

Juho Saarinen

Direct3D 12 ja sen käyttöohjeen tekeminen

Insinööri (AMK),
tietotekniikka

Kevät 2016



KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä: Saarinen Juho

Työn nimi: Direct3D 12 ja sen käyttöohjeen tekeminen

Tutkintonimike: Tietotekniikan Insinööri

Asiasanat: Direct3D, DirectX, Ohjelmointi käyttöohje, C++

Tämän opinnäytetyön tavoitteena oli luoda ohjelmointikäyttöohje Direct3D 12 version ohjelmointirajapintaan. Käyttöohje vaatii tietokoneessa Windows 10 -käyttöjärjestelmän ja Visual Studio 2015 -kehitystyökalun. Käyttöohje tullaan julkaisemaan kaikkien saatavaksi internetissä.

Koska käyttöohjeen tekeminen vaatii rajapinnan osaamista, ensimmäisenä piti opiskella Direct3D 12 -ohjelmointirajapinta. Direct3D 12:ta on selitetty opinnäytetyön ensimmäisessä osassa. Toinen osa kertoo Visual Studion mukana saatavasta D3D12-malliohjelmasta, jota lähdettiin kehittämään käyttöohjeessa. Kolmas osa työstä kuvaa uusille ohjelmoijille suunnatun ohjelmointikäyttöohjeen tekemisen prosessia.

Opinnäytetyön lopputuloksena oli valmis käyttöohje. Käyttöohjeen avulla uudet C++-ohjelmoijat pystyvät piirtämään objekteja ruudulle käyttämällä Direct3D 12 -rajapintaa.

ABSTRACT

Author: Saarinen Juho

Title of Publication: Direct3D and Making Tutorial for it

Degree Title: Engineer, Information Technology

Keywords: Direct3D, DirectX, Programming tutorial, C++

The aim of this thesis was to create a programming tutorial for the version 12 application programming interface (API) for Direct3D. The tutorial requires the use of Windows 10 operating system and Visual Studio 2015 integrated development environment. The tutorial will be published on internet and will be freely available for everyone.

Researching Direct3D 12 was done first, since making a tutorial for any API requires the knowledge of API itself. What had been learned about Direct3D 12 (D3D12) is disclosed in the first part of the thesis. The second part of the thesis describes the breakdown of Direct3D 12 sample program which is available with Visual Studio. The third part is about the process of creating a programming tutorial for new programmers. The tutorial is based on D3D12 sample program.

The final result of this thesis was finished tutorial. This tutorial can be used by any new C++ programmer to create application where objects are rendered on screen using Direct3D 12 application programming interface.

TERMILUETTELO

Abstraktio	Tapa vähentää monimutkaisuutta
D3D	Direct3D-grafiikkarajapinnan lyhenne.
Efekti	Graafinen lisätehoste.
fps	Kuvastaa, montako kuvaa voidaan piirtää yhden sekunnin aikana.
Ohjelmointirajapinta	Paketti valmiita rutiineja, protokollia ja työkaluja ohjelmistojen kehittämiseen.
Porttaaminen	Kehitysympäristön tai alustan vaihtaminen.
Säie	Komentosekvenssi, joita voi suorittaa eri prosessoriytimillä samanaikaisesti.
UWP	Lyhenne Microsoftin "Universal Windows Platform"-mallista
Verteksi	Piirtoprimitiivien kärkipiste.
Wrapper-luokka	Luokka, jolla voidaan piilottaa tai tiivistää toisen luokan toimivuutta.

SISÄLLYS

<u>1 JOHDANTO.....</u>	<u>1</u>
<u>2 DIRECTX & DIRECT3D.....</u>	<u>3</u>
<u>2.1 Direct3D.....</u>	<u>4</u>
<u>2.2 Direct3D:n historia.....</u>	<u>5</u>
<u>2.3 Direct3D 12.....</u>	<u>7</u>
<u>2.3.1 Komentojonot ja -niput.....</u>	<u>8</u>
<u>2.3.2 Asynkroninen laskenta.....</u>	<u>9</u>
<u>2.3.3 Pipeline State Objektit.....</u>	<u>11</u>
<u>2.3.4 Monen näytönohjaimen hallinta.....</u>	<u>13</u>
<u>3 MALLIOHJELMA.....</u>	<u>15</u>
<u>4 KÄYTTÖOHJEEN TEKEMINEN.....</u>	<u>19</u>
<u>5 YHTEENVETO.....</u>	<u>24</u>
<u>LÄHTEET.....</u>	<u>25</u>
<u>LIITTEET</u>	

1 JOHDANTO

Pelimoottoreita käytetään nykyisin videopelien lisäksi esimerkiksi erilaisissa simulaatioissa sekä elokuvateollisuuden tehosteiden tuottamisessa. Näille käyttöalueille on nykyisin saatavilla korkeatasoisia yleiskäyttöisiä pelimoottoreita. Yleiskäyttöiset pelimoottorit ovat laajoja ja jatkuvan kehityksen alaisia projekteja. Niiden ylläpitämiseen vaadittavia resursseja saadaan pienennettyä käyttämällä muiden tahojen tuottamia kehityskirjastoja, -ympäristöjä ja -rajapintoja. Muun muassa pelien ja simulaatioiden fysiikkalaskentaan käytetään yleensä valmista fysiikkakirjastoa, kuten Nvidian PhysX:ää.

Microsoftilla on iso markkina-asema videopelien keskuudessa Windows-käyttöjärjestelmällä, joka on suosituin käyttöjärjestelmä tietokonepelaajien keskuudessa [4]. Microsoft on julkaissut myös kolme Xbox-pelikonsolia tietokoneilla pelaamisen rinnalle. He kehittivät myös DirectX-ohjelmointirajapintapaketin multimediasovelluskehittäjille. Tämä rajapintapaketti pitää sisällään muun muassa grafiikan piirtämiseen suunnatun Direct3D-rajapinnan [1].

Grafiikan piirtämiseen näytölle käytetään yleensä valmista ohjelmointirajapintaa. Nykyisin käytössä olevista grafiikkarajapinnoista OpenGL [2] julkaistiin ensimmäinen versio tammikuussa 1992, Direct3D kesäkuussa 1995 ja uusimpana OpenGL:n kehittäjäorganisaatio Khronos Group julkaisi helmikuussa 2016 Vulkanin [3]. OpenGL ja Vulkan on käytettävissä useilla eri alustoilla, kun taas Direct3D on käytettävissä vain Microsoftin alustoilla.

Mooren lain mukaan mikropiireissä olevien komponenttien määrä tuplaantuu joka vuosi [4]. Tämä on johtanut prosessoreiden laskentatehojen kasvuun, esimerkiksi vuonna 2000 julkaistujen prosessoreiden kellotaajuudet olivat 800 MHz luokkaa, mutta vuonna 2005 Advanced Micro Devices Inc. (AMD) julkaisi prosessorin, jossa oli kaksi ydintä toimimassa 2000 MHz kellotaajuudella [5].

Prosessorien kaksiytimisyys toi mahdollisuuden suorittaa esimerkiksi ohjelmallista grafiikkaa yhdellä ytimellä, ja samanaikaisesti toisella ytimellä voidaan suorittaa grafiikkarajapinnan komentoja. Vuonna 2008 Intel julkaisi tehokäyttäjille suunnatun i7-prosessorin, joka sisälsi jo neljä fyysistä ydintä [5], joissa jokaisessa pys-

tyy suorittamaan kahta komentoa samanaikaisesti. Näytönohjainta pystyi kuitenkin käskyttämään grafiikkarajapintojen avulla vain yhdeltä säikeeltä, jonka takia syntyi uusi pullonkaula näytönohjaimen ja prosessorin välille. Tätä ongelmaa Microsoft korjasi vuonna 2009 julkaisemalla Direct3Dstä uuden version, joka mahdollisti komentojen nauhoittamisen usealla säikeellä.

Ohjelmointirajapinnat ja kehityskirjastot saattavat kuitenkin laajentua isoiksi kokonaisuuksiksi. Sen takia niitä käyttäville ohjelmoijille toteutetaan referenssisivut, esimerkkiohjelmia ja käyttöohjeita. Käyttöohjeiden on tarkoitus opettaa niiden luoja käyttämään ohjeiden kohdetta, kuten esimerkiksi grafiikkarajapintaa.

Tämän opinnäytetyön tavoitteena on luoda Direct3D:n uusimpaan versioon käyttöohje henkilölle, joka osaa C++-ohjelmointikielen ja etsii uusia haasteita vaikka pelimoottorihjelmoinnin saralta. Henkilökohtaisena tavoitteena on oppia D3D12:n perusteet ja ohjelmointikäyttöohjeiden tekeminen.

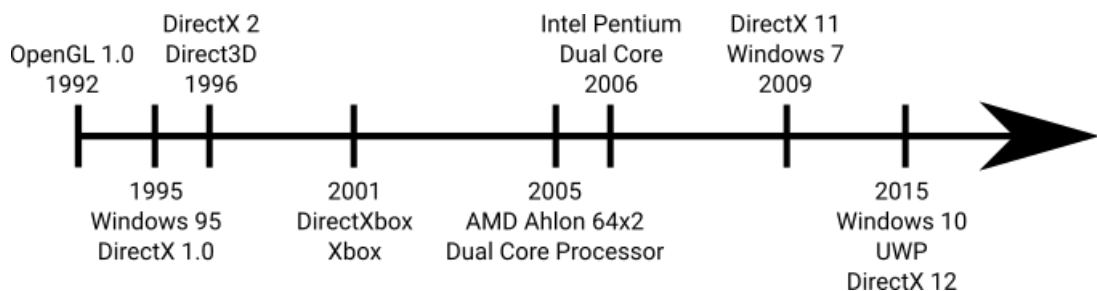
2 DIRECTX & DIRECT3D

DirectX on Microsoftin kehittämä kokoelma ohjelmointirajapintoja, joita käyttämällä ohjelmoijat pääsevät käsiksi muun muassa hiireen, näytönohjaimeen ja äänilaitteisiin. Tähän kokoelmaan kuuluu muun muassa XAudio2, äänien ja äänilaitteiden hallintaan kehitetty ohjelmointirajapinta, XInput, eli HID-laitteiden hallintaan tarkoitettu rajapinta, sekä Direct3D, 3D-grafiikkaan piirtämiseen suunnattu grafiikkarajapinta.

Ensimmäinen DirectX:n versio kehitettiin pelien ja muiden multimediasovellusten ohjelmointia varten Microsoftin Windows 95 -käyttöjärjestelmälle. DirectX:n käyttäminen kuitenkin yleistyi hitaasti, koska DirectX oli saatavilla vain Windows 95:lle, joka kulutti enemmän resursseja kuin kilpaileva DOS-käyttöjärjestelmä. [6]

Microsoft jatkoi kuitenkin DirectX:n kehittämistä ja lisäsi myös sen markkinointia kehittäjille. Seuraava versio toi mukanaan Direct3D-grafiikkarajapinnan, jonka mukana DirectX:n käyttäminen yleistyi. Näin vuosikymmeniä myöhemmin huomaa, kuinka Microsoftin päätös jatkaa DirectX:n kehittämistä kannatti, sillä nykyisin DirectX on yksi eniten käytetyistä rajapinnoista uusissa peleissä.

DirectX:n versiosta 8 lähtien Microsoft kehitti myös DirectXbox-ohjelmointirajapintapaketin, joka oli suunnattu Xbox-tuoteperheelle. Nykyisin Microsoft on kuitenkin keskittynyt 'Universal Windows Platform' -sovelluskehitysmalliin (UWP-malli). UWP-mallilla toteutetut sovellukset on tarkoitettu toimimaan kaikilla Windows alustoilla, eli Windows-käyttöjärjestelmän tietokoneilla, Xbox-pelikonsoleilla ja Windows-älypuhelimilla, ilman suurempien muutoksien tarvetta. UWP-mallin ilmestymisen myötä DirectXbox:n käyttämistä ei suositella, vaan on suositeltavaa käyttää DirectX:n uusinta versiota 12.

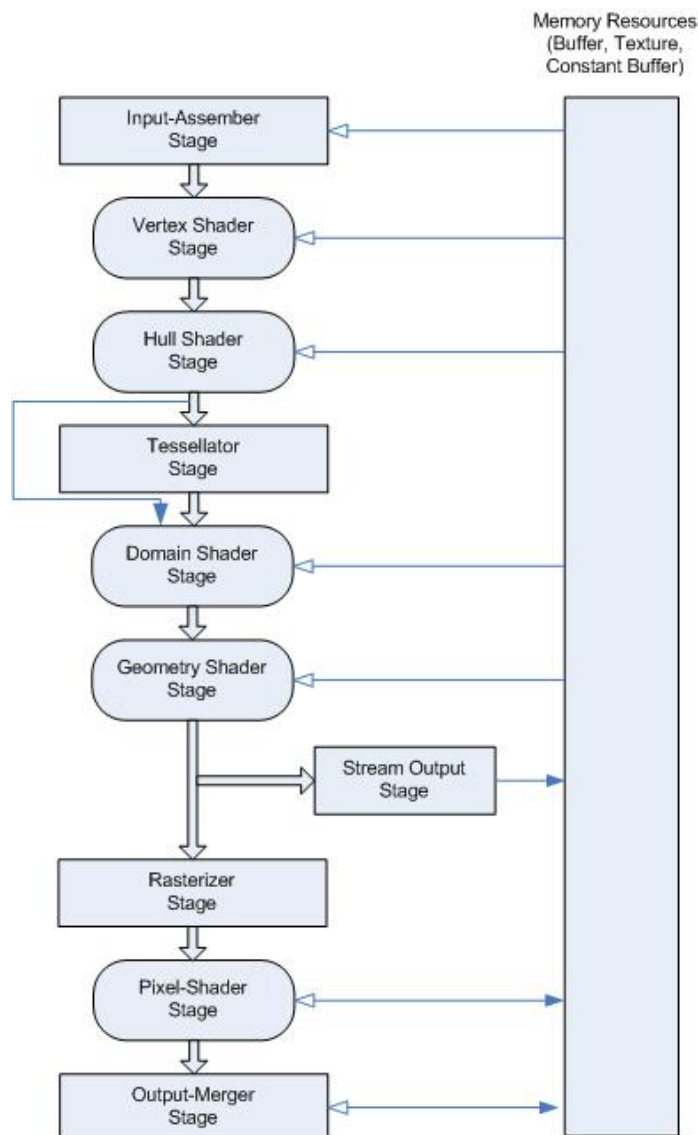


Kuva 1. DirectX:n ja prosessoreiden kehityksen aikajana

2.1 Direct3D

Direct3D (D3D) on Microsoftin suunnittelema 3D-grafiikkapiirtämisen alemman tason ohjelmointirajapinta, joka antaa kehittäjälle muun muassa mahdollisuuden manipuloida 3D-objekteja sekä käyttää näytönohjaimen laitteistokiihdytystä objektien piirtämiseen. [7.]

D3D piilottaa näytönohjaimien välillä eroavat arkkitehtuurit, antamalla kehittäjien käyttöön grafiikkarajapinnan. Direct3D:n julkaisusta on kuitenkin jo yli kaksikymmentä vuotta, joten se on myös ymmärrettävästi laajentunut ja siitä on tullut vaikeaselkoisempi. Vaikka Direct3D on nykyisin laaja, se on silti helpommin käytettävä, kuin ohjelmoida käyttämällä näytönohjaimenarkkitehtuurin konekieltä.



Kuva 2. Direct3D 11 -grafiikan liukuhihna [8]

Käytettäessä grafiikkarajapintaa grafiikkaobjektien piirtokomennot käyvät läpi useita peräkkäisiä operaatioita näytönohjaimessa. Tämä on grafiikan liukuhihna (kuva 2), joka kuvaa piirron eri vaiheiden tehtäviä ja niiden välillä liikkuvaa informaatiota.

Input-Assembler-vaiheessa kaikki näytönohjaimelle lähetetty data otetaan vastaan näytönohjaimella. Verteksivarjostin käsittelee piirtyvät verteksipisteet. Yleensä ”kameroiden” ja muiden objektien näytönohjaimella liikuttaminen tapahtuu tässä vaiheessa. Geometriavarjostimessa käsitellään pelimaailmassa olevien objektien muodot. Geometriavarjostimet eivät ole pakollisia, mutta jos niitä on käytetty, niiden tulos tallennetaan näytönohjaimelle Stream-Output-vaiheessa. Rasterizer-vaihe valmistele objektit pikselivarjostimelle. Pikselivarjostimissa tapahtuu muun muassa pikselikohtainen valaistus ja erilaiset jälkikäsittelyefektit. Output-Merger-vaiheessa putken kaikki ominaisuudet yhdistetään ja näytetään ruudulla. [8.]

2.2 Direct3D:n historia

Direct3D:n julkaisuvaiheessa sillä oli vain kaksi kilpailijaa, OpenGL ja Glide, jotka molemmat olivat erittäin suosittuja kehittäjien keskuudessa. OpenGL:n kehittäminen jatkuu vieläkin, mutta Gliden, joka oli yhden näytönohjainvalmistajan tuottama rajapinta, kehittäminen lopetettiin 2000-luvun alussa. Gliden kehittämisen loppua uusia grafiikkarajapintoja, jotka olisivat olleet suosittuja, ei julkaistu yli kymmeneen vuoteen, ennen kuin näytönohjainvalmistaja AMD kertoi kehittävänsä uutta rajapintaa, jossa on huomattavasti vähemmän abstraktiota kehittäjän ja näytönohjaimen laitteiston välillä.

AMD:n Mantle-grafiikkarajapinnan suurimpana tavoitteena oli poistaa pullonkaula, joka oli syntynyt moniytimisten prosessoreiden myötä. Tämän pullonkaulan poistamisen oli tarkoitus tapahtua antamalla kehittäjälle mahdollisuuden käyttää piirtokomentoja usealta eri säikeeltä samanaikaisesti sekä vähentämällä kehittäjän ja näytönohjaimen välistä abstraktiota. Direct3D:n kehittäjä Microsoft kuitenkin ilmoitti päivittävänsä rajapintaansa suuresti, jotta pullonkaulasta päästäisiin eroon. OpenGL:n kehitysorganisaatio, Khronos Group, ilmoitti myös kehittävänsä

uutta rajapintaa samalla tavoitteella. AMD ja Khronos tekivät sopimuksen, jonka seurauksena Mantlen kehittäminen loppui ja sen kehitysresurssit siirtyivät Khronoksen uuteen rajapintaan, Vulkanisiin [9]. Myös tietokone- ja älypuhelinvalmistaja Apple julkisti uuden laitteistoläheisemmän rajapinnan, joka on tarkoitettu käytettäväksi sen älypuhelimissa.

Direct3D:n ensimmäinen versio julkaistiin osana DirectX:n toista versiota, vuonna 1996. Direct3D-grafiikkarajapinnan ensimmäisen version käyttäminen oli kuitenkin hankalaa ja vaati enemmän resursseja kuin kilpailijansa, joten sen omaksuminen oli hidasta.

Microsoft ei kuitenkaan ottanut OpenGL:ää osaksi DirectX:ää, vaan kehitti Direct3D:tä pelien kehittäjien tarpeiden mukaan. DirectX:n kahdeksannessa versiossa D3D korvasi myös 2D-piirtämisen kehitetyn DirectDraw-rajapinnan [10].

Ennen DirectX:n yhdeksättä versiota Microsoft mainitsi harvoin Direct3D:n versionumeroita, mutta DirectX 9:n mukana tuli Direct3D 9. Tästä johtuen jotkut uudet kehittäjät ovat valitettavasti unohtaneet, että DirectX-kokoelmasta löytyy muitakin rajapintoja kuin D3D.

Direct3D 10 toi suuria muutoksia rajapintaan. Aikaisemmissa versioissa näytönohjaimien ei ollut pakko tukea kaikkia ominaisuuksia, joten kehittäjien tuli tarkistaa näytönohjaimelta, tukeeko se haluttua ominaisuutta. D3D10:n kanssa oli asetettu vaatimus, joka määräsi, mitkä kaikki ominaisuudet on löydettävä, jotta näytönohjainta voi kutsua DirectX 10 -yhteensopivaksi.

Hieman DirectX 10:n julkaisun jälkeen Microsoft julkaisi Direct3D:hen pienen päivityksen, version 10.1. D3D10.1:n mukana esiteltiin ominaisuustasojen konsepti, joka on oleellinen Direct3D 12:ssa. Ominaisuustasojen on tarkoitus ilmaista tarkemmin, mitä ominaisuuksia vaaditaan näytönohjaimelta. Ominaisuustasot-konseptissa korkeampi taso pitää sisällään kaikki matalamman tason ominaisuudet, mutta tuo myös uusia, tai ainakin paranneltuja, ominaisuuksia, jotka ovat saattaneet olla vapaavalintaisia ominaisuuksia matalammalla tasolla [11].

Direct3D 11:n mukana tuli huomattavasti parannuksia kymmenenteen versioon, mutta suurin osa D3D10.1:n tukevista näytönohjaimista tuki suoraan myös D3D11:tä. Sen mukana tuli kuitenkin myös uusia ominaisuuksia, kuten laskenta-

varjostimet. Laskentavarjostimien avulla voidaan tehdä myös vaativaa laskentaa näytönohjaimella, lähes yhtä hyvin kuin Nvidian CUDA- tai Khroksen OpenCL laskentakielillä.

D3D11 toi myös erittäin tärkeän ominaisuuden, alustavan tuen käyttää D3D:n komentoja monella eri säikeellä. Täten kehittäjät pystyivät hyödyntämään paremmin nykyistä, moniytimistä prosessoriarkkitehtuuria. Tämä tuki oli kuitenkin alkeellinen ja onkin työstetty uudelleen uusimpaan D3D:n versioon.

Microsoft käytti Direct3D 11:n kanssa runsaasti ominaisuustaso-konseptia, kuten muun muassa Xbox One -pelikonsolin kanssa, jossa taso 11.X toi matalamman abstraktion ja muita ominaisuuksia, jotka tulivat tietokoneelle käytettäväksi vasta Direct3D 12:ssa.

Kesäkuussa 2015 Microsoft julkaisi Direct3D 12:n, joka on edeltäjiään tehokkaampi muun muassa, koska kehittäjän ja näytönohjaimen välistä abstraktiota on vähennetty. Muita muutoksia on parannettu tuki monisäikeistetylle piirtämiselle sekä siirtämällä ohjelmien näytönohjaimella olevan muistin hallinnan kehittäjien vastuulle [12].

2.3 Direct3D 12

Direct3D 12 (D3D12) on uusin versio Direct3D-rajapinnasta. Siinä vähennettiin kehittäjän ja näytönohjaimen välistä abstraktiota antamalla muun muassa kehittäjille vastuu ohjelman muistinkäytöstä ja grafiikkarajapinnan käyttämisestä useilla ytimillä. Nämä muutokset tekevät D3D12:sta myös vaikeammin käytettävän edeltäjiinsä verrattuna, mutta sillä pystyy käyttämään näytönohjainta tehokkaammin.

Vaikka D3D12 teki suuria muutoksia rajapintaan, Microsoft jatkoi Direct3D nimityksen käyttämistä, eikä tehnyt samaa valintaa kuin Khronos, joka nimesi madalletun abstraktion grafiikkarajapinnan Vulkaniksi.

Abstraktiota vähentämällä pystyttiin myös vähentämään prosessorin, näytönohjaimen, sekä niiden välissä olevia pullonkauloja. Ne ovat olleet suurimmat ongelmat hienosäätäessä kuvien piirtoon menevää aikaa.

Ominaisuustasojen lisäksi Direct3D 12:ssa käytetään resurssienhallinnan ja valinnaisien ominaisuuksien kanssa portaita. Näiden portaiden avulla kehittäjä voi selvittää, kuinka hyvin näytönohjain tukee ominaisuuksia. Nämä kaikki porrastetut ominaisuudet ovat valinnaisia, mutta vaativat tuen laitteistotasolta. Niiden avulla voidaan kuitenkin hyödyntää näytönohjaimen laskentatehoa paremmin.

Direct3D 12:sta pitää myös muistaa, että se on vielä erittäin uutta teknologiaa. Esimerkiksi Hitman-pelin ohjelmoijan Jonas Meyerin mukaan D3D12-sovelluksessa on tällä hetkellä vaikeata hyödyntää laskentatehoa yhtä hyvin kuin D3D11-sovelluksessa. Hän kuitenkin uskoo näytönohjaimien ajureiden ja laitteiston kehittyessä myös D3D12:n tehonkäytön tulevan paranemaan. [13.]

2.3.1 Komentojonot ja -niput

Direct3D 11:n mukana tuli komentolistat, jotka toimivat pohjana Direct3D 12:n komentojonoille. D3D11:n pitää luoda väliaikainen laskennallinen konteksti, jota voi käskyttää komentolistan avulla. Kaikki komentolistalle laitettut käskyt esikäsitellään väliaikaisessa kontekstissa heti. Tätä esikäsitelyä Microsoft kutsuu nauhoittamiseksi. [14.]

Komentolistojen avulla osa pelimaailmasta voidaan nauhoittaa yhdellä säikeellä ja samanaikaisesti pystytään nauhoittamaan loput maailmasta muilla säikeillä [14]. Jotta nauhoitukset toteutuisivat eli piirtyisivät ruudulle, ne pitää vielä toistaa suorassa kontekstissa. Toistaminen tapahtuu lähettämällä esikäsitelty komento lista ja sen vaatimat tiedot näytönohjaimelle. Tätä D3D11:n suoraa kontekstia pystyy kuitenkin käsittelemään vain yhdeltä säikeeltä, eli yksi säe joutuu tekemään enemmän työtä kuin muut.

Direct3D 12:ssa ei ole suoraa kontekstia, vaan jokaiselle säikeelle voi luoda oman komentojonon. Komentojonoihin voi nauhoittaa grafiikanpiirtokomentoja miltä tahansa säikeeltä. Komentoiono voidaan käskää suoriutumaan miltä tahansa säikeeltä, mutta näytönohjain pystyy käsittelemään vain yhtä komentojonoa kerrallaan. Tästä johtuen Nvidia suosittelee, että vain yksi säe on vastuussa komentojonojen suorittamisen käskyttämisestä [15].

Microsoft sai komentojonoilla vähennettyä yhden säikeen työkuormaa, mutta usean komentojonon synkronointi keskenään annettiin kehittäjien vastuulle. Koska kehittäjät ovat vastuussa synkronoinnista, he voivat myös hienosäätää sitä, jolloin on mahdollista käyttää näytönohjaimen resursseja paremmin. Vastuun mukana tulee myös riski toteuttaa synkronointi huonosti, jolloin resursseja saattaa mennä hukkaan.

Direct3D 12:n mukana tuli myös komentoniput täydentämään komentojonoja. Komentoniput on suunniteltu lyhyiden, mutta useasti käytettävien komentojonojen tallentamiseen näytönohjaimelle. Komentonippu voi esimerkiksi pitää sisällään neliön piirtämiseen vaadittavat käskyt. Näin joka kerta kun piirretään samankaltainen neliö, ei tarvitse aina laittaa samoja komentoja jonoon, vaan voidaan kutsua valmista komentonippua. Näin saadaan vähennettyä prosessorin ja näytönohjaimen välillä kulkevaa tietoa, jolloin myös kyseinen pullonkaulan haitat vähenevät.

Yksi komentonipun eduista komentojonoon verrattuna on se että esinauhoitettua komentonippua voidaan käyttää usein, jolloin prosessorin ei tarvitse esikäsitellä työtä toistuvasti. Komentojonoa ei voi käyttää montaa kertaa, vaan se pitää luoda jokaisen suorituksen jälkeen uudelleen. Komentonipun voi myös tehdä millä säikeellä tahansa ja sitä voidaan käyttää muidenkin säikeiden komentojonoissa.

[15]

2.3.2 Asynkroninen laskenta

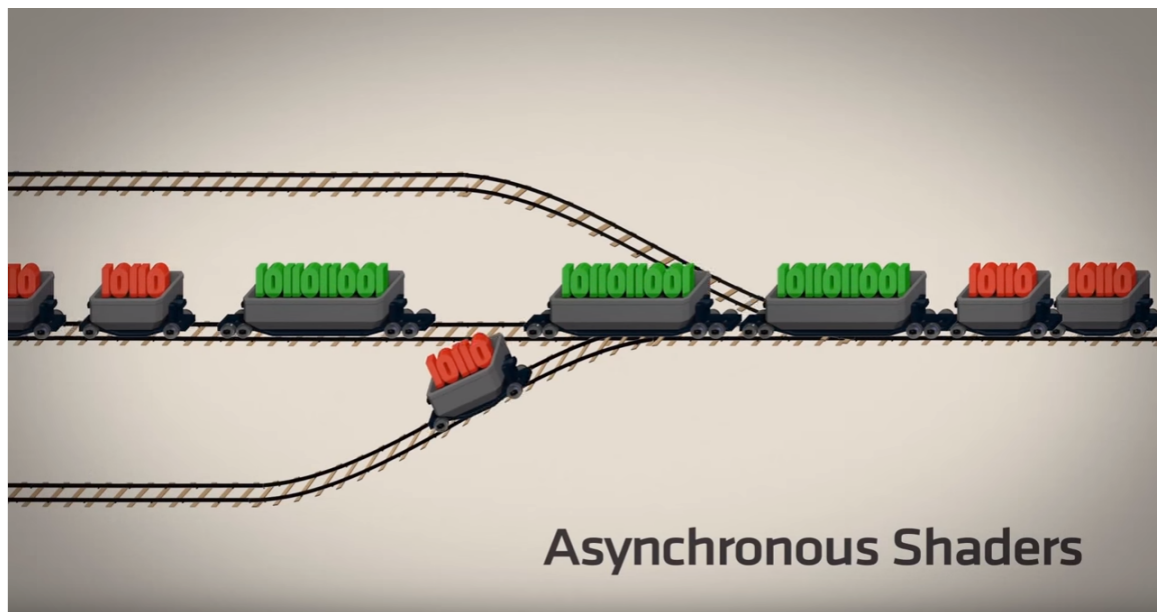
Yksi Direct3D 12:n vapaavalintaisista ominaisuuksista, joka vaatii tuen laitteistotasolta, on asynkroninen laskenta. Asynkroninen laskenta vaatii, että näytönohjaimessa on monta laskentamoottoria. Yksi laskentamoottori pitää sisällään tuhansia laskentaytimiä, joille laskentakuorma voidaan jakaa tasaisesti.

Videopelit ja simulaatiot ovat täynnä laskutehtäviä. Jos ilman laskentamoottoreita tehtävän tekemiseen vaaditaan vain 70 % ytimistä, niin joko 30 % ytimistä on tyhjän panttina tai kaikki ytimet toimivat 70 % täydestä tehostaan. Laskentamoottoreilla saman tehtävän suorittamisessa saattaa kestää kauemmin, koska sillä voi olla käytettävissä vähemmän laskentaytimiä, mutta samanaikaisesti voidaan

suorittaa useita laskutehtäviä. Näin kaikki ytimet voivat jatkuvasti toimia täydellä tehollansa, jonka takia saadaan pienennettyä ruudun piirtoon käytettyä aikaa.

Asynkroninen laskenta ei kuitenkaan ole täydellinen ratkaisu kaikkeen kuten muun muassa Jonas Meyer on kertonut: ”Asynkroninen laskenta ei ole mikään taikasauva, joka antaa suuret hyödyt heti” [16]. Hän kertoi haastattelussa kuinka saivat vain 5-10 % enemmän tehoa irti asynkronisilla varjostimilla. Lisäksi hän mainitsi, kuinka asynkronisiin varjostimiin on erittäin hankalaa tehdä hienosäätöjä.

Asynkronisten varjostimien hienosäädön vaikeus saattaa johtua siitä, miten ne toimivat. AMD:llä asynkroniset varjostimet ovat automaattinen prosessi, joka toimii samalla tavalla kuin moottoritielle meno. Kun auto tulee moottoritielle rampilta, se pääsee siellä olevaan tyhjään väliin. (kuva 3) [17.]



Kuva 3. Asynkronisten varjostimien toiminta [17.]

2.3.3 Pipeline State -objektit

Kuten aikaisemmin on esitetty, kuvan näkyminen ruudulle vaatii monen eri vaiheen läpikäyntiä (kuva 3) [8]. Direct3D 10 toi kehittäjälle mahdollisuuden vaihtaa liukuhihnalla käytettäviä asetuksia, State Objektilla. State Objektien avulla voi-

daan esimerkiksi liukuhinnan lopussa kertoa, miten päällekkäin asetetut pikselit yhdistetään keskenään Blend State -objektilla.

Direct3D 12 toi mukanaan jatkokehitetyn mallin D3D10 State Objektille, Pipeline State Objektit (PSO). PSO pitää sisällään usean State Objektin tiedot yhdessä paketissa. Tämän ansiosta PSO:iden avulla voidaan tehdä erilaisia valmiita asetuspaketteja näytönohjaimen muistiin, joiden välillä voi vaihdella tarpeen tullen ilman ylimääräisiä grafiikkarajapinnan kutsuja [18].

Intelin mukaan tälle jatkokehitykselle oli tarvetta, koska rajapintakutsut ovat raskaita [19]. Jokainen rajapintakutsu käyttää prosessointitehoa, ja koska aikaisemmin jokaista asetusta käsiteltiin itsenäisesti, kutsuja tapahtui useita. Nykyisin kaikki asetukset voidaan asettaa kerralla, ja lähettää pakettina näytönohjaimelle. Tällöin tarvitaan vain yksi rajapintakutsu, kun halutaan vaihtaa usean State Objektin arvoa samanaikaisesti.

Yksinkertaistettuna kun tekee kaksi Pipeline State Objektia, niihin molempiin voidaan asettaa esimerkiksi eri varjostimet ja Blend State. Sitten valmiit PSO:t lähetetään näytönohjaimelle, ja jatkossa käytettävä PSO-objekti voidaan valita vain yhdellä rajapintakutsulla.

Esimerkkinä malliohjelmassa olevan Pipeline State Objektin luominen. Ensin rakennetaan molemmat varjostimet createPSTask ja createVSTask-funktioilla.

```
auto createPipelineTask = (createPSTask && createVSTask).then([this]() { ... });
```

Kun varjostimet on rakennettu, voidaan alkaa käsittelemään PSO:ta. Ensimmäisenä määritetään verteksivarjostimen sisään menevät tiedot float3 pos ja float3 color.

```
static const D3D12_INPUT_ELEMENT_DESC inputLayout[] =
{ { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
  D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 },
  { "COLOR", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12,
  D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };
```


Seuraavaksi luodaan uusi Pipeline State Objekti ja sille annetaan tiedoksi edellä määritetty verkteksivarjostimen `inputLayout`, grafiikkajuuren paikka ja ladatut varjostimet.

```
D3D12_GRAPHICS_PIPELINE_STATE_DESC state = {};
state.InputLayout = { inputLayout, _countof(inputLayout) };
state.pRootSignature = m_rootSignature.Get();
state.VS = { &m_vertexShader[0], m_vertexShader.size() };
state.PS = { &m_pixelShader[0], m_pixelShader.size() };
```

Sitten määritetään grafiikkaliukuhinnan asetukset, asettamalla niihin Direct3D 12:n oletusarvot.

```
state.RasterizerState = CD3DX12_RASTERIZER_DESC(D3D12_DEFAULT);
state.BlendState = CD3DX12_BLEND_DESC(D3D12_DEFAULT);
state.DepthStencilState = CD3DX12_DEPTH_STENCIL_DESC(D3D12_DEFAULT);
state.SampleMask = UINT_MAX; //BlendStaten pikselien maskausta varten
```

Ennen PSO:n luomista määritetään millaisia objekteja PSO:lla käsitellään.

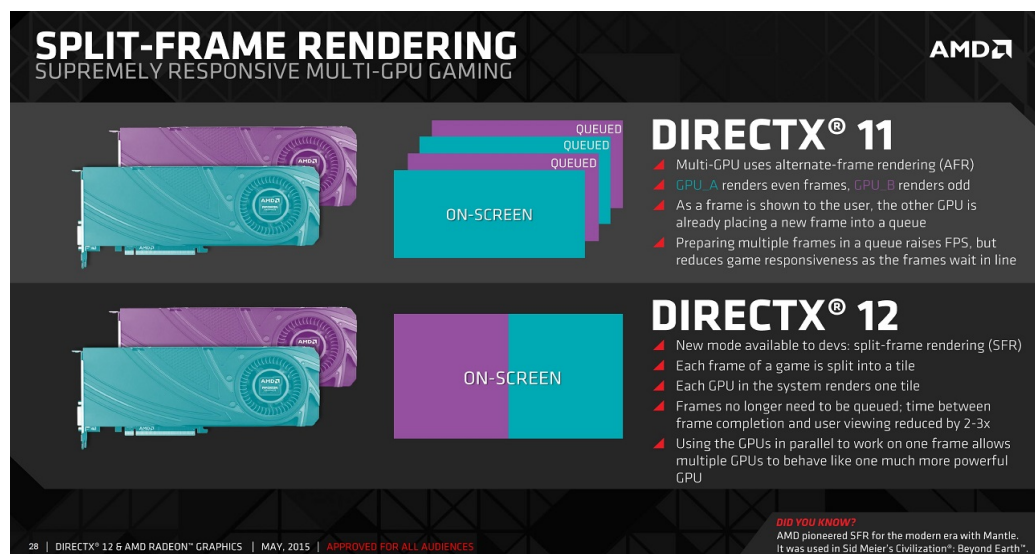
```
state.PrimitiveTopologyType = D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE;
state.NumRenderTargets = 1;
state.RTVFormats[0] = DXGI_FORMAT_B8G8R8A8_UNORM;
state.DSVFormat = DXGI_FORMAT_D32_FLOAT;
state.SampleDesc.Count = 1;
```

Viimeisenä PSO luodaan edellä syötetyistä tiedoista ja varjostimet poistetaan prosessorin muistista.

```
DX::ThrowIfFailed(m_deviceResources->GetD3DDevice()->CreateGraphicsPipelineState(&state, IID_PPV_ARGS(&m_pipelineState)));
m_vertexShader.clear();
m_pixelShader.clear();
```

2.3.4 Monen näytönohjaimen hallinta

Vuonna 2004 Nvidia julkaisi SLI-tekniikan, jota AMD seurasi seuraavana vuonna Crossfire-tekniikalla, jonka avulla yksi prosessori voi hyödyntää useaa näytönohjainta samanaikaisesti. Nämä teknologiat mahdollistivat paremman fps:n saamisen, tuomalla muun muassa mahdollisuuden alkaa piirtämään seuraavaa ruutua toisella näytönohjaimella, kun ensimmäiselle piirretään vielä edeltävää (kuva 4).



Kuva 4. Split-Frame-piirtäminen. [20.]

Direct3D 12:ssa Microsoft lisäsi mahdollisuuden käsitellä jokaista asennettua näytönohjainta erikseen. D3D12:n avulla ruudun voi jakaa osiin, ja jokainen osa voidaan piirtää eri näytönohjaimella (kuva 4) [20]. Toinen ominaisuus, joka puuttuu aikaisemmista teknologioista, on yhdistetyt näytönohjaimen muistit. Yhdistettyjen muistien avulla käytettävä muistiavaruus kasvaa, jolloin prosessorin ei tarvitse yhtä usein siirtää tietoa näytönohjaimelle.

Näiden uusien ominaisuuksien käyttäminen on kehittäjien vastuulla. Näin voidaan myös hienosäätää monen näytönohjaimen keskinäistä käyttäytymistä, joka puolestaan voi johtaa ruutujen piirtämiseen kuluvan ajan vähenemiseen.

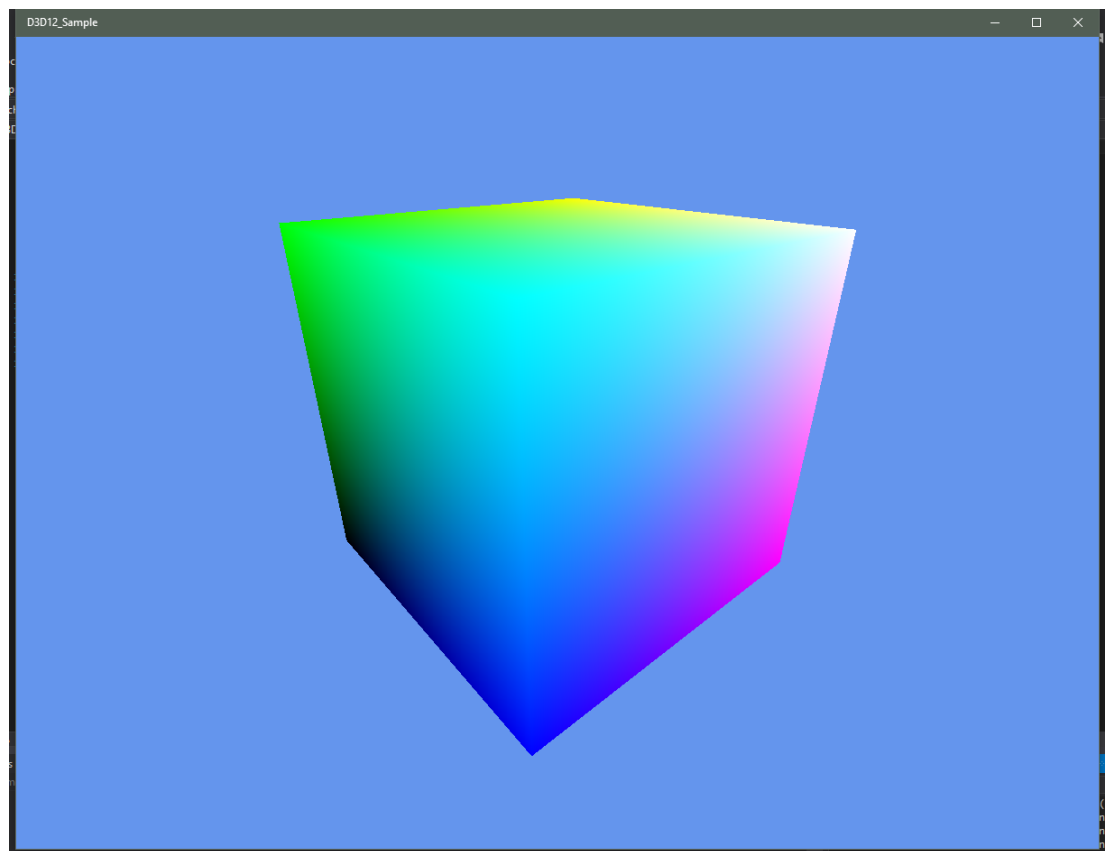
Yksi uusi teknologia, joka voi saada suuresti hyötyä monen näytönohjaimen hallinnasta, on virtuaalitodellisuus. Virtuaalitodellisuudessa yleensä piirretään samaa pelimaailmaa kahdesta eri kuvakulmasta. Koska D3D12:sta ruudun eri puo-

let voidaan piirtää eri näytönohjaimella, eri kuvakulmista johtuvat laskutoimitukset voidaan aina toteuttaa myös omalla näytönohjaimellaan.

3 MALLIOHJELMA

Direct3D 12:n herätti ilmestyessä paljon kiinnostusta kehittäjien keskuudesta, mutta tarjolla oli vain keskeneräiset referenssisivut ja joitakin esityksiä, joilla D3D12 oli markkinoitu, mutta ei esimerkkikoodeja tai käyttöohjeita. Microsoft viimeisteli referenssisivut muutaman viikon sisällä, ja lisäsi esimerkkejä sekä ohjeita niille kehittäjille, jotka haluavat siirtyä Direct3D 11:sta uudempaan versioon. Näiden päivitysten jälkeen oli vihdoin mahdollista piirtää kuutio ruudulle Direct3D 12:ta käyttämällä, vaikka aikaisempaa kokemusta Direct3D-rajapinnasta ei ole.

Pari kuukautta myöhemmin D3D12-sivut oli taas päivitetty. Tällä kertaa siellä oli julkaistu erilaisia aloitusoppaita, esimerkkiohjelmaa ja tarkempia kuvauksia uusista ominaisuuksista. Samoihin aikoihin Visual Studio 2015 -kehitysympäristö sai myös päivityksen, jonka mukana tuli muun muassa yksinkertainen Direct3D 12 -malliohjelma.



Kuva 5. Direct3D 12 -malliohjelman tuottama kuva

Visual Studio Direct3D 12 -malliohjelma on toteutettu Microsoftin "Universal Windows Platform"-mallilla (UWP) (kuva 5). UWP-mallilla kehitetty ohjelma toimii kaikilla Windows-pohjaisilla alustoilla ilman, että kehittäjän tarvitsisi tehdä suu-rempia muutoksia.

UWP-malli näkyy D3D12 -malliohjelman ikkunankäsittelykoodissa. Ikkunan käsittelyssä on käytetty sellaisia avainsanoja ja symboleita, jotka eivät kuulu C++:n teknisiin määrittelyihin tai tarkoittavat jotakin muuta. Esimerkiksi sieltä löytyy termi 'ref class', jota ei löydy C++:sta, ja sen yhteydessä esille tulevasta '^'-merkkistä, jolla normaalisti kuvataan bittikohtaista xor-operaatiota.

Käyttöohjeen kannalta UWP-mallilla ei kuitenkaan ole merkitystä, sillä kaikki sen yhteydessä, ja muutkin malliohjelman koodit ovat hyvin kommentoitua ja selkeää. Koodissa muuttujat, objektit ja funktiot on nimetty kuvaavasti. Funktioista löytyy myös selkeitä kommentteja mitä ja miksi milloinkin tapahtuu.

DirectXHelper.h-tiedosto pitää sisällään funktioita, joiden tarkoitus on avustaa DirectX:n käyttämisessä. Näitä funktioita on muun muassa ThrowIfFailed, jonka avulla voi etsiä Win32-rajapinnasta tulevia virheitä, ja ReadDataAsync, jonka avulla tiedosto voidaan lukea kiintolevyllä nykyisen tehtävän rinnalla.

d3dx12.h-tiedosto sisältää yksinkertaisia funktioita ja rakenteita. Esimerkiksi sen sisältä löytyvät CD3DX12_RECT ja CD3DX12_BOX-rakenteet, jotka sisältävät nelikulmioiden ja laatikoiden perustiedot, kuten reunojen/nurkkien sijainnin.

StepTimer.h sisältää animaatioiden ja simulaatioiden kiinteään ajastukseen tarvittavan luokan. Kiinteällä ajastuksella voidaan esimerkiksi päivittää pelin sisältämät fysiikkamallit kiinteällä aikavälillä, jolloin tulokset ovat vakaammat. Esimerkiksi pelaaja ei liiku oven toiselle puolelle, vaikka pelimaailman päivityksessä olisi kulunut oletettua kauemmin.

ShaderStructures.h-tiedosto sisältää varjostimien käyttämän ModelViewProjection-rakenteen (MVP) sekä verteksien paikkaa ja väriä kuvaavan rakenteen. ModelViewProjection rakenne on kolme matriisia: Model-matriisin avulla objektia/skeneä voidaan esimerkiksi liikuttaa. View-matriisin avulla määritetään kameran näyttämä alue.

SamplePixelShader.hlsl (liite 1) ja SampleVertexShader.hlsl (liite 2) tiedostot ovat yksinkertaisia varjostimia, joiden avulla laatikko väritetään. Verkteksivarjostimessa vertekspisteitä siirretään käyttämällä MVP-matriisia, jonka jälkeen vertekspisteiden värit välitetään pikselivarjostimelle, jossa ruudun pikselit väritetään.

DeviceResources-luokka sisältää muun muassa komentojonon, näytönohjaimelle muistiosoitteen ja ikkunan, johon ohjelman tuottama kuva piirretään. Kyseinen luokka on jaettu DeviceResources.h-tiedostoon, jossa esitellään luokka ja sen funktiot, sekä DeviceResources.cpp-tiedostoon, jossa luokan funktiot on implementoitu. DeviceResources-luokassa alustetaan näytönohjain ja sen resurssit, kuten Direct3D. Keskeisiä Direct3D 12 -ominaisuuksia, kuten komentojonoa käytetään tästä luokasta luodun objektin avulla.

Sample3DSceneRenderer.cpp-tiedostossa luodaan aktiivinen pelimaailma. Siitä löytyy muun muassa Update-, Render- ja CreateDeviceDependentResources-funktiot, joiden avulla pelimaailma päivitetään sekä piirretään ruudulle. CreateDeviceDependentResources:issa rakennetaan värikäs kuutio (kuva 5), jota pyöritetään Update-funktiossa. Render-funktiossa käytetään DeviceResources-objektin komentolistaa piirtämään aikaisemmin luotu kuutio ruudulle.

App.cpp-tiedostosta löytyy ohjelman aloitus sekä UWP-ikkunan luonti ja tapahtumat, kuten ikkunan sulkeminen ja koon muuttaminen. Tämä tiedosto on myös hyvä paikka luoda hiiren tai muiden haluttujen input-laitteiden käyttöön suunnitellut objektit.

Projektiin luodaan myös luokka, ja sen tiedostot, siihen valitun nimien perusteella, NimiAppMain. Tämä luokka yhdistää SceneRenderer- ja App-objektit. Lisäksi siinä voidaan asettaa StepTimer:n avulla SceneRenderer toteutumaan vakiolla aikavälillä.

Edellä mainittujen tiedostojen lisäksi malliohjelmasta löytyy muitakin, käyttöohjeen kannalta vähemmän tärkeitä, tiedostoja, kuten Package.appxmanifest, esikääntämiseen suunnatut pch-tiedostot ja ohjelman rekisteriavain. Työn liitteestä 3 löytyy ohjelman luokkakaaviot, joiden avulla näkee tarkemmin luokkien ja rakenteiden rakenteet.

Vaikka malliohjelma on valmis Direct3D 12 -ohjelma, siitä puuttuu ainakin monisäikeistys, tekstuureiden sekä 3D-mallien käsittely ja peliobjektit. Peliobjektit lisäämällä voi toteuttaa yksinkertaisen monisäikeistyksen. Peliobjektit myös pitävät sisällään käsitellyt tekstuurit ja 3D-mallit.

4 KÄYTTÖOHJEEN TEKEMINEN

Direct3D 12 -grafiikkarajapinnan perusteiden opiskelun ja saatavilla olevien dokumentaation tutkimisen jälkeen alkoi käyttöohjeen tekemisprosessin suunnittelu.

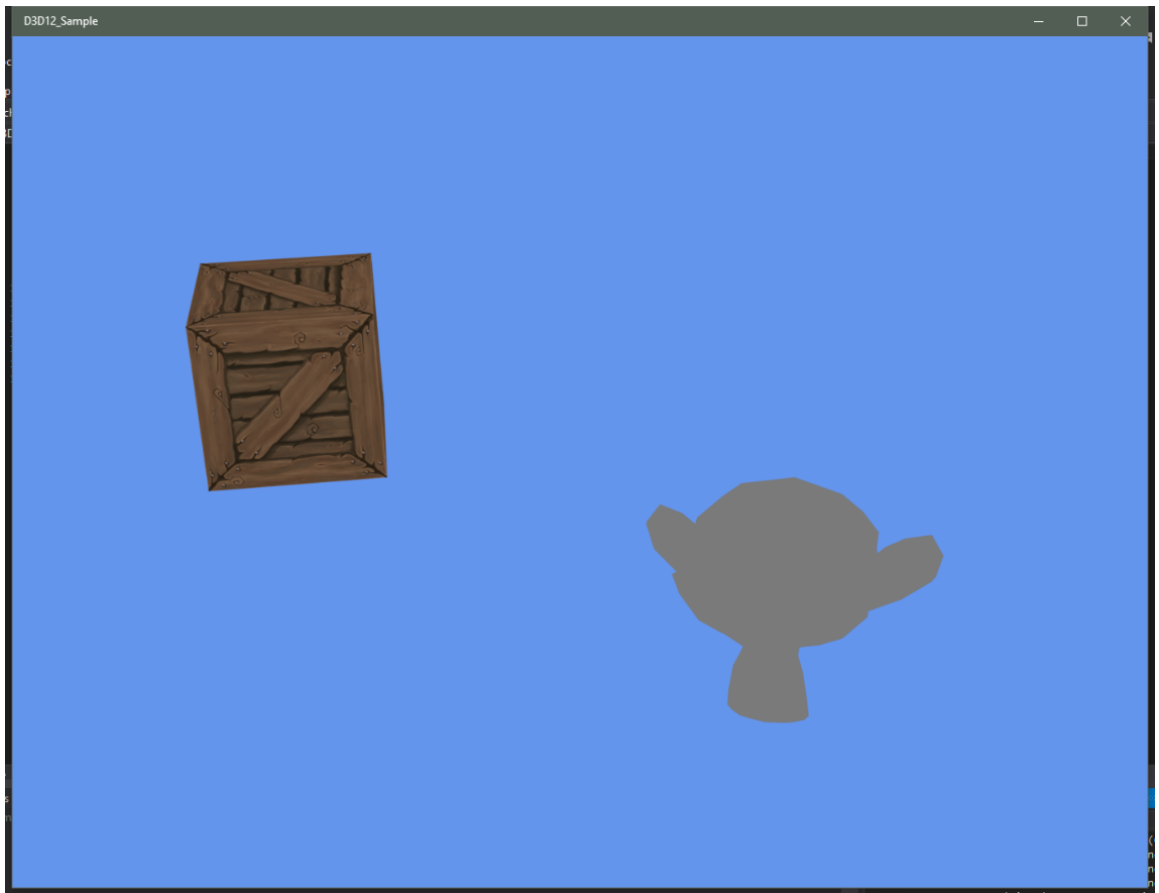
Ennen kuin käyttöohjetta voi alkaa tekemään, se kannattaa suunnitella hyvin. Ensimmäisenä suunnitellessa käyttöohjetta mietitään sen tarpeellisuutta, jonka jälkeen sille rajataan kohdeyleisö. Heidän avulla voidaan määrittää, mitä tietoja lukijan tulisi osata jo entuudestaan. Suunnittelun lopuksi määritetään käyttöohjeen lopputulos.

Suunnitelmissa rajattiin ensimmäisenä pois Direct3D:n yhdenkymmenestä versioista kahdenteentoista versioon siirtymisestä kertova käyttöohje. Tämä päätös johtui oletuksesta, että kokeneemmat D3D11:n käyttäjät, ja todennäköisesti myös Microsoft, tekevät kyseisiä käyttöohjeita paljon, jolloin on vaikeampaa saada näkyvyyttä. Tämän vuoksi jäi mahdollisuudet toteuttaa käyttöohje OpenGL:stä siirtyville tai D3D12:n uusille ohjelmoijille. OpenGL:n ja Direct3D 12:n ero oli kuitenkin huomattu erittäin suureksi. Tämän takia OpenGL:stä siirtyville sopii hyvin myös uusille ohjelmoijille suunnattu ohje. Käyttöohjeen suuntaamista uusille ohjelmoijille tuki myös hyvin dokumentoitu ja kaikkien saatavilla oleva malliohjelma. Samalla jätettiin pois koko DirectX:ään suuntautuvat käyttöohjeet rajallisen aikataulutuksen vuoksi.

Kohdeyleisön rajaamista varten piti tutkia, minkä tasoiset kehittäjät ymmärtäisivät malliohjelmaa vähäisellä lisädokumentoinnilla. C++-perusteiden osaamisen vaatimus näkyi heti. Malliohjelma on jo valmiiksi erittäin hyvin dokumentoitu, eikä se sisältänyt UWP-mallin lisäksi muita vaikeita konsepteja, joten C++-perusteiden osaaminen valittiin ainoaksi aikaisemmin tarvittavaksi taidoksi. Kohdeyleisöksi määriteltiin siis C++-ohjelmoijat, jotka etsivät uusia haasteita.

Koska oletuksena oli vain C++-perusteiden osaaminen, käyttöohjeeseen sisällytettiin heti monisäikeistykseen käyttöä opettava kappale. Samalla käyttöohjeeseen suunniteltiin kappaleet malliohjelman selventämisestä ja Visual Studio-kehitysympäristön käyttämisestä. Pelimaailman valaistuksesta ja ruudulla näkyvän ku-

van jälkikäsitteystä kertovat kappaleet jätettiin pois. Niiden toteuttamiseen ei ole tullut uusia vaatimuksia, joten niihin aikaisemmin suunnatut käyttöohjeet ovat vieläkin toimivia.



Kuva 6. Käyttöohjeen lopputulo

Käyttöohjeiden lopputulokseksi määritettiin pelimaailma, jossa näkyy pyöriä peliobjekteja. Objektissa on tekstuurit ja niiden käyttämät 3D-malli on ladattu tietokoneen kiintolevyllä. (kuva 6)

Käyttöohjeiden tekemistä varten etsittiin ohjeita, jotka kertoisivat ohjelmointiohjeiden tekemisestä. Videopohjaisiin käyttöohjeisiin ja oppituntien tekemiseen löytyi ohjeita. Niistä ei ollut paljoa hyötyä, koska kaikille ei sovi videopohjaiset käyttöohjeet.

Hyvän ohjeen tekemiseen löytyi muutama neuvo. Ne toistuivat, oli ohjeessa kyse ruuanlaittamisesta, auton korjaamisesta tai videopohjaisten ohjelmointiohjeiden tuottamisesta. Ohjeiden pitää olla tarpeeksi selviä ja yksinkertaisia, että niistä voi käyttää sanontaa "väännetty rautalangasta". Niissä pitää myös olla sopivasti taustatietoa eri asioihin. Tietoa ei kuitenkaan saa olla liikaa, jolloin lukija voi huk-

kua tietoon, eikä liian vähän, jolloin ohje saattaa jäädä epäselväksi. Ohjeissa on myös suositeltavaa kertoa, miksi jokin asia neuvotaan tekemään tietyllä tavalla. Lisäksi ohjeiden pitää noudattaa selvää logiikkaa, jotta lukijan ei tarvitse siirtyä edestakaisin eri kohtien välillä.

Ohjelmointikäyttöohjeiden tekemistä varten kannattaa myös tutkia valmiita, vastaavasta asiasta kertovia, käyttöohjeita. Näin saadaan selvitettyä esimerkiksi kuinka usein lähdekoodia voidaan sijoittaa ohjeen joukkoon, ja kuinka pitkissä pätkissä koodia kannattaa olla. Pitkät koodipätkät saattavat jäädä liian epäselviksi lukijalle. Ohjelmointikäyttöohjeissa on suotavaa viitata käytettävissä oleviin virallisiin materiaaleihin.

Viittausten etsiminen Direct3D 12:n virallisiin materiaaleihin oli helppoa, koska Microsoft on kasvattanut D3D12:n virallisen dokumentaation määrää ja parantanut niiden laatua. Lisäksi D3D12:sta on keskusteltu useissa ohjelmoijille ja pelinkehittäjille suunnatuissa konferensseissa. Niistä oli helppoa löytää varmistuksia virallisessa dokumentaatioissa kerrottaviin asioihin.

Aiemmin suunniteltua käyttöohjetta ryhdyttiin jatkokehittämään. Käyttöohjeeseen lisättiin myös D3D12:n uusille ominaisuuksille omat kohdat. Yleisesti D3D12:sta kertova kappale sijoitettiin käyttöohjeen alkuun ja heti sen jälkeen Visual Studion asentamisesta kertova kappale. Kolmanneksi kappaleeksi valittiin Visual Studion käyttämisestä kertova kappale. Kappaleessa kerrottiin Visual Studion tärkeimmistä ominaisuuksista: projektien luomisesta, luodun projektin asetusten määrittämisestä ja käännettävän ohjelman ajamisesta ja virheidenetsinnästä. Neljännessä kappaleessa pilkottiin malliohjelma ensin tiedostotasolla ja sitten alakappaleissa kerrottiin funktioiden toteutumisjärjestyksessä niiden toiminta.

Malliohjelman pilkkominen järkevästi oli kuitenkin vaikeaa, koska ei löytynyt kovin montaa käyttöohjetta, jotka perustuvat laajaan malliohjelmaan. Lisäksi Microsoftin omat kommentit ja nimeämiskäytännöt olivat jo selkeitä. Joten ohjelman selittäminen tarkemmin, kokemattomammille kehittäjiä varten, oli vaikeaa.

Ensimmäinen malliohjelman kehittämiskappale kertoo olio-ohjelmoinnista ja peliobjektien sekä Pipeline State -objektien luomisesta. Pipeline State -objekteja varten luotiin wrapper-luokka, PipelineStateObject. Wrapper-luokan avulla tietyjä ase-

tuksia, joita ei yleensä pidä muuttaa, ei tarvitse kirjoittaa uudelleen. GameObject-luokalla voi luoda peliobjekteja, jotka sisältävät niiden käyttämän tekstuurin, 3D-mallin ja osoitteen peliobjektin käyttämään PipelineStateObject:iin. Peliobjektilla on myös Sample3DSceneRenderer:in mukaiset Update- ja Render-funktiot.

Toinen alakappale kertoo monisäikeisyydestä ja yksinkertaisen monisäikeistysjärjestelmän toteuttamisesta. Siinä tehdään kaksi työjonoa, yksi pelilogiikatoille ja toinen grafiikanpiirtotöille. Pelilogiikkajonossa käsitellään pelimaailmassa olevien objektien Update-funktiot, joiden lopussa aina kyseisen objektin Render-funktio asetetaan grafiikanpiirtojonolle. Järjestelmässä on prosessorisäikeiden verran ohjelmäsäikeitä, jotka on määritetty suorittamaan työjonoja. Puolet ohjelmäsäikeistä asettaa pelilogiikkajonon etusijalle, kuin loput taas asettaa grafiikanpiirtojonon etusijalle.

Kuvatiedostot, eli tekstuurit, ovat viidennen alakappaleen aihe. Siinä kerrotaan kuinka ReadAsync-funktiolla tiedostot luetaan kiintolevytä välimuistiin, josta stb-image-kirjasto käsittelee tekstuurit. ReadAsync-funktio on teoreettisesti stl- ja stb-image-kirjastojen tiedostonlukuja nopeampi, joten se valittiin käytettäväksi. stb-image-kirjasto valittiin tekstuureiden käsittelyyn, koska se osaa käsitellä useita kuvatiedostoformaatteja, on vapaasti saatavilla [21] ja osaa lukea tiedostot välimuistista käsiteltäväksi.

Kuvan lukeminen muistiin ei kuitenkaan riitä piirtämään sitä näytölle, vaan sen pitää käydä läpi ainakin verteksivarjostivaihe näytönohjaimen liukuhihnalta (kuva 2). Tämä vaatii myös uuden varjostimen tekemistä ja käyttöön asettamista, johon on varattu oma kappale käyttöohjeessa.

Tekstuureiden lukemisen jälkeen on järkevää lukea ja käsitellä 3D-mallit, joten niistä kertova kappale, viides alakappale on seuraavana ohjeessa. Tähänkin valittiin valmis, vapaasti saatavilla oleva kirjasto Assimp [22]. Assimp-kirjasto valittiin, koska se tukee useita 3D-mallinnusformaatteja ja siihen löytyy laaja virallinen dokumentaatio [22]. Assimp-kirjasto ei kuitenkaan tue mallien käsittelyä välimuistista, joten sen yhteydessä on käytetty kirjaston mukana tulevaa tiedostonlukua.

Lopetuskappaleessa kerrotaan teoriassa, miten ohjelmaa voidaan lähteä parantamaan. Ensimmäisenä kehoitetaan opiskelemaan monisäikeistyksestä enem-

män, ja lukemaan lisää varjostimista, muun muassa linkkaamalla Nvidian valaistusohjeeseen [23]. Seuraavana annetaan yksinkertainen esimerkki komentonippujen käyttämisestä. Toiseksi viimeisenä alakappaleena lopetuksessa on teoriaa monen näytönohjaimen käyttämisestä, jota ei ohjeen tekemisessä käytetyssä kehitysympäristössä voinut testata. Käyttöohje lopetetaan neuvomalla lukija lukemaan Jason Gregory'n kirja pelimoottoreiden arkkitehtuurista [24], koska se kertoo hyvin pelimoottoreista, joten lukijalla on mahdollisuus saada siitä selville mitä seuraavaksi kannattaa lähteä tekemään.

5 YHTEENVETO

Tämän opinnäytetyön tavoitteena oli toteuttaa käyttöohje Direct3D 12 -grafiikkarajapinnalle. Lopputuloksena saadulla käyttöohjeella [25] kuka tahansa, joka osaa C++-perusteet, pystyy toteuttamaan Direct3D 12 -sovelluksen. Ohjeen käytettävyyttä ei kuitenkaan ehditty testaamaan työaikataulussa. Työ toteutettiin hyvään aikaan, sillä saman kohdeyleisön kanssa on kilpailemassa vain yksi muu valmiiksi tehty käyttöohje.

Henkilökohtaisena tavoitteena oli oppia käyttämään uutta matalamman tason grafiikkarajapintaa ja tekemään ohjelmointikäyttöohjeita. Vaikka mielestäni opin molemmista perusteet hyvin, toivon saavani niistä vielä enemmän kokemusta. Työaikana huomasin erityisesti, kuinka suuresti Direct3D 12 eroaa aikaisemmin käyttämästäni OpenGL-grafiikkarajapinnasta. Toinen merkittävä huomio oli, kuinka huonosti uudesta rajapinnasta löytyy tietoa.

Työn koon olin mielestäni valinnut alussa hyvin, sillä silloin suunniteltu aikataulu tuntui sopivalta. Ikävä kyllä lomien aikana ja jälkeen, työn raportoinnin kirjoittamisen aloittaminen oli odotettua hankalampaa. Tämän takia dokumentointiaikataulu tuntui tiukalta.

Työtä voisi jatkaa vielä tekemällä saman ohjelman vanhemmalla Direct3D:n versiolla, OpenGL:llä ja/tai Vulkanilla. Sitten voi selvittää, miten D3D12 pärjää näyttöohjaimen tehojen käyttämisessä muita grafiikkarajapintoja vastaan. Lisäksi työtä voisi jatkaa lisäämällä siihen mukaan loppuja DirectX-paketin mukana tulevia rajapintoja.

LÄHTEET

WWW-julkaisuihin viitattu 27.4.2016.

- 1 Microsoft: Direct3D Overview. [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx)>
- 2 Khronos Organization: OpenGL Overview [WWW-julkaisu]
<<https://www.khronos.org/opengl/>>
- 3 Khronos Organization: Vulkan Overview [WWW-julkaisu]
<<https://www.khronos.org/vulkan/>>
- 4 Intel: Moore's Law [WWW-julkaisu]
<<http://www.intel.com/content/www/us/en/history/museum-gordon-moore-law.html>>
- 5 Computer Hope: Computer processor history [WWW-julkaisu]
<<http://www.computerhope.com/history/processor.htm>>
- 6 Loyd Case: Building the Ultimate Game PC. Que 1999.
ISBN: 978-0-7897-2204-1
- 7 Microsoft: Direct3D Graphics [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/bb153256\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb153256(v=vs.85).aspx)>
- 8 Microsoft: Graphics Pipeline [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882(v=vs.85).aspx)>
- 9 AMD: One of Mantle's Futures: Vulkan [WWW-julkaisu]
<<https://community.amd.com/community/gaming/blog/2015/05/12/one-of-mantles-futures-vulkan>>
- 10 Microsoft: DirectX Software Development Kit [WWW-julkaisu]
<<https://www.microsoft.com/en-us/download/details.aspx?id=9977>>

- 11 Microsoft: Direct3D Feature Levels [WWW-julkaisu]
<<https://blogs.msdn.microsoft.com/chuckw/2012/06/20/direct3d-feature-levels/>>
- 12 Microsoft: What is Direct3D 12? [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/dn899228\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dn899228(v=vs.85).aspx)>
- 13 WccfTech: DX12 Gains Will Take Time [WWW-julkaisu]
<<http://wccfttech.com/hitman-lead-dev-dx12-gains-time-ditching-dx11/>>
- 14 Microsoft: Command List [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/ff476885\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff476885(v=vs.85).aspx)>
- 14 Nvidia: DX12 Do's And Don'ts [WWW-julkaisu]
<<https://developer.nvidia.com/dx12-dos-and-donts>>
- 15 Microsoft: Command Queues and Command Lists [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/dn899114\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dn899114(v=vs.85).aspx)>
- 16 WccfTech: Async Compute [WWW-julkaisu]
<<http://wccfttech.com/async-compute-boosted-hitmans-performance-510-amd-cards-devs-super-hard-tune/>>
- 17 AMD: AMD Simplified: Asynchronous Shaders [WWW-julkaisu]
<<https://www.youtube.com/watch?v=v3dUhep0rBs>>
- 18 Microsoft: Managing Graphics Pipeline State [WWW-julkaisu]
<[https://msdn.microsoft.com/en-us/library/windows/desktop/dn899196\(v=vs.85\).aspx#pipeline_state_overview](https://msdn.microsoft.com/en-us/library/windows/desktop/dn899196(v=vs.85).aspx#pipeline_state_overview)>
- 19 Intel: Direct3D 12 Overview Part 2: Pipeline State Object [WWW-julkaisu]
<<https://software.intel.com/en-us/blogs/2014/07/23/direct3d-12-overview-part-2-pipeline-state-object>>

- 20 Overclock3D: AMD Explains DX12 Multi GPU Benefits [WWW-julkaisu]
<[http://www.overclock3d.net/articles/gpu_displays/
amd_explains_dx12_multi_gpu_benefits/1](http://www.overclock3d.net/articles/gpu_displays/amd_explains_dx12_multi_gpu_benefits/1)>
- 21 Nothings: stb-library [WWW-julkaisu]
<<https://github.com/nothings/stb>>
- 22 Assimp: Open Asset Import Library [WWW-julkaisu]
<<http://www.assimp.org>>
- 23 Nvidia: Cg Tutorial Chapter 5. Lighting [WWW-julkaisu]
<http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter05.html>
- 24 Jason Gregory: Game Engine Architecture, Second Edition. CRC Press
2014. ISBN: 978-1466560017
- 25 Juho Saarinen: Direct3D 12 Tutorial [WWW-julkaisu]
<<http://www.zeitt.net/d3d12tutorial>>

LIITTEET

Liite 1. SampleVertexShader.hlsl

Liite 2. SamplePixelShader.hlsl

Liite 3. Malliohjelman Luokkakaaviot

LIITE 1. SAMPLEVERTEXSHADER.HLSL

```
// A constant buffer that stores the three basic column-major matrices for composing geometry.
cbuffer ModelViewProjectionConstantBuffer : register(b0)
{
    matrix model;
    matrix view;
    matrix projection;
};

// Per-vertex data used as input to the vertex shader.
struct VertexShaderInput
{
    float3 pos : POSITION;
    float3 color : COLOR0;
};

// Per-pixel color data passed through the pixel shader.
struct PixelShaderInput
{
    float4 pos : SV_POSITION;
    float3 color : COLOR0;
};

// Simple shader to do vertex processing on the GPU.
PixelShaderInput main(VertexShaderInput input)
{
    PixelShaderInput output;
    float4 pos = float4(input.pos, 1.0f);

    // Transform the vertex position into projected space.
    pos = mul(pos, model);
    pos = mul(pos, view);
    pos = mul(pos, projection);
    output.pos = pos;
```

```
// Pass the color through without modification.  
output.color = input.color;  
  
return output;  
}
```

LIITE 2. SAMPLEPIXELSHADER.HLSL

```
// Per-pixel color data passed through the pixel shader.
struct PixelShaderInput
{
    float4 pos : SV_POSITION;
    float3 color : COLOR0;
};

// A pass-through function for the (interpolated) color data.
float4 main(PixelShaderInput input) : SV_TARGET
{
    return float4(input.color, 1.0f);
}
```

