

# **Sävytinohjelmointi**

Janne Timonen

Seminaaritutkielma  
HELSINGIN YLIOPISTO  
Tietojenkäsittelytieteen laitos

Helsinki, 4. marraskuuta 2015

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Janne Timonen			
Työn nimi — Arbetets titel — Title			
Sävytinohjelmointi			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages	
Seminaaritutkielma	4. marraskuuta 2015	6	
Tiivistelmä — Referat — Abstract			
Tiivistelmä.			
Avainsanat — Nyckelord — Keywords			
avainsana 1, avainsana 2, avainsana 3			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

## Sisältö

<b>1</b>	<b>Johdanto</b>	<b>1</b>
<b>2</b>	<b>3D-grafiikka</b>	<b>1</b>
<b>3</b>	<b>Sävyttimien historiaa</b>	<b>2</b>
<b>4</b>	<b>Korkean tason sävytinkielet</b>	<b>2</b>
<b>5</b>	<b>Ohjelmoitavat sävyttimet</b>	<b>3</b>
5.1	Kärkipistesävyttimet . . . . .	3
5.2	Tesselaatiosävytin . . . . .	4
5.3	Geometriasävyttimet . . . . .	4
5.4	Pikseli-/fragmenttisävyttimet . . . . .	5
	<b>Lähteet</b>	<b>5</b>

# 1 Johdanto

*Sävyttimet* ovat ohjelmia, joiden tehtävänä *grafikkaliukuhihnalla* (asteittain etenevässä prosessissa tuottaa 2d-kuvaa) on *sävyttää*, eli tuottaa tietyillä tavoilla dataa kuvaksi. Tämä voi tarkoittaa esimerkiksi jonkin objektin piirtämistä sijainnin mukaan, per-pikseli -värinmäärittystä, pinnanmuotojen simulointia tai muita erikoistehostemaisiakin keinoja. Sävytin ottaa syöteenään elementin, esimerkiksi monikulmion *kärkipisteen*, *primitiivin*, eli monikulmion kuten kolmion, tai *fragmentin*, kuten pikselin, ja tuottaa syötteestä tuloksena muunnettuja elementtejä, joiden määrä voi vaihdella nolasta 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-grafiikassa, ja erityisesti tämän tutkielman tapauksessa *rasterointia* 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 realistisesti käytettävissä reaaliaikaisessa käytössä, kuten peleissä, mutta enakkoon renderöitynä tuottaa lähes fotorealistista kuvaa. Säteenjäljitykseen, tai vastaaviin tekniikoihin, ei tässä tutkielmassa enää palata.

Rasterointiin tähtäävässä 3D-grafiikassa hyödynnetään niin kutsuttua liukuhinaa (pipeline), joka on pääasiassa sarja tietyssä järjestyksessä tehtäviä askeleita, tai työvaiheita.

### 3 Sävyttimien historiaa

Ennen ensimmäisiä *grafiikkakiuhdyttimiä*, kuten 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 hoidettavaksi kiinteällä liukuhihnalla, mutta itse liukuhinnan 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-sävytys).

Myöhemmin tulivat ensimmäiset *ohjelmoitavaa renderointiliukuhihnaa* tukevat näytönohjainpiirit, joissa kiinteän liukuhinnan pystyi korvaamaan omilla vapaasti ohjelmoitavilla sävyttimillä. Korvaamalla kiinteän liukuhinnan laskenta nykyajan grafiikkapiirien tukemilla ohjelmoitavilla sävyttimillä saavutetaan vapaus muokata vapaasti laskenta- ja muokkausoperaatioita, mikä antaa mahdollisuuden piirtää kuvaa enemminkin luovuuden rajoissa, kuin ennaltamääriteltujen ehtojen. Ensimmäiset sävyttinmallit 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 konekielten poikkeavuuksista. Myöhemmin verteksi-, eli kärkipiste-, ja pikselisävyttimet alkoivat yleistyä. Sävyttimien käyttö grafiikkaliukuhihnalla mahdollistaa rinnakkaistamisen erittäin hyvin. [AM02]

Eräs edelläkävijöitä sävyttimien saralla oli tietokoneanimaatioelokuviin 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) ja Microsoftin HLSL (High Level Shading Language), jotka syntyivät samasta yhteistyöprojektista (lähde?) sekä OpenGL:n GLSL -kielet. Näistä Cg on jo deprekoitunut. Tässä tutkielmassa esimerkikielenä 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 pääpiirteissään kärkipistesävyttimestä, joka antaa laskutuloksensa (vaihtoehtoiselle) geometriasävyttimelle, joka voi luoda jopa uusia geometriaprimitiivejä, ja antaa puolestaan tuloksensa pikselisävyttimelle.

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 ovat sijoitettu 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 x-y-z -koordinaatistossa malli- tai maailma-avaruudessa, sekä pinnan normaalivektorin. Tulostena kärkipistesävytin antaa kärkipisteen, joka on käynyt läpi valaistus- ja muunnosvaiheet, ja joka ilmaistaan nyt normalisoidussa kuvausavaruudessa. Vähintään kärkipistesävyttimen tulee siis antaa tuloksena kärkipiste uniformina tietona. [Puh08]

Yleisesti siis kärkipistesävytin voi muokata monikulmion kärkipisteen, normaalin, tekstuurikoordinaattien ja paikan arvoja. Sävyttimellä voi luoda esimerkiksi tuulessa heiluvat puiden oksat.

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

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 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.

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].

Muita tesselaation avulla saavutettavia keinoja ovat 3D-mallien siloittaminen PN(point normal)-kolmioiden avulla [Vla01]. Dynaamisella tesselaatiolla voidaan esimerkiksi skaalata mallien piirron tarkkuutta näkyvyyden suhteen muuttamalla yksityiskohtien määrää lennosta [Nvi10]. Tällöin esimerkiksi avarassa ulkoilmapelinäkyvässä kaukaa katsottuna jostain mallista piirretään malli muutamalla monikulmiolla, ja lähestyttäessä monikulmioiden määrää lisätään dynaamisesti, kunnes läheltä katsottuna malli voidaan piirtää tuhansista monikulmioista.

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

## 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.

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 primitiiveja [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. Sävyttimellä hoidettavia tyypillisiä käyttötapauksia ovat esimerkiksi *dynaaminen tessellaatio*???

## 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. Pikselisävyttimen tärkeimpiin tehtäviin lukeutuvatkin teksturointi ja valaistuksen laskenta.

## Lähteet

- [AM02] Akenine-Möller, Tomas;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.
- [Nvi10] Nvidia: *Tessellation*. <http://www.nvidia.com/object/tessellation.html>, 2010.
- [Puh08] Puhakka, Antti: *3D-grafikka*. Talentum, 1. painos, 2008.
- [She08] Sherrod, Allen: *Game Graphics Programming*. Charles River Media, 1. painos, 2008.



- [Vla01] Vlachos, Alex: *Curved PN triangles*. I3D '01 Proceedings of the 2001 symposium on Interactive 3D graphics, 2001.