Sauli Korhonen

VARJOSTINOHJELMIEN KÄYTTÖÖN PERUSTUVAN REAALIAIKAISEN GRAFIIKAN OHJELMOINTI

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2013

TIIVISTELMÄ

Korhonen, Sauli
Varjostinohjelmien käyttöön perustuvan reaaliaikaisen grafiikan ohjelmointi
Jyväskylä: Jyväskylän yliopisto, 2013, 31 s.
Tietojärjestelmätiede, kandidaatin tutkielma
Ohjaaja: Halttunen, Veikko

Reaaliaikaisen tietokonegrafiikan näyttävyyden kasvu esimerkiksi tietokonepeleissä on ollut nopeaa. Grafiikan piirrosta vastaava näytönohjain onkin tällä hetkellä tietokoneen laitteiston komponenteista nopeimmin kehitetty. Tehokkuuden lisääntyminen on saavutettu erityisesti rinnakkaisten suoritinytimien määrän lisäyksellä: siinä missä mikrotietokoneiden keskussuorittimissa käytetään tällä hetkellä kahdesta kuuteen laskentaydintä, on näytönohjaimissa ytimien lukumäärä useita satoja tai jopa tuhansia. Näytönohjainten arkkitehtuurin muutoksen myötä myös ohjelmointiin käytettäviä menetelmiä on ollut tarpeen kehittää.

Tässä tutkielmassa tarkastellaan varjostinohjelmien käyttöön perustuvaa tapaa ohjelmoida reaaliaikaista grafiikkaa. Varjostinohjelmien käyttöönotto reaaliaikaisen grafiikan ohjelmoinnissa on merkittävä uudistus, joka vaatii ohjelmoijilta uudenlaisen menetelmän omaksumista. Tutkielmassa luodaan katsaus kolmiulotteisen reaaliaikaisen grafiikan periaatteisiin, käsitellään varjostinohjelmien käyttöön perustuvan ohjelmointitavan myötä tehtyjä keskeisiä muutoksia ja arvioidaan niiden vaikutuksia.

Asiasanat: Tietokonegrafiikka, tosiaikakäsittely, näytönohjaimet

ABSTRACT

Korhonen, Sauli
Real-time graphics programming with shader programs
Jyväskylä: University of Jyväskylä, 2013, 31 p.
Information Systems, Bachelor's Thesis
Supervisor: Halttunen, Veikko

The rise of visual impressiveness of real-time computer graphics, for instance in computer games, has been rapid. Display adapters, with which graphics processing is performed, have been developed faster than any other computer hardware component. The increase in processing power has been achieved by increasing the amount of parallel graphics processing cores: when a workstation central processing unit has usually 2-6 computation cores, a fast display adapter has hundreds or even thousands of them. As the hardware architecture of display adapters has changed, graphics programming methods have had to be altered as well.

In this thesis I focus on the method of using shader programs in real-time computer graphics programming. The introduction of shader programs has been a significant development step, which requires developers to adopt new programming methods. This paper provides an overview of modern three-dimensional real-time graphics. The changes that were introduced with shader programming are discussed together with the effects they have had on graphics programming.

Keywords: computer graphics, real-time processing, display adapters

KUVIOT

SISÄLLYS

1 JOHDANTO

Tietokoneiden ominaisuuksiin kohdistettujen vaatimusten lisääntyminen, yritysmaailman kilpailu sekä teknologisen kehityksen tuomat uudet mahdollisuudet toimivat uusien teknisten ratkaisujen kehittämisessä merkittävinä ajavina voimina. Muun muassa esityshetkellä koneellisesti piirretyn eli reaaliaikaisen tietokonegrafiikan alueella on ohjelmoinnin lähtökohta muuttunut merkittävästi. 2000-luvun alussa käynnistyneessä kehityksessä grafiikan piirtäminen pyritään tekemään näytönohjaimissa suoritettavilla erityisillä varjostinohjelmilla (Luebke & Humphreys, 2007; Rost ym, 2009). Varjostinohjelmat ovat näytönohjaimella suoritettavia grafiikan piirtoa ohjaavia ohjelmia.

Vaikka varjotehosteiden piirto onkin yksi tyypillinen varjostinohjelmien käyttökohde, on suomen kielessä vakiintunut *varjostin*-termi hieman yksipuolinen. Varjostimen englanninkielisen termin *shader* voi lukea käsittävän sekä varjostamisen että sävyttämisen. Grafiikkatehosteiden laskeminen on pohjimmiltaan kuvaruudun värisävyjen manipulointia, minkä vuoksi *sävytin*-termi voisi olla kuvaavampi vaihtoehto. Tässä tutkielmassa käytetään vakiintunutta varjostin-termiä.

Tietokoneiden laskentatehon kasvu tunnettua Mooren lakia mukaillen sekä menetelmien ja algoritmien kehittyminen on johtanut siihen, että tavallisilla mikrotietokoneilla voidaan suorittaa reaaliajassa sellaista laskentaa joka aikaisemmin olisi vaatinut pitkällisen esilaskennan. Reaaliaikaisen tietokonegrafiikan alueella laskentatehon kasvu on ollut vielä huomattavasti Mooren lakiin pohjautuvaa tietokoneiden suorituskyvyn kasvuolettamaa nopeampaa (Geer, 2005).

Erityisen konkreettisella tavalla grafiikkaominaisuuksien kehitys näyttäytyy tietokonepelien visuaalisena kehityksenä. Pelialan ja grafiikkateknologian suhde on sikäli tiivis, että peliteollisuuden ajamana kuluttajahintaisten ja yleisesti saatavilla olevien mutta silti tehokkaiden näytönohjainten kehitys on ollut nopeaa (Rhyne, 2000; Xubo, Milo & Xiaoyue, 2009). Näyttävä graafinen ulkoasu tuo peleihin useimmiten lisäarvoa ja toimii alalla vahvana kilpailuetuna.

Tässä työssä käsitellään reaaliaikaisen grafiikan esittämiseen kykenevien laitealustojen joukosta mikrotietokoneita. Tutkielman ulkopuolelle jäävät esi-

merkiksi pelikonsolit ja älypuhelimet, vaikka samat periaatteet ovat niihinkin sovellettavissa. Tutkielman painotus on sellaisen grafiikan ohjelmoinnissa, jonka piirron laatua ja tarkkuutta vähennetään piirtonopeuden hyväksi.

Yhden määritelmän mukaan reaaliaikaisen grafiikan ehtona on, että se kuvastaa kolmiulotteista virtuaalista maailmaa tai tilaa, johon käyttäjä on interaktiivisessa vaikutussuhteessa (Akenine-Moller, Haines & Hoffman, 2008). Tässä tutkielmassa käytetään reaaliaikaisen grafiikan yhteydessä tätä määritelmää. Tiukaksi määritelmän tekee vaatimus siitä, että reaaliaikaisen grafiikan on esitettävä kolmiulotteista tilaa.

Grafiikan ohjelmointia käsitellään Open Graphics Layer (OpenGL) -rajapinnan mukaisesti. OpenGL on Khronos Groupin hallinnoima grafiikan ohjelmointiin tarkoitettu rajapinta. OpenGL:n yhteyteen kuuluu varjostinohjelmien ohjelmointikieli OpenGL Shading Language (GLSL). OpenGL-rajapinnan valintaa tukee muun muassa sen avoin luonne, toteutus useille eri laitealustoille, rojaltivapaus ja hallitseva asema graafisten ammattilaissovellusten käyttämänä grafiikkarajapintana (Lengyel, 2011).

Tutkielman tutkimusongelma on: kuinka reaaliaikaisen grafiikan ohjelmointi on muuttunut varjostinohjelmien käyttöön perustuvan menetelmän myötä. Tutkielmassa luodaan katsaus reaaliaikaisen grafiikan vallitseviin menetelmiin ja keskeisiin viimeaikaisiin muutoksiin sekä arvioidaan muutoksiin liittyviä etuja ja haittapuolia. Tutkielma toimii samalla tietolähteenä nykyaikaisen reaaliaikagrafiikan periaatteista.

Tutkielma on suoritettu tutustumalla alan kirjallisuuteen ja tutkimustyöhön. Laajempaa näkökulmaa erilaisista sovellusalueista on kerätty tutustumalla tietokonegrafiikkaa käsitteleviin tieteellisiin aikakauslehtiin ja konferenssijulkaisuihin. Teknisen tason tietoa on kerätty tutustumalla ohjelmaesimerkkeihin sekä grafiikkarajapintojen ja -kirjastojen dokumentaatioihin.

Alan kehityksen suuntaus osoittaa, että varjostinohjelmia hyödyntävä grafiikkaohjelmointi tulee olemaan vallitseva tapa (Maestri, 2009). Siksi on tärkeää, että tietokonegrafiikan parissa työskentelevät ohjelmoijat tuntevat uudet menetelmät, siitäkin huolimatta että tietokonegrafiikan ohjelmointi on vaikeaksi ja sekavaksi koettu aihealue. Grafiikkaohjelmointi on merkittävä sovelluskehityksen osa-alue tuotettaessa esimerkiksi peli-, simulointi- tai visualisointiohjelmistoja.

Sekä korkeakoulujen tietokonegrafiikan kursseilla että aloittelijatason kirjallisuudessa esitetään edelleen usein vanhanaikainen tapa ohjelmoida reaaliaikaista grafiikkaa. Näytönohjainten uusia ominaisuuksia ei vanhanaikaista tapaa käyttämällä ole mahdollista hyödyntää. Vanhanaikaisen tavan käyttäminen kursseilla on perusteltavissa sillä, että sitä käyttäen voidaan nopeasti aikaansaada yksinkertaisia tuloksia. Lähemmin tarkasteltaessa kuitenkin ilmenee, että monet opettajista eivät ole edes tietoisia alan kehityksestä ja varjostinohjelmoinnin mahdollisuuksista. (Angel & Shreiner, 2011; McKesson, 2011). Tämä tutkielma tarjoaa lähtökohdan aihealueen opetteluun ja tietojen päivittämiseen.

Tutkielman toisessa luvussa syvennytään käsittelemään tietokonegrafiikan ohjelmoinnin periaatteita sekä reaaliaikaisen grafiikan erityispiirteitä. Luvun tarkoituksena on antaa lukijalle ymmärrys tietokonegrafiikan ohjelmoin-

nin erityispiirteistä. Tämä antaa valmiuksia ymmärtää varjostinohjelmiin perustuvan kehityksen syitä.

Kolmannessa luvussa käsitellään varjostinohjelmiin perustuvan grafiikan ohjelmointimenetelmään liittyviä muutoksia näytönohjainten arkkitehtuurin ja ohjelmoinnin näkökulmasta sekä esitetään arvio muutosten merkityksestä. Neljännessä luvussa käsitellään varjostinohjelmien ohjelmointia. Varjostinohjelmien ohjelmoinnin käsittely toimii tiivistelmänä niistä lisäyksistä, jotka ohjelmoijan on tunnettava muutoksen myötä. Tutkielman lopuksi vertaillaan uuden ja vanhan menetelmän keskeisiä eroavaisuuksia, pohditaan nykytilanteen merkitystä ja esitetään arvio alan tulevasta kehityksestä.

2 REAALIAIKAINEN TIETOKONEGRAFIIKKA

Tässä luvussa luodaan katsaus reaaliaikaisen tietokonegrafiikan periaatteisiin. Luvussa tutustutaan kahteen erilaiseen piirtomenetelmään, joita käytetään runsaasti reaaliaikaisen grafiikan piirrossa. Luku auttaa ymmärtämään tietokonegrafiikan ohjelmoinnin periaatteita. Luvun lopuksi syvennytään reaaliaikaisen grafiikan erityispiirteiden ja suorituskykyyn kohdistuvien vaatimusten käsittelyyn. Suorituskykyyn kohdistuvien vaatimusten ymmärtäminen auttaa hahmottamaan seuraavassa luvussa käsiteltävän grafiikan piirtoa koskevan uudistuksen merkitystä.

2.1 Tietokonegrafiikan periaatteet

Tietokonegrafiikan piirtäminen perustuu siihen, että matemaattisessa muodossa olevia esityksiä muunnetaan graafisiksi merkityssisältöisiksi näkymiksi. Kohteiden todenmukaisen tarkka jäljittely ei etenkään reaaliaikaisen grafiikan piirrossa ole tarkoituksenmukaista. Realististen menetelmien käyttö vaatisi niin paljon laskentaresursseja, ettei grafiikan esittäminen reaaliajassa olisi mahdollista. Tarkkojen ja realististen algoritmien kehittäminen on lisäksi työlästä ja ilmiöiden mallinnus täysin todellisuutta vastaavalla tarkkuudella mahdotonta. Niinpä käytetään menetelmiä, jotka tuottavat riittävän realistisia tuloksia.

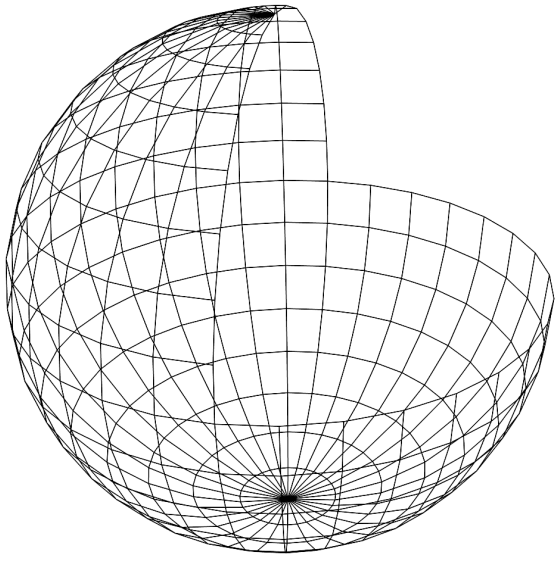
Tietokonegrafiikassa on siis kyse näkymiä ja ilmiöitä jäljittelevien illuusioiden piirrosta. Samanlaisen illuusion luontiin voi olla useita eri menetelmiä, mutta yhteistä kaikille menetelmille on matemaattisten algoritmien keskeinen rooli. Grafiikan piirto toteutetaan joukolla algoritmeja, joilla käsitellään muun muassa muotoja, valoa ja liikettä sekä jäljitellään ihmissilmän toimintaa. (Glassner, 1995; McKesson, 2011).

2.1.1 Polygoneihin perustuva piirto

Vallitseva reaaliaikaisen grafiikan piirrossa käytetty menetelmä perustuu kohteiden esittämiseen polygoneista rakennettuina verkkoina (Watt, 1999). *Polygoni* on kaksiulotteinen monikulmio. Polygoneista rakennettua kohdetta kutsutaan *polygoniverkoksi* ja kohteen taiteellista laatimisprosessia *mallintamiseksi*. Jos luodaan useista polygoneista koostuva litteä taso jota taivutellaan polygonien välisistä saumoista ja yhdistetään tiettyjen polygonien reunat toisiinsa, saadaan aikaiseksi kolmiulotteinen kohde. Sama toimii myös toisin päin: mikä tahansa polygoneista mallinnettu kolmiulotteinen kohde voidaan suoristaa kaksiulotteiselle tasolle. Polygoniverkkomallinnuksen periaatteissa voi nähdä yhtymäkohtia paperintaittelun kanssa.

Kohteiden mallinnus polygoniverkkoina on useisiin käyttökohteisiin sopiva menetelmä. Polygoneista koostuvien kohteiden piirto on suhteellisen nopeaa ja näyttävää, minkä vuoksi polygoniverkoista mallinnettuja kohteita käyttävällä piirtomenetelmällä on vallitseva asema reaaliaikaisissa grafiikkasovelluksissa. Nykyiset reaaliaikaisen grafiikan piirto- ja näytönohjainteknologiat on kehitetty polygoneista rakentuvaa grafiikkaa ajatellen, mikä osaltaan auttaa menetelmää säilyttämään asemansa.

Tyypillinen kolmiulotteinen näkymä koostuu useista erillisistä kolmiulotteisista kohteista. Jokaisen kohteen pinta muodostuu polygoneista. Polygoneista rakentuu kolmiulotteiselle kohteelle sen pinnan muotoinen verkko. Mikäli kohteesta piirretään näkyviin vain polygonien väliset saumat, saadaan kohteesta rautalankakehysmalli joka paljastaa kohteen rakenteen. (Lengyel, 2011). Kuviossa 1 esitetään pallonmuotoinen polygoniverkkona mallinnettu kappale rautalankakehysmallina. Kohteesta on leikattu pois yksi neljäsosa, jotta sen ontto rakenne olisi selvemmin havaittavissa.



KUVIO 1 Polygoniverkkona mallinnettu kohde, joka on piirretty rautalankakehysmallina

Polygoniverkkoina mallinnettaessa kohteille mallinnetaan vain pinta ja ne jäävät sisältä ontoiksi. Pelkkä pinnan mallintaminen on keskeinen syy siihen miksi polygoneihin perustuvan grafiikan piirto on moniin muihin menetelmiin nähden nopeaa. Tyypillisesti polygonit väritetään kuvioinnilla ja valoa jäljittelevillä tehosteilla, jolloin aikaansaadaan uskottavia illuusioita erilaisista esineistä. Kuviossa 2 esitetään kohde, jonka pinta on väritetty puukuvioinnilla. Lisäksi pinnan värityksessä on käytetty varjoja ja heijastusta tekemään esityksestä elävämpi ja uskottavampi.

KUVIO 2 Polygoniverkkona mallinnettu kohde, joka on kuvioitu ja valaistu

Graafista esittämistä varten pelkkä pinnan mallintaminen on tarkoituksenmukaista, sillä kohteiden sisäinen rakenne tai koostumus on harvoin havaittavissa (Hess, 2010). Tyypillisesti pinnasta tehdään umpinainen, jotta kohteen ontto sisäpuoli ei olisi havaittavissa. Tietokonepeleissä polygoniverkkoja voidaan piirron lisäksi käyttää määrittämään kohteiden fyysiset äärirajat, jottei esimerkiksi pelin maailmassa pelihahmon olisi mahdollista kulkea kohteiden reunojen läpi (Ericson, 2005). Mikäli pelimaailman kohteet eivät ole umpinaisia, voi se visuaalisten häiriöiden lisäksi aiheuttaa ongelmia pelin mekaniikassa. Kuvion 2 kohde ei havainnollisuuden vuoksi ole umpinainen, mikä tekee siitä poikkeuksellisen.

Kohteiden jääminen ontoiksi on polygoniverkkona mallinnettaessa menetelmästä johtuva ominaisuus, jolta ei ole mahdollista välttyä. Mikäli kohteen sisäinen rakenne halutaan tehdä havaittavaksi, joudutaan halutunlainen illuusio toteuttamaan erikoismenetelmin. Esimerkiksi läpinäkyvä lasipallo voidaan mallintaa samoin kuin puinenkin, mutta kohteen pinnoituksessa käytetään erilaista tilanteeseen soveltuvaa tehostetta. Tilanteesta riippuen tehokkaiden ja hyväksyttävän laadukkaiden lopputuloksen tuottavien tekniikoiden kehittäminen voi olla haasteellista. Mikäli ontolle kohteelle halutaan tehdä havaittava sisäinen rakenne, on ulkopinnan lisäksi myös sisus mallinnettava. Yksityiskohtaisemmasta

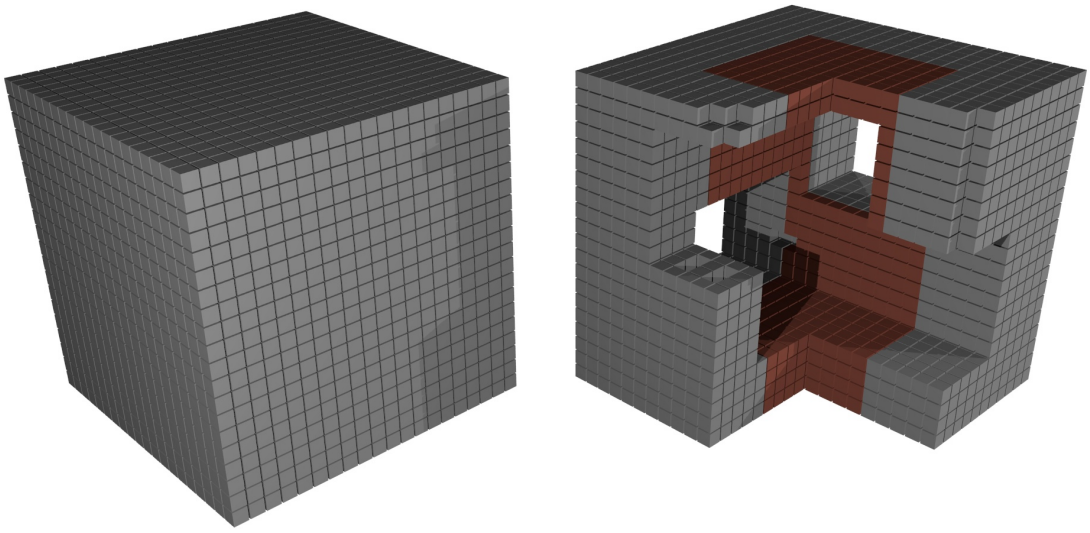
mallintamisesta huolimatta kohteen polygoniverkon sisäpuoli jää edelleen ontoksi, mutta sen havaitseminen ei ole yhtä ilmeistä. Kuviossa 3 on puolikkaalle pallonmuotoiselle kohteelle mallinnettu myös sisäpuoli. Kohde esittää halkaistun pallon sijaan halkaistua kuorta. Kohteen sisäpuoli on tässä tapauksessa kuoren sisä- ja ulkoreunojen välissä oleva alue, ja tässä kohteessa kyseinen kohteen sisään jäävä alue on ontto.

KUVIO 3 Kohde, jolle on mallinnettu yksinkertainen sisäpuolen rakenne

Vaikka polygoniverkkoihin perustuvalla piirtotavalla saavutetaan hyvä piirtonopeuden ja näyttävyyden välinen suhde, menetelmä ei ole ongelmaton. Menetelmään liittyviä keskeisiä haasteita ovat polygoniverkkoina mallintamisen vaikeus ja huono sopivuus sellaisiin käyttötarkoituksiin, joissa kohteen sisäisellä rakenteella on merkitystä. On olemassa myös muita reaaliaikaiseen piirtoon soveltuvia menetelmiä, joilla on erilaiset vahvuudet ja heikkoudet. Yksi lupaava vaihtoehtoinen menetelmä on vokseleihin perustuva piirto.

2.1.2 Vokseleihin perustuva piirto

Siinä missä polygoneilla voidaan mallintaa vain kohteen pinta, vokselella saadaan mallinnetuksi myös kohteen sisus. *Vokseli* voidaan käsittää pienenpieneksi kuutioksi. Vokselimallinnettu kohde voidaan käsittää kolmiulotteiseksi taulukoksi, jonka jokainen alkio on joko tyhjä tai sisältää vokselin. Kuviossa 4 vasemmanpuolisen kohteen jokaisessa alkiossa on samanlainen vokseli. Oikeanpuoleisen kohteen kaikki alkiot eivät sisällä vokselia. Lisäksi kohteen sisäosassa käytetään toisenvärisiä vokseleita.



KUVIO 4 Vokseleista mallinnettuja kohteita

Vokseleihin perustuva piirtomenetelmä on laajasti käytössä muun muassa lääketieteellisen visualisoinnin ja mallinnuksen piirissä. Vokseleihin perustuvan piirron laajemman soveltamisen tulevaisuudennäkymä on lupaava, sillä nykyisillä mikrotietokoneilla saavutettava laskuteho alkaa riittää mielekkään tarkkojen vokselimallinnettujen kohteiden esittämiseen reaaliaikaisesti. (Engel ym., 2004). Kohteiden mallintamista vokseleita käyttäen pidetään myös intuitiivisempana kuin polygonien käyttöä. Vokselimallinnuksen periaatteissa voidaan nähdä yhtymäkohtia kuutionmuotoisilla rakennuspalikoilla rakentelun kanssa.

2.2 Reaaliaikaisen grafiikan erityispiirteitä

Näyttävän reaaliaikaisen grafiikan esittämiseen liittyy teknisiä ja sisällöllisiä kompromisseja. Käytettävissä olevat piirtomenetelmät ja kohdelaitteiston suorituskyky asettavat absoluuttisia rajoituksia. Grafiikan näyttävyys riippuu piirtomenetelmien lisäksi kohteiden mallinnuksen tarkkuudesta.

Jotta reaaliaikaisen grafiikan esittäminen riittävän häiriöttömästi toteutuisi, on tapahtumien visuaalisen päivitystiheyden oltava riittävän suuri. Ruudunpäivitysnopeudella ilmaistaan kuinka monta erillistä kuvaa sekunnin aikana näyttölaitteella esitetään.

Graafisia kohteita on animoitava uskottavasti. Animointi perustuu siihen, että liikkeessä olevien kohteiden sijainti, muoto, asento, varjostus ja muut ominaisuudet muuttuvat jokaisessa yksittäisessä ruudussa. Kun muutokset ovat säännönmukaisia, syntyy liikkeen ja elävyyden kokemus. Liikkeellä on ajallinen kesto, eli animoidun grafiikan yksi merkittävä ulottuvuus on aika. (Williams, 2009).

Näyttävyyden on oltava korkeatasoista mutta piirron erittäin nopeaa. NTSC-televisiostandardista juontuva yleisesti käytetty tavoitearvo graafisten

sovellusten kuten pelien ruudunpäivitysnopeudelle on 60 piirrettyä ruutua sekunnissa. (Luebke & Humphreys, 2007; Gregory & Lander, 2009).

Mikäli sovelluksen ruudunpäivitysnopeuden tavoitearvoksi valitaan tavanomainen 60 ruutua sekunnissa, on yksittäisen ruudun piirtämiseen aikaa kuudeskymmenesosasekunnin eli noin 16 millisekunnin mittainen aika. Tässä ajassa on suoritettava sovelluksen tilan päivitys sekä yhden ruudun piirto. Lisäksi sovelluksessa on jatkuvasti käynnissä muita samanaikaisesti suoritettavia toimintoja ja käyttöjärjestelmän vuoronnus voi varata vaihtelevasti resursseja muiden käynnissä olevien ohjelmien tarpeisiin. (Gregory & Lander, 2009).

Jotta reaaliaikaisesta grafiikasta saataisiin piirrettyä näyttävää vähäisessä ajassa, toteutetaan piirtoalgoritmit sellaisella tarkkuudella joka riittää halutun illuusion luomiseksi mutta on todenmukaisen käyttäytymisen mallintamiseen riittämätön (Rost, 2009). Kohteiden pinnoituksessa käytetään tehosteita, joiden avulla saadaan yksinkertaisia grafiikkaresursseja käyttämällä toteutettua huomattavan yksityiskohtaiselta vaikuttavia esityksiä. Pinnoittamiseen tavallisesti käytettäviä tehosteita ovat pintakuviointi sekä valon heijastumisen jäljittely.

2.3 Grafiikan ohjelmointi

Tietokonelaitteiston grafiikkaominaisuudet toteutetaan näytönohjaimella, joka on grafiikan piirtoon erikoistettu komponentti. Grafiikan ohjelmoinnissa näytönohjainta käytetään grafiikkaohjelmointirajapinnan kautta. Rajapinta koostuu joukosta funktioita, joiden abstraktion taso on riittävän korkea laajan yhteensopivuuden saavuttamiseksi mutta samalla niin matala että tehokas ja joustava käyttö olisi mahdollista. (Woo, Neider & Davis, 1997; Rost ym. 2009.) Tässä tutkielmassa tarkastelun kohteena on OpenGL-grafiikkaohjelmointirajapinta.

OpenGL noudattaa asiakas-palvelin-arkkitehtuuria: OpenGL-toteutus toimii palvelimena asiakasohjelmalle. OpenGL-palvelimeen kohdistettavat funktiokutsut tunnistaa siitä, että niiden nimi alkaa kirjaimin *gl*, kuten esimerkiksi kutsussa *glEnable(GL_LIGHTING)*. OpenGL-palvelin toteuttaa asiakasohjelmalta saapuneet käskyt saapumisjärjestyksessä ja suorittaa grafiikan piirtoa niiden perusteella. OpenGL-palvelin on tilakone, jonka tilaa asiakasohjelma voi muuttaa. Eri tiloja käyttämällä voidaan säätää piirtoa ja siten muunnella samasta datasta piirrettyä tulosta. Esimerkiksi kutsumalla *glEnable(GL_LIGHTING)* asetetaan päälle valaistuksen käsittely.

Kulloinkin käytössä olevaa piirtotilaa kutsutaan piirtokontekstiksi (Woo, Neider & Davis, 1997; Rost ym. 2009; McKesson, 2011.). Kaikkia näkymän kohteita ei tarvitse piirtää käyttäen samaa kontekstia, vaan sitä voidaan vaihdella eri kohteiden piirron välillä. Piirrettyjen kohteiden yhdistäminen yhdeksi ruuduksi tehdään siinä vaiheessa kun kaikki näkymän kohteet on piirretty ja suoritetaan viimeistelytoimenpiteitä.

Yksinkertaisimmassa mahdollisessa grafiikkasovelluksessa on luotava ikkuna johon piirretään, alustettava piirtokonteksti, syötettävä piirtoon käytettävä data kuten polygonit, asetettava piirtoon käytettävät säätöarvot sopiviksi ja

annettava kohteiden piirtokomennot piirtojärjestyksessä. Piirtokomentojen välillä on tavallisesti muunneltava piirtokontekstia. Grafiikan ohjelmointia helpottamaan on kehitetty lukuisia apukirjastoja ja rakenteita, joita käyttämällä ohjelmoijan ei juurikaan tarvitse tehdä matalan tason ohjelmointia.

3 GRAFIIKAN PIIRRON MENETELMIEN MUUTOKSET

Entistä tehokkaampia tietoteknisiä ratkaisuja kehitettäessä pelkkä laskentatehon kasvattaminen ei ole mielekästä, vaan myös menetelmiä on uudistettava. Varjostinohjelmien käyttöönotto reaaliaikaisen grafiikan ohjelmoinnissa on viimeisin merkittävä grafiikan ohjelmoinnin uudistus. Uudistuksen taustalla on näytönohjainten arkkitehtuurin muutos, jossa on lisätty rinnakkain suoritettavan laskennan kapasiteettia. Tässä luvussa käsitellään keskeiset näytönohjainten arkkitehtuurin ja grafiikan ohjelmoinnin muutokset. Luvun lopuksi esitetään arvio uudistuksen merkityksestä sekä keskeisistä eduista ja haittapuolista.

3.1 Näytönohjaimen roolin muutos

Tärkeimmät näytönohjaimen sisältämät komponentit ovat grafiikkasuoritin, näyttömuisti, väyläliitäntä ja ulostuloliitäntä. Varhaiset näytönohjaimet käsittivät vain ulostuloliitännän ja näyttömuistin. Grafiikan piirto suoritettiin keskussuorittimella ja tulos syötettiin näytönohjaimen muistiin. Näyttömuistissa kulloinkin oleva data projisoitiin ulostuloliitännän kautta näyttölaitteelle.

Vaatimusten lisääntymisen myötä näytönohjaimiin on kehitetty apuominaisuuksia, joita käyttäen tiettyjä grafiikkaoperaatioita on mahdollista toteuttaa näytönohjaimeen lisätyllä grafiikkasuorittimella. Näytönohjaimen tarjoamia piirtoa tehostavia aputoimintoja kutsutaan grafiikkakiihdytykseksi. (Gregory, 2009). Grafiikkaohjelmointirajapintoihin kehitetään nopeasti näytönohjainten uusimpiakin ominaisuuksia hyödyntävät lisäykset. OpenGL-rajapinnasta on tällä hetkellä olemassa jo 17 virallista versiota. Ensimmäinen versio, 1.0, julkaistiin vuonna 1992 ja viimeisin versio, 4.3, vuonna 2012 (Khronos Group, 2012).

Varjostinohjelmien käyttöönotto on merkittävä laajennus näytönohjaimen tarjoamiin apuominaisuuksiin. Varjostinohjelmia käyttämällä voidaan määritellä halutunlaisiksi kaikki ne aputoiminnot, joiden suorittamisessa hyödynnetään näytönohjainta. Ennen varjostinohjelmien käyttömahdollisuutta erilaisia apu-

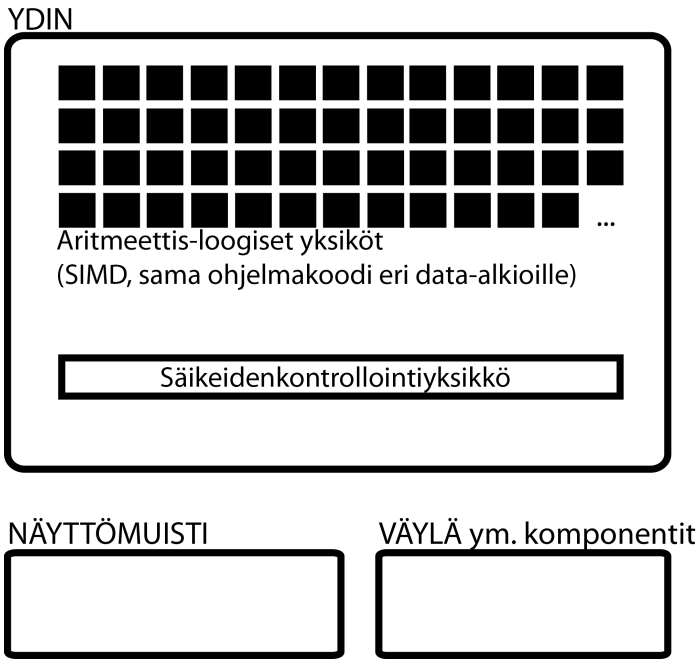
toimintoja oli käytettävissä vain tietty ennalta määrätty joukko, minkä vuoksi eräiden tehosteiden toteuttaminen vaati monimutkaista ohjelmointia tai saattoi olla kokonaan mahdotonta. Koska varjostinohjelmia käyttäen näytönohjaimen toimintaa voidaan räätälöidä täsmälleen käyttötarpeen mukaiseksi, voidaan niillä tuottaa parempitasoista grafiikkaa paremmalla piirtonopeudella. (Sherrod, 2008).

Vanhojen menetelmien toiminnan takaaminen myös uudessa ympäristössä on tietoteknisessä kehitystyössä merkittävä haaste. Vanhojen grafiikkaohjelmien toimivuuden takaamiseksi varjostinohjelmiin liittyvät uudistukset on kohdistettu vain tiettyihin selkeärajaisesti määriteltyihin piirron osiin. Aluksi uudistukset olivat saatavilla vaihtoehtoisena laajennuksena. Uudemmissa määritelmäversioissa menetelmien roolit ovat vaihtuneet siten, että vanha toteutustapa on saatavissa enää vaihtoehtoisena laajennuksena. (Rost ym., 2009). OpenGL-grafiikkaohjelmointirajapinnassa jako uuteen ja vanhaan toteutustapaan tehtiin määritelmäversion 3.2 myötä vuonna 2009. Määritelmäversiosta 3.2 alkaen on varjostinohjelmien käyttö pakollista. Versio 3.2 ja sitä uudemmat versiot luetaan kuuluvaksi *ydinprofiiliin*. Vanha toteutustapa on käytettävissä tätä vanhempien määritelmäversioiden kanssa, ja ne kuuluvat *yhteensopivuusprofiiliin*. (Khronos Group, 2012).

3.2 Näytönohjaimen arkkitehtuurin kehitys

Näytönohjaimen grafiikkasuorittimen käskyjoukko on suppea ja erikoistettu reaaliaikaisen grafiikan piirron edellyttämien toimintojen suorittamiseen. Grafiikkasuorittimiin on lisätty paljon rinnakkain toimivia aritmeettis-loogisia laskentayksiköitä, joille yksikön kontrollilogiikka jakaa suoritustaakkaa tasaisesti. Piirto-operaatioiden jako useisiin säikeisiin tapahtuu automaattisesti, joten ohjelmoijan ei tarvitse huolehtia säikeistyksen toteuttamisesta. Yksisäikeisistä operaatioista on siis grafiikkasuorittimissa siirrytty monisäikeisiin hajautettuihin operaatioihin. (Fatahalian & Houston, 2008).

Rinnakkaisen suorituksen lisääminen yksittäisen säikeen suoritustehon kasvattamisen sijasta on tämänhetkinen suoritinten kehityksen suuntaus (Owens ym., 2008). Siinä missä tietokoneiden keskussuorittimissa käytetään tällä hetkellä tyypillisesti kahdesta kuuteen ydintä, on näytönohjainten suorittimien ydinten määrä kasvatettu useisiin satoihin tai jopa tuhansiin (kuvio 5).



KUVIO 5 Näytönohjaimen laitteistotason arkkitehtuuri

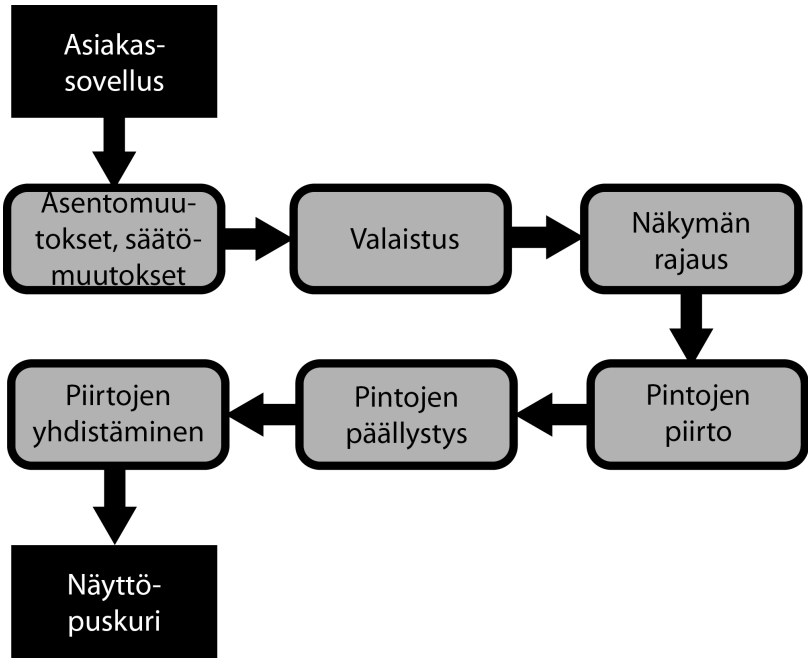
Rinnakkainen laskenta sopii näytönohjaimille hyvin, sillä yksinkertaisista perusrakenteista koostuvan grafiikan piirtäminen on luontevasti ositettavissa asynkronisiin rinnakkaisiin operaatioihin. Laskenta noudattaa single instruction, multiple data -periaatetta, eli jokaiseen laskennan alaisen kohteen alkioon sovelletaan sama operaatio. Käytettävien ruudunpäivitysnopeuksien kohtalaisen matalan tason johdosta rinnakkainen asynkroninen piirto ei ole erityisen viiveherkkää. (Fatahalian & Houston, 2008).

Vaikka piirto-operaatioiden jakaminen rinnakkain suoritettaviin säikeisiin tarjoaa tuntuvan edun piirtonopeudessa, aiheuttaa se myös ongelman säikeiden välisen tiedonsiirron kanssa. Joidenkin tehosteiden toteutus edellyttää, että säikeille annetaan käyttöön muissa säikeissä laskettuja tuloksia. Tietyt tehosteet siis vaativat tiedonsiirtoa säikeiden välillä, mutta kesken piirto-operaation tiedonsiirto on mahdotonta. Tällöin piirto on jaettava useaan peräkkäin suoritettavaan erilliseen operaatioon, jotta säikeiden tuottama data saadaan tallennettua välitulokseksi ja synkronoitua uuden operaation lähtöarvojoukoksi. Tällainen menettely häivyttää osan rinnakkaisella laskennalla saavutettavasta piirtonopeusedusta.

3.3 Grafiikan piirtoketju

Grafiikan piirtoketjuksi kutsutaan sitä näytönohjaimella suoritettavaa toimintojen sarjaa, jonka lopputuloksena saadaan valmis piirretty ruutu. Piirtoketju toimii työvaihe kerrallaan siten, että jokainen työvaihe saa alkuarvot, suorittaa tie-

tyt toimenpiteet ja lähettää tuloksen eteenpäin. Työvaiheiden keskinäinen järjestys ja rajapinta on määrätynlainen. Piirtoketjun lähtöarvojoukkona on piirtoon käytettävä data ja tuloksena näyttömuistiin piirretty kuva, joka on valmis näyttölaitteella esitettäväksi (kuvio 6).



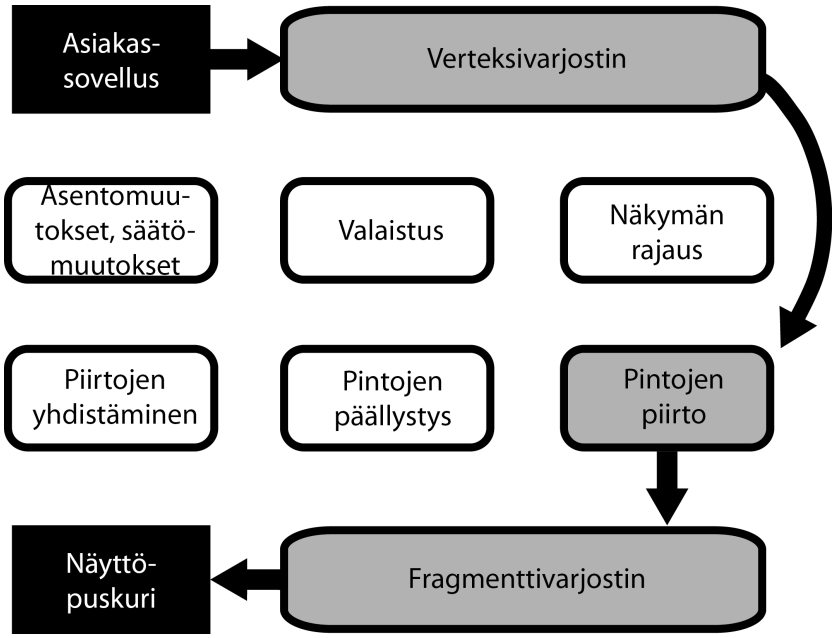
KUVIO 6 Suoraviivaistus OpenGL:n säädettävästä piirtoketjusta

Vanhassa piirtoketjussa työvaiheet ovat toiminnaltaan joko täysin lukittuja tai tietyin vivuin säädettävissä. Säätöjen avulla piirtokontekstia voidaan vaihdella kesken piirron. Vanhaa piirtoketjua kutsutaan säätömahdollisuuden vuoksi *säädettäväksi piirtoketjuksi*. Esimerkiksi kutsulla *glViewport(...)* voidaan säätää näkymänrajausvaiheen toimintaa. (Rost ym. 2009; Marroquim & Maximo, 2009; Pozzer, 2009.)

3.4 Uudistettu grafiikan piirtoketju

Kun varjostinohjelmat otettiin osaksi piirtoketjua, jäi osa vanhan piirtoketjun työvaiheista tarpeettomiksi. Uudessa piirtoketjussa ohjelmoijan odotetaan itse toteuttavan varjostinohjelmia käyttäen ne tarvittavat operaatiot, jotka aikaisemmin suoritettiin poistetuissa työvaiheissa (kuvio 7). Ohjelmoitavien työvaiheiden piirtokontekstia muunnellaan käytettävää varjostinohjelmaa tai varjostinohjelmien kombinaatiota vaihtelemalla. Uudistettua piirtoketjua kutsutaan myös *ohjelmoitavaksi piirtoketjuksi*. Vanhaa säädettävän piirtoketjun ja uuden ohjelmoitavan piirtoketjun käsitettä ei tule sekoittaa keskenään, sillä ne ovat toisilleen vastakkaiset (Rost ym., 2009; Pozzer, 2009). Muutoksen myötä grafiikan

ohjelmointi on muuttunut työläämmäksi, mutta tarjoaa enemmän vapauksia. (Rost ym. 2009; Marroquim & Maximo, 2009; Pozzer, 2009.)



KUVIO 7 Suoraviivaistus OpenGL:n ohjelmoitavasta piirtoketjusta

Verteksi- ja fragmenttioperaatiot ovat ensimmäiset ohjelmoitaviksi avatut piirtoketjun vaiheet. Kumpaakin vaihetta varten on laadittava tietynlaiset varjostinohjelmat. Verteksioperaatiot suoritetaan verteksivarjostinohjelmilla ja fragmenttioperaatiot fragmenttivarjostinohjelmilla. Myöhemmin ohjelmoitavaksi avattiin myös geometriavarjostinohjelmilla suoritettavat geometriaoperaatiot. Niiden suorituksen voi kuviossa 7 yksinkertaistetusti lukea sijoittuvan verteksivarjostimen loppuun.

3.5 Muutosten arviointi

Mitä tahansa tietoteknistä kehityssuuntausta tarkasteltaessa on huomioitava, että vallitsevaksi nousseelle suuntaukselle on ollut tarjolla joukko vaihtoehtoja. Jos grafiikkaohjelmoinnissa olisi pitäydytty säädettävän piirtoketjun käytössä, olisi sitä todennäköisesti kehitetty monipuolisemmaksi ja näin vastattu vaatimuksiin entistä laajemman tehostejoukon toteutusmahdollisuuksista. Ohjelmoitavan piirtoketjun käyttöönottoon vaikutti joustavamman menetelmän tarpeen lisäksi myös muut tekijät, kuten näytönohjainten laitearkkitehtuurissa tapahtunut rinnakkaisen laskentakapasiteetin lisäys.

Ohjelmoitavan piirtoketjun tarjoamien mahdollisuuksien monipuolisuudesta on seurannut etuja, joita ei menetelmää suunniteltaessa osattu välttämättä ottaa huomioon. Säädettävä piirtoketju ja sitä käyttäen hyödynnettävissä olevat

grafiikkakiihdytyksen ominaisuudet kehitettiin polygoneihin perustuvan grafiikan piirtoa varten. Polygoneihin perustuvan grafiikan piirron tehostaminen oli lähtökohtana myös ohjelmoitavaa piirtoketjua kehitettäessä. Ohjelmoitavaa piirtoketjua voidaan kuitenkin muokata tarjoamaan grafiikkakiihdytystä yhtä lailla myös vaihtoehtoisia piirtotapoja, kuten vokseleihin perustuvaa piirtoa, varten. (Engel ym., 2004).

Koska näytönohjainten grafiikkasuorittimien teoreettinen laskentateho on huomattavan suuri tavanomaisiin keskussuorittimiin nähden, ryhdyttiin näytönohjainten laskentakapasiteettia nopeasti hyödyntämään myös muuhun kuin grafiikan piirtoon. Näytönohjaimia hyödyntävien yleislaskentasovellusten kehittämisestä on tullut kokonaan uusi ohjelmoinnin osa-alue, ja tarkoitusta varten on kehitetty uusia ohjelmointirajapintoja. (Nickolls, 2010). Yksi tällainen keskussuoritinta ja grafiikkasuoritinta samanaikaisesti hyödyntävä ympäristö on Khronos Groupin kehittämä Open Computing Language (OpenCL).

Käytön yksinkertaisuus säilyy edelleen merkittävänä säädettävän piirtoketjun etuna verrattuna ohjelmoitavaan piirtoketjuun. OpenGL:n säädettävää piirtoketjua käyttävissä vanhoissa määritelmäversioissa on sisäänrakennettuna useita ominaisuuksia, jotka ohjelmoijan tulee uusia määritelmäversioita käyttäessään toteuttaa itse. Grafiikan ohjelmoinnista on tullut entistä monimutkaisempaa, mikä tekee siitä erityisesti aloittelijalle entistä vaikeammin omaksuttavaa. Toisaalta ohjelmoitavaa piirtoketjua käytettäessä on mahdollista laatia tietynlaiset varjostinohjelmat sekä niiden kanssa käytettävä apukirjasto, joita käyttämällä voidaan säilyttää säädettävästä piirtoketjusta tuttu grafiikkarajapinnan käyttötapa (Rost ym., 2009).

Varjostinohjelmien hyödyntäminen nykyisellä tavalla ei suinkaan ole grafiikkaohjelmoinnin lopulliseksi jäävä muoto, vaan ala on jatkuvan kehityksen kohteena. On ennakoitavissa, että piirtoketjusta tullaan jatkossa avaamaan yhä uusia operaatioita ohjelmoitavuuden piiriin siten, että ohjelmoija voi itse määrittää piirtoketjun toiminnan vastaamaan tarpeitaan entistä tarkemmin. Tulevan kehityksen ennakointi vaatii perusteellista perehtymistä alan uusimpiin julkaisuihin.

4 VARJOSTINOHJELMIEN OHJELMOINTI

Reaaliaikaisen grafiikan ohjelmointi on uudistunut siten, että grafiikan piirrossa käytetään näytönohjaimen tarjoamaa grafiikkakiihdytystä. Grafiikkakiihdytystä käyttävät operaatiot ohjelmoidaan erityisillä varjostinohjelmilla, jotka suoritetaan näytönohjaimella. Vaikka varjostinohjelmia käyttämällä onkin mahdollista toteuttaa laajalti erilaisia tehosteita, on varjostinohjelmia käyttävässä ohjelmassa huolehdittava oikeellisesta varjostinten alustuksesta ja piirtoon käytettävän datan syöttämisestä. Varjostinohjelmia käyttävää ohjelmaa kutsutaan *isäntäohjelmaksi*. Mikäli grafiikan ohjelmoinnissa on tarkoitus käyttää ajanmukaisia menetelmiä, on varjostinohjelmien käyttö on pakollista. Tässä luvussa esitellään se lisätietämys, joka tarvitaan vanhaan ohjelmointitapaan nähden.

4.1 Varjostinohjelmien ominaispiirteet

Kaikki varjostinohjelmat omaavat tiettyjä yhteisiä piirteitä, kuten sisään- ja ulostulodataan liitettävien muuttujien esittely, erikoistettujen perustyyppien kuten matriisien ja vektorien käyttö, ohjelmien matemaattinen luonne sekä suhteellisen lyhyt ohjelmakoodi.

OpenGL-varjostinohjelmassa on ohjelmoijalla käytettävissään tiettyjä implisiittisesti määriteltyjä globaaleja muuttujia, joita tarvitaan toteuttamaan rajapinta työvaiheiden välillä. Lisäksi ohjelmoijan on eksplisiittisesti esiteltävä varjostinyksikön muuhun piirtoketjuun yhdistävä sisään- ja ulostulodata käyttämällä in- ja out-määreiden kanssa esiteltyjä muuttujia. Sisään- ja ulostulodataksi kelpaa vain tietynlainen piirtoketjun kannalta oleellinen data. Uniform-määreellä esiteltyjä muuttujia käytetään tuomaan varjostinta käyttävän isäntäohjelman asettamia arvoja varjostinohjelman käyttöön. Uniform-muuttujien avulla välitettävä data on vapaasti ohjelmoijan päätettävissä, kunhan se on tyypiltään GLSL-määrityksen mukaista. (Rost ym., 2009).

GLSL-varjostinohjelmia saa olla yhdessä lähdekooditiedostossa yksi kappale. Varjostinohjelman aloituspisteeksi on määrätty parametriton main-funktio, jonka paluutyyppi on void.

4.2 OpenGL Shading Language -varjostinkieli

GLSL (OpenGL Shading Language) on proseduraalinen korkean tason ohjelmointikieli. Se on kehitetty käytettäväksi varjostinohjelmien tuottamiseen OpenGL-grafiikkaohjelmoinnin yhteydessä. GLSL:stä on tullut merkittävä osa OpenGL-määritystä. GLSL muistuttaa C- tai C++-kieltä ollen niihin nähden kuitenkin huomattavan suoraviivaistettu varjostinohjemien ohjelmointiin (Rost ym, 2009).

Silmukat, ehtolauseet ja funktiokutsut ovat käytännössä identtisiä C:n kanssa. Kielen tietotyyppien joukkoon kuuluu grafiikkalaskennassa oleellisia tietotyyppejä, kuten *vektorit* monesta komponentista koostuvan datan käsittelyyn, *matriisit* moniulotteisen ja monesta komponentista koostuvan datan käsittelyyn sekä *samplerit* kuvadatan lukemiseen muistista. Vektoreita käytetään tavallisesti esimerkiksi väriarvojen käsittelyyn, matriiseja kohteiden asentomuunnosten suorittamiseen ja samplereita kuvadatan välitykseen isäntäohjelmalta varjostinohjelmille. Boolean-tyyppi on tuettu samalla tavoin kuin C++:ssa. Globaaleille muuttujille voidaan antaa joko in- out- tai uniform-määre, jolla määrätään globaalille muuttujalle rooli tiedonvälityksessä piirtoketjun muiden osien kanssa. Kieleen on myös sisäänrakennettu suuri joukko grafiikkalaskennassa oleellisia matemaattisia funktioita, kuten esimerkiksi trigonometriset funktiot, geometriset funktiot sekä vektori- ja matriisifunktiot. (Kessenich, 2008; Rost ym., 2009).

4.3 Yhtenäistetty varjostinmalli

OpenGL-määritys edellyttää, että näytönohjaimen piirtoketjussa käytetään varjostinohjelmia kahteen eri työvaiheeseen, verteksioperaatioiden ja fragmenttioperaatioiden suorittamiseen (Rost ym., 2009). Eri työvaiheiden varjostinohjelmat poikkeavat toisistaan.

Ensimmäisen sukupolven varjostintekniikkaa käyttävissä näytönohjaimissa eri työvaiheiden varjostinohjelmien ajamiseen käytettiin työvaihekohtaisesti eriytettyjä laskentayksiköitä. Uusien yhtenäistetyn varjostinmallin mukaisesti toteutettujen näytönohjainten laskentayksiköt eivät ole eriytettyjä, vaan niillä kaikilla voidaan suorittaa kaikkien eri työvaiheiden varjostinohjelmia.

Yhtenäistetyn varjostinmallin käyttö tehostaa näytönohjaimen toimintaa, sillä kaikki laskentayksiköt osallistuvat laskentaan vaikka eri työvaiheiden varjostinohjelmien keskinäinen laskentatarve olisi epäsuhtainen. Ennen yhtenäistettyä varjostinmallia eriytettyjen laskentayksiköiden kapasiteetin keskinäisen

suhteen päättäminen näytönohjaimen rakenteelliseen toteutukseen oli ongelmallista (Luebke & Humphreys, 2007).

4.4 Erityyppiset varjostinohjelmat

Varjostinohjelmia on olemassa kolmea eri tyyppiä: verteksti- fragmentti-, ja geometriavarjostinohjelmia. Kukin erityyppisistä varjostinohjelmista kohdistetaan piirtoketjun eri vaiheessa sijaitsevaan varjostinyksikköön, ja niillä on oma tehtävänsä. OpenGL:n määritelmäversiossa 2.0 vuonna 2004 käyttöön otettiin verteksi- ja fragmenttivarjostinohjelmat. Määritelmäversiossa 3.2 vuonna 2008 käyttöön otettiin myös geometriavarjostinohjelmat (Khronos Group, 2012). Näistä verteksi- ja fragmenttivarjostimien määrittäminen on pakollista ja geometriavarjostimien määrittäminen vaihtoehtoista.

4.4.1 Verteksivarjostinohjelmat

Verteksivarjostin on ohjelmoitavissa oleva piirtoketjun yksikkö, joka sijoittuu ketjun alkuvaiheeseen. Verteksivarjostimen toiminnallisuus määritetään verteksivarjostinohjelmia käyttäen. Verteksivarjostimen nimi tulee siitä, että sillä suoritettavat operaatiot kohdistuvat kohteiden vertekseihin eli geometriapisteisiin, joista polygonit ja polygoniverkot muodostuvat. Grafiikan jokaiselle verteksille suoritetaan yksitellen varjostinohjelmassa määritetyt toimenpiteet näytönohjaimen rinnakkaista laskentaa hyödyntäen. Tyypillisiä verteksivarjostimella suoritettavia toimintoja ovat verteksien sijaintimuunnosten, valaistusarvojen, valon heijastusarvojen ja pinnoituskoordinaattien käsittely. (Sherrod, 2008; Rost ym., 2009).

Verteksivarjostinohjelman sisääntuloarvot välittyvät piirtoketjua edeltävistä työvaiheista. Verteksivarjostinohjelma tuottaa ulostuloarvoina sekä implisiittisiä piirtoketjun sisäisiä muuttujia seuraavien työvaiheiden käyttöön että käyttäjän eksplisiittisesti määrittämiä muuttujia ketjussa myöhemmin käytettävien varjostinyksiköiden käyttöön. (Rost ym., 2009).

4.4.2 Fragmenttivarjostinohjelmat

Fragmenttivarjostin on toinen OpenGL:n piirtoketjun ohjelmoitava yksikkö, ja se sijoittuu ketjun loppuvaiheeseen. Fragmenttivarjostimen toiminnallisuus määritetään fragmenttivarjostinohjelmin. Fragmenttivarjostin suorittaa määritellyt toimenpiteet piirrettävän ruudun jokaiseen yksittäiseen kuvapisteeseen. Tyypillisiä fragmenttivarjostimella suoritettavia toimenpiteitä ovat polygonien pinnoitusten maalaus,sumu- ja väriarvojen laskeminen sekä erilaiset kuvatehosteet. (Rost ym., 2009).

Fragmenttivarjostinohjelman sisääntuloarvojen tulee täsmätä verteksivarjostinohjelmassa käytettyjen ulostuloarvojen kanssa. Mikäli piirtoketjussa käytetään lisäksi geometriavarjostinohjelmaa, tulee sisääntuloarvojen täsmätä myös sen ulostuloarvojen kanssa. Fragmenttivarjostimen ulostuloarvoja käytetään ketjun seuraavissa työvaiheissa, joissa piirrettävä ruutu viimeistellään (Rost ym., 2009).

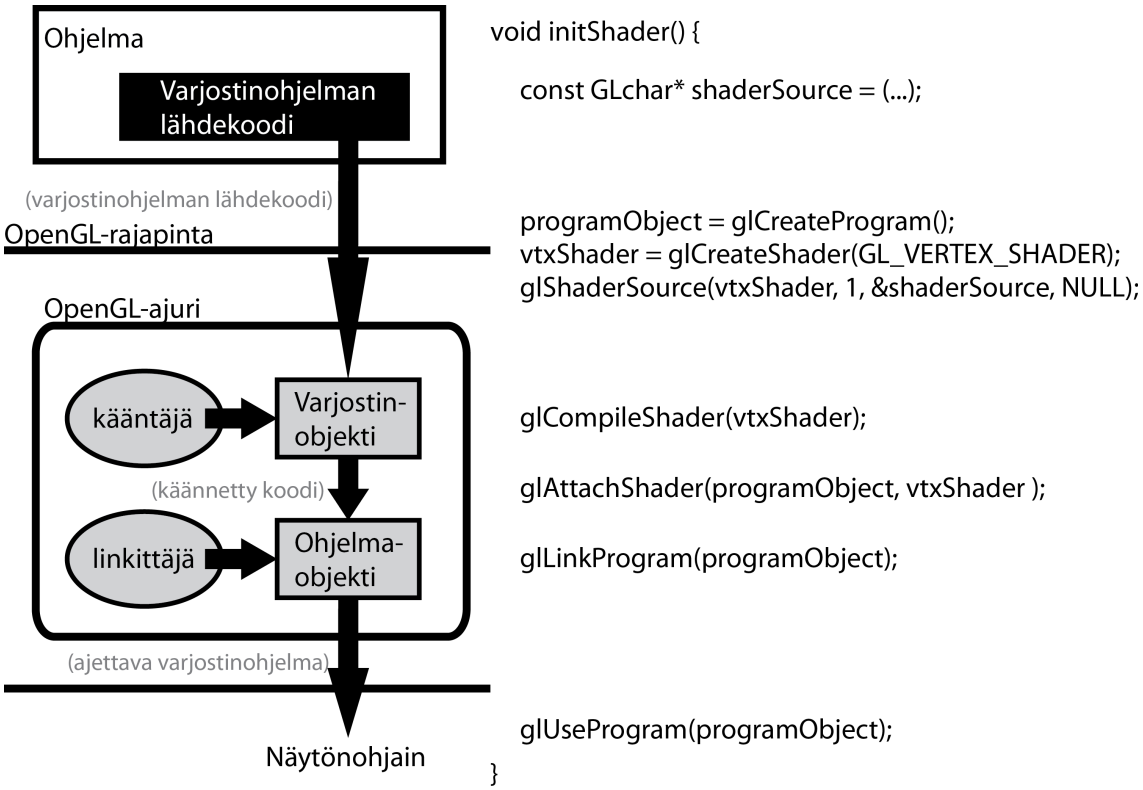
4.4.3 Geometriavarjostinohjelmat

Geometriavarjostin on kolmas OpenGL:n piirtoketjun ohjelmoitavissa olevista yksiköistä, ja se sijoittuu verteksivarjostimen perään piirtoketjussa. Geometriavarjostimen toiminta määritetään geometriavarjostinohjelmia käyttäen. Geometriavarjostinta voidaan käyttää luomaan kokonaan uutta geometriaa, eli tyypillisesti lisäämään näkymään uusia polygoneja. (Sherrod, 2008). Verteksi- ja geometriavarjostinten välistä eroa voi aluksi olla vaikea hahmottaa. Verteksivarjostimella voidaan muokata ennalta määritettyjä kohteisiin mallinnettuja polygoneja, kun taas geometriavarjostimella voidaan luoda kokonaan uusia polygoneja.

Koska geometriavarjostinohjelma suoritetaan piirtoketjussa verteksi- ja fragmenttiohjelmien välissä, tulee sen sisääntuloarvojen täsmätä verteksivarjostinohjelman ulostuloarvojen kanssa ja ulostuloarvojen tulee täsmätä geometriavarjostinohjelman sisääntuloarvojen kanssa (Sherrod, 2008).

4.5 Varjostinohjelman kääntäminen ja linkittäminen

Varjostinohjelmat tallennetaan isäntäohjelman datan joukkoon selkokielisessä tekstimuodossa. Varjostinohjelmat käännetään konekieliseen muotoon vasta käyttöönoton yhteydessä, tyypillisesti osana isäntäohjelman käynnistysrutiinia. OpenGL-ajuri, joka on osa näytönohjaimen ajuriohjelmistoa, suorittaa käännösoperaation. Varjostinohjelman käyttöönotto muodostuu useasta vaiheesta ja näitä vaiheita vastaavista funktiokutsuista kuvion 8 esittämällä tavalla. Käännetystä varjostinohjelmasta käytetään nimitystä *varjostinobjekti*. Kun varjostinobjektit yhdistetään yhdeksi tehosteeksi, käytetään kokonaisuudesta nimitystä *ohjelmaobjekti*.



KUVIO 8 Varjostinohjelman saattaminen lähdekoodista käyttöön

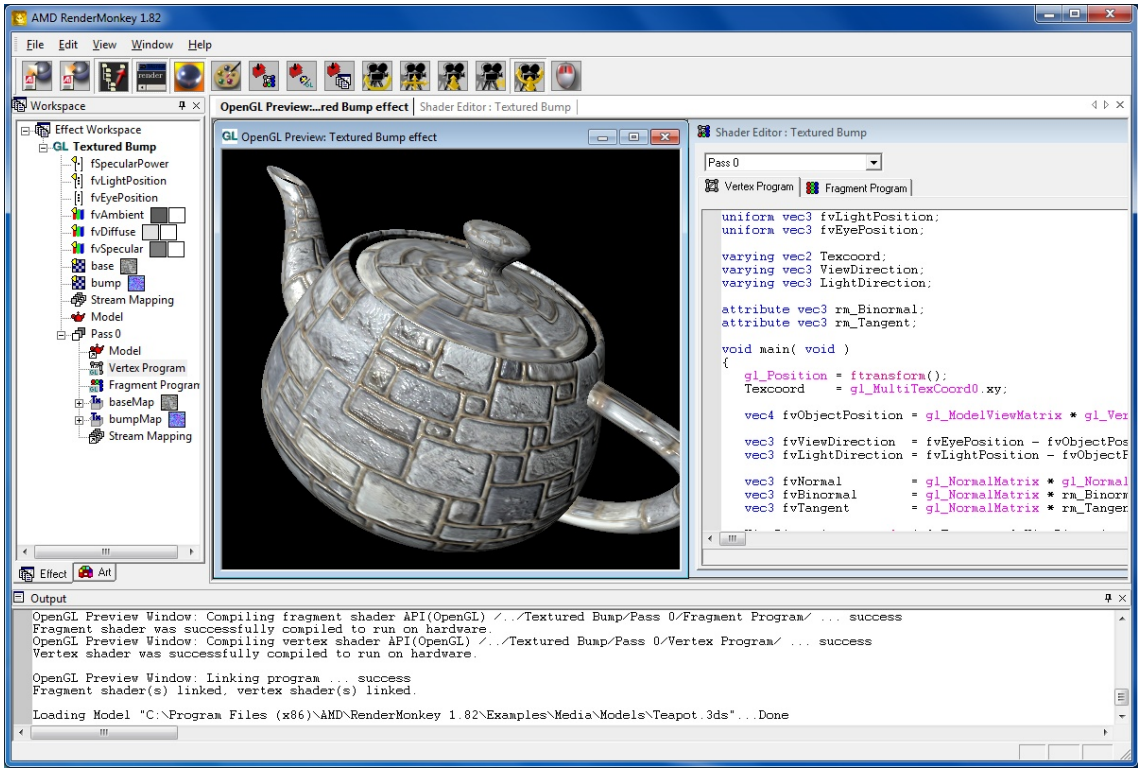
Käyttöönotto alkaa sillä, että OpenGL-ajuriin varataan varjostinobjekti, joka tulee sisältämään varjostinohjelman. Tämän jälkeen varjostinohjelman lähdekoodi toimitetaan varjostinobjektiin ja käännetään voimakkaasti optimoiduksi konekieleksi. Käännetty varjostinohjelma voidaan linkittää ohjelmaobjektiin ja ottaa käyttöön toteuttamaan osa piirtoketjua. Varjostinohjelmien kääntäminen vasta käyttöönoton yhteydessä mahdollistaa varjostinohjelmien mahdollisimman hyvän yhteensopivuuden. Tämän haittapuolena on se, että ihmisen luettavissa olevassa muodossa säilötyt varjostinohjelmien lähdekoodit on helppo siepata ohjelmistosta. (Rost ym., 2009).

4.6 Varjostinohjelmien kehittäminen

Kuten muussakin ohjelmoinnissa, yksinkertainen tekstieditori riittää varjostinohjelmien kehittämiseen. Useimmissa ohjelmointikielissä ohjelmat käännetään kääntäjäohjelmalla väli- tai konekieleksi kohdealustalla suorittamista varten. Kääntäjä antaa arvokasta palautetta ohjelman toiminnasta ja mahdollisista ongelmakohdista. Tavallisesti kääntäjää käytetään jatkuvasti ohjelmointityön apuvälineenä.

Varjostinohjelmat käännetään OpenGL-ajuriin sisältyvällä kääntäjällä vasta käyttöönoton yhteydessä isäntäohjelman toimesta. Isäntäohjelma voi kerätä kääntäjältä varjostinohjelman toimintaan liittyvää arvokasta tietoa. Jos käytettävissä ei ole kehittämisen avuksi soveltuvaa isäntäohjelmaa, voidaan käyttää

varjostinohjelmien ohjelmointiympäristöjä. Varjostinohjelmien ohjelmointiympäristö toimii kehitettäville varjostinohjelmille isäntäohjelmana. Se sisältää tavallisesti tekstieditorin kattavalla syntaksikorostuksella, esikatseluikkunan varjostintehosteiden tarkasteluun reaaliajassa, tekstikonsolinäkymän kääntäjän ilmoitusten selaamiseen, projektienselausnäkymän sekä näkymän varjostinohjelmille syötettävän datan ja parametrien määritystä varten. Kattavimmin varustellut ympäristöt sisältävät lisäksi debuggerin. Kuviossa 9 on esitetty ruutukaappaus AMD RenderMonkey -kehitysympäristöstä. Muita OpenGL-varjostimien kehittämiseen soveltuvia ympäristöjä ovat muun muassa Typhoon Labs Shader Designer ja glslDevil. (Baert, 2010).



KUVIO 9 AMD RenderMonkey -ympäristö varjostinohjelmien kehittämiseen

OpenGL:ää tukevia kehitysympäristöjä ei ole kovin useita. Usein parhaaksi vaihtoehdoksi esitetyn AMD RenderMonkey -ympäristönkin kehitys on lopetettu (Advanced Micro Devices, Inc., 2012). Microsoftin HLSL- ja NVIDIAn Cg-varjostinkieliä varten on olemassa monipuolisempi valikoima kehitystyökaluja. Yhdeksi syyksi kehitysympäristöjen vähäisyydelle on esitetty sitä, että vaativissa käyttökohteissa ne eivät onnistu jäljittelemään riittävän hyvin isäntäohjelmien toimintaa (Petrie, 2011).

5 YHTEENVETO

Tutkielmassa on käyty läpi varjostinohjelmiin perustuvan reaaliaikaisen tietokonegrafiikan ohjelmoinnin periaatteet. Tutkielmassa on keskitytty niihin muutoksiin, joita reaaliaikaisen grafiikan piirron menetelmiin on varjostinohjelmiin perustuvan ohjelmointitavan käyttöönoton myötä tehty. Tässä luvussa vertaillaan vanhaa ja uutta menetelmää sekä esitetään arvio reaaliaikaisen grafiikan tulevasta kehityssuuntauksesta.

Ohjelmoitava piirtoketju on korvannut säädettävän piirtoketjun reaaliaikaisen tietokonegrafiikan ohjelmoinnissa. Muutos on ollut välttämätön, jotta näytönohjaimiin kehitettyjen ominaisuuksien täysimääräinen hyödyntäminen olisi mahdollista. (Rost ym., 2009.)

Merkittävimmät ohjelmoitavan piirtoketjun edut saavutetaan grafiikan näyttävyydessä ja piirron nopeudessa. Ohjelmoitavuus mahdollistaa myös piirtoketjun toiminnan joustavan muuntelun, jolloin voidaan toteuttaa kokonaan uudenlaisia tehosteita. Ohjelmoitavan piirtoketjun käyttäminen kuitenkin tekee grafiikan ohjelmoinnista monimutkaisempaa sekä vaatii ohjelmoijalta valmiutta muokata ja luoda varjostinohjelmia. (Rost ym., 2009.)

Nykyään suositellaan käytettäväksi ohjelmoitavaa piirtoketjua aina kun se on mahdollista. Jotta vanhat grafiikkasovellukset toimisivat edelleen, on vanha menetelmä jätetty vaihtoehtoiseksi toteutustavaksi grafiikkarajapintojen uusiin versioihin. (Rost ym., 2009.)

Suorittimien ydinten määrän kasvattaminen on tällä hetkellä tietotekniseessä kehityksessä voimakas suuntaus. Näytönohjainten grafiikkasuorittimissa ydinten määrä on kasvatettu jo huomattavan suureksi (Owens & Davis, 2007). On ennakoitavissa, että myös keskussuorittimien ydinten määrää kasvatetaan seuraavien kymmenen vuoden aikana. Tämän myötä monisäikeisen ohjelmoinnin painoarvo tulee kasvamaan, mikä voi synnyttää täysin uusia ohjelmointityylejä ja -kieliä.

Eräs jo olemassa oleva ympäristö monisäikeiseen ohjelmointiin on Khronos Groupin kehittämä Open Computing Language (OpenCL). OpenCL:n lähtökohtana on, että sen avulla voidaan ohjelmoida tehokkaita monisäieohjelmia jotka hyödyntävät heterogeenisia laskentaresursseja, kuten sekä keskussuoritin-

ta että näytönohjainta yhdessä, yleistarkoituksellisiin tarpeisiin. (Rosenberg, 2011.)

Grafiikkaohjelmointi kehittyy seuraavien kymmenen vuoden aikana todennäköisesti siten, että piirtoketjusta avataan lisää työvaiheita ohjelmoitavaksi (Owens & Davis, 2007). Grafiikan ohjelmoinnin menetelmät tulevat todennäköisesti muotoutumaan jossain määrin uudelleen. Näytönohjainten suorittimien laskentatehon kasvattaminen johtaa todennäköisesti siihen, että näytönohjaimilla voidaan tuottaa fotorealistiselta vaikuttavaa grafiikkaa reaaliaikaisesti. Grafiikkaohjelmien ja sisältöjen kehitystyö tulee olemaan yksityiskohtaisuuden kasvun johdosta vaativaa. Kolmiulotteinen skannaus ja liikekaappaus yleistyvät sisällön tuotannossa, jolloin erittäin yksityiskohtaisten kohteiden mallintaminen esimerkiksi peleissä käytettäväksi nopeutuu.

Tämän tutkielman aihealue on laaja ja nopean kehitystyön alaisena. Antoisa jatkotutkimusaihe voisi olla yksityiskohtainen käsittely tämän tutkielman aihealueesta, jostakin sen osasta tai siihen liittyvästä aihealueesta. Esimerkiksi grafiikkaohjelmoinnin kehityksestä kiinnostuneelle katsaus OpenGL-rajapinnan eri määritelmäversioihin vanhimmasta uusimpaan tarjoaa näköalan grafiikkaohjelmoinnin muutoksista kahden vuosikymmenen ajalta. Varjostinohjelmien testausmenetelmät on merkityksekäs tutkimusaihe, sillä varjostinohjelmien monimutkaisuuden lisääntymisen vuoksi testaamisen merkitys kasvaa. Varjostinohjelmien toimintaa voidaan tarkastella intuitiivisesti tarkastelemalla piirtotuloksia, mutta monessa tapauksessa on hyödyllistä testata yksityiskohtaisesti myös varjostinohjelmien sisäistä toimintaa. Grafiikkalaskennan lisäksi näytönohjaimilla voidaan erityisiä rajapintoja käyttäen suorittaa myös yleislaskentaa. Näytönohjaimilla suoritettava yleislaskenta esimerkiksi OpenCL:ää käyttäen on laaja aihealue, joka tarjoaa mahdollisuuksia esimerkiksi näytönohjainta ja keskussuoritinta käyttävien laskenta-algoritmien vertailuun. Grafiikkakiihdytyksen hyödyntäminen vokseleihin perustuvan grafiikan piirrossa on aihe, joka antaa vaihtoehtoisen näkökulman varjostinohjelmointiin.

LÄHTEET

Advancer Micro Devices, Inc. (2012, 15. lokakuuta). RenderMonkey™ Toolsuite. Haettu 7.2.2013 osoitteesta http://developer.amd.com/resources/archive/archived-tools/gpu-tools-archive/rendermonkey-toolsuite/

Akenine-Moller, T., Haines, E., Hoffman, N. (2008). *Real-Time Rendering.* (3. uud. painos). Wellesley: A K Peters, Ltd.

Angel, E., Shreiner, D. (2011). Teaching a Shader-Based Introduction to Computer Graphics. *IEEE Computer Graphics And Applications*, 31(2), 9-13.

Baert, J. (2010, 7. helmikuuta). GLSL Shader IDE's. Haettu 7.2.2013 osoitteesta http://www.forceflow.be/2010/02/07/glsl-shader-ides/

Fatahalian, K., Houston, M. (2008). A closer look at GPUs. *Communications of the ACM*, 51(10), 50-57.

Geer, D. (2005). Taking the graphics processor beyond graphics. *Computer*, 38(9), 14-16.

Glassner, A. S. (1995). *Principles of Digital Image Synthesis.* volume two. San Francisco: Morgan Kaufmann Publishers, Inc.

Gregory, J. (2009). *Game Engine Architecture*. Natick, Massachusetts: A K Peters, Ltd.

Engel, K., Hadwiger, M., Kniss, J. M., Lefohn, A. E., Salama, C. R., Weiskopf, D. (2004). Real-time volume graphics. *ACM SIGGRAPH 2004 Course Notes Article No. 29*.

Ericson, C. (2005). *Real-Time Collision Detection*. San Francisco: Morgan Kaufmann Publishers, Inc.

Hess, R. (2010). *Blender Foundations: The Essential Guide to Learning Blender 2.6*. Focal Press.

Kessenich, J. (2008). *The OpenGL Shading Language*. Haettu 8.11.2011 osoitteesta http://www.cse.chalmers.se/edu/year/2010/course/TDA361/GLSLangSpec.Full.1.30.08.pdf

Khronos Group. (2012, 6. lokakuuta). History of OpenGL. Haettu 7.2.2013 osoitteesta http://www.opengl.org/wiki/History_of_OpenGL

Lengyel, E. (2011). *Mathematics for 3D Game Programming and Computer Graphics.* (3. uud. painos). Boston: Course Technology.

Luebke, D., Humphreys, G. (2007). How GPUs Work. *Computer*, 40(2), 96-100.

Maestri, G. (2009). Games Graphics Unplugged. *Computer Graphics World*, 32(5), 26.

Marroquim, R., Maximo, A. (2010). Introduction to GPU Programming with GLSL. *Computer Graphics and Image Processing (SIBGRAPI TUTORIALS), 2009 Tutorials of the XXII Brazilian Symposium on*, 3-16.

McKesson J. L. (2011). *Learning Modern 3D Graphics Programming*. Haettu 29.1.2012 osoitteesta http://www.arcsynthesis.org/gltut/

Nickolls, J. (2010). The GPU Computing Era. *Micro* 30(2), 56-69.

Owens, J., Davis, UC. (2007). GPU Architecture Overview. *ACM SIGGRAPH 2007 courses*.

Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C. (2008). GPU Computing. *Proceedings of the IEEE*, 96(5), 879-899.

Petrie, J. (2011, 24. maaliskuuta). Is there a successor to RenderMonkey? [Vastaus 1]. Viesti lähetetty keskustelulistalle http://gamedev.stackexchange.com/questions/10132/is-there-a-successor-to-rendermonkey

Pozzer, C., T. (2009). *OpenGL Shading Language*. Haettu 8.11.2011 osoitteesta http://www-usr.inf.ufsm.br/~pozzer/disciplinas/cga_15_opengl_shading.pdf

Rhyne, T.-M. (2000). Computer games' influence on scientific and information visualization. *Computer*, 33(12), 154-159.

Rosenberg, O. (2011). *OpenCL Overview. Khronos Group*. Haettu 3.2.2012 osoitteesta http://www.khronos.org/assets/uploads/developers/library/overview/opencl-overview.pdf

Rost, R. J., Licea-Kane, B., Ginsburg, D., Kessenich, J. M., Lichtenbelt, B., Malan, H. (2009). *OpenGL Shading Language*. (3. uud. painos). Boston: Addison-Wesley.

Sherrod, A. (2008). *Game Graphics Programming.* Boston: Charles River Media.

Watt, A. (1999). *3D Computer Graphics*. (3. uud. painos). Harlow: Addison-Wesley.

Williams, R. (2009). *The Animator's Survival Kit, Expanded Edition: A Manual of Methods, Principles and Formulas for Classical, Computer, Games, Stop Motion and Internet Animators*. (2. painos). London: Faber & Faber.

Woo, M., Neider, J., Davis, T. (1997). *Opengl Programming Guide: The Official Guide to Learning Opengl, Version 1.1. (*2. uud. painos*)*. Addison-Wesley.

Yang, X., Yip, M., Xiaoyue, X. (2009). Visual Effects in Computer Games. *Computer* 42(7), 48-56.