Ohjelmoitavat sävyttimet

Janne Timonen

Seminaaritutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 5. marraskuuta 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	5. marraskuuta 2015	7

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, esimerkiksi monikulmion *kärkipisteen*, *primitiivin*, eli monikulmion kuten kolmion, tai *fragmentin*, kuten pikselin, ja tuottaa syötteestä tuloksena muunnettuja elementtejä, joiden määrä voi vaihdella nollasta useaan, riippuen sävyttimestä ja sen suorittamasta tehtävästä [Gre14].

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 esimerkiksi mahdollista tehdä ja kuinka se tapahtuu.

2 3D-grafiikka

3D-grafiikkassa. ja erityisesti tämän tutkielman tapauksessa *rasterointia*, eli kuvan muuttamista pikseleillä esitettävään muotoon (rasterikuvaksi), hyödyntävässä 3D-grafiikan reaaliaikaisessa renderoinnissa tuotetaan kolmiulotteisista malleista ja asioista kaksiulotteinen representaatio, eli esimerkiksi katsojan näkemä kuva tietokoneen ruudulla. Tarkkaa reaaliaikaisuutta vaativien grafiikkasovellusten, kuten pelien, tapauksessa nopea kuvan piirtäminen nousee tärkeäksi vaatimukseksi, jolloin monikulmioista, eli *polygoneista*, tuotetaan rasteroimalla kaksiulotteista kuvaa tavoitteena ruudunpäivitysnopeus, joka vaikuttaa ihmisen silmään sulavalta (noin 30 ruutua sekunnissa (30 *fps*) [Gre14].

Ei-reaaliaikaisiin 3D-grafiikan renderointitapoihin lukeutuu esimerkiksi *säteenjäljitys* (ray tracing), jossa esimerkiksi 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.

2.1 Grafiikkaliukuhihna

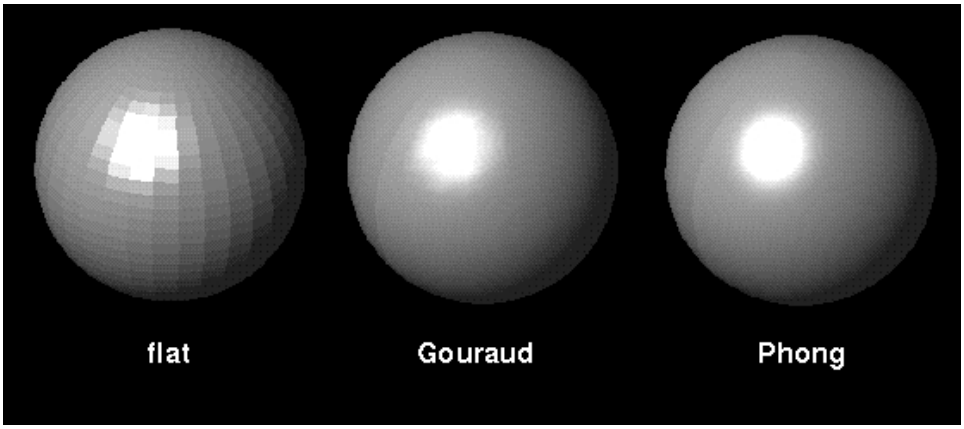
Rasterointiin tähtäävässä 3D-grafiikassa hyödynnetään niin kutsuttua liukuhihnaa (pipeline), joka on pääasiassa sarja tietyssä järjestyksessä tehtäviä

askeleita, tai työvaiheita, joilla 3D-malleista luodaan 2D-rasterikuva. Liukuhihna, ja erityisesti *grafiikkaliukuhihna* on siis prosessi, jolla lopullinen kuva tuotetaan. Tähän tarkoitukseen suosituimmat grafiikkaohjelmointirajapinnat (API) ovat OpenGL ja Direct3D.

Sävyttimien ymmärtämisen kannalta on hyödyllistä ymmärtää myös erilaiset 3D-grafiikkaan liittyv't *koordinaatistot*, joissa 3D-malleja käsitellään. Näihin lukeutuvat muun muassa *mallikoordinaatisto* (voidaan puhua myös koordinaatiston sijaan avaruudesta, esim *model space*, jossa ei kyseisen 3D-mallin lisäksi ole mitään muuta, ja *maailmakoordinaatisto*, jossa malli taas esiintyy suhteessa muihin malleihin, jotka maailmakoordinaatistoon on kytketty.

3 Sävyttimien historiaa

Ennen ensimmäisiä *grafiikkakiihdyttimiä*, joihin kuului muun muassa 3Dfx:n Voodoo, grafiikan renderöinti tapahtui prosessorilla (CPU), jonka työtaakkaa erilliset grafiikkapiirit kehitettiin vähentämään. Ennen ohjelmoitavia sävyttimiä grafiikkaa tuotettiin käyttämällä hyväksi näytönohjaimien (GPU) *kiinteää liukuhihnaa* (Fixed-Function Pipeline). Kehittäjä saattoi siis antaa raskaat laskutyöt näytönohjaimelle, tai erilliselle kiihdytinkortille, 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 esimerkki kuvassa 1.



Kuva 1: Tasainen, Gouraud- ja Phong-sävytys

Myöhemmin tulivat ensimmäiset *ohjelmoitavaa renderointiliukuhihnaa* tukevat näytönohjainpiirit, joissa kiinteän liukuhihnan pystyi korvaamaan omilla vapaasti ohjelmoitavilla sävyttimillä. Korvaamalla kiinteän liukuhih-

nan laskenta nykyajan grafiikkapiirien tukemilla ohjelmoitavilla sävyttimillä saavutetaan vapaus muokata vapaasti laskenta- ja muokkausoperaatioita, mikä antaa mahdollisuuden piirtää kuvaa enemminkin luovuuden rajoissa, kuin ennaltamääriteltyjen ehtojen. Ensimmäiset sävytinmallit tukivat ainoastaan alemman tason konekielillä ohjelmointia. Sen lisäksi, että kehittäjien täytyi luoda sävyttimen konekielellä, täytyi sävytin luoda lisäksi usein erikseen sekä OpenGL- että Direct3D-rajapinnoille johtuen näiden kahden suosituimman rajapinnan konekielien poikkeavuuksista. 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 Korkean tason sävytinkielet

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 [**?**]. Näistä kolmesta Cg on jo deprekoitunut. Tässä tutkielmassa esimerkkikielenä käytetään pääasiassa Direct3D:n HLSL:ää.

5 Ohjelmoitavat sävyttimet

Ohjelmoitavat sävyttimet antavat algoritmeillaan mahdollisuuden muokata vapaasti ja reaaliaikaisesti kuvaa muodostaviin elementteihin liittyviä attribuutteja. Yksi sävytinvaihe ottaa syötteenään vastaan edellisen 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.

5.1 Kärkipistesävyttimet

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 väreily.

Kärkipistesävyttimen tärkeimpiin tehtäviin kuuluu siis tehdä tarvittavat koordinaatistomuunnokset syötteinä saaduille kärkipisteille. Ensimmäisenä tehdään *mallimuunnosa*, eli muunnos mallikoordinaatistosta maailmakoordinaatistoon, jolloin kärkipiste sidotaan omasta paikallisesta koordinaatistostaan absoluuttisesti maailmanäkymään. Tämän jälkeen tehdään *katsojamuunnos* maailmakoordinaatistosta katsojan koordinaatistoon, jolloin origo on katsotaan katsojan "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

```
struct VSInput
{
        float4 Pos : POSITION;
        float3 Normal : NORMAL;
        float2 Texcoord : TEXCOORD0;
};
```

```
PSInput VertexShader(VSInput In)
{
        PSInput Out;

        Out.Normal = mul(In.Normal, (float3x3)g_mWorld);
        Out.WorldPos = mul(In.Pos, g_mWorld);
        Out.Pos = mul(Out.WorldPos, g_mViewProj);
        Out.Texcoord = In.Texcoord;

        return Out;
}
```

5.2 Tesselaatiosävytin

Tutkielman sävyttimistä uusin on Shader Model 5.0:n, rajapintojen kohdalla DirectX 11 ja OpenGL 4.0, myötä tullut tesselaatiosävytin. *Tessellaatiossa* 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.

Tessellaatiosävytin on jaettu kolmeen vaiheeseen, joista ensimmäinen ja kolmas ovat ohjelmoitavia sävytinvaiheita: *Hull Shader*, traditionaalinen tesselaatiovaihe*Tessellation Stage* ja *Domain Shader* [Mic11]

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 2 näkyvän Stalker: Call of Pripyat -pelin hahmon kaasunaamarin suodattimen reunoja on saatu tesselaatiolla luonnollisemmalla tavalla kaartuviksi verrattuna yksinkertaisempaan ja vähäisemmillä monikulmioilla piirrettyyn malliin.

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

Kuva 2: Monikulmiomallin silottaminen tesselaation avulla pelissä Stalker: Call of Pripyat

lähestyttäessä monikulmioiden määrää lisätään dynaamisesti, kunnes läheltä katsottuna malli voidaan piirtää tuhansista monikulmioista [Gre14].

5.3 Geometriasävyttimet

Geometriasävyttimet on kärkipiste- ja pikselisävyttimiin verrattuna uudempi sävytin sen tultua esitellyksi DirectX 10:n myötä. Geometriasävytin sijaitsee renderointiliukuhihnalla tesselaatiosävyttimen jälkeen, ja ennen pikselisävytintä. Sen käyttö ei ole pakollista, ja vielä esimerkiksi pelien sävytyksessä on otettava huomioon jonkinlainen varakeino mikäli geometriasävyttimien käyttö ei ole mahdollista. 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 yhteen, tai hylätä kokonaan. Toisin kuin kärkipistesävytin, joka kykenee käsittelemään vain yhden monikulmion kärkipisteen kerrallaan, pystyy geometriasävytin “näkemään” koko primitiivin kaikkine kärkipisteineen.

5.4 Pikseli-/fragmenttisävyttimet

Pikselisävytin, tai fragmenttisävytin (riippuen sävytinkielen terminologiasta; myöhemmin tekstissä puhutaan HLSL:n konvention mukaan pikselisävyttimestä), on graafinen funktio, joka laskee muunnoksia per-pikseli -periaatteella, eli muunnokset voidaan tehdä jokaiselle yksittäiselle pikselille, tai muulle fragmentille, erikseen []. Pikselisävytin ajetaan kerran per pikseli, 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 kuvan 1 Phong-sävytys luodaan pikselisävyttimellä. Pikselisävyttimen tärkeimpiin tehtäviin lukeutuvatkin teksturointi ja valaistuksen laskenta.

Lähteet

[Ake02] Akenine-Möller Tomas and 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.

[Mic11] Microsoft: *Programming Guide for Direct3D 11.* `https://msdn.microsoft.com/en-us/library/windows/desktop/ff476345`, 2011.

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

[Nvi10] Nvidia: *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.

[Sil15] Silicon Graphics and Khronos Group: *OpenGL Shading Language.* `https://www.opengl.org/wiki/OpenGL_Shading_Language`, 2015.

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