Ohjelmoitavat sävyttimet

Janne Timonen

Seminaaritutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 16. joulukuuta 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Janne Timonen

Työn nimi — Arbetets titel — Title

Ohjelmoitavat sävyttimet

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminaaritutkielma	16. joulukuuta 2015	13

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Sävyttimet ovat ohjelmia, joiden tehtävänä *grafiikkaliukuhihnalla* (asteittain etenevä prosessi tuottaa kuvia näytettäväksi) on *sävyttää* eli tuottaa tietyillä tavoilla dataa kuvaksi. Tämä voi tarkoittaa esimerkiksi jonkin objektin piirtämistä sijainnin mukaan, pikselikohtaista värinmääritystä, pinnanmuotojen simulointia tai muita erikoistehostemaisiakin keinoja.

Sävytin ottaa syötteenään elementin, kuten esimerkiksi monikulmion kärkipisteen, kokonaisen monikulmion tai yksittäisen pikselin. Saadusta syötteestä sävytin tuottaa tuloksena muunnettuja elementtejä, joiden määrä voi vaihdella nollasta useaan, riippuen sävyttimestä ja sen suorittamasta tehtävästä [Gre14, s. 500]. Sävyttimien käyttö mahdollistaa myös hyvin rinnakkaisuuden käytön, kun muunnoksia tehdään suurille datamäärille kerrallaan, esimerkiksi kaikille ruudun pikseleille. Moderneilla grafiikkapiireillä onkin useita sävytinliukuhihnoja rinnakkaisuusmahdollisuuksien hyödyntämiseksi.

Tutkielmassa tarkastellaan aluksi hieman yleisesti 3D-grafiikkaa, jotta sävyttimien roolia grafiikan tuottamisessa voi ymmärtää paremmin ja hahmottaa niiden tehtävää, minkä jälkeen käydään läpi sävyttimien historian päävaiheet, ja kuinka nykyisiin ohjelmoitaviin sävyttimiin on päädytty. Tämän jälkeen siirrytään asian varsinaiseen ytimeen eli ohjelmoitaviin sävyttimiin, ja käydään vaiheittain läpi tällä hetkellä käytettävien sävyttimien toimintaa, mitä niillä on mahdollista tehdä ja kuinka se tapahtuu. Lopuksi tarkastellaan hieman tarkemmin kuinka sävytin voidaan kääntää ja sitoa mukaan grafiikan tuottoon.

2 3D-grafiikka

3D-grafiikassa tuotetaan kolmiulotteisista malleista kaksiulotteinen representaatio eli esimerkiksi katsojan näkemä kuva tietokoneen ruudulla. Erityisesti tämän tutkielman tapauksessa 3D-grafiikan reaaliaikaisen piirron menetelmänä käsitellään *rasterointia* eli kuvan muuttamista pikseleillä esitettävään muotoon niin kutsutuksi rasterikuvaksi. Rasteroinnissa kolmiulotteiset mallit on projisoitu ja leikattu kuvaan halutulla tavalla, minkä jälkeen kuvasta piirretään kaksiulotteinen pikseleistä koostuva kuva. Tarkkaa reaaliaikaisuutta vaativien grafiikkasovellusten, kuten pelien, tapauksessa nopea kuvan piirtäminen nousee tärkeäksi vaatimukseksi, jolloin monikulmioista eli *polygoneista* tuotetaan rasteroimalla kaksiulotteista kuvaa. Tavoitteena on saavuttaa ruudunpäivitysnopeus, joka vaikuttaa ihmisen silmään sulavalta, eli ainakin noin 30 ruutua sekunnissa. [Gre14, s. 444-445].

Ei-reaaliaikaisiin 3D-grafiikan renderointitapoihin lukeutuu esimerkiksi *säteenjäljitys* (Ray tracing), jossa valaistus ja valon heijastukset lasketaan simuloiden tarkemmin todellista maailmaa. Esimerkiksi voidaan laskea pinnalta heijastuneen valon heijastuminen uudestaan joltain toiselta pinnalta.

[Puh08, s.405-]. Tällainen renderointi on hidasta, eikä vielä tällä hetkellä ole järkevästi hyödynnettävissä reaaliaikaisessa käytössä, kuten peleissä, mutta ennakkoon renderoituna tuottaa lähes fotorealistista kuvaa. Säteenjäljitykseen tai vastaaviin tekniikoihin ei tässä tutkielmassa enää palata.

Sävyttimien ymmärtämisen kannalta on hyödyllistä ymmärtää myös erilaiset 3D-grafiikkaan liittyvät *koordinaatistot*, joissa 3D-malleja käsitellään. Näihin lukeutuvat muun muassa *mallikoordinaatisto* (voidaan puhua myös koordinaatiston sijaan avaruudesta, kuten englanniksi *model space*), jossa ei kyseisen 3D-mallin lisäksi ole mitään muuta, ja *maailmakoordinaatisto*, jossa esiintyy useita eri malleja. LISÄÄ!!!

2.1 Grafiikkaliukuhihna

Rasterointiin tähtäävässä 3D-grafiikassa hyödynnetään niin kutsuttua liukuhihnaa (pipeline), joka on pääasiassa sarja tietyssä järjestyksessä tehtäviä askeleita tai työvaiheita, joilla 3D-malleista luodaan 2D-rasterikuva. Liukuhihna ja erityisesti *grafiikkaliukuhihna* on siis prosessi, jolla lopullinen kuva tuotetaan. Peligrafiikan tuottamiseen rasteroinnin keinoilla suosituimmat grafiikkaohjelmointirajapinnat ovat OpenGL ja Direct3D, jotka kummatkin toimivat saman grafiikkaliukuhihna-ajattelun mukaisesti.

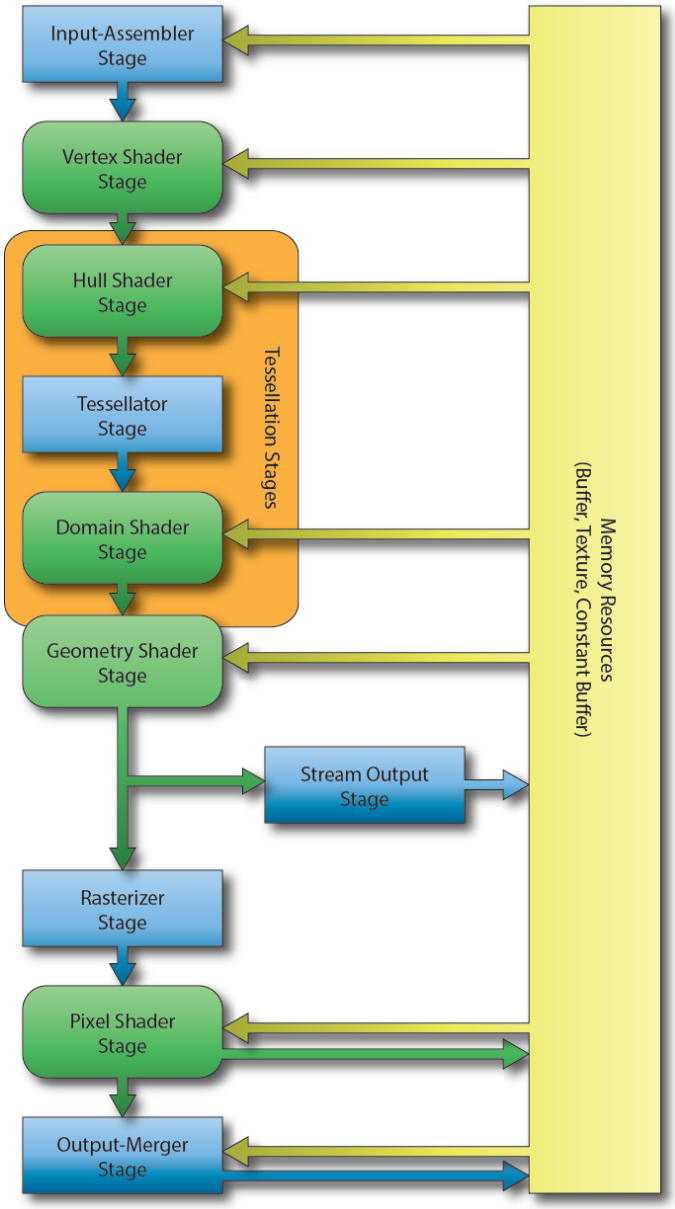
Valmistajien grafiikkapiirien tukemia sävyttimiä eritellään *Shader Model* -versioinnin avulla, joka kertoo millaista grafiikkaliukuhihnaa ja mitä sävyttimiä jokin laite tukee. Tutkielman tarkastelema grafiikkaliukuhihna on Shader Model 5.0:n mukainen, jolloin voidaan esitellä kaikki nykyään peleissä mahdollisesti käytössä olevat sävyttimet. Grafiikkaliukuhihna koostuu kärkipiste-, tesselaatio-, geometria- ja pikselisävyttimestä. Kärkipiste- ja pikselisävytin ovat

Grafiikkaliukuhihnaan kuuluu myös joitain kiinteitä vaiheita, jotka eivät varsinaisesti ole sävyttimiä tai ainakaan ohjelmoitavia sellaisia. Näihin vaiheisiin ei myöhemmin paneuduta lukuun ottamatta tesselaatiosävyttimen kiinteää vaihetta. Liukuhihnan vaiheet ja järjestys ovat esitetty kuvassa 1 [Mic11].

3 Sävyttimien historiaa

3.1 Kiinteä liukuhihna

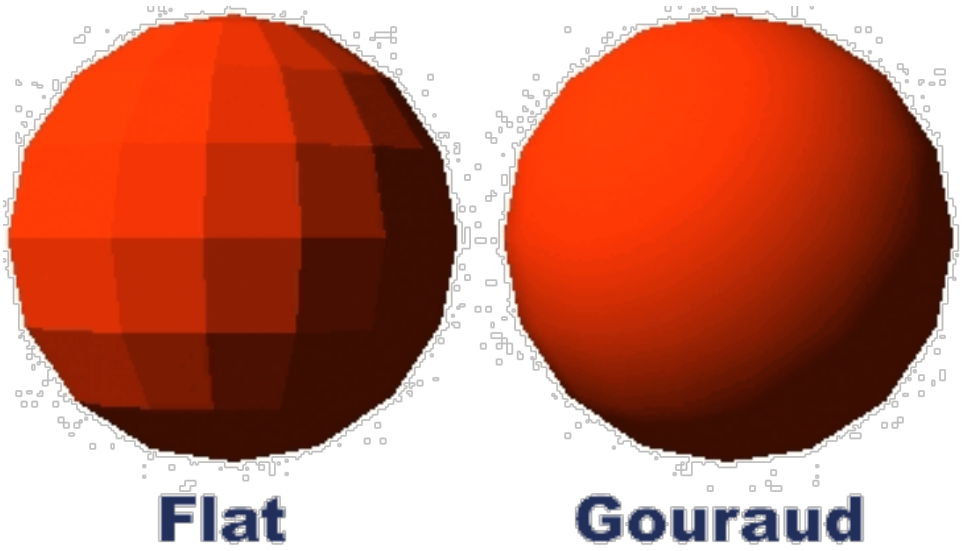
Ennen ensimmäisiä *grafiikkakiihdyttimiä*, joihin kuului muun muassa 3Dfx:n Voodoo, grafiikan piirto tapahtui prosessorilla, jonka työtaakkaa erilliset grafiikkakiihdyttimet kehitettiin vähentämään. Vapaasti ohjelmoitavia sävyttimiä ei vielä tuolloin kuitenkaan ollut, ja grafiikkaa tuotettiin käyttämällä hyväksi grafiikkapiirien *kiinteää liukuhihnaa* (Fixed-Function Pipeline). Kehittäjä saattoi siis antaa raskaat laskutyöt grafiikkakiihdyttimelle hoidettavaksi



Kuva 1: Grafiikkaliukuhihna

käyttäen kiinteää liukuhihnaa, mutta itse liukuhihnan suorittamiin funktioihin ei voinut puuttua muuten kuin parametrien avulla. Kiinteä liukuhihna näytönohjaimessa nopeutti laskentaa ja toi mukanaan mahdollisuuksia luoda standardioperaatioiden rajoissa graafisia tehokeinoja ja efektejä, esimerkiksi Goraud-sävytyksen, josta on esimerkki kuvassa 2. Verrattuna monikulmion tasaiseen väritykseen Gouraud-sävytyksessä kärkipisteiden väriarvot interpoloidaan monikulmion sisällä, jolloin saadaan tasaisempi ja luonnollisemman

näköinen heijastus.



Kuva 2: Tasainen ja Gouraud-sävytys

Myöhemmin tulivat ensimmäiset *ohjelmoitavaa grafiikkaliukuhihnaa* tukevat näytönohjainpiirit, joissa kiinteän liukuhihnan pystyi korvaamaan omilla vapaasti ohjelmoitavilla sävyttimillä. Korvaamalla kiinteän liukuhihnan laskenta nykyajan grafiikkapiirien tukemilla ohjelmoitavilla sävyttimillä voidaan vapaasti muokata sävyttimien käyttäytymistä ja saavutetaan mahdollisuus piirtää kuvaa enemminkin luovuuden rajoissa kuin ennaltamääriteltyjen ehtojen.

3.2 Alemman tason kielinen sävytinohjelmointi

Ensimmäiset ohjelmoitavan liukuhihnan sävytinmallit tukivat ainoastaan alemman tason konekieliin pohjautuvilla kielillä ohjelmointia. Tällaiset kielet lainasivat periaatteensa prosessoriarkkitehtuurista, jossa konekielen käskykanta on yksinkertainen ja rajoitettu. Sen lisäksi, että kehittäjien täytyi luoda sävytin, täytyi se luoda lisäksi usein erikseen sekä OpenGL- että Direct3D-rajapinnoille johtuen näiden kahden suosituimman rajapinnan kielien poikkeavuuksista . Aluksi laitteet tukivat ainoastaan kärkipiste- ja pikselisävytintä, laitekohtaisesti jopa vain ensimmäistä näistä. Silloiset rajoitukset konekäskyjen määrässä ja rekistereiden koossa tekivät silti vielä joistain asioista haasteellisia toteuttaa. Esimerkiksi pikselikohtainen valaistus, jonka nykyään voi ajatella olevan suoraviivaisesti toteutettavissa, oli tuolloin hankala toteuttaa laitteiston rajoituksista johtuen ja vaati kehittäjiltä erilaisia keinoja toimiakseen reaaliajassa [She08, s. 174-176].

Eräs edelläkävijöitä sävyttimien saralla oli tietokoneanimaatioelokuvis-

taan tunnettu Pixar-yhtiö jo 1980-luvun lopussa kehittämällään *RenderMan*-kielellä, jota on käytetty satojen elokuvien visuaalisiin efekteihin ja animaatioelokuvien piirtämiseen [Pix15].

3.3 Korkean tason sävytinkielet

Ohjelmoitavien sävyttimien alkuaikojen jälkeen alettiin sävyttimien luomisessa siirtyä alemman tason kielistä erillisiin korkean tason sävytinkieliin. Alemman tason kieliin verrattuna korkean tason kielillä on useita hyötyjä, ja ne pätevät myös korkean tason sävytinohjelmoinnissa: helpompi luettavuus, kirjoitettavuus, muokattavuus, virheiden etsintä ja löytäminen sekä yleisesti kehitysvauhdin nopeus [She08, s.183-185]. Ohjelmoitavien sävyttimien tultua kasvoi myös tarve korkean tason sävytinkielille, joista mainittavimpina muodostuivat C-pohjaiset Nvidian Cg- (C for Graphics), [Nvi03] Microsoftin HLSL- (High Level Shading Language) ja OpenGL:n [Khr15] GLSL. Cg ja HLSL syntyivät Microsoftin ja Nvidian yhteistyöprojektin tuloksena, ja ovatkin lähes identtiset [She08, s. 198]. Näistä kielistä Cg on jo vanhentunut, eikä sitä enää päivitetä [Nvi12]. HLSL ja GLSL ovat monin osin samankaltaisia, ja ne tukevat samanlaisia sisäänrakennettuja funktioita. Eroina on muun muassa joidenkin samojen datatyyppien poikkeavat nimeämiskäytännöt. Esimerkiksi neljän komponentin vektori on GLSL-kielellä ilmaistuna `vec4`, kun taas HLSL-kielellä se on `float4` [She08, s. 198]. Tässä tutkielmassa esimerkkikielenä käytetään pääasiassa Direct3D-rajapinnalle tarkoitettua HLSL:ää.

4 Ohjelmoitavat sävyttimet

Ohjelmoitavien sävyttimien osalta liukuhihna rakentuu tällä hetkellä pääpiirteissään *kärkipiste-*, *tesselaatio*, *geometria-* ja *pikseli*sävyttimestä. Yksi sävytinvaihe ottaa syötteenään vastaan edellisen tulosteen, joten jokainen vaihe voi jatkaa seuraavan datan työstämistä, kun on saanut edellisen työn valmiiksi. Tämä grafiikkaliukuhihnan malli mahdollistaa sävytinprosessoinnin rinnakkaistamisen erittäin hyvin [Ake02].

Tesselaatiosävytin ja geometriasävytin ovat uudempia tulokkaita, ja niiden käyttö ei ole pakollista, jolloin niiden toiminnallisuus voidaan jättää pois tai korvata muilla tavoin. Esimerkiksi ennen tesselaatiolle omistettua omaa sävytintä tesselaatiosävytys toteutettiin geometriasävyttimen avuin [Sch14]. Tässä luvussa tarkastellaan erilaisia ohjelmoitavia sävyttimiä siinä järjestyksessä, jossa ne toimivat grafiikkaliukuhihnalla.

4.1 Kärkipistesävytin

Kärkipistesävytin, tai *verteksisävytin*, ajetaan kerran jokaista monikulmion, tarkemmin kolmion, kärkipistettä kohden. Kärkipistesävytin ottaa syöttee-

nään kärkipisteen *attribuuttitiedon*, joka sisältää muun muassa kyseisen kärkipisteen sijainnin malli- tai maailmakoordinaatistossa, sekä pinnan normaalivektorin. Tulosteena kärkipistesävytin antaa yhden kärkipisteen, joka on käynyt läpi valaistuksen ja koordinaatiston muunnosvaiheet, ja joka ilmaistaan *normalisoidussa kuvausavaruudessa*. Yleisesti siis kärkipistesävytin voi muokata monikulmion kärkipisteen, normaalin, tekstuurikoordinaattien ja paikan arvoja. Sävyttimellä voi luoda esimerkiksi tuulessa heiluvat puiden oksat tai vedenpinnan väreilyn.

Kärkipistesävyttimen tärkeimpiin tehtäviin kuuluu tehdä tarvittavat koordinaatistomuunnokset syötteinä saaduille kärkipisteille. Ensimmäisenä tehdään *mallimuunnos*, eli muunnos mallikoordinaatistosta maailmakoordinaatistoon, jolloin kärkipiste sidotaan omasta paikallisesta koordinaatistostaan absoluuttisesti maailmanäkymään. Tämän jälkeen tehdään *katsojamuunnos* maailmakoordinaatistosta katsojan koordinaatistoon, jolloin origo on katsotaan "kameran" näkökulmasta. Viimeiseksi suoritetaan *projektio-* tai *perspektiivimuunnos*, jolloin tuotetaan renessanssiajan kuvataiteesta tuttu luonnollisen elämän persepektiiviä jäljittelevä kuva, jossa kauempana olevat kohteet näkyvät pienempinä katsojan *näköfrustumissa*. Tätä tilaa kutsutaan normalisoiduksi kuva-avaruudeksi [Puh08].Vähintään kärkipistesävyttimen tulee siis antaa tuloksena kärkipiste muunnettuna normalisoituun kuva-avaruuteen *uniformina* eli vakiomuotoisena tietona [Puh08].

Kärkipistesävytin on pakollinen vaihe grafiikkaliukuhihnalla, ja sen täytyy vähintään kuljettaa lävitseen syötteenä saatu kärkipiste, vaikkei laskusuorituksia tehtäisikään. Alla HLSL-kielellä kirjoitettu yksinkertainen kärkipistesävytin syötteineen ja tuloksineen [Mic11]:

```
// Input / Output structures

struct VS_INPUT
{
    float4 vPosition    : POSITION;
    float3 vNormal      : NORMAL;
    float2 vTexcoord    : TEXCOORD0;
};

struct VS_OUTPUT
{
    float3 vNormal      : NORMAL;
    float2 vTexcoord    : TEXCOORD0;
    float4 vPosition    : SV_POSITION;
};

// Vertex Shader

VS_OUTPUT VSMain( VS_INPUT Input )
{
    VS_OUTPUT Output;

    Output.vPosition = mul( Input.vPosition,
```

```
        g_mWorldViewProjection );
    Output.vNormal = mul( Input.vNormal, (float3x3)g_mWorld );
    Output.vTexcoord = Input.vTexcoord;

    return Output;
}
```

Syöte- ja tulos-structeissa on määritelty vektoreina (`float`n) kärkipisteen sijainti, normaali ja tekstuurikoordinaatti. `VSMain`-metodi ottaa syötteenään kyseiset arvot ja tekee sijainnille sekä normaalille matriisimuunnokset. Sijainnille tehdään maailma-katsojamuunnos katsojan koordinaatistoon ja normaalille maailmamuunnos. Tekstuurikoordinaatti kulkee esimerkissä sävyttimen läpi muuttumatta. Vaikka esimerkki on yksinkertainen, noudattelee sävyttimien ohjelmointi samaa logiikkaa grafiikkaliukuhihnan kaikissa vaiheissa.

4.2 Tesselaatiosävytin

Tutkielman sävyttimistä uusin on Shader Model 5.0:n, rajapintojen kohdalla DirectX 11 ja OpenGL 4.0, myötä tullut tesselaatiosävytin. *Tesselaatiossa* primitiivi, *monikulmioverkko* (polygon mesh), eli kärkipisteistä ja reunaviivoista kohtaava kolmioiden joukko[Puh08], jaetaan pienempiin osasiin, kuten vaikkapa kolmio kahteen pienempään kolmioon [Nvi10]. Yksinkertaisesti siis tesselaatio on monikulmioiden rikkomista ja jakamista pienempiin ja hienompiin osasiin.

Tesselaatiosävytin on jaettu kolmeen vaiheeseen, joista ensimmäinen ja kolmas ovat ohjelmoitavia sävytinvaiheita. Vaiheet alkaen ensimmäisestä ovat *Hull Shader*, varsinaisen työn tekevä kiinteä tesselaatiovaihe *Tessellation Stage* ja *Domain Shader* [Mic11]. OpenGL-terminologiassa vastaavat vaiheiden nimet ovat Control, Primitive Generator ja Evaluation. Tesselaatiossa ensimmäisen sävytinvaiheen tehtävä on määrittää paljonko syötettä tesseloidaan. Kiinteä vaihe hoitaa tämän jälkeen laskutyön saamansa syötteen perusteella, minkä jälkeen kolmannen vaiheen sävytin työstää tesseloidun datan, ja määrittää sen kärkipisteiden sijainnit. Käytännössä tesselaation kolmas vaihe toimii kuten tavallinen kärkipistesävytin.

Menetelmänä tessellointi ei välttämättä tuo suoraan ulkonäöllisiä mullistuksia esimerkiksi pelien ulkonäköön, sillä ulkoasun kannalta ei ole merkitystä onko esimerkiksi neliö piirretty kahden vai kymmenien kolmioiden avulla. Sen sijaan yhdistämällä tesselaatioon muita tekniikoita, ja laittamalla monikulmioista pilkotut palaset esittämään uutta informaatiota, saadaan piirtoa monimutkaisemmaksi ja realistisemmaksi [Nvi10]. Esimerkiksi monimutkaisen geometrian luominen lennosta on ennen tesselaatiosävytintä ollut vaikeata, kun työ on tehty ensin prosessorilla ja ladattu sen jälkeen grafiikkapiirille. Shader Model 5 -version myötä grafiikkapiirien tukiessa tesselaatiota laitetasolla voidaan tesselointi tehdä tehokkaasti [Sch14].

Kuva 3: Nyrjäytyskartan realistisemmat pintaerot

Eräs tesseloinnin tekniikka on *nyrjäyttäminen* (Displacement mapping), jossa tesseloidun pinnan kärkipisteitä nostetaan tai lasketaan korkeusattribuutin perusteella, jolloin saadaan luotua epätasaisia pintoja [Nvi10]. Nyrjäytystekniikkaa havainnollistaa kuva 3. Tessellointi säästää myös muistia ja kaistanleveyttä mahdollistamalla yksityiskohtaiset pinnat pieniresoluutioisilla tekstuureilla [Mic11] [Nvi10].

Tesselaation avulla saavutettava keino on myös 3D-mallien silottaminen normaalipiste-kolmioiden (PN-triangle) avulla [Vla01]. Esimerkki kuvassa 4 näkyvän Stalker: Call of Pripyat -pelin hahmon kaasunaamarin suodattimen reunoja on saatu tesselaatiolla luonnollisemmalla tavalla kaartuviksi (DX11) verrattuna yksinkertaisempaan ja vähäisemmillä monikulmioilla piirrettyyn malliin (DX10).

Dynaamisella tesselaatiolla voidaan esimerkiksi skaalata mallien piirron tarkkuutta näkyvyyden suhteen muuttamalla yksityiskohtien määrää lennosta [Nvi10]. Tällöin esimerkiksi avarassa ulkoilmapelinäkymässä kaukaa katsottuna jostain mallista piirretään malli muutamalla monikulmiolla, ja lähestyttäessä monikulmioiden määrää lisätään dynaamisesti, kunnes läheltä katsottuna malli voidaan piirtää tuhansista monikulmioista [Gre14].

4.3 Geometriasävytin

Geometriasävyttimet ovat kärkipiste- ja pikselisävyttimiin verrattuina uudempi tekniikka sen tultua esitellyksi Shader Model 4.0:n ja DirectX 10:n myötä. Geometriasävytin sijaitsee grafiikkaliukuhihnalla tesselaatiosävyttimen jälkeen ja ennen pikselisävytintä. Geometriasävytin on vaihtoehtoinen osa liukuhihnaa. Geometriasävyttimellä tyypillisesti luotuihin asioihin kuuluvat esimerkiksi *partikkelijärjestelmät* ja *mainostaulut*. Ennen tesselaatiosävyttimen tuloa geometriaävyttimellä hoidettavia tyypillisiä käyttötapauk-

Kuva 4: Monikulmiomallin silottaminen tesselaation avulla pelissä Stalker: Call of Pripyat

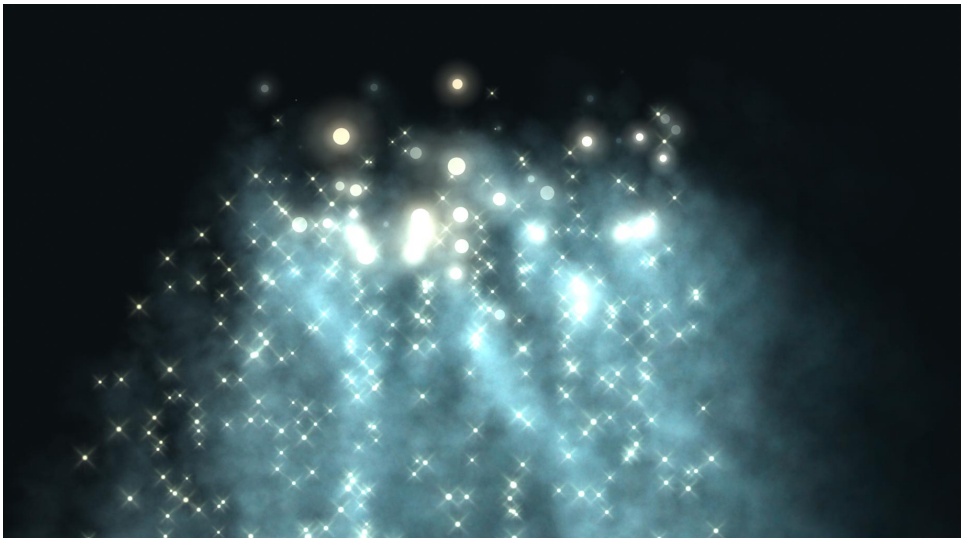
sia oli esimerkiksi aiemmin esitelty dynaaminen tesselaatio [Mic11]. Vaikka geometriasävyttimellä on mahdollista tehdä yksinkertaista tesselaatiota, se muodostuu grafiikkaliukuhihnalle helposti pullonkaulaksi [Sch14].

Geometriasävytin ottaa syötteenään n-kärkipisteestä muodostuvia primitiiveja, kuten pisteitä (n=1), suoria (n=2) tai kolmioita (n=3). Syötteistä geometriasävytin muokkaa, valikoi ja jopa luo uusia primitiivejä [Gre14]. Tuloksena voi siis olla nollasta useampaan primitiiviä, jotka eivät välttämättä ole samaa tyyppiä kuin syötteenä saadut. Geometriasävytin voi esimerkiksi yhdistellä kolmioita uudeksi monikulmioksi, tai hylätä kolmioita kokonaan. Toisin kuin kärkipistesävytin, joka kykenee käsittelemään vain yhden monikulmion kärkipisteen kerrallaan, pystyy geometriasävytin "näkemään" koko primitiivin kaikkine kärkipisteineen.

4.4 Pikselisävytin

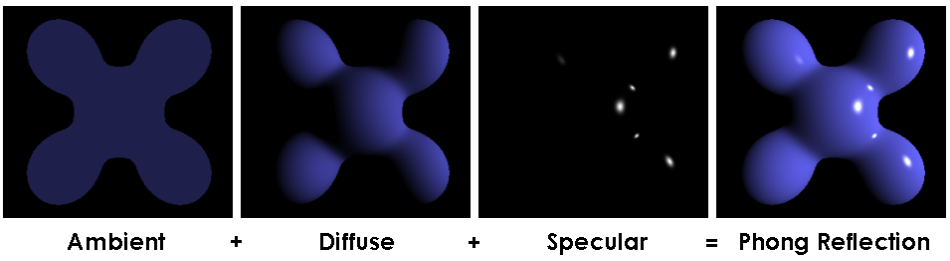
Pikselisävytin (OpenGL-terminologiassa fragmenttisävytin, on graafinen funktio, joka laskee muunnoksia pikselikohtaisesti. Sävyttimen voidaan tehdä jokaiselle yksittäiselle pikselille erikseen []. Pikselisävytin ajetaan kerran yhtä pikseliä kohden, ja useita kertoja jokaista syötteenä saatua monikulmiota kohden, sillä sävytin käsittelee jokaista monikulmion pikseliä erikseen. Pikselin väriarvo sekä Z-syvyys lasketaan syötteenä saatun vektorimuotoisen datan, kuten normaalivektorin, värin, tekstuurikoordinaattien, interpoloitujen valonlähteiden suuntien ja katsojan suunnan, perusteella. Erityisesti pikseliin kohdistuva valaistuksen laskenta voidaan johtaa edellisistä [Puh08].

Monet näyttävät 3d-peleissä käytettävät tehostekeinot, kuten pinnan kuh-



Kuva 5: Partikkelijärjestelmä ja mainostauluttaminen

mutus tai *Fresnel-heijastus* [Laz05], luodaan juuri pikselisävyttimien tasolla. Myös Gouraud-sävytystä realistisempi Phong sävytys luodaan pikselisävyttimellä. Phong-sävytyksen eri komponentit ovat esitelty kuvassa 6. Pikselisävyttimen tärkeimpiin tehtäviin lukeutuvatkin teksturointi ja valaistuksen laskenta.

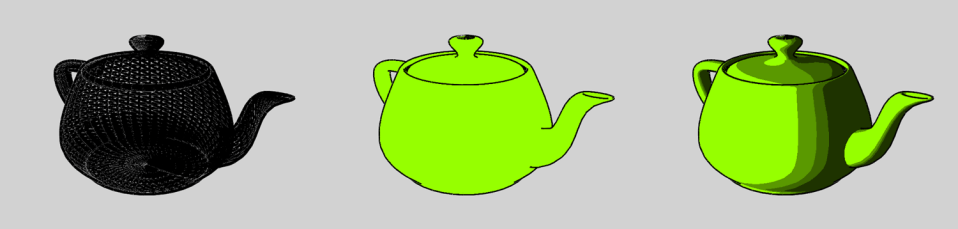


Kuva 6: Phong-sävytys

Eräs tietokonepeleissäkin hyödynnetty keino on pikselisävyttimellä luotava *Cel-sävytys*. Cel-sävytyksessä ... . Valaistuksessa pikseleiden väriarvot lasketaan katsojan suunnan ja pinnan normaalivektorin avulla siten, että väriarvo muuttuu portaittain sitä tummemmaksi mitä enemmän pinta osoittaa pois katsojasta. Cel-sävytyksestä on esimerkki kuvassa 7.

4.5 Laskentasävytin

Shader Model 5.0 toi myös tuen *laskentasävyttimelle* (englanniksi Compute Shader). Laskentasävyttimen on enemmän yleiskäyttöinen ja löyhästi määri-



Kuva 7: Cel-sävytys

telty sävytin kuin muut tutkielmassa esitellyt sävyttimet. Laskentasävytin ei varsinaisesti kuulu grafiikkaliukuhihnalle, eikä sen syötteitä ja tuloksia ole samalla tavoin tarkasti määritelty kuin muilla sävyttimillä. Laskentasävytin voi siis ottaa syötteenään muutakin dataa kuin grafiikkalaskennassa käytettyjä objekteja kuten kolmioita. Eräs käyttötarkoitus laskentasävyttimelle onkin *GPGPU*, eli yleiskäyttöinen laskenta, jolloin näytönohjainten rinnakkaisuutta hyödynnetään erilaisten raskaiden laskutöiden suorituksessa [Uni15]. Tietokonepelien kohdalla laskentasävytintä voidaan hyödyntää esimerkiksi fysiikan mallinnuksen tai tekoälyn toiminnan laskemisessa [Mic11, Luku: Compute Shader Overview]. Laskentasävyttimille on olemassa omia ohjelmointirajapintoja, joihin kuuluvat etenkin Nvidian CUDA, Kronos Groupin OpenCL sekä Microsoftin DirectCompute.

5 Sävytinohjelman luominen ja käyttäminen

Sävyttimen luominen. Esimerkiksi käyttäen hyväksi HLSL-kieltä tapahtuisi sävyttäjän luomisprosessi näin...HLSL:n tapauksessa sävyttimen voi ladata joko esikäännettynä tai ajoaikaisesti tavutauluna.

Kirjoitettu .hlsl -sävytintiedosto tulee ensin kääntää sävytinobjektiksi, eli HLSL:n tapauksessa .cso-tiedostoksi (complied shader object). Kääntämiskutsun voi HLSL:n tapauksessa tehdä suoraan Microsoftin Visual Studio -kehitysympäristössä käyttäen fxc.exe HLSL-kääntäjää. Itse kääntökutsu on pääpiirteissään seuraavanlainen:

```
HRESULT hr = D3DCompileFromFile( srcFile , defines ,
    D3D_COMPILE_STANDARD_FILE_INCLUDE, entryPoint , profile ,
    flags , 0, &shaderBlob , &errorBlob );
```

Oleellista kutsussa ovat parametrit *srcFile* eli lähdetiedosto, sekä *defines*, jonka tulee viitata sävytinfunktion nimeen. Lähdetiedostona voisi olla esimerkiksi kärkipistesävytin VSexample.hlsl ja sävytinfunktiona tiedoston koodista löytyvä VSmain. Muista kääntöfunktion parametreista ei tarvitse tässä esimerkissä välittää.

Kun sävytin on käännetty, luodaan siitä varsinainen sävytinobjekti.

Seuraavaksi kirjoitettu sävytysohjelma käännetään sellaiseen muotoon, jota voidaan hyödyntää grafiikan laskennassa. Grafiikka-ajurit pitävät huolen, että sävytinohjelma päätyy grafiikkapiirin ajettavaksi.

6 Yhteenveto

Tutkielmassa esiteltiin 3D-grafiikassa ja erityisesti tietokonepelien grafiikan piirtämisessä käytettäviä ohjelmoitavia sävyttimiä, niiden historiaa ja mihin niitä voi käyttää. Sävyttimet muodostavat omat moduulinsa grafiikkaliukuhihnalla, joka on vaiheittainen prosessi ruudulla näkyvän grafiikan tuottamiseksi.

Kärkipiste- ja pikselisävytin ovat perinteisiä ja sävyttimiä, joita on käytetty 3D-grafiikan tehokeinojen perustana jo pitkään. Uudempia tulokkaita ovat geometriasävytin ja viimeisinpänä tesselaatiosävytin, joista etenkin viimeinen vaikuttaisi tarjoavan monia uudenlaisia keinoja luoda entistä realistisemman näköistä grafiikka reaaliajassa.

Ohjelmoitavat sävyttimet ovat grafiikkalaitteiston tukeman grafiikkaliukuhihnan vaiheita, joiden avulla on mahdollista vaikuttaa piirrettävään kuvaan.

Lähteet

[Ake02] Akenine-Möller, Tomas ja Haines, Eric: *Real-Time Rendering.* A K Peters/CRC Press, 2. painos, 2002.

[Gre14] Gregory, Jason: *Game Engine Architecture.* A K Peters/CRC Press, 2. painos, 2014.

[Khr15] Khronos Group ja SGI: *OpenGL Shading Language.* `https://www.opengl.org/wiki/OpenGL_Shading_Language`, 2015.

[Laz05] Lazániy, István ja Szirmay-Kalos, László: *Fresnel term approximations for metals.* WSCG '2005: Short Papers: The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2005 in co-operation with EUROGRAPHICS, s. 77-80., 2005.

[Mic11] Microsoft: *Direct3D 11 Graphics.* `https://msdn.microsoft.com/en-us/library/windows/desktop/ff476080(v=vs.85).aspx`, 2011.

[Nvi03] Nvidia Corporation: *The Cg Tutorial.* `http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html`, 2003.

[Nvi10] Nvidia Corporation: *DirectX 11 Tessellation.* `http://www.nvidia.com/object/tessellation.html`, 2010.

[Nvi12] Nvidia Corporation: *Cg Documentation*. `http://http.developer.nvidia.com/Cg/`, 2012.

[Pix15] Pixar: *A Brief Introduction To RenderMan*. `http://renderman.pixar.com/view/brief-introduction-to-renderman`, 2015.

[Puh08] Puhakka, Antti: *3D-grafiikka*. Talentum, 1. painos, 2008.

[Sch14] Schäfer, H. ja Niessner M. ja Keinert, B. ja Stamminger, M. ja C. Loop: *State of the Art Report on Real-time Rendering with Hardware Tessellation*. EUROGRAPHICS 2014/ S, 2014.

[She08] Sherrod, Allen: *Game Graphics Programming*. Charles River Media, 1. painos, 2008.

[Uni15] Unity Technologies: *DirectX 11 and OpenGL Core*. `http://docs.unity3d.com/Manual/UsingDX11GL3Features.html`, 2015.

[Vla01] Vlachos, Alex ja Peters, Jörg ja Boyd, Chas ja Mitchell, Jason L.: *Curved PN triangles*. I3D '01 Proceedings of the 2001 symposium on Interactive 3D graphics, 2001.