LuK-tutkielma: Reaaliaikaiset varjot ja varjosärmiöalgoritmi

TUOMAS MÄKILÄ[*]

17/04/2004

[*] tusuma@utu.fi

TURUN YLIOPISTO
Informaatioteknologian laitos

TUOMAS MÄKILÄ: Reaaliaikaiset varjot ja varjosärmiöalgoritmi
LuK-tutkielma, 29 s.
Tietotekniikan DI-koulutusohjelma
Huhtikuu 2004

Tietokonegrafiikassa tilan hahmottamisen ja tunnelman kannalta oleellisia ovat varjot. Varjoja käytetään nykyään lähes jokaisessa reaaliaikaisessa kolmiulotteisessa sovelluksessa. Tässä tutkielmassa tutustutaan tärkeimpien reaaliaikaisten varjoalgoritmien toimintaan. Esittelyssä ovat valevarjo-, projektiovarjo-, varjokartta- ja varjosärmiöalgoritmit. Erityisen tarkasti tarkastellaan viimeksi mainittua, josta esitellään useita erilaisia toteutusvaihtoehtoja. Jokaista toteutusta pyritään lähestymään reaaliaikaisuuden näkökulmasta ja pohtimaan kunkin menetelmän etuja sekä haittoja reaaliaikaisen grafiikan tuottamisessa.

Tutkielman aihealueeseen perehdytään lähdekirjallisuuden avulla. Tärkeimpänä lähteenä on Frank Crown 1977 kirjoittama artikkeli Shadow algorithms for computer graphics, jossa esitellään varjosärmiöalgoritmin perusidea. Tutkielmassa tarkastellaan myös muita varjosärmiöalgoritmin toteutusmenetelmiä ja variaatioita esitteleviä keskeisiä artikkeleita.

Varjosärmiöalgoritmi soveltuu tietyissä olosuhteissa reaaliaikaisten varjojen piirtämiseen. Algoritmi tuottaa fysikaalisen käsityksen mukaisia tarkkoja kovia varjoja ja on riippumaton piirrettävän maailman valonlähteiden sijoittelusta. Jotta varjosärmiöalgoritmi toimisi oikein, tulee varjostavan kappaleen olla tietyn muotoinen ja algoritmin toteutuksessa tulee ottaa huomioon useita erityistapauksia. Varjosärmiöalgoritmin käyttäminen pehmeiden varjojen piirtämiseen on todettu useissa tutkimuksissa mahdolliseksi, mutta käytännön sovelluksissa on vielä ratkaisemattomia ongelmia. Varjosärmiöalgoritmin kanssa samankaltaisia varjoja saadaan aikaan varjokarttamenetelmällä.

Asiasanat: kolmiulotteinen grafiikka, reaaliaikaisuus, algoritmit, varjösärmiöalgoritmi

Sisällys

1. Johdanto

Reaaliaikainen kolmiulotteinen tietokonegrafiikka, 3D-grafiikka, tuli tavallisen PC-käyttäjän ruudulle noin kymmenen vuotta sitten. Silloin grafiikka oli vielä varsin karkeata, ja monet arkielämässä itsestään selvät yksityiskohdat puuttuivat kokonaan. Eräs tällainen yksityiskohta olivat esineiden luomat varjot. Vasta muutaman viime vuoden aikana reaaliaikaiset varjot ovat yleistyneet niin peleissä kuin 3D-suunnitteluohjelmissakin. Varjojen avulla paitsi luodaan aidomman näköisiä maailmoja tietokoneen ruudulle niin myös helpotetaan kolmiulotteisen tilan ja erityisesti siinä olevien esineiden keskinäisten suhteiden hahmottamista.

Tämän työn tavoitteena on esitellä yleisimpien tietokonegrafiikassa käytettyjen reaaliaikaisten varjostusmenetelmien toimintaa. Erityisesti tarkastellaan varjosärmiöalgoritmin kehitystä ja sen eri variaatioita. Jokaisen menetelmän sopivuutta käytännön reaaliaikaisiin sovelluksiin pyritään arvioimaan laadullisesti. Tarkempia arvioita, esim. algoritmien yksityiskohtaisia kompleksisuusanalyysejä, ei tässä tutkielmassa. Koska työ käsittelee nimenomaan reaaliaikaisia menetelmiä, jäävät useat varjostusalgoritmit tämän työn ulkopuolelle. Niihin voi tutustua Woon, Poulinin ja Fournierin varjostusmenetelmiin tekemässä yleiskatsauksesta [7].

Tutkielma esittelee ensin kolmannessa luvussa 3D-grafiikkaan ja varjoihin liittyviä peruskäsitteitä. Neljännessä luvussa käydään läpi yleisimmät reaaliaikaiset varjostusmenetelmät. Viidennessä esitellään alkuperäinen varjosärmiöalgoritmi sekä useita siitä tehtyjä muunnoksia, joissa on joko ratkaistu alkuperäisen algoritmin puutteita tai lisätty siihen uusia ominaisuuksia. Luvun lopussa on lyhyt vertailu eri reaaliaikaisten varjostusmenetelmien ominaisuuksista ja kompleksisuuksista. Ennen loppuyhteenvetoa pohditaan vielä lyhyesti reaaliaikaisten varjojen käytännön sovelluksia.

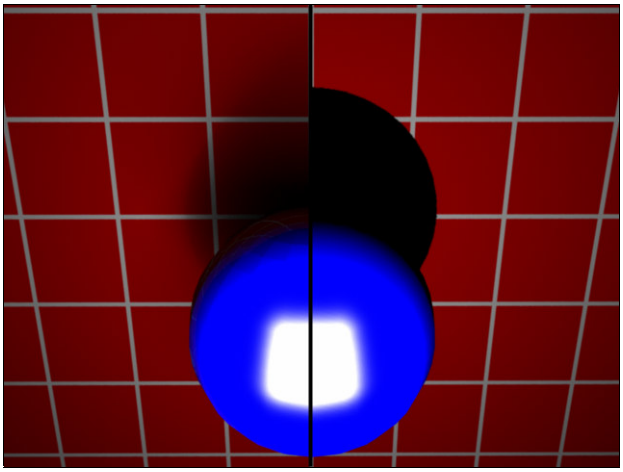
2. Peruskäsitteitä

Aluksi on tarpeellista käydä läpi reaaliaikaisten varjojen luomiseen kiinteästi liittyviä 3D-grafiikan peruskäsitteitä. Osaltaan nämä toimivat kertauksena ja helpottavat tutkielmaan tutustumista. Toisaalta nämä perusteet auttavat kiinnittämään huomion niihin lähtökohtiin, joiden pohjalta eri varjostusalgoritmeja tässä työssä tarkastellaan. Tässä luvussa myös esitellään tutkielmassa käytettävä termistö. Ensin luvussa käydään läpi varjojen fysikaaliset perusteet ja niihin liittyvät käsitteet. Tämän jälkeen keskitytään reaaliaikaisen mallintamisen oleellisiin osa-alueisiin: kolmiulotteiseen malliin, renderointiputken perusideaan sekä z- ja sapluunapuskurien toimintaan.

2.1. Varjojen fysikaaliset perusteet

Reaalimaailmassa varjot syntyvät vastaanottavan kappaleen pinnalle, kun kappaleen ja valonlähteen väliin tulee toinen, **varjostava kappale** (occluder), joka heikentää tai kokonaan estää valonsäteiden kulkua. Aluetta, joka on täysin varjossa eli jonka pintaan osuvan valon intensiteetti lähenee nollaa, sanotaan **täysvarjoksi** (umbra), ja aluetta, joka on osittain varjossa eli jonka pintaan osuvan valon intensiteetti on heikentynyt, sanotaan **puolivarjoksi** (penumbra). Puolivarjo ei ole tasainen, vaan siirtymä täysvarjosta täysin valaistuun alueeseen on liukuva. Tilanne, jossa täysvarjo pääsee syntymään, edellyttää tasan yhtä pistemäistä valonlähdettä. Luonnossa esiintyvillä tai ihmisen valmistamilla valonlähteillä on aina tietty nollasta eroava pinta-ala, joten pistemäinen valonlähde on teoreettinen käsite. Näin ollen arkielämässä esiintyvät varjot koostuvat aina sekä täys- että puolivarjosta. Lisäksi valon sironta ilmakehässä ja näin syntyvä taustavalo vaikeuttavat teräväreunaisten varjojen syntymistä varsinkin päivisin.

Tietokonegrafiikasta puhuttaessa alueen on katsottu olevan varjossa, jos sen ja valonlähteen välissä on varjostava kappale [7]. Alueen intensiteetin ei siis tarvitse vähentyä nollaan varjostavan kappaleen vaikutuksesta. Tällainen tilanne saattaa syntyä esimerkiksi silloin, jos varjostava kappale on osittain läpinäkyvä (esim. värjätty linssi) tai valonlähde ei ole pistemäinen. Lähestymistapa on ymmärrettävä, sillä varjostavien kappaleiden läpinäkyvyyden ja ei-pistemäiset valonlähteet huomioon ottavat varjostusalgoritmit kykenevät teoriassa laskemaan kaikki fysikaalisen käsityksen mukaiset puolivarjot. Yleensä varjostusalgoritmit kuitenkin jättävät varjostavan kappaleen osittaisen läpinäkyvyyden huomiotta ja mallintavat korkeintaan ei-pistemäiset valonlähteet.



Kuva 1. Levymäisen neliönmallisen valonlähteen valaisema pallo. Keskiviivan vasemmalla puolella on pallon luoma pehmeä varjo ja oikealla pallon luoma kova varjo

Varjoalgoritmit jaetaan kahteen ryhmään riippuen siitä, ottavatko ne puolivarjon huomioon vai eivät. Puolivarjosta piittaamattomien algoritmien sanotaan generoivan **kovia varjoja** (hard shadows). Tällöin kuvan piste on joko varjossa tai ei ole. Puolivarjon mallintavien algoritmien generoimia varjoja sanotaan puolestaan **pehmeiksi varjoiksi** (soft shadows). Kuvassa 1 esitellään esimerkki pehmeistä ja kovista varjoista.

2.2. Kolmiulotteinen malli

Kolmiulotteista tietokonegrafiikkaa käsiteltäessä määritellään tietorakenne, johon malli käsiteltävästä kolmiulotteisesta kappaleesta tallennetaan. Seuraavassa esitellään ns. polygonigrafiikan konsepti, joka on hyvin yleinen erityisesti reaaliaikaisen 3D-grafiikan piirissä. Ideana on koostaa kolmiulotteinen malli useista monikulmioista.

Pienin yksikkö, josta mallin muodostaminen aloitetaan, on kolmiulotteisessa avaruudessa sijaitseva **kärkipiste** (vertex). Piste voidaan kuvata kolmen liukuluvun yhdistelmänä siten, että nämä liukuluvut sisältävät pisteen x-, y- ja z-akselin suuntaiset koordinaatit. Yhdistämällä kaksi avaruuden pistettä saadaan aikaiseksi suora, joka kuvaa monikulmion yhtä **särmää** (edge). Särmän kuvaamiseen tietorakenteessa riittää siis kaksi pistettä. Vähintään yhdestä pisteestä toisissaan kiinni olevia särmiä yhdistelemällä saadaan aikaiseksi **monikulmioita** (polygon). Suljetun monikulmion muodostaminen vaatii, että särmät muodostavat suljetun piirin. Tietorakenteessa monikulmio voidaan kuvata reunojen taulukkona tai listana. On huomioitavaa, että särmien piirtojärjestys on yleensä merkityksellinen, sillä se määrää monikulmion sivun suunnan. Voidaan esimerkiksi määritellä, että polygoni, jonka särmät piirretään katsojan kuvakulmasta myötäpäivään, on **katsojaa kohden** (front-facing polygon) ja polygoni, jonka särmät piirretään vastapäivään, on **katsojasta poispäin** (back-facing polygon).

Reaaliaikajärjestelmissä monikulmiot redusoidaan yleensä **kolmioiksi** (triangle). Tällöin useat kolmiulotteiset laskutoimitukset helpottuvat huomattavasti, sillä kolmio muodostaa aina yhden tason. Myös tietorakenne yksinkertaistuu, sillä kolmion kuvaamiseen voidaan käyttää kiinteää kolmipaikkaista taulukkoa.

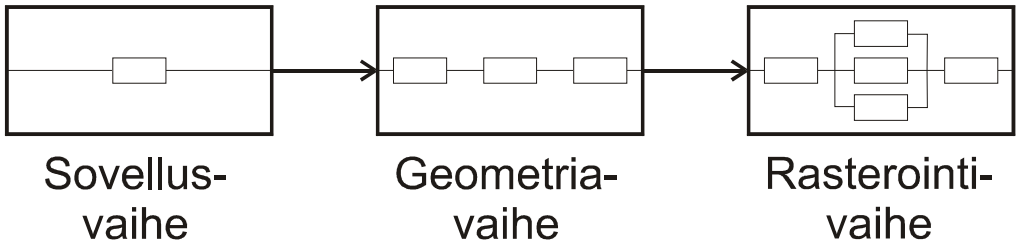
Näistä perusosista muodostetaan varsinainen kolmiulotteinen **malli** (model). Malli rakennetaan yhdistelemällä monikulmioita toisiinsa. Lisäämällä mallin monikulmiomäärää saadaan mallin yksityiskohtaisuutta parannettua. Vaikka mallit periaatteessa rakentuvatkin pelkästään suorista viivoista, käytettäessä riittävän tarkkaa mallia saadaan kuvaruudulle käytännössä myös kaarevia pintoja.

Yllä oleva malli ja sitä kuvaava tietorakenne kuvaavat vain kolmiulotteisen mallin geometrisen muodon. Joskus monikulmioiden yhteyteen tallennetaan tiedot myös niiden normaaleista esim. kappaleen valaisemista ja piirtämisen nopeuttamista (näkymättömien pintojen poisto) varten. Yleensä normaalit lasketaan kuitenkin reaaliaikaisesti. Lisäksi malli sisältää tavallisesti tiedot pintojen ominaisuuksista ja **tekstuureista** (texture) eli pinnoille piirrettävistä bittikarttakuvista.

Useista malleista voidaan rakentaa kokonainen kolmiulotteinen **maailma** (scene). Maailmaan liittyy kiinteänä osana myös tiedot kameroiden ja valojen paikoista.

2.3. *Renderointiputki*

Kolmiulotteinen malli ja sen kuvaus tietokoneen muistissa esittävät erittäin abstraktilla tasolla kolmiulotteisia kappaleita. Tietokonegrafiikassa oleellista on konkretisoida kolmiulotteinen malli ja saada piirrettyä siitä kuva tietokoneen ruudulle. Tämä tehtävä ei ole aivan yksinkertainen, vaan erilaisia toteutusratkaisuja lienee tuhansia. Eräs ratkaisu löytyy Real-time Rendering -kirjasta, jossa esitellään **renderointiputki**-käsite (rendering pipeline) [6]. Renderointiputki käsittää tärkeimmät piirtoprosessin vaiheet. Vaiheet on esitetty kuvassa 2.



Kuva 2. Renderointiputki voidaan jakaa kuvassa näkyvään kolmeen päävaiheeseen. Pipeline-arkkitehtuurin mukaisesti vaihe voi siirtyä käsittelemään seuraavan kuvaruudun tietoja heti, kun nykyisen ruudun tulostetiedot on siirretty seuraavalle vaiheelle. Geometria- ja rasterointivaiheet voidaan edelleen jakaa pipeline-arkkitehtuurin mukaisiin osavaiheisiin. Rasterointivaiheessa voidaan lisäksi tietyt osavaiheet suorittaa rinnakkaisesti.

Renderointiputki jaetaan kolmeen päävaiheeseen: sovellus-, geometria ja rasterointivaiheeseen. Nämä vaiheet jaetaan vielä osavaiheisiin, joissa varsinainen

toiminnallisuus implementoidaan. Renderointiputki perustuu pipeline- arkkitehtuuriin, jossa vaiheet suoritetaan peräkkäin siten, että edellisen vaiheen tulos ohjataan seuraavan vaiheen syötteeksi. Toteutustavan mukaan putken erilliset vaiheet voivat toimia samanaikaisesti, joskin putken hitain osavaihe määrittelee koko putken nopeuden. Renderointiputken osavaiheita voidaan jakaa myös rinnakkaisesti suoritettaviksi, mikäli vaiheiden luonne tämän sallii. Renderointiputkiarkkitehtuuri mahdollistaa grafiikan modulaarisen käsittelyn.

Sovellusvaiheessa muokataan kolmiulotteisen maailman osia. Näihin muokkaustoimiin kuuluvat esim. kolmiulotteisten mallien liikuttaminen ja animointi, uusien mallien lisääminen maailmaan sekä kameroiden ja valojen paikkojen määrittely. Sovellusvaihe toteutetaan yleensä kokonaan ohjelmallisesti. Vaiheen tuottamat tiedot kolmiulotteisesta maailmasta siirretään geometriavaiheelle jatkokäsittelyä varten.

Geometriavaiheessa kolmiulotteista mallia muokataan siten, että se voidaan vaivatta piirtää näyttöruudulle. Tämä tarkoittaa mm. valojen ja varjojen laskemista, kameran näköpiirissä olevan alueen päättelemistä, näkymättömien pintojen poistamista, ruudulle piirrettävän osuuden rajaamista kolmiulotteisesta mallista. Geometriavaiheen tuloksena syntyvät tiedot ruudulle piirrettävistä kolmioista väri- ja tekstuuriarvoineen.

Rasterointivaiheessa kolmiulotteinen kuva piirretään lopullisesti kuvaruudulle. Geometriavaiheesta saadut monikulmiot muunnetaan ruudulla näkyviksi kuvapisteiksi. Rasterointivaiheessa monikulmiot yleensä lisäksi pinnoitetaan (teksturoidaan) bittikarttakuvilla. Rasterointivaihe suoritetaan nykyisin yleensä erillisellä 3D-näytönohjaimella, joka muuntaa hyvin nopeasti piirrettävien kolmioiden pistetiedot konkreettisiksi pinnoitetuiksi monikulmioiksi näyttömuistiin.

Nykyisin sovellusvaihe toteutetaan yleisesti ohjelmistopuolella ja osa geometriavaiheesta sekä rasterointivaihe taas kolmiulotteisen grafiikan käsittelyyn tarkoitetulla laitteistolla (3D-näytönohjaimet) [6].

Muutama renderointiputkeen liittyvä käsite esiintyy usein reaaliaikaisista varjoista puhuttaessa. Ensimmäinen käsite on **näkyvän pinnan algoritmi** (visible-surface algorithm). Tällaisella algoritmilla päätellään renderointiputken geometriavaiheen alussa nopeasti, mitkä pinnat ovat kokonaan katsojalta näkymättömissä. Tällaisten näkymättömien pintojen tarkastelu voidaan seuraavissa vaiheissa kokonaan sivuuttaa ja näin vähentää raskaiden laskutoimitusten määrää renderointiputken loppuosassa. Hieman nimensä vastaisesti poistetaan näkyvän pinnan algoritmilla käsittelystä vain varmasti lopullisessa kuvassa näkymättömät pinnat. Niinpä osa näkyvän pinnan algoritmien maailmaan jättämistä polygoneista saattaa jäädä piilon myöhemmin esim. z-puskurin (ks. luku 2.4.) avulla tehtävissä tarkemmissa näkyvyystarkasteluissa. Näkyvän pinnan algoritmi on yhteisnimitys joukolle toisistaan poikkeavia algoritmeja, joista osaa voidaan käyttää rinnakkain ja osa on keskenään vaihtoehtoisia.

Toinen erityisesti vanhemmassa tietokonegrafiikkaa käsittelevässä kirjallisuudessa esiin tuleva käsite on **Monikulmio-juovamuunnos** (polygon scan conversion). Tämä tarkoittaa geometriavaiheesta tulevien monikulmioiden (=polygon) muuntamista monitorille piirrettäviksi viivoiksi (=scanline). Käytännössä monikulmio-juovamuunnos vastaa renderointiputken rasterointivaihetta, ja sen synonyymina käytetäänkin usein termiä **rasterointi** (rasterization).

2.4. Z- ja sapluunapuskurit

Erilaiset puskurit ovat tietokonegrafiikan tuottamisessa avainasemassa, koska käsiteltävänä on suuri määrä kuvapisteitä, joihin kohdistetaan erilaisia operaatioita. Perinteiset, kaksiulotteiset, 2D-näytönohjaimet sisältävät **väripuskurin** (color buffer), johon muodostettu tieto kuvapisteistä piirrettään monitorin ruudulle. Väripuskurin koko määrittelee käytettävissä olevat resoluutiot ja yksittäisen resoluution enimmäisvärimäärän. Nykyaikaiset 3D-näytönohjaimet sisältävät väripuskurin lisäksi useita erilaisia puskureita, joiden avulla kolmiulotteisen datan käsittelyä voidaan nopeuttaa. Tästä seuraa, että algoritmit, jotka hyödyntävät 3D-näytönohjainten puskureita, saadaan todennäköisesti toimimaan käytännön sovelluksissa nopeammin kuin vastaavat puskureita hyödyntämättömät algoritmit. Yleisesti reaaliaikaiseen tietokonegrafiikkaan tarkoitettuja algoritmeja ja erityisesti saman algoritmin eri toteutuksia tarkasteltaessa tulee kiinnittää huomiota siihen, kuinka hyvin ne käyttävät näytönohjainten ominaisuuksia hyväkseen, sillä tästä voidaan tehdä päätelmiä algoritmien todellisista nopeuksista.

Z-puskuri (Z-buffer) sisältää muodostettavan kuvan syvyystiedon. Tämän vuoksi z-puskurin on oltava vähintään kuvapuskurin kokoinen. Syvyystiedon avulla määritellään, piirretäänkö piste edellisen pisteen päälle vai ei. Mikäli piirrettävän piste on lähempänä katsojaa kuin kuvaan jo piirretty piste, piirretään uusi tilalle.

Sapluunapuskuri (stencil buffer) sisältää tiedon niistä pisteistä, jotka näytönohjaimen tulee piirtää ruudulle. Yksinkertaisimmillaan puskuri sisältää kuvapistettä kohden yhden bitin, joka ilmoittaa, piirretäänkö kyseinen piste. Yleensä käytetään kuitenkin isompia sapluunapuskureita, esimerkiksi kahdeksan bittiä kuvapistettä kohden. Sapluunapuskurin avulla pystytään tekemään kuvaan erilaisia erikoistehosteita. Sapluunapuskurin merkitys z-puskuriin verrattuna on huomattavasti vähäisempi, eikä sitä kaikissa sovelluksissa käytetä. Se on kuitenkin tärkeässä osassa myöhemmin käsiteltävässä varjosärmiöalgoritmin 3D-laitteistolähtöisessä sovelluksessa.

3. Reaaliaikaisia varjostusmenetelmiä

3.1. Yleistä

Puhuttaessa tietokonegrafiikasta englanninkieliset termit **shading** ja **shadowing** menevät helposti sekaisin tai ainakin niiden erot hämärtyvät. Suomeksi termit kääntyvät **sävytykseksi** ja **varjostukseksi** antaen jo paremman kuvan käsiteltävästä asiasta. Sävytyksessä yksittäiselle kappaleelle määritellään sen väritys. Yksinkertaisimmillaan tämä tarkoittaa tasaisia värejä koko kappaleelle. Toisena ääripäänä taas on teksturoitu ja valonlähteiden mukaan sävytetty kappale, joka saattaa näyttää erittäinkin realistiselta ja tunnelmallisesti "varjostetulta".

Vaikka kappaleita pystytäänkin sävyttämään useiden valonlähteiden mukaan ja vieläpä melko nopeasti, ei sävytyksellä saada aikaan kappaleiden välisiä vuorovaikutuksia, eli kappaleet eivät luo varjoja toistensa ylle. Sävytyksellä ei myöskään saavuteta **itsevarjostusta** (self-shadowing) eli kappaleen itseensä luomia varjoja. Tästä esimerkkinä vaikkapa auton sivupeili, joka heittää varjon auton kylkeen.

Varjostustekniikoissa on se ongelma, että vaikka niitä onkin tutkittu jo 1970-luvulta lähtien, eivät niiden nopeudet ole ratkaisevasti parantuneet. Oikeiden (tai oikean suuntaisten) varjojen laskeminen kuvaan vaatii moninkertaisen työn pelkän varjostamattoman kuvan laskemiseen nähden. Näin ollen vasta viime aikoina ovat oikeat varjot tulleet osaksi reaaliaikaista 3D-grafiikkaa, vaikka ei-reaaliaikaisessa tietokonegrafiikassa niitä onkin käytetty jo pitkään.

Syyt varjojen käyttöön tietokonegrafiikassa ovat samat kuin reaalimaailmassakin. Varjot auttavat hahmottamaan kappaleiden muotoja ja sijaintia toisiinsa nähden paremmin. Tietokonegrafiikassa tämä vielä korostuu, kun monitorin kuvasta puuttuu todellinen kolmas ulottuvuus. Lisäämällä tietokoneen muodostamaan kuvaan vielä heijastavat pinnat pystytään vieläkin paremmin määrittelemään kappaleiden sijainnit virtuaalimaailmassa. Heijastukset eivät kuitenkaan kuulu tämän tutkielman aihealueeseen. Toinen syy varjojen käyttöön on tietenkin kuvan tunnelmallisuuden lisääminen. Esimerkiksi syvät varjot täydentävät mielikuvaa synkästä luolastosta.

3.2. Valevarjot

Tässä tutkielmassa on mainittu termi oikea varjo useamman kerran. Aiemmissa luvuissa selvitettiin varjojen fysikaalisia perusteita, ja ns. oikeiksi varjoiksi voidaan laskea sellaiset varjot, jotka on laskettu näiden periaatteiden mukaisesti eli tarkastelemalla valonlähteistä lähtevän valon pääsyä varjostettaville kappaleille varjostavien kappaleiden ohi. Tietokoneohjelmia ei ole sidottu fysikaaliseen maailmaan, joten

varjostusmenetelmien ei tarvitse perustua fysikaalisiin malleihin. Reaaliaikaisen grafiikan tapauksessa on mahdollista käyttää vaihtoehtoisia, usein epätarkempia, varjostusmalleja. Tällaisten mallien käyttö tulee ratkaista sovelluskohtaisesti, sillä fysiikan lakien vastaisesti toimivat varjot kiinnittävät helposti katselijan huomion. Toisaalta jos epätodelliset varjot sopivat grafiikan tyyliin tai varjojen käyttöalue on hyvin rajattu, voi lopputulos olla samanveroinen fysikaalisiin malleihin perustuviin varjoihin verrattuna.

Suosituin tekniikka **valevarjon** (fake shadows) luomiseen on tummemman tekstuurin käyttö,. Tekstuuri sijoitetaan kappaleen alle maata tai lattiaa vasten. Yksinkertaisimmillaan kyseessä on tumma soikio, joka on suoraan varjostavan kappaleen alapuolella. Realistisemmaksi menetelmää voidaan kehittää laskemalla varjotekstuuri etukäteen muistuttamaan kappaletta. Lisäaskel kohti oikeita varjoja on siirtää dynaamisesti varjotekstuurin paikkaa valonlähteen mukaan. Tällöin lähestytään jo projisoitujen varjojen käsitettä, jota käsitellään seuraavassa luvussa.

Toinen oikeiden ja valevarjojen välimaastossa liikkuva varjostustekniikka on dynaamisten kappaleiden sävytys staattisen ympäristön mukaan. Staattisen ympäristön varjot lasketaan etukäteen ja liikkuvien kappaleiden sävytys määrätään sen mukaan, onko niiden ympäristö varjossa vai ei. Yksinkertaisimmillaan tässä tekniikassa kappale sävytetään tummemmilla väreillä, kun sen keskipiste on varjossa olevan ympäristön osan päällä. Käytännössä menetelmän soveltaminen vaatii, että ympäristön ja dynaamisten kappaleiden välillä on jonkinlainen selkeä yhteys. Esimerkiksi voidaan mainita auto, joka kulkee jonkun tietyn maastonkohdan päällä.

Valevarjotekniikoiden ongelma on se, että vaikka kappaleet luovat "varjoja" maahan tai ympäristö varjostaa kappaleita, eivät kappaleet varjosta millään lailla toisiaan. Parhaiten valevarjotekniikat sopivatkin esimerkiksi pelisovelluksiin, joissa ympäristön realistisuus ei ole pääasia. Suurimmassa osassa nykyään julkaistavista kaupallisista pelisovelluksista käytetään kuitenkin todenmukaisempia varjostusmenetelmiä.

3.3. *Projisoidut tasovarjot*

Projisoiduilla varjoilla (projection shadows) varjostava kappale piirretään uudelleen varjostettavalle pinnalle käyttämällä ympäristöä tummempaa väritystä. Menetelmä on varsin yksinkertainen, kun apuna käytetään kuvassa 3 olevaa projektiomatriisia. Matriisiin avulla varjo saadaan projisoitua mielivaltaiselle tasolle. Tämä tapahtuu kertomalla kappaleen kärkipisteet matriisilla. Lopputuloksena saatava levymäinen kappale piirretään varjostettavan tason pinnalle tasaisella ympäristöä tummemmalla värillä, jolloin syntyy vaikutelma varjosta. Myös varjostettavan pinnan värityksen tummentaminen varjon alueelta on mahdollista.

$$M = \begin{pmatrix} \bar{n} \cdot \bar{l} + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \bar{n} \cdot \bar{l} + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \bar{n} \cdot \bar{l} + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \bar{n} \cdot \bar{l} \end{pmatrix}$$

Kuva 3. Matriisi valonlähteen **l** aiheuttaman varjon projisoimiseksi tasolle: $\mathbf{n} \bullet \mathbf{x} + d = 0$ [6]

Menetelmä ei ole täysin ongelmaton. Projisoidun varjon sijoittaminen sopivan matkan päähän varjostettavasta tasosta siten, että se ei z-puskurin tarkkuusvirheen vuoksi sekoitu tasoon tai että se ei näytä leijuvan tason yllä, on vaikeaa. Tämä ongelma voidaan kiertää muokkaamalla piirtojärjestystä. Mikäli valonlähde on varjostettavan tason ja varjostavan kappaleen välissä, aiheuttaa projektiomatriisin käyttäminen virheellisen **vastavarjon** (anti-shadow) muodostumisen varjostettavalle pinnalle. Näin ollen ennen projektiomatriisilla kertomista tulee tutkia, onko valonlähde varjostavan kappaleen ja tason välissä. Myös varjon leikkaaminen tason kokoiseksi vaatii laskuaikaa.

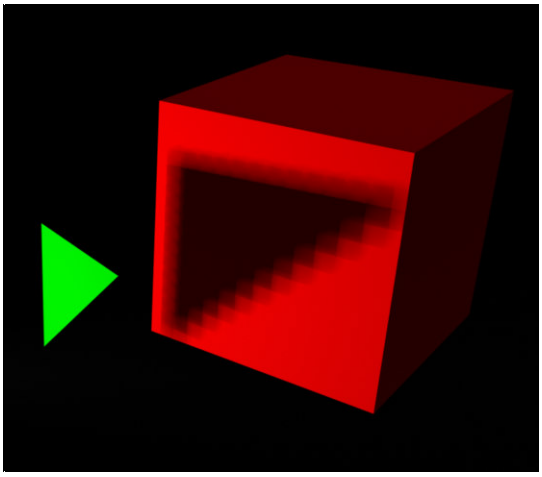
Suurin ongelma on kuitenkin valevarjojakin koskeva vuorovaikutuksen puute muiden kappaleiden kanssa, sillä projisointitekniikalla varjot saadaan luontevasti luotua vain tasomaisiin kappaleisiin ja yleensä tietokonegrafiikassa käytetään monimutkaisempia kappaleita. Projisoidulla varjoilla päästään kuitenkin melko hyvään ja ennen kaikkea nopeaan lopputulokseen tilanteissa, joissa varjostettava kappale on hyvin yksinkertainen. Varjostettavan kappaleen monimutkaistuessa algoritmin suorituskyky laskee nopeasti, sillä varjo pitää projisoida ja leikata jokaista varjostettavan kappaleen polygonia kohden.

3.4. Varjokartta

Varjokartta-menetelmä (shadow map) mahdollistaa oikeiden varjojen piirtämisen reaaliajassa hyödyntämällä z-puskuria. Menetelmässä on keskeistä fysikaalinen tosiasia, että valonlähde ei näe luomiaan varjoja.

Varjojen laskeminen tapahtuu kahdessa vaiheessa. Ensimmäisessä muodostetaan kuva valonlähteestä. Kuvan syvyysarvot tallennetaan normaalisti z-puskuriin. Koska z-puskurin arvoja käytetään seuraavassa vaiheessa varjojen muodostamiseen, kutsutaan tätä z-puskuria varjokartaksi. Seuraavassa vaiheessa muodostetaan kuva havainnoitsijasta päin. Kuvapisteiden syvyysarvoja (suhteutettuna valonlähteeseen) verrataan varjokartan arvoihin. Jotta vertailu voidaan suorittaa, tulee varjokartan valonlähteen suhteen muodostettujen pisteiden koordinaatit muuntaa havainnoitsijan eli

kameran koordinaatistoon. Mikäli piirrettävän pisteen syvyysarvo on lähes sama kuin varjokartan vastaavan pisteen syvyysarvo, näkee valonlähde kyseisen pisteen ja on näin ollen valaistu. Muussa tapauksessa piirrettävä piste on varjossa.



Kuva 4. Varjokartta, jossa on liian alhainen resoluutio, vähentää varjon laatua ja aiheuttaa sen reunaan aliasilmiötä. Aliasilmiö näkyy kuvan varjossa reunan sahalaitaisuutena.

Varjokartta-menetelmä perustuu digitaaliseen näytteenottoon ja tämä aiheuttaa muutamia ongelmia. Ensiksikin varjon laatu riippuu hyvin paljon varjokartan näytteenottotarkkuudesta eli valonlähteestä muodostetun kuvan resoluutiosta. Tähän liittyvä **aliasilmiö** (aliasing) eli liian alhaisen näytteenottoresoluution aiheuttama epäjatkuvuus synnyttää varjon reunaan sahalaitaisuutta. Nämä kaksi ongelmaa voidaan havaita kuvasta 4, jossa varjon muodostamiseen on käytetty hyvin pientä varjokarttaa (vrt. kuva 6). Kolmas ongelma on ns. **aliasilmiöstä johtuva itsevarjostus** (self-shadow aliasing) eli näytteenoton epätarkkuudesta johtuva häiriö, jossa tasainen pinta luo varjoja itseensä. Myös menetelmän luonne itsessään aiheuttaa ongelmia. Varjohan muodostetaan siihen suuntaan, johon valonlähde "katsoo". Niinpä yhden varjokartan käytöllä pystytään käytännössä mallintamaan vain kohdevalaisimen tyyppistä valonlähdettä. Mikäli halutaan käyttää joka suuntaan valaisevaa valonlähdettä eli ympärisäteilevää valonlähdettä, joudutaan tekemään yleensä kuusi varjokarttaa hyvänlaatuisen varjon tuottamiseksi. Varjokartan projektiosta riippuen määrää voidaan pienentää, mutta tämä aiheuttaa varjoon vääristymiä.

Varjokartalla on hyviä puolia. Varjostuksen vaatima laskenta-aika pystytään helposti ennustamaan, sillä menetelmässä piirretään sama kuva kahteen kertaan eri kuvakulmista. Näin ollen algoritmin kompleksisuus kasvaa lineaarisesti kolmiulotteisen maailman monimutkaisuuden mukaan. Varjokartta ei myöskään ole riippuvainen käytetystä kolmiulotteisesta mallista, vaan algoritmi toimii, jos z-puskuri saadaan muodostettua.

3.5. Varjosärmiö

Varjosärmiö-algoritmi (shadow volume) perustuu siihen ideaan, että jokainen varjostava kappale muodostaa kolmiulotteisen särmiön, johon tietyn valonlähteen valosäteen eivät pääse. Tätä algoritmia käsitellään tarkemmin seuraavassa luvussa.

4. Varjosärmiöalgoritmi

Edellä on esitelty lyhyesti useita erilaisia tapoja ja algoritmeja reaaliaikaisten varjojen luomiseen. Ongelma kaikissa esitellyissä algoritmeissa on se, että ne eivät joko pysty luomaan täysin tarkkoja varjoja tai pystyvät siihen vain joissakin tapauksissa. Varjokartta-algoritmi luo varjoja kaikenmuotoisille pinnoille, mutta varjojen reunoista tulee helposti epätarkkoja. Projisoidut varjot ovat alkuperäisen kappaleen muotoisia, mutta niiden käyttöalue on reaaliaikaisessa grafiikassa rajattu tasopinnoille.

Varjosärmiöalgoritmi mahdollistaa tarkat varjot säilyttäen kuitenkin reaaliaikaisuuteen vaadittavan piirtotahdin, kunhan käytettävät mallit pysyvät kohtuullisen kokoisina ja ne on rakennettu suljetuista tasopolygoneista. Reaaliaikaisessa grafiikassa ainakin jälkimmäinen vaatimus on mahdollista täyttää, koska mallit on rakennettu kolmioista. Varjosärmiömenetelmä pystyy laskemaan ympärisäteilevän valonlähteen aiheuttamat varjot ja kappaleiden itsevarjostuksen oletusarvoisesti. Lisäksi varjosärmiöalgoritmista on kehitelty toteutuksia, jotka hyödyntävät vahvasti nykyisten 3D-näytönohjaimien ominaisuuksia.

Monipuolisesta toiminnallisuudesta huolimatta varjosärmiöalgoritmi ei ole kuitenkaan täydellinen ratkaisu reaaliaikaisiin varjostustehtäviin. Ongelmat liittyvät suurelta osin algoritmin käytännön toteutuksiin. Suuremmilla malleilla algoritmi muuttuu yksinkertaisesti liian hitaaksi reaaliaikaisiin sovelluksiin. Toteutuksessa tulee ottaa myös huomioon kameran ja varjon suhteeseen liittyviä erikoistapauksia, sillä muuten algoritmin tuottavat varjot piirtyvät väärin.

Tässä luvussa eritellään Frank Crown esittelemään algoritmin perusideaan [1]. Koska Crown teksti ei juuri ota kantaa algoritmin toteutukseen, esitellään useita vaihtoehtoisia ja ehkä toisiaan tukevia toteutusideoita. Näkökulma pysyy edelleen reaaliaikaisten varjojen muodostamisessa. Luvun lopussa vertaillaan eri toteutustapojen teoreettisia nopeuksia verraten niitä myös edellisessä luvussa esiteltyihin algoritmeihin. Lopuksi pohditaan, onko varjosärmiöalgoritmi varteenotettava vaihtoehto reaaliaikaisten varjojen muodostamiseen.

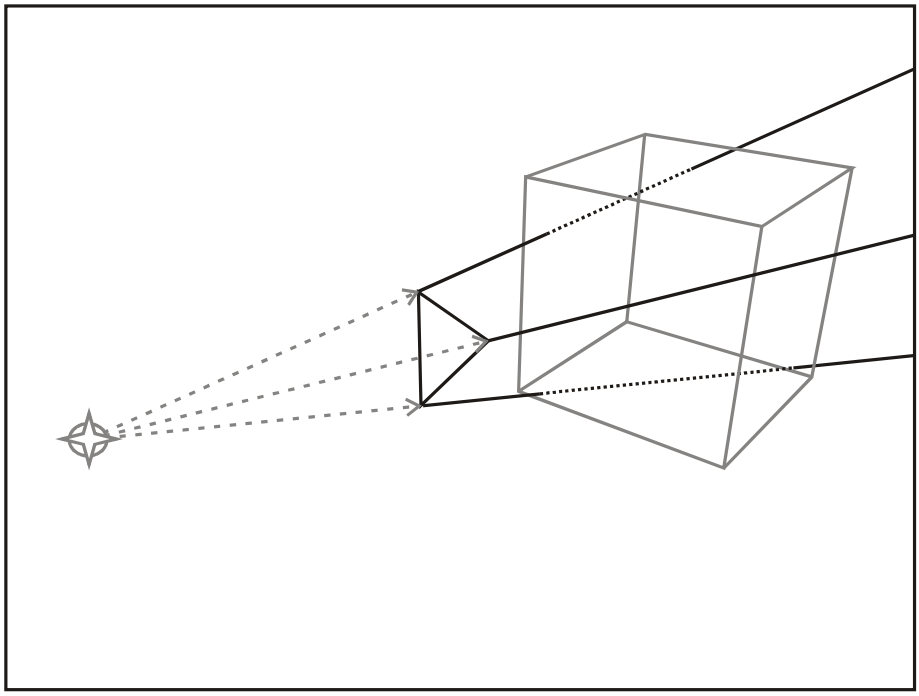
4.1. Algoritmin perusidea

Frank Crow esitteli varjosärmiöalgoritmin periaatteen vuonna 1977 Shadow Algorithms for Computer Graphics -kirjoituksessaan [1]. Algoritmin perusidea on yksinkertainen ja perustuu siihen havaintoon, että jokainen varjostava kappale synnyttää tilavuuden, johon valonlähteen valosäteet eivät pääse. Tätä tilavuutta kutsutaan varjosärmiöksi. Ne kappaleet tai kappaleiden osat, jotka ovat varjosärmiön sisällä, ovat varjossa valonlähteen suhteen.

Varjosärmiön muodostaminen alkaa varjostavan kappaleen **siluetin** (silhouette edge) eli ääriviivojen etsimisellä valonlähteestä katsottuna. Siluetti muodostetaan tarkastelemalla varjostavan kappaleen särmiä. Yhteen särmään liittyy yleensä kaksi polygonia. Ne särmät, joissa toinen näistä polygoneista on valonlähdettä kohti ja toinen valonlähteestä poispäin, kuuluvat varjostavan kappaleen siluettiin. Menetelmä ei toimi siinä tilanteessa, jossa kappale on topologisesti avoin. Tällaiset kappaleet aiheuttavat varjosärmiöalgoritmille myös muita ongelmia. Aihetta käsitellään tarkemmin luvussa 4.2.1.

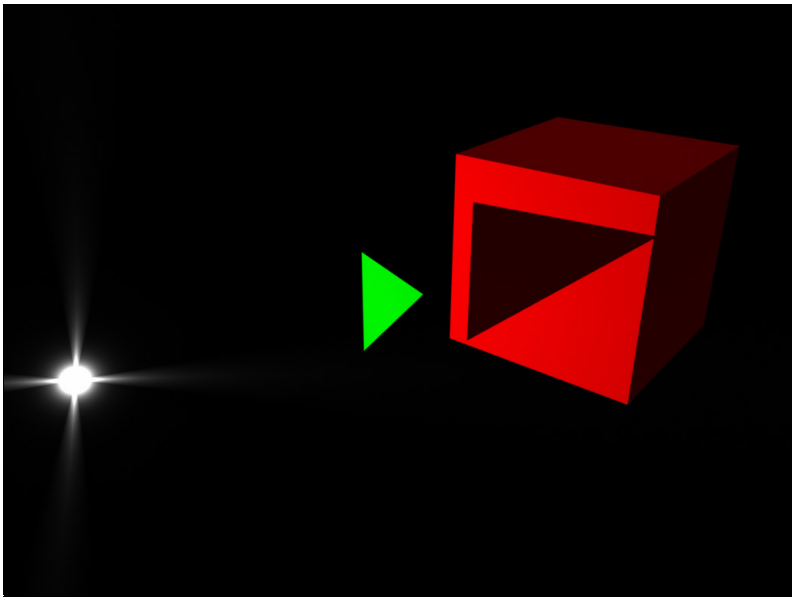
Kappaleen siluetti muodostaa varjosärmiön toisen pään. Siluetin särmien avulla pystytään muodostamaan varjosärmiön sivut. Sivut ovat monikulmioita, jotka muodostuvat siluetin särmästä, tämän särmän kummankin kärkipisteen kautta valonlähteen mukaan projisoiduista kahdesta viivasta sekä näiden viivojen päätepisteistä, jotka voidaan leikata esim. kameran näkemän tilavuuden mukaan. Päätepisteet voidaan jättää avoimiksi, jolloin varjosärmiö jatkuu äärettömyyksiin. Varjosärmiön sivujen ulkopintojen tulee osoittaa ulospäin varjosärmiön keskustasta, jolloin varjosärmiön sisälle muodostuu yhtenäinen tila.

Kun varjosärmiö on muodostettu, arvioidaan seuraavaksi, ovatko varjostettavat kappaleet varjossa. Arviointi tapahtuu laskemalla, kuinka monta varjosärmiön katsojaa kohti ja katsojasta poispäin olevaa ulkosivua kohdataan matkalla havainnoitsijasta kappaleeseen. Mikäli näitä ei ole yhtä paljon, on kappale tai sen tarkasteltava osa varjossa. Jos katsoja on valmiiksi varjossa, tulee tämä ottaa huomioon laskettaessa katsojaa kohti olevien ulkosivujen määrää. Käytännössä tätä määrää siis lisätään niiden varjosärmiöiden lukumäärällä, joiden sisällä katsoja on. Eräs menetelmä oikean aloitusarvon päättelemiseen esitellään luvussa 4.2.1.



Kuva 5. Varjosärmiön perusidea. Kolmio esittää kuvassa varjostavaa kappaletta. Varjostavan kappaleen ääriviivojen ja valonlähteen avulla muodostetaan varjosärmiön sivut. Varjossa oleva osa varjostettavasta kuutiosta jää näin syntyvän varjosärmiön sisälle.

Algoritmin toiminta havainnollistuu kuvan 5 avulla. Kuvassa on valonlähde, kuutio ja sitä varjostava kolmio esitettynä katsojan näkökulmasta. Varjosärmiö muodostetaan kolmion sivujen ja valonlähteen avulla jatkamalla valonsäteitä kolmion kärkipisteistä. Tässä tapauksessa varjosärmiö muotoutuu päästä leikatun pyramidin muotoiseksi: Valonlähde sijaitsee kuvitteellisen pyramidin kärjessä. Särmiö esitetään kuvassa paksummalla mustalla viivalla piirrettynä. Huomataan, että osa kuutiosta jää varjosärmiön sisälle. Tähän osaan kuutiosta ei valonlähde näy, ja se on näin ollen varjossa. Kun kuutiota piirretään, tarkistetaan, onko piirrettävä kuvapiste varjosärmiön sisällä. Mikäli näin on, valonlähde ei vaikuta kyseisen kuvapisteen valoisuuteen, ja se piirretään tummalla värillä. Muulloin kuvapiste väritetään sävytysalgoritmin mukaisesti. Lopputulos vastaa kuvan 6 varjostettua kuutiota.



Kuva 6. Varjostettu kuutio. Kuva on renderoitu 3D-ohjelmalla havainnollistamaan kuvan 5 todellista lopputulosta. Valkoinen kirkas tähti kuvaa pistemäisen valonlähteen paikkaa.

Crow esittelee vain algoritmin perusidean eikä juuri ota kantaa sen toteutukseen. Moniin toteutuksen kannalta oleellisiin kysymyksiin, kuten esim. syvyyslaskurin alustamiseen oikealla arvolla ja laitteiston hyödyntämiseen, on otettu kantaa myöhemmissä aiheesta tehdyissä kirjoituksissa, joita esitellään seuraavassa.

4.2. *Algoritmin toteutustapoja*

4.2.1. Perusteiden ratkontaa

Philippe Bergeron täsmentää 1986 julkaistussa artikkelissa monia varjosärmiöalgoritmin käytännön toteutusta koskevia, mutta Crown avoimeksi jättämiä kysymyksiä[2]. Artikkelissaan hän rakentaa yleistä toteutusta varjosärmiöalgoritmille, joten kaikki artikkelin kohdat eivät ole oleellisia sovellettaessa algoritmia reaaliaikaiseen tietokonegrafiikkaan. Artikkelin ideat syvyyslaskurin aloitusarvon määrittelyyn, **topologisesti avointen** (topologically open) mallien varjostamiseen ja varjosärmiön koon rajoittamiseen soveltuvat kuitenkin myös reaaliaikaiseen grafiikkaan.

Alkuperäinen varjosärmiöalgoritmi olettaa mallien olevan **topologisesti suljettuja** (topologically closed), eli mallin jokaisen polygonin jokainen särmä on kiinni mallin toisessa polygonissa, jolloin mallille muodostuu täysin suljettu ulkopinta. Tällöin syvyyslaskurin arvoa lisätään yhdellä lävistettäessä katsojaan päin olevan varjosärmiön kylki ja vastaavasti vähennetään yhdellä lävistettäessä katsojasta poispäin oleva varjosärmiön kylki. Jos syvyyslaskurin arvo piirrettävän pisteen kohdalla on nolla, ei piste ole varjossa [1]. Bergeron lisää mahdollisuuden käyttää varjostavana kappaleena topologisesti avoimia malleja. Alkuperäistä algoritmia muutetaan siten, että

varjosärmiön sellaiset kyljet, jotka ovat kappaleen äärireunojen jatkeita, aiheuttavat syvyyslaskuriin yhden yksikön muutoksen. Varjosärmiön sellaiset kyljet, jotka ovat kahden polygonin yhteisen särmän jatkeita, aiheuttavat laskuriin puolestaan kahden yksikön muutoksen. Tällä tavoin vältetään alkuperäisen algoritmin ongelmat tilanteessa, jossa varjosärmiön kylkien lukumäärä ei ole kahdella jaollinen [2].

Kun kamera leijailee kolmiulotteisessa maailmassa, saattaa eteen tulla tilanne, jossa kamera on itse varjossa eli varjosärmiön sisällä. Tällöin varjosärmiöalgoritmi ei toimi oikein, jos syvyyslaskuria ei ole alustettu nollasta poikkeavaan arvoon. Esimerkiksi poistuminen varjosta aiheuttaa laskuriin vähennyksen, ja eteen tulevat pisteet tulkitaan virheellisesti varjostetuiksi. Bergeron esittelee menetelmän syvyyslaskurin alustamiseen oikealla arvolla, mikäli kamera on varjossa. Menetelmän toiminnan kannalta oleellista on, että varjosärmiö suljetaan kummastakin päästä. Varjosärmiön loppupäähän projisoidaan varjostavan mallin valonlähteestä poispäin osoittavista polygoneista **capping-malli** (capping model), joka sulkee varjosärmiön. Itse menetelmä perustuu siihen, että ennen varsinaista rasterointivaihetta valitaan piirtoalueelta mikä tahansa piste, josta katsotaan kohtisuoraan kolmiulotteiseen maailmaan. Tämän jälkeen käydään läpi kaikki vastaantulevat mallien ja varjosärmiöiden kyljet ja lisätään nollasta alkavaa laskuria taulukon 1 mukaan. Laskurin lopullisen arvon negaatio määrää syvyyslaskurin aloitusarvon. Jotta menetelmä toimisi oikein, yhdenkään varjosärmiön kylki ei saa leikata piirtoaluetta, sillä tällöin vain osa kameran linssistä on varjossa ja valittu aloituspiste vaikuttaa laskurin arvon määräytymiseen.

Taulukko 1. Bergeronin malli syvyyslaskurin alkuarvon laskemiseen. Laskeminen suoritetaan tarkastelemalla maailmaa mielivaltaisesta piirtoalueen pisteestä kohtisuoraa viivaa pitkin. Syvyyslaskurin nollasta alkavaa arvoa muutetaan taulukon ilmoittaman lisäyksen verran sen mukaan, minkä tyyppisiin kylkiin törmätään ja miten kyljet suhtautuvat kameraan ja valonlähteeseen. Kyljet jaetaan itse mallien kylkiin, malleista laskettujen varjosärmiöiden sivukylkiin ja varjosärmiön sulkevan capping-mallin kylkiin.

<table>
<tr>
<td>
1) Näkymän mallien kyljet
Jos kylki on kohti valonlähdettä
Jos kylki on kohti kameraa
Lisäys = 1
Muuten (Poispäin kamerasta)
Lisäys = -1
Muuten (Poispäin valonlähteestä)
Jos kylki on kohti kameraa
Lisäys = -1
Muuten (Poispäin kamerasta)
Lisäys = 1
</td>
<td>
2) Varjosärmiöiden sivukyljet
(Valonlähde ei vaikuta tässä tapauksessa)
Jos kylki on luotu mallin kyljen avoimesta särmästä
Jos kylki on kohti kameraa
Lisäys = 1
Muuten (Poispäin kamerasta)
Lisäys = -1
Muuten (Luotu mallin kahden kyljen jakamasta särmästä)
Jos kylki on kohti kameraa
Lisäys = 2
Muuten (Poispäin kamerasta)
Lisäys = -2
</td>
</tr>
<tr>
<td colspan="2">
3) Varjosärmiön lopetuskyljet eli
capping-mallin kyljet
Jos kylki on kohti valonlähdettä
Jos kylki on kohti kameraa
Lisäys = -1
Muuten (Poispäin kamerasta)
Lisäys = 1
Muuten (Poispäin valonlähteestä)
Jos kylki on kohti kameraa
Lisäys = 1
Muuten (Poispäin kamerasta)
Lisäys = -1
</td>
</tr>
</table>

Alkuperäisessä Crown varjosärmiöalgoritmissa varjosärmiö oletetaan äärettömän pituiseksi. Käytännössä varjosärmiö pitää kuitenkin sulkea, koska monet algoritmin toteutustavat tätä vaativat. Esim. edellä esitelty varjolaskurin aloitusarvon laskeva menetelmä ei toimi oikein, mikäli varjosärmiö on pohjasta avonainen. Menetelmän yhteydessä varjosärmiön sulkemiseen käytettyä capping-mallia käytetään yleisesti. Sulkemiskohdan määrittämiseksi Bergeron ottaa käyttöön valomallin, jossa valon intensiteetti pienenee päätyen lopulta nollaksi tietyn säteen päässä valon keskipisteestä [2]. Säteen sisäpuolella olevista kappaleista muodostetut varjosärmiöt voidaan sulkea kyseisen säteen etäisyydellä valonlähteestä. Tämän säteen ulkopuolella olevat kappaleet eivät luo varjoja, joten niille ei tarvitse muodostaa omaa varjosärmiötä.

Varjosärmiöiden pituutta voidaan rajata myös kameran näkemän alueen mukaisesti [6]. Tämä on luonnollinen rajausmenetelmä, sillä myös kolmiulotteisessa kuvassa varsinaisesti näkyvät kappaleet rajataan usein tällä tavalla **leikkaustasojen** (clipping plane) avulla, ja näin ollen samoja funktioita voidaan käyttää varjosärmiön sivujen leikkaukseen. Leikkaustasot noudattavat kameran sivu-, ylä- ja alareunoja. Lisäksi yleensä käytetään etu- ja takaleikkaustasoja, jotka poistavat liian lähellä tai liian kaukana kamerasta olevat maailman osat. Tämä aiheuttaa kuitenkin ongelmia varjosärmiöalgoritmien toteutuksissa (esim. Heidmannin menetelmä, ks. 4.2.4), sillä leikkaustasot saattavat katkaista varjosärmiön sivut liian aikaisin ja jättää varjosärmiön avoimeksi, jolloin varjo piirtyy virheellisesti tai jää kokonaan piirtymättä [9]. Varjon

virheellinen piirtyminen liittyy tilanteisiin, joissa etu- tai takaleikkaustaso katkaisee varjosärmiön osittain. Ongelmaa on yritetty kiertää projisoimalla capping-mallia näille tasoille, mutta täysin yhtenäisen projektion muodostaminen ei ole triviaali tehtävä, joten kaikkien virheiden poistaminen varjosta on haastavaa.

4.2.2. BSP-puutoteutus

Varjosärmiöalgoritmi vaatii monia laskutoimituksia päätelläkseen, mitkä osat kappaleista ovat varjossa ja mitkä eivät. Chin ja Feiner esittelevät **BSP-puuhun** (binary space partitioning tree) perustuvan menetelmän, jossa tämä laskutehoa vievä tieto tallennetaan ennalta [4].

BSP-puu on binaarinen hakupuu, josta voidaan nopeasti hakea tietoja maailman polygonien keskinäisistä suhteista. Puu muodostetaan aloittamalla tarkastelu jostakin maailman polygonista ja asettamalla tämä polygoni puun juureksi. Seuraavat polygonit lisätään joko puun vasempaan tai oikeaan haaraan sen mukaan, ovatko ne puuhun aiemmin asetetun polygonin kautta kulkevan tason edessä vai takana. Mikäli tämä taso menee puuhun lisättävän polygonin läpi, halkaistaan polygoni tason mukaisesti kahtia ja puoliskot asetetaan puun eri haaroihin.

Chin ja Feiner käyttävät menetelmässään BSP-puusta erikoistettua **SVBSP-puurakenetta** (shadow volume BSP) [4]. Puun solmuihin tallennetaan **varjotasot** (shadow planes) eli varjosärmiön reunasivut. SVBSP-puussa vasempaan haaraan sijoitetaan solmun varjosärmiön sisälle jäävät polygonit (”in”-solmu) ja oikeaan haaraan varjosärmiön ulkopuolelle jäävät polygonit (”out”-solmu).

Itse algoritmi järjestää ensin maailman polygonit siten, että lähimpänä valonlähdettä oleva polygoni käsitellään ensin ja sitten edetään järjestyksessä kohti kauimmaista polygonia; järjestelyyn voidaan käyttää alkuperäistä BSP-puuta. Järjestely tehdään siksi, että mielivaltaisesti valittua polygonia voivat varjostaa vain lähempänä valonlähdettä sijaitsevat polygonit. Näin vältetään tilanne, jossa pitäisi palata tarkastelemaan jo käsiteltyjä polygoneja.

Seuraavaksi SVBSP-puu alustetaan tyhjällä ”out”-solmulla. Tämän jälkeen aletaan käsitellä maailman polygoneja sijaintijärjestyksessä. Polygonien suhdetta vertaillaan puussa jo oleviin varjotasoihin. Mikäli polygoni päätyy ”out”-solmuun, voidaan päätellä sen olevan valossa. Tällöin merkataan polygoni valaistuksi ja muodostetaan sen aiheuttamat varjotasot käytetyn valonlähteen suhteen ja tallennetaan nämä solmuun. Mikäli polygoni päätyy ”in”-solmuun, on se kokonaan varjossa. Tällöin varjotasoja ei tarvitse muodostaa ja tallentaa puuhun, koska aiemmin käsiteltyjen polygonien varjotasot kattavat jo polygonin (teoreettisesti) muodostaman varjosärmiön. Polygoni merkataan varjostetuksi. Kolmantena tapauksena polygoni voi joutua osittain jonkin solmun varjotasojen sisään. Tällöin polygoni halkaistaan varjotasojen mukaisesti kahteen osaan ja kummallekin suoritetaan edellä kuvattu tarkastelu.

Kun kaikki polygonit on käyty läpi, voidaan SVBSP-puu poistaa muistista. Menetelmän soveltamisen lopputulokseksi saadaan maailma, jossa alkuperäisen maailman polygonit ovat paloiteltuina varjostettuihin ja varjostamattomiin. Menetelmää voidaan käyttää myös usean valonlähteen kanssa pehmeiden varjojen muodostamiseen. Tällöin edellisen valonlähteen mukaan luotu maailma annetaan syötteenä uudelle käsittelykierrokselle, joka toteutetaan uudelleenalustetun SVBSP-puun avulla. Usean valonlähteen menetelmässä jokaisen valaistun polygonin täytyy säilyttää tieto siitä, mitkä valonlähteistä sitä valaisevat.

Chinin ja Feinerin BSP-puuhun perustuvaa varjosärmiöalgoritmia voidaan soveltaa käytännössä lähinnä staattisen ympäristön varjostuksen toteuttamiseen, sillä SVBSP-puu pitää laskea aina maailman muuttuessa, ja tämä on liian hidasta reaaliaikaisiin sovelluksiin (ks. 4.3). Näin ollen liikkuvien kappaleiden aiheuttamat varjot pitäisi laskea jollain muulla menetelmällä. BSP-puihin perustuvan varjosärmiöalgoritmin käytännön merkitystä vähentää myös se, että staattisen ympäristön varjojen laskemiseen on olemassa menetelmiä, jotka tarjoavat paremman ajonaikaisen suorituskyvyn, esim. varjojen tallentaminen suoraan tekstuureihin.

4.2.3. Pehmeät varjot syvyyspuskurin avulla

Suurin ongelma tässä tutkielmassa esitellyissä varjosärmiöalgoritmin toteutusideoissa on keskittyminen vain yhden tai korkeintaan muutaman valonlähteen tuottamien varjojen tarkasteluun. Jotta varjosärmiöitä voitaisiin myös jatkossa käyttää mahdollisimman realististen kuvien aikaansaamiseen, tulisi myös ei-pistemäisten valonlähteiden aikaansaamia pehmeitä varjoja pystyä mallintamaan.

Brotman ja Badler esittelevät erään menetelmän, joka kykenee muodostamaan pehmeitä varjoja suoraan varjosärmiöalgoritmin perusideaa hyödyntäen [3]. Keskeistä menetelmässä on laajennettu syvyyspuskuri, joka sisältää kutakin ruudulle piirrettävää pistettä kohden syvyystiedon lisäksi linkitetyssä listassa tiedot kunkin valonlähteen luomien varjosärmiöiden lukumäärästä ja varjosärmiöiden pisteeseen aiheuttamasta varjosta. Yksittäinen valonlähde voi menetelmässä olla mielivaltaisen muotoinen monitahkoinen suljettu polygoni. Valonlähdekappaleen pinta approksimoidaan useilla pistemäisillä valonlähteillä. Näin syntyvä valonlähde mallintaa alkuperäistä valonlähdettä, kunhan vain pistemäisten valonlähteiden asettelussa käytettävä algoritmi ja pistemäisten valonlähteiden määrä on tarkoituksenmukainen. Mahdollisimman realistisen lopputuloksen saavuttamiseksi Brotman ja Badler suosittelevat tasaisen satunnaisen peitteen luomista valonlähdekappaleen pinnalle.

Itse algoritmi toimii laskemalla normaalisti kuvaan piirrettävien pisteiden tiedot kuva- ja laajennettuun syvyyspuskuriin. Tämän jälkeen aletaan laskea varjostavien kappaleiden kuvaan luomia varjoja kappale kerrallaan. Jokaista valonlähteen pistettä kohden muodostetaan varjosärmiö ja tarkastetaan, mitkä maailman pisteet jäävät

särmiön sisälle. Näiden pisteiden kappaletason **hämäryysarvoa** (darkness level) lisätään yhdellä. Hämäryysarvo kertoo, kuinka moni yhden valonlähteen pisteistä lähtevä valonsäde ei saavuta piirrettävää pistettä. Kun kaikki valonlähteen pisteet on käyty läpi, tarkistetaan, onko kappale mahdollisesti peittänyt useampia valonlähteen pisteitä kuin aiemmat kappaleet. Mikäli näin on, asetetaan kyseisten pisteiden hämäryysarvoksi äsken laskettu uusi arvo. Lopuksi yhdistetään eri valonlähteiden luomat hämäryysarvot ja muodostetaan näiden arvojen avulla lopullinen kuva muokkaamalla kuvapisteiden väriarvojen intensiteettiä.

Menetelmä soveltaa varsin suoraviivaisesti varjosärmialgoritmia pehmeiden varjojen luomiseen. Käytännössä se vain simuloi isompaa valonlähdettä usean pistemäisen valonlähteen avulla ja toistaa alkuperäistä algoritmia riittävän monta kertaa. Näin ollen pehmeiden varjojen hinta on moninkertainen yksittäiseen pistemäiseen valonlähteeseen verrattuna, koska menetelmällä joudutaan laskemaan varjosärmiö jokaiselle varjostavalle kappaleelle jokaista pistemäistä valonlähdettä kohden.

Brotman ja Badler osoittavat, että varjosärmiöalgoritmin avulla on mahdollista piirtää pehmeitä varjoja. Laskentakustannukset ovat kuitenkin ainakin edellä mainitulla perusalgoritmilla niin isot, että sovellukset reaaliaikaisessa grafiikassa eivät ole ainakaan nykyisillä konetehoilla mahdollisia. Brotmanin ja Badlerin algoritmia voidaan kuitenkin kehittää reaaliaikaiseen grafiikkaan paremmin soveltuvaksi mm. optimoimalla laskettavien varjosärmiöiden määrää. Menetelmän käyttämän muistin määrää voidaan myös pienentää. Muokkaamalla pisteiden väriarvojen intensiteettiä kunkin valonlähteen tarkastelun jälkeen ei laajennettua syvyyspuskuria tarvita vaan hämäryysarvot voidaan tallentaa yksittäisiin muuttujiin.

4.2.4. Laitteistolähtöinen ratkaisu

Nopeaa reaaliaikaista grafiikkaa tuotettaessa erityisesti renderointiputken rasterointivaihe tulisi jättää tarkoitukseen erityisesti suunnitellun laitteiston huoleksi [6]. Suurin osa nykyisistä 3D-näytönohjaimista hallitseekin juuri tämän osavaiheen erittäin hyvin. Tuntuisi siis loogiselta ratkaisulta tehokkaan varjoalgoritmin aikaansaamiseksi siirtää mahdollisimman suuri osa varjojen piirtämiseen liittyvästä laskennasta tähän työvaiheeseen. Tim Heidmann tarjoaa vahvasti 3D-näytönohjainten tekniikkaa hyödyntävän lähestymistavan varjojen muodostamiseen varjosärmiöalgoritmin avulla [5]. Algoritmi käyttää hyväkseen 3D-laitteiston eri puskureita (ks. luku 3.4.), jotka ohjaavat kuvan muodostusta ruudulle rasterointivaiheen aikana.

Itse algoritmi on melko yksinkertainen. Ensin piirretään kuva vain taustavalolla valaistuna väripuskuriin, joka siis sisältää tiedot kuvan kuvapisteiden väriarvoista. Samalla lasketaan syvyyspuskuriin kuvassa näkyvien pisteiden syvyysarvot.

Seuraavaksi lasketaan tietyn valolähteen aiheuttamat varjosärmiöt sivu kerrallaan. Väri- ja syvyyspuskureita ei muuteta, vaan käytetään hyväksi sapluunapuskuria. Vertailemalla piirrettävien pisteiden ja laskettavan varjosärmiön sivun syvyysarvoja muutetaan sapluunapuskurin tilaa siten, että pisteen jäädessä katsojaa kohti olevan varjosärmiön sivun taakse puskurin arvoa kyseisen pisteen kohdalla lisätään yhdellä ja pisteen jäädessä katsojasta poispäin olevan sivun taakse vastaavasti vähennetään yhdellä. Lopuksi sapluunapuskurin arvon pitäisi olla nolla niiden pisteiden kohdalla, jotka eivät jää varjosärmiön sisään ja ovat valaistuina. Kuva piirretään uudelleen niiden pisteiden osalta, jotka ovat sapluunapuskurin mukaan valossa, käyttäen valaistukseen käsiteltävänä olevaa valonlähdettä.

Nollaamalla sapluunapuskuri, piirtämällä varjosärmiöt uudelleen eri valonlähteen suhteen ja lisäämällä tämän valonlähteen vaikutukset väripuskuriin valaistuilla alueilla saadaan mallinnettua mielivaltaista määrää valonlähteitä. Isomman valonlähteen luomat pehmeät varjot saadaan aikaan muuntamalla isompi valonlähde useiksi pistemäisiksi valonlähteiksi (ks. 4.2.3.) ja käyttämällä em. menetelmää.

Heidmannin menetelmä mahdollistaa teoriassa tarkkojen reaaliaikaisten varjojen piirtämisen jopa tavallisella PC-koneella, koska se käyttää hyväkseen nykyisin PC-koneen vakiovarusteisiin kuuluvaa 3D-näytönohjainta. Heidmannin esittämä algoritmi ei kuitenkaan ole täysin ongelmaton. Maailmoissa, joissa etu- tai takaleikkaustaso leikkaa varjosärmiön, eivät syvyysarvot muodostu oikein ja tuloksena on virheellisesti varjostettu kuva. Everitt ja Kilgard ovat kehittäneet sapluunapuskuriin perustuvan menetelmän, jossa tämä ongelma on ratkaistu [9]. Menetelmän tarkempi esittely kuitenkin sivuutetaan tässä tutkielmassa.

4.2.5. Reaaliaikaiset pehmeät varjot

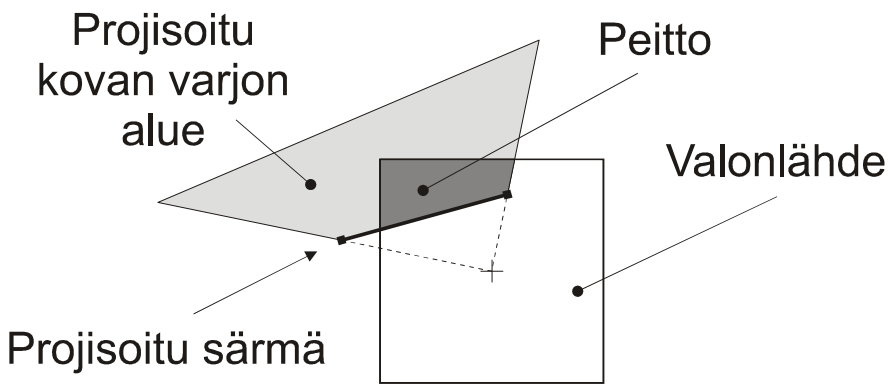
Tämän tutkielman kirjoitushetkellä yksi tuoreimmista varjosärmiöalgoritmeista esittelee menetelmän reaaliaikaisten pehmeiden varjojen piirtämiseen varjosärmiöalgoritmin avulla [8]. Algoritmi perustuu osittain edellä esiteltyyn Heidmannin laitteistolähtöiseen ratkaisuun.

Erotuksena aiemmin esiteltyyn Brotmanin ja Badlerin (ks.4.2.3) menetelmään, jossa laskettiin pisteeseen pääsemättömien valonsäteiden määrää, tämä algoritmi tarkastelee, kuinka suuren osan valonlähteestä varjostetun alueen kukin piste ”näkee”. Tätä tarkoitusta varten esitellään vielä yksi puskurityyppi: **näkyvyyspuskuri** (V-buffer). Kunkin pisteen valomäärä tallennetaan puskuriin **näkyvyyskertoimena** (visibility factor). Mikäli piste ei ole lainkaan varjossa, näkyvyyskerroin saa arvon yksi, kun taas täysin varjossa oleva piste saa kertoimeksi nolla. Osittain varjostettu piste saa arvon nollan ja yhden väliltä sen mukaan, kuinka suuri osa valonlähteestä pistettä valaisee.

Jotta näkyvyystarkastelua ei tarvitsisi tehdä jokaista ruudun pistettä kohden, algoritmi rajaa tarkasteltavaa aluetta ennen maailman piirtämistä ruudulle. Rajaus tapahtuu muodostamalla valonlähteen ja varjostavan kappaleen siluetin avulla alue, jonka sisälle puolivarjo jää. **Puolivarjon tarkan tilavuuden** (exact penumbra volume) muodostaminen ei ole kuitenkaan tehtävän monimutkaisuuden vuoksi reaaliaikaisessa sovelluksessa järkevää, vaan laskennallisesti tehokkaampi tapa on arvioida puolivarjon alue todellista isommaksi ja suorittaa rajattu määrä ylimääräisiä pistekohtaisia tarkasteluja. Algoritmi esittelee **puolivarjolohkon** (penumbra wedge), joka nopeuttaa laskemista vaikuttamatta kuitenkaan lopulliseen varjoon. Jokaisen varjostavan kappaleen siluetin särmän suhteen muodostetaan yksi puolivarjolohko. Puolivarjolohkot muodostetaan toisistaan riippumatta, joten ne voivat mennä osittain toistensa päälle.

Pehmeän varjon piirtäminen tapahtuu kahdessa vaiheessa. Ensimmäisessä vaiheessa maailma piirretään näkyvyyspuskuriin käyttäen alkuperäistä varjosärmiöalgoritmia valonlähteen keskipisteen suhteen. Tämän vaihe piirtää täysvarjon todellista isommaksi. Vaihe on kuitenkin oleellinen, jotta täysvarjon alue piirtyy kuvaan oikein.

Toisessa vaiheessa piirretään näkyvyyspuskuriin puolivarjolohkojen sisältämät alueet piste kerrallaan ja muodostetaan näkymän puolivarjot. Kutakin tarkasteltavaa pistettä kohden projisoidaan puolivarjolohkon muodostaneen siluetin särmä valonlähteeseen tarkasteltavan pisteen suhteen. Projisoidun särmän ja valonlähteen keskipisteen avulla voidaan laskea särmän aiheuttama **peitto** (coverage) valonlähteeseen kuvan 7 mukaisesti ja päivittää tarkasteltavan pisteen näkyvyyskerrointa saadun tuloksen verran. On huomioitavaa, että peiton suuruuteen vaikuttaa vain valonlähteen päälle osuva projisoidun särmän osa. Päivityksen suunta (lisäys vai vähennys) riippuu siitä, onko tarkasteltava piste em. siluetin särmän mukaan lasketun kovan varjon määrittelevän varjosärmiön sivun sisä- vai ulkopuolella; sisäpuoli on se puoli, joka sisältää kovan varjon. Tällä varmistetaan paitsi ensimmäisessä vaiheessa yliarvioidun täysvarjon pienentyminen myös useampaan puolivarjolohkoon kuuluvien pisteiden piirtyminen oikein.



Kuva 7. Peiton arvioiminen valonlähteeseen projisoidun särmän avulla. Särmä projisoidaan valonlähteeseen tarkasteltavan pisteen ja särmän päätepisteiden avulla. Koska särmä on osa varjostavan mallin siluettia, vastaa valonlähteen keskipisteen ja projisoidun särmän päätepisteiden avulla laskettu alue valonlähteeseen projisoitua kovan varjon aluetta (kuvassa harmaa alue). Varsinainen peitto on kovan varjon alueen ja valonlähteen alueen leikkaus (kuvassa tumman harmaa alue). Kuvaa tarkastelemalla huomataan, että peitto riippuu vain projisoidun särmän valonlähteen alueelle osuvasta osasta.

Kunkin puolivarjolohkon vaikutus pisteen näkyvyyskertoimeen riippuu vain puolivarjolohkon synnyttäneen särmän projektiosta valonlähteeseen ja tällöinkin vain suoraan valonlähteen päälle osuvasta särmän osasta. Näin ollen voidaan etukäteen laskea neliulotteinen taulukko, josta näkyvyyskerroin voidaan ajonaikana suoraan lukea. Kun taulukossa määrätään kullekin värille näkyvyyskerroin, mahdollistaa menetelmä moniväriset valonlähteet. Teoriassa valonlähteenä voitaisiin käyttää bittikarttakuvaa tai jopa animoituja tekstuureja.

Algoritmin ohjelmistopohjainen toteutus on pystynyt tekijöidensä mukaan piirtämään 512 x 512 -kokoisen tason päällä leijuvaa varjostettua ihmishahmoa esittävän kuvan kuudessa sekunnissa 1,7 GHz:n Pentium 4 -prosessorilla varustetulla PC-koneella. Tekijät arvelevatkin, että algoritmin 3D-näytönohjainta käyttävä toteutus pystyisi tuottamaan pehmeästi varjostettuja kolmiulotteisia maailmoja reaaliaikaisesti.

Vaikka algoritmi onkin mahdollisesti tarpeeksi nopea reaaliaikaisiin sovelluksiin, ei sen piirtojälki ole vielä täysin virheetöntä. Vaikka tekijät ovatkin parantaneet algoritmin vakautta tekemällä puolivarjolohkoista täysin toisistaan riippumattomia, on tämä luonut myös uusia ongelmia. Yksittäinen piste saattaa sijaita useassa puolivarjolohkossa, ja sen näkyvyyskerrointa joudutaan näin muokkaamaan useita kertoja, jolloin piirtovirheen mahdollisuus kasvaa. Lisäksi algoritmin lopputulos ei ole fysikaalisen tarkka, sillä varjostettavan kappaleen siluettireuna muodostetaan valonlähteen keskipisteen mukaan koko valonlähteen sijaan.

4.3. Reaaliaikaisten varjostusmenetelmien vertailua

Tietokonegrafiikassa käytettyjä erilaisia varjostustapoja on vertailtu kattavasti Woon, Poulinin ja Fournierin tekemässä katsauksessa vuodelta 1990. Katsauksessa he ovat tarkastelleet erityisesti algoritmien kompleksisuutta ja muistintarvetta. [7]

Katsauksesta käy ilmi, että varjokartta-algoritmi on varjosärmiöalgoritmia nopeampi varjon piirtämisessä. Sen sijaan ennen varsinaista piirtämistä kummankin esivalmistelut voidaan laskea lineaarisessa ajassa, jos varjokartan koko katsotaan vakioksi. Esivalmistelujen lisäksi varjonkartan koko vaikuttaa varjokartta-algoritmin tapauksessa tarvittavan muistin määrään. Katsaus ei suoraan käsittele projektiovarjoja, mutta kyseessä on käytännössä varjostavan kappaleen piirtäminen maailmaan toiseen kertaan, joten kompleksisuus riippuu vain varjostavan kappaleen monimutkaisuudesta, jos varjo piirretään yhdelle tasolle. Projektiovarjomenetelmän kompleksisuus kasvaa nopeasti, mikäli varjostettava kappale on tasoa monimutkaisempi, sillä projektiotarkastelu joudutaan suorittamaan varjostettavan kappaleen jokaiselle polygonille. Valevarjojen kompleksisuutta on vaikeaa arvioida, koska kyseessä ei ole mikään yhtenäinen menetelmä. Yksinkertaisimmilla valevarjoilla ei nimensä mukaisesti ole mitään riippuvuutta varjostavaan kappaleeseen, joten kompleksisuudet voidaan arvioida vakioaikaisiksi.

Laskennallisesti nopein algoritmi ei välttämättä aina ole paras mahdollinen, vaan myös ei-mitattavissa olevat piirteet voivat vaikuttaa varjostusmenetelmän valintaan. Nopeuden ja muistinkulutuksen lisäksi tulee miettiä mm. varjojen tarkkuutta, varjostettavan ympäristön muotoja, varjostavan kappaleen muotoja, tarvetta pehmeille varjoille ja useiden valonlähteiden huomioon ottamista. Kaikki algoritmit eivät välttämättä pysty täyttämään kaikkia näitä laadullisia tarpeita. Lisäksi eteen voi tulla puhtaasti subjektiivisia valintoja, koska vaikkapa hieman epätarkat varjokartat voivat joissain tilanteissa miellyttää silmää tarkkaa varjosärmiöllä toteutettua varjoa enemmän.

Eri varjostusmenetelmiä on vertailtu taulukossa 2. Taulukossa n tarkoittaa yksittäisten monikulmioiden määrää piirrettävässä maailmassa, E särmien keskimääräistä määrää maailman kappaleiden monikulmioissa (E on vakio reaaliaikaisissa sovelluksissa, sillä kolmioissa on aina kolme särmää), p varjokartan poikkisuuntaista resoluutiota ja q varjokartan pystysuuntaista resoluutiota. Muistintarve kuvaa varjonmuodostuksessa tarvittavan muistin määrää, esivalmistelujen kompleksisuus kuvaa ajonaikaisten esivalmistelujen vaatimaa aikaa ja kompleksisuus itse varjon piirtämiseen kuluvaa aikaa. Yhden ruudun piirtämiseen vaadittu kokonaisaika saadaan siis näiden kahden arvon yhteistuloksena. Valevarjoja ja projisoitua tasovarjoa lukuun ottamatta arvot on otettu mainitusta katsauksesta [7]. Taulukon muut kentät on täytetty tässä tutkielmassa aiemmin ilmikäyneiden seikkojen nojalla.

Taulukko 2. Reaaliaikaisten varjoalgoritmien vertailua. n on maailman yksittäisten monikulmioiden määrä, E särmien keskimääräinen määrä monikulmioissa, p varjokartan vaakasuuntainen resoluutio ja q varjokartan pystysuuntainen resoluutio.

Menetelmän nimi	Laadulliset ominaisuudet	Muistintarve	Esivalmistelujen kompleksisuus	Kompleksisuus
Valevarjot	- Varjo ei noudata fysiikan lakeja - Nopein varjostusmenetelmä	O(1)	O(1)	O(1)
Projisoidut tasovarjot	- Varjo voidaan laskea järkevästi vain tasopinnoille	O(En)	O(1)	Tasopinnalle: O(En) Muuten: $O(En^2)$
Varjokartta	- Useaan suuntaan säteilevän valonlähteen mallintamiseen tarvitaan käytännössä useampi varjokartta - Varjostettavan kappaleen muoto ei ole oleellinen - Näytteenotosta johtuen epätarkka varjo	O(pq)	O(Enpq)	O(1)
Varjosärmiö (perustoteutus)	- Fysikaalisesti tarkat varjot - Kappaleiden muoto asettaa rajoituksia algoritmin soveltuvuuteen	O(En)	O(En)	O(En)
Varjosärmiö (Bsp-puutoteutus)	- Samat kuin perustoteutuksessa - Nopea varjostus, jos kappaleet staattisia	O(En)	$O(En^2)$	O(log(En))

5. Reaaliaikaisten varjojen käytännön sovelluksia

Reaaliaikaisia varjoja hyödynnetään eniten viihdeteollisuudessa. Erityisesti tietokonepelien grafiikassa varjoja on alettu käyttää viime vuosina. Vaikka monet peleistä hyödyntävätkin valmiiksi laskettuja ja valevarjoja, on yhä useammassa pelissä mahdollisuus valita käyttöön mm. varjokartta-algoritmilla toteutettuja monimutkaisempia varjoja. Varjojen käyttö peleissä on toisaalta pelkkä visuaalinen efekti, jolla pyritään luomaan näyttävämpiä ja realistisemman oloisia pelimaailmoja. Toisaalta varjot auttavat pelaajaa hahmottamaan sijaintiaan nopeatempoisissa 3D-peleissä. Joissakin peleissä varjoista on myös tehty pelaajan hahmon kätkevä pelillinen elementti.

Kuva 8. Splinter Cell. Liikkuva pelihahmo aiheuttaa varjoja ympärilleen. Pelimaailman esineet eivät varjosta vain staattisia kappaleita vaan myös pelihahmon. Kuvassa verkkoaita luo varjon pelihahmon ylle. Huomattavaa on kuvan oikeassa alalaidassa asesymbolin yläpuolella oleva mittari, joka osoittaa, miten hyvin pelihahmo on varjossa ja vihollisten huomaamattomissa. (© Ubisoft)

Esimerkkinä voidaan mainita kirjoitushetkellä ehkä parhaiten varjot mallintava tietokonepeli Splinter Cell. Pelissä pelaajan tulee hiiviskellä läpi realististen kenttien pysytellen mahdollisimman paljon varjossa, sillä paljastuminen tekoälyn ohjaamille vastustajille saattaa johtaa yksittäisen tehtävän epäonnistumiseen. Pelaaja puolestaan voi havaita vastustajat ilman suoraa näköyhteyttä tarkkailemalla näiden seinille ja

muille pinnoille heittämiä varjoja. Pelistä otetusta esimerkkikuvasta (kuva 8) kannattaa huomioida, että pelkästään pelaajan hahmo ei luo reaaliaikaisia varjoja ympäristöön, vaan myös ympäristö muodostaa varjoja (verkkoaidan varjo pelihahmon ja tämän takana olevan koneen yllä).

Toinen näkyvä viihdeteollisuuden haara elokuvateollisuus näyttävine tietokone-efekteineen ei sinällään voi suoraan hyödyntää reaaliaikaisia varjoja, sillä elokuvien ns. CGI-kohtaukset (Computer Generated Imagery) lasketaan luonnollisesti etukäteen ja siirretään filmille ennen niiden esittämistä. CGI-animaatioiden tekemiseen käytettävien mallinnusohjelmien käyttöä reaaliaikaiset varjot sen sijaan tehostavat, kun ohjelman käyttäjä voi reaaliaikaisesti katsella karkeata versiota tuotoksestaan varjojen kera.

Samankaltaisia mallinnusohjelmia voidaan käyttää myös hyödyllisempien sovellusten suunnitteluun. Nämä ns. CAD-ohjelmat (Computer Aided Design) ovat laajasti insinöörien ja muiden suunnittelijoiden käytössä alasta riippumatta. 3D-mallinuksen avulla pystytään suunniteltua tuotetta tarkastelemaan useista kuvakulmista ja jopa sijoittamaan se suunniteltuun ympäristöön ennen varsinaisten konkreettisten mallien tekoa. Reaaliaikaisen varjostuksen avulla voidaan mallin toimivuutta tarkkailla eri valaistusolosuhteissa. Tämä voi olla hyödyllistä esim. auton turvavalaistusta suunniteltaessa tai rakennuksen valoja aseteltaessa.

6. Yhteenveto

Menetelmiä varjojen luomiseen on kolmiulotteisessa tietokonegrafiikassa kehitelty jo kolmisenkymmentä vuotta. Tutkielmassa käsitelty varjosärmiöalgoritmi on yksi ensimmäisiä varjostusmenetelmiä. Vaikka tämä on vanha algoritmi, sen luomat varjot ovat fysikaalisesti oikean muotoisia ja valonlähteet voidaan sijoittaa varjostettavaan maailmaan vapaasti. Vaikka varjosärmiöalgoritmi ei sovellukaan täysin mielivaltaisen muotoisten kappaleiden varjostukseen, reaaliaikaisessa grafiikassa käytettyjä kolmioihin perustuvia malleja voidaan käyttää varjosärmiömenetelmän kanssa.

Reaaliaikaiset varjot ovat viime vuosina tulleet ammattilaisten käyttämistä simulaattoreista ja mallinnusohjelmista myös kotisovelluksiin, sillä varjot luovat tunnelmaa tietokonepeleihin ja helpottavat kolmiulotteisen maailman hahmottamista suunnitteluohjelmissa. Erilaiset projektiovarjo- ja varjokartta-algoritmit ovat olleet vahvoilla eri algoritmien välisessä kilvassa suhteellisen nopeutensa vuoksi, mutta konetehojen kasvaessa tarkkoja varjoja luovan varjosärmiöalgoritmin kiinnostavuus ohjelmistonkehittäjien silmissä varmasti kasvaa. Myös uudet algoritmin muunnokset, jotka mahdollistavat pehmeiden varjojen luomisen reaaliaikaisesti sekä bittikarttavalonlähteiden käyttämisen, kiinnostavat yhä realistisempaan ulkoasuun pyrkiviä ohjelmistonkehittäjiä. Varjokartta-algoritmi ei kuitenkaan katoa käytännön sovelluksista, sillä sekin pystyy muodostamaan pehmeitä varjoja ja on varjostettavan maailman luomiseksi joustavampi kuin varjosärmiöalgoritmi.

Tulevaisuudessa varjosärmiöalgoritmia käytetään yhä enemmän, sillä suurin osa algoritmin käytännön toteutuksiin liittyvistä ongelmista, kuten algoritmin sovittaminen laitteistolle sopivaksi ja varjossa olevan kameran aiheuttamat virheet varjossa, on viime vuosina ratkaistu. Tästä huolimatta algoritmissa on vielä paljon kehitettävää. Haasteina ovat laitteiston tarjoamien mahdollisuuksien entistä parempi hyödyntäminen sekä algoritmin yleinen optimointi. Vaikka varjosärmiöalgoritmin avulla pystytäänkin piirtämään fysikaalisesti täysin oikeanlaisia kovia varjoja, ei pehmeiden varjojen kohdalla saavuteta yhtä hyvää lopputulosta. Näin ollen kaikissa tilanteissa virheettä toimivan pehmeitä varjoja piirtävän varjosärmiöalgoritmin kehittäminen jatkuu. Lisäksi uudenlaiset mallinnustavat ja esim. väliaineiden vaikutus varjoihin (vesi, savu) voivat jatkossa luoda uusia mielenkiintoisia tutkimusmahdollisuuksia myös varjosärmiön ympärille.

Lähteet

[1] F. Crow. Shadow algorithms for computer graphics. ACM Computer Graphics, 11(2):242–247, 1977.

[2] P. Bergeron. A general version of crow's shadow volumes. IEEE Computer Graphics & Applications, 6(9):17–28, 1986.

[3] L. S. Brotman, N. Badler. Generating soft shadows with a depth buffer algorithm. IEEE CG&A, 4(10):5–24, Lokakuu 1984.

[4] N. Chin, S. Feiner. Near Real-Time Shadow Generation Using BSP Trees. ACM Computer Graphics, 23(3): 99–106, Heinäkuu 1989.

[5] T. Heidmann. Real shadows, real time. Iris Universe, 18:28–31, 1991. Silicon Graphics, Inc.

[6] T. Möller, E. Haines. Real-Time Rendering. A K Peters, 1999.

[7] A. Woo, P. Poulin, A. Fournier. A survey of shadow algorithms. IEEE Computer Graphics and Applications, 10(6):13–32, Marraskuu 1990.

[8] U. Assarsson, T. Akenine-Möller. A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware. ACM Transactions of Graphics (Proceedings of ACM SIGGRAPH), 22(3):511–520, Heinäkuu 2003.

[9] C. Everitt, M. J. Kilgard. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. NVIDIA Corporation, 2002.