelinäkömän kameroiden läpi katsoen. Scene view:ssä käyttäjä pystyy liikkumaan ympäri pelinäkömää ja muokkaamaan sitä kesken testiajon. Alueessa 2 on korostettu peliobjekti nimeltään ”ScanningLidarObject”. Korostettuna siitä nähdään sen sijainti ja osoitussuunta alueessa 4, sekä sen lokali koordinaatisto pelinäkömässä. Kappaletta pystyy liikuttamaan raahaamalla sitä siitä lähtevistä koordinaatistonuolista. Alue 3 on projekti-ikkuna. Sieltä löytyy projektissa käytössä olevat tiedostot, ja sieltä pystyy helposti navigoimaan tiedostot löytäen ja järjestelemään niitä. Alue 4 on ikkuna peliobjektin ominaisuuksien tarkkailemiseen, ja niiden arvojen

vaihtamiseen. Sieltä näkee ja on mahdollista vaihtaa esimerkiksi mihin suuntaan peliobjekti osoittaa ja missä koordinaateissa se sijaitsee. [30]

Pelinäkymässä esitetään peliobjekteja. Peliobjekti on säiliö, minkä sisälle toteutetaan kyseisen kappaleen toiminnallisuudet ja ominaisuudet. Esimerkiksi ohjelmakoodi (engl. Script) siitä, miten kappale liikkuu, tai tieto mihin suuntaan kyseinen kappale osoittaa [31]. Ohjelmakoodi on Unityn automaattisesti ja käyttäjän kirjoittamaa koodia, joka liitetään osaksi peliobjektia. Sillä luodaan peliobjektin käyttäytyminen. Unityn sovelluksien ohjelmointikielenä käytetään C# -ohjelmointikieltä. Ohjelmistokoodin kirjoittaminen tapahtuu missä tahansa tekstieditorissa [32]. Tämän työn ohjelmakoodit on toteutettu C#-ohjelmointikielellä ja tekstieditorina on käytetty Microsoft Visual Studio Community 2019 -versiota.

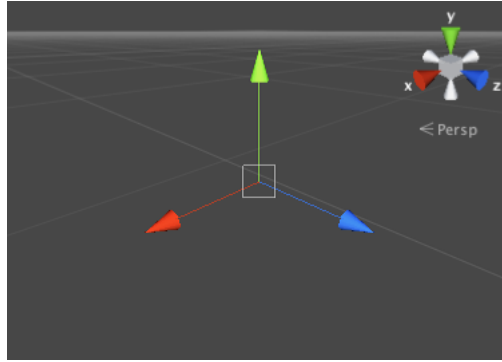
Peliobjekti kokonaisuudessaan pystytään tallentamaan *Prefabiksi*, eli esivalmisteeiksi (engl. Prefabricated). Esivalmistettu peliobjekti on Unityn systeemi luoda, tallentaa ja muokata kokonaisia peliobjekteja ja niiden säilössä olevia komponentteja helposti uudelleenkäytettäväksi objektiksi [33]. Tässä työssä luodaan eri LIDAR-järjestelmistä esivalmistetut peliobjektit, jotka on mahdollista asettaa suoraan pelinäkymään.

3.3 Kappaleen transformaatio Unityssä

Jokaisella objektilla Unityssä on transformaatio (engl. Transform). Kappaleen transformaatio tarkoittaa Unityssä objektin sijaintia maailmassa, sen rotaatiota sekä sen skaalausta. Peliobjektin transformaatio periytyy lapsiobjekteille, eli kun isäntäobjektin transformaatiota muutetaan, muuttuu sen lapsiobjektien transformaatio yhtä paljon. [34]

Unityssä rotaatiot esitetään kahdella eri tavalla kolmiulotteisessa avaruudessa. Ihmisille helpompi tapa lukea ja ymmärtää kulmia ovat Eulerin kulmat (engl. Euler angles). Toinen ilmaisutapa on *kvaterniset*-kulmat (engl. Quaternions), mitä Unity käyttää sisäisesti eikä käyttäjän varsinaisesti tarvitse niitä ymmärtää. Unity-editorissa kulmat esitetään Eulerin asteina ja käyttäjä pystyy sieltä muuttamaan objektin rotaatiota. Nämä esitetään kolmen arvon yhdistelmänä koordinaattisysteemin akseleiden ympäri.

Unityn koordinaatistosysteemi on vasenkätinen, missä Y-akseli osoittaa ylöspäin, ja pinta luodaan X- ja Z-akseleilla. Kuvassa 9 on Unityn koordinaattiakselit ja niiden osoittamat suunnat.



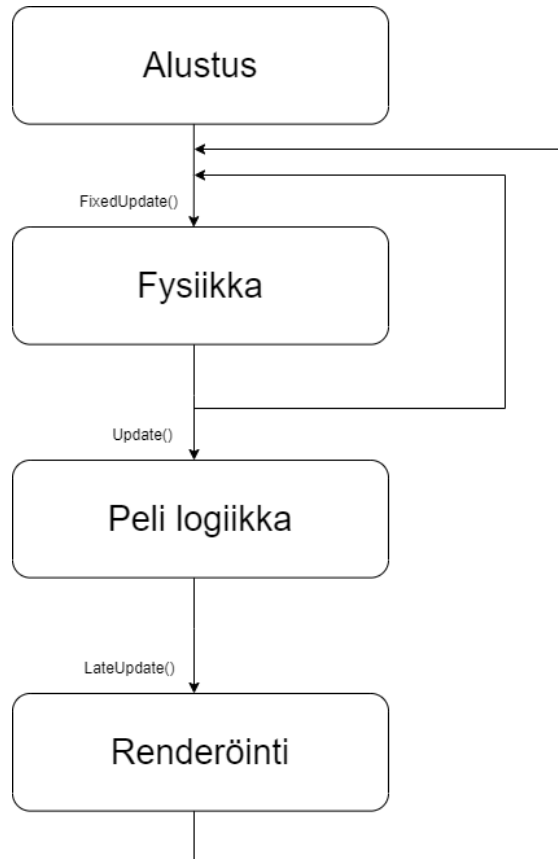
Kuva 9: Unity3D:n vasenkätinen koordinaatisto [35]

Kuvassa 9 punainen nuoli osoittaa x-akselin positiiviseen suuntaan, sininen nuoli z-akselin positiiviseen suuntaan ja vihreä nuoli osoittaa y-akselin positiiviseen suuntaan. Vasenkätisen koordinaatiston lisäksi Unityn rotaatiot toimivat vasemman käden säännöllä.

3.4 Unityn pelisilmukka

Unityssä toteutettu sovellus suorittaa tiettyjä ennalta määritettyjä funktioita tietyssä järjestyksessä. Tätä järjestystä kutsutaan *pelisilmukaksi*. Sovellus suorittaa pelisilmukan funktioita koko ajan ajan. Joidenkin funktioiden kutsuun käyttäjä pääsee vaikuttamaan, mutta tietyt funktiot ovat Unityn sisäisiä funktioita, joihin ei ole pääsyä. Jotta näitä ennalta määritettyjä funktioita pääsee peliobjekti kutsumaan, täytyy peliobjektilla olla periytettynä luokka *MonoBehaviour* [36].

Pelisilmukan tehtävä on lukea käyttäjän syöte (engl. Input), päivittää pelin objektien ominaisuuksia perustuen käyttäjän syötteeseen, sekä renderöinti, eli yksittäisen kuvan (engl. Frame) muodostaminen [37,38]. Mitä nopeammin ohjelma suorittaa pelisilmukkaa, sitä korkeampi on ruudunpäivitysnopeus eli Frames Per Second (FPS). Esimerkiksi mikäli peli suorittaa 60 FPS:n suorituskyvyllä, pelisilmukka suorittaa itseään 60 kertaa per sekunti. [39] Jos pelisilmukan kutsuminen loppuu, sovellus lakkaa toimimasta. Kuvassa 10 näytetään Unityn pelisilmukan järjestys ajan aikana. Liitteessä 1 on tarkempi ja yksityiskohtaisempi kuva tästä tapahtumien järjestyksestä englannin kielellä lähteestä [40].



Kuva 10: Unityn pelisilmukan järjestys tiivistettynä, mukailen lähdettä [40].

Ensimmäinen tapahtuma kun ajetaan Unity-projekti, on pelinäkömään alustaminen. Alustaminen kestää ensimmäiseen kuvanpäivitykseen alkuun asti. Tässä vaiheessa ajetaan yleensä ohjelmistokoodit, missä käyttäjä asettaa parametrejä peliobjekteilleen. Näitä alustamisfunktioita kutsutaan vain kerran.

Seuraavassa tapahtumassa tapahtuu yleensä fysiikan toimintojen kutsut. Tätä silmukkaa on mahdollista kutsua useamman kerran kuvanvaihdoksen aikana. Tätä silmukkaa kutsuu *FixedUpdate*-funktio. *FixedUpdate*-funktion kutsulle annetaan tietty aikaväli, minkä välein tätä kutsutaan riippumatta ajon suorituskyvystä. *FixedUpdate*ä käytetään fysiikkaominaisuuksien ohjelmoinnissa tämän tasaisen aikavälikutsun takia, mikä ei onnistuisi suorituskykyyn keskittyvässä *Update*-päivitysfunktiossa. [41]

Vakioarvo *FixedUpdate*en kutsun aikavälille on Unityssä 0,02 sekuntia, eli 50 kutsua per sekunti. Tämän arvon vaihtaminen onnistuu Unity-editorissa. [41] Pienin mahdollinen aikaväli Unityssä on 0,0001 sekuntia, eli 10 000 kutsua per sekunti.

Seuraavaksi pelilogiikan lukeminen tapahtuu *Update*-funktiossa. *Update*-funktioita kutsutaan aina kerran ruudunpäivityksen aikana. Eli jos ohjelmiston suorituskyky on 50 FPS, tällöin *Update*-funktioita kutsutaan 50 kertaa sekuntiin. Yleisesti logiikka tapahtuu tässä

funktiossa, mutta koska Update on erittäin suorituskykypainotteinen, on jotkut tapahtumat parempi kirjoittaa pelisilmukan muissa funktioissa [40].

Koska aktiivisen sovelluksen ruudunpäivitysnopeus voi vaihdella paljon ajon aikana, on myös Update-funktion sisällä tapahtuva liikehinnän logiikka hyvin epätasaisen kutsunopeudeltaan. Jotta tämän epätasaisen nopeuden pystyisi välttämään, on hyvä käyttää peliobjektien liikuttamiseen Time.deltaTime arvoa, jos käyttäjä haluaa vakionopeuksisen liikkeen. Time.deltaTime on liukulukuinen arvo viime kuvan ja nykyisen kuvan aikaväli sekunteina [42]. Esimerkiksi jos yhdellä tietokoneella sovellus toimii 20 FPS ja toisella 200 FPS ja peliobjektin liikehdintään ei sisällytetä Time.deltaTimeia, liikkuu korkeamman FPS:n tietokoneella peliobjekti nopeammin johtuen Update -funktion kutsujen tiheyden ja lukumäärän takia. Jos peliobjekti on ohjelmoitu liikkumaan 1 yksikkö per kutsu, liikkuisi tämä 20 yksikköä sekuntiin heikommalla tietokoneella, kun taas 200 yksikköä vahvemmalla tietokoneella.

Nyt nämä liikkeet kerrotaan Time.deltaTimeilla. 20 FPS kutsuu itseään 20 kertaa sekuntiin, eli kuvien välillä on 0.05 sekuntia aikaväliä ja samalla tavalla 200 FPS:llä on kuvien välillä 0.005 sekuntia aikaa. Annetaan peliobjektille liikenopeus 1, nyt muodostuu yhtälö (7)

$$\text{liike 1 sekunnin aikana} = \text{FPS} * \text{nopeus} * \text{Time.deltaTime}. \quad (7)$$

Saadaan 20 FPS:n suorituskyvyille liikkeeksi yhden sekunnin aikana 1. Sama arvo 1 tulee myös 200 FPS:llä.

Viimeisenä ennen kuvan renderöintiä kutsutaan LateUpdate-funktio. Tätä kutsutaan, kun Update-funktion sisällä olevat tapahtumat on suoritettu. [40]

4. SIMULAATION TOTEUTUS

Lopputuotteena on tarkoitus olla LIDARin toiminnallisuuden logiikka toteutettuna ideaalisissa olosuhteissa. Ideaalinen olosuhde tarkoittaa tässä kontekstissa häiriöiden eliminoinnista, kuten esimerkiksi vesisadetta tai pölypilviä ei oteta huomioon, koska lasersäteet eivät aina pääse niistä läpi, ja se tuo lisää haasteita. Lasersäteen ei haluta heijastuvan takaisin mistään muusta esteestä kuin kohteesta, mitä halutaan mitata. Oletuksena tässä työssä on, että kaikki takaisinsäteilevät datapisteet ovat totuudenmukaisia. Työn laajuuden kannalta ei ole kannattavaa ottaa aivan kaikkia fysiikan lakeja tai fysiikan ilmiöitä mukaan.

Simuloitavan LIDARin arvoja tulee pystyä vaihtamaan, kuten esimerkiksi kanavien lukumäärää, näkökenttää tai pyörimisnopeutta ilman lähdekoodin muuttamista. Simuloidun LIDARin pitää pystyä myös liittämään liikkuvaan kulkuneuvoon, missä skannaaminen jatkuu liikkeessä. Ulostulevat datapisteet tallennetaan johonkin tietorakenteeseen ja tiedostomuotoon, jotta kerätyllä datalla pystyisi tekemään jotakin. Tämä voi esimerkiksi tarkoittaa pistepilven muodostamista tai saadun informaation siirtämistä ROSiin (Robot Operating System).

Käyttäjänä toimii tutkija, joka haluaa tarkastella simulaatiosta saatua informaatiota. Tutkija luo haluamansa maailman Unityyn. Tähän maailmaan hän pystyy liittämään valmiin LIDAR-järjestelmän esivalmistetun peliobjektin ja halutessaan muuttamaan tämän arvoja. Simuloitua LIDAR-järjestelmää käytetään etäisyyksien mittaamiseen ilman oikeaa LIDARia. Järjestelmä on luotu mahdollisimman helppokäyttöiseksi, mutta Unity-editorin käytön osaamisesta on hyötyä ja käyttäjän tulee osata luoda pelinäkymiä mitä tutkia. Tässä työssä muodostetut pelinäkymät ovat erittäin yksinkertaiset, jotta ruudunpäivitysnopeus olisi mahdollisimman nopea ja mittauspisteet mitä mitataan, olisivat mahdollisimman selkeät. Työ on luotu ja testattu Windows käyttöjärjestelmässä.

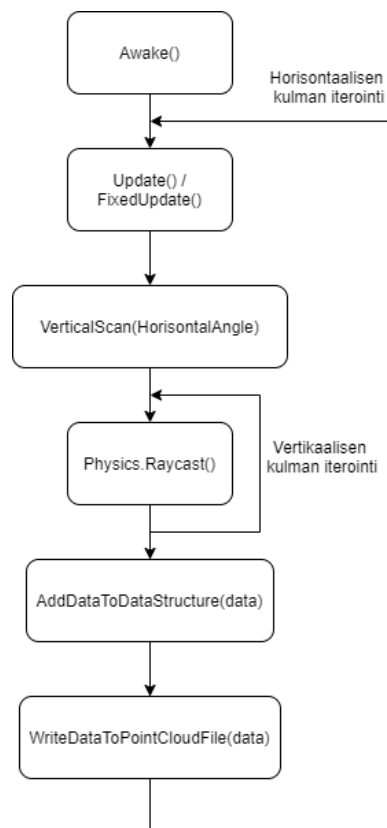
Lopputulokseksi tulee kolme esivalmistettua LIDAR peliobjektia. Näistä kaksi on ympäri skannaavan LIDARin peliobjektit toteutettuna eri pelisilmukan päivitysfunktiolla. Kolmas esivalmiste on Flash-LIDARin kaltainen ratkaisu, jossa kaikki datapisteet mitataan yhtä aikaa isolta näkökentältä.

Esivalmisteiden rakentaminen aloitetaan lisäämällä tyhjä peliobjekti-säiliö pelinäkymään ja lisätään sen sisälle erillinen sylinterinmuotoinen peliobjekti. Lisätään siihen haluttu väri

ja ohjelmointitiedosto logiikkaa varten. Annetaan myös isäntäobjektille tagi "lidar" yksikötestejä varten sekä vaihdetaan sen kerros (engl. Layer) "Ignore Raycast".

4.1 Skannaavan LIDARin toteutus

Tässä alaluvussa tarkastellaan ympäri skannaavan LIDAR peliobjektin logiikan toteutus. Skannaaminen tapahtuu pelisilmukan päivitysfunktiota (Update tai FixedUpdate) käyttämällä. FixedUpdate on kuitenkin se funktio, missä kaikki fysiikkapohjaiset toiminnallisuudet halutaan toteuttaa. Kuvassa 11 on skannaavan LIDARin funktioiden järjestys ja miten ne silmukoivat.



Kuva 11: Skannaavan LIDARin ohjelman rakenne.

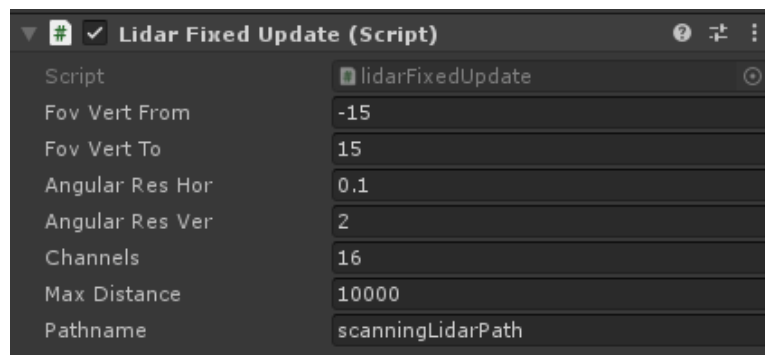
Kuvassa 11 ohjelma alustetaan Unityn Awake-funktiolla. Ennen varsinaisen ajon alkua ohjelman käynnistettyä, Awake-funktiossa varmistetaan, että sopiva pistepilvitiedosto löytyy sille osoitetussa sijainnissa. Jos sitä ei ole, se luodaan. Ensimmäisessä päivitysfunktion kutsussa skannaaminen alkaa suoraan objektin edestä. Tätä rotaatiota kasvatetaan sitten käyttäjän antamalla arvolla joka päivitysfunktion kutsulla.

Riippumatta kumpaa päivitysfunktiota käytetään, LIDAR-objektille annetaan arvot kuvien 12 ja 13 mukaan Unity-editoria ja kooditiedostoa käyttäen. LIDARille annettavat arvot

lukuun ottamatta maksimietäisyyttä ovat peräisin lähteestä [13]. Unityn etäisyydet eivät varsinaisesti ole metreissä, mutta voidaan ajatella, että yksi yksikkö etäisyyttä vastaa yhtä metriä oikeassa maailmassa [43].

```
public float fovVertFrom ; // lowest angle that the device can see. -15 for puck
public float fovVertTo; // highest angle that the device can see. 15 for puck
public float angularResHor; // depends on rot. speed. 300-1200 rpm -> 0.1-0.4 degrees
public float angularResVer; //2 degrees for velodyne puck
public int channels; // 16 for velodyne Puck
public float maxDistance = 100f; // 100m for puck,
public string pathname = "PointcloudFile";
```

Kuva 12: Editorissa näkyvien parametrien julkiset muuttujat ohjelmistotiedostossa.



Kuva 13: Editorissa annettavat objektin parametrit.

Kuvassa 12 näkyvät muuttujat ovat julkisia muuttujia, jotta ne löydettäisiin Unity-editorista, eli kuvasta 13 sen peliobjektin alla, mihin tämä ohjelmistokoodi on sijoitettu. Näitä arvoja pystytään myös vaihtamaan kesken simulaation ilman ajon sammuttamista, kun ohjelmaa ajetaan editorissa. Peliobjektille annetaan arvoja sen mukaan, miten sen halutaan käyttäytyä.

Skannaaminen alkaa lukemalla kuvista 12 ja 13 annetut arvot, joita käytetään tässä loogiikassa hyödyksi. Päivitysfunktion tehtävänä on pyörittää objektia y-akselin ympäri, eli kun sitä kutsutaan, pyörähtää objekti vasemman käden säännön mukaan positiiviseen suuntaan kuvassa 13 näkyvän "Angular Res Hor" kulman verran. Mikäli rotaatio menee yli 360 astetta, Unity tunnistaa tämän ja muuttaa rotaation takaisin 0 asteeseen aloittaen uuden kierroksen [44].

Päivitysfunktion kutsun aikana tehdään skannaus vertikaalisessa suunnassa yhtäaikaisesti yhden kuvanvaihdon aikana. Toisin sanoen, kaikki säteet lähtevät yhtäaikaisesti tässä simulaatiossa. Oikean maailman tilanteessa näin ei käy, mutta koska kyseessä on erittäin lyhyt aika, noin $2.3 \mu\text{s}$ per säde [13], voidaan approksimoida sitä lähettämällä säteet yhtäaikaisesti tässä simulaatiossa suorituskyvyn säästämiseksi. Tämä vastaisi

noin 430 000 funktiokutsua per sekunti. Kuvassa 14 näytetään tämän pystysuuntaisen skannauksen toteutus.

```

hitData[] scanData = new hitData[channels]; //
for (int i = 0; i < channels; i++) // as many layers of raycast as there are channels assigned for this gameobject
{
    float verAngleOfChannel = fovVertFrom + (i * angularResVer);
    Vector3 direction = transform.TransformDirection(Quaternion.Euler(verAngleOfChannel, horAngle, 0) * Vector3.forward);
    if(Physics.Raycast(transform.position,direction, out hit, maxDistance))
    {
        Vector3 localCoords = transform.InverseTransformPoint(hit.point);
        scanData[i] = new hitData()
        {
            coords = localCoords,
            channel = i
        };
    }
}
channelsDataPerScan verScan = new channelsDataPerScan()
{
    rotPos = horAngle,
    hits = scanData,
    timeStamp = Time.time
};

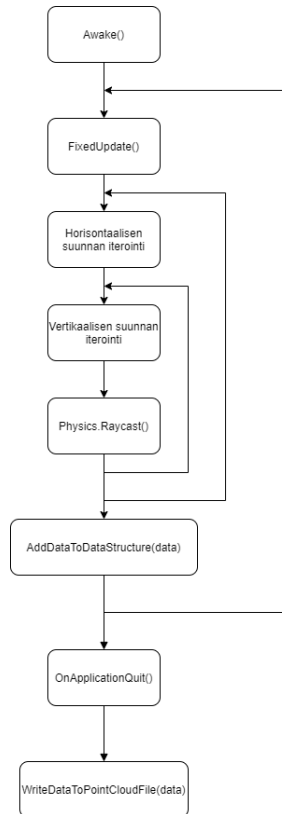
```

Kuva 14: Toteutettu pystysuuntainen skannaus.

Halutut datapisteet saadaan Unityn fysiikkamoottorin metodilla *Raycast*. Jos *Raycast* törmää objektiin, saadaan tämän osumapisteen etäisyys lähtökohteesta. *RayCast* tallentaa osumansa globaalissa koordinaatistossa [35], eli se pitää muuttaa LIDAR-objektin lokaaliin koordinaatistoon. Kun kaikki vertikaalisen suunnan skannaukset on suoritettu, kirjoitetaan saatujen pisteiden koordinaatit pistepilvitiedostoon.

4.2 Flash-LIDAR

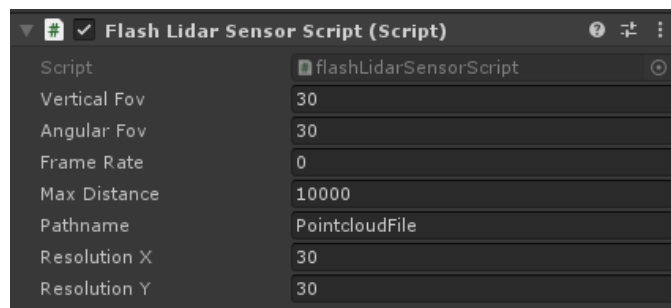
Flash-LIDAR toteutetaan vain pelisilmukan *FixedUpdate*-funktioita käyttäen. Tämän pysyisi toteuttamaan myös *Update*-funktiossa, mutta halutaan että datapisteet tulevat aina tasaisin aikavälein. Kuvassa 15 näytetään toteutuksen rakenne.



Kuva 15: Flash LIDARin ohjelman rakenne.

Ohjelman ajo alkaa Awake-funktion kutsulla. Siellä varmistetaan, että polku pistepilvitiedoston kirjoittamiseen löytyy, sekä varmistetaan oikeat parametrien arvot edellä toteutettuihin funktioihin. Siellä myös vaihdetaan FixedUpdate-funktion kutsujen tiheys ilman että käyttäjä vaihtaa sitä manuaalisesti asetuksista.

Kun kaikki kerrokset on käyty läpi horisontaalisessa suunnassa sekä korkeussuunnassa, lisätään saadut pisteet yhteen isoon tietorakenteeseen. Kuvassa 16 näytetään tarvittavat parametrit, mitä Flash LIDARin simulaatio tarvitsee.



Kuva 16: Unity-editorin parametrit Flash LIDARille.

"Vertical FoV" ja "Angular FoV" ovat toteutetun Flash LIDAR peliobjektin näkökenttä pysty- ja vaakasuunnassa. "Resolution X" ja "Resolution Y" ovat datapisteiden määrä

leveys- ja korkeussuunnassa, jotka asetetaan ohjelmakoodissa tasavälein. *Frame rate* vaihtaa *FixedUpdate*n kutsujen tiheysnopeutta. Esimerkiksi jos annetaan arvo 25, tarkoittaa se 25 kertaa per sekunti, jolloin kutsujen aikaväli olisi tämän luvun käänteisarvo eli 0,04 sekuntia. Tämä asetetaan *Fixed time step* -arvoksi ilman käyntiä Unityn aika-asetuksissa.

Flash LIDARin toteutuksessa kuva otetaan laajalta alueelta yhtäaikaisesti. Toisin kuin skannaavan LIDARin toteutuksessa tehtiin, tässä otetaan sekä vertikaalinen että horisontaalinen skannaus yhden *FixedUpdate* kutsun aikana. Kuvassa 17 näytetään tämän toteutus.

```

hitData[] flashData = new hitData[hitdatasize];
int index = 0;
for (float j = -sight; j < sight; j += rotHor) // sight = angular fov / 2, so if horisontal fov is 40, we have view of -20 and +20
{
    for (float i = -verticalSight; i < verticalSight; i += rotVert) // verticalsight = vertical fov/2, if vertical fov is 40, we have -20 and +20
    {
        Vector3 direction = transform.TransformDirection(Quaternion.Euler(i, j, 0) * Vector3.forward);
        if (Physics.Raycast(transform.position, direction, out hit, maxDistance))
        {
            Vector3 localCoords = transform.InverseTransformPoint(hit.point);
            flashData[index] = new hitData()
            {
                coords = localCoords,
                index_ = index
            };
            index++;
        }
    }
}
data.Add(flashData);
}

```

Kuva 17: Flash LIDARin skannauksen toteutus

Kuvassa 17 näkyvät muuttujat "sight" ja "verticalsight" ovat puolet kuvan 16 annetuista "Vertical FoV" ja "Angular FoV" arvoista. Tämän tarkoituksena on, että tämä LIDAR peliohjelma mittaa suoraan edestäpäin, eikä vaihda skannauskulmaa kesken ajon. Aina kun vertikaalisen suunnan skannaukset on tehty, horisontaalinen kulma siirtyy seuraavaan ja vertikaalinen skannaus aloitetaan taas matalimmasta pisteestä. Kun kaikki skannaukset on suoritettu kutsun aikana, lisätään nämä datapisteet tietorakenteeseen.

Kuvassa 15 näkyvää Unityn funktiota *OnApplicationQuit*:ia kutsutaan, kun ohjelman ajo loppuu. Siellä kirjoitetaan Flash LIDARin ajon aikana mitaamat datapisteet pistepilvitiedostoon. Tässä toteutuksessa pistepilvitiedosto kirjoitetaan ajon lopetettua, koska joka kutsulla tulee paljon enemmän mitattuja pisteitä verrattuna skannaavaan LIDARIin.

4.3 Simulaation ulostulo

Simulaatiosta saadut datapisteet tallennetaan ajon aikana C#:in tietorakenteeseen, sekä kirjoitetaan koordinaatit ylös pistepilvitiedostoon. Pistepilvitiedostoon tallennetaan datapisteitä kolmiulotteisessa maailmassa. Nämä pisteet tallennetaan LIDARin paikallisessa

koordinaatistossa x-,y- ja z-koordinaatteihin. Toisin sanoen, LIDAR sijaitsee aina origossa riippumatta sen sijainnista globaalissa koordinaatistossa.

Riippuen tiedostoformaattista, pystytään myös tallentamaan muuta informaatiota. Tässä työssä tallennetaan datapisteet .xyz-pistepilvitiedostomuotoon. Tiedostomuodossa .xyz jokaisesta datapisteestä kirjoitetaan x-,y- ja z-koordinaatit tässä järjestyksessä, joiden välissä on aina sama tietty välimerkki. Jokaisen kolmesta pisteestä koostuvan koordinaatin jälkeen tapahtuu rivinvaihdos. Tyhjä osuma, eli kun Raycast ei osu mihinkään, tallennetaan (0,0,0) koordinaatteihin. Näin kaikista lähtevistä Raycasteista saadaan ulostulo.

Tässä työssä pistepilvitiedosto kirjoitetaan ajon aikana skannaavissa LIDAReissa. Flash LIDARin toteutuksessa se kirjoitetaan ajon jälkeen säästämään suorituskykyä. Datapisteet tallennetaan kuitenkin aina ajon aikana C#:n tietorakenteeseen, jonka sisältöä pystyy lähettämään ja käsittelemään ajon aikanakin.

Pistepilvitiedoston datapisteistä pystytään muodostamaan 3D-verkostokuva (engl. Mesh image) 3D-mallinnusohjelmilla. Käytämme tässä työssä ilmaista mallinnusohjelmaa *Meshlab* [45], johon pystyy syöttämään Unityn ajosta tuotettu .xyz pistepilvitiedosto, ja tuottamaan siitä *pistepilvikuvaa*.

4.4 Yksikkötestaus

Yksikkötestaaminen (engl. Unit test) kuuluu hyvään ohjelmistosuunnitteluun. Yksikkötestaaminen on ohjelmiston komponenttien testaamista automaattisesti varmistaakseen toimivuuden ilman käyttäjän manuaalista testaamista [46]. Tässä ohjelmassa testataan LIDARien ohjelmistokoodien toimivuudet ja luotettavuudet.

Unityssä yksikkötestaaminen onnistuu Unity Test Runnerilla helposti. Unity Test Runner on työkalu, millä pystytään testaamaan ohjelman koodia Edit-moodissa ja Play-moodissa, sekä eri alustoilla ilman kyseisen alustan fyysistä liitäntää [47]. Edit-moodi testaa vain Unity-editorissa valmiina olevaa koodia, kun taas Play-moodilla on mahdollista ottaa huomioon ajon aikana tapahtuvat muutokset [48]. Eli Play-moodissa testataan ohjelmistokoodit, mitkä vaativat ohjelman käynnissä olemista, kuten Update- tai FixedUpdate-funktioiden kutsua.

Tässä työssä Edit-moodin tärkein testaus tässä sovelluksessa on se, että pelinäkömästä löytyy "lidar"-tagin omaava objekti, mikä on laitettu tämän työn tekemissä esivalmistetuihin LIDAR peliobjekteihin. Muita testejä mitä voi tehdä on esimerkiksi varmistaa, että LIDAR on ajoneuvon lapsiobjekti, jolloin tämä seuraisi ajoneuvoa.

Play-moodin testeissä tarkistetaan, että staattisen LIDARin sijainti ei muutu sen skannauksen aikana, skannauksen aikana pyörähdyskulma vaihtuu, yhden dataskannauksen aikana saa sen oikean määrän datapisteitä, oikeat reaktiot osumista ja ei-osumista sekä varmistus siitä, että skannaava laite pystyy pyörimään akselinsa ympäri. Tarkistetaan myös, että datapisteiden ulostulo on oikeanlaista, eli sopivaa .xyz tiedostoformaattiin.

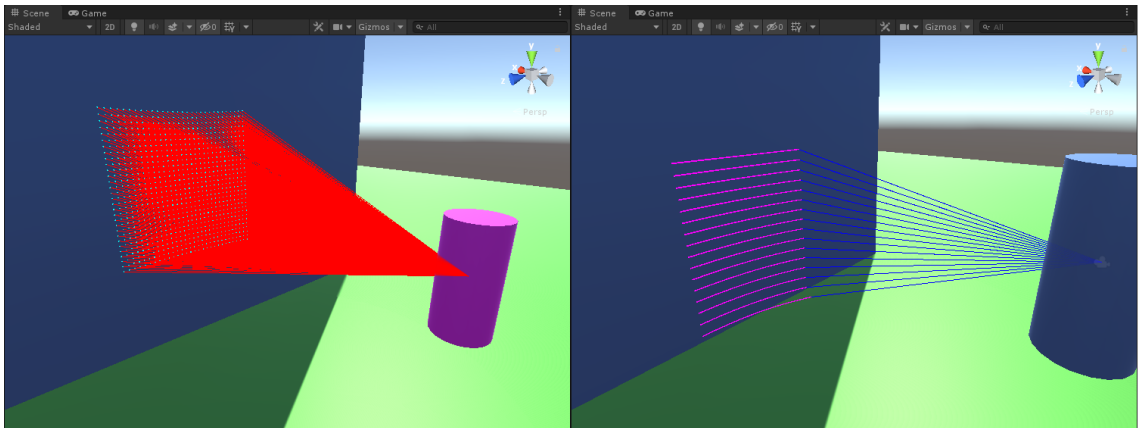
Raycastille Unityllä on itse todennäköisesti omat testinsä, vaikka niistä ei löydy dokumentaatiota. Tässä työssä myös testataan Raycastin toimivuutta myös kahdella tavalla. Toinen tapa on laittaa LIDAR-peliobjekti onton pallon sisään keskelle, minkä säde tunnetaan ja siitä verrataan sen tunnettua sädettä ja Raycastin antamaa etäisyyttä. Toinen testaus on sijoittamalla Raycastin antamat koordinaatit Pythagoraan lauseeseen ja laskemalla siitä etäisyys kaavan (8) mukaan

$$D(P_0, P_1) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}. \quad (8)$$

Etäisyys pisteiden P_0 ja P_1 välillä saadaan summaamalla (x,y,z) -koordinaattien erotusten neliöt keskenään ja tästä summasta otetaan neliöjuuri. Voidaan todeta, että luodut esivalmiit LIDAR-peliobjektit pääsevät läpi jokaisesta luodusta testistä.

5. SIMULOINNIN TULOKSET

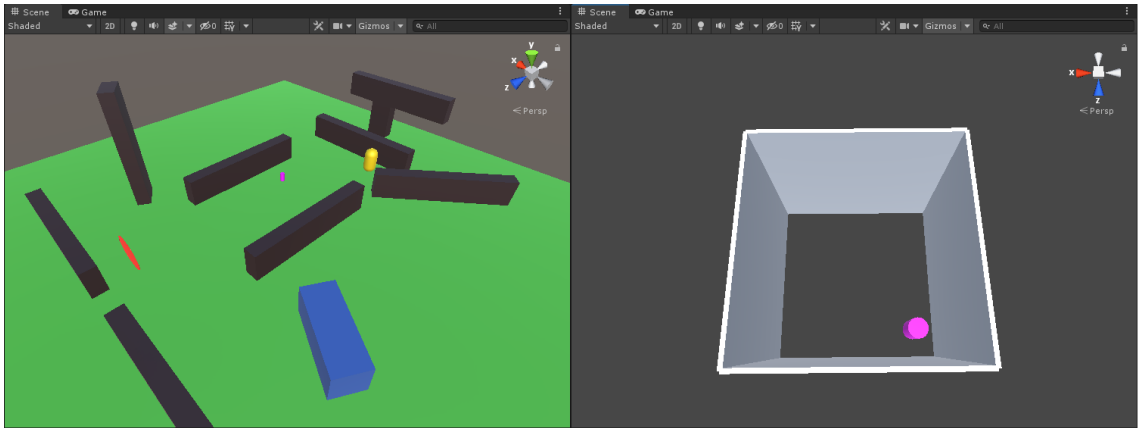
Unityn Debug-työkalulla on mahdollista tarkastella RayCastin osumia ajon aikana edito- rissa. Debug.Drawlinea pystytään käyttämään sovellusta debugatessa ja tarkastele- maan, mihin Raycast osuu. Ohjelman kehitysvaiheessa on myös hyvä varmistaa Ray- castin toiminnallisuus tämän avulla. Kuvassa 18 näkyvät luotujen LIDAR peliobjektien Raycast säteet käyttämällä Debug.Drawline -metodia.



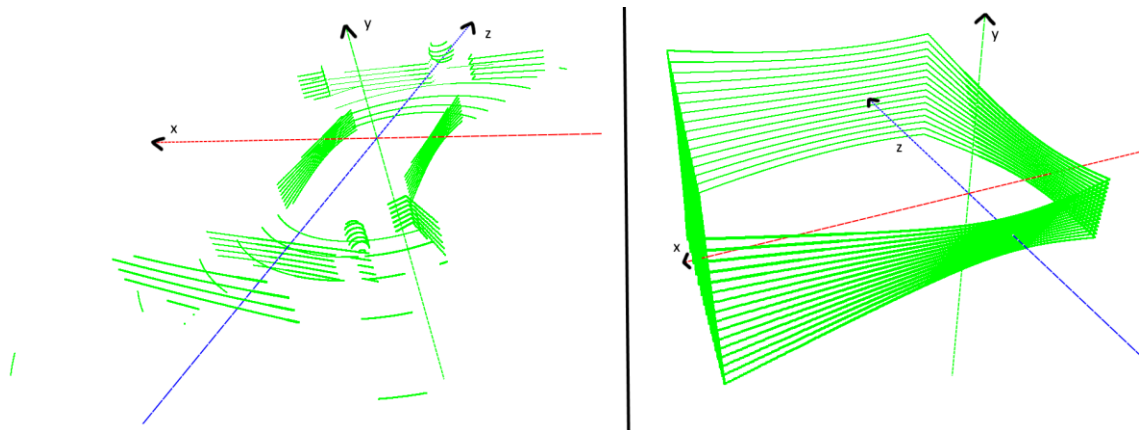
Kuva 18: Toteutetut skannaukset Debug.Drawline -työkalulla katsoen, vasemmalla on Flash LIDAR ja oikealla skannaava LIDAR.

Kuvassa 18 vasemmalla on Flash LIDAR ja oikealla on skannaavan LIDARin toteutus. Jokaisella FixedUpdate kutsulla Flash LIDAR mittaa tämän kokoisen alueen osumapisteet ja skannaava LIDAR vain vertikaalisessa suunnassa sijaitsevat pisteet, mikä näkyy oikean puolen kuvassa sinisellä viivalla. Flash-LIDAR osoittaa suoraan eteenpäin ja skannaava LIDAR pyörii ympäri vasemman käden rotaatiosäännön mukaan. Debug.Drawlineä käyttäen nähdään visuaalisesti, että skannaukset näyttäisivät totuudenmukaisilta. Visuaalisesti tarkkailemalla on kuitenkin vaikea saada simulaation mittauspisteistä selvää. Saaduista mittauspisteistä pitää saada pistepilveäkin muodostettua.

Luodaan kaksi esimerkki maailmaa staattiselle LIDARille. Yhdestä maailmasta luodaan hieman monimutkaisempi lisäämällä siihen erimuotoisia esteitä mitattavaksi. Toinen luotu maailma on huone, minkä sisälle LIDAR-peliobjekti sijoitetaan. Näissä LIDAR skanna-aa staattisesti pysyen paikallaan, samalla pyörien ympäriinsä. Ympäri skannaava LI- DAR-peliobjekti on parempi valinta, kun mitataan ympärillä olevia kohteita paikallaan. Kuvassa 19 on luodut maailmat ja kuvassa 20 on LIDAR-peliobjektin kirjoittaman pistepilvitiedoston pisteistä luodut pistepilvikuvat Meshlab ohjelmalla.



Kuva 19: Kaksi toteutettua testimaailmaa, kuvissa vaaleanpunainen sylinteri esittää LIDAR-peliobjektin sijaintia, josta se skannaa paikallaan.

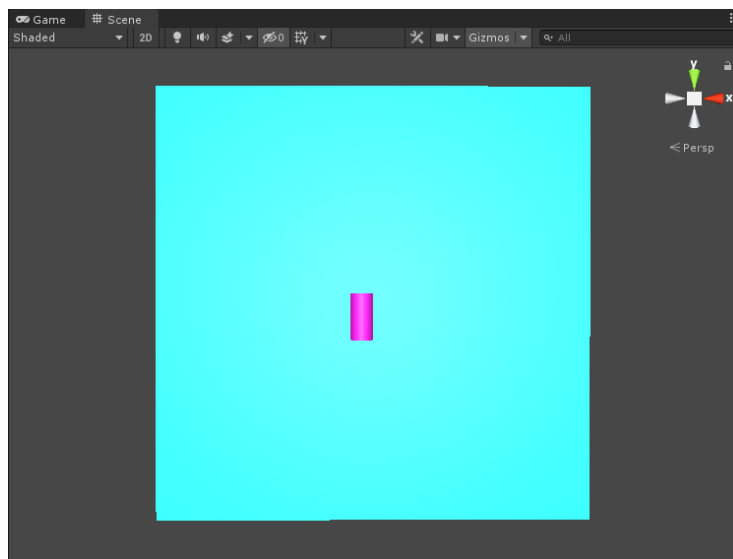


Kuva 20: Skannaavan LIDARin pistepilvitiedostosta luodut pistepilvikuvat. Koordinaatiakselit leikkaavat toisensa origossa, eli jossa LIDAR-peliobjekti on sijainnut.

Kuvista nähdään, että pisteet tuottavat aika tarkan näköisen pistepilvikuvan alkuperäisistä maailmoista. Vasemmasta kuvasta huomataan, miten koordinaatistosysteemin vaihtaminen vaikuttaa kuvan luontiin. Meshlabin koordinaatistosysteemi on oikeankäden säännön mukainen. Oikean käden koordinaatistosysteemissä positiivinen z-akselin osoittaa vastakkaiseen suuntaan verrattuna vasemman käden koordinaatistosysteemiin. Vasemmasta kuvasta huomataan myös, miten peliobjektin *osumalaatikko* (engl. Hitbox) vaikuttaa Raycastin osumiin. Kuvassa 19 näkyvä punainen kyltti on pallo-objekti, mitä on ohennettu visuaalisesti. Sille on kuitenkin jätetty näkymätön ympyrämäinen osumalaatikko säteellä r , josta sitten Raycast ottaa osumapisteensä. On siis tärkeä vaihtaa kap-

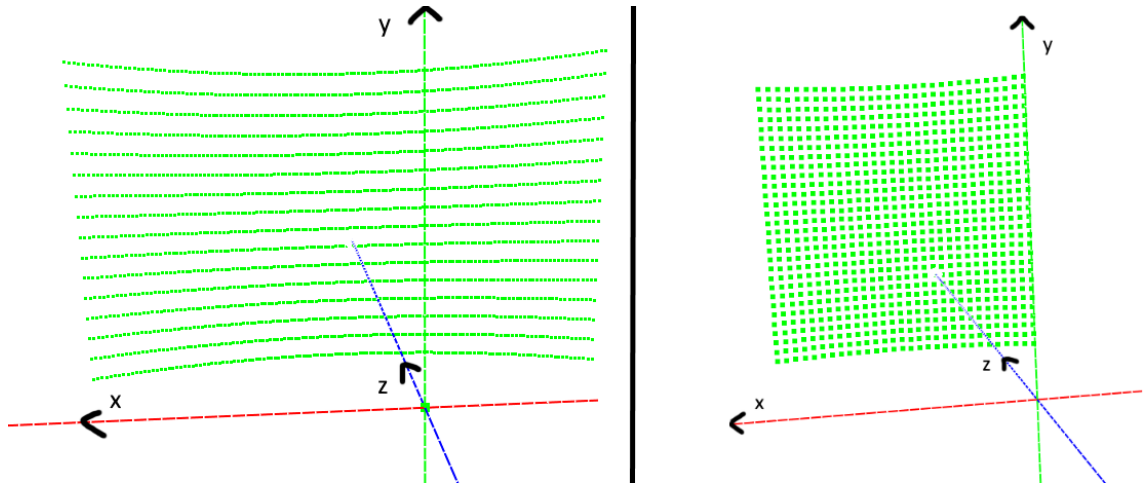
paleen ominaisuuksia tarkasti. Oikeanpuoleisesta kuvasta huomataan, minkä seinän lähellä LIDAR-peliobjekti on sijainnut mitatessa. Mitä ohuemmin ovat mitatut pisteet sijoittuneet, sitä lähempänä on LIDAR ollut. Ja mitä harvemmin mittapisteitä vertikaalisessa suunnassa, sitä kauemmaksi on säde lähtenyt.

Katsotaan seuraavaksi liikkuvan LIDARin mittaustuloksia. Otetaan tähän testiin Flash LIDAR mukaan. Luodaan maailma, missä LIDAR asetetaan 1000 yksikön päähän palkista, minkä koko on 1000x1000x1. Palkin ohuin kohta asetetaan kohtisuoraan LIDAR-peliobjektia kohti. Seinä on suora, ja se on tasainen. Kuvassa 21 näytetään tämä skenaario. LIDARin aloitusrotaatio on suoraan z-akselia pitkin kohti seinää.



Kuva 21: Liikkuvan LIDARin skenaario. Liike tapahtuu kohti seinää.

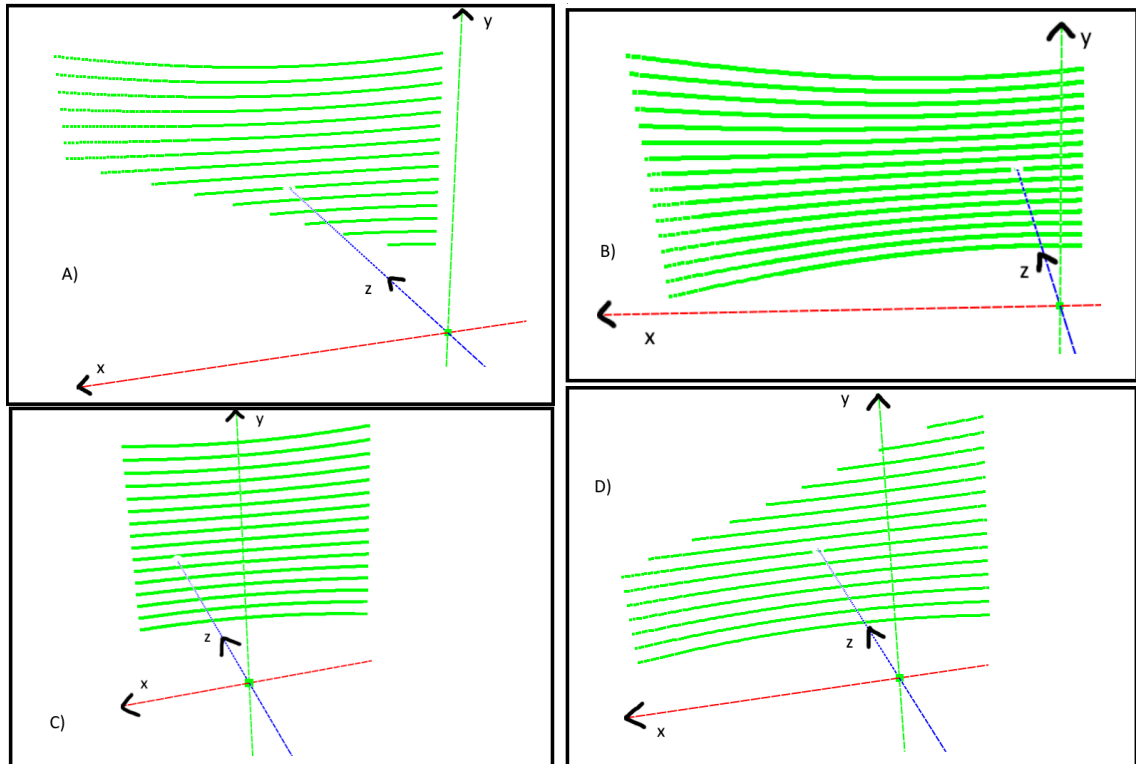
Kuvassa 21 LIDAR-peliobjekti on asetettu 1000 yksikön etäisyydelle seinästä. Sen aloitusrotaatio ja -sijainti on kohtisuoraan keskelle seinää. Lisätään maailmaan tyhjä peliobjekti, johon sisällytetään liikehdintään tarvittava ohjelmistokoodi. Sijoitetaan LIDAR-esivalmiste tämän tyhjän peliobjektin sisälle, jotta se seuraa sitä. Liikkumisen ohjelmistokoodi lisättiin FixedUpdate-funktion sisälle. Kuvassa 22 on paikallaan olevien LIDAReiden muodostama kuva kuvan 21 näkymästä.



Kuva 22: Vasemmalla skannaavan LIDARin ja oikealla Flash LIDARin tuottama kuva seinästä paikallaan ollessa.

Kuvan 22 molemmissa tapauksissa z-akseli läpäisee skannauksen keskikohdasta, tässä virhenäkymää tuottaa kuvakulma, mistä pistepilvikuva on otettu. Skannaavan LIDARin tapauksessa skannaaminen aloitettiin hieman palkin ulkopuolelta, jotta ei tulisi turhaa tyhjiä skannauksia. Kuvaan on myös asetettu Meshlabin oikeankäden koordinaatisto, jossa akselit leikkaavat toisensa origossa.

Lisätään seuraavaksi liike. Flash LIDAR tuottaa aina saman näköisen kuvan, jos kaikki säteet osuvat tässä tasaisen seinän tapauksessa. Keskitytään skannaavan LIDARin tuottamiin kuviin. Kuvassa 23 näytetään muodostetut pistepilvikuvat, kun LIDAR ohjelmoidaan liikkumaan ylös, alas, oikealle ja vasemmalle samalla skannaten.



Kuva 23: Liikkuvan skannaavan LIDARin tuottamat kuvat. Suunnat, mihin LIDAR liikkuu kuvassa ovat seuraavat: A) Alas; B) Vasemmalle; C) Oikealle; D) Ylös. Kuvaan piirretty koordinaatisto, missä akselit leikkaavat origossa.

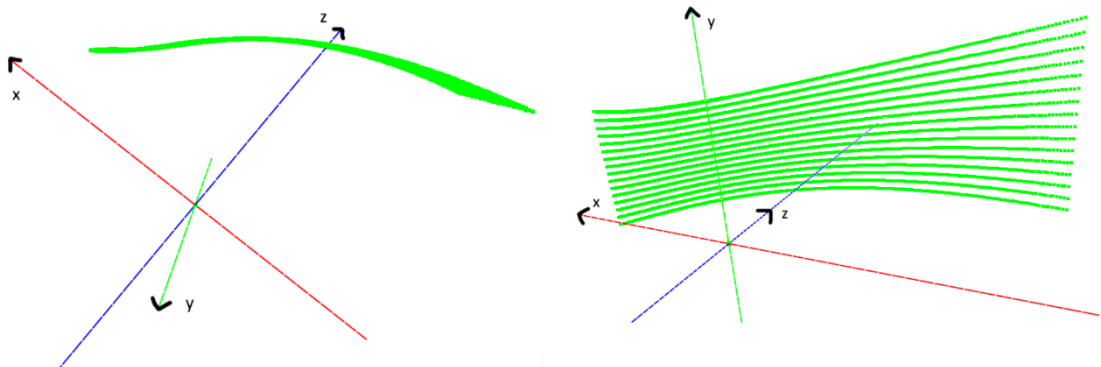
LIDAR-peliobjektin ja sen ajoneuvon aloitussijainti ja -rotaatio ovat samat kaikissa kuvissa. Myös liikehdinnän nopeus on sama kaikissa tapauksissa, sillä liikehdinnän logiikka on asetettu FixedUpdate-funktion sisälle, ja nopeus pysyy samana yhtälön (7) mukaan. FPS ei vaikuta FixedUpdaten sisällä ja Time.deltaTime on FixedUpdate-funktion kutsujen tiheys, mikä on vakio.

Kuvassa 23 A)-kuva on otettu, kun LIDARin liike tapahtuu alaspäin, eli negatiiviseen y-suuntaan. D)-kuvassa liike on positiiviseen y-suuntaan. B)-kuvassa liike on negatiiviseen x-suuntaan ja C)-kuvassa positiiviseen x-suuntaan. Tuotetuista kuvista pitää huomioida, että alkuperältään vasemman käden koordinaattisysteemin koordinaatit siirretään sellaisenaan oikean käden koordinaatistosysteemiin.

Kuvasta 23 huomataan, miten liikehdintä aiheuttaa vääristymää geometriassa verrattuna kuvan 22 vasemmanpuoleiseen paikallaan olevaan skannaukseen. A)-kuvassa huomataan, että kaikki Raycastit eivät skannaakaan koko seinän leveyttä, vaan kuvattu pistepilvi muistuttaa puolisuunnikasta. Sama tapahtuu D)-kuvan tapauksessa. B)-kuvassa liikutaan vasemmalle, eli pitkitetään kuvaa liikkumalla vasemman käden rotaatiosäännön vastakkaiseen suuntaan. Huomataan kuvasta, että seinä on liikkunut x-akselin yli ja että

kuva on myös venynyt pituussuunnassa. Kuvassa C) on tapahtunut päinvastoin. Liikutaan oikealle, eli vasemman käden rotaatiosäännön rotaation suuntaisesti. Näin ollen tämä pienentää kuvaa.

Seuraavaksi seinää kohti tapahtuva liikehdintä. Flash LIDARin tapauksessa tulee vain suorakulmion muotoisia kuvia useita kerroksia peräkkäin. Kuvassa 24 ovat kuvat sivustapäin ja ylhäältä katsottuna, kun skannaava LIDAR-peliobjekti liikkuu kohti kuvan 22 seinää.



Kuva 24: Skannaavan LIDARin mittaamat pisteet, kun liikehdintä tapahtuu seinää kohti. Vasemmalla kuva katsottuna ylhäältä päin, oikealla katsotaan sivustapäin. Koordinaattiakselit leikkaavat toisensa origossa.

Huomataan, miten liikehdinnän aiheuttama vääristymä muodostuu. Pitää muistaa, että Unityssä on vasemman käden koordinaatistosteemi, kun taas Meshlab tuottaa kuvan oikean käden koordinaatistossa. Kuvan 24 piirtäminen aloitetaan oikean puolen kuvan oikeasta yläkulmasta ja viimeinen mittausta tapahtuu vasemmassa alareunassa. Nähdään kuvasta, että saadut pisteet lähenevät z-arvoa 0, koska liikehdintäkin tapahtuu z-akselia pitkin positiiviseen suuntaan negatiivisesta koordinaatista. Näin ollen, etäisyys pienenee samalla.

Ajan huomioonottaminen vaikuttaa olevan suurin ongelma simulaatiossa. Tietokoneen suorituskyky ei riitä, vaikka miten yksinkertaisen 3D-ympäristön rakentaisi Unityssä, eli täydellistä kopiota oikean maailman LIDARista ei ole mahdollista tehdä näillä komponenteilla. Mutta sitä pystytään approksimoimaan. Mutta jos aikaa ei tarvitse ottaa tarkasti huomioon, eli käyttö on pelkästään etäisyyden mittaamiseen, on kyseessä täysin toimiva kokonaisuus.

Pystytään myös ajattelemaan, että simulaatiossa aika liikkuu hitaammin verrattuna reaali maailman aikaan. Esimerkiksi 1 sekunti simulaatiossa vastaa 0.001 sekuntia reaali maailmassa. Nyt pystytään ajattelemaan että 1 FPS simulaatioajossa vastaa 1000 FPS.

Tämä tietenkin tarkoittaa simulaation keston pidentämistä, mutta nyt onnistuu LIDARin simulointi entistä tarkemmin. Tässä täytyy tietenkin ottaa huomioon muiden liikkuvien objektien nopeudet, kuten esimerkiksi esteiden liikehdintä ja LIDARin pyöriminen ja vaihtaa niiden nopeuksia tarpeen mukaan.

Koordinaattien paikantamisessa tämä simulaatio toimii hyvin. Ainoaksi ongelmaksi koitui peliobjektin transformaation skaalausarvojen huolimaton vaihtaminen. Jos pitää nämä skaalausarvot arvossa 1, ei pitäisi olla mitään ongelmaa. Tämä kuitenkin tuottaa oikean näköistä pistepilvikuvaa, mutta saadut koordinaattipisteet ovat vain skaalattuna suhteessa näihin arvoihin. Osumakoordinaatit on mahdollista kirjoittaa helposti pistepilvitiedostoon ja koordinaattien tallennus ajon aikana tietorakenteeseen onnistuu myös.

6. YHTEENVETO

Tässä työssä käsiteltiin LIDAR-tekniikkaa ja miten se toimii, sekä pelimoottori Unity3D:n ympäristöä ja sen toimintaa. Työssä toteutettiin simulaatio LIDAR-tekniikasta Unity3D-ympäristössä ideaalilanteessa ja ideaaliolosuhteissa. Tavoitteena oli toteuttaa mahdollisimman oikean maailman etäisyyksiä ja koordinaattipisteitä mittaava LIDAR-järjestelmä muistuttava peliobjekti mahdollisimman hyvällä suorituskyvyllä ja luotettavilla tuloksilla.

Tämän simulaation ohjelmistokoodia muutettiin useaan kertaan. Alkuvaiheessa tämän työn peliobjekti muistutti kokonaisuudessaan Flash LIDARin toimintaa skannaamalla ison näköalueen ympäriltä kerrallaan joka päivitysfunktion kutsu. Tämä oli erittäin raskas ratkaisu. Lopulta se muutettiin ympäri laitteen siivuittain skannaavaksi ja toteutettiin erillinen Flash LIDAR -peliobjekti. Lopputulokseksi saatiin, että LIDARin toimintaa on mahdollista simuloida Unity3D-ympäristössä.

Toteutettiin kaksi LIDAR-peliobjektia: Ympäri kerroksittain skannaava LIDAR sekä yhteen suuntaan osoittava kerralla koko näkökentän skannaava Flash LIDAR. Skannaava LIDAR toteutettiin sekä Update- että FixedUpdate-funktioita käyttäen, kun taas Flash LIDAR toteutettiin pelkästään FixedUpdate-funktion sisällä. Fysiikkaominaisuudet kannattaa Unityssä aina tehdä FixedUpdate -funktion sisällä, jota pystytään kutsumaan aina tietyin vakioaikaväleihin riippumatta ohjelman suorituskyvystä. Saadut pisteet eivät kuitenkaan eroa näiden kahden päivitysfunktion toteutuksen välillä lainkaan.

Pistepilvikuvan luominen ajon jälkeen onnistuu, ja se tuottaa oikean näköistä kuvaa. Ainoa ero on käytetyn 3D-mallinnusohjelmiston koordinaatistosysteemin oikeakätisyys verrattuna Unityn vasemman käden koordinaatistosysteemiin. Huomattiin että skannaava LIDAR aiheuttaa liikkumisesta johtuvaa vääristymää kuvassa, mitä kuuluukin tapahtua. Flash LIDAR skannaa tasaisesti yhteen suuntaan vaihtamatta näkökenttää, niin sen tuottama kuva ei aiheuttanut vääristymää.

Työssä kuitenkin käytettiin vain yhtä pelimoottoria, Unity3D:tä. On varmasti muitakin pelimoottoreita, joista löytyy riittävän hyvä fysiikkamoottori simuloimaan LIDARia. Näitä tuloksia sitten pystyisi vertailemaan keskenään, ja tarkastelemaan, löytyykö mitään suurempia eroja pelimoottorien fysiikkamoottorien välillä.

Kehittämisvaihtoehtoja tähän työhön on esimerkiksi tarkempi fysiikan lakien ja valon etenemisen huomioonottaminen tarkemmin, kuten esimerkiksi valon monitie-etenemisen

huomioonottaminen ja valon heijastumisen tuottamat ongelmat. Pienien partikkeleiden vaikutus lasersäteeseen osuimiin on myös mahdollinen kehitysvaihtoehto. Pistepilvikuvan muodostaminen liikkeessä ajon aikana reaaliaikaisesti, kohteen RGB-arvojen saaminen ja liitettävyys ROSiin ovat myös hyviä jatkokehitysvaihtoehtoja. Myös esiteltyjä aiheita pystyisi selittämään syvällisemmin ja tarkemmin.

LÄHTEET

- [1] G. Likourezos. Jan. 28, 1958: A laser is born. *IEEE Spectrum* 1992;29(5):43.
- [2] D. Lindley. Invention of the Maser and Laser. 2005; Saatavilla: <https://physics.aps.org/story/v15/st4>. Katsottu 20.10. 2021.
- [3] C. V. Poulton, A. Yaacobi, D. B. Cole, M. J. Byrd, M. Raval, D. Vermeulen, et al. Coherent solid-state LIDAR with silicon photonic optical phased arrays. *Opt.Lett.* 2017 Oct;42(20):4091-4094.
- [4] NOAA. What is lidar? 2021; Saatavilla: <https://oceanservice.noaa.gov/facts/lidar.html>. Katsottu 20.10. 2021.
- [5] Unity Technologies, G. Cameron, jacob-platin, michael-pinol, A. Trang, vidur-vij. Robotics simulation in Unity is as easy as 1, 2, 3! 2020; Saatavilla: <https://blog.unity.com/technology/robotics-simulation-in-unity-is-as-easy-as-1-2-3>. Katsottu 20.10. 2021.
- [6] Virtual Tilt. Unity vs Unreal for beginners. 2021; Saatavilla: <https://virtuallytilt.com/unity-vs-unreal-for-beginners/>. Katsottu 20.10. 2021.
- [7] The requirements on pulsed laser diodes for use in atmospheric LiDAR. - 2019 IEEE High Power Diode Lasers and Systems Conference (HPD); 2019.
- [8] Renishaw plc. Optical encoders and LiDAR scanning. 2021; Saatavilla: <https://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244>. Katsottu 20.10. 2021.
- [9] IEEE standard definitions of laser-maser terms. *IEEE Std 586-1980* 1980:0_1.
- [10] Säteilyturvakeskus. Laserluokat. 2019; Saatavilla: <https://www.stuk.fi/aiheet/laserit/laserluokat>. Katsottu 20.10. 2021.
- [11] M. Paul. Introduction to LiDAR. 2019:1.
- [12] VELODYNE LIDAR. A Guide to Lidar Wavelengths for Autonomous Vehicles and Driver Assistance. 2021; Saatavilla: <https://velodynelidar.com/blog/guide-to-lidar-wavelengths/>. Katsottu 20.10. 2021.
- [13] Velodyne Lidar. Puck Manual. 2019; Saatavilla: <https://velodynelidar.com/products/puck/>. Katsottu 20.10. 2021.
- [14] Approaches of Road Boundary and Obstacle Detection Using LIDAR. ; 09; ; 2013.
- [15] Sensor Technology: 2D vs. 3D. 2020; Saatavilla: <https://redzone.com/nr/sensor-technology-2d-vs-3d/>. Katsottu 20.10. 2021.
- [16] A. B. Kamwa. Lidar tulee autoihin - scifistä tulee todellisuutta. 2021; Saatavilla: <https://etn.fi/index.php/13-news/12209-lidar-tulee-autoihin-scifista-tulee-todellisuutta>. Katsottu 20.10. 2021.
- [17] M. Khader, S. Cherian, Texas Instruments. An Introduction to Automotive LIDAR. 2020; Saatavilla: <https://www.ti.com/lit/pdf/slyy150>. Katsottu 20.10. 2021.
- [18] D. Wang, C. Watkins, H. Xie. MEMS Mirrors for LiDAR: A Review. *Micromachines* 2020;11(5).
- [19] H. Sarbolandi, M. Plack, A. Kolb. Pulse Based Time-of-Flight Range Sensing. *Sensors* 2018;18(6).
- [20] R. Lange, P. Seitz. Solid-state time-of-flight range camera. *IEEE Journal of Quantum Electronics* 2001;37(3):390-397.
- [21] Q. Wang, Y. Tan, Z. Mei. Computational Methods of Acquisition and Processing of 3D Point Cloud Data for Construction Applications. *Archives of Computational Methods in Engineering* 2020;27:479-499.
- [22] R. Horaud, M. Hansard, G. Evangelidis, C. M  nier. An overview of depth cameras and range scanners based on time-of-flight technologies. *Mach Vision Appl* 2016;27(7):1005-1020.
- [23] Velodyne Lidar. Puck. 2021; Saatavilla: <https://velodynelidar.com/products/puck/>. Katsottu 20.10. 2021.

- [24] RoboSense. RS-Bpearl. 2020; Saatavilla: <https://www.robosense.ai/en/rslidar/RS-Bpearl>. Katsottu 20.10. 2021.
- [25] Unity Technologies. Unity platform. 2021; Saatavilla: <https://unity.com/products/unity-platform>. Katsottu 20.10. 2021.
- [26] Unity Technologies. Unity personal. 2021; Saatavilla: <https://store.unity.com/products/unity-personal>. Katsottu 20.10. 2021.
- [27] Unity Technologies. Physics. 2021; Saatavilla: <https://docs.unity3d.com/2020.1/Documentation/Manual/PhysicsSection.html>. Katsottu 20.10. 2021.
- [28] Computer Hope. Physics engine. 2019; Saatavilla: <https://www.computerhope.com/jargon/p/physics-engine.htm>. Katsottu 20.10. 2021.
- [29] Unity Technologies. Multiplatform. 2021; Saatavilla: <https://unity.com/features/multiplatform>. Katsottu 20.10. 2021.
- [30] Unity Technologies. Unity's interface. 2021; Saatavilla: <https://docs.unity3d.com/Manual/UsingTheEditor.html>. Katsottu 20.10. 2021.
- [31] Unity Technologies. GameObject. 2017; Saatavilla: <https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html>. Katsottu 20.10. 2021.
- [32] Unity Technologies. Coding in C# in Unity for beginners. 2021; Saatavilla: <https://unity3d.com/learning-c-sharp-in-unity-for-beginners>. Katsottu 20.10. 2021.
- [33] Unity Technologies. Prefabs. 2021; Saatavilla: <https://docs.unity3d.com/Manual/Prefabs.html>. Katsottu 20.10. 2021.
- [34] Unity Technologies. Transforms. 2017; Saatavilla: <https://docs.unity3d.com/560/Documentation/Manual/Transforms.html>. Katsottu 20.10. 2021.
- [35] Unity Technologies. RaycastHit. 2021; Saatavilla: <https://docs.unity3d.com/ScriptReference/RaycastHit.html>. Katsottu 20.10. 2021.
- [36] Unity Technologies. Important Classes - MonoBehaviour. 2021; Saatavilla: <https://docs.unity3d.com/Manual/class-MonoBehaviour.html>. Katsottu 20.10. 2021.
- [37] R. Nystrom. Game loop. 2014; Saatavilla: <https://gameprogrammingpatterns.com/game-loop.html>. Katsottu 20.10. 2021.
- [38] Techopedia. Rendering. 2021; Saatavilla: <https://www.techopedia.com/definition/9163/rendering>. Katsottu 20.10. 2021.
- [39] S. Madhav. Game Programming Algorithms and Techniques: A Platform-Agnostic Approach. 1st ed.: Addison-Wesley Professional; 2013.
- [40] Unity Technologies. Order of execution for event functions. 2019; Saatavilla: <https://docs.unity3d.com/Manual/ExecutionOrder.html>. Katsottu 20.10. 2021.
- [41] Unity Technologies. MonoBehaviour.FixedUpdate(). 2021; Saatavilla: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html>. Katsottu 20.10. 2021.
- [42] Unity Technologies. Time.deltaTime. 2021; Saatavilla: <https://docs.unity3d.com/ScriptReference/Time-deltaTime.html>. Katsottu 20.10. 2021.
- [43] Unity Technologies. Preparing Assets for Unity. 2018; Saatavilla: <https://docs.unity3d.com/2019.3/Documentation/Manual/BestPracticeMakingBelievableVisuals1.html>. Katsottu 20.10. 2021.
- [44] Unity Technologies. Rotation and Orientation in Unity. 2021; Saatavilla: <https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html>. Katsottu 20.10. 2021.
- [45] P. Cignoni, M. Callieri, Corsini Massimiliano, M. Dellepiane, F. Ganovelli, G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. Eurographics Italian Chapter Conference: The Eurographics Association; 2008. p. 129-136.
- [46] Unity Technologies. Unit Testing. 2021; Saatavilla: <https://docs.unity3d.com/Manual/testing-editortestsrunner.html>. Katsottu 20.10. 2021.
- [47] Unity Technologies. Unity Test Runner. 2020; Saatavilla: <https://docs.unity3d.com/2017.4/Documentation/Manual/testing-editortestsrunner.html>. Katsottu 20.10. 2021.
- [48] Unity Technologies. Edit Mode vs. Play Mode tests. 2019; Saatavilla: <https://docs.unity3d.com/Packages/com.unity.test-framework@1.0/manual/edit-mode-vs-play-mode-tests.html>. Katsottu 20.10. 2021.

LIITTEET

LIITE 1: Unityn pelisilmukka kokonaisuudessaan [40]:

