Mikko Kemppainen

3D-renderöinti OpenGL-ohjelmointirajapinnalla

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2017

Jyväskylän yliopisto

Tietotekniikka

Tekijä: Mikko Kemppainen

Yhteystiedot: mikko.t.a.kemppainen@student.jyu.fi

Ohjaaja: Marjaana Nokka

Työn nimi: 3D-renderöinti OpenGL-ohjelmointirajapinnalla

Title in English: 3D-rendering with OpenGL

Työ: Kandidaatintutkielma

Suuntautumisvaihtoehto: Tietotekniikka

Sivumäärä: 23+0

Tiivistelmä: 3D-tietokonegrafiikalla on lukuisia käyttökohteita esimerkiksi mallintamisessa, animaatioissa ja peleissä. Vähänkään vaativampi 3D-renderöinti suoritetaan yleensä näytönohjaimen avulla. Tässä tutkielmassa selvitetään, miten 3D-renderöinti tapahtuu käyttäen Open Graphics Library -rajapintaa. Moderneissa OpenGL:n versioissa grafiikkaa ohjelmoidaan niin kutsuttujen varjostinohjelmien avulla. Tässä tutkielmassa esitellään varjostimien ohjelmoinnin periaatteet ja niiden ohjelmointiin käytettävän ohjelmointikielen OpenGL Shading Languagen käyttö. Tutkielma on toteutettu kirjallisuuskatsauksena ja lähteinä on käytetty pääasiassa konferenssijulkaisuja ja varsinkin OpenGL:n kotisivuilla suositeltuja kirjoja.

Avainsanat: OpenGL, renderöinti, GLSL, tietokonegrafiikka

Abstract: 3D-graphics have numerous uses, such as in modeling, animation and games. All demanding graphics prosessing is usually done with a dedicated graphics processing unit. In this thesis we will find out how 3D rendering works using Open Graphics Library. Modern OpenGL versions use programmable shaders for graphics programming. In this thesis we will introduce the fundamentals of shader programming and the use of OpenGL Shading Language that is used for programming them. This thesis is conducted as a literary review that mostly uses conference publications and particularly books recommended on the home page of OpenGL.

Keywords: OpenGL, rendering, GLSL, computer graphics

Mikko Kemppainen

Kuviot

Sisältö

1 Johdanto

Tämän tutkielman tavoitteena on selvittää, miten 3D-renderöinti modernin näytönohjaimen avulla tapahtuu käyttäen Open Graphics Library -rajapintaa. 3D-renderöinti eli kolmiulotteinen piirtäminen tarkoittaa prosessia, jossa kolmiulotteisesta virtuaalimaisemasta muodostetaan kaksiulotteinen kuva. Näyttö on pohjimmiltaan kaksiulotteinen, joten 3D-maisemalle pitää tehdä erinäisiä transformaatioita, jotta siitä saa muodostettua vakuuttavan kaksiulotteisen kuvan, jonka voi piirtää näytölle. Tapoja esittää ja piirtää 3D-maisemia on useita. Yleisimmin käytetty tapa on polygonipohjainen renderöinti, jota myös OpenGL käyttää. Polygonipohjaisessa renderöinnissä 3D-maisema mallinnetaan polygoniverkon avulla. Verkko sitten rasteroidaan, eli muutetaan pikseleiksi. Muita tapoja renderöidä ovat esimerkiksi säteenseuranta (ray tracing) ja vokselipohjainen renderöinti.

Open Graphics Library eli OpenGL on Chronos Groupin hallinnoima avoin standardi ja alan eniten käytetty 2D- ja 3D-grafiikkaohjelmointirajapinta (Chronos Group 2017b). OpenGL toimii muun muassa Windows, Linux ja macOS käyttöjärjestelmissä. OpenGL for Embedded Systems (OpenGL ES) on OpenGL:n versio, joka on suunniteltu sulautettuihin järjestelmiin ja sitä käytetään varsinkin älypuhelimissa. Web Graphics Library (WebGL), on javascript-rajapinta, joka on läheisesti yhdenmukainen OpenGL ES -rajapinnan kanssa ja toimii useimmissa verkkoselaimissa. Kaikki edellä mainitut rajapinnat poikkeavat toisistaan vain vähän, joten samankaltaisella rajapinnalla voidaan tuottaa grafiikkaa, joka toimii lukuisissa eri kohteissa.

Muita grafiikkaohjelmointirajapintoja ovat esimerkiksi Windowsissa toimiva Direct3D ja Applen käyttöjärjestelmissä toimiva Metal. Näiden rajapintojen käyttö on sidottu käyttöjärjestelmään, joten niiden käyttö on rajattua sovelluksissa, jotka tähtäävät toimintaan useissa eri laitteissa. Chronos Groupin kehittämä Vulkan on myös laitteistoriippumaton rajapinta, jota pidetään OpenGL:n seuraajana. Se ei kuitenkaan soveltuisi erityisen hyvin kirjallisuuskatsauksen aiheeksi, koska se julkaistiin vasta 2016 helmikuussa, eikä siitä ole siten vielä kovin paljoa kirjallisuutta. Näistä syistä tämän tutkielman aiheena on OpenGL-rajapinta, eikä jokin edellä mainituista.

Grafiikkaa tuotetaan OpenGL:n avulla ohjelmoimalla varjostimia (shaders). Varjostimet ovat ohjelmia, joita ajetaan näytönohjaimessa. Yleensä kun puhutaan varjostamisesta, syntyy mielikuva siitä efektistä, joka syntyy, kun valonlähteen edessä on kappale ja pintaan osuu sen takia vähemmän valoa. Kun tässä tutkielmassa puhutaan varjostamisesta, sillä tarkoitetaan pikselien värien muuttamista valonlähteiden ja pinnan materiaalien ja muiden mahdollisten ominaisuuksien perusteella, eikä pelkästään värien tummentamista varjojen vaikutuksesta. Varjostimia ohjelmoidaan käyttämällä OpenGL:n varjostuskieltä nimeltä OpenGL Shading Language, josta käytetään lyhennettä GLSL. GLSL-kielestä kerrotaan tarkemmin luvussa 3.2.

Grafiikan piirtämisessä on monia vaiheita ja niitä voidaan suorittaa samanaikaisesti. Tästä syystä piirtämiseen liittyviä peräkkäisiä vaiheita voidaankin kutsua yhdessä piirtoliukuhihnaksi. Sisääntulona OpenGL:lle on 3D-data, joka kuvaa geometriaa, jota halutaan piirtää ja ulostulona on kuva piirretystä maisemasta.

Perustavaa grafiikkaprosessoinnissa on rinnakkaisuus. Primitiivit, verteksit ja pikselifragmentit ovat helposti rinnakkaistettavissa piirtoliukuhihnan jokaisessa vaiheessa. Esimerkiksi kolmion kukin verteksi voidaan prosessoida rinnakkain, kaksi kolmiota voidaan prosessoida rinnakkain ja kolmion jokainen pikselifragmentti voidaan prosessoida rinnakkain. (Blythe 2008) Tämän vuoksi näytönohjaimet ovatkin kehittyneet sellaisiksi kuin ne ovat eli erittäin tehokkaiksi rinnakkaislaskennassa.

Seuraavassa luvussa kerrotaan, miten OpenGL ja näytönohjaimet ovat kehittyneet vuosien varrella. Luvussa 3 kerrotaan OpenGL-ohjelmoinnin tärkeimmät periaatteet ja varjostimien ohjelmointikielestä GLSL:stä. Luvussa 4 kuvaillaan tarkemmin piirtoliukuhihnassa olevia vaiheita ja mitä niillä voidaan tehdä. Luvussa 5 kerrotaan vielä jonkin verran varjostustekniikoista. Lopuksi esitetään johtopäätökset, joissa kerrotaan varsinkin OpenGL:n tulevaisuuden näkymistä.

2 OpenGL:n kehitys

Näytönohjaimet ovat kehittyneet huimasti siitä, kun ne ensimmäisen kerran ilmaantuivat. Suorituskyvyn kehittymisen lisäksi näytönohjaimen arkkitehtuuri ja tapa piirtää ovat muuttuneet. OpenGL on myös luonnollisesti joutunut kehittymään ja muuttumaan sen mukaan, mitä ominaisuuksia näytönohjaimissa oli tarjolla.

Ensimmäinen versio OpenGL:stä julkaistiin 1994 ja se implementoi kiinteän piirtoliukuhihnan (fixed-function pipeline). Kiinteässä piirtoliukuhihnassa kaikki mahdolliset operaatiot on määritelty etukäteen, joten ainoa tapa muuttaa ohjelman toimintaa on muuttaa syötteitä. Tämä liukuhihna on perusteena monille seuraaville OpenGL:n versiolle, jotka lisäsivät siihen toiminnallisuutta. (Angel ja Shreiner 2013).

Kiinteä piirtoliukuhihna oli hyvin rajoitettu ja tuki vain pientä joukkoa varjostustekniikoita. Lisäksi kullekin varjostintekniikalle täytyi erikseen uhrata huomattava määrä transistoreita. Vastaus näihin ongelmiin oli ohjelmoitava piirtoliukuhihna. (Haines 2006). OpenGL 2.0 lisäsi tuen ohjelmoitaville varjostimille. Käytettävissä olevat varjostimet olivat verteksivarjostin, jonka avulla voi vaikuttaa 3D-geometriaan, ja fragmenttivarjostin, joka pystyy varjostamaan pikseleitä. (Angel ja Shreiner 2013).

Ennen OpenGL:n versiota 3.0 ominaisuuksia oltiin vain lisätty, mutta ei koskaan poistettu. OpenGL 3.0 otti käyttöön ominaisuuksien vanhentamismallin, joka määrittelee, miten ominaisuuksia poistetaan OpenGL:stä. Syynä tälle oli, että jotkin vanhat ominaisuudet eivät ole enää yhtä tehokkaita kuin modernimmat metodit. Tämä myös mahdollisti näytönohjaimen paremman suorituskyvyn sekä yksinkertaistamisen. (Angel ja Shreiner 2013).

OpenGL 3.1 poisti kiinteän piirtoliukuhihnan sekä muita vähäisempiä ominaisuuksia käytöstä. Poistetut ominaisuudet saa kuitenkin vielä takaisin käyttämällä laajennuksia. OpenGL 3.1 versiossa myös vaaditaan, että kaikki data laitetaan puskuriolioihin, jotka ladataan näytönohjaimen muistiin. Tämä muutos auttoi vähentämään erinäisten systeemiarkkitehtuurien rajoituksien vaikutuksia liittyen näytönohjaimiin. (Angel ja Shreiner 2013).

OpenGL 3.2 lisäsi piirtoliukuhihnaan geometriavarjostimen, jonka avulla pystyy manipu-

loimaan geometrisia primitiivejä. OpenGL 3.2 lisäsi myös kontekstiprofiilit, joiden avulla pystyy helpommin valitsemaan, mitä OpenGL:n ominaisuuksia haluaa käyttää. OpenGL 4.1 lisäsi tuen tesselaatiovarjostimille. Tesselointivaihe koostuu kahdesta varjostimesta: tesselaation hallinta- ja laskentavarjostimesta. (Angel ja Shreiner 2013).

Viimeisin OpenGL:n versio on OpenGL 4.5 ja se lisäsi muun muassa suoran pääsyn olioihin ilman, että niitä tarvitsisi sitoa kontekstiin. Tästä on hyötyä varsinkin väliohjelmistoille. OpenGL 4.5 myös paransi vakautta ja lisäsi ominaisuuksia DirectX 11 emulaatiolle. (Chronos Group 2014)

3 OpenGL-ohjelmointi

OpenGL:ää käytetään aina jonkin isäntäkielen, kuten vaikka C++:n avulla. Isäntäkielellä täytyy muun muassa alustaa piirtokonteksti ja ladata 3D-data OpenGL:lle. Varjostimet täytyy myös kääntää ja linkittää. Tässä luvussa tehdään yleiskatsaus OpenGL-ohjelmointiin ja käydään läpi tärkeimmät konseptit.

3.1 Datapuskurit ja primitiivit

Kaikki data OpenGL:ssa täytyy tallentaa puskuriolioihin. Niitä tarvitaan useimpiin asioihin, mitä OpenGL:ssä voidaan tehdä. Ensin täytyy luoda uusi puskuriolio, joka on aluksi oletuksena tyhjä. Puskureiden dataa voidaan manipuloida monin tavoin. Data voidaan muun muassa antaa suoraan, osa datasta voidaan korvata tai dataa voidaan generoida OpenGL:n avulla ja tallentaa toisiin puskureihin. (Shreiner ym. 2013).

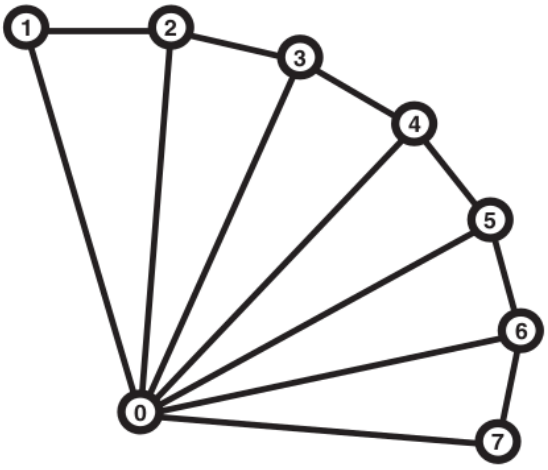
Kun verteksidata on tallennettuna puskuriin, ne voidaan sitten piirtää. Piirrettäessä OpenGL lukee puskurioliosta osan tai kaikki vertekseistä (Shreiner ym. 2013). Piirtämiskomennot käynnistävät sitten piirtoliukuhihnan ja tulos piirretään ruudulle.

Kolmiulotteinen maisema mallinnetaan geometristen primitiivien avulla. Primitiivit OpenGL:n kontekstissa tarkoittavat osia, joista kaikki geometria rakentuu. Primitiivit taas koostuvat vertekseistä eli monikulmion kärjistä. Verteksit ovat pisteitä avaruudessa, joilla voi sijainnin lisäksi olla erinäisiä attribuutteja, kuten väri ja pinnan normaali. Verteksit tallennetaan aina datapuskureihin, joista näytönohjain saa ne käyttöönsä. Kun piirtäminen aloitetaan, samalla määritellään, mitä primitiivejä vertekseistä muodostetaan. Primitiivit voivat olla joko pisteitä, viivoja tai kolmioita. Lisäksi on vielä primitiivi nimeltä ”patch”, jota käytetään tesseloinnissa. Tässä tutkielmassa käytetään patch-primitiivistä englanninkielistä nimitystä, koska sille ei löytynyt sopivaa suomenkielistä käännöstä. Tesseloinnista ja patch-primitiiveistä kerrotaan lisää luvussa 4.2.

Pisteitä piirrettäessä jokainen verteksi puskurissa vastaa aina yhtä pistettä. Pisteet piirretään oletuksena yhden pikselin kokoisena, mutta niiden kokoa voi muuttaa. Pisteille voidaan käyt-

tää myös tekstuureja. Viivat voi piirtää joko siten, että kaksi peräkkäistä verteksiä muodostaa aina erillisen viivan tai vertekseistä muodostetaan yhtenäinen viivaketju tai rengas. Viivojen paksuutta voi myös muuttaa. (Shreiner ym. 2013)

Kolmioiden piirtämiseen on myös useita vaihtoehtoja. Kolmioita voi piirtää yksinkertaisesti siten, että kolme peräkkäistä verteksiä muodostaa aina oman kolmionsa. Kolmioita voidaan piirtää myös siten, että samat verteksit kuuluvat useampiin kolmioihin. Kolmioketjussa kolme ensimmäistä verteksiä muodostavat kolmion ja jokainen uusi verteksi sen jälkeen muodostaa kolmion kahden edellisen verteksin kanssa. Kolmioviuhkassa ensimmäisen kolmion jälkeen jokainen verteksi yhdistetään ensimmäiseen sekä edelliseen verteksiin, mikä saa aikaan viuhkaa muistuttavan kuvion kolmioista, kuten kuviossa 1 näkyy. (Shreiner ym. 2013)



Kuvio 1. Kolmioviuhka (Shreiner ym. 2013)

Kolmioille voidaan määritellä etu- sekä takapuoli ja ne voidaan piirtää eri tavalla riippuen siitä, että kumman puolen katsoja näkee. Oletuksena molemmat puolet kuitenkin piirretään samalla tavalla. Ne kolmiot, joiden verteksien esiintymisjärjestys kulkee vastapäivään, osoittavat oletuksena eteenpäin ja myötäpäivään kulkevat taaksepäin. Tämän pystyy tarvittaessa vaihtamaan toisin päin. (Shreiner ym. 2013) Takapintojen poistovaiheessa voidaan takaperin olevat kolmiot valinnaisesti poistaa. Ne usein poistetaan, koska jos kappale on suljettu, taaksepäin osoittavien kolmioiden edessä on aina eteenpäin osoittavia kolmioita. (Sellers, Wright ja Haemel 2015) Suljetuissa kappaleissa takaperin olevia kolmioita ei siis koskaan

pääse näkemään ja ne poistamalla voidaan säästää prosessointiaikaa.

3.2 GLSL-kieli

GLSL-kieltä (OpenGL Shading Language) käytetään varjostimien ohjelmointiin ja se muistuttaa C-ohjelmointikieltä. Käytettävissä ovat muun muassa samanlaiset ehtolauseet ja silmukkarakenteet. Kielessä on myös esiprosessori, joka muistuttaa C-kielen vastaavaa. GLSL määrittelee lisäksi useita uusia datatyyppejä vektoreille ja matriiseille. (Shreiner ym. 2013)

3.2.1 Varjostimien kääntäminen

Varjostimien tekeminen OpenGL-ohjelmaan muistuttaa käännettävien ohjelmointikielien, kuten C-ohjelmointikielen käyttämistä. Varjostimia varten täytyy luoda OpenGL-olioita ja liittää ne varjostinohjelmaan. Olioita luodaan useita, koska samoin kuin muissakin ohjelmointikielissä samoja funktioita voidaan käyttää useissa eri paikoissa. (Shreiner ym. 2013) Varjostimien yhteistä toiminnallisuutta ei siten tarvitse kirjoittaa jokaiseen varjostimeen erikseen.

Kääntäjä analysoi koodin ja tarkistaa onko siinä virheitä ja kääntää sen sitten OpenGL-olioksi. Oliot liitetään sitten varjostinohjelmaan. Lopuksi varjostinohjelma linkitetään ajettavaksi ohjelmaksi. Vielä täytyy tarkistaa, että onnistuiko linkitys, koska linkitysvaiheessa voi olla virheitä. Varjostinta voidaan sen jälkeen käyttää piirtoliukuhihnassa. (Shreiner ym. 2013)

3.2.2 Erityismuuttujat

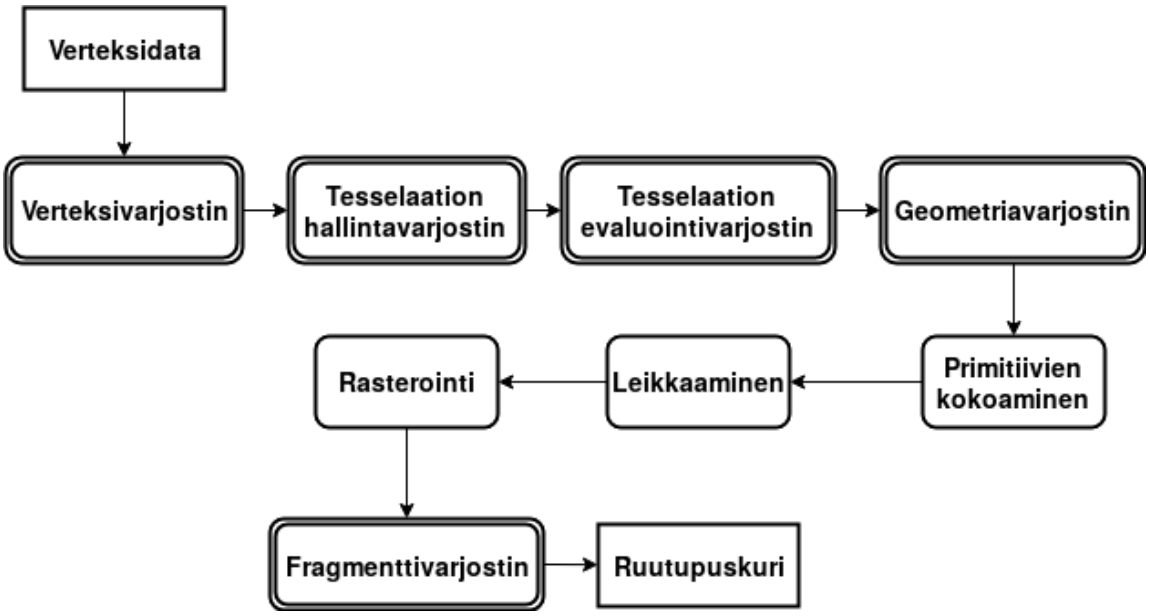
Varjostimiin pitää saada jollain tavalla data sisään sekä ulos. Tähän tarkoitukseen on käytettävissä erityiset sisään- ja ulostulomuuttujat sekä uniform-muuttujat. Sisääntulomuuttujat sisältävät tiedon, joka tulee varjostimelle. Ne yhdistävät piirtoliukuhihnan eri vaiheet toisiinsa yhdessä ulostulomuuttujien kanssa. Kaikki varjostimet tekevät joitain asioita syötteillään ja tulokset kirjoitetaan aina ulostulomuuttujiin. Esimerkiksi verteksivarjostimen ulostulona on aina vähintään verteksin sijainti ja yleensä muitakin verteksiattribuutteja. (Rodríguez 2013)

Uniform-määreellä merkittyjen muuttujien arvo pysyy samana renderöintikerran aikana ja ne ovat käytettävissä piirtoliukuhihnan joka vaiheessa. Niiden avulla saadaan varjostimelle tieto muun muassa valonlähteistä. Ohjelmoitavissa varjostimissa kun ei ole sisäänrakennettua konseptia valonlähteille.(Rodríguez 2013).

Uniform-muuttujia voidaan muuttaa jokaisella erillisellä renderöintikerralla. Ne voidaan ladata siis uudelleen jokaisen piirtokutsun välissä. Niitä voi käyttää esimerkiksi ajan kulumisen esittämiseen. Esimerkiksi jonkin valonlähteen kirkkautta voidaan muuttaa ajan funktiona. Niitä on pakko käyttää joihinkin asioihin kuten tekstuurien tallentamiseen.

4 Piirtoliukuhihna

3D-kuvan piirtämisessä on monia vaiheita ja näytönohjaimet ovat kehittyneet piirtämään ne tietyllä tavalla. Nykyään useimpia vaiheita pystytään ohjelmoimaan. Ainoat ohjelmoitavat vaiheet, mitä on pakko määritellä ovat verteksivarjostin ja fragmenttivarjostin. Kuviossa 2 näkyy yksinkertaistettuna näytönohjaimen piirtoliukuhihna, niin kuin OpenGL sen ymmärtää. Tässä luvussa käydään ensin pinnallisesti läpi näytönohjaimen piirtoliukuhihna ja sen jälkeen aliluvuissa kerrotaan piirtoliukuhihnan vaiheista tarkemmin.



Kuvio 2. Piirtoliukuhihna

Piirtoliukuhihna voidaan jakaa karkeasti kahteen osaan: geometriavaiheeseen, jossa käsitellään 3D-dataa eli verteksejä ja primitiivejä, ja rasterointivaiheeseen, jossa käsitellään fragmentteja eli pikseleitä. Geometriavaiheessa voidaan manipuloida verteksejä ja primitiivejä eri tavoin. Tähän vaiheeseen kuuluvat verteksivarjostin, tesselaatiovarjostin ja geometriavarjostin. Geometriavaiheessa koordinaatit siirretään näytön koordinaatteihin ja geometriasta muodostetaan niin kutsuttu rautalankamalli.

Rasterointivaiheessa rautalankamallin kolmioihin luodaan pinnat ja täytetään fragmenteilla. Sitten fragmenttivarjostin prosessoi jokaisen fragmentin ja suorittaa lopullisen varjostuksen. Prosessoidut fragmentit lähetetään sitten ruutupuskuriin, joka piirretään mahdollisesti näy-

tölle.

4.1 Verteksivarjostin ja koordinaatistot

Tyypillisesti 3D-data on määritelty oliokeskisessä koordinaatistossa eli koordinaatiston origon on yleensä olion keskellä tai jossain muussa paikassa, josta sitä on loogista kiertää. Origo ei kuitenkaan yleensä ole kovin kaukana oliosta, koska muutoin transformaatiot aiheuttaisivat samalla turhia siirtymiä. Nämä koordinaatit täytyy muuntaa sellaisiksi, että maisemalla on yksi globaali origo, johon verrattuna kaikki oliot on määritelty. Näitä koordinaatteja kutsutaan maailmakoordinaatistoksi (world space) ja tässä koordinaatistossa tyypillisesti suoritetaan varjostusta.(Shreiner ym. 2013; Sellers, Wright ja Haemel 2015)

Maailmakoordinaatisto voidaan muuntaa vielä katsojan koordinaatistoon, jossa katsoja (tai kamera) on koordinaatiston origossa. Siitä on hyötyä joissain tilanteissa. Geometriavaiheen lopuksi koordinaattien täytyy olla aina kuvaruutukoordinaatistossa, jossa koordinaatit vastaavat sijainteja näytöllä. Koordinaattien muunnokset suoritetaan kertomalla koordinaatteja muunnosmatriiseilla. (Shreiner ym. 2013; Sellers, Wright ja Haemel 2015) Näistä muunnoksista ja siihen liittyvistä matriisilaskennasta ei tässä tutkielmassa kerrota tarkemmin, koska se on hieman tutkielman rajauksen ulkopuolella.

Koordinaattien muutos kuvaruutukoordinaatteihin täytyy suorittaa geometriavaiheen aikana. Tämä ei ole minkään tietyn varjostimen tehtävä, koska tesselaatio- ja geometriavarjostin ovat vapaavalintaisia. Jos tesselaatio- ja geometriavarjostimia ei ole määritelty, se on verteksivarjostimen tehtävä.

Verteksivarjostin käsittelee verteksejä yksi kerrallaan. Se ei pysty saamaan tietoa muista vertekseistä, eikä se tiedä primitiiveistä. Sen tehtävänä on ottaa syötteenä yksi verteksi ja antaa ulostulona yksi verteksi. Verteksivarjostin saa syötteeksi sisääntulomuuttujissa verteksiattributteja ja se pystyy käyttämään myös uniform-muuttujia. Yleisimpiä verteksiominaisuuksia ovat tekstuurikoordinaatit, normaalit sekä värit. Koska OpenGL on kehittynyt olemaan mahdollisimman joustava, verteksiattribuutteihin pystyy laittamaan periaatteessa mitä tahansa ominaisuuksia. (Rodríguez 2013)

4.2 Tesselaatio- ja geometriavarjostin

Tesselaatiovarjostin on vapaavalintainen osa piirtoliukuhihnaa. Tesselointi on prosessi, jossa korkeamman tason primitiivi jaetaan pienempiin primitiiveihin, kuten kolmioihin. Tesselaatiota voidaan käyttää muun muassa yksityiskohtien tason (Level of detail) säätämiseen (Gomes ym. 2012). Tesselaatiovaihe on järjestyksessä verteksivarjostimen jälkeen ja se koostuu kahdesta ohjelmoitavasta osasta: tesselaation hallintavarjostimesta ja evaluointivarjostimesta. Niiden välissä on tesselaatiomoottori, joka on kiinteä osa tesselaatiovaihetta. (Sellers, Wright ja Haemel 2015)

Tesselaatiovarjostin prosessoi ainoastaan patch-primitiivejä. Jos piirtoliukuhihnalla, jossa on määriteltynä tesselaatiovarjostin, yritetään prosessoida muita primitiivejä kuin patch-primitiivejä, se tuottaa virheen. Samoin syntyy virhe, jos patch-primitiivejä yritetään prosessoida liukuhihnalla, jossa ei ole määriteltynä tesselaatiovarjostinta. (Sellers, Wright ja Haemel 2015; Shreiner ym. 2013; Chronos Group 2017a)

Patch-primitiivit ovat yksinkertaisesti listoja vertekseistä. Toisin kuin muista primitiiveistä patch-primitiiveistä ei voida suoraan tietää, että kuinka monta peräkkäistä verteksiä kuuluu samaan primitiiviin. Se täytyy eksplisiittisesti määrätä, kun niitä aletaan piirtämään. (Sellers, Wright ja Haemel 2015; Shreiner ym. 2013; Chronos Group 2017a)

Tesselaation hallintavarjostin saa sisääntuloksi patch-primitiivin ja antaa patch-primitiivin myös ulostuloksi. Se pystyy muuttamaan ulostuloprimitiivin verteksien määrää, mutta määrä pitää olla määritetty ohjelman linkitysvaiheessa. Tesselaation hallintavarjostin päättää myös tesselaation määrän, eli moneenko osaan primitiivit jaetaan. Tämä suoritetaan asettamalla attribuutit, jotka määrittävät tesselointitasot. Varjostin ajetaan kerran jokaiselle ulostuloverteksille ja se generoi niille attribuutit ja sijainnit. (Sellers, Wright ja Haemel 2015; Shreiner ym. 2013; Chronos Group 2017a)

Seuraavaksi tesselaatiomoottori jakaa tesselaation hallintavarjostimen tuottamat primitiivit pienempiin primitiiveihin määrättyjen tesselaatiotasojen perusteella. Tesselaation evaluointivarjostin ajetaan sitten jokaiselle tuotetulle verteksille ja se määrittää niiden sijainnit ja muut attribuutit. Evaluointivarjostimen tehtävänä on myös siirtää verteksikoordinaatit näytön koordinaatteihin, ellei geometriavarjostinta ole määritelty. (Sellers, Wright ja Haemel

2015; Shreiner ym. 2013; Chronos Group 2017a)

Geometriavarjostin on järjestyksessä tesselaatiovaiheen jälkeen. Geometriavarjostin saa sisääntulona kokonaisia primitiivejä kokoelmana verteksejä. Geometriavarjostin pystyy tuottamaan vaihtelevan määrän verteksejä eli se pystyy poistamaan ja lisäämään verteksejä. Geometriavarjostin pystyy myös muuttamaan primitiivien tyyppejä. Esimerkiksi kolmiot voidaan muuttaa pisteiksi. Se pääsee käsiksi myös tarvittaessa primitiivin viereisiin primitiiveihin. Esimerkiksi geometriavarjostin voi nähdä kolmion sekä viereiset kolme kolmiota eli se näkee parhaimmillaan kuusi verteksiä. (Sellers, Wright ja Haemel 2015; Shreiner ym. 2013)

4.3 Primitiivien leikkaus, poisto ja rasterointi

Kolmiulotteisesta maisemasta pystytään näkemään aina vain osa. Piirtämistä näytölle voidaankin verrata kuvan ottamiseen kameralla. Osa maisemasta, jonka kamera näkee on pyramidin muotoinen alue, jonka kärki on kameran linssissä ja laajenee kohti horisonttia. OpenGL leikkaa lisäksi näköalueesta pois liian kaukana ja liian lähellä olevat osat. Erittäin lähellä kameraa olevat kappaleet näyttävät kasvavan äärimmäisen suuriksi ja liian kaukana olevia kappaleita ei kannata piirtää muun muassa suorituskyvyn takia. Tätä leikatun pyramidin muotoista aluetta kutsutaan näköfrustrumiksi. (Shreiner ym. 2013; Sellers, Wright ja Haemel 2015)

Kaikki näköfrustrumin ulkopuolelle jäävä geometria poistetaan, koska sitä ei pystytä näkemään ja sen prosessointi pidemmälle olisi turhaa. Osittain frustrumin sisälle jäävät kappaleet leikataan. OpenGL tuottaa uutta geometriaa primitiivien näköfrustrumin sisään jäävästä osasta ja primitiiviä leikkaavasta tasosta. (Shreiner ym. 2013; Sellers, Wright ja Haemel 2015).

Rasterointivaiheessa rautalankamallin tasot päällystetään. Verteksien perusteella data muutetaan diskreetiksi 2D-dataksi eli fragmenteiksi. Rasteroija määrittelee, mitkä ruudun pikselit geometria peittää. Rasteroija interpoloi lineaarisesti datan jokaiselle muuttujalle ja lähettää nämä fragmentit fragmenttivarjostimelle. Interpolointi tarkoittaa sitä, että mitä lähempänä fragmentin sijainti on verteksiä, sitä lähempänä fragmentin attribuuttien arvot ovat sen verteksin attribuuttien arvoja. (Shreiner ym. 2013)

4.4 Fragmenttivarjostin ja pikselioperaatiot

Fragmenttivarjostin saa rasteroijalta fragmentit. Se määrittelee fragmenttien lopullisen värin ennen kuin ne piirretään ruutuun (Sellers, Wright ja Haemel 2015). Fragmenttivarjostimessa suoritetaan hienostuneempi varjostus ja voidaan saada aikaan monenlaisia efektejä. Varjostus joka suoritetaan fragmenttivarjostimessa on kuitenkin prosessori-intensiivistä, koska fragmentteja voi syntyä kustakin osasta geometriaa potentiaalisesti miljoonia ja varjostin ajetaan jokaiselle fragmentille erikseen. Luvussa 5 mainitut varjostustekniikat suoritetaan kaikki fragmenttivarjostimessa, ellei toisin mainita.

Ennen kuin fragmenttivarjostimen tuottamat fragmentit piirretään ruutupuskuriin, niille voidaan tehdä erinäisiä operaatioita. Fragmentit voidaan poistaa määrittämällä alue, jonka ulkopuolella olevia fragmentteja ei piirretä. Voidaan myös käyttää puskuria, jossa on kullekin pikselille määrätty arvo ja sen mukaan valitaan piirretäänkö fragmentti vai ei. Syvyystestissä fragmentin z-koordinaattia eli etäisyyttä kamerasta verrataan syvyyspuskurissa olevaan arvoon. Syvyyspuskuri sisältää jokaisen pikselin etäisyyden kameraan. Syvyystestissä verrataan fragmentin z-koordinaattia syvyyspuskurissa olevaan arvoon, ja jos fragmentin arvo on pienempi kuin syvyyspuskurissa oleva arvo, se piirretään ruutupuskuriin. Fragmentin väri voidaan myös yhdistää jo ruutupuskurissa olevaan väriin. Värin yhdistämistä voi säätää monin tavoin OpenGL:ssä. (Sellers, Wright ja Haemel 2015)

5 Varjostus

Tapoja valaista on useita, mutta tässä tutkielmassa käsitellään tarkemmin vain Phong mallia. Aikaisemmin käytössä olleessa OpenGL:n kiinteässä piirtohihnassa käytettiin muokattua Phong-mallia. Toinen esimerkki valaistusmalleista on Gouraud-varjostus, joka suoritetaan verteksivarjostimessa. Phong-varjostus on tarkempaa kuin Gouraud, mutta myös prosessori-intensiivisempää, koska se ajetaan jokaiselle fragmentille, kun taas Gouraud-varjostus ajetaan jokaiselle verteksille.

5.1 Phong-valaistusmalli

Phong-valaistusmallissa kokonaisvalaistus koostuu kolmesta eri komponentista: ympäristön valosta (ambient), hajaantuvasta valosta (diffuse) ja spekulaarisesta (specular) eli heijastuvasta valosta. Ympäristön valo ei tule erityisesti mistään suunnasta ja edustaa likimääräistä ympäristön valoa. Se voidaan laskea valonlähteiden yhteisvaikutuksena tai laskea etukäteen yhtenä globaalina efektinä. Koska ympäristön valo ei muutu eri primitiivien välillä se voidaan antaa varjostimelle uniform-muuttujalla. (Shreiner ym. 2013)

Hajaantuva valo heijastuu pinnasta joka suuntaan (Shreiner ym. 2013). Tämä valaistus on sitä voimakkaampi, mitä suorempaan valo valaisee pintaa. Hajaantuvan valon laskemiseen tarvitaan pinnan normaalia sekä valolahteen sijaintia, mutta ei katsojan sijaintia. Se riippuu myös pinnan väristä. (Shreiner ym. 2013). Hajaantuva valaistus vastaa siis sitä, kun mattapintaista tasoa valaistaan.

Spekulaarinen heijastus on valoa, joka heijastuu suoraan pinnasta. Tätä ominaisuutta heijastaa valoa kutsutaan kiiltävyydeksi. Spekulaaristen heijastusten laskemiseen täytyy tietää pinnan ja valonlahteen kulma suhteessa katsojaan. Siihen tarvitaan tieto pinnan normaalista, valonlahteen suunnasta ja katsojan sijainnista.(Shreiner ym. 2013)

Phong valaistusmallin tekee hyvin mitä se yrittää. Se mallintaa pintojen heijastukset ja hajaantuvan valon ja värit kohtalaisen realistisesti. Siinä on kuitenkin rajoituksensa. Tällä mallilla ei voida mallintaa varjoja, jotka syntyvät kun toinen kappale on valonlahteen edessä.

Ympäristön valoa ei voida myöskään mallintaa kovin tarkasti. (Shreiner ym. 2013)

5.2 Valonlähteiden tyypit

Kaikkein yksinkertaisin tapa valaista on täysin ilman valonlähteitä. Kaikki ei ole kuitenkaan mustaa, vaan vertekseille annetaan värit ja ne interpoloidaan rasteroidessa. Väri voi tulla myös tekstuureista tai materiaalien ominaisuuksista verteksiattribuuttien sijaan. (Shreiner ym. 2013)

Jos valonlähde on todella kaukana, siitä tuleva valo voidaan approksimoida olemaan samassa kulmassa jokaiseen tason pisteeseen nähden. Tällaisia valonlähteitä kutsutaan suuntavalonlähteiksi (directional light) ja niiden implementointi on helpompaa ja laskenta on nopeampaa kuin muun tyyppisten valonlähteiden. Tällaisella valonlähteellä voidaan mallintaa esimerkiksi aurinkoa. (Shreiner ym. 2013).

Pistevalonlähteet ovat valonlähteitä, jotka ovat maiseman lähellä tai sisällä, kuten lamput tai katuvalot. Pistevalonlähteiden välinen kulma jokaiseen tason pisteeseen on eri, joten niitä ei voida esittää uniform-muuttujilla. Lisäksi valon voimakkuus vähenee siirryttäessä kauemmaksi valonlähteestä. Spottivalo (spotlight) muistuttaa pistevalonlähdettä, mutta sen valokeila on rajattu näkymään vain tietyssä kulmassa. (Shreiner ym. 2013).

5.3 Tekstuurit

Tekstuurien voidaan ajatella olevan kuvia, joilla pinnat voidaan päällystää (Shreiner ym. 2013). Niillä voidaan saada helposti pinta näyttämään esimerkiksi puulta. Tekstuureita käytetään fragmenttivarjostimessa pikseleiden värien määrittelemisessä. On myös tekniikoita, joissa tekstuureihin on värien sijaan tallennettuna jotain muuta, kuten esimerkiksi tietoa pinnan normaaleista. (Haines 2006)

Monitekstuurointi on tekniikka, jossa samalla kertaa käytetään useampaa tekstuuria. Näytönohjain pystyy samalla kertaa laittamaan tasolle monta eri tekstuuria kerralla. On nopeampaa laittaa yhdellä renderöintikerralla monta tekstuuria pinnalle sen sijaan, että laitettaisiin yksittäisiä tekstuureja monessa vaiheessa. Moniteksturoinnilla on monia käyttökohteita: ob-

jekteihin voi liittää siirtokuvia (decals), ympäristöön saa lisää yksityiskohtia ja niin edelleen.(Haines 2006).

Epätasaisuuksien teksturointi (bump mapping) on tekniikka, jossa tekstuuriin on tallennettuna värin sijaan pinnan normaali kussakin pisteessä. Tekstuurissa normaalisti olevia RGB-väriarvoja nollasta 255:teen vastaavat normaalivektorin koordinaatit -1:stä 1:teen. Esimerkiksi RGB-vektoria (58, 128, 235) vastaa normaalivektori (-0.65, 0.00, 0.84). (Haines 2006).

6 Yhteenveto

Tässä tutkielmassa esiteltiin, mitä on 3D-renderöinti ja miten sitä OpenGL:n tapauksessa ohjelmoidaan. OpenGL:n ohjelmoinnista kerrottiin pääkohdat ja millainen näytönohjaimen piirtoliukuhihna on. Renderöinnin eri vaiheet käytiin läpi ja selvitettiin, mitä niissä voi tehdä. Grafiikkaohjelmoinnissa tarvitsee tietää kohtalaisen paljon lineaari- ja matriisialgebraa, mutta tässä tutkielmassa tämä sivuutettiin ja keskityttiin enemmän grafiikkaohjelmoinnin pääperiaatteisiin.

OpenGL on kehittynyt ja muuttunut paljon siitä, kun se aluksi julkaistiin. Sen on täytynyt muuttua näytönohjaimien kehityksen myötä ja lisätä tuki uusille näytönohjaimen ohjelmoitaville osille. OpenGL on yli 20 vuotta vanha, joten sille on kertynyt ominaisuuksia, joita ei nykyaikana ole järkevää käyttää.

Vulkanin myötä OpenGL ei ole enää ainoa vaihtoehto laitteistoriippumattomalle grafiikkaohjelmointirajapinnalle. Vulkan, kuten OpenGL:kin, on Khronos Groupin kehittämä ohjelmointirajapinta. Vulkania on jo alettu käyttämään joissain peleissä ja pelimoottoreissa. Tulevaisuudessa Vulkan tulee korvaamaan OpenGL:n osassa käyttökohteista, varsinkin peleissä ja muissa korkeaa suorituskykyä vaativissa kohteissa. Koska Vulkan on matalamman tason ohjelmointirajapinta, sillä pystytään saavuttamaan parempi suorituskyky ja siinä on enemmän mahdollisuuksia optimointiin.

OpenGL ei ole kuitenkaan häviämässä minnekään, ainakaan lähiaikoina. Monet sovellukset eivät tarvitse Vulkanin parempaa suorituskykyä, joten niillä ei ole tarvetta luopua OpenGL:stä. Lisäksi OpenGL on korkeamman tason rajapinta, joten sitä on myös helpompi käyttää kuin Vulkania. Jatkotutkimuksen voisi vertailla OpenGL:n ja Vulkanin ja mahdollisesti muiden rajapintojen hyötyjä erilaisissa käyttökohteissa.

Lähteet

Angel, Edward, ja Dave Shreiner. 2013. “An introduction to OpenGL programming”. Teoksessa *ACM SIGGRAPH 2013 Courses,* 3. ACM.

Blythe, D. 2008. “Rise of the Graphics Processor”. ID: 1, *Proceedings of the IEEE* 96 (5): 761–778. doi:10.1109/JPROC.2008.917718.

Chronos Group. 2014. “Khronos Group Announces Key Advances in OpenGL Ecosystem”. Elokuu. Viitattu 25. huhtikuuta 2017. https://www.khronos.org/news/press/khronos-group-announces-key-advances-in-opengl-ecosystem.

———. 2017a. “The OpenGL Graphics System: A Specification (Version 4.5 (Core Profile) - October 24, 2016)”. Maaliskuu. Viitattu 23. maaliskuuta 2017. https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf.

———. 2017b. “The OpenGL homepage”. Maaliskuu. Viitattu 23. maaliskuuta 2017. https://www.opengl.org/.

Gomes, T., L. Estevo, R. de Toledo ja P. R. Cavalcanti. 2012. “A Survey of GLSL Examples”. Teoksessa *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials,* 60–73. ID: 1. doi:10.1109/SIBGRAPI-T.2012.11.

Haines, E. 2006. “An introductory tour of interactive rendering”. ID: 1, *IEEE Computer Graphics and Applications* 26 (1): 76–87. doi:10.1109/MCG.2006.9.

Rodríguez, Jacobo. 2013. *GLSL Essentials* [kielellä English]. ID: 10825528. Olton, GB: Packt Publishing. ISBN: 9781849698016.

Sellers, Graham, Richard S. Wright ja Nicholas Haemel. 2015. *OpenGL Superbible: Comprehensive Tutorial and Reference.* 7th. Addison-Wesley Professional. ISBN: 0672337479, 9780672337475.

Shreiner, Dave, Graham Sellers, John Kessenich ja Bill Licea-Kane. 2013. *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3.* Addison-Wesley.