Ohjelmoitavat sävyttimet

Janne Timonen

Seminaaritutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 14. 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	14. joulukuuta 2015	11

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, per-pikseli -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].

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-grafiikkassa 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. 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 on globaalia, eli sisäisesti jo olemassa 3D-mallissa, ennen mahdollista projektiota kaksiulotteiseksi kuvaksi [Puh08]. 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.

2.1 Grafiikkaliukuhihna

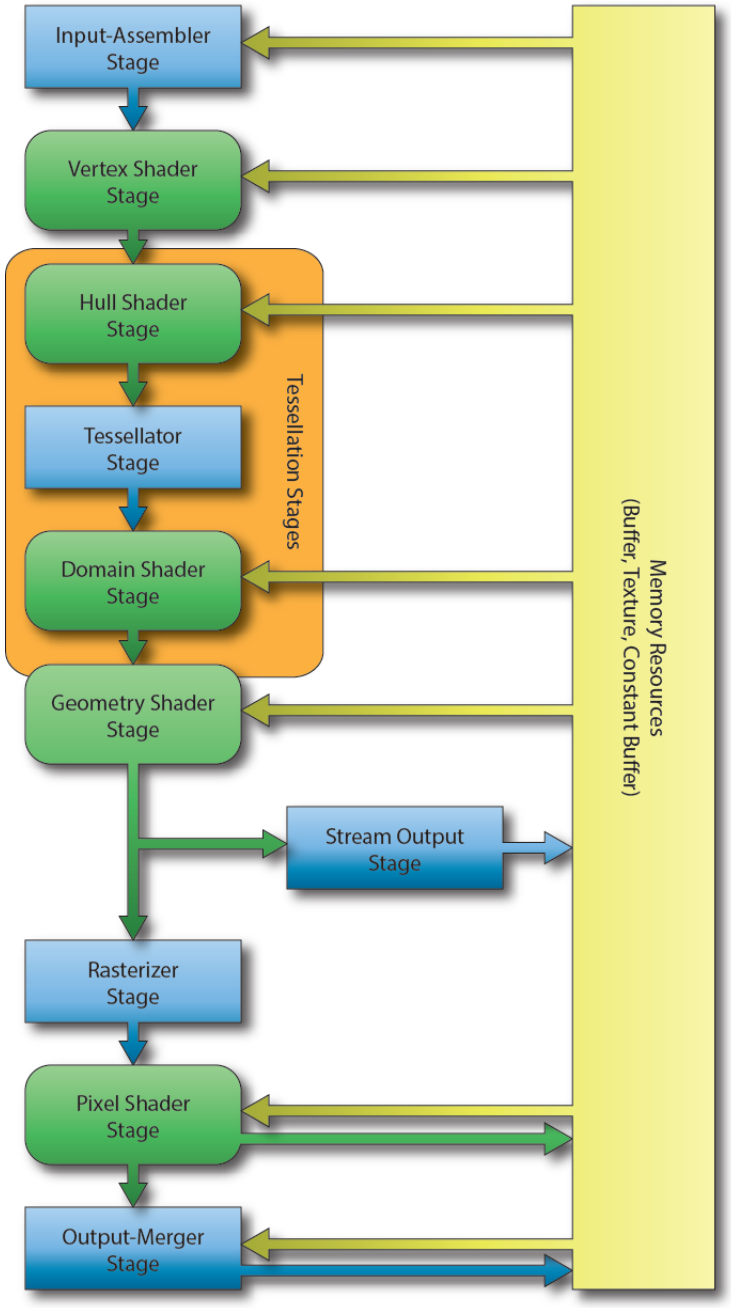
Rasterointiin tähtäävässä 3D-grafiikassa hyödynnetään niin kutsuttua liukuhihnaa (pipeline), joka on pääasiassa sarja tietyssä järjestyksessä tehtäviä askeleita, tai työvaiheita, joilla 3D-malleista luodaan 2D-rasterikuva. Liukuhihna, ja erityisesti *grafiikkaliukuhihna* on siis prosessi, jolla lopullinen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

Shader Model 5 mukainen grafiikkaliukuhihna alkaa kärkipistesävyttimestä. Seuraavana on tesselaatiosävytin, joka koostuu kolmesta vaiheesta: ensimmäisenä on *Hull*-sävytin, toisena kiinteä tesselaatiovaihe, jonka jälkeen on *Domain*-sävytin. Kolmantena vaiheena on geometriasävytin, jonka jälkeen suoritetaan kiinteä rasterointivaihe. Viimeisenä sävytinvaiheena on pikselisävytin. Liukuhihnan vaiheet ovat esitetty kuvassa 1 [Mic11].

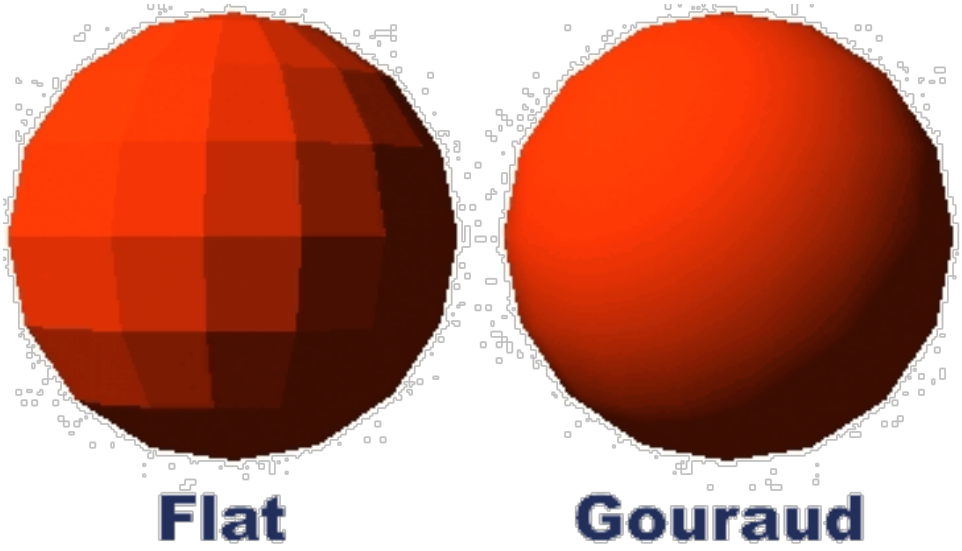
3 Sävyttimien historiaa

Ennen ensimmäisiä *grafiikkakiihdyttimiä*, joihin kuului muun muassa 3Dfx:n Voodoo, grafiikan renderöinti tapahtui prosessorilla, jonka työtaakkaa erilliset grafiikkakiihdyttimet kehitettiin vähentämään. Ennen ohjelmoitavia sävyttimiä grafiikkaa tuotettiin käyttämällä hyväksi grafiikkapiirien *kiinteää liukuhihnaa* (Fixed-Function Pipeline). Kehittäjä saattoi siis antaa raskaat laskutyöt grafiikkakiihdyttimelle hoidettavaksi kiinteällä liukuhihnalla, 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 luonnollisemman näköinen heijastus.

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 saavutetaan vapaus muokata vapaasti sävyttimien käyttäytymistä ja mahdollisuus piirtää kuvaa enemminkin luovuuden rajoissa kuin ennaltamääriteltyjen ehtojen. Ensimmäiset sävytinmallit tukivat ainoastaan alemman tason konekieliin pohjautuvilla kielillä ohjelmointia. Sen lisäksi, että kehittäjien täytyi luoda sävytin, täytyi se luoda lisäksi usein erikseen sekä OpenGL- että



Kuva 1: Grafiikkaliukuhihna



Kuva 2: Tasainen ja Gouraud-sävytys

Direct3D-rajapinnoille johtuen näiden kahden suosituimman rajapinnan kielien poikkeavuuksista [She08, s. 174]. Myöhemmin verteksi-, eli kärkipiste-, ja pikselisävyttimet alkoivat yleistyä. Sävyttimien käyttö grafiikkaliukuhihnalla mahdollistaa rinnakkaistamisen erittäin hyvin [Ake02].

Eräs edelläkävijöitä sävyttimien saralla oli tietokoneanimaatioelokuvistaan tunnettu Pixar-yhtiö kehittämällään *RenderMan*-kielellä, jota käytettiin muun muassa Toy Story -elokuvan tuottamiseen [**?**]

4 Ohjelmoitavat sävyttimet

Ohjelmoitavien sävyttimien alkuaikoina oli sävyttimien luomiseen siis mahdollista käyttää vain alemman tason konekieliin pohjautuvia sävytinkieliä. Korkean tason kielillä on useita hyötyjä konekieliin nähden, ja ne pätevät myös korkean tason sävytinohjelmoinnissa: helpompi luettavuus, kirjoitettavuus, muokattavuus, virheiden etsintä ja löytäminen sekä yleisesti kehitysvauhdin nopeus [She08]. Ohjelmoitavien sävyttimien tultua kasvoi myös tarve korkean tason sävytinkielille, joista mainittavimpina muodostuivat C-pohjaiset Nvidian Cg (C for Graphics) [Nvi03] ja Microsoftin HLSL (High Level Shading Language) -kielet, jotka syntyivät samasta yhteistyöprojektista, ja ovatkin hyvin samankaltaiset, sekä OpenGL:n GLSL -kieli [Khr15]. Näistä kolmesta Cg on jo vanhentunut, eikä sitä enää ylläpidetä. Tässä tutkielmassa esimerkkikielenä käytetään pääasiassa Direct3D:n HLSL:ää.

Ohjelmoitavat sävyttimet muokkaavat kuvaa muodostaviin elementteihin liittyviä attribuutteja. Yksi sävytinvaihe ottaa syötteenään vastaan edelli-

sen tulosteen, joten jokainen vaihe voi jatkaa seuraavan datan työstämistä, kun on saanut edellisen työn valmiiksi. Ohjelmoitavien sävyttimien osalta liukuhihna rakentuu tällä hetkellä pääpiirteissään *kärkipiste-*, *tesselaatio*, *geometria-* ja *pikseli*sävyttimestä. Kärkipistesävytin antaa laskutuloksensa (vaihtoehtoiselle) tesselaatiosävyttimelle, joka antaa tuloksensa (vaihtoehtoiselle) geometriasävyttimelle, joka antaa puolestaan tuloksensa pikselisävyttimelle. Tesselaatiosävytin ja geometriasävytin ovat uudempia tulokkaita, ja niiden käyttö ei ole pakollista, jolloin nämä efektit voidaan jättää pois tai korvata muulla tavoin, esimerkiksi ennen tesselaatiolle omistettua omaa sävytintä tesselaatiosävytys toteutettiin oman kiinteän vaiheen ja geometriasävyttimen avuin.

Sävyttimien käyttö mahdollistaa myös hyvin rinnakkaisuuden käytön, kun muunnoksia tehdään suurille datamäärille kerrallaan, esimerkiksi kaikille ruudun pikseleille. Moderneille grafiikkapiireillä onkin useita sävytinliukuhihnoja rinnakkaisuusmahdollisuuksien hyödyntämiseksi.

Tässä luvussa tarkastellaan erilaisia ohjelmoitavia sävyttimiä siinä järjestyksessä, jossa ne toimivat grafiikkaliukuhihnalla. Eri sävyttimien julkaisun ajankohdan yhteydessä voidaan mainita vastaavan *sävytinmallin* (englanniksi Shader Model) versio tai vastaava Direct3D:n versio.

4.1 Kärkipistesävytin

Kärkipistesävytin, tai *verteksisävytin*, ajetaan kerran jokaista monikulmion, tarkemmin kolmion, kärkipistettä kohden. Kärkipistesävytin ottaa syötteenään 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 siis 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 yksinkertaisen kärkipistesävyttimen rakenne 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;
}
```

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.

Pelkkänä menetelmänä tessellointi ei välttämättä tunnu tuovan mitään mullistavaa esimerkiksi pelien ulkonäköön, sillä ulkoasun kannalta ei ole merkitystä onko esimerkiksi neliö renderoitu kahden vai satojen kolmioiden avulla. Sen sijaan yhdistämällä tesselaatioon muita tekniikoita, ja laittamalla monikulmioista pilkotut palaset esittämään uutta informaatiota, saadaan graafista esitystä realistisemmaksi [Nvi10].

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

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

Kuva 3: Monikulmiomallin silottaminen tesselaation avulla pelissä Stalker: Call of Pripyat

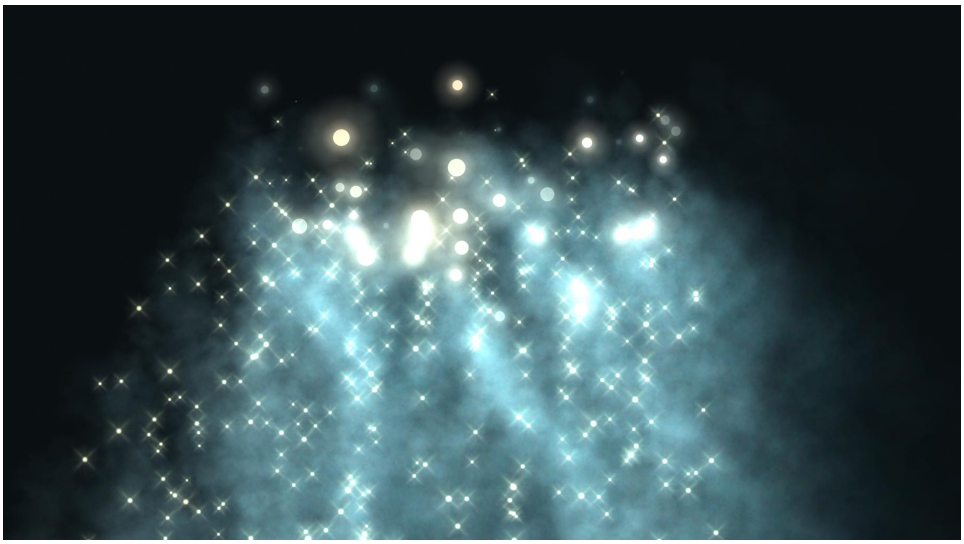
Dynaamisella tesselaatiolla voidaan esimerkiksi skaalata mallien piirron tarkkuutta näkyvyyden suhteen muuttamalla yksityiskohtien määrää lennosta [Nvi10]. Tällöin esimerkiksi avarassa ulkoilmapelinäkymässä kaukaa

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 grafiikkaliukuhihnalla, joten sen käyttö ei ole pakollista. Geometriasävyttimellä tyypillisesti luotuihin asioihin kuuluvat esimerkiksi partikkelijärjestelmät ja *mainostaulut*. Ennen tesselaatiosävyttimen tuloa geometriaävyttimellä hoidettavia tyypillisiä käyttötapauksia oli esimerkiksi aiemmin esitelty dynaaminen tesselaatio.

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.

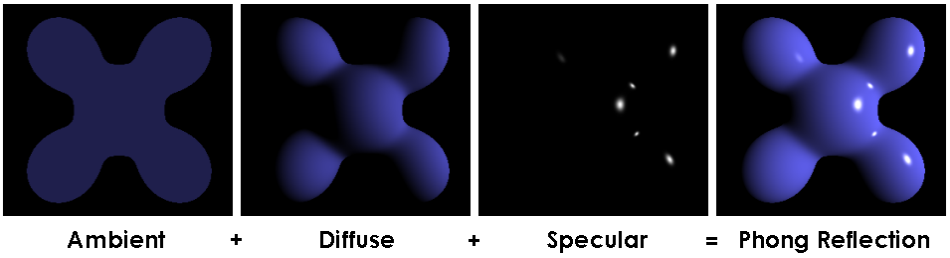


Kuva 4: Partikkelijärjestelmä ja mainostauluttaminen

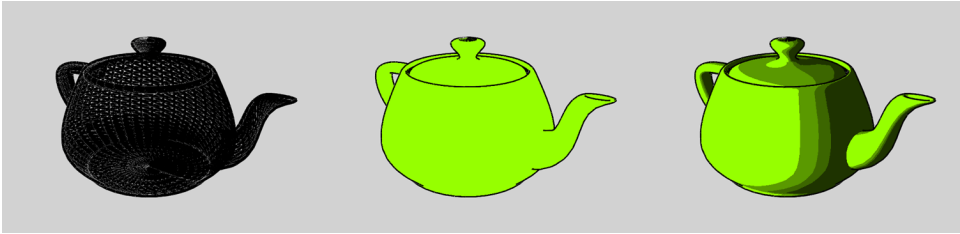
4.4 Pikselisävytin

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 kuhmutus tai *Fresnel-heijastus*, luodaan juuri pikselisävyttimien tasolla. Myös Gouraud-sävytystä realistisempi Phong sävytys luodaan pikselisävyttimellä. Phong-sävytyksen eri komponentit ovat esitelty kuvassa 5. Pikselisävyttimen tärkeimpiin tehtäviin lukeutuvatkin teksturointi ja valaistuksen laskenta.



Kuva 5: Phong-sävytys



Kuva 6: Cel-sävytys

4.5 Laskentasävytin

Shader Model 5.0 toi myös tuen *laskentasävyttimelle* (englanniksi Compute Shader). Laskentasävyttimen on enemmän yleiskäyttöinen ja löyhästi määritelty sävytin kuin muut tutkielmassa esitellyt sävyttimet. Laskentasävytin ei kuulu grafiikkaliukuhihnalla mihinkään tiettyyn kohtaan 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...

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.

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

6 Yhteenveto

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.

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

[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.