Reaaliaikaiset varjoalgoritmit

Atso Kauppinen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Maaliskuu 2008

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi / Ohjelmistotuotanto
Atso Kauppinen: Reaaliaikaiset varjoalgoritmit
Pro gradu -tutkielma, 93 sivua
Maaliskuu 2008

Varjolaskennasta on viime vuosina tullut merkittävä osa reaaliaikaista grafiikkaa. Toisin kuin oikeaoppiset säteenjäljitystekniikat, reaaliaika-algoritmit eivät pyri fysikaalisesti täydellisiin varjoihin, vaan niiden tavoitteena on tuottaa vakuuttavan näköisiä kuvia mahdollisimman yksinkertaisin menetelmin. Algoritmit ovat siis kompromisseja visuaalisen tarkkuuden, sekä prosessori- ja muistiystävällisyyden välillä.

Tämän tutkimuksen puitteissa käydään läpi kaikki keskeiset ja yleisessä käytössä olevat varjoalgoritmit: tasoprojisointi, varjotilavuudet sekä varjokartat. Menetelmistä annetaan perusteellinen kuvaus, käyden läpi toteutusyksityiskohtien lisäksi niiden heikkoudet ja vahvuudet, sekä tavat, joiden avulla niiden toimintaa on mahdollista tehostaa. Lopuksi luodaan vielä silmäys pehmeiden varjojen muodostamiseen sekä kartoitetaan sitä, mihin suuntaan reaaliaikaiset varjolaskentamenetelmät ovat mahdollisesti lähiaikoina kehittymässä.

Avainsanat ja -sanonnat: 3D, grafiikka, reaaliaika, varjo, tasoprojisointi, varjotilavuus, varjokartta, pehmeät varjot

Sisällys

1. Johdanto

Varjojen merkitys reaaliaikaisessa 3D-grafiikassa on kasvanut vuosi vuodelta, ja ne ovat perinteisten säteenjäljitys ym. tekniikoiden ohella nousseetkin merkittäväksi tutkimuskohteeksi tietokonegrafiikan saralla. Koska menetelmien tavoitteena on pystyä päivittämään näkymää hyvin nopealla tahdilla, oikeaoppiset varjoalgoritmit eivät sellaisenaan kelpaa. Niiden rinnalle onkin täytynyt kehittää menetelmiä, jotka yksinkertaistavat varjolaskennan perusajatuksen äärimmilleen, ja kykenevät näin, fysikaalisesti virheellisestikin, tuottamaan vakuuttavan näköisiä lopputuloksia. Keskeinen ongelma ei olekaan matematiikka sinänsä vaan se, kuinka laskennan määrä pystytään rajoittamaan mahdollisimman vähäiseksi ja keskittymään vain siihen, mikä on kullakin hetkellä näkymän kannalta oleellista.

Varjomallinnukseen pätevät samat lainalaisuudet kuin muuhunkin 3D-ohjelmointiin. Tärkeää on mm. rajata käsittelyn ulkopuolelle kaikki sellainen, mikä ei vaikuta kulloinkin piirrettävän näkymän sisältöön. Tässä tutkielmassa ei puututa 3D-ohjelmoinnin perusasioihin (jotka toki sinänsä kuuluvat kiinteästi myös moniin myöhemmin esiteltävistä menetelmistä), vaikka aluksi tehdäänkin pikainen yhteenveto yleisimpien rajapintojen tarjoamista palveluista sekä varjojen perusfysiikasta.

Woo et al. [1990] käyvät kattavasti läpi varjoalgoritmien laajaa kirjoa, eikä kaikkiin heidän esittelemiinsä menetelmiin tässä tutkielmassa kajota. Kaikkien mahdollisten ideoiden pohjalta on tavallaan ihmeellistä, että yleisesti käytössä olevat reaaliaikaiset varjoalgoritmit voidaan huoletta jakaa vain kolmeen haaraan, joista niistäkin vain kaksi täyttää todelliset yleiskäytettävyyden vaatimukset. Tämän tutkimuksen puitteissa pyritään antamaan mahdollisimman kattava kuva näiden kaikkien menetelmien sisällöstä, ja tutkimaan niiden kehittymistä 3D-grafiikan historian alkuhämäristä aivan viime vuosiin saakka. Lisäksi lopussa käydään läpi reaaliaikaisten varjoalgoritmien seuraava, pehmeiden varjojen sukupolvi, joka kykenee luomaan huomattavasti entistä realistisempia ja vaikuttavampia graafisia esityksiä, käyttäen hyväksi uusimpien näytönohjainten ominaisuuksia.

2. Yleistä varjolaskennasta

Varjot näyttelevät keskeistä roolia, kun pyritään luomaan todellisen kaltaisia kuvaesityksiä. Ne luovat kuvaan realismia, ja auttavat katsojaa myös hahmottamaan kuvassa olevien esineiden asentoa, muotoa sekä ja sijaintia ollen näin kuvan sisällön ymmärtämisen kannalta tärkeässä osassa [Mamassian et al., 1998]. Varjot auttavat myös määrittämään toisistaan irti olevien esineiden

etäisyyttä toisistaan tai maasta. Lisäksi varjot voivat kuvastaa joko maanpinnan tai toisten esineiden, joihin varjo lankeaa, muotoja. Varjot lisäävät kuvaan myös informaatiota: toisinaan esineen todellinen muoto ja olemus voivat selvitä ainoastaan varjon avulla.

2.1. Perusasioita varjoista ja valoista

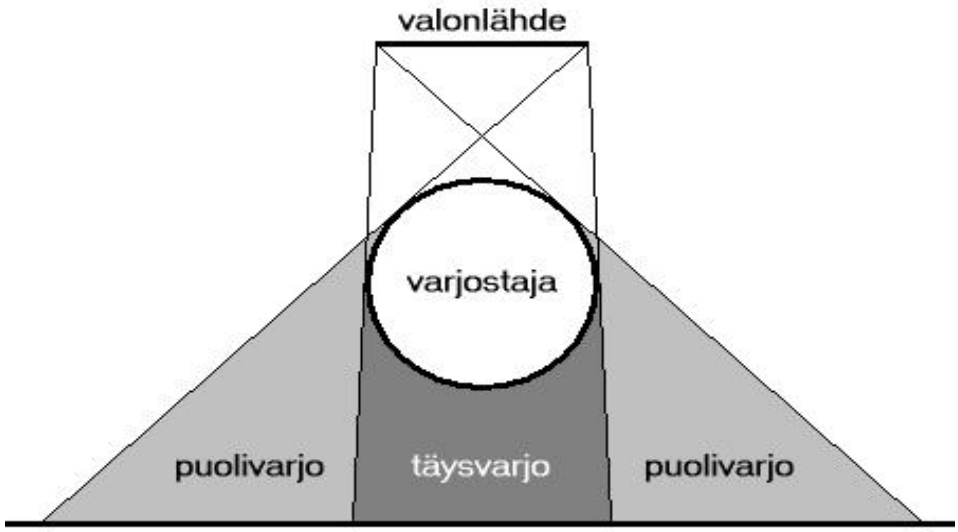
Yksinkertaisimmalla tasolla varjo muodostuu kolmesta eri osatekijästä: valosta, varjostavasta esineestä ja esineestä johon varjo lankeaa, eli varjostettavasta [Hasenfrantz et al., 2003]. Varjon ominaisuuksiin vaikuttavat näiden osatekijöiden koko, etäisyys toisistaan sekä niiden asento toisiinsa nähden. On huomioitava, että koveria muotoja sisältävä esine voi langettaa varjoa itseensä ollen näin samaan aikaan sekä varjostava että varjostettava esine. Tämä, sinänsä itsestäänselvyydeltä kuulostava asia on hyvin oleellinen huomio, kun varjoja luodaan matemaattisesti. Lisäksi niin valoja kuin kummankin tyyppisiä esineitä voi vaikuttaa toisiinsa samaan aikaan useampia. Tämä on niin ikään tärkeä erikoistapaus varjojen laskennassa.

Todellisessa maailmassa tilanne varjojen suhteen ei ole näin yksinkertainen. Edellä esiteltyjen osatekijöiden lisäksi varjoihin, kuten tietysti muuhunkin valaistukseen, vaikuttavat esineiden pinnoilta lähtevät heijastukset. Säteenjäljitystekniikoilla (ray tracing) on jo pitkään pyritty mallintamaan näiden heijastuksen aiheuttamia ilmiöitä, mutta ainakin toistaiseksi nämä menetelmät eivät kykene likimainkaan sellaiseen ajalliseen tehokkuuteen, että niitä pystyttäisiin käyttämään saumatonta vuorovaikutteisuutta vaativissa ohjelmistoissa. Niinpä reaaliaikaiset varjoalgoritmit keskittyvät käyttämään varjojen mallintamisessa vain valoa, varjostajaa ja varjostettavaa.

Varjostavia valoja on kolmea eri perustyyppiä: pistevaloja, suunnattuja valoja ja kohdevaloja. Pistevaloilla on äärellinen sijainti, mutta ei erikseen määriteltyä suuntaa, eli ne hohtavat valoa kaikkiin suuntiin. Suunnatuilla valoilla puolestaan on suunta, mutta ei äärellistä sijaintia. Ne valaisevat kaikkia esineitä samansuuntaisesti. Aurinko sijaitsee graafisen laskennan näkökulmasta ajateltuna äärettömän kaukana, ja se on hyvä esimerkki suunnatusta valosta. Kohdevalolla on sekä sijainti että suunta ja usein myös aukeamiskulma, joka määrittää valon vaikutusalueen.

Riippuen siitä, varjostaako esine jossain kohdassa itseään vai varjostettavaa, on kyseinen kohta varjosta joko kiinnitetty tai langennut [Hasenfrantz et al., 2003]. Kiinnitetty varjo on ikään kuin kiinni siinä esineessä joka sen on aiheuttanutkin. Valosta poispäin osoittava puoli esineestä on todellisessa elämässä aina kiinnitetyn varjon varjostama. Langennut varjo taas on saanut alkunsa jostain muusta esineestä. Esimerkiksi varjo joka viilentää päivänvarjon alla istuvaa ihmistä, on langennut varjo.

Viimeisenä perusasiana varjon alueet jaetaan vielä kahteen eri osaan: täysvarjoon ja puolivarjoon (kuva 1). Täysvarjo (tai kokovarjo) on kokonaan varjostavan esineen peittämä, eli yhdestäkään sen pisteestä ei ole suoraa näköyhteyttä valonlähteeseen. Puolivarjosta on valonlähteeseen vajavainen näkyvyys. Tällä alueella valon vaikutus on pienempi, ja valaistus vähäisempi kuin sellaisella alueella, josta valonlähde on kokonaan näkyvissä.



Kuva 1. Varjon osat

2.2. 3D-rajapintojen toiminnasta

Varjolaskennan kannalta on oleellista omata tuntemus sekä 3D-grafiikan matemaattisista muunnoksista että piirtopinnoista, joilla lopputulos esitetään. Seuraavaksi käydään läpi muutamat keskeiset työkalut, joita 3D-rajapinnat tarjoavat varjomallinnuksen ja muun piirtämisen avuksi. Tämän tutkielman esimerkeissä käytetään ensisijaisesti OpenGL:ää, mutta niiden sovittaminen esimerkiksi Direct3D-rajapinnalle pitäisi olla melko virtaviivaista.

2.2.1. Muunnoskanava

Varjoalgoritmiikan kannalta on tärkeää ymmärtää muunnoskanavan (transform pipeline) toimintaperiaate, sillä lähes kaikki sen kohdat tulevat jossain vaiheessa käsitellyiksi. Tässä luvussa esitellään muunnoskanava sellaisena kuin se OpenGL–rajapinnassa suoritetaan. Jotkut kohdat voivat muissa rajapinnoissa hieman erota, mutta periaate on sama.

Muunnoskanava on muunnosmatriiseista sekä muista, toisinaan vapaavalintaisista ja 3D-rajapinnan määrittelystä riippuvaisista operaatioista koostuva ketju, jonka läpi kolmiulotteisessa maailmassa määritelty piste kulkee ennen päätymistään kaksiulotteiseksi koordinaatiksi tietokoneen näyttömuistiin (kuva 2). 3D-näkymän kärkipisteet kuuluvat yleensä malliin,

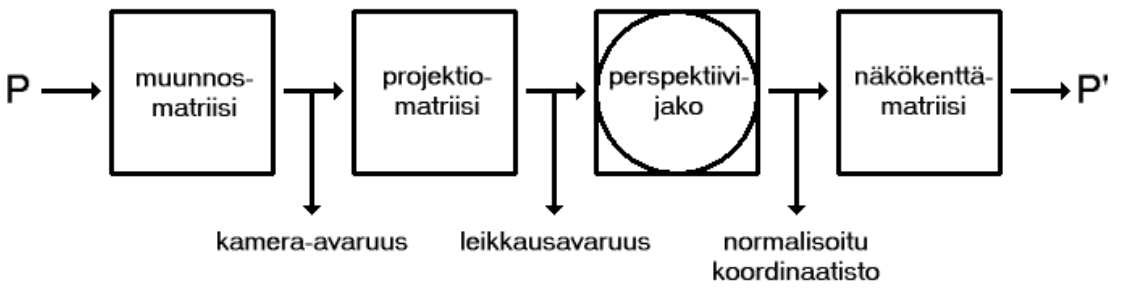
jossa ne yhdistetään monikulmioilla toisiinsa. Piste määritellään mallin paikallisessa koordinaatistossa, mistä se saadaan mallin omalla muunnosmatriisilla siirrettyä globaaliin avaruuteen, joka määrittää sille absoluuttisen sijainnin suhteessa kaikkiin muihin maailman esineisiin. Tämän jälkeen piste muunnetaan kamera-avaruuteen, missä koordinaatit kuvaavat sijainnin suhteessa kameraan. Käytännössä tämä tapahtuu kertomalla piste kameran (jonka suunta ja sijainti on määritelty kuten esineilläkin) käänteismatriisilla.

Kameran näkökenttä muodostaa pyramidin, jonka huippu on leikattu pois. Kamera sijaitsee pyramidin huipulla, ja näkökentän etu- ja takareuna rajaavat katkaistun pyramidin ylä- ja alapinnan. Näkökentän avautumiskulmat puolestaan ratkaisevat sen, kuinka leveä pyramidista tulee.

Piste kerrotaan kameran ominaisuuksista koostetulla projektiomatriisilla, joka muuntaa näköpyramidin homogeeniseksi leikkausavaruudeksi, mistä piste voidaan w-komponentin suhteen normalisoimalla siirtää nk. normalisoituun koordinaatistoon. Siellä kaikki näkökentän sisällä olevat pisteet ovat kuutiossa, jonka akselien ääripisteet ovat -1 ja 1. Perspektiivinen projektiomatriisi synnyttää syvyysvaikutelman projisoimalla kameran lähellä olevat pisteet suuremmalle alueelle kuin takana olevat. Mikäli näkökenttä on kuution muotoinen, muunnosta kutsutaan ortografiseksi projektioksi. Tällöin perspektiivistä vääristymää ei synny.

Pisteen kaksiulotteinen koordinaatti saadaan kertomalla normalisoitu koordinaattipiste näkökenttämuunnoksella. Se muuntaa x ja y koordinaattien arvot suhteellisiksi paikoiksi näytön pikseliresoluutiossa. Pikselin normalisoitu etäisyys kamerasta muunnetaan välille [0, 1]. Mitä suurempi syvyysarvo on, sitä kauempana kamerasta piste sijaitsee.

Homogeenisellä koordinaatistolla on eräs mielenkiintoinen ominaispiirre: asettamalla w-komponentti nollaksi, piste projisoituu origosta katsottuna äärettömyyteen. Suunnattu valo voidaan esimerkiksi esittää tällaisena koordinaattina. Kuten myöhemmin tullaan huomaamaan, w-komponentin nollaamisesta on paljon apua myös varjojen luomisessa.



Kuva 2. OpenGL:n muunnoskanava. P on piste globaalissa koordinaatistossa ja P’ on sama piste kuvaruudulla.

2.2.2. Piirtopinnat

Kun matemaattisesti esitettävä kolmiulotteinen maailma on edellisen luvun mukaisesti siirretty kaksiulotteiselle tasolle, joudutaan tekemisiin piirtopintojen kanssa. 3D-rajapinnat tarjoavat käyttäjälle useita erilaisia piirtopintoja eli puskureita, joita voidaan käyttää monin eri tavoin myös varjolaskennan apuna. Seuraavassa on lyhyt esittely yleisimmistä puskureista ja niiden käyttötarkoituksista. Perusteellisempia esittelyjä niiden tarjoamista mahdollisuuksista löytyy mm. eri 3D-rajapintojen kotisivuilta.

Puskureista keskeisin on väripuskuri (color buffer), joka sisältää ruudulla näkyvän kuvan. Itse asiassa väripuskureitakin on yleensä useampia, joista yksi on kerrallaan näkyvissä, ja muille piirretään samaan aikaan. Kun piirtäminen on saatu suoritettua, kuvat voidaan vaihtaa nopeasti päinvastaisiksi välttäen piirtämisestä aiheutuva ruudun välkkyminen.

Kolmiulotteisen grafiikan kannalta syvyyspuskurilla (depth buffer) on erittäin keskeinen merkitys. Se sisältää kutakin piirtopinnan pikseliä kohti syvyysarvon, joka saadaan selville aiemmin esiteltyjen muunnosten yhteydessä. Syvyyspuskurin avulla voidaan tehdä pikselitason syvyystarkastelua ja varmistaa, että kaukana kamerasta olevat pisteet eivät vahingossa piirry lähempänä olevien päälle.

Sapluunapuskuri (stencil buffer) on bittisyvyydeltään edellisiä pienempi, harvoin yli kahdeksaa bittiä, ja se käsittelee ainoastaan kokonaislukuja. Sitä käytetään useimmiten rajaamaan alueita pois ruudusta piirto-operaatioiden ajaksi, mutta sille voidaan myös antaa hyvin monipuolisia komentoja riippuen syvyystestin tuloksesta. Sapluunapuskuria hyödynnetään perinteisesti paljon juuri varjolaskennassa.

Keräyspuskuri (accumulation buffer) on sen sijaan bittisyvyydeltään yleensä hyvin tarkka. Se on tarkoitettu useamman kuvan keskinäiseen sulauttamiseen (blending). Keräyspuskuriin ei ole mahdollista piirtää suoraan, vaan kuva piirretään ensin väripuskuriin, ja siirretään sen jälkeen keräyspuskuriin jonkin sulautusoperaation saattelemana. Kun kaikki kuvat on sulautettu, keräyspuskuri voidaan kopioida väripuskuriin ja näyttää ruudulla. Sulauttamisen avulla voidaan mm. luoda ristikuvia ja liikesumennuksia (motion blur), sekä (tämä on varjostuksen kannalta oleellista) hallitusti muuttaa kuvan valoisuutta halutuissa paikoissa.

Kaikkien yllä mainittujen puskurien resoluutio on sama, ja ne luodaan ohjelmaa käynnistettäessä. Lisäksi on mahdollista luoda erillisiä bittikarttoja,

joiden resoluutio ja elinaika ovat ohjelmoijan päätettävissä. Näitä kutsutaan tekstikartoiksi (texture map). Tekstikarttoja voi olla periaatteessa kuinka paljon tahansa, ja ne voivat sisältää mitä tahansa tietoa. Useimmiten niiden pikselit (tekselit) sisältävät väriarvoja, mutta myöhemmin tullaan huomaamaan, että tekstikarttoja voi käyttää myös esimerkiksi syvyyspuskureina. Mikä tahansa puskuri voidaan periaatteessa kopioida toiseen, eivätkä tekstuurit tee tässä poikkeusta. Käytännössä kaikki nykyiset näytönohjaimet tarjoavat myös mahdollisuuden piirtää väripuskurin sijasta suoraan tekstikarttaan. Näin ne poistavat tarpeen hitaahkolle kopiointioperaatiolle.

Paremman kuvanlaadun saavuttamiseksi kuvatekstuureista on tapana luoda hierarkkinen pyramidimalli, nk. mipmap [Williams, 1983]. Se tehdään luomalla kuvasta uusia, puolitettuja tekstikarttoja, jotka muodostavat hierarkian: ylemmän, epätarkemman kartan yksi pikseli on suodatettu alemman kartan neljästä pikselistä jne. Koska tämä on nykyisten 3D-rajapintojen vakiotoiminto, voidaan tekniikkaa soveltaa tehokkaasti monin eri tavoin.

2.2.3. Varjostimet

Nykyiset näytönohjaimet mahdollistavat pienimuotoisten ohjelmien, eli varjostimien (shader) suorittamisen niiden omilla prosessoreilla. Varjostimien avulla on mahdollista suorittaa hyvin tehokkaita kärkipiste- ja pikselikohtaisia operaatioita samaan aikaan kun tietokoneen keskusprosessori tekee jotain muuta. Tehokkuuden lisäksi varjostimet tuovat mukanaan myös rutkasti ilmaisuvoimaa. Varjostimet jaetaan kahteen eri luokkaan: verteksi- ja pikselivarjostimiin (vertex shader ja pixel tai fragment shader). Ne ajetaan eri kohdassa muunnosketjua, mutta ne pystyvät kuitenkin myös jakamaan tietoa keskenään.

Verteksivarjostimet voivat korvata muunnoskanavan. Yksinkertaisimmillaan ne saavat parametrina kärkipisteen, ja palauttavat sen halutulla tavalla muunneltuna takaisin näytönohjaimelle. Verteksivarjostimelle on mahdollista antaa parametreja, joita voidaan käyttää pisteen koordinaatin, väriarvojen ym. muuntamiseen. Varjostin voi saada syötteenään vain yhden pisteen kerrallaan, eikä se myöskään voi luoda uusia kärkipisteitä. Tämä rajoitus mahdollistaa useiden pisteiden käsittelyn prosessorilla samaan aikaan.

Pikselivarjostimet puolestaan korvaavat pikselien piirto-operaation. Ne määrittävät sen, millä värillä mikäkin pikseli piirretään, vai piirretäänkö sitä ollenkaan. Operaatiossa käytetään hyväksi aiemmin esiteltyjä puskureita sekä pikselivarjostimelle mahdollisesti välitettyjä ylimääräisiä parametreja. Myöskään pikselivarjostin ei pääse käsiksi mihinkään muuhun väripuskurin

pikseliin kuin siihen, jota se on kulloinkin käsittelemässä. Niillä on kuitenkin lukuoikeus tekstikarttojen pikselitietoon.

Edellisten lisäksi uusimmat näytönohjaimet ovat alkaneet tukea myös nk. geometriavarjostimia (geometry shader), joiden avulla on mahdollista luoda uusia kärkipisteitä sekä monikulmioita sen jälkeen kun pisteet ovat jo kulkeneet verteksivarjostimen läpi. Tämä ominaisuus on tullut mukaan 3D rajapintoihin vasta aivan hiljattain (Direct3D 10 tukee sitä, mutta OpenGL:n perusvarjostimissa se ei tätä kirjoitettaessa ole vielä mukana), ja on siksi vielä vähäisessä käytössä. Uuden geometrian generointi varjostimissa on kuitenkin hyödyllinen ja paljon toivottu ominaisuus esimerkiksi varjotilavuuksia luotaessa.

2.3. Reaaliaikaisuuden rajoituksia

Kuten aiemmin jo todettiin, varjojen reaaliaikainen laskenta aiheuttaa suuria rajoituksia algoritmien monimutkaisuudelle. Niinpä kaikkein kehittyneimmät säteenjäljitystekniikat eivät sovellu käytettäväksi tosiaikaisuuden yhteydessä. Vaikuttavimman lopputuloksen aikaansaamiseksi käytetään kuitenkin monesti tapaa, jossa staattiset varjot lasketaan etukäteen, ja piirretään sitten oikeisiin paikkoihin ilman ajonaikaista laskentaa. Tämän tutkimuksen painopiste on kuitenkin nimenomaan dynaamisesti laskettavissa varjoissa.

Pintaheijastusten puuttuminen reaaliaikaisesti lasketusta ympäristöstä on ongelma, mikä korjaamattomana johtaa siihen, että kaikki pisteet mihin valo ei suoraan paista, ovat täysin pimeitä. Todellisessa elämässä valo heijastuu lähes kaikista pinnoista ja valaisee myös näitä paikkoja. Pintaheijastusten kaltaisen vaikutelman aikaansaamiseksi käytetään tavallisesti taustavaloa, joka on tasaisesti läsnä kaikkialla. Tämä ei tietenkään millään tavalla vastaa oikeita heijastuksia, mutta tarjoaa niistä parhaimmillaan onnistuneen illuusion.

Sen lisäksi että tosiaikaisuuden saavuttamiseksi on luovuttava valojen pintaheijastuksista, ongelmaksi muodostuu myös äärellisen kokoisen valonlähteen mallintaminen. Tämän vuoksi perinteiset varjoalgoritmit olettavatkin valonlähteen olevan äärettömän pieni piste. Tällaista tilannetta ei kuitenkaan esiinny todellisuudessa, eli lopputuloksena syntyy luonnottoman teräviä ja tarkkoja varjoja. Tämän ongelman korjaaminen on viime vuosina ollut aktiivisen pohdinnan kohteena, ja sille omistetaan tässäkin tutkielmassa yksi luku.

2.4. Usean valonlähteen piirtäminen

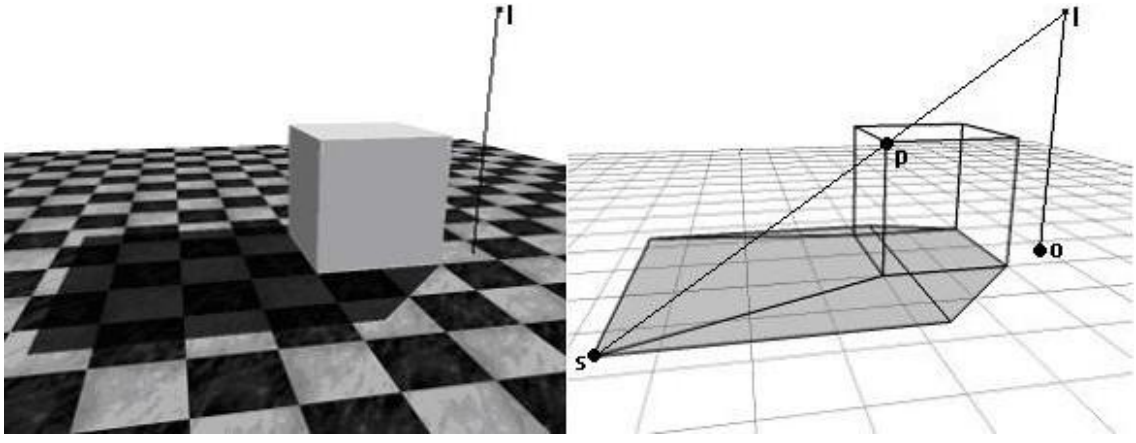
Monta valoa sisältävä maailma voidaan varjostaa kahdella tavalla. Ensimmäisessä vaihtoehdossa näkymä piirretään pitämällä kaikki valot päällä, ja jälkikäteen himmentämällä varjoon jääviä kohtia halutun verran. Toinen

vaihtoehto on aloittaa mustasta ruudusta, ja piirtää näkymä monta kertaa siten, että vain kukin yksittäinen valo on kerrallaan aktiivisena. Ensimmäinen keino on nopeampi, koska näkymä tarvitsee piirtää vain kerran, mutta tällöin esimerkiksi kohdevalojen synnyttämät kirkkaasti valaistut kohdat näkyvät varjostavien esineiden läpi, mikä ei tietenkään ole oikein.

Oikeaoppisempaa onkin tehdä valaistus jälkimmäisellä tavalla. Ensin piirretään koko näkymä pitämällä ainoastaan taustavalaistus päällä. Tämän jälkeen piirretään näkymä kullekin valolle erikseen siten, että varjostetut kohdat jäävät mustiksi. Piirretty kohta sulautetaan sitten keräyspuskurissa edelliseen kuvaan ja kun kaikki valot on saatu käsiteltyä, lopputulos kopioidaan ruudulle. Tämä takaa fyysisesti oikeaoppisen lopputuloksen ollen kuitenkin ensimmäistä menetelmää raskaampi, koska näkymä on piirrettävä monta kertaa. Kerroksittaisen piirtämisen tehostamiseen onkin laitevalmistajien kohdalta panostettu paljon [Morein, 2000]. Kannattaa myös huomata ero valaistus- ja varjostusarvojen välillä. Pikseli, jonka valaistusarvo on 1, piirretään täydellä valaistuksella, kun taas valaistusarvo 0 tarkoittaa pisteen olevan kokonaan varjossa. Varjostusarvojen kohdalla arvot ovat toisinpäin.

3. Varjojen tasoprojisointi

Varjojen tasoprojisointi on yleisesti käytetyistä varjoalgoritmeista selkeästi yksinkertaisin. Periaate on, että kutakin 3D-mallin pistettä työnnetään valon (äärettömän pienestä) pisteestä poispäin, kunnes se saavuttaa varjostettavan tason (kuva 3). Tasoprojisointi vaatiikin, että varjostettavan esineen on oltava tasainen pinta. Tästä ja myöhemmin käsiteltävistä muista puutteista johtuen, menetelmää voidaan käyttää vain tietyissä erikoistapauksissa, eikä se pääse missään suhteessa samalle yleiskäytettävyyden tasolle kuin kehittyneemmät algoritmit. Sen periaate on kuitenkin hyvä tuntea, koska monet sen kohdalla esitetyt perusajatukset ja ongelmat toistuvat myöhemmin myös muiden menetelmien yhteydessä.



Kuva 3. Tasoprojisoinnin perusidea. Piste p projisoidaan tasolle valonlähteen suhteen [Ambrož, 2006].

3.1. Tasoprojisointimatriisi

James Blinn [1988] kehitti matriisin (kuva 4), jonka avulla mikä tahansa kolmiulotteinen piste voidaan projisoida vapaavalintaista tasoa vasten. Kaavan esittäminen matriisina on tärkeää, sillä sen avulla operaatio voidaan helposti yhdistää muunnoskanavaan, ja näin hyödyntää näytönohjainten laskuominaisuuksia varjon muodostamisessa.

Blinnin matriisi tekee pisteelle projektion, eli muuntaa sen kolmiulotteisesta avaruudesta kaksiulotteiseen. Tämä aiheuttaa sen, että ilman erikoiskäsittelyä myös varjostettavasta tasosta katsottuna valonlähteen takana olevat esineet projisoituvat tasolle. Sama virhe syntyy myös silloin, kun varjostava esine on valosta katsottuna varjostettavan takana. Virheelliset projisoinnit voidaan estää käsittelemällä ne koodissa erikoistapauksina, tai käyttämällä ylimääräisiä leikkaustasoja. Heckbert ja Herf [1997] tarjoavat elegantin ratkaisun: pisteet voidaan muuntaa valon sijainnin ja varjostettavan nelikulmion (joka sijaitsee kohdetasolla) rajoittamaan näkökenttään ja siitä edelleen leikkausavaruuteen, jolloin kaikki tämän alueen ulkopuolelle jäävät pisteet on helppo rajata pois. Tähän aiheeseen palataan luvussa 6.1.

$$S = \begin{pmatrix} T\,L - L.x\,T.x & -L.y\,T.x & -L.z\,T.x & -L.w\,T.x \\ -L.x\,T.y & T\,L - L.y\,T.y & -L.z * T.y & -L.w\,T.y \\ -L.x\,T.z & -L.y\,T.z & T\,L - L.z\,T.z & -L.w\,T.z \\ -L.x\,T.w & -L.y\,T.w & -L.z\,T.z & T\,L - L.w\,T.w \end{pmatrix} \begin{pmatrix} P.x \\ P.y \\ P.z \\ P.w \end{pmatrix}$$

Kuva 4. Matriisi, joka projisoi pisteen P tasolle T (T.w on etäisyys origosta). L on valonlähteen sijainti. Mikäli sen w-komponentti on 0, kyseessä on suunnattu valo.

3.2. Tasoprojisoinnin käyttö

Menetelmän käyttö on periaatteessa hyvin yksinkertaista. Aluksi piirretään varjostettava esine, tämän jälkeen projisoidut varjot, ja lopuksi muut esineet. Varjostettavia malleja piirrettäessä pidetään valonlähteet päällä. Sen jälkeen ne voidaan ottaa hetkeksi pois. Varjot piirretään yleensä himmentämällä, eli vähentämällä niiden täyttämien pikselien valoisuusarvoa. Näin saadaan aikaan tummat, mutta ei täysin mustat varjot. Kun varjot on piirretty, palautetaan valot taas käyttöön, ja piirretään loput esineet. Varjoja piirrettäessä on huomioitava, että varjostava malli on ensin muunnettava omasta paikallisesta

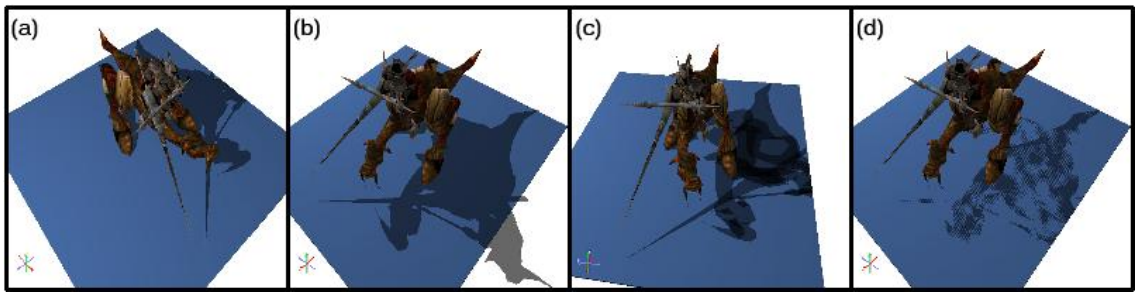
avaruudestaan globaaliin avaruuteen. Tällöin sen pisteet ovat samassa avaruudessa kuin valokin.

Varjojen projisointia ei välttämättä tarvitse tehdä jokaisella kerralla. Mikäli valo, varjostava ja varjostettava esine eivät muutu riittää, että varjo piirretään vain kerran tekstikarttaan, ja kiinnitetään tekstuuri tämän jälkeen varjostettavaan pintaan. Tätä ajatusta voidaan hyödyntää mm. pehmeiden varjojen esilaskennassa [Heckbert ja Herf, 1997], kuten myöhemmin tullaan huomaamaan.

Kannattaa huomata, että useimmiten piirtorutiinien aikana aktivoituna oleva piilopintojen poisto (backface culling) ei saa olla päällä kun varjoa projisoidaan. Tämä johtuu siitä, että ne monikulmiot jotka eivät ole näkyvissä kameran näkökulmasta, voivat hyvinkin näkyä valon suunnasta katsottaessa. Jos nämä monikulmiot jäävät piirtämättä, tuloksena on epätäydellinen varjo.

Tällaisenaan algoritmi on ongelmallinen, koska se projisoi pisteet äärettömälle tasolle. Mikäli varjon halutaan lankeavan vain monikulmioille, käy jossain vaiheessa todennäköisesti niin, että osa varjosta ylittää halutut rajat ja jää ilmaan leijumaan (kuva 5b). Lisäksi, koska algoritmi projisoi tasolle kaikki varjostavan mallin monikulmiot, tulee vähänkään monimutkaisempien esineiden kanssa eteen tilanne, jossa osa monikulmioista piirretään ainakin osittain toistensa päälle. Koska tummennus tehdään jokaiselle monikulmioille erikseen, ne kohdat joissa niitä on päällekkäin enemmän muuttuvat tummemmiksi kuin ne, joissa päällekkäispiirtoja on vähemmän (kuva 5c). Tämä ei ole haluttu toiminto kuin korkeintaan joidenkin läpikuultavien esineiden kohdalla, joten asiassa joudutaan turvautumaan ensimmäistä kertaa sapluunapuskurin apuun.

Mark Kilgard [1999] laajentaa algoritmia korjatakseen mainitut ongelmat. Ennen kuin varjostettavaa esinettä aletaan piirtää, tyhjennetään sapluunapuskurin pisteet arvoon nolla. Tämän jälkeen annetaan rajapinnalle ohje, että aina kun ruudulle piirretään piste, vastaava sapluunapuskurin arvo muutetaan ykköseksi. Tämän jälkeen varjostettava esine piirretään kuten ennenkin. Nyt piirretyn esineen peittämä alue kuvaruudulla on merkitty sapluunaan ykkösinä, muiden pisteiden ollessa nollia. Ennen varjojen piirtämistä muutetaan toimintaa siten, että jos pikseliä vastaava sapluunan arvo on nolla, piste jätetään piirtämättä. Muussa tapauksessa piste piirretään, mutta heti piirtämisen jälkeen sapluunapiste nollataan. Tämä estää varjopisteiden himmentämisen useaan kertaan, ja ylivarjostuminen korjautuu. Koska sapluunassa ykkösiä on vain niissä kohdissa joihin varjostavaa esinettä on piirretty, varjostettavien monikulmioiden reunat ylittäneet varjotkin näkyvät nyt oikein.



Kuva 5. Oikean näköinen varjo (a) ja tasoprojisoinnissa esiintyviä ongelmia: varjostettavan mallin yli vuotava varjo (b), päällekkäiset tummennukset (c) ja syvyyspuskurin epätarkkuus (d) [Ambrož, 2006].

Projisoituja varjoja käytettäessä tulee vastaan myös ongelma, joka muodostuu vaikeaksi ratkaistavaksi myöhemmin varjokarttojenkin kohdalla. Koska varjon pisteet projisoidaan täsmälleen varjostettavan esineen pinnalle, pitäisi pinnan pisteen ja samassa kohtaa olevan varjopisteen syvyysarvojen olla täsmälleen samat, ja varjon näin ollen kokonaan näkyvissä. Käytännössä syvyyspuskurin bittien määrä kuitenkin rajoittaa siihen tallennettujen arvojen tarkkuutta, ja pyöristysepätarkkuuksista johtuen osa varjosta painuu varjostettavan pinnan taakse (kuva 5d). Mikäli piirtäminen hoidetaan algoritmia mukaillen, eli varjot piirretään ennen varjostavia objekteja, syvyystarkastelua ei aina tarvitse edes käyttää [Kilgard, 1999]. Jatkossa tilanne kuitenkin muuttuu monimutkaisemmaksi, joten syvyystarkastelun poistaminen ei riitä. Ratkaisu on, että piirron aikana pikselikohtaisia syvyysarvoja vähennetään tietyn määrän verran, jolloin varjo asettuu syvyystarkastelussa varjostettavan pinnan eteen. Sama asia on periaatteessa tehtävissä myös kolmiulotteisilla muunnoksilla, mutta tällöin varjo irtoaisi esineestä ja alkaisi leijua, mikä olisi läheltä tarkasteltuna huomattava virhe. Kun siirto tehdään pikselikohtaisesti, sillä ei ole vaikutusta esineiden sijaintiin kolmiulotteisessa avaruudessa. Syvyysarvojen lisääminen ja vähentäminen ovat perustoimintoja 3D-rajapinnoissa. Esimerkiksi OpenGL hoitaa asian glPolygonOffset–funktiolla.

3.3. Tasoprojisoinnin rajoituksia

Tasoprojisoinnin rajoituksista on jo aikaisemmin käynyt ilmi, että se vaatii varjostettavaksi esineeksi tasaisen, erikseen määritellyn pinnan. Tällaisia on helppo keksiä todellisestakin maailmasta (esimerkiksi jalkapallokenttä), eli sen suhteen tasoprojisointi puolustaa paikkaansa monissa tilanteissa.

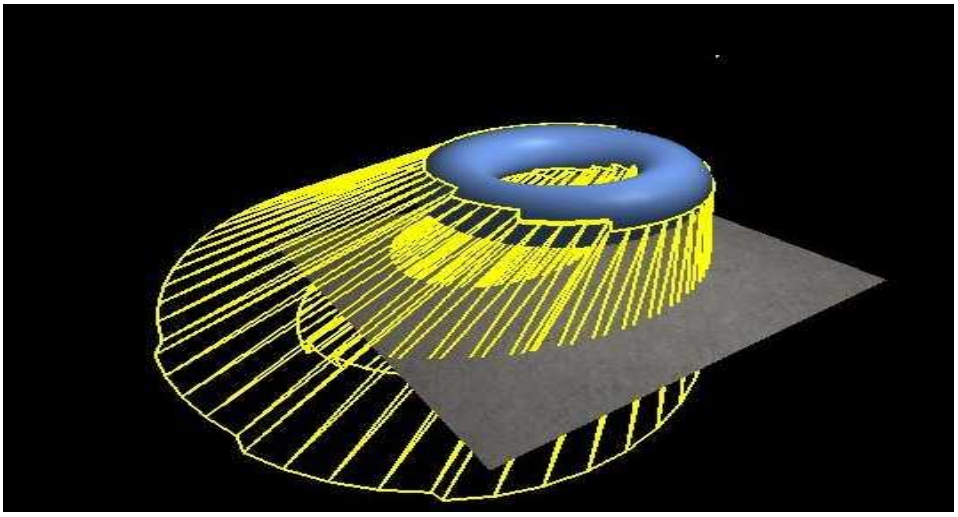
Rajoitukset eivät ikävä kyllä lopu tähän. Merkittävä ja helposti havaittava puute on se, että algoritmi tekee jyrkän jaon varjostavan ja varjostettavan esineen välille, eikä näin ollen pysty langettamaan varjoa itsensä päälle. Virhe

näkyy kaikissa malleissa, jotka sisältävät koveria osia (muun kaltaiset esineet ovat todellisessa maailmassa harvinaisia), ja johtaa siihen, että realistisen lopputuloksen aikaansaamiseksi tarvitaan kehittyneempiä ja yleiskäyttöisempiä algoritmeja.

4. Varjotilavuudet

Varjotilavuusmenetelmä tarjoaa ratkaisun tasoprojisoitujen varjojen ongelmiin. Siinä missä edellinen menetelmä vaati jokaisen vastaanottajan olevan erikseen määritelty ja muodoltaan tasainen, ei varjotilavuuksiin perustuva varjolaskenta vaadi kumpaakaan. Tämän lisäksi varjotilavuudet eivät myöskään tee eroa varjostavan ja varjostettavan esineen välillä, vaan koveratkin mallit varjostuvat oikeaoppisesti. Tämän vuoksi sitä kutsutaan itsevarjostavaksi menetelmäksi. Varjotilavuudet eroavat niin tasoprojisoiduista varjoista kuin varjokartoistakin siinä, että niitä ei muodosteta mitään pintaa tai tasoa vasten. Tämän vuoksi ne eivät myöskään aiheuta ongelmia syvyyspuskurin epätarkkuuden kanssa.

Kun tasoprojisoinnin periaate oli työntää mallin pisteet varjon tasolle erityistä muunnosmatriisia käyttämällä, varjotilavuuksien kohdalla huomio kiinnittyy pisteitä yhdistäviin reunoihin. Yksinkertaisimmillaan varjotilavuuksien laskenta tapahtuu siten, että mallin kaikkien monikulmioiden reunoja venytetään valopisteestä poispäin, jolloin jokaista reunaa kohti muodostuu nelikulmio. Varjonelikulmion yksi reuna on sama kuin se varjostavan monikulmion reuna, mistä venytys aloitettiin. Toinen puolestaan rajoittuu yleensä näkökentän laitoihin. Kun varjomonikulmiot on laskettu, niistä saadaan rakennettua uusia esineitä, joita kutsutaan varjomalleiksi tai -tilavuuksiksi (kuva 6). Mallin jokaisen monikulmion venyttäminen yksitellen johtaa hyvin suureen määrään varjomalleja (yksi kutakin mallin monikulmiota kohti), joten käytännössä pyritään aina etsimään mallin siluettireunat valosta katsottuna, ja venyttämään ainoastaan niitä. Tästä kerrotaan enemmän luvussa 4.2.1.



Kuva 6. Rautalankamalli avoimesta ja äärellisestä varjotilavuudesta. Kukin reunanelikulmio on koostettu kahdesta kolmiosta [Guinot, 2005].

Koska varjotilavuus voi olla hyvinkin monimutkainen, sen kapseloiman tilan laskeminen kolmiulotteisessa avaruudessa olisi erittäin raskas toimenpide. Tämän lisäksi varjoon vain osittain jäävien esineiden leikkeleminen varjon reunoja vasten mutkistaisi menetelmää entisestään. Franklin Crow [1977] kehitti kuitenkin menetelmän, jonka ansiosta varjon sisään jäävät alueet pystytään määrittelemään yksinkertaisesti kaksiulotteisessa avaruudessa. Nykyisien näytönohjaimien ominaisuuksia hyödyntämällä varjotilavuuksista onkin tullut erittäin monipuolinen ja tehokas varjoalgoritmi.

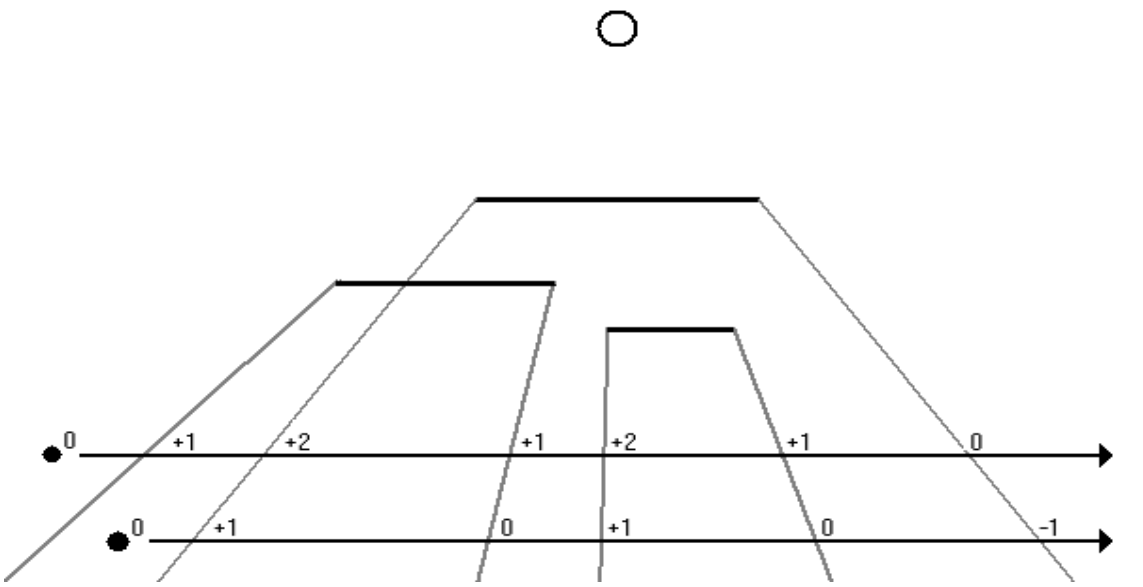
Varjotilavuuksien käyttö jakautuu kahteen erilliseen osaan: varjotilavuuden luomiseen ja sen piirtämiseen. Mallin luominen tapahtuu luonnollisesti ensin, mutta ennen siihen perehtymistä on syytä käydä läpi piirtämiseen liittyvät asiat, sillä mallin koostamisen kannalta on tärkeää ymmärtää, miten menetelmä toimii.

4.1. Varjotilavuuksien toiminta

Kun Crown menetelmän avulla halutaan selvittää onko piste P varjossa, on ensin määritettävä kameran sijaintipisteestä P:hen kulkeva säde. Algoritmi pohjautuu sille yksinkertaiselle havainnolle, että aina kun säde lävistää varjotilavuuden reunan, mennään joko sisälle varjoon, tai tullaan sieltä ulos. Sisäänmeno esineen langettamaan varjoon tapahtuu, kun säde läpäisee sen kameraan päin osoittavan varjopinnan. Kun säde menee kamerasta poispäin osoittavan varjopinnan läpi, tullaan vastaavasti varjosta ulos. Vastakkaisiin suuntiin osoittavat varjopinnat siis ikään kuin kumoavat toisensa.

Crown algoritmi perustuu laskuriin, jonka avulla voidaan näköpisteestä lähetetyn säteen avulla tutkia minkä tahansa pisteen varjostusta. Aluksi

asetetaan laskuri nollaan, ja aletaan seurata sädettä kamerasta poispäin. Kun säde läpäisee varjomallin etureunan, laskuria lisätään yhdellä. Kun se taas kulkee takareunan läpi, sitä vähennetään yhdellä. Kun piste P on saavutettu, tarkastetaan laskurin arvo. Jos se on nolla, on menty yhtä monen etu- ja takapinnan läpi. Täten varjot ovat kumonneet toisensa ja piste P on valossa. Jos arvo taas on jotain muuta, piste on vähintään yhden varjotilavuuden sisällä, eli varjossa (kuva 7).



Kuva 7. Crown menetelmän toiminta yksinkertaisimmillaan. Huomaa, kuinka alemman, varjon sisältä lähtevän säteen laskuri saa vääriä arvoja.

Crown algoritmi on laskurinsa vuoksi luonteeltaan kumulatiivinen ja näin ollen altis virhetilanteille, mikäli laskenta menee yhdessäkään kohtaa pieleen. Menetelmä mm. vaatii, että jokaisella varjomallilla on sekä etu- että takareuna. Monimutkaisempien 3D-näkymien kohdalla tämä tarkoittaa, että jokainen varjomalli on oltava suljettu. Suljetulla esineellä tarkoitetaan mallia, jonka pinnat eivät sisällä aukkoja eivätkä risteäviä reunoja (reunaan kiinnittyy useampi kuin kaksi monikulmiota). Toisin sanoen, jokaisen reunan on liityttävä tasan kahteen monikulmioon. Mikäli varjomalli ei ole suljettu, aiheuttaa avoimesta päästä kulkeva säde sen, ettei mallin pintaa läpäistä ollenkaan, ja laskuri saa virheellisiä arvoja. Toinen ongelma liittyy kameran sijaintiin. Jos näköpiste on itse varjossa, laskurin alkutila (nolla) on jo lähtökohtaisesti väärin. Sen varjotilavuuden reunoista jossa kamera on, tulee läpäistyksi ainoastaan toinen, eikä laskuri toimi oikein.

Varjomallin sulkeminen onnistuu periaatteessa helposti. Sen loppupää (eli valosta kauempana oleva reuna) voidaan hoitaa siten, että venytetään mallia valosta poispäin kunnes se leikkautuu näkökentän reunoja vasten. Nyt mikään

säde kamerasta näkökentän pisteeseen P ei voi kulkea avoimen reunan läpi. Käytännössä varjomallien takareunat työnnetäänkin aina äärettömyyteen. Varjomallin valoa lähellä oleva pää ei ole käytännössä ongelma, koska syvyyspuskuri pitää huolen siitä, ettei kamerapisteestä lähtenyt säde pysty läpäisemään varjostavaa esinettä. Näin varjostava esine itsessään tukkii varjotilavuuden alkupään.

Varjossa olevan kameran ongelma voidaan yrittää estää alustamalla laskuri johonkin muuhun arvoon kuin nollaan. Jokaisen varjon, jonka sisällä kamera on, pitäisi lisätä alkuarvoa yhdellä. Tällaisten varjojen määrän laskeminen on, kuten aiemmin todettiin, pahimmillaan erittäin raskasta. Lisäksi se ei korjaa tilannetta, jossa varjomalli leikkaa näkökentän etutason vain osittain. Tällainen tapaus aiheuttaa sen, että osa ruudusta varjostuu oikein mutta osa ei. Tällöin ainoa vaihtoehto on leikata ja sulkea varjomalli näkökenttää vasten, mikä on osoittautunut äärimmäisen hankalaksi toimenpiteeksi. Crown menetelmää onkin kehitetty eteenpäin juuri tämän ongelman takia. Kannattaa vielä huomata, että takatason kanssa ongelmaa ei pääse syntymään, koska väärä varjo ilmestyisi ainoastaan takareunan taakse, eikä siellä sijaitsevia pisteitä tietenkään edes piirretä.

Koska jokainen silmästä näkökentän pisteeseen kulkeva säde jossain vaiheessa läpäisee näkökentän etureunan, voidaan varjotarkastelu tehdä pikselikohtaisesti samalla kun kuvaa piirretään. Tästä johtuen menetelmä saadaan hyvin luontevasti yksinkertaistettua kaksiulotteiseksi ongelmaksi. Pikselitason operaatioissa suureen osaan nousevat jo aikaisemmin esitellyt syvyys- ja sapluunapuskurit, joiden avulla piirrettävät pisteet ja niiden varjostus on luontevaa määritellä.

4.1.1. Zpass-menetelmä

Crown säteenseurantamenetelmä sellaisenaan olisi hidas toteutettava, mutta Tim Heidmann [1991] ymmärsi hyödyntää menetelmän kaksiulotteista luonnetta, ja käyttää aiemmin ideoituja näyttöpuskureita [Fournier and Fussell, 1988] varjolaskennan apuna. Lopputuloksena koko varjomallien piirtäminen pystytään suorittamaan pikselikohtaisesti syvyys- ja sapluunapuskurien avulla. On syytä huomauttaa, että vaikka varjotilavuuksia monesti kutsutaan sapluunavarjoiksi (stencil shadow), niiden piirtäminen on mahdollista toteuttaa ilman sapluunapuskuriakin, esimerkiksi alfakanavan avulla [Roettger et al., 2002]. Toistaiseksi keskitytään kuitenkin sapluunan käyttöön, koska nykyisin on käytännössä mahdotonta hankkia tietokonetta, jonka näytönohjaimessa se ei olisi vakiotoiminto.

Heidmann aloittaa projisoimalla pisteen P ruudulle. Kameran ja P:n välissä olevat varjopinnat ovat ne, joiden syvyystarkastelu pisteen P kohdalla menee

läpi. Laskemalla tällaisten pintojen määrä tiedetään, kuinka monen varjopinnan läpi säde kamerasta P:hen kulkee. Kamerasta poispäin osoittavien pintojen määrä vähennetään kameraan päin osoittavien määrästä, aivan kuten Crown menetelmässä.

P:t saadaan selville piirtämällä koko näkymä ensin syvyyspuskuriin ilman varjomalleja. Kuten aiemmin todettiin, sapluunapuskurilla voidaan suorittaa syvyystarkastelun tuloksesta riippuen erilaisia toimintoja, joten laskuri on luontevaa toteuttaa sen avulla. Aluksi kunkin sapluunapisteen alkuarvoksi asetetaan nolla, minkä jälkeen piirretään kameraan päin osoittavat varjomallin monikulmiot. Jos syvyystarkastelu P:n kohdalla menee läpi, eli monikulmion piste on P:n ja kameran välissä, sapluuna-arvoa lisätään yhdellä. Tämän jälkeen asetetaan sapluunaan toiminto, joka vähentää arvoa aina kun syvyystarkastelu menee läpi, ja piirretään kamerasta poispäin osoittavat monikulmiot. Lopputuloksena kullekin P:lle on sapluunassa tieto siitä, kuinka monen varjon sisällä se sijaitsee. Lopullinen näkymä voidaan piirtää asettamalla väripuskuriin kirjoitus päälle ja sapluunalle toiminto, joka piirtää pikselin näytölle vain, mikäli pistettä vastaava sapluuna-arvo on nolla (eli piste ei ole varjossa). Lopputuloksena on näkymä, jonka varjostetut kohdat ovat mustia. Käytännössä ruutu on ennen piirtoa jo alustettu taustavalolla, ja usean valon tapauksessa (luku 2.4) mahdollisesti aiemmin käsiteltyjen valojen valaisemilla näkymillä.

Se, osoittavatko varjon monikulmiot kameraa kohti, saadaan selville esimerkiksi niiden normaalivektorien ja kameran suuntavektorin pistetulolla. Tehokkaampaa ja käytännöllisempää on kuitenkin käyttää 3D-rajapintojen tarjoamaa kaksiulotteista piilopintojen poistoa (OpenGL:ssä toimintoa käytetään glCullFace- ja glFrontFace-funktioiden avulla). Toiminto vaatii, että kaikkien varjomonikulmioiden kärkipisteet kiertävät kaksiulotteisessa tarkastelussa ruudulla samaan suuntaan (yleensä vastapäivään). Tämä ei ole suuri ongelma, mutta vaatii huolellisuutta kun varjomalleja kootaan.

On syytä pitää mielessä, että sapluunapuskuri ei osaa käsitellä negatiivisia kokonaislukuja. Jos sapluuna-arvoon nolla sovelletaan vähennystoimintoa, arvo jää ennalleen. Tämän vuoksi zpass-menetelmä tekee ensin kaikki lisäysoperaatiot, ja vähentää poispäin osoittavien monikulmioiden määrän vasta sitten. Crown algoritmilla on ominaisuus, että mikäli kamera ei ole varjossa ja kaikki varjomallit ovat suljettuja, sen lopputulos ei voi koskaan jäädä negatiiviseksi. On siis mahdollista poistua vain niin monesta varjosta kuin mihin on menty. Rajaaminen tapahtuu myös sapluunapuskurin yläarvoille, eli esimerkiksi lisäysoperaatio kahdeksanbittisen sapluunan pisteeseen joka sisältää arvon 255, aiheuttaa arvon pysymisen samana. Yläraja

on kuitenkin vaikeampi saavuttaa, ja se aiheuttaa harvoin ongelmia. Nykyisin näytönohjaimet tukevat nk. kiertävää sapluunaa (luku 4.3.2), mikä käytännössä poistaa edellä mainitut virhetilanteet.

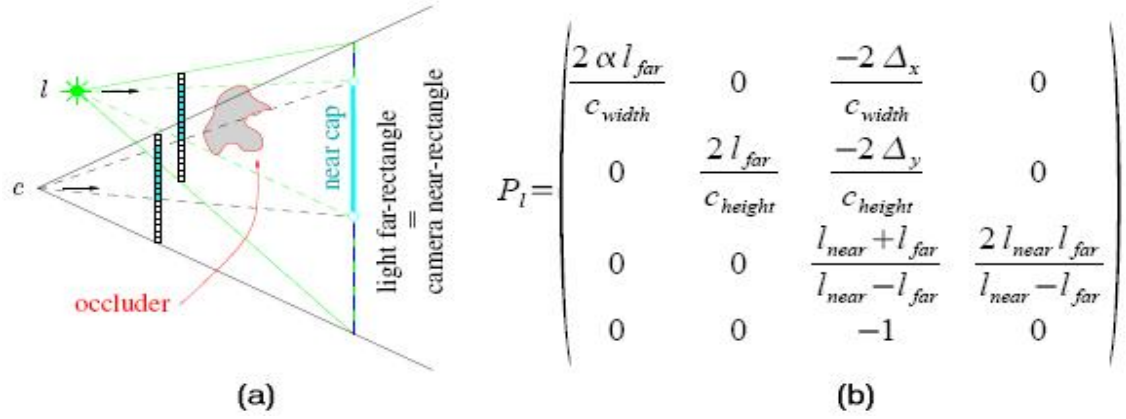
Koska Heidmannin menetelmä perustuu Crown algoritmiin, se sisältää myös samat perusongelmat. Varjossa oleva kamera oireilee joko varjojen katoamisina tai niiden muuttumisena käänteisiksi. Samankaltainen ongelma syntyy, jos takaapäin lankeava varjo leikkaa näkökentän etureunan. Ainoa keino selvitä ongelmasta on etsiä leikkauskohta varjomallin ja näkökentän etureunan välillä, ja täyttää avoimeksi jäävä alue keinotekoisesti. Tehtävä on monimutkainen, koska varjoja voi olla paljon ja ne voivat osua näkökenttään monista eri paikoista aiheuttaen runsaasti erikoistapauksia. Everitt ja Kilgard [2002] listaavat ratkaisuyrityksiä sekä ongelmia, joihin ne ovat kaatuneet. Esimerkiksi Diefenbachin [1996] menetelmä ei ole yleispätevä, ja useat muut kärsivät numeerisista epätarkkuuksista [Batagelo et al., 1999]. Pyöristysepätarkkuudet ovatkin suurin este matemaattisesti toimivien ratkaisujen käyttöön. Ne esimerkiksi estävät varjomallin suoran projisoinnin näkökentän etureunaan. Ongelmat ilmenevät mm. varjossa näkyvinä aukkoina ja päällekkäisyyksinä. Hankaluutena kaikissa menetelmissä on myös se, että ne on tehtävä pääasiassa keskusprosessorilla. Tämä sekä siirtää piirtovastuuta pois näytönohjaimelta että edelleen lisää epätarkkuutta (sama laskutoimitus ei välttämättä tuota samaa likiarvoa keskusprosessorilla ja näytönohjaimella). Tämän vuoksi ongelma on käytännössä ratkaistava kaksiulotteisesti.

4.1.2. ZP+-menetelmä

Kaikista etutason leikkausongelmiin kehitetyistä menetelmistä, ZP+-algoritmi [Hornus et al., 2005] lienee pätevin. Se pyrkii korjaamaan virheet luomalla virtuaalisen kameran valon sijaintipisteeseen ja suuntaamalla sen siten, että valon ja kameranäkymän pisteet vastaavat tarkalleen toisiaan. Tämä mahdollistaa näkökentän etutasolle lankeavien varjojen paikantamisen pikselitasolla, ja vähentää merkittävästi numeerisista epätarkkuuksista johtuvia virheitä. Tätä ylimääräistä piirtokertaa, sekä muutamaa muuta myöhemmin esiteltävää vaihetta lukuun ottamatta, ZP+-menetelmä toimii täsmälleen kuten zpass-algoritmikin, eikä vaadi myöhemmin esiteltävien, vielä yleiskäyttöisempien menetelmien tapaan suljettuja varjomalleja.

Ihannetilanteessa ZP+-menetelmä ei tuota uusia monikulmioita näkökentän etutasoa peittämään. Sen sijaan se luo projektion alueesta, joka ulottuu valosta kameran etutasolle. Tällä alueella sijaitsevat monikulmiot ovat ne, jotka aiheuttavat zpass-menetelmän ongelmat. Projektio tehdään luomalla vino näkökenttä, jonka kärki on valon sijaintipisteessä, ja takareuna on kohdistettuna tarkasti kameran näkökentän etureunaan (kuva 8a). Jos valo ja

kamera sijaitsevat kameran näkökentän samalla puolella, molempien katsesuunta on sama. Muussa tapauksessa ne ovat päinvastaiset. Valon näkökentän etureuna asetetaan siten, että se on varjostavan esineen takana mutta kuitenkin sen verran kaukana valosta, ettei pikselikohtainen tarkkuus kärsi. Hornus et al. [2005] antavat sen määrittämiseen kaavan: $l_n = l_f d / d_{max}$, missä l_f on valon takareunan etäisyys, d on etäisyys valosta varjostavaan esineeseen, ja d_{max} on etäisyys valosta kaukaisimpaan kameran etureunan pisteeseen. Kun näkökenttä on luotu, sille voidaan määritellä projektiomatriisi, jonka avulla valonäkymä on helposti piirrettävissä normaaliin tapaan (kuva 8b).



Kuva 8. ZP+-menetelmän näkökenttä (a) ja sen projektiomatriisi (b) [Hornus et al., 2005]. l on valo ja c kamera. Matriisissa Δ on l – c, ja α on 1, jos kamera ja valo ovat kameran etutason samalla puolella. Muussa tapauksessa se on -1.

Koska pikselit valon ja kameran piirtotasolla vastaavat toisiaan, etureunaan lankeavat varjot piirtyvät suoraan sapluunapuskuriin. Korjaustoimenpide tehdään piirtämällä kaikki varjostavan mallin valoon päin osoittavat monikulmiot, ja lisäämällä sapluunan arvoa jokaisessa piirretyssä pisteessä. Kun tämä on tehty, kaikki sapluunapuskurin pikselit sisältävät oikeat alkuarvot, ja näkymä voidaan nyt varjostaa normaalisti kamerasta katsottuna.

ZP+-algoritmin epätarkkuudet eivät ole samaa luokkaa kuin kolmiulotteisissa korjausmenetelmissä, mutta niitä ilmenee jonkin verran kameran etutason ja varjomonikulmioiden leikkauspisteissä. Vaikka virheet eivät yleensä olekaan kovin näkyviä [Hornus et al., 2005], on yleispätevyyden takia keksittävä keino, jolla ne saadaan korjattua. Tämä vaatii ensin vääriä pikseleitä tuottavan monikulmion ja kameran etutason leikkauspisteiden laskemista. Tämän jälkeen täytyy vielä piirtää monikulmiot korjaamaan virheelliset alueet. Korjausmenetelmä sisältää useita erikoistapauksia, ja suurimpana ongelmana ovat tilanteet, joissa varjomallin reunat kulkevat näkökentän molemmin puolin, sitä kuitenkaan leikkaamatta. Näitä tilanteita ei

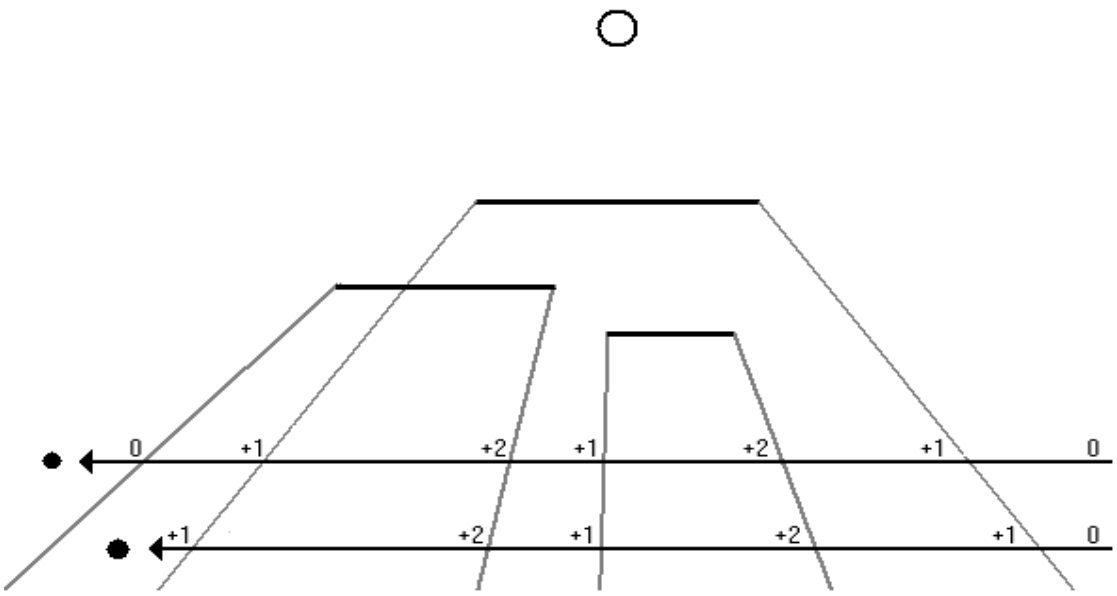
Hornusin et al. [2005] mukaan ole mahdollista korjata ollenkaan, vaan on siirryttävä vielä yleisemmän menetelmän käyttöön.

4.1.3. Zfail-menetelmä

Zfail-menetelmä ratkaisee zpass-algoritmin ongelmat hyvin yleispätevällä tavalla. Sen idea on pohjimmiltaan yksinkertainen, ja se keksittiinkin kahdessa paikassa samoihin aikoihin. Bill Bilodeau ja Mike Songy [1999] ehtivät patentoimaan sen ennen kuin John Carmack [2000] ehti julkaista oman versionsa. Nimi zfail viittaa siihen, että Crown ja Heidmannin syvyystarkastus käännetään toisin päin. Bilodeaun ja Songyn menetelmä tunnettiin aluksi nimellä reversed zpass, ja Carmackin keksintöä kutsutaan myös nimellä Carmack's reverse. Zfail-algoritmilla tarkoitetaan ensisijaisesti Carmackin näkemystä asiasta. Bilodeaun ja Songyn menetelmä toimii hieman eri tavalla syvyystarkastelun suhteen, mutta on muuten idealtaan täysin sama.

Zfail-menetelmä siirtää näkökentän etutason ongelmat takatasolle, koska siellä ne ovat helpommin ratkaistavissa. Toisin kuin zpass-menetelmä, zfail ei laske etutason ja P:n välissä olevia varjomonikulmioita, vaan P:n ja takareunan välissä olevat. Sapluunaoperaatiot suoritetaan nyt niille varjomallin pisteille, jotka ovat P:n takana, eli joiden syvyystarkastelu epäonnistuu.

Zfail-menetelmän toiminta alkaa kuten edellisessäkin algoritmissa, eli aluksi piirretään näkymä syvyyspuskuriin. Sitten kytketään syvyyspuskuriin kirjoittaminen pois päältä ja asetetaan (tyhjennetylle) sapluunapuskurille toiminto, joka lisää pisteen arvoa yhdellä, mikäli syvyystesti epäonnistuu. Tämän jälkeen piirretään kamerasta poispäin osoittavat varjomonikulmiot. Menetelmä on siis täysin sama kuin zpass-algoritmissa, mutta se tehdään suhteessa näkökentän takareunaan. Tätä logiikkaa seuraten asetetaan sapluuna vähentämään pisteen arvoa jos syvyystesti epäonnistuu, minkä jälkeen piirretään kameraan päin osoittavat varjomonikulmiot. Lopputulos on sama kuin zpass-menetelmässä, eli sapluuna-arvo nolla tarkoittaa pisteen olevan valossa (kuva 9). Erona on kuitenkin se, että P:n ja näkökentän etureunan välissä olevat varjopinnat eivät enää vaikuta lopputulokseen. Näin ollen sillä, onko kamera itse varjossa, ei ole merkitystä.



Kuva 9. Zfail-menetelmä. Nyt alemmankin säteen laskuri toimii oikein (vrt. kuva 7).

Ennen kuin zfail-menetelmä on täydellinen, on keksittävä ratkaisu takareunan leikkausongelmalle. Everitt ja Kilgard [2002] kehittivät tällaisen. Pohjana heillä oli Jim Blinnin [1993] kehittämä projektiomatriisi (kuva 10), joka kykenee työntämään perspektiivisen näkökentän takareunan äärettömän kauas. Tämä toimenpide yhdessä varjomallien sulkemisen (joka käsitellään vähän myöhemmin) kanssa ratkaisee kaikki näkökentän leikkauksiin liittyvät ongelmat.

$$\lim_{f\to\infty} P = P' = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Kuva 10. Perspektiivinen projektiomatriisi joka työntää näkökentän takareunan äärettömyyteen. Kirjaimet viittaavat näkökentän reunojen sijaintiin etutasolla samaan tapaan kuin OpenGL:n funktiossa glFrustum, josta matriisi on johdettu.

Näkökentän takareunan siirtäminen äärettömyyteen ei vaikuta merkittävästi syvyyspuskurin tarkkuuteen. Itse asiassa vaikutus on marginaalinen, suurimmillaankin käytännössä vain prosentin luokkaa [Everitt and Kilgard, 2002]. Matriisin puutteena on kuitenkin se, että takareuna voidaan viedä äärettömyyteen ainoastaan perspektiivisillä näkökentillä. Jos halutaan

käyttää ortografista projisointia, tämä lähestymistapa ei ole mahdollinen. Everittin ja Kilgardin ehdotuksesta nykyiset näytönohjaimet mahdollistavat kuitenkin näkökentän etu- ja takareunalle tehtävien näkymäleikkausten poistamisen käytöstä nk. syvyysrajaimen (depth clamp) avulla. Mikäli leikkaus estetään, takareunan ongelma poistuu äärellistenkin näkökenttien yhteydessä, jolloin ortografiset projektiot ovat mahdollisia.

Merkittävämpi ongelma juontaa juurensa varjomallien sulkemiseen. Zpass-menetelmässä varjostava malli tukki varjomallien etureunat, ja takareunat voitiin jättää avonaisiksi, kunhan varjomalleja vain venytettiin vähintään näkökentän reunoihin. Tämä ei valitettavasti riitä zfail-menetelmän yhteydessä. Syy on että säde, joka zpass-menetelmässä lähetettiin P:tä kohti kameran sijaintipisteestä, voidaan nyt käsittää äärettömän monena säteenä jotka kaikki lähtevät P:tä kohti näkökentän takareunalta. Jos säde lähtee sellaisesta paikasta jossa varjomalli leikkautuu, se on jo valmiiksi kyseisen varjomallin sisällä, ja näin sen alkuarvo nolla on väärin. On huomattava, että tämä ongelma koskee myös äärettömiä näkökenttiä, mutta niiden kohdalla virhe voidaan korjata sulkemalla varjomallit. Tällöin äärettömyydestä lähtevä säde ei enää pääse varjomallin sisään kulkematta jonkun sen reunan läpi, ja laskuri toimii oikein. Varjomallien sulkeminen tekee zfail-menetelmästä yleispätevän, mutta zpass-algoritmia monimutkaisemman ja hitaamman.

4.2. Varjomallien koostaminen

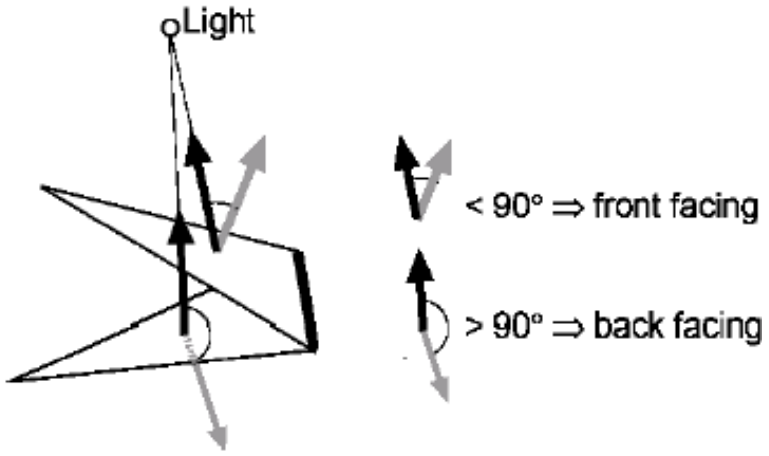
Varjotilavuuden luominen varjostavan mallin jokaiselle monikulmiolle erikseen olisi siis periaatteessa täysin toimiva ratkaisu, mutta käytännössä se johtaisi niin suureen varjojen määrään, ettei se reaaliaikaisuuden näkökulmasta ole järkevää. Tehokkuussyyt kannustavatkin varjomallien lukumäärän minimoimiseen, sekä niiden rakenteen yksinkertaistamiseen.

Paras tapa tämän saavuttamiseksi on laskea varjostavan esineen ääriviivat, ja koostaa varjomallit niiden avulla. Tällöin voidaan parhaassa tapauksessa päästä siihen, että kutakin esinettä varten tarvitaan vain yksi varjomalli. Täsmällisten ääriviivojen löytäminen on kuitenkin matemaattisesti monimutkainen toimenpide, ja siksi onkin järkevintä pyrkiä löytämään hyvä kompromissi siluettien määrittämiseen ja mallien piirtämiseen käytettävän ajan väliltä. Monimutkaisista silueteista luotujen varjomallien sulkeminen on myös paljon vaikeampaa kuin yksinkertaisempien.

4.2.1. Mallin ääriviivojen määrittäminen

Crow [1977] käyttää omassa varhaisessa menetelmäkuvauksessaan nk. Sutherlandin mahdollisten siluettien algoritmia [Sutherland, 1973]. Sen mukaan monikulmioista koostuvan mallin mahdollinen siluettireuna on sellainen, johon

kiinnittyneistä monikulmioista toinen osoittaa valoa kohti, ja toinen valosta poispäin (kuva 11). Tämä ei tietenkään takaa täydellistä siluettireunaa koverille esineille, mutta on toiminut hyvänä kompromissina näihin päiviin saakka. Kun tarkastus tehdään kaikille mallin reunoille, saadaan lopputulokseksi lista mahdollisista siluettireunoista. Jos malliin ei ole tallennettu tarkempaa tietoa reunoista, ne etsitään käymällä läpi kaikki mallin monikulmiot.



Kuva 11. Mahdollisen siluettireunan määrittäminen [Brabec and Seidel, 2003]

Monikulmion suunta valoon nähden saadaan selvitettyä laskemalla sen normaalivektorin ja siitä valoon kulkevan vektorin pistetulo. Positiivinen arvo tarkoittaa monikulmion osoittavan valoa kohti, negatiivinen taas päinvastaista. Laskennan yhteydessä kannattaa muistaa, että monikulmio on oltava määriteltynä samassa avaruudessa kuin valo. Mikäli siluettilaskenta tehdään keskusprosessorilla, on nopeinta muuntaa valopiste varjostavan esineen paikalliseen koordinaatistoon, ja muuntaa varjomalli globaaliin avaruuteen varjostavan esineen matriisilla. Joissakin tilanteissa valon ja monikulmioiden saattaminen samaan avaruuteen on kuitenkin haasteellista. Vaikeuksia syntyy esimerkiksi ihomallinnettujen tai muuten dynaamisesti näytönohjaimella animoitujen esineiden kanssa. Tästä aiheesta lisää luvussa 4.2.3.

Philippe Bergeron [1986] laati Crown algoritmista yleisen menetelmän, joka kykenee tukemaan myös kokeellisempia piirtomenetelmiä. Tähän ominaisuuteen ei syvennytä enempää, koska käytännössä kaikki reaaliaikamallit perustuvat nykyisin tasapintaisista kolmioista koostuviin esineisiin. Mielenkiintoisempana yksityiskohtana Bergeron esittää yksinkertaisen tavan, jolla varjomallinnus saadaan tukemaan myös avoimia malleja. Avoimet reunat (jotka liittyvät vain yhteen monikulmioon) tulkitaan siluettireunoiksi. Näille tehtävät varjonelikulmiot tallennetaan omaan listaansa ja piirretään muista erillään. Kun kameraa kohti osoittavaa, avoimesta reunasta luotua varjopintaa piirretään, sapluuna-arvoa lisätään tavalliseen

tapaan, mutta piirrettäessä tavallista, suljetusta reunasta tehtyä varjomonikulmioita, sapluuna-arvoa lisätäänkin kahdella. Poispäin osoittavien reunojen kohdalla arvoa vähennetään samaan tapaan (zfail-menetelmässä operaatiot ovat toisinpäin). Jatkossa kuitenkin oletetaan kaikkien varjotilavuuksin varjostettavien mallien olevan suljettuja. Tämä siitäkin syystä, ettei kahdella lisäys ole yleinen toiminto sapluunapuskureissa.

Ääriviivojen analyyttistä laskemista on tutkittu paljon (mm. [Barequet et al., 2001] ja [Markosian et al., 1997]). Niiden avulla on mahdollista luoda erittäin pitkälle optimoituja varjomalleja, mutta johtuen laskennan epätriviaalista luonteesta, ne joudutaan tekemään puhtaasti keskusprosessorilla. Lisäksi monet analyyttiset menetelmät vaativat esilaskettua tietämystä piirrettävistä malleista, ja näiden tietojen jatkuva päivittäminen muuttuvassa näkymässä voi olla raskasta. Näistä menetelmistä on kuitenkin apua, jos halutaan luoda suljettuja siluettireunoja, joista on hyötyä varsinkin pehmeiden varjojen muodostamisessa (ks. luku 6.3.1).

4.2.2. Suljettu ja ääretön varjotilavuus

Kärkipisteet voidaan työntää äärettömyyteen nollaamalla niiden w-koordinaatti. Tätä ominaisuutta apuna käyttäen jokaisesta siluettireunasta luodaan nelikulmio, jonka etureunan (valosta katsottuna) muodostavat alkuperäisen reunan kärkipisteet. Takareuna taas työnnetään valon ja pisteen sijaintia sekä w-koordinaattia hyväksikäyttäen äärettömyyteen. Kilgardin [2001a] menetelmällä varjomallista pystytään samalla periaatteella tekemään suljettu.

Varjomalli luodaan käymällä läpi kaikki varjostavan mallin reunat, ja tarkastelemalla reunaan liittyviä kahta monikulmiota (joiden oletetaan tässä tapauksessa olevan kolmioita). Oletetaan, että valo sijaitsee pisteessä L. Jokaiselle varjostavan mallin reunalle tehdään seuraava operaatio.

Jos reuna on mahdollinen siluettireuna, luodaan äärettömyyteen jatkettu varjonelikulmio, jonka ääripisteet ovat A, B, (A.xL.w – L.xA.w, A.yL.w – L.yA.w, A.zL.w – L.zA.w, 0) ja (B.xL.w – L.xB.w, B.yL.w – L.yB.w, B.zL.w – L.zB.w, 0), missä A ja B ovat ääriviivan kärkipisteet. Tässä vaiheessa on tärkeää pitää huoli, että luodun nelikulmion pisteet ovat oikeassa järjestyksessä.

Jos molemmat reunaan liittyvät kolmiot osoittavat valosta poispäin, luodaan varjokolmio, jonka ääripisteet ovat (A.xL.w – L.xA.w, A.yL.w – L.yA.w, A.zL.w – L.zA.w, 0), (B.xL.w – L.xB.w, B.yL.w – L.yB.w, B.zL.w

– L.zB.w, 0) ja (C.xL.w – L.xC.w, C.yL.w – L.yC.w, C.zL.w – L.zC.w, 0), joissa A, B ja C ovat varjostavan mallin kolmion kärkipisteet.

Jos molemmat reunaan liittyvät monikulmiot osoittavat valoon päin, luodaan varjokolmio, jonka ääripisteet ovat A, B ja C, eli samat kuin varjostavan mallin pisteet.

Ideana on siis käyttää varjostavan mallin omia monikulmioita tukkimaan varjotilavuuden etu- ja takareunat. Luoduista kolmioista saadaan koostettua äärettömyyteen jatkuva ja suljettu varjomalli. Jättämällä kaksi viimeistä vaihetta pois, menetelmällä on mahdollista tehdä myös avoin malli zpass-menetelmää varten.

4.2.3. Varjotilavuuden laskeminen näytönohjaimella

Brabec ja Seidel [2003] luettelevat etuja, joita saavutetaan kun varjomallit lasketaan kokonaisuudessaan näytönohjaimella. Ensinnäkin silloin voidaan taata saman laskutoimituksen tuottavan aina saman likiarvon. Jos sama laskutoimitus tehdään sekä keskusprosessorilla että näytönohjaimella, lopputuloksissa voi olla pieniä mutta kiusallisia eroja, jotka aiheuttavat ongelmia, kuten etureunan sulkemisen kohdalla aiemmin todettiin. Lisäksi kolmiulotteisten mallien tallentaminen ja käsittely keskusprosessorin puolella muuttuu helpommaksi, kun sen ei tarvitse huolehtia mallien muuntamisesta. Keskusprosessori myös vapautuu siluettilaskennasta, jolloin näytönohjaimen ei tarvitse odotella prosessin päättymistä, vaan transformointi ja piirtäminen voidaan suorittaa taustalla. Viimeinen etu liittyy siihen, että kaikki varjomalleihin liittyvä tieto pystytään tallentamaan näytönohjaimen muistiin. Tämä nopeuttaa piirtämistä huomattavasti, eikä aikaa kulu suuren tietomäärän jatkuvaan kuljettamiseen prosessoreiden välillä. Koska nykyiset näytönohjaimet pystyvät muokkaamaan malleja reaaliaikaisesti, tuntuisi tuhlaukselta tehdä sama keskusprosessorilla, ja sen jälkeen syöttää tieto valmiiksi muunnettuna lähes tyhjän muunnoskanavan läpi ruudulle. Näytönohjainten kasvava nopeus myös kannustaa kuormittamaan niitä mahdollisimman paljon, ja antamaan keskusprosessorille aikaa keskittyä muuhun ohjelmalogiikkaan.

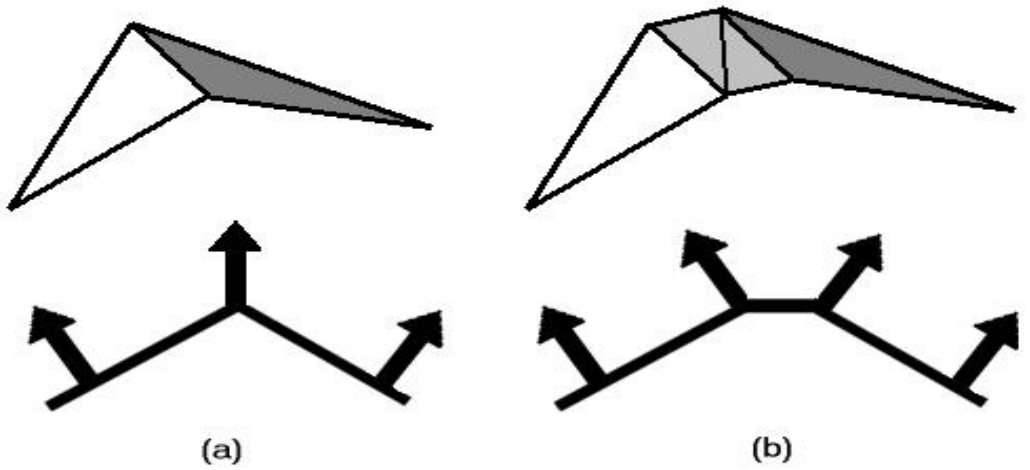
Ihomallit (skin meshes) ovat tyypillinen esimerkki näytönohjainten mahdollisuuksista sekä ongelmista, joita aiheutuu kun osa toiminnallisuudesta joudutaan pilkkomaan prosessorien kesken. Ihomalli on tavallinen kärkipisteistä ja niitä yhdistävistä monikulmioista koostuva malli, mutta sen erottaa muista malleista se, että kukin kärkipiste muunnetaan globaaliin avaruuteen useamman eri muunnosmatriisin avulla. Matriisien vaikutus

muunnokseen määritetään antamalla pisteelle jokaista vaikuttavaa matriisia kohti painoarvo. Mitä suurempi painoarvo on, sitä suurempi on myös matriisin vaikutus. Painoarvojen summa on aina yksi.

Nykyisissä näytönohjaimissa tällaiset mallit on helppoa ja tehokasta muuntaa verteksivarjostimissa. Varjostuksen näkökulmasta ongelmaksi kuitenkin muodostuu se, että koska laskenta tehdään näytönohjaimella, keskusprosessori ei enää missään vaiheessa pääse käsiksi globaaliin avaruuteen muunnettuun malliin, josta varjosiluetit pitäisi laskea. Mikäli siluetti halutaan määrittää analyyttisin menetelmin, joudutaan varjomallien lisäksi suuri osa muunnoskanavaa toteuttamaan keskusprosessorilla, mikä luonnollisesti heikentää näytönohjaimista saatavaa hyötyä. Geometriavarjostimet mahdollistavat siluettilaskennan näissäkin tapauksissa kokonaan näytönohjaimella, mutta koska toimintoa ei ole vielä olemassa lähellekään kaikissa laitteistoissa, on tarpeen käydä läpi pari menetelmää, joiden avulla ihomallit on mahdollista varjostaa tehokkaasti lähes kokonaan näytönohjaimella.

Chris Brennanin [2002] menetelmä mahdollistaa varjomallin pitämisen näytönohjaimen muistissa muuttumattomana koko ohjelman suorituksen ajan. Hintana tosin on, että tietoa joudutaan syöttämään muunnoskanavaan varsin runsaasti. Algoritmi myös automaattisesti sulkee kaikki varjomallit. Menetelmä luo esilasketun varjomallin, joka jo valmiiksi sisältää reunoista muodostettavat nelikulmiot, että näytönohjain pystyy käsittelemään sitä ilman muuta tietoa. Varjomallin luonti tapahtuu tekemällä seuraavat toimenpiteet kaikille varjostavan mallin kolmioille.

Ensin kopioidaan kolmio varjomalliin ja lasketaan sen normaali. Tämä normaali asetetaan kolmion kaikille kärkipisteille. Sen jälkeen käydään läpi kaikki kolmion reunat. Mikäli reuna on jo käsitelty (jokainen reuna liittyy kahteen kolmioon, eli tulee suljetussa mallissa käsitellyksi kahteen kertaan), luodaan nelikulmio, joka koostuu aiemmin käsitellyn ja nyt käsiteltävän reunan pisteistä. Sitten poistetaan reuna käsiteltyjen reunojen listalta. Mikäli reunaa ei ole käsitelty, se lisätään käsiteltyjen reunojen listaan. Tärkeää on kiinnittää huomiota siihen, että kun reuna tulee käsitellyksi toiseen kertaan, aiemmin käsitellyn reunan kärkipisteillä on sama sijainti, mutta eri normaali. Tämä johtuu siitä, että kaikki kolmion pisteet saavat nimenomaan kolmion itsensä normaalin (kuva 12). Toisin sanoen, jokaista reunaa kohden luodun nelikulmion vastakkaiset reunat osoittavat viereisen kolmion osoittamaan suuntaan.



Kuva 12. Varjostavan mallin kaksi kolmiota normaalivektoreineen (a), ja niiden pohjalta luodun varjomallin vastaavat pisteet ja normaalit (b). Reunakolmioita on hieman venytetty, että se näkyy paremmin.

Lopputuloksena varjomalli sisältää kaikki alkuperäisen varjostavan mallin kolmiot sekä nelikulmiot, jotka on muodostettu varjostavan mallin reunoista. Se, mitkä näistä nelikulmioista lopulta venytetään, jää näytönohjaimen päätettäväksi. Varjomallin sisältämän tiedon perusteella on hyvin yksikertaista tehdä verteksivarjostin, joka laskee pistetulon saamansa kärkipisteen normaalin ja pisteestä valoon kulkevan vektorin välillä. Jos tulos on negatiivinen, piste työnnetään äärettömyyteen. Muussa tapauksessa se pidetään paikallaan. Varjostin ei siis tarvitse mitään erillistä tietoa mallista, ja pystyy toimimaan itsenäisesti kunkin pisteen kohdalla. Tämä mahdollistaa varjomallin pitämisen näytönohjaimen muistissa.

Edellinen menetelmä toimii hyvin muuttumattomien mallien kohdalla, mutta ihomallien toteuttaminen vaatii hieman lisää työtä. Varjomallin pisteiden normaalit laskettiin vastaamaan vierekkäisten monikulmioiden normaaleita, mutta ihomallien ominaispiirteisiin kuuluu, että monikulmion eri pisteet voivat muuntua eri tavalla painotuksistaan riippuen. Tästä johtuen monikulmioiden muoto ja suunta toisiinsa nähden eivät pysy samana, ja menetelmä tuottaa virheellisiä tuloksia, koska väärät kärkipisteet voivat tulla työnnetyksi äärettömyyteen.

Brennan [2002] itse suosittelee tapaa, jossa painotukset lasketaan myös varjomallien monikulmioille. Tämä tehdään yksinkertaisesti laskemalla niihin liittyvien kärkipisteiden keskiarvot kunkin matriisin painoarvolle. Nyt monikulmioiden normaalit voidaan laskea painotusten mukaan, ja lopputuloksena ne saavat oikean arvon. Korjaus voidaan siis tehdä jo esilaskentavaiheessa, eikä sillä ole merkitystä itse piirtämisen nopeuteen.

Kilgard ja Everitt [2003] puuttuvat Brennanin menetelmän suurimpiin heikkouksiin. Menetelmä kasvattaa muunnoskanavaan syötettävän tiedon määrää merkittävästi, ja pakottaa käyttämään suljettuja varjotilavuuksia. Verteksivarjostimet eivät välttämättä takaa tehonnousua, jos ne joutuvat käsittelemään liian suuren määrän pisteitä. Tällöin keskusprosessorivetoinen ratkaisu voi muuttua nopeammaksi. Hyväksi havaittu kompromissi on käyttää keskusprosessoria laskemaan mahdolliset siluettireunat, ja syöttää tämän jälkeen niiden indeksit näytönohjaimelle, minne pisteiden koordinaatit on tallennettu jo aikaisemmin ([Kilgard and Everitt, 2003] ja [van Waveren, 2005]).

Kun varjomalli luodaan, jokainen kärkipiste tallennetaan näytönohjaimelle kahtena peräkkäisenä kopiona. Ensimmäinen sisältää homogeenisen koordinaatin, joka on täsmälleen sama kuin varjomallissa. Toisessa w-koordinaatti on nolla. Ainoastaan kärkipisteiden sijainnit tarvitsee tallentaa. Näin näytönohjaimella käsiteltävän tiedon määrä on paljon pienempi (8 desimaaliarvoa kärkipistettä kohti) kuin Brabecin menetelmässä.

Siluettimääritykset tehdään keskusprosessorilla. Jos piirrettävä siluettireuna muodostuu kärkipisteistä joiden indeksit ovat A ja B, näytönohjaimelle annetaan ohje piirtää nelikulmio, jonka kärkipisteiden indeksit ovat A*2, B*2, A*2+1 ja B*2+1. Täytyy muistaa, että pisteiden järjestyksellä on merkitystä eteen- ja taaksepäin osoittavien reunanelikulmioiden määrittelyssä. Oikea suunta riippuu siitä, mihin suuntaan käsiteltävä reuna kulkee valoon päin osoittavassa kolmiossa. Myös varjomallin alku- ja loppupää voidaan tarvittaessa syöttää verteksivarjostimelle helposti. Alkupään kolmioiden indeksit ovat A*2, B*2 ja C*2 ja loppupään vastaavasti A*2+1, B*2+1 ja C*2+1, missä A, B ja C ovat varjostavan mallin kolmion kärkipisteiden indeksit. Indeksit kerrotaan kahdella, koska näytönohjaimella olevaan kärkipistelistaan on tallennettu kärkipisteet kahteen kertaan. Ensimmäinen on paikalleen jäävä piste, ja toinen on äärettömyyteen työnnettävä.

Verteksivarjostin voidaan kuvata yksinkertaisimmillaan seuraavalla kaavalla [Kilgard and Everitt, 2003]: $P' = P * P.w + (P * L.w - L * P.w) * (1 - P.w)$, missä P on pisteen ja L valon sijainti. Käytännössä verteksivarjostimen on huolehdittava myös mahdollisista ihomuunnoksista. Koordinaattiparin ensimmäinen piste jää aina paikalleen, mutta jälkimmäinen työnnetään äärettömän kauas valosta, eikä näytönohjaimen tarvitse tietää mitään muuta kuin pisteen indeksi.

Tämän menetelmän heikkoutena on luonnollisesti se, että pisteiden muunnokset globaaliin avaruuteen joudutaan tekemään keskusprosessorilla. Toisaalta muunnoskanavan läpi kulkevien pisteiden määrä on huomattavan paljon pienempi kuin Brabecin algoritmissa.

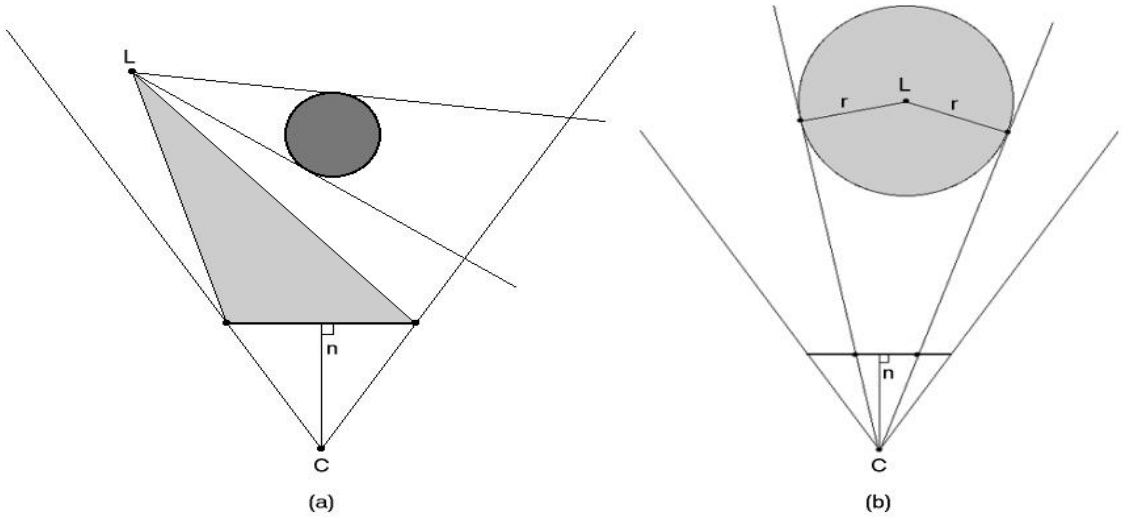
4.3. Varjotilavuusmenetelmän toteutuksesta

Varjotilavuudet ovat hyvin raskaita laskettavia, ja vievät aktiivisesti käytettynä merkittävän osan ohjelmien suoritusajasta. Sen vuoksi on tärkeää perehtyä tapoihin, joilla suoritusta saadaan tehostettua. Varjotilavuuksissa optimointi keskittyy ensisijaisesti piirrettävien pikseleiden määrän minimointiin.

4.3.1. Optimointi

Tietotekniikassa joudutaan yleensä turvautumaan kompromisseihin menetelmien yleispätevyyden ja tehokkuuden välillä. Samoin on asia myös varjomallien kohdalla. Zpass-menetelmä on ongelmineenkin monessa suhteessa tehokkaampi kuin Zfail. Tämä lähinnä siksi, ettei varjomallien tarvitse olla suljettuja, ja täten niiden luominen on huomattavasti helpompaa. Zpass-algoritmia tulisikin suosia aina kun pakottavaa tarvetta zfailin käytölle ei ole. Menetelmien sekoittaminen on mahdollista, koska niitä voidaan vaihdella mallien välillä ilman, että näkymän kokonaisvarjostus menee pieleen. Sen sijaan saman mallin kohdalla menetelmien sekoittaminen on vaikeampaa, mm. koska ääretöntä (zfail) ja yleensä äärellistä (zpass ja ZP+) näkökenttää käyttävien menetelmien syvyysarvot eivät ole yhteensopivia [Hornus et al., 2005].

Periaatteessa zfailia tarvitaan vain, kun varjomalli leikkaa näkökentän etureunan (eikä aina silloinkaan, jos ZP+ menetelmä toimii). Eric Lengyel [2002] kehitti ZP+-menetelmää ennakoineen, yksinkertaisen tavan selvittää, milloin tämä tapahtuu. Tarvitsee vain luoda kalteva nelisivuinen pyramidi, jonka huippu on valon sijaintipisteessä, ja alareunan pisteet näkökentän etureunan kulmapisteissä. Mikäli jokin näkymän esineistä leikkaa tämän pyramidin se tarkoittaa, että esine on valon ja näkökentän etureunan välissä (kuva 13a). ZP+ -tyylisen piirtämisen sijaan tarkastelu voidaan tehdä esimerkiksi vertaamalla kunkin esineen kapseloimia yksinkertaisia geometrisia malleja, kuten palloja tai laatikoita, pyramidiin. Tämä johtaa hieman epätarkkaan tulokseen (jolloin zfail-menetelmää käytetään enemmän kuin on tarpeen), mutta on varsinkin suurissa näkymissä laskennallisesti tehokas toteuttaa.



Kuva 13. Näkökentän ja valonlähteen yhdistävä tilavuus (a), sekä valon vaikutusalueen projisointi näkökenttään (b) [Lengyel, 2002]. L ja C ovat valon ja kameran sijaintipisteet, r on valon vaikutusalueen säde ja n on kameran etutason etäisyys.

Avoimien varjomallienkaan käyttäminen ei kuitenkaan ratkaise sitä ongelmaa, että äärettömyyteen jatkuvat varjotilavuudet peittävät monesti ruudusta suuria alueita. Lisäksi varjomalleista on piirrettävä sekä kameraan päin että siitä poispäin osoittavat monikulmiot, mikä kasvattaa piirrettävien pikseleiden määrää entisestään. On syytä huomata, että pienikin varjo voi vaatia valtavan määrän suuria varjomonikulmioita.

Lengyel [2002] jatkaa optimointia käyttämällä 3D-rajapintojen tarjoamaa saksitestausta (scissor test), jolla voidaan rajoittaa ruudulta suorakulmion muotoinen piirtoalue. Kaikki sen ulkopuolelle kohdistuvat piirto-operaatiot keskeytetään jo ennen sapluuna- tai syvyyspuskuria. Koska näkymä joudutaan piirtämään erikseen jokaiselle valolle, kannattaa sakset asettaa estämään sellaisten pisteiden käsittely, mihin valo ei vaikuta. Lengyelin tapa rajoittaa piirrettävää aluetta on antaa valoille vaikutusalue. Sitä voidaan kuvata pallolla, jonka keskus on valon sijaintipisteessä, ja jonka säde on valon kantama (kuva 13b). Tällainen pallo on helppo projisoida normaalin muunnoskanavan läpi ruudulle, ja muodostaa sitten pienin mahdollinen suorakulmio, joka kapseloi sen sisäänsä. Kun tämä suorakulmio annetaan näytönohjaimen saksitoiminnolle, ainoastaan valon vaikutusalueella olevat pisteet piirretään ruutuun.

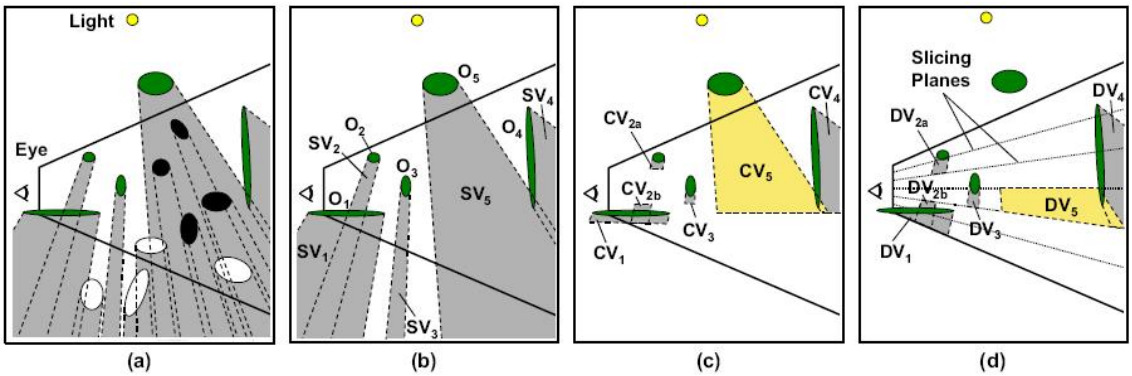
Toinen tehostamiskeino on Everittin ja Kilgardin [2003] ideoima syvyysrajoitin (depth bounds), joka suoritetaan aktivoituna ollessaan heti saksitestauksen jälkeen. Sen toimintakin muistuttaa saksitestausta. Erona on, että piirrettävät pisteet rajataan syvyysarvojen mukaan. Jos pisteen z-

koordinaatti ei ole syvyysrajoittimen määrittelemällä alueella, piste hylätään. Syvyysrajoitinta voidaan käyttää Lengyelin mallin mukaisesti rajoittamaan näkymän piirtämistä sellaisella alueella, jonne valon vaikutus ei ulotu. Vastaavasti varjomalleja piirrettäessä voidaan estää sapluunaoperaatiot sellaisille pisteille, joiden syvyysarvo on valon kantamattomissa.

Zpass- ja zfail-menetelmät, saksitestauksen sekä syvyysrajoittimen voi yhdistää tehokkaasti [McGuire et al., 2003]. Aluksi luodaan Lengyelin [2002] esityksen mukainen pyramidi valosta näkökentän etureunaan. Tämän avulla voidaan selvittää, onko kamerapiste varjossa ja valita, käytetäänkö zpass- vai zfail-menetelmää. Seuraavaksi projisoidaan valon vaikutusalue ruudulle, ja asetetaan saksitoiminto leikkaamaan kaikki sen ulkopuolelle jäävät alueet pois. Jos tilavuus projisoituu ruudun ulkopuolelle, se ei ole näkyvissä. Tällöin piirtäminen voidaan jättää kokonaan tekemättä. Tämän jälkeen luodaan uusi näkökenttä, jonka etureuna määritetään saksineliön ääripisteiden avulla (kuva 13b). Tämä näkökenttä kapseloi sisäänsä vain sellaiset alueet, joihin valo vaikuttaa. Kun valokohtaiset rajoitukset on näin saatu tehdyksi, voidaan siirtyä käsittelemään esineitä. Kunkin esineen kohdalla tehdään seuraavat toimenpiteet.

Lasketaan esineelle varjomalli. Jos käytetään zpass-menetelmää, varjon ei tarvitse olla suljettu. Luodaan tilavuus T, joka kapseloi valon vaikutusalueen, saksien mukaan luodun näkökentän ja varjomallin leikkauksen. Tämän jälkeen asetetaan syvyysrajat T:n ääripisteiden z-koordinaattien mukaan, ja piirretään varjomalli. Tämä menetelmä rajaa tehokkaasti ulos kaikki sellaiset pisteet, jotka eivät vaikuta lopulliseen näkymään.

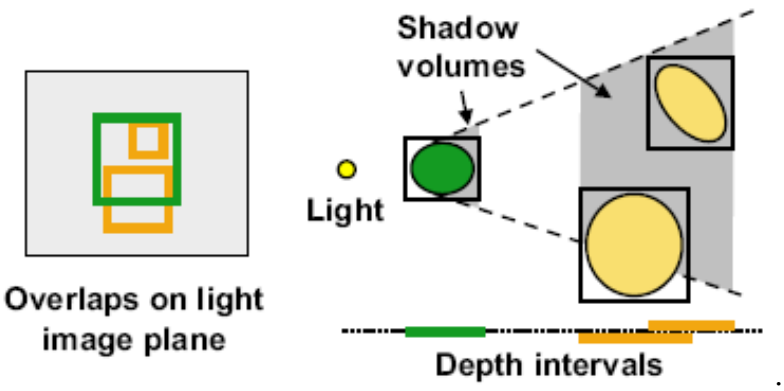
On tapauksia, joissa edellinen menetelmä ei kuitenkaan takaa riittävää tehokkuutta. Esimerkiksi näkymä, jossa on paljon avoimia tiloja ja suunnattuja valoja aiheuttaa sen, että tilavuus T täyttää koko näkökentän, eikä leikkauksista ole hyötyä. Nk. CC-menetelmä (CC tulee sanoista culling ja clamping) rajoittaa varjomallien määrää ja kokoa eri tavalla. Sen tavoitteena on poistaa käsittelystä sellaiset varjomallit, jotka ovat itsekin varjossa, tai joilla ei muuten ole vaikutusta näkökenttään [Lloyd et al., 2004]. Lisäksi pyritään olemaan piirtämättä sellaisia osia varjomalleista, jotka eivät langeta näkyviä varjoja (eli eivät lävistä mitään varjostettavaa esinettä). Menetelmä toteutetaan esineet sulkevien tilavuuksien sekä pikselitason hierarkkisten näkyvyystarkastelujen (occlusion culling) avulla, jotka ovat perusominaisuus nykyisissä näytönohjaimissa. Niissä ruutuun piirretään jotain, minkä jälkeen tarkastelu ilmoittaa sen, kuinka suuri osa piirretyistä pikseleistä läpäisi syvyystarkastelun (eli on oikeasti näkyvissä).



Kuva 14. CC-menetelmän vaiheet [Lloyd et al., 2004]

CC-menetelmä koostuu kolmesta osasta. Ensimmäisessä vaiheessa karsitaan kaikista esineistä (kuva 14a) listaan vain ne varjostavat ja varjostettavat mallit, jotka vaikuttavat kuvaan (kuva 14b). Aluksi piirretään näkymä valon näkökulmasta syvyyskarttaan. Tämän jälkeen suoritetaan hierarkkinen näkyvyystarkastelu kunkin esineen kapseloivalle tilavuudelle. Jos tilavuuden yksikään piste ei ole näkyvissä, esine on kokonaan varjossa ja se hylätään. Muut lisätään mahdollisten varjostavien esineiden listaan (PSC). Sama tehdään kameran näkökulmasta, ja lisätään näkyvät esineet mahdollisten varjostettavien listaan.

Toisessa vaiheessa varjot leikataan siten, että ainoastaan varjostukseen tarvittavat osat piirretään (kuva 14c). Esine S on T:n varjostama jos S ja T ovat valosta katsottuna limittäin tai päällekkäin, ja S:n kaukaisin kärkipiste on kauempana kuin T:n lähin kärkipiste (kuva 15). Tämän jälkeen jokaiselle PSC:n esineelle luodaan lista sen varjostamien esineiden valosta katsottuna lähimmistä ja kaukaisimmista kärkipisteistä. Edellisten avulla voidaan määrittää ne syvyysalueet, joissa varjo lankeaa varjostettavaan esineeseen ja tuottaa näkyviä varjoja. Varjomalli on tarpeen piirtää ainoastaan näille syvyysväleille. Jokaiselle välille luodaan erillinen varjomalli, joka on leikattu alkuperäisestä



Kuva 15. Varjomallien syvyysalueiden määrittäminen [Lloyd et al., 2004]

Varjojen sulkeminen voidaan välttää pilkkomalla näkökenttä osiin tasoilla, jotka kulkevat kamerapisteen läpi ja osoittavat valoa kohti. Pilkottujen varjomallien syvyysääriarvoja siirretään sen verran, että niiden avoimet etu- ja takareunat osuvat näkökentän pilkkoville tasoille (kuva 14d). Tämän jälkeen leikkauskohta on aina katseen suuntainen, eikä kamerapisteestä lähtevä säde voi koskaan kulkea läpi avoimesta reunasta. Lopputuloksena lähes kaikki varjojen kannalta turhat kohdat voidaan jättää piirtämättä.

Muista optimointimenetelmistä on syytä mainita myös, että varsinkin avonaisten varjomallien reunoihin on mahdollista soveltaa geometrisia nauhatyyppejä (triangle tai quad strip), jolloin kärkipisteitä tarvitsee syöttää muunnoskanavaan vähemmän. Menetelmää voidaan tehostaa edelleen piirtämällä reunanelikulmiot kolmioina, joiden yksi kärkipiste on venytetty äärettömyyteen [Everitt and Kilgard, 2002]. Kannattaa myös olla tarkkana, ettei sellaisia esineitä joiden varjot eivät näy (tällaisia ovat usein sisänäkymissä esimerkiksi lattiat ja seinät), oteta varjostavien esineiden joukkoon lainkaan. Lisäksi paikallaan pysyvien valojen ja esineiden tapauksessa voidaan tietenkin käyttää esilaskettuja varjomalleja.

4.3.2. Tuki 3D-rajapinnoissa

Aiemmin on jo puhuttu saksitestistä, syvyysrajoittimesta ja -rajaimesta sekä hierarkkisesta näkyvyystarkastelusta. Nykyiset näytönohjaimet tarjoavat muitakin ominaisuuksia, joista on hyötyä varjotilavuuksien piirtämisessä. Seuraavaksi käydään läpi muutamat keskeisimmät.

Ympärimenevä sapluuna (stencil wrap) [Everitt and Kilgard, 2002] estää sapluuna-arvojen ylivuodot. Sen sijaan että arvot rajautuisivat nollaan ja bittisyvyyden puitteissa suurimpaan arvoon, ne pyörähtävätkin ympäri. Kun suurin arvo ylitetään, sapluunan arvoksi tulee nolla ja vastaavasti toisinpäin. Ympärimenevä sapluuna käytännössä estää laskurin virhearvot, sillä esim. kahdeksanbittisellä sapluunalla virhe syntyy vasta, kun sisäkkäin on vähintään 256 varjoa.

Everitt ja Kilgard [2002] ehdottivat myös kaksipuolista sapluunapuskuria (two sided stencil), joka mahdollistaa eri sapluunaoperaatiot eteen- ja taaksepäin osoittaville pinnoille. Tämä tekee mahdolliseksi myös varjomallien syöttämisen näytönohjaimelle yhdellä kertaa. Piirtämisen määrä ei varsinaisesti vähene, mutta ominaisuus auttaa näytönohjaimia optimoimaan omaa toimintaansa, ja tehostaa näin ollen suoritusta.

Uusimpien näytönohjaimien tukemat geometriavarjostimet mullistanevat varjotilavuuksien piirtämisen täysin. Niillä pystytään luomaan uusia

kärkipisteitä ja monikulmioita sen jälkeen kun piste on kulkenut verteksivarjostimen läpi. Varjomallit voidaan täten koota käytännössä kokonaan näytönohjaimella. Niiden avulla varjotilavuudet muuttuvat tulevaisuudessa todennäköisesti paljon tehokkaammiksi, ja ainakin paljon helpommiksi toteuttaa.

Monet ongelmat jäävät kuitenkin yhä auki. Sitä mukaa kun 3D-näkymät käyvät monimutkaisemmiksi, monimutkaistuvat myös varjomallit. Varjotilavuusmenetelmä ei myöskään kykene reaaliaikaisesti varjostamaan kuin tasaisista kolmioista koostuvia malleja. Esimerkiksi läpikuultavat pinnat ja tekstuuripohjaiset tehosteet (kuten savu ja tuli) ovat käytännössä mahdottomia varjostettavia varjotilavuuksien avulla, vaikka niitä käytetäänkin nykyisin erittäin paljon. Tämä on ajanut kehitystyötä viime vuosina yhä enemmän bittikarttapohjaisten varjostusmenetelmien suuntaan.

5. Varjokartat

Varjokartat [Williams, 1978] ovat menetelmä, jossa varjot projisoidaan kaksiulotteiselle tasolle valon näkökulmasta katsottuna. Niillä on edellisiin menetelmiin verrattuna se etu, että ne eivät ole riippuvaisia siitä, millaisia tai millä tavalla mallinnettuja esineitä näkymä sisältää. Periaatteessa kaikki mitä voidaan piirtää, voidaan myös varjostaa. Varjokartat ovat myös ainoa varjoalgoritmi, jonka tehokkuus ei riipu suoraan varjostettavan näkymän monimutkaisuudesta [Woo et al., 1990]. Esilaskettujen ja paikallaan pysyvien varjojen tapauksessa tämä onkin totta, eikä hidastuminen reaaliajassa laskettujen varjojenkaan kohdalla tapahdu samassa suhteessa kuin esimerkiksi varjomalleilla.

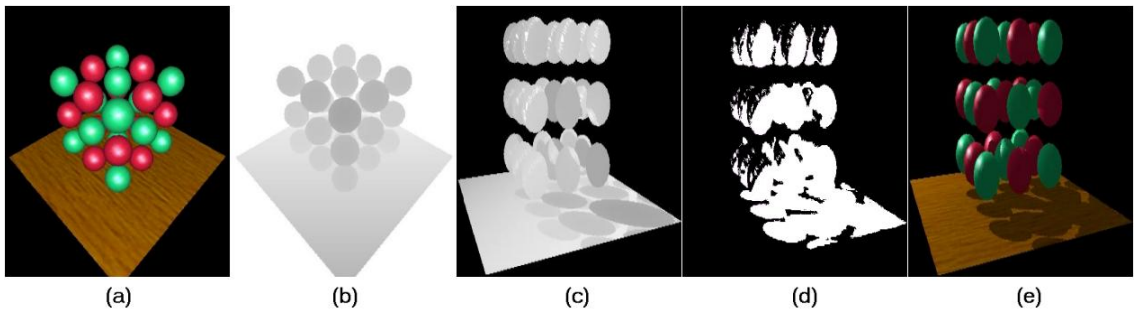
Näkymäriippumattomuudestaan, helppokäyttöisyydestään ja näytönohjainten kiihdytettäväksi sopivasta luonteestaan johtuen, varjokartat ovat olleet kaikista varjostusmenetelmistä viime vuosina kenties ahkerimman tutkimuksen kohteena. Eikä suosio rajoitu ainoastaan reaaliaikaiseen mallintamiseen: mm. Pixar käyttää varjokarttoja 3D-animoitujen elokuviensa varjostamiseen [Upstill, 1990].

5.1. Varjokarttojen toiminta

Varjokartat toimivat kameran tapaan. Varjostavista esineistä luodaan kaksiulotteinen projektio, jonka avulla kameranäkymän varjoalueet voidaan selvittää. Voidaankin sanoa, että kun tasoprojisoidut varjot perustuivat pisteisiin ja varjomallit tilavuuksiin, varjokartoissa keskeinen tekijä on taso, jolle varjo muodostetaan. Tämä tekee niistä erityisen hyvän menetelmän toteutettavaksi näytönohjainten puskurien avulla.

5.1.1. Tasainen varjokartta

Oletetaan aluksi, että käytettävä valo on kohdevalo, jonka sijainti, suunta ja vaikutusalueen avauskulma voidaan esittää samalla tavalla kuin kamerankin. Näin valoa voidaan käyttää kamerana, ja sen näkökentän etureunalle pystytään muodostamaan kuva, joka käsittää esineet valosta katsottuna (kuva 16a). Suunnattujen valojen tapauksessa voidaan käyttää ortografista projisointia, mutta pistevalojen kohdalla tilanne on hieman monimutkaisempi (ks. luku 5.1.2). Pelkkä kuva ei vielä riitä varjojen luomiseen vaan on lisäksi tiedettävä, missä kohtaa valonsäde osuu varjostavaan tasoon. Tämä saadaan selville piirtämällä kuvaan pisteen syvyysarvot (kuva 16b). Tällaista tekstikarttaa kutsutaan joko syvyyskartaksi (depth map) tai varjokartaksi (shadow map).



Kuva 16. Varjokartan toiminta: näkymä valosta (a), syvyyskartta (b), syvyyskartan arvot kameranäkymästä (c), varjostusarvokartta (d) ja lopullinen kuva (e) [Kilgard, 2001b].

Varjokarttamenetelmä jakautuu kahteen osaan: ensimmäisessä luodaan varjokartta, ja toisessa piirretään kameranäkymä, joka varjostetaan luotua varjokarttaa hyödyntäen. Menetelmä siis vaatii näkymän piirtämisen kahteen kertaan kutakin valonlähdettä kohti. Ensimmäisessä vaiheessa piirretään näkymä valon katsekulmasta syvyyskarttaan, joka pidetään tallessa ainakin kameranäkymän piirtämiseen saakka (kuva 16b). Toisessa vaiheessa näkymä piirretään kamerasta katsottuna. Kukin kameranäkymän piste muunnetaan ensin valonlähteen normalisoituun koordinaatistoon. Näin saadaan selville kyseistä pistettä vastaava pikseli varjokartassa (kuva 16c). Jos muunnetun pisteen syvyysarvo (joka siis muunnoksen jälkeen on etäisyys valosta) on suurempi kuin vastaava varjokarttaan tallennettu syvyysarvo tiedetään, että valon ja piirrettävän pisteen välissä on oltava toinen piste. Tällöin muunnettu piste on varjossa. Muussa tapauksessa se on valossa ja voidaan piirtää normaalisti. Jos piste projisoituu varjokartan ulkopuolelle, valaistuksen ratkaisu on yksiselitteinen. Tapauksesta riippuen pisteen voidaan tulkita olevan joko valossa tai varjossa.

Muunnos kamera-avaruudesta valoavaruuteen tapahtuu siirtämällä piirrettävä piste ensin globaaliin avaruuteen, ja sieltä edelleen valon normalisoituun koordinaatistoon (kuva 17). Ensimmäinen muunnos tapahtuu kertomalla piste kameran muunnosmatriisilla. Tämän jälkeen siirto valoavaruuteen tapahtuu valon käänteisellä muunnosmatriisilla, ja projektio edelleen valon projektiomatriisilla. Näiden jälkeen tulee vielä muunnos, jolla piste saadaan normalisoidusta avaruudesta $[-1,1]^3$ sellaiselle välille, että näytönohjain voi löytää vastaavan pikselin. Tämä kaikki voidaan luonnollisesti esittää yhdellä matriisilla.

$$\begin{pmatrix} s \\ t \\ r \\ q \end{pmatrix} = \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot P_{valo} \cdot M_{valo}^{-1} \cdot M_{kamera} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Kuva 17. Muunnosketju kamera-avaruudesta varjokartan pikseliksi. M tarkoittaa muunnosmatriisia ja P projektiomatriisia. Vasemmanpuoleisin matriisi muuttaa koordinaatit normalisoidusta avaruudesta välille [0,1]. Se voi vaihdella käytettävästä 3D-rajapinnasta riippuen.

Valaistuksellisesti oikeaoppinen menetelmä siis vaatii näkymän piirtämistä kaksi kertaa jokaiselle valolle. Lisäksi jokainen kameranäkymään piirrettävä piste on muunnettava varjokarttaan näkyvyystarkastelua varten. Toimenpide saattaa vaikuttaa raskaalta, mutta se ei ole ollenkaan niin hidas kuin voisi ajatella. Näytönohjaimet nimittäin ovat tukeneet perspektiivisiä tekstikarttoja jo ainakin vuosikymmenen ajan osana vakiotoiminnallisuuttaan. Tätä toiminnallisuutta hyödyntämällä varjokartat on mahdollista toteuttaa hyvin tehokkaasti, kuten Segal et al. [1992] osoittavat. Jokaista pikseliä ei tarvitse siirtää kamera-avaruudesta valoavaruuteen erikseen, vaan siirto voidaan tehdä ainoastaan mallien kärkipisteille. Tämän jälkeen näytönohjain laskee varjokarttakoordinaattien arvot kullekin piirrettävän monikulmion pikselille automaattisesti. Perspektiivisesti oikeaoppinen väliarvolaskenta tehdään aina riippumatta siitä, suoritetaanko syvyysvertailu 3D-rajapintojen omien ominaisuuksien avulla vai pikselivarjostimissa. Automaattisesta perspektiivisestä interpoloinnista on jatkossa haittaakin, mutta tässä tapauksessa ne toimivat koko varjokarttamenetelmän reaaliaikaisen toteutuksen pohjana.

Valoavaruudessa tapahtuvan syvyystarkastelun tuloksena saadaan piirrettävälle pikselille varjostusarvo. Williamsin [1978] menetelmässä tämä on yksinkertainen totuusarvo, eli piste on joko varjossa tai ei. Esimerkiksi

läpikuultavien esineiden kohdalla se voi kuitenkin olla jotain muutakin. Varjostusarvoista voidaan myös piirtää kuva, joka erottaa kameranäkymässä valaistut pisteet varjostetuista (kuva 16d). Sitä voidaan myöhemmin esimerkiksi suodattaa pehmeämpien varjoreunojen saavuttamiseksi. Varsinkin suodatusten osalta on syytä ymmärtää ero syvyyskartan ja varjostusarvokartan (attenuation map) välillä.

Varjokartta on piirrettävä uudelleen aina kun valon tai jonkun sen vaikutusalueella olevan esineen sijainti tai suunta muuttuu. Kameran liikkuminen ei sen sijaan vaikuta tasaiseen varjokarttaan. Mikäli muuttuvia esineitä ei ole, varjokartat voidaan piirtää kerran ja tallentaa tekstikartoiksi näytönohjaimen muistiin. Tämän jälkeen niiden käyttäminen on erittäin tehokasta. Kannattaa myös huomata, että mikäli näkymä sisältää sekä paikallaan pysyviä että liikkuvia esineitä, voidaan staattiset esineet piirtää yhteen muuttumattomaan varjokarttaan. Kun näkymään lisätään liikkuvia esineitä, niistä voidaan luoda oma varjokarttansa jättämällä staattiset esineet piirtämättä. Tämän jälkeen nämä voidaan yhdistää lopullisen, piirtämisessä käytettävän varjokartan luomiseksi.

5.1.2. Pistevalot

Varjotilavuudet käsittelivät pistevaloja automaattisesti, mutta koska varjokartta on projisoitava tasolle, se voi suoraan tukea ainoastaan kohdevaloja tai suunnattuja valoja. Joka suuntaan paistavat pistevalot, ovat ongelmallisempia. Käytetyin keino on piirtää kuusi varjokarttaa, joista kukin kattaa valosta katsottuna yhden suunnan [Dietrich, 2001]. Kun kaikki kartat määritetään neliöiksi ja kunkin suunnan valon aukeamiskulmaksi annetaan 90 astetta, saadaan aikaan nk. kuutiokartta (cube map), jonka tasot kapseloivat valon sisäänsä. Kuution akselit kulkevat samaan suuntaan kuin globaalin koordinaatiston akselit.

Kuutiokarttaa käsitellään kuten normaaliakin varjokarttaa. Ainoana erona on, että pistettä varjostettaessa täytyy tietää mihin kuudesta varjokartasta piste muunnetaan. Tämä voidaan tehdä aktivoimalla yksi sivu kerrallaan, ja määrittelemällä kartan ulkopuolelle projisoidut pisteet olemaan varjossa. Tulokset voidaan yhdistää esimerkiksi sapluunapuskuria käyttämällä. Toinen, huomattavasti vähemmän erityisvaiheita vaativa tapa, on laskea suuntavektori pisteestä valoon, ja päättää käytettävä varjokartta sen perusteella.

Suuntavektori voidaan laskea verteksivarjostimessa kullekin mallin kärkipisteelle, ja palauttaa se tämän jälkeen näytönohjaimelle. Pikselivarjostin saa vektorin valmiiksi interpoloituna. Koska kuution akselit ovat samansuuntaiset kuin globaalin koordinaatistonkin, voi varjostin vektorin normalisoituaan määrittää sen, mitä kuution sivua pitää käyttää.

Koska kuuden eri varjokartan piirtäminen on hyvin raskasta, on tärkeää, ettei mitään tarpeetonta piirretä. Esimerkiksi kaikki sellaiset kuution sivut jotka eivät vaikuta kameranäkymään, kannattaa jättää piirtämättä. On syytä huomata, että mikäli pistevalo sijaitsee näkökentän ulkopuolella, ainakin yksi kuution tasoista voidaan aina jättää piirtämättä [King, 2004]. Näytönohjaimet tukevat kuutiotekstuureita nykyisin hyvin kattavasti mikä tarkoittaa, että sen sivujen hakeminen muistista on tehokkaampaa, kuin jos varjokartat tallennettaisiin erikseen. Kuitenkin pahimmillaan jopa kuusi piirtokertaa vaativa algoritmi on auttamatta liian hidas jatkuvaan käyttöön.

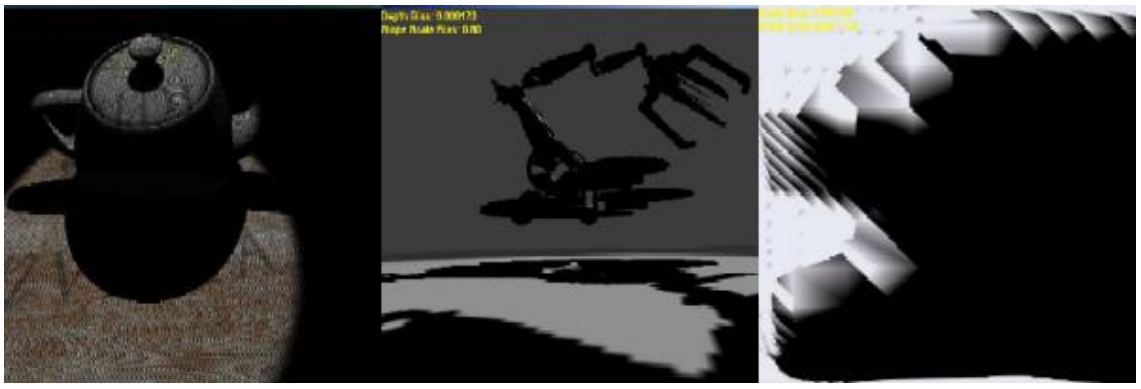
Projektiot eivät onneksi rajoitu perspektiivisiin projisointeihin, vaan on matemaattisesti täysin mahdollista muuntaa jopa kokonainen pallo yhdelle tasolle. Tällaiset suuren avauskulman näkökentät ovat matemaattisesti helppoja toteuttaa, mutta niiden ongelmana on se, etteivät ne ole luonteeltaan hyperbolisia. Toisin sanoen, niiden tuottamassa avaruudessa suorat reunat eivät säily suorina. Kärkipisteet voidaan muuntaa oikeaoppisesti, mutta koska näytönohjaimet käyttävät vain perspektiivisiä interpolointeja monikulmioiden piirtämiseen, eivät mallien reunat piirry oikein. Tämä puolestaan aiheuttaa epätarkkuutta pikseleiden syvyysarvojen kanssa.

Brabec et al. [2002b] tutkivat eri projisointien soveltuvuutta 3D-grafiikkaan ja päätyivät siihen, että 180 asteen paraboloidiprojektio [Heidrich and Seidel, 1998] sopii käyttötarkoituksiin parhaiten tarkkuutensa, tehokkuutensa ja helppokäyttöisyytensä ansiosta. Projisoimalla ruudulle kerralla puolipallo, ei täydellisen pistevaloon tarvita kuin kaksi varjokarttaa. Projisointi tapahtuu varjostimilla hieman samaan tapaan kuin kuutiokarttojen kohdalla. Varjokartan koordinaatit määritellään varjostimessa, ja tämän jälkeen syvyysarvo lasketaan etäisyytenä piirrettävästä pisteestä valon sijaintipisteeseen. Piirrettäessä on osattava valita oikea pallonpuolisko. Pyöristysongelmista johtuen tieto siitä kumpaa pallonpuoliskoa käytetään, kannattaa tallentaa alfakanavaan. Tällä saadaan ratkaistua myös tilanne, jossa monikulmion kärkipisteet sijaitsevat eri pallonpuoliskoilla [Brabec et al., 2002b].

Vaikka edellä esitellyssä projektiossa syvyyspuskurin epätarkkuudet saadaan kutakuinkin kuriin, ei niistä silti päästä kokonaan eroon. Käytännössä tällaisten projektioiden käyttäminen vaatiikin erittäin yksityiskohtaisia malleja. Monista näytönohjaimista tosin löytyy nykyisin ominaisuuksia, joilla mallin monikulmioita voidaan automaattisesti pilkkoa pienempiin osiin (tessellation). Tämä voidaan tehdä uusimmissa laitteissa myös geometriavarjostimia käyttäen.

5.2. Tasaisen varjokartan ongelmat

Pistevalojen toteutus ei ole varjokarttojen ainoa ongelma. Vaikka menetelmä onnistuu korjaamaan edellisten algoritmien suurimmat puutteet, sillä on myös heikkoutensa. Suurimmat ongelmat liittyvät sen bittikarttapohjaisuuteen. Siinä missä vektoripohjaiset tasoprojisointi- ja varjotilavuusmenetelmä tuottivat tarkkoja ja teräviä reunoja, ei resoluutiosta riippuvainen bittikarttagrafiikka kykene samanlaiseen skaalautuvuuteen. Mikäli varjokartan resoluutio on liian pieni, varjot voivat muuttua epätarkoiksi. Tällaista ilmiötä kutsutaan laskostumiseksi (aliasing). Toinen, hieman helpommin ratkaistava ongelma on jo tasoprojisoiduista varjoista tuttu syvyysarvojen epätarkkuus. Näiden ongelmien ratkaisemiseksi on ehdotettu lukemattomia eri tapoja, ja seuraavissa luvuissa käydään läpi keskeisimmät.



Kuva 18. Väärä varjostuminen (vasemmalla) sekä perspektiivinen (keskellä) ja projektiivinen (oikealla) laskostuminen ovat varjokarttamenetelmän perusongelmat [King, 2004].

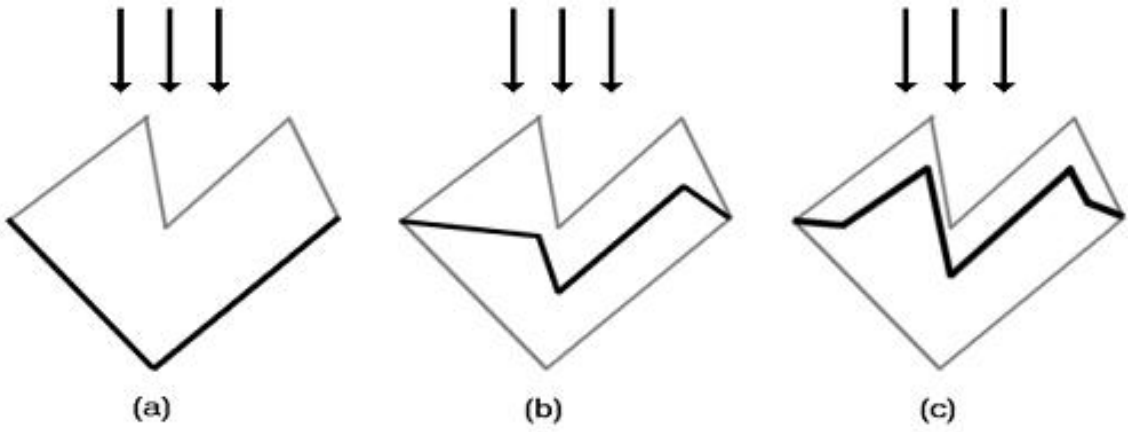
5.2.1. Syvyysarvojen epätarkkuus

Mikäli syvyysvertailun molempien osapuolien etäisyys valosta on sama, muunnettu piste sijaitsee täsmälleen varjostavalla tasolla, ja on näin ollen valossa. Kuten tasoprojisoitujen varjojen tapauksessa kuitenkin jo aikaisemmin todettiin, tämä ei ole välttämättä tilanne tietokonegrafiikan tapauksessa. Syvyyskarttojen bittileveyden aiheuttamat pyöristysepätarkkuudet johtavat helposti siihen, että osa pinnalla olevista pisteistä painuu varjostettavan pinnan taakse, ja tulee väärin varjostetuksi. Tasoprojisoitujen varjojen tapauksessa ongelma saatiin ratkaistua vähentämällä varjopikseleiden syvyysarvoa piirtovaiheessa. Koska syvyysmuunnos tehtiin projisoinnin jälkeen, hieman suuremmallakaan muutoksella ei ollut kielteistä vaikutusta näkymään. Varjokarttojen kohdalla tilanne on kuitenkin toinen. Syvyysarvon muuttaminen varjokartassa tarkoittaa sitä, että muunnos näkyy konkreettisesti myös varjon sijainnissa. Siirtämällä varjoa liikaa, aiheutetaan kiinnitetyn varjon

irtoaminen varjostavasta esineestä. Seurauksena on nk. Peter-Pan-ilmiö, eli varjo alkaa leijua. Oikean muunnosarvon löytämiseen ei ole yhtä oikeaa ratkaisua, vaan se riippuu aina näkymän ominaisuuksista.

Mark Kilgard [2001b] luettelee joitakin neuvoja oikean syvyysmuunnoksen määrittämiseen. Yleensä varjon syvyysarvoa kannattaa lisätä mieluummin liikaa kuin liian vähän. Tämä johtuu siitä, että leijuva varjo ei ole yleensä yhtä helposti havaittava virhe kuin väärä varjostuminen. Lisäksi kannattaa huomata, että yleensä suurellakaan syvyysarvojen työntämisellä ei ole vaikutusta lankeavan varjon piirtymiseen. Toinen huomion arvoinen seikka on, että jyrkästi valoon päin kallellaan olevat tasot vaativat muunnosta enemmän kuin muut tasot. Tämä on otettu huomioon myös 3D-rajapintojen syvyysmuunnosfunktioissa. Esimerkiksi OpenGL:n glPolygonOffset-funktiolla on mahdollista määrittää myös kulman jyrkkyyden vaikutus. Muunnoksen määrää voi myös vähentää sitä mukaa kun syvyyspuskurin bittileveys kasvaa. Tämän lisäksi Kilgard muistuttaa, ettei varjokartan pintojen sijaintia saa muuttaa kolmiulotteisissa muunnoksissa. Tällöin syvyysarvot eivät muutu perspektiivijaosta johtuen tasaisesti, eikä pinnan kulmaa valoon nähden voida enää ottaa huomioon.

Syvyysmuunnoksen lisäksi voidaan käyttää muitakin menetelmiä. Wang ja Molnar [1994] eivät kirjoita syvyyskarttaan valoon päin osoittavia pintoja, vaan valosta poispäin osoittavat (kuva 19a). Tämä menetelmä takaa sen, että joitakin aivan ohuita esineitä lukuun ottamatta esineiden valoisa pinta näkyy oikein. Syvyystesti muuttuu siten, että muunnettu piste on valaistu, mikäli sen etäisyys valosta on pienempi kuin varjokartassa olevan pisteen. Tämä menetelmä luonnollisesti vaatii, että kaikki näkymän esineet ovat suljettuja.



Kuva 19. Syvyysarvojen määrittyminen Wangin ja Molnarin [1994] (a), Woon [1992] (b) sekä Weiskopfin ja Ertlin [2003] (c) menetelmissä. Nuolet osoittavat

valon suunnan, harmaa on esine, ja musta väri osoittaa varjokartan syvyystasot.

Andrew Woo [1992] kirjoittaa omassa menetelmässään varjokarttaan kahden valoa lähimmän pinnan syvyyskeskiarvon (kuva 19b). Menetelmä toimii myös avoimilla malleilla, mutta silloin käytetään vain lähimpänä valoa olevaa pintaa, ja algoritmin hyöty häviää. Menetelmä on nykyisin toteutettavissa nk. syvyyskuorintamenetelmällä (depth peeling), jota käytetään yleisemmin läpikuultavien tasojen toteuttamiseen [Everitt et al., 2001]. Woon menetelmä vaatii enemmän piirrettävää kuin tavallinen varjokartta, koska suljettujen mallienkaan tapauksessa piilopintojen poistoa ei voida käyttää.

Weiskopf ja Ertl [2003] ovat havainneet tiettyjä tapauksia, joissa Woon algoritmi tuottaa vääriä tuloksia. Erityisen hankalia alueita ovat mallien reunat, joissa eteen- ja taaksepäin osoittavat pinnat ovat lähellä toisiaan sekä jyrkästi vaihtelevat pinnan suunnat, jotka aiheuttavat sen, että osa esineestä saattaa jäädä virheellisesti varjostumatta. Heidän menetelmänsä parantaa Woon algoritmia lisäämällä siihen myös mahdollisuuden normaaliin syvyysmuunnokseen (kuva 19c). Muunnos lisätään lähimpänä valoa olevan pinnan syvyysarvoon ennen muunnetun pisteen vertailua. Kehittäjät suosittelevat muunnoksen laskemista pistekohtaisesti pikselivarjostimessa. Tällöin syvyysarvoon lisätään joko käyttäjän itse määrittelemä vakiomuunnos tai kahden valoa lähimmän pinnan etäisyys jaettuna kahdella. Pienempi näistä arvoista valitaan. Vakiomuunnoksen ansiosta menetelmä ei vaadi valosta poispäin osoittavien pintojen piirtämistä, vaan varjo pysyy aina varjostavan pinnan takana. Kun kaksi pintaa tulee hyvin lähelle toisiaan, varjolaskenta tapahtuu niiden keskiarvoja käyttämällä, jolloin varjo pysyy niiden välissä, ja molemmat pinnat varjostuvat oikein. Koska syvyystaso normaalisti määrittyy kahden valoa lähimmän pinnan keskiarvona, voidaan syvyysmuunnosta käyttää tavallista huolettomammin ilman välitöntä pelkoa varjon leijumisesta. Tämänkin menetelmän heikkoutena on tosin se, että varjokartta on piirrettävä kaksi kertaa.

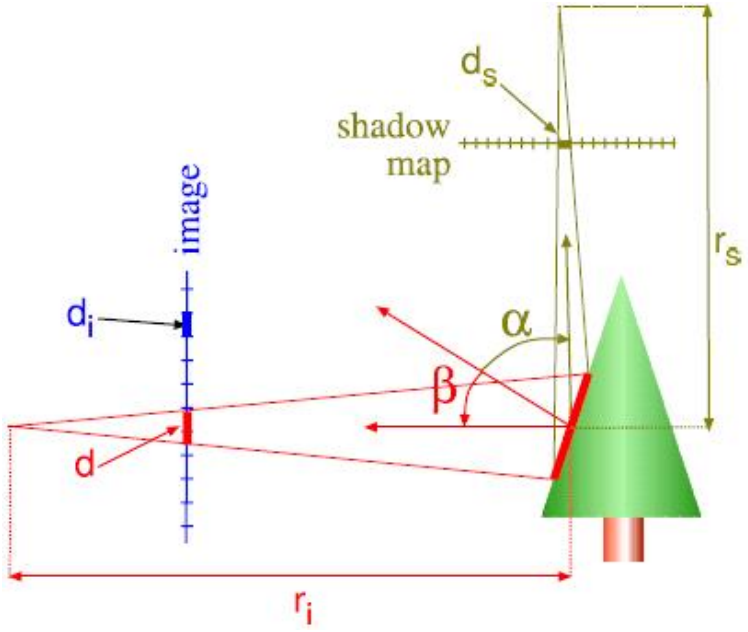
Brabec et al. [2002a] huomasivat, ettei varjokartan syvyysarvojen tarvitse olla perspektiivisiä. Kameranäkymässä se, että syvyysarvot ovat etualalla tarkempia, on hyvin perusteltavissa sillä, että tällä alueella yksityiskohdat myös näkyvät tarkemmin. Valojen kohdalla tilanne ei kuitenkaan ole sama. On hyvin tavallista, että valosta katsottuna etäisimmät esineet ovat lähimpänä kameraa. Tällaisissa tapauksissa käy niin, että esineiden syvyysarvot varjokartassa eroavat toisistaan vain vähän, aiheuttaen suuria ongelmia syvyysarvojen vertailussa. Ratkaisuksi tarjotaan verteksivarjostimessa tehtävää operaatiota, jossa z-koordinaatin arvoa ei lähetetäkään projektion läpi

sellaisenaan, vaan sen arvo kerrotaan vektorin w-komponentilla. Tämä kompensoi leikkausavaruuden normalisoinnissa tapahtuvan w-koordinaatilla jaon, ja tekee syvyysarvoista lineaarisia. Nyt syvyysarvot pysyvät yhtä tarkkoina kaikilla varjokartan tasoilla. Menetelmän heikkoutena on, että virheellisen z-arvon interpolointi voi tuottaa ongelmia varsinkin suurten monikulmioiden kanssa [Wimmer et al., 2004].

5.2.2. Laskostuminen

Laskostuminen on kaikille bittikartoille ominainen ongelma. Koska tekstuureiden resoluutio on sidottu johonkin tiettyyn tarkkuuteen, kuvan suurentaminen johtaa siihen, että yksittäiset pisteet tulevat näkyviin aiheuttaen nk. palikoitumisilmiön. Laskostumisen määrä riippuu siis käytettävän bittikartan koosta ja siitä, kuinka suurena se ruudulla näkyy. Ongelmaa helpottaa tekstuurien resoluution kasvattaminen, mutta käytännön syyt prosessoritehon ja muistitilan suhteen sanelevat niiden koolle ylärajat. Laskostumisesta onkin lähes mahdotonta päästä kokonaan eroon, mutta erilaisia korjausmenetelmiä on kehitetty lukuisia. Huomio keskittyy nimenomaan varjojen reunoihin, sillä ne ovat ainoa paikka joissa laskostumista ilmenee.

Stamminger ja Drettakis [2002] jakavat varjokarttoihin liittyvät laskostumisongelmat kahteen ryhmään: projektiiviseen ja perspektiiviseen. Projektiivista laskostumista (projective aliasing) esiintyy tilanteissa, joissa varjon reuna osuu varjostettavaan pintaan hyvin loivassa kulmassa, ja varjokartan pikselit venyvät sitä vasten. Perspektiivinen laskostuminen (perspective aliasing) taas johtuu siitä, että pieni osa varjokarttaa heijastetaan hyvin lähelle kameraa, jolloin näytöllä näkyvän kuvan tarkkuus on paljon suurempi kuin varjon. Tätä esiintyy lähinnä sellaisten esineiden kohdalla, jotka ovat kaukana valosta, mutta lähellä kameraa. Tällöin, perspektiivisestä vääristyksestä johtuen, esine näkyy valosta katsottuna vain pienenä, ja sen varjoon käytetään vain vähän varjokartan pikseleitä. Kameran näkökulmasta katsottuna varjo sen sijaan täyttää suuren osan näkökenttää, ja tällöin varjon yksittäiset pikselit tulevat selvästi näkyviin. Laskostumisesta on annettu myös formaali esitys (kuva 20) [Stamminger and Drettakis, 2002], jonka ymmärtäminen auttaa sisäistämään myöhemmin käsiteltäviä korjausmenetelmiä.



$$d = d_S \frac{r_S}{r_i} \frac{\cos\beta}{\cos\alpha}$$

Kuva 20. Laskostumisen formaali esitys [Stamminger and Drettakis, 2002].

Kuvassa d_s on varjokartan pikseli, joka langettaa varjon varjostettavaan pintaan etäisyydellä r_s ja kulmassa α. Pisteen koko on tässä kohdassa $d_s r_s/\cos$ α. Tämän pohjalta voidaan laskea, minkä kokoisena piste näkyy kameran näkökentän etureunassa. Mikäli tämä koko, d, on suurempi kuin väripuskurin pikselin koko, varjokartan pikseli kattaa useamman väripuskurin pisteen, ja tästä seuraa laskostumista. Kun varjokartan resoluutiota kasvatetaan, d_s:n ja sitä kautta myös d:n koko pienenee, ja laskostuminen vähenee. Jatkon kannalta on myös hyvä huomata, että jos valo on suunnattu, eli sillä ei ole perspektiiviä, sen säde on koko matkaltaan ds:n paksuinen. Tällöin r_s:n arvolla ole merkitystä.

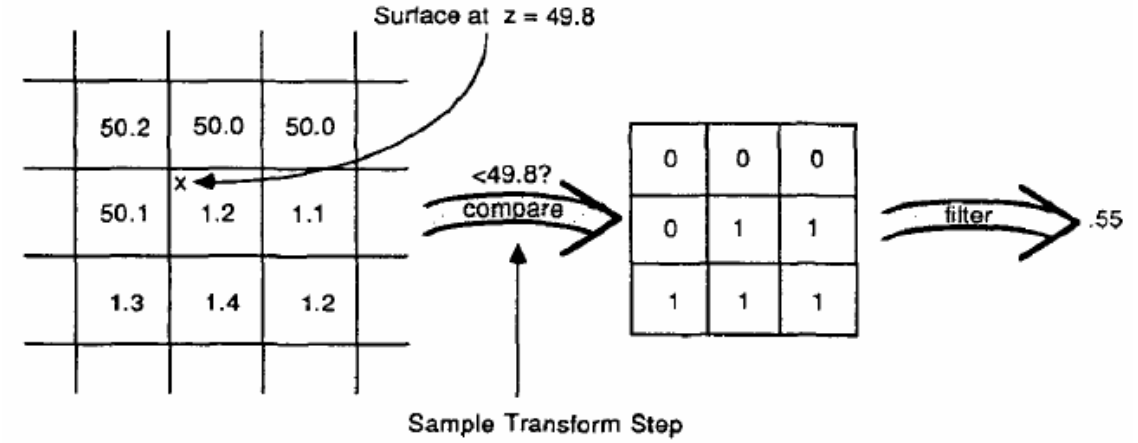
Jo aiemmin todettiin projektiivista laskostumista syntyvän, kun varjon reuna osuu varjostettavaan pintaan hyvin loivassa kulmassa. Kuvan perusteella voidaankin todeta, että mitä suuremman arvon cos β/cos α saa, sitä pahempaa projektiivinen laskostuminen on. Perspektiivinen laskostuminen puolestaan on sitä suurempi, mitä suuremman arvon $d_s r_s/r_i$ saa. Ihannetilanne olisi sellainen, missä d_s ja d olisivat kaikissa pisteissä samat. Tähän aiheeseen palataan luvussa 5.4.3.

Kummallekin laskostumistyypille on omat, tosin hyvin pitkälle toisiaan tukevat, korjausmenetelmänsä. Lisäksi, muiden bittikarttojen tavoin, myös varjokarttojen kohdalla voidaan käyttää suodatusta näkyvien reunojen pehmentämiseen. Tämä ei kuitenkaan ole aivan niin yksinkertaista kuin tavallisten tekstuurien kanssa.

5.3. Varjokartan suodattaminen

Tavallisten bittikarttakuvien pisteiden arvot suodatetaan tietyn alueen näytteiden perusteella. Näistä tekstikartoista rakennetaan usein nk. mipmap, joka tallentaa kuvasta versioita eri tarkkuuksilla. Syvyyskartalle vastaavaa suodatusta ei kuitenkaan voi tehdä, koska se sotkisi syvyysarvot aiheuttaen varjotarkastelujen epäonnistumisia. Varjostusarvokartalla tätä rajoitusta ei ole, joten sitä voidaan suodattaa vapaasti. Mm. Daniel Scherzer [2005] ehdottaa ensin generoimaan koko varjostusarvokartan, ja tämän jälkeen pehmentämään sen varjoalueiden reunat väripuskurissa. Tämä on kuitenkin kankea ja hidas tapa. Lisäksi se vaatii erilaista pehmennystä eri etäisyyksillä oleville alueille, koska tasainen suodatus sotkee kaukana olevien varjojen yksityiskohdat.

Prosenttisuodatin (percentage closer filter) [Reeves et al., 1987] suodattaa varjostusarvokartan vertaamalla muunnetun pisteen syvyyttä myös vastaavan varjokarttapikselin naapureihin. Varjostusarvo muodostetaan sen mukaan, kuinka suuri osa pikseleistä on pisteen edessä (kuva 21). Kuvan esimerkissä muunnettu piste on etäisyydellä 49.8. Normaalisti valaistusarvokarttaan tallennettaisiin siis arvo 1, koska syvyystarkastelu osoittaa muunnetun pisteen olevan kokonaan varjossa. Prosenttisuodatin kuitenkin vertaa syvyysarvoon myös ympäröiviä pikseleitä, ja muodostaa lopullisen varjostusasteen valaistujen naapuripisteiden osuutena välillä 0-1. Prosenttisuodatin ei siis varsinaisesti tee mitään laskostumisen syille, vaan ainoastaan pehmentää valaistusarvokarttaa sen verran, että palikat varjojen reunoilla pehmenevät.



Kuva 21. Prosenttisuodattimen toiminta K:n arvolla 1. Vasemmalta lähtien varjokartta, jonka pohjalta syvyystarkastelu tehdään, naapuripikseleiden varjostusarvot muunnetun pisteen suhteen, ja viimeisenä siitä suodatettu lopullinen varjostusarvo [Reeves et al., 1987].

Suodattimelle voidaan antaa arvo K sen mukaan, kuinka suurelta alueelta se näytteitä hakee. Suuret K:n arvot johtavat pehmeämpiin reunoihin, mutta

luonnollisesti myös hidastavat algoritmin suorittamista. Menetelmässä voidaan soveltaa myös satunnaisotantaa, jossa kaikkia K:n alueella olevia pisteitä ei hyödynnetä, vaan poimitaan niistä enemmän tai vähemmän satunnaisesti vain tietty osa. Varsinkin suurilla K:n arvoilla prosenttisuodatinta on käytetty myös luomaan illuusiota puolivarjoista, mutta vaikka reunat pehmentyvätkin, ei tällä menetelmällä ole tietenkään mitään tekemistä todellisten puolivarjojen kanssa. Nykyisin prosenttisuodattimen käyttö on automaattinen toimenpide käytännössä kaikissa sisäisesti varjokarttoja tukevissa näytönohjaimissa.

Donnelly ja Lauritzen [2006] ymmärsivät menetelmän perusluonteen siten, että se itse asiassa käyttää K:n alueella olevien syvyysarvojen jakaumaa. Tämä innoitti heitä kehittämään uuden, todennäköisyyslaskentaan perustuvan tekniikkansa, varianssivarjokartan (variance shadow map). Perusajatus on olla välittämättä yksittäisistä varjokartan pisteistä, ja perustaa syvyystarkastelu sen sijaan syvyysjakaumille. Aluksi luodaan kaksi syvyyskarttaa. Ensimmäisen pisteisiin tallennetaan ympäröivien pikseleiden syvyyskeskiarvot, ja jälkimmäiseen keskiarvojen neliöt. Molemmat kartat voidaan laskemisen jälkeen suodattaa vapaasti. Valaistustarkastelua tehtäessä menetelmä käyttää todennäköisyyslaskennan sääntöjä (mm. Tšebyševin epäyhtälöä) varjostusasteen määrittämiseen. Tarkka matemaattinen kuvaus löytyy mm. Donnellyn ja Lauritzenin [2006] tekstistä. Kevin Meyers [2007] antaa menetelmästä konkreettisen kuvauksen koodiesimerkkeineen.

Varianssivarjokarttojen suurin vahvuus on se, että niitä voidaan suodattaa käytännössä vapaasti. Lisäksi niistä voidaan rakentaa mipmappeja tehokkaamman syvyysvertailun saavuttamiseksi. Kumpikaan ei ole mahdollista prosenttisuodattimen yhteydessä. Kuten monet muutkin matemaattisesti monimutkaiset menetelmät, varianssivarjokartat kärsivät kuitenkin numeerisista epätarkkuuksista, jotka aiheuttavat mm. valojen vuotamista. Varsinkin 16 bittisillä syvyyspuskureilla tämä on ollut jossain määrin merkittävä ongelma, mutta kehittäjät uskovat, että jatkossa operaatiot voidaan tehdä 32 bittisellä puskurilla, jolloin ongelma saadaan kuriin. Varianssivarjokartat saattavatkin jossain vaiheessa syrjäyttää prosenttisuodattimen, joten niiden olemassaolosta on hyvä olla tietoinen.

5.4. Laskostumisen korjaaminen

Edelliset menetelmät parantavat varjojen laatua pehmentämällä niiden reunoja. Tämä ei kuitenkaan ole tarpeeksi, mikäli varjokartan pikselit piirtyvät ruudulle huomattavan suurina. Varjokartta ja kamera onkin pystyttävä kohdentamaan siten, että laskostumiseen johtavat tilanteet kävisivät mahdollisimman harvinaisiksi.

Tärkein asia varjon kohdistamisessa on se, että pyritään minimoimaan varjokarttaan jäävän hukkatilan määrä [Brabec et al., 2002a]. Hukkatilalla tarkoitetaan sellaisia alueita, jotka eivät vaikuta kameranäkymään. Näitä ovat esimerkiksi kameran näkökentän ulkopuolelle jäävät alueet. Scherzer [2005] muistuttaa lisäksi, että näkökentän perusteella laskettava varjon projektioalue ei riitä, sillä nykyisissä sovelluksissa näkökenttä on usein erittäin suuri tai jopa ääretön. Näin on päätettävä tarkemmin, mitkä osat näkökentän sisältä otetaan mukaan käsittelyyn.

Brabec et al. [2002a] projisoivat näkymän keskeiset pisteet valon näkymään, ja laskevat lopullisen varjokarttaan piirrettävän alueen analyyttisesti. Tämä takaa hyvin tiiviin näkökentän ja vain vähän hukkatilaa. Toinen heidän esittelemänsä tapa muistuttaa hieman Lengyelin [2002] menetelmää, joka esiteltiin luvussa 4.3.1. Siinä ruudulle projisoidaan ainoastaan näkymän keskeiset esineet kapseloivat tilavuudet, ja lasketaan varjokartan ääripisteet niiden mukaan. Kapselointi ei tällä jälkimmäisellä tavalla ole yhtä tiukka, mutta varsinkin näkymät, jotka sisältävät paljon monimutkaisia esineitä, hyötyvät tämän pienen epätarkkuuden tuomasta lisätehosta. Samassa tekstissä muistutetaan myös syvyyssuunnan kapseloinnista, ja sen huomattavasta hyödystä syvyysarvojen tarkkuuden kanssa. Nämä menetelmät ovat hyvä alku laskostumisen ehkäisemiseen, mutta ne eivät varsinkaan suurissa näkymissä vielä riitä tarkentamaan varjoja riittävästi.

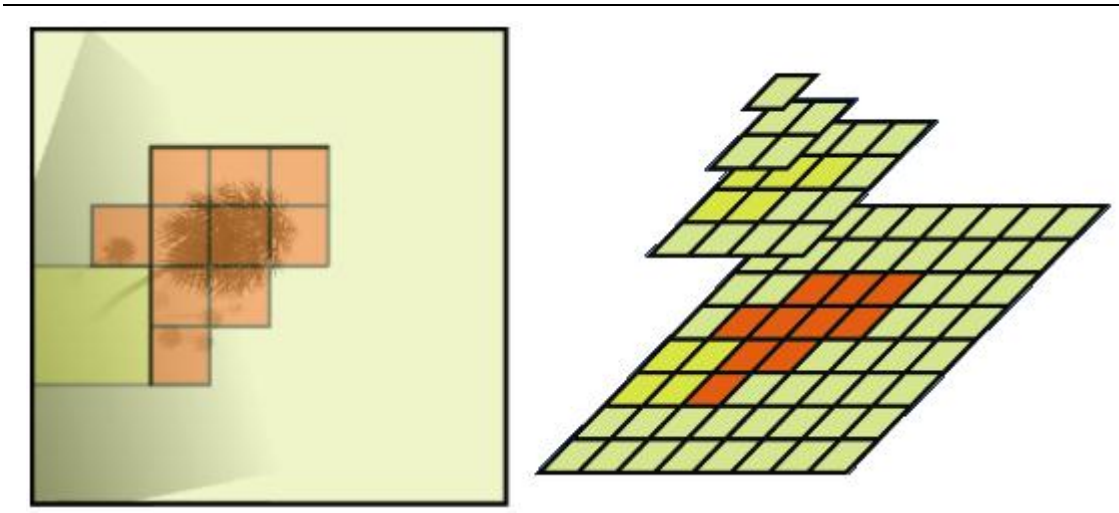
Muut korjausmenetelmät voidaan jakaa kahteen ryhmään: niihin joissa suuri varjokartta pilkotaan pienempiin osiin ja niihin, joissa karttaa vääristetään. Molempien tarkoituksena on taata parempi resoluutio varjokartan tärkeille alueille. Vain ensin mainitut auttavat projektiiviseen laskostumiseen, jälkimmäisten keskittyessä korjaamaan perspektiivisyydestä johtuvat varjokartan epätarkkuudet. Projektiivinen laskostuminen ei ole ratkaistavissa yleispätevästi, vaan vaatii aina näkymän analysointia [Wimmer et al., 2004]. Ainoaksi tavaksi jääkin pyrkiä löytämään varjojen reuna-alueet, ja ratkaista ongelma niitä tarkentamalla. Perspektiivisessä laskostumisessa tilanne ei ole aivan näin paha, vaan ongelmaan on kehitetty yleispäteviäkin ratkaisumalleja.

5.4.1. Mukautuva varjokartta

Mukautuva varjokartta (adaptive shadow map) [Fernando et al., 2001] edustaa tämän tutkielman kentässä poikkeustapausta sikäli, ettei sitä ole vielä kovin onnistuneesti pystytty käyttämään reaaliaikaisissa sovelluksissa. Tästä huolimatta sen asema on tärkeä, koska se, ja siitä edelleen kehitetyt menetelmät ovat ainoita, jotka pystyvät tehokkaasti ehkäisemään sekä projektiivista että perspektiivistä laskostumista [Lefohn et al., 2007].

Projektiivista laskostumista voidaan korjata ainoastaan nostamalla varjokartan resoluutiota niissä kohdissa, missä palikoituvia reunoja ilmenee. Laskostumisen havaitsemiseksi täytyy pystyä jotenkin selvittämään, minkä kokoisena kukin varjon piste näkyy kamerasta katsottuna. Tämä olisi periaatteessa tehtävissä Stammingerin ja Drettakisin [2002] kaavalla (kuva 20), mutta sen soveltaminen jokaiselle pikselille olisi liian hidasta reaaliaikaiseen käyttöön. Mukautuva varjokartta tarjoaa menetelmään ratkaisun, joka hyödyntää ovelalla tavalla näytönohjainten toiminnallisuutta.

Mukautuva varjokartta tallentaa laskostumisen osalta kriittiset kohdat suuremmalla resoluutiolla kuin sellaiset kohdat, joissa laskostumisen vaaraa ei ole. Tämä toteutetaan tallentamalla varjokartta yhden suuren kuva sijasta puumaiseen rakenteeseen (kuva 22). Kartan hierarkian juurisolmuna on tavallinen, melko pienellä resoluutiolla kuvattu varjokartta. Solmu on jaettu neljään soluun. Mikäli solussa ei ole laskostumisen kannalta kriittisiä alueita, varjostukseen voidaan käyttää juuren tekstikarttaa. Jos taas solun alueella vaaditaan tarkempaa varjoa, soluun kiinnitetään uusi solmu, joka on rakenteeltaan juuren kaltainen. Uusi solmu sisältää tekstikartan, johon piirretään juuren solua vastaavan neljänneksen alue paremmalla resoluutiolla. Mukautuvan varjokartan rakenne on puumainen hierarkia, eli kaikki solut voivat sisältää edelleen uusia lapsia.



Kuva 22. Mukautuva varjokartta. Vasemmalla tarkennetut alueet, ja oikealla niistä rakennettu puurakenne [Lefohn et al., 2005].

Mukautuva varjokartta voidaan sitoa tiettyihin tehokkuus- ja muistivaatimuksiin, jolloin hierarkiaa kasataan vain niin kauan kunnes asetetut rajat ovat tulleet vastaan. Näin puumallin rakentamiseen kuluva aika voidaan rajoittaa tarvittaessa jopa reaaliaikaisuuden vaatimalle tasolle. Mitä tiukemmiksi rajat asetetaan, sitä heikompi on varjojen laatu. Tehokkuuden

saavuttamisessa auttaa myös menetelmän progressiivinen luonne. Hierarkiaa ei tarvitse rakentaa joka kerta alusta saakka (ellei näkymä muutu täydellisesti), vaan menetelmä pitää kirjaa rakenteen osien käyttötiheydestä, ja poistaa niitä mahdollisesti tarvittavien uusien solmujen tieltä. Tämän ansiosta menetelmän on helppo mukautua erilaisiin muistivaatimuksiin: poistamalla tarpeettomiksi käyneitä solmuja voidaan vapauttaa tilaa uusille, jolloin aiemmin lasketut ja edelleen käytössä olevan solmut voidaan pitää muistissa. Kehittäjät tarjoavat oman yksinkertaisen heuristiikkansa määrittämään sen, milloin uusi solmu on lisättävä ja vanhoja poistettava [Fernando et al., 2001].

Mukautuvien varjokarttojen ongelmana on niiden raskaus. Pullonkauloiksi nousevat varsinkin laskostuvien reuna-alueiden tunnistaminen sekä hierarkkisen tietomallin ylläpito, joka tekstuurien käsittelyn vuoksi vaatii käytännössä yhteistyötä näytönohjaimen kanssa. Lisäksi mukautuva varjokartta on riippuvainen kameran sijainnista, eli se on yleensä laskettava ainakin osittain uudelleen kun kamera liikkuu.

Epätarkkojen reuna-alueiden löytämisessä suurin ongelma on selvittää, kuinka suurena mikäkin varjokartan piste kamerassa näkyy. Fernando et al. [2001] selvittävät tämän käyttämällä hyväksi näytönohjainten mipmap-tekniikkaa. Näytönohjaimen piirtologiikka pyrkii aina löytämään kullekin syvyystasolle mahdollisimman sopivan mipmap-tason välttääkseen kuvien turhaa skaalaamista. Tätä ominaisuutta voidaan hyödyntää luomalla tekstuuri, jonka jokaisen mipmap-tason väriarvoiksi kirjoitetaan kyseisen tason resoluutio. Varjokartta piirretään käyttäen luotua tekstikarttaa esineiden päällystämiseen. Koska näytönohjain automaattisesti käyttää sellaista mipmap-tasoa jossa skaalaamista tarvitaan vähiten, kunkin pisteen koko globaalissa koordinaatistossa on nyt luettavissa pikselin väriarvosta. Kun näkymää piirretään kameran näkökulmasta, voidaan varjostusta laskettaessa tarkastaa, onko varjokartan pikseli riittävän tarkka. Mikäli näin ei ole, lasketaan käsiteltävän pisteen sisältävälle solulle uusi solmu, jossa varjoalue on piirrettynä tarkemmin. Menetelmä on mielenkiintoinen, mutta lievästä epämääräisyydestään johtuen, se voi toisinaan hukata varjojen reunoja varsinkin hyvin yksityiskohtaisten mallien kohdalla [Lefohn et al., 2007]. Lisäksi uusien karttojen piirtäminen hierarkiaan on raskasta varsinkin, jos piirrettävä näkymä muuttuu paljon.

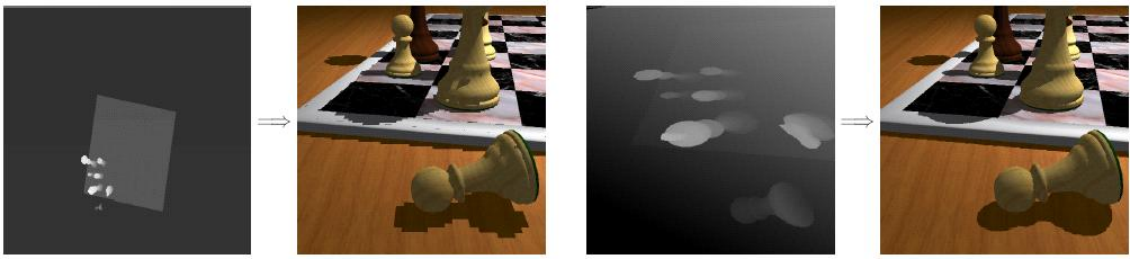
Menetelmän vaatimat tietorakenteet on saatu hiljattain toteutettua suurelta osin myös näytönohjaimella [Lefohn et al., 2005], mikä tekee mukautuvista varjokartoista entistä paremman vaihtoehdon reaaliaikaisuuden näkökulmasta. Riittävää tehokkuutta ja tarkkuutta ei silti vieläkään ole kyetty saavuttamaan [Zhang et al., 2006a]. Viimeisin yritys mukautuvien varjokarttojen

tehostamiseksi on Lefohnin et al. [2007] kehittämä malli, joka perustuu hierarkian kasaavien iteraatiokierrosten optimoinnille, ja näytönohjainten paremmalle hyväksikäytölle. Uusi menetelmä pääsee kehittäjiensä mukaan jo lähelle sitä tehoa, minkä reaaliaikasovellukset vaativat. Tämä antaisi merkkejä siitä, että lähivuosina mukautuvat varjokartat saattavat tulla suositummaksi kuin ne nyt ovat.

5.4.2. Perspektiiviset varjokartat

Tarkemman varjon piirtämiseksi lähelle kameraa ei välttämättä tarvita suurempaa resoluutiota. Vaihtoehtona on piirtää varjokartta siten, että kameran näkökentän etualalla olevat varjot saavat varjokartasta käyttöönsä suuremman osan kuin takana olevat. Kuten jo aiemmin todettiin, projektiomatriisi vääristää näkymää suurentamalla kameran lähellä olevia malleja suhteessa takana oleviin. Stamminger ja Drettakis [2002] ymmärsivät soveltaa tätä havaintoa varjojen tarkentamiseen. He nimesivät menetelmänsä perspektiiviseksi varjokartaksi (perspective shadow map).

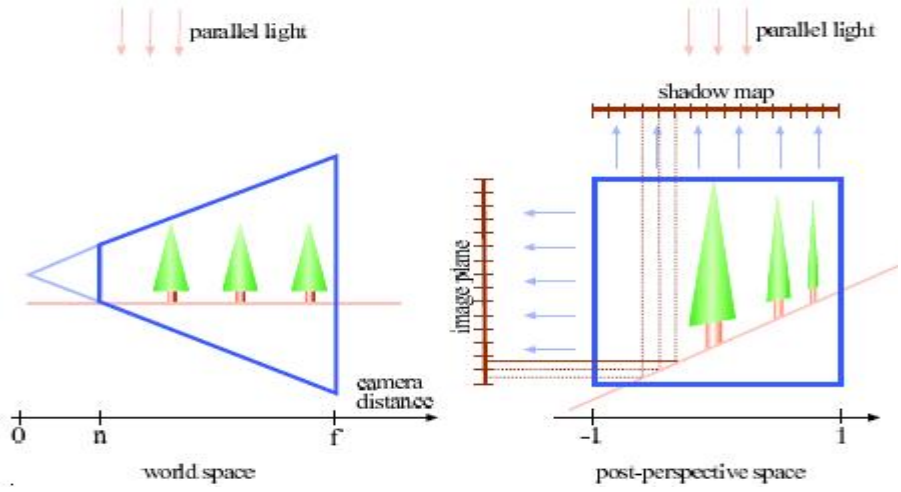
Perspektiivinen muunnos siirtää kaikki näkökentässä sijaitsevat kärkipisteet normalisoituun koordinaatistoon välille $[-1,1]^3$ siten, että esineet kasvavat kuution etureunaa lähestyttäessä. Stammingerin ja Drettakisin [2002] ajatuksena on siirtää näkymä ja valo ensin kameran normalisoituun avaruuteen, ja piirtää varjokartta vasta siellä. Täten lähellä kameraa olevat esineet näkyvät suurempina, ja niille tuotetaan tarkempi varjo kuin kamerasta kaukana oleville. Toteutus on periaatteessa yksinkertainen: muunnetaan näkymän kärkipisteet normalisoituun avaruuteen kameran matriiseilla, ja tehdään tämän jälkeen sama globaalissa koordinaatistossa määritellylle valolle. Nyt varjokartan luominen ja piirtäminen käy samalla tavalla kuin ennenkin. Lopputuloksena kameran lähellä olevat varjot piirtyvät tarkemmin kuin normaalisti. Tarkkuus on pois näkökentän takareunoilla sijaitsevista varjoista, mutta niiden kohdalla yleensä riittääkin pienempi resoluutio (kuva 23).



Kuva 23. Vasemmalla näkymä varjostettuna tasaisella varjokartalla, oikealla sama perspektiivistä varjokarttaa käyttäen [Stamminger and Drettakis, 2002]

Ongelmaksi muodostuu kuitenkin se, että koska valo kerrotaan projektiomatriisilla, myös w-komponentti voi muuttua. Täten äärettömästä sijainnista voi tulla äärellinen ja toisinpäin. Tämä vaikuttaa valon tyyppiin: suunnattu valo voi muuttua kohdevaloksi, ja kohdevalo suunnatuksi valoksi. Projektio myös siirtää kaikki kameran takana olevat valot äärettömyyteen näkökentän toiselle puolelle. Stamminger ja Drettakis [2002] huomauttavat, että jälkimmäinen tilanne saadaan korjattua kirjoittamalla varjokarttaan valosta kauimpana olevien pisteiden syvyysarvot. Tarkempi analyysi erikoistapauksista löytyy Stammingerin ja Drettakisin tekstistä.

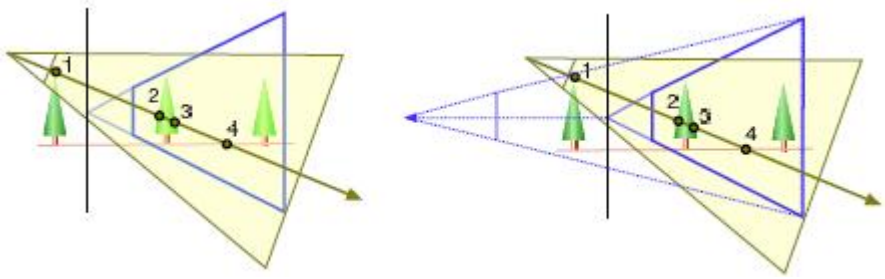
Mikäli valolla on muunnoksen jälkeen äärellinen sijainti, voi laskostumista esiintyä edelleen. Kaikkein pahimmillaan tilanne on silloin, kun suunnattu valo paistaa kaukaa edestä suoraan kameraa kohti. Tällöin lähellä kameraa sijaitsevat esineet ovat perspektiiviseksi muuttuneesta valosta katsottuna hyvin pieniä, ja varjon tarkkuus laskee tasaisten varjokarttojen tasolle. Tätä tilannetta kutsutaan taistelevien näkökenttien (duelling frusta) ongelmaksi, ja se on hankala tilanne kaikille varjokartta-algoritmeille. Parhaat tapaukset ovat ne, jolloin pistevalo sijaitsee samalla xy-tasolla kuin kamera, tai kun suunnattu valo osuu näkökenttään kohtisuorassa katseen suuntavektoriin nähden (kuva 24). Näissä tilanteissa laskostuminen voidaan välttää täysin, ja jälkimmäinen tilanne onkin toiminut lähtökohtana hieman myöhemmin esiteltäville menetelmille.



Kuva 24. Tilanne jossa suunnattu valo paistaa näkökenttään kohtisuorassa. Tässä tilanteessa laskostumista ei esiinny lainkaan [Stamminger and Drettakis, 2002].

Aiemmin todettiin, että projektio siirtää kameran takana olevat valot näkökentän toiselle puolelle. Sama koskee myös esineitä. Tämä aiheuttaa virheen, jos kameran takana oleva esine langettaa varjon kameran eteen. Äärettömän kauas näkökentän eteen siirtyvä esine häviää valon näkökentästä, ja sen varjo jää langettamatta. Stammingerin ja Drettakisin [2002] ratkaisu on

luoda virtuaalinen kamera (kuva 25), jota käytetään varjokartan luontiin, mutta ei itse näkymän piirtämiseen. Se luodaan globaalissa koordinaatistossa liikuttamalla oikean kameran sijaintipistettä taaksepäin niin kauan, että kaikki valon, kameran ja koko näkymän leikkauksessa sijaitsevat esineet mahtuvat sen sisään. Tämä varjostaa viimeisetkin ongelmaesineet, mutta samalla on syytä huomata, että varjon laatu kameran etualalla heikkenee.

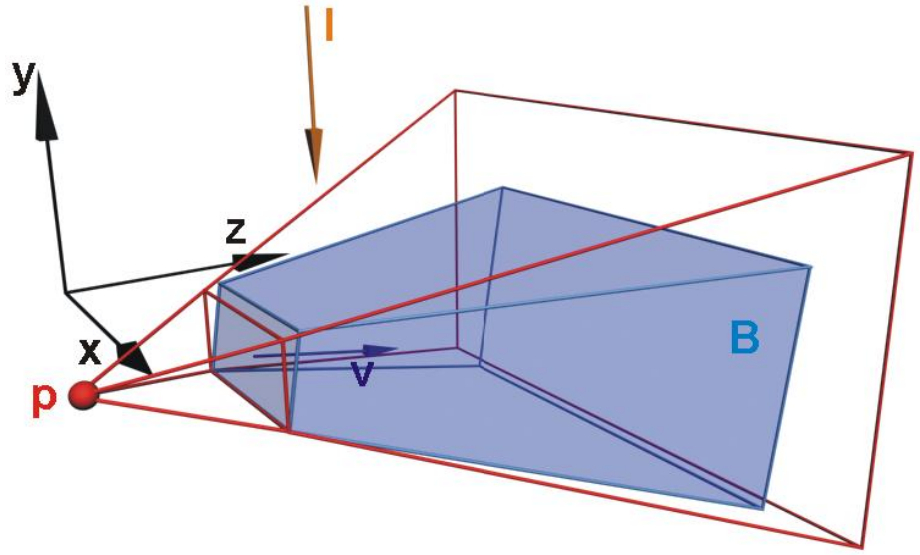


Kuva 25. Virtuaalinen kamera, joka kapseloi kaikki varjostavat esineet [Stamminger and Drettakis, 2002].

Perspektiivisen varjokartan periaate toimii hyvin, mutta menetelmä sisältää myös ikäviä piirteitä. Wimmer et al. [2004] kiinnittivät huomionsa menetelmän ongelmiin. Monet erikoistapaukset tekevät siitä varsin monimutkaisen toteutettavan, eikä näkymän käsittely normalisoidussa koordinaatistossa ole luontevaa. Myös virtuaalisen kameran luominen on epätriviaali tehtävä, ja pahimmassa tapauksessa se laskee varjon laatua merkittävästi. Lisäksi menetelmä korostaa usein liikaa näkökentän etualalla olevia esineitä, jolloin taaempana olevien varjojen tarkkuus kärsii. Näiden ongelmien lisäksi projektiivinen varjokartta on laskettava uudelleen aina kun kamera liikkuu, ja perspektiivin muunnos myös pahentaa syvyyspuskurin epätarkkuuteen liittyviä ongelmia.

5.4.3. Vääristyksen määrittäminen

Perspektiivisen varjokartan ongelmien jälkeen on syytä perehtyä tilanteeseen, jossa sen hyvät ominaisuudet tulevat parhaiten esiin. Aiemmin mainittiin, että laskostuminen poistuu kokonaan tilanteessa, jossa suunnattu valo paistaa näkökenttään kohtisuorassa kameran suuntavektoriin nähden. Wimmer et al. [2004] käyttivät tätä tilannetta apunaan kehittäessään Stammingerin ja Drettakisin [2002] menetelmää edelleen. Heidän lähtökohtansa on, ettei varjokarttaa välttämättä tarvitse vääristää kameran näkökentän mukaan, kuten virtuaalinen kamera jo aiemminkin osoitti. Sen sijaan kameran näkymä voidaan kapseloida toiseen näkökenttään, jonka etu- ja takareunat ovat samansuuntaisia valon suuntavektoriin nähden (kuva 26).



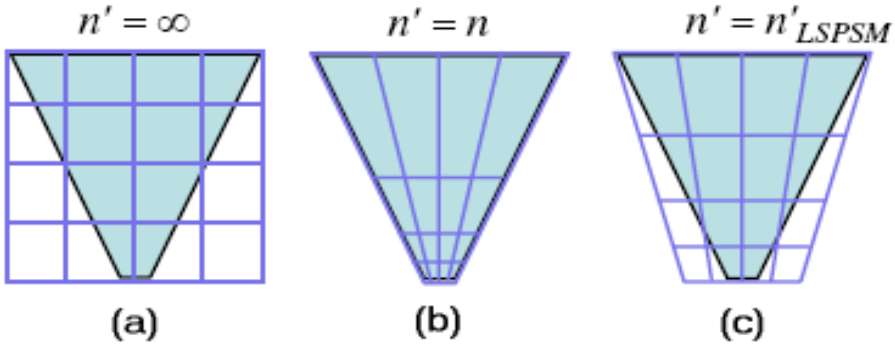
Kuva 26. Virtuaalisen näkökentän määrittäminen valonavaruudessa. l on valon, ja v kameran suuntavektori. B on kameran näkökenttä, ja p on piste, jonka avulla virtuaalista näkökenttää voidaan säätää [Wimmer et al., 2004].

Kun varjokarttaa vääristetään tämän uuden näkökentän mukaan, voidaan aina olla varmoja, että valo paistaa siihen kohtisuorassa. Samalla voidaan välttää perspektiivisen varjokartan erikoistapaukset. Tämä johtuu siitä, että koska virtuaalinen näkökenttä koostetaan aina suhteessa valoavaruuteen, valon on oltava tyypiltään suunnattu. Valon kääntymistäkään ei enää tapahdu.

Näkökentän määrittely aloitetaan luomalla tilavuus, joka sisältää sekä kameran näkökentän että kaikki tarvittavat varjostavat esineet. Tämän jälkeen luodaan virtuaalinen näkökenttä, joka puolestaan kapseloi sisäänsä edellä mainitun tilavuuden ja jonka y-akseli on vastakkaissuuntainen valon suuntavektoriin nähden. Nyt jäljellä on enää viimeinen, ja vääristyksen kannalta keskeisin vaihe: oikean etäisyyden määrittäminen virtuaalisen näkökentän huippupisteestä etutasolle.

Varjokartan vääristymistä voidaan säätää liikuttamalla virtuaalisen näköpisteen kärkeä p eteen- tai taaksepäin, pitäen näkökentän etutaso paikallaan. Etäisyyttä pisteen ja etutason välillä merkitään n':llä. Vääristyminen perustuu siihen, että varjokartan pinta globaalissa avaruudessa on aina samansuuntainen kuin näkökentän xz-taso. Vääristys kohdistuu täten nimenomaan tälle tasolle. Tarkasteltaessa tilannetta, jossa kameran näkökenttä on samansuuntainen kuin virtuaalinen näkökenttä, voidaan havaita vääristysten vaikutus. Jos p viedään taaksepäin äärettömyyteen, tuloksena on tasainen varjokartta (kuva 27a). Mikäli p taas on sama kuin kameran sijaintipiste, tuloksena on Stammingerin ja Drettakisin [2002] esittelemä

vääristys (kuva 27b). Valoavaruudessa määritelty näkökenttä antaa mahdollisuuden säätää vääristymää portaattomasti (kuva 27c).



Kuva 27. Vääristymät valonlähteestä nähtynä: tasainen varjokartta (a), perspektiivinen varjokartta (b) ja valoavaruudessa määritelty varjokartta (c). n' on pisteen p etäisyys virtuaalisen näkökentän etureunasta. Valoavaruudessa määritetyssä kartassa se määräytyy vapaasti (n'_{LSPSM}), ja perspektiivisessä varjokartassa se on sama kuin kameran etutason etäisyys n [Lloyd et. al., 2006].

Arvo n':lle voidaan valita vapaasti kameran sijaintipisteen ja äärettömyyden välillä. Wimmer et al. [2004] tutkivat kirjoituksessaan erilaisia tapoja laskea sille ihannearvo, joka tuottaisi aina mahdollisimman laadukkaat varjot. Koska valo on suunnattu ja paistaa näkökenttään kohtisuoraan ylhäältä, Stammingerin ja Drettakisin [2002] kaavasta (ks. kuva 20) voidaan mitätöidä r_s:n merkitys. Ideaalitilanne olisi siis se, että d ja d_s olisivat aina yhtä suuret. Wimmer et al. [2004] osoittavat tämän ratkaisemisen logaritmiseksi ongelmaksi. Logaritmiset vääristymät eivät taas näytä oikeilta, koska näytönohjaimet eivät tue logaritmisia interpolaatioita (ks. luku 5.1.2). Kirjoittajat päätyvät esittämään n-arvon laskemiseksi seuraavaa kaavaa, jossa z_n ja z_f ovat etäisyyksiä kamerapisteestä kameran etu- ja takareunaan, ja γ on kulma valon ja kameran suuntavektorien välillä:

$$n' = \frac{z_n + \sqrt{z_f z_n}}{\sin \gamma}$$

Joitakin satunnaistilanteita lukuun ottamatta, tämän kaavan todettiin antavan paras kompromissi eri syvyyksissä esiintyvien varjojen tarkkuuteen. Vääristyksen vaikutus kuitenkin vähenee kun valo ei enää ole kohtisuorassa katsevektoriin nähden. Pahimmillaan tilanne on silloin, kun valo ja kamera osoittavat toisiaan kohti, jolloin perspektiivistä vääristymistä ei enää tapahdu. Varjokarttojen muuntamista valoavaruudessa on tutkittu paljon. Mm. Lloyd et al. [2006] ja Zhang et al. [2006b] analysoivat vääristymien käyttämistä eri tilanteissa hyvin perinpohjaisesti.

Vääristymisen määrittelyyn voi toki käyttää muitakin menetelmiä kuin virtuaalista näkökenttää. Chongin ja Gortlerin [2004] menetelmässä on mahdollista valita pieni määrä tasoja, joilla varjokartan pikselin voidaan periaatteessa taata vastaavan tasan yhtä kameran pikseliä. Menetelmä on jossain määrin raskas, sillä se vaatii erillisen varjokartan piirtämistä jokaista määriteltyä tasoa varten. Tästä, ja muista pienistä ongelmistaan huolimatta se on hyvä vaihtoehto esimerkiksi seinien ja lattioiden kaltaisten esineiden varjostamiseen. Toinen paljon suosiota saanut menetelmä on Martinin ja Tanin [2004] kehittämä vääristysmalli, missä kameran näkökentän kärkipisteet projisoidaan valoon ja niiden avulla muodostetaan pisteet kapseloiva puolisuunnikas. Muuntamalla valokartta tähän nk. t-avaruuteen voidaan kameran näkökentän ulkopuolelle jäävä hukkatila poistaa lähes kokonaan. Menetelmän avulla saadaan vähennettyä myös kaikkiin vääristysmenetelmiin liittyvää varjojen reunojen väreilyä, jota esiintyy varjon tarkkuuden muuttuessa pitkin näkökenttää.

Kaikkien vääristyksiin perustuvien menetelmien kohdalla on oltava tarkkana syvyyspuskurin tarkkuuden kanssa. Vääristymästä johtuen ongelma on niissä paljon suurempi kuin tasaisten varjokarttojen kohdalla. Varsinkin puolisuunnikkaan määrittelemään t-avaruuteen muunnettuina kärkipisteiden syvyydet puristuvat käytännössä samoiksi [Martin and Tan, 2004]. Menetelmän kehittäjät kehottavatkin käyttämään Brabecin et al. [2002a] esittelemiä lineaarisia syvyysarvoja.

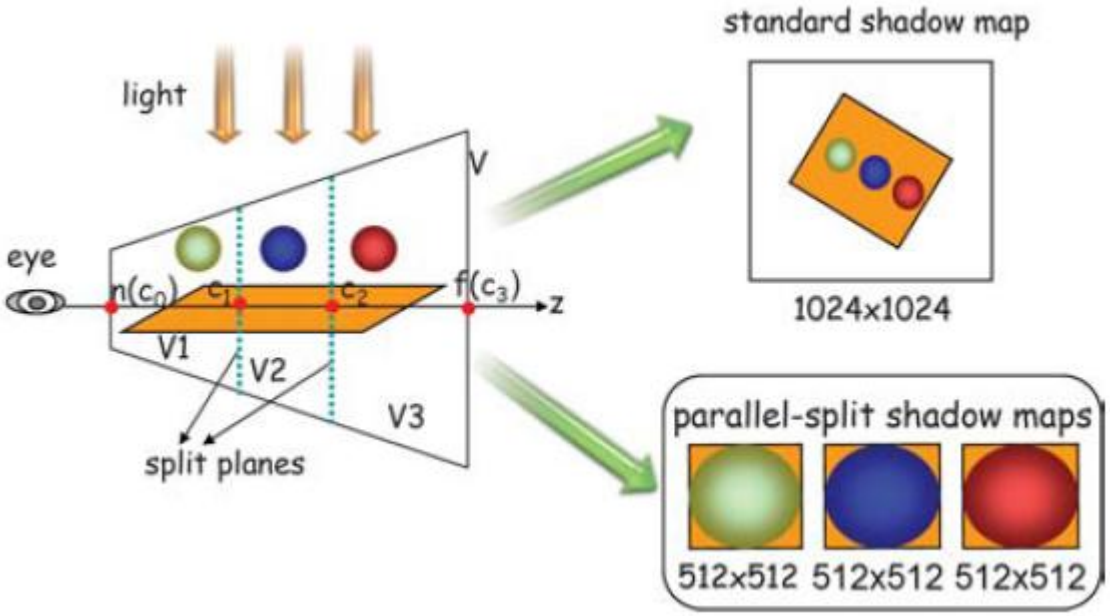
5.4.4. Näkökentän jakaminen

Varsinkin suurten näkymien kohdalla, joissa näkymä ulottuu kauas ja valo on suunnattua, on yleensä parasta jakaa näkökenttä pienempiin osiin. Tämä jako voidaan tehdä monella tavalla [Lloyd et al., 2006], mutta tässä tutkielmassa käydään läpi näkökentän pilkkominen sen syvyysakselia pitkin. Tällöin menetelmään pystytään kätevästi soveltamaan myös valoavaruudessa määriteltyjä perspektiivisiä varjokarttoja. Näkökentän pilkkominen muistuttaa jossain määrin mukautuvaa varjokarttaa, mutta iteratiivisen, varjojen reunoihin kohdistuvan tarkentamisen sijaan siinä luodaan vain yksitasoinen jako suhteessa kameraan.

Kuten jo aiemmin todettiin, laskostumisen kannalta täydellisen tilanteen (jossa valon ja kameran pisteet ovat kaikkialla samankokoiset) ratkaiseminen on logaritminen ongelma, eikä sovellu näytönohjainten piirtorutiineihin. Näkökentän pilkkomisella useisiin syvyystasoihin voidaan kuitenkin estimoida tätä tilannetta [Zhang et al., 2006a]. Tadamura et al. [2001] käyttivät syvyysarvojen mukaan pilkkomista ensimmäisinä, mutta sittemmin Zhang et al. [2006a] ovat kehittäneet menetelmää eteenpäin. Tadamuran et al. [2001]

menetelmä mm. määrittää näkökenttää leikkaavat tasot monimutkaisin laskutoimituksin, käyttää vaihtelevan kokoisia varjokarttoja ja vaatii erillisen piirtokerran jokaiselle näkymän osalle. Uudempi menetelmä on päässyt eroon näistä kaikista ongelmista. Se myös olettaa, että päävalonlähde on aina suunnattu. Tämä ehto saavutetaan määrittelemällä leikeltävä näkökenttä suhteessa valoon.

Menetelmä alkaa näkökentän pilkkomisella syvyysakselia pitkin. Tämän jälkeen sen jokaiselle osalle määritetään oma varjokartta, joka kattaa vain kulloisenkin näkökentän osan. Varjokartat piirretään tavalliseen tapaan. Piirtokertoja tulee useita, mutta siitä ei ole merkittävää haittaa, koska piirto kohdistuu kulloinkin vain yhteen näkökentän osaan, eikä samaa kohtaa koskaan piirretä kahdesti. Kameranäkymän piirto toimii lähdes normaaliin tapaan. Ainoa muutos tavalliseen on se, että on selvitettävä varjokartta, mihin piste kuuluu.



Kuva 28. Pilkottu näkökenttä. Oikealla ylhäällä normaali varjokartta, alhaalla pilkkomismenetelmällä syntyneet varjokartat. c0-c3 ovat näkökentän osien (V1-V3) pilkkomiseen käytetyt etäisyydet kamerasta (n on etureuna ja f takareuna) [Zhang et al., 2006a].

Zhang et al. [2006a] vertailevat kolmea eri tapaa, millä näkökenttä voidaan jakaa: tasaisesti, logaritmisesti tai käytännöllisesti. Kaksi ensimmäistä vastaavat ominaisuuksiltaan tasaista ja perspektiivistä varjokarttaa. Tasainen jako pilkkoo näkökentän osat samankokoisiin osiin, mikä johtaa varjojen epätarkkuuteen kameran lähellä. Logaritminen tapa taas on tarkka edessä, mutta taaemmaksi mentäessä varjojen laatu heikkenee liian nopeasti. Kehittäjien suosittelema ns. käytännöllinen jakoperuste on kahden edellisen keskiarvo, johon on vielä

lisätty ylimääräinen muunnos käytännön sanelemien säätöjen tekemiseen. Tämän menetelmän avulla näkökentän jakaminen käy helpommin kuin esimerkiksi Tadamuran et al. [2001] menetelmän monimutkaisilla laskutoimituksilla.

Kun näkökenttä on pilkottu, sen jokaiselle osalle luodaan oma varjokarttansa, joiden koko on periaatteessa vapaavalintainen. Kutakin kuvaa piirrettäessä asetetaan kamera siten, että sen näkökenttä kapseloi mahdollisimman tiukasti vain sellaiset esineet, jotka langettavat varjoa näkökentän käsiteltävälle osalle. Tämä vaihe mahdollistaa hukkatilan minimoinnin paremmin kuin yksi koko näkymän kattava varjokartta (kuva 28). Voi myös käydä niin, ettei näkökentän alueella ole yhtään varjostettavaa esinettä. Tällöin varjokartta voidaan jättää kokonaan piirtämättä.

Kameranäkymän piirtäminen tapahtuu kuten normaalistikin, mutta nyt kutakin pistettä kohti täytyy ensin selvittää oikea varjokartta. Tämä voidaan tehdä joko piirtämällä jokainen näkökentän osa erikseen [Tadamura et al., 2001], tai käyttämällä pikselivarjostinta päättämään oikea varjokartta pisteen syvyysarvon perusteella. Siellä kartan selvittäminen on yksinkertainen syvyysvertailutoiminto. Täytyy kuitenkin muistaa, että pisteiden syvyysarvot ovat nyt välillä 0-1, mikä tarkoittaa, että näkökentän leikkaustasojen syvyysarvot on vertailua varten muunnettava ensin tähän koordinaatistoon. Zhang et al. [2006a] antavat esimerkin Direct3D-ympäristölle. Usein käy niin, että jokin esine ylittää leikkaustason, ja on täten kahden tai useamman näkökenttäosan alueella samaan aikaan. Tadamura et al. [2001] menetelmällä tämä vaati käytännössä ylimääräisten leikkaustasojen määrittelyä, mutta pikselivarjostimien ansiosta niitä ei tarvita, ja näkymä tarvitsee piirtää vain kerran.

Näkökentän jakamisella saavutetaan siis monia etuja. Se sopii täydellisesti yhteen varjokarttojen vääristysmenetelmien kanssa, ja auttaa myös ratkaisemaan niihin liittyviä ongelmia. Esimerkiksi taistelevien näkökenttien aiheuttama palikoituminen vähenee huomattavasti, kun näkökentän etualalle määritellään erillinen varjokartta. Esineiden tehokkaasta kapseloinnista johtuen varjokarttojen ala tulee myös paremmin hyödynnetyksi. Näistä syistä syvyyskarttaa voidaan kutistaa, jolloin niiden kokonaismuistinkulutus on pienempi kuin yhtä, suurta kuvaa käytettäessä. Uusimmat näytönohjaimet tukevat usean eri kuvapinnan piirtämistä samaan aikaan, mikä on omiaan tehostamaan tämän menetelmän toimintaa. Muissa laitteissa voidaan käyttää esimerkiksi kuutiokarttoja nopeamman tekstuurikäsittelyn saavuttamiseksi.

Lloyd et al. [2006], jotka testasivat kattavasti varjokarttojen käsittelyyn ja optimointiin liittyviä menetelmiä, totesivat näkökentän pilkkomisen ja

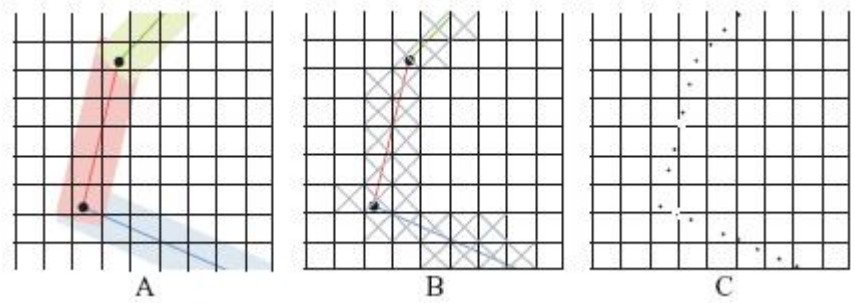
valoavaruudessa määritellyn tekstikartan vääristämisen parhaaksi menetelmäksi varsinkin suurissa näkymissä, joissa näkökenttä jatkuu pitkälle. Molemmat menetelmät ovat lisäksi toteutettavissa suhteellisen helposti ilman monimutkaisia tietorakenteita ja erikoistapauksia.

5.5. Varjokarttojen muunnelmat

Varjokarttojen ei tarvitse välttämättä sisältää ainoastaan syvyysarvoja, vaan niihin voidaan tallentaa, esimerkiksi väripuskuria hyödyntäen, paljon muutakin. Seuraavassa käydään läpi pari menetelmää, jotka käyttävät piirtopintoja hyvin monipuolisesti. Niissä kiteytyy myös varjokarttojen monipuolisuus: usein näytönohjaimista löytyy kuin yllättäen juuri riittävät toiminnot kokeellisempienkin ratkaisujen tekemiseen, jos mielikuvitusta vain löytyy.

5.5.1. Siluettikartta

Laskostumisen tärkein aiheuttaja on se, etteivät bittikarttakuvat pysty approksimoimaan varjon reunaa riittävän tarkasti. Sen et al. [2003] keksivät käyttää syvyyskartan lisäksi myös toista kuvaa, siluettikarttaa, jonka pikselin väriarvoihin tallennetaan koordinaattipiste, joka ilmaisee varjon reunan tarkan kulkureitin pisteen läpi (kuva 29). Piirtämällä viiva pikseleiden koordinaattien väliin, saadaan muodostettua huomattavasti paljon tarkempi reuna kuin ainoastaan pikseleitä käyttämällä. Tämä mahdollistaa myös projektiivisen laskostumisen korjaamisen. Siluettikartan pisteet sijaitsevat limittäin suhteessa varjokarttaan, eli sen pikseleiden kulmat ovat neljän ympäröivän varjokartan pikselin keskipisteet. Saman logiikan mukaan normaali varjokartta voidaan ajatella siluettikarttana, jossa kaikki koordinaatit ovat aina pisteen keskellä. Sen et al. [2003] tallentavat koordinaattipisteet pikselin paikallisessa koordinaatistossa suhteessa sen reunoihin.



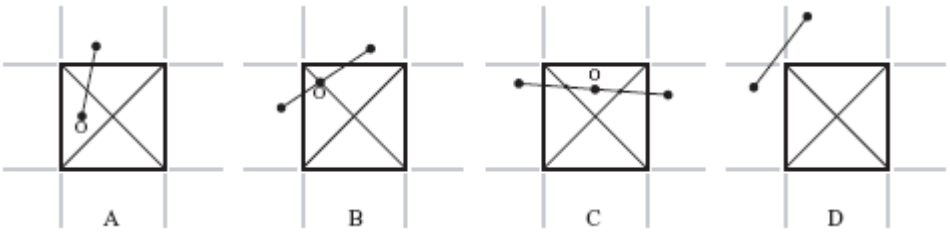
Kuva 29. Siluettikartta. Piirrettävät siluettiviivat (a), käsiteltävät pikselit (b) ja kunkin pisteen koordinaatti (c) [Sen et al., 2003].

Siluettikarttamenetelmä koostuu kolmesta vaiheesta. Ensimmäisessä piirretään syvyyskartta samalla tavalla kuin aikaisemminkin. Ainoa asia mikä

kannattaa ottaa huomioon, on se, että syvyysarvot täytyy pystyä lukemaan takaisin. Tämä voi vaatia joillakin rajapinnoilla niiden kirjoittamista värikomponentteihin.

Toisessa vaiheessa luodaan itse syvyyskartta. Tämä alkaa etsimällä varjostavalle mallille siluettireunat samaan tapaan kuin varjomallien tapauksessa. Ne piirretään valon näkökulmasta siluettikarttaan antamalla pikselivarjostimelle ylimääräisinä syötteinä reunan kärkipisteiden koordinaatit. Piirrettävän viivan on oltava sen verran paksu, että kaikki pisteet joiden kautta reuna kulkee, tulevat varmasti piirretyksi (kuva 29a). Tämä takaa sen, ettei yksikään siluettireunan piste jää määrittelemättä. Paksusta viivasta johtuen piirretyksi tulee sellaisiakin pisteitä, jotka eivät sisällä ääriviivaa. Lopullinen päätös siitä, asetetaanko piste kuuluvaksi siluettireunaan, tehdään pikselivarjostimessa.

Se, onko käsiteltävä siluettikartan piste varjon reunalla, saadaan selville reunan kärkipisteiden avulla. Jos toinen niistä sijaitsee pikselin alueella, piste on yksiselitteisesti reunalla (kuva 30a). Muussa tapauksessa luodaan kaksi suoraa, jotka lävistävät pisteen vastakkaisista kulmista. Kärkipisteiden ja väliarvolaskennan avulla saadaan selvitettyä näiden suorien ja reunan leikkauspisteet. Jos reuna leikkaa käsiteltävän pikselin alueella vain toisen halkaisijan (kuva 30b), leikkauspiste tallennetaan koordinaatiksi (muutettuna pisteen paikalliseen koordinaatistoon). Jos se leikkaa molemmat halkaisijat, lopullinen koordinaatti on näiden pisteiden puolivälissä (kuva 30c). Jos leikkauspiste taas on pisteen alueen ulkopuolella, koordinaatti jätetään tyhjäksi (kuva 30d). Pyöristysepätarkkuudet vaativat käytännössä sen, että niukasti kulmien ulkopuolelta kulkeva reuna lasketaan kulkevaksi pisteen kautta. Nämä, vain niukasti naapurin puolella olevat leikkauspisteet muunnetaan lopuksi pikselin rajojen sisään, että jatko toimii moitteettomasti. Jos reunan syvyys koordinaattipisteessä on suurempi kuin kaikissa kulmiin liittyvissä syvyyskartan pikseleissä, piste (riippumatta siitä onko se reuna vai ei) on varjossa, eikä koordinaattia aseteta.



Kuva 30. Siluettikartan pisteen halkaisijoiden käyttö [Sen et al., 2003].

Kolmannessa vaiheessa näkymä piirretään kameran näkökulmasta. Näkyvyystarkastelu tehdään käsiteltävän siluettipisteen kulmiin liittyville

syvyyskartan pisteille. Jos niillä kaikilla on sama arvo, piste on yksikäsitteisesti joko valossa tai varjossa, eikä siluettikarttaa tarvitse käyttää. Muussa tapauksessa varjostus selvitetään jakamalla käsiteltävä siluettipiste ensin neljään osaan käyttämällä sen, ja sen neljän naapurin koordinaatteja. Tämän jälkeen selvitetään, minkä neljänneksen alueella piirrettävä kameranäkymän pikseli sijaitsee, ja päätetään varjostus kyseisen neljänneksen kulmaa vastaavan syvyyskartta-arvon perusteella. Myös tämä toimenpide tehdään pikselivarjostimessa. Koska numeeriset epätarkkuudet voivat toisinaan johtaa siihen, että jotain pistettä luullaan virheellisesti reunaksi, kaikki siluettikartan koordinaattiarvot alustetaan olemaan pisteen keskellä.

Menetelmä on kehittäjiensä mukaan nopeampi kuin varjotilavuusmenetelmä, mutta monimutkaisista varjostimistaan ja runsaista tekstikarttojen takaisinluvuistaan johtuen se on selvästi hitaampi kuin tavalliset varjokartat. Koska kukin siluettikartan piste sisältää vain yhden koordinaatin, menetelmä ei myöskään pysty selviämään tilanteista, joissa kaksi reunaa osuu saman pisteen alueelle. Asiassa normaalisti auttava varjokartan resoluution kasvattaminen johtaa suurempaan määrään pikselikohtaista laskentaa, joten tällaiset tilanteet eivät ole ratkaistavissa ilman merkittävää tehonalennusta.

Suurin ongelma on kuitenkin se, että siluettikarttamenetelmä vaatii mallien reunojen laskemista, ja täten myös tietämystä piirrettävästä geometriasta. Näin sen yhteydessä menetetään monet varjokarttojen parhaista puolista, ja sitä voidaankin kutsua enemmän varjokarttojen ja –mallien hybridimenetelmäksi. Varjokarttojen laskostumisen korjaamista varjomonikulmioiden avulla on tutkittu muutenkin. Tunnetuimmat suuntaukset lienevät McCoolin [2000] tapa rakentaa varjomallit syvyyskartan pohjalta, sekä Chanin ja Durandin [2004] vastaava menetelmä, jossa varjomallit piirretään ainoastaan niille alueille, missä laskostumista ilmenee. Kaikki hybridimenetelmät perivät varjomalleista ja –kartoista niin hyviä kuin huonojakin puolia, eikä niitä käydä tämän tutkielman puitteissa läpi tarkemmin.

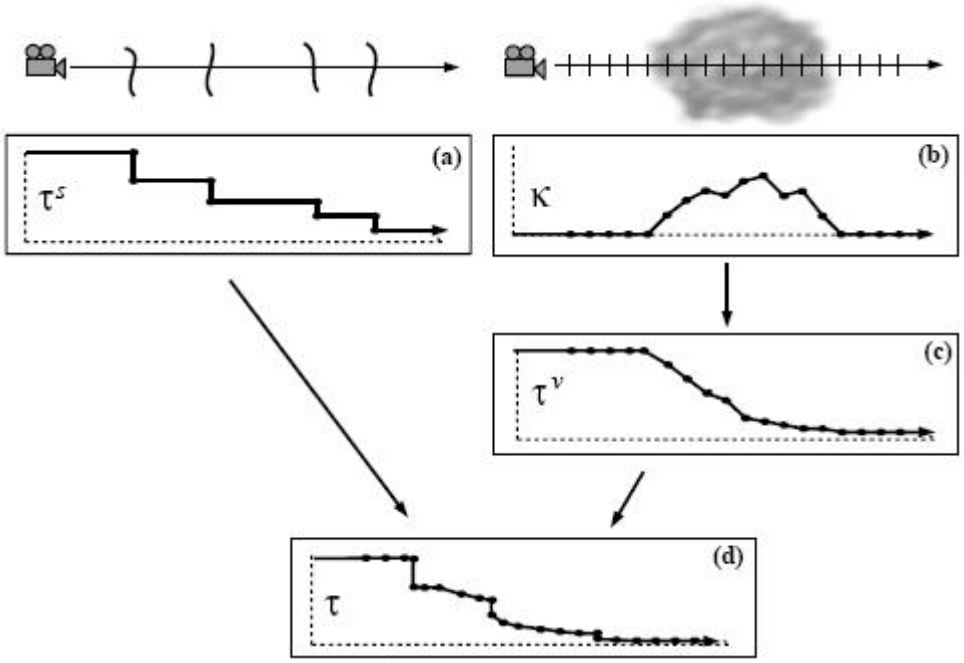
5.5.2. Hienojakoisten esineiden varjostaminen

Syvät varjokartat (deep shadow maps) [Lokovich and Veach, 2000] ovat menetelmä, jonka avulla on mahdollista varjostaa läpikuultavia pintoja ja tiheydeltään vaihtelevia, esimerkiksi kaasumaisia elementtejä. Lisäksi yhdessä pikselisuodatuksen kanssa käytettynä se tarjoaa itesevarjostavan menetelmän hyvin hienojakoisten esineiden (esimerkiksi hiusten) varjostamiseen. Molemmat tapaukset ovat sellaisia, joita normaaleilla varjokartoilla ei ole käytännössä mahdollista toteuttaa. Syvien varjokarttojen menetelmä ei sellaisenaan toimi riittävän nopeasti reaaliaikaisten sovellusten tarpeisiin, mutta ainutlaatuisista ominaisuuksistaan johtuen sen pohjalta on tehty

yksinkertaistettuja versioita, jotka kykenevät myös tosiaikaiseen varjostamiseen.

Syvä varjokartta ei tallenna syvyysarvoja vaan valaisufunktioita, jotka laskevat varjostusarvot kullekin pisteelle. Funktio saa syötteenään muunnetun pisteen syvyysarvon ja palauttaa luvun välillä 0-1 riippuen siitä, kuinka paljon valoa muunnetun pisteen syvyyteen pääsee. Savu on hyvä esimerkki tällaisen toiminnallisuuden käytöstä. Valo valaisee savupilven reunoja, mutta syvemmälle pilveen mentäessä valonsäde heikkenee, kunnes riittävän kauas jatkettuaan se ei valaise enää ollenkaan.

Funktion suoritus, sellaisena kuin Lokovich ja Veach [2000] sen esittelevät, lähtee liikkeelle arvosta 1. Tämä tarkoittaa täyttä valaistusastetta. Tämän jälkeen sädettä lähdetään seuraamaan valosta poispäin. Valoarvo muuttuu kahden alifunktion perusteella. Ensimmäiselle voidaan määritellä tietyille syvyysarvoille tasoja, jotka muuttavat valaistusta porrasmaisesti (kuva 31a). Tyypillinen esimerkki tällaisesta tasosta on esimerkiksi himmennetty ikkuna. Toinen funktio on hieman samankaltainen, mutta siinä himmennyksen aste tasojen välillä päätetään laskemalla väliarvo edellisen ja seuraavan tason perusteella (kuva 31c). Lopullinen valaistusarvo saadaan yhdistämällä nämä tekijät (kuva 31d).



Kuva 31. Syvän varjokartan funktiot [Lokovich and Veach, 2000]. τ^s on porrasmaisesti, ja τ^v aineen tiheyden (funkio κ) mukaan laskeva valaistus, jotka kertomalla saadaan lopullinen valaistusfunktio τ.

Funktioiden palautusarvoja voidaan suodattaa samalla tavalla kuin mitä tahansa tekstikarttoja. Näin saadaan otanta valaisuarvoista tietyllä säteellä,

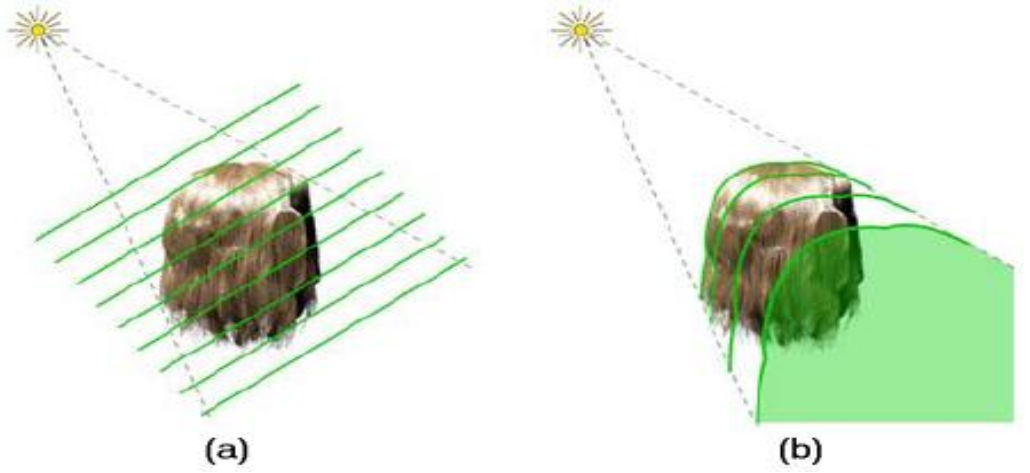
mitä voidaan käyttää esimerkiksi hiusten varjostamiseen. Yksittäiset hiukset ovat liian ohuita suurillekin varjokartoille, joten mallin avulla voidaan määrittää tiheysjakauma, mikä mahdollistaa realistisen näköisen itsevarjostuvuuden ohuille esineille.

Kim ja Neumann [2001] yksinkertaistivat syvän varjokartan ajatuksen voidakseen käyttää sitä reaaliaikaiseen hiusten varjostamiseen. Heidän menetelmänsä pilkkoo ensiksi varjostettavan esineen valoon päin osoittavilla tasoilla (kuva 32a). Tämän jälkeen näkymä piirretään valon näkökulmasta niin monta kertaa kuin tasoja on, siirtäen valon näkökentän etutaso joka piirtokerralla vastaavalle tasolle. Kun tämä on tehty, kaikki edellisessä vaiheessa piirretyt kuvat käydään läpi, kulkien järjestyksessä valosta poispäin. Edellinen kuva sulautetaan aina seuraavaan siten, että lopputuloksena on sarja kuvia, joista kukin sisältää varjostuksen määrän tasoa vastaavalla etäisyydellä valosta. Näiden karttojen perusteella saadaan helposti selvitettyä, mikä valaistuksen määrä vallitsee missäkin pisteessä tietyllä etäisyydellä valonlähteestä.

Kimin ja Neumannin menetelmä vaatii paljon piirtokertoja, eli sillä ei voi reaaliaikaisesti saavuttaa kovin vaikuttavia lopputuloksia. Mertens et al. [2004] kehittivät sitä lähemmäksi Lokovichin ja Veachin [2000] menetelmää. He keräävät näytteitä piirrettävästä esineestä, ja tilastoivat tiheydet ryppäiksi. Kun valonsäde saapuu tällaisen ryppään alueelle, se himmenee lineaarisesti määritellyn tiheyden mukaisesti.

Viimeisin kehitelmä reaaliaikaisten hiusmallien luomiseen lienee Yukselin ja Keyserin [2007] menetelmä. Kuten edellisetkin menetelmät, se vaatii k tekstikarttaa, joiden perusteella varjostus lasketaan. Algoritmi etenee kolmessa vaiheessa.

Ensimmäinen piirtää tavallisen syvyyskartan, ja jakaa varjostettavan mallin k osaan kunkin varjokartan pikselin kohdalta. Tässä erona esimerkiksi Kimin ja Neumannin [2001] menetelmään on se, että osat voivat olla eri pikseleille erikokoisia riippuen siitä, kuinka syvä malli kussakin kohtaa on. Saman tekstikartan pisteet siis viittaavat aina samaan k:n arvoon, mutta eivät välttämättä samaan syvyysarvoon (kuva 32b).



Kuva 32. Kimin ja Neumannin [2001] (a) ja Yukselin ja Keyserin [2007] (b) tavat jakaa syvyyskartan tasot [Yuksel and Keyser, 2007].

Toisessa vaiheessa malli piirretään uudelleen, jolloin kunkin pisteen kohdalla tarkastetaan kuinka kaukana valosta se on, ja lisätään varjostusastetta kaikille sen takana oleville k:n varjotasoille. Syvyys- ja varjostavuusarvojen esittämiseen käytetään tekstikartan värikomponentteja. Näin niiden sulauttaminen keskenään hoituu nopeasti näytönohjaimella, ja kolme tasoa voidaan esittää yhden pikselin avulla [Yuksel and Keyser, 2007]. Kolmannessa vaiheessa näkymä piirretään, ja varjostusarvot muunnetulle pisteelle voidaan hakea syvyyttä vastaavalta tasolta.

Edellisiä menetelmiä on käytetty lähinnä hiusten varjostamiseen, mutta ne voidaan saada helposti tukemaan myös esimerkiksi itsevarjostavia savutehosteita. Tämä alue tulleekin jatkossa olemaan yhä enemmän tutkinnan kohteena. Varjostusarvojen vapaasta määrittämisestä yksinkertaisten totuusarvojen sijaan on kyse myös silloin, kun on tarkoitus mallintaa kovien varjojen sijasta pehmeitä puolivarjoja.

6. Pehmeät varjot

Toistaiseksi on käsitelty ainoastaan äärettömän pieniä valonlähteitä ja varjoja, joiden valaistusarvot perustuvat totuusarvoihin. Tällaisia varjoja ei kuitenkaan esiinny todellisuudessa, vaan oikeasti valonlähteellä on aina äärellinen tila, mikä aiheuttaa varjoreunojen luonnollisen pehmenemisen. Pehmeät varjot siis luovat kuvaan realismia, ja tekevät näkymästä myös monesti selkeämmän, sillä kovareunaiset varjot sekoittuvat helposti muihin graafisiin elementteihin. Aikaisemmin esiteltyjä suodatusmenetelmiä ei lasketa pehmeiden varjoalgoritmien joukkoon, vaan pehmeät varjoreunat, sellaisina kuin ne tästä eteenpäin käsitetään, määräytyvät sen mukaan, kuinka suuri osa valonlähteestä on mistäkin pisteestä katsottuna näkyvissä. Aiemmin tilanne oli periaatteessa

sama, mutta koska valonlähde oli äärettömän pieni, se oli aina joko kokonaan esillä, tai kokonaan varjostavan esineen peittämä. Vastedes ei siis pyritä kysymään, näkyykö valonlähde varjostettavasta pisteestä, vaan kuinka suuri osa valonlähteestä on näkyvissä. Varjon ominaisuuksiin vaikuttaa jatkossa siis valon, varjostavan ja varjostettavan pinnan etäisyyksien ja asennon lisäksi myös valon koko. Aluetta, jossa valonlähde on näkyvissä vain osittain, kutsutaan puolivarjoksi.

Todellisuudessa tilallinen valonlähde koostuu äärettömästä määrästä pistevaloja joten on selvää, ettei tällaista tilannetta ole mahdollista tietokoneella (varsinkaan reaaliajassa) mallintaa. Käytännössä kaikki pehmeitä varjoja tosiaikaisesti muodostavat algoritmit pyrkivätkin luomaan vain mahdollisimman uskottavia illuusioita, eivätkä edes pyri olemaan fysikaalisesti oikeita. Lähestymistapa on hyväksyttävä, koska pehmeät varjot ovat luonnostaankin utuisia ja epäselviä, eikä niiden mahdollisia virheitä varsinkaan animoidussa näkymässä helposti huomaa.

Reaaliaikaisessa tietokonegrafiikassa puolivarjot pilkotaan käytännön syistä usein sisempään ja ulompaan osaan. Nämä puoliskot erotetaan jakamalla puolivarjoalueet kovien varjojen reunanelikulmiolla eli tasoilla, jotka kulkevat varjostavan mallin siluettireunojen ja valon keskipisteen kautta. Useimmat reaaliaikamenetelmät pyrkivät vaalentamaan sisempää puolivarjoaluetta ja tummentamaan ulompaa. On syytä huomauttaa, että sisempi ja ulompi puolivarjo ovat puhtaasti ohjelmointiteknisiä käsitteitä, eikä niitä useinkaan voida todellisessa maailmassa erottaa toisistaan. Kovan varjon ulkopuolelle leviävien puolivarjojen tummentamiseen ja vaalentamiseen perustuvat menetelmät kärsivät myös monista ongelmista, joita ei aiempien algoritmien kohdalla ole tullut vastaan. Esimerkiksi yhden valon ja useiden esineiden tapauksessa käy usein niin, että lopullinen pehmeä varjo on suurempi kuin yksittäisistä malleista lähtevien erillisten varjojen unioni [Hasenfrantz et al., 2003]. On myös täysin mahdollista, että laajan valonlähteen synnyttämät puolivarjot hävittävät täysvarjoalueen kokonaan.

Koska varjostusasteet eivät enää ole totuusarvoja vaan ne määräytyvät vapaasti välillä [0-1], luvussa 2.4 esiteltyä usean valon piirtoalgoritmia on syytä hieman muuttaa. Pehmeiden varjojen tapauksessa näkymä piirretään ensin väri- ja syvyyspuskuriin siten, että käsiteltävä valo on päällä. Tämän jälkeen lasketaan varjostusastekartta, jonka avulla piirretyn kuvan varjoalueita himmennetään (yleensä alfakanavan avulla). Tämä tulos sulautetaan sitten taustavalon ja muiden valojen muodostamien näkymien joukkoon keräyspuskurissa. Tämä menetelmä on jonkin verran hitaampi kuin aikaisempi,

koska ensimmäinen piirto joudutaan tekemään myös sellaisille pikseleille, jotka eivät näy lopullisessa kuvassa.

Hasenfrantz et al. [2003] käyvät kirjoituksessaan perinpohjaisesti läpi yleisimmät pehmeitä varjoja piirtävät algoritmit. Erikoisominaisuuksistaan johtuen niille ominaista on erikoistapauksien paljous sekä yleinen epämääräisyys, mikä vaikeuttaa myös niiden luokittelua. Tässä tutkielmassa käydään aluksi läpi muutama perusmenetelmä, joita voidaan käyttää reaaliaikaisesti korkeintaan hyvin rajoitetuissa tapauksissa. Tämän jälkeen tehdään silmäys pistevaloja laajentaviin menetelmiin, joissa fysikaalinen oikeaoppisuus on hylätty tehokkuuden hyväksi. Pääpaino on uudemmissa, takaisinprojisointiin perustuvissa menetelmissä, joiden avulla voidaan nykyisten näytönohjainten ominaisuuksien avulla saavuttaa jo varsin uskottavia, ja lähes oikeaoppisiakin pehmeitä varjoja.

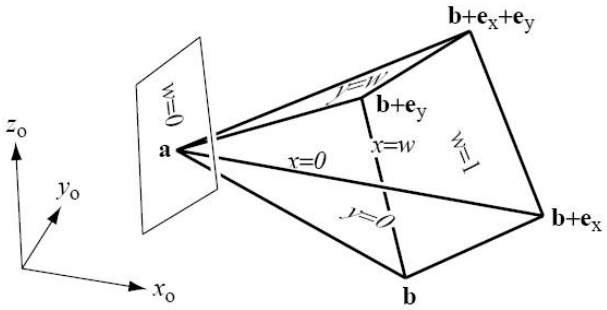
6.1. Kovien varjojen yhdistäminen

Yksinkertaisin tapa piirtää pehmeitä varjoja, on approksimoida tilannetta, jossa tilallinen valonlähde koostuu äärettömästä määrästä pistevaloja. Oikeaoppisinta onkin jakaa valonlähde näytepisteisiin, ja piirtää jokaista vastaava kova varjo. Tämän jälkeen varjot sulautetaan esimerkiksi keräyspuskurissa (tai erillisessä tekstikartassa, mikäli keräyspuskurin resoluutio ei ole riittävä) yhdeksi varjokartaksi, jota voidaan käyttää normaalin tekstikartan tapaan. Tämä luku käsittelee menetelmiä, jotka mallintavat tilallista valoa juuri näytepisteiden avulla. Koska jokainen piste vaatii oman piirtoprosessinsa, on kuitenkin selvää, ettei näitä menetelmiä ole mahdollista käyttää reaaliaikaiseen varjojen luontiin muuten kuin hyvin rajatuissa näkymissä tai laitteistoympäristöissä. Ne antavat kuitenkin hyvän kuvan siitä, miten pehmeät varjot käytännössä toimivat.

Heckbert ja Herf [1997] sekä Herf [1997] luovat kullekin näytepisteelle oman, totuusarvoista koostuvan kuvansa, ja sulauttavat ne sitten aiemmin mainitulla tavalla yhdeksi. Sulauttaminen tehdään yksinkertaisen lisäysoperaation avulla, joten se voidaan suorittaa joka varjostusarvokartalle erikseen heti kun se on laskettu. Näin menetelmän muistivaatimukset pysyvät pieninä (yksi tekstikartta totuusarvoista koostuvan varjon laskemiseen, ja keräyspuskuri tulosten yhdistämistä varten).

Menetelmä perustuu luvussa 3 esitetyn tasoprojisoinnin käyttöön. Tilallisen valon alueelta valitaan ensin vapaasti n kappaletta näytepisteitä (yleensä käytetään ainakin valon ääripisteitä, ja muut näytteet voidaan valita joko säännöllisesti tai esimerkiksi satunnaisotantaa käyttäen). Tämän jälkeen rajataan nelikulmion muotoinen alue, jota vasten varjo langetetaan. Näiden avulla luodaan näkökenttäpyramidi, jonka kärki asetetaan valon

näytepisteeseen, ja alareuna rajoitetaan varjostettavaan nelikulmioon. Näkökentän luominen tapahtuu erityisen matriisin avulla (kuva 33). Menetelmä rinnastuu tasoprojisoituihin varjoihin siinäkin suhteessa, että varjostavat ja varjostettavat esineet on eriteltävä toisistaan, ja että varjostettavan esineen on oltava tasainen pinta. Tämä rajaa luonnollisesti pois myös itsevarjostuvuuden.



$$\mathbf{M} = \begin{pmatrix} \alpha_x n_{xx} & \alpha_x n_{xy} & \alpha_x n_{xz} & -\alpha_x \mathbf{n}_x \cdot \mathbf{b} \\ \alpha_y n_{yx} & \alpha_y n_{yy} & \alpha_y n_{yz} & -\alpha_y \mathbf{n}_y \cdot \mathbf{b} \\ 0 & 0 & 0 & 1 \\ \alpha_w n_{wx} & \alpha_w n_{wy} & \alpha_w n_{wz} & -\alpha_w \mathbf{n}_w \cdot \mathbf{a} \end{pmatrix}$$

$$\mathbf{e}_w = \mathbf{b} - \mathbf{a}$$

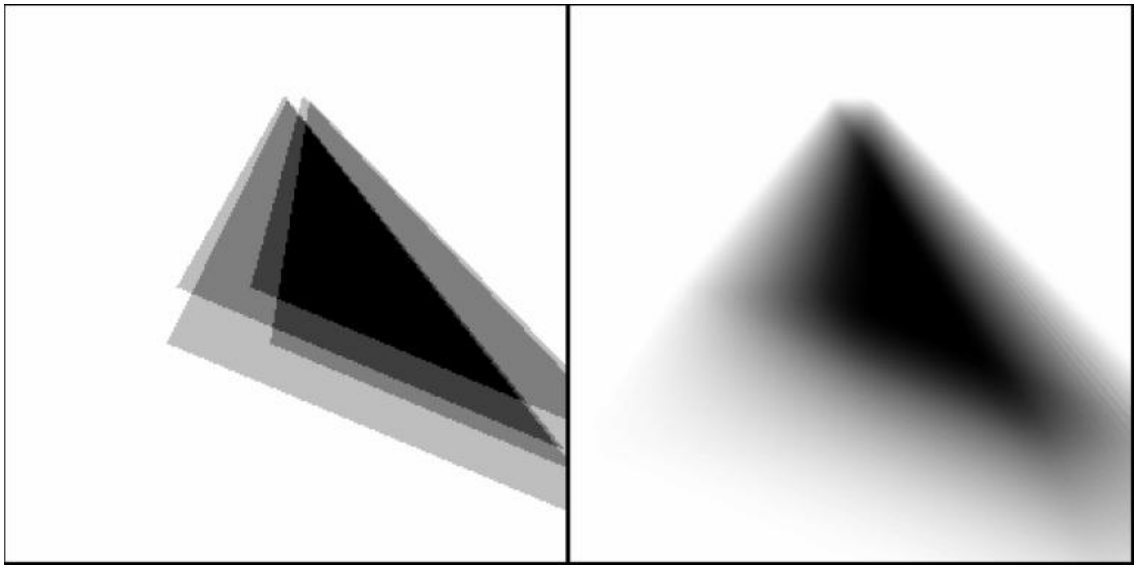
$$\mathbf{n}_x = \mathbf{e}_w \times \mathbf{e}_y \qquad \alpha_x = 1/\mathbf{n}_x \cdot \mathbf{e}_x$$

$$\mathbf{n}_y = \mathbf{e}_x \times \mathbf{e}_w \qquad \alpha_y = 1/\mathbf{n}_y \cdot \mathbf{e}_y$$

$$\mathbf{n}_w = \mathbf{e}_y \times \mathbf{e}_x \qquad \alpha_w = 1/\mathbf{n}_w \cdot \mathbf{e}_w$$

Kuva 33. Mallin projisointi tasolle [Heckbert and Herf, 1997]

Menetelmä toimii siten, että kullekin näytepisteelle määritellään samankokoinen varjokartta (kuten aiemmin mainittiin, samaakin karttaa voi käyttää), ja nollataan kaikkien pikselien arvot. Tämän jälkeen piirretään näytepisteen näkökenttää käyttäen kaikki ennalta valitut varjostavat esineet siten, että kaikki piirrettävät pikselit asetetaan varjokarttaan ykkösiksi. Nyt se voidaan sulauttaa keräyspuskuriin käyttämällä yhteenlaskuoperaatiota. Kun kaikki näytepisteet on käsitelty, keräyspuskuri sisältää tiedon siitä, kuinka monta näytepistettä kustakin varjokartan pikselistä on näkymättömissä, eli varjostavan esineen takana. Arvot saadaan välille [0,1] jakamalla pikseleiden arvot näytepisteiden määrällä. Syntynyt varjostusarvokartta voidaan piirtää varjostettavan nelikulmion päälle kuten mikä tahansa tekstikartta, ja staattisten näkymien tapauksessa se voidaan tallentaa myöhempää käyttöä varten.



Kuva 34. Pehmeä varjo 4:stä (vasemmalla) ja 256:sta (oikealla) näytepisteestä estimoituna [Heckbert and Herf, 1997].

Esilasketut varjot ovatkin menetelmän yleisin käyttötarkoitus, sillä piirtokertojen määrä on sama kuin näytepisteiden määrä kerrottuna varjostettavien pintojen määrällä. Se on normaaleilla laitteilla auttamatta liian suuri reaaliaikaiseen varjojen luomiseen, sillä näytepisteitä tarvitaan yleensä suhteellisen suuri määrä ennen kuin lopputulos näyttää oikeasti pehmeältä varjolta (kuva 34). Toisaalta menetelmä sopii hyvin moniajoon, koska eri näytepisteet voidaan projisoida ja lisätä varjostuskarttaan toisistaan riippumatta [Isard et al., 2002]. Tällöin menetelmällä voidaan päästä jopa reaaliaikaisuuden vaatimaan tehoon, mikäli prosesseja pystytään ajamaan rinnakkain riittävän monta [Hasenfrantz et al., 2003]. Uusimmat näytönohjaimet tukevat useamman eri näkymän piirtämistä samaan aikaan, mikä on tämän menetelmän kannalta erinomaisen hyödyllinen toiminto.

Agrawala et al. [2000] kehittivät menetelmästä itsevarjostavan, jolloin se ei vaadi varjostavien ja varjostettavien esineiden erottamista toisistaan. Näin ollen se soveltuu varsinkin suurten valonlähteiden käsittelyyn huomattavasti edellä mainittua menetelmää paremmin. Menetelmä perustuu edelleen valon näytepisteiden valintaan. Ne valitaan samalla tavalla kuin Heckbertin ja Herfinkin menetelmissä, mutta kaikkien katsesuunnaksi asetetaan valon normaalivektori (edellisessä katsesuunnat osoittivat aina varjostettavaa nelikulmiota kohti, ja näin ollen ne vaihtelivat eri näytepisteissä). Kun tämä on tehty, jokaisesta näytepisteestä piirretään perinteinen syvyyskartta.

Lopulliset varjostusarvot lasketaan kerroksittaiseen varjokarttaan, jonka kunkin tason pikselit sisältävät paitsi syvyysarvon, myös laskurin. Yhdistäminen tehdään kohdistamalla näytepisteiden syvyyskartat ensin

keskenään valon keskipisteen suhteen, ja käsittelemällä niiden jokainen pikseli s seuraavalla tavalla. Aluksi etsitään vastaava pikseli kerroksittaisen varjokartan ensimmäiseltä tasolta, ja verrataan karttojen syvyysarvoja keskenään. Jos s_z on sama kuin kerroksen vastaavan pisteen syvyysarvo, pikselin laskuria lisätään yhdellä. Muussa tapauksessa siirrytään seuraavalle tasolle ja toistetaan tarkastelu siellä. Jos syvyyttä ei löydy miltään tasolta, luodaan uusi taso, jonka syvyysarvoksi asetetaan s_z, ja laskuriksi alustetaan 1. Kun käsittely on tehty kaikkien valonäytteiden kartoille, kerroksittaisen syvyyskartan laskurit kertovat näkyvien näytteiden määrän varjokartan pisteen kullakin syvyystasolla. Käsittelyn päätteeksi kaikki laskurit normalisoidaan jakamalla ne näytepisteiden lukumäärällä. Jokainen laskurin arvo kertoo nyt valaistusarvon kunkin varjokartan pikselin tietyllä tasolla (huomaa, että edellisessä menetelmässä laskettiin valaistusarvon sijaan varjostusarvo).

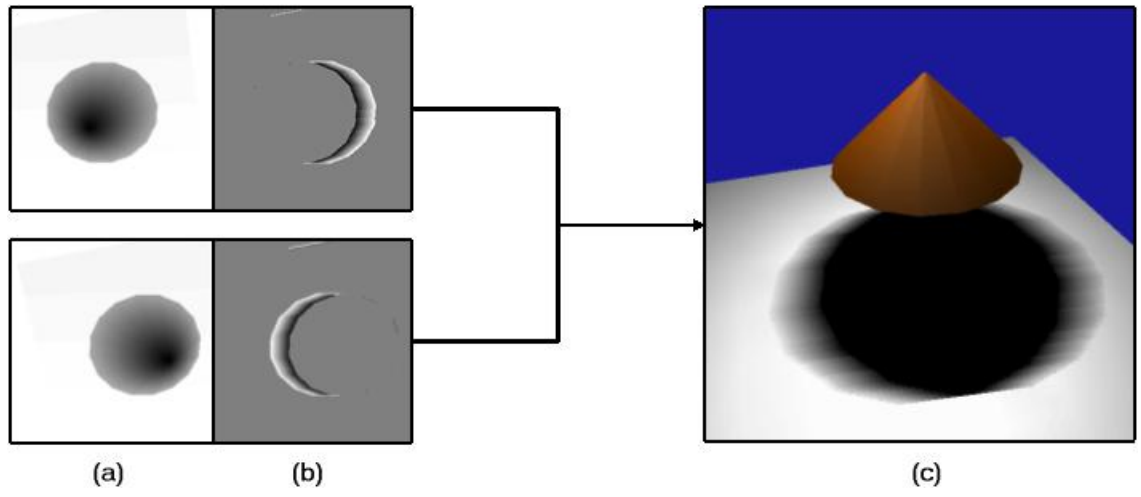
Näkymä varjostetaan muuntamalla piirrettävä piste normaalisti valoavaruuteen. Tämän jälkeen pisteen syvyyttä verrataan varjokartan syvyysarvoihin aloittaen ensimmäisestä tasosta (joka kuvaa lähimpänä valoa olevat pinnat). Jos muunnetun pisteen ja tason pikselin syvyysarvot ovat samat, piste varjostetaan saadun valaistusarvon mukaisesti. Mikäli mikään taso ei vastaa muunnetun pisteen syvyysarvoa, piste on kokonaan varjossa.

Agrawalan et al. [2000] menetelmä vaatii edelleen näkymän piirtämistä jokaiselle näytepisteelle erikseen, eli siitä ei ole yleiskäyttöiseksi reaaliaika-algoritmiksi. Se on kuitenkin huomattavasti nopeampi ja yleiskäyttöisempi kuin Heckbertin ja Herfin [1997] menetelmä, koska nyt koko näkymä voidaan varjostaa yhdellä kertaa. Muistivaatimuksetkaan eivät ole kohtuuttomat, sillä ainakaan kehittäjien mukaan varjokartan tasoja ei tarvita kovin montaa, ja vuoden 2000 jälkeen näytönohjainten muistimäärät ovat kasvaneet räjähdysmäisesti. Myös tämä menetelmä voi hyödyntää moniajoa, joskin synkronointi tasojen luomisen suhteen muuttuu hieman edellistä menetelmää monimutkaisemmaksi. Kehittäjiensä mukaan menetelmä voi lisäksi kärsiä epätarkkuusongelmista varjokarttojen kohdistamisen yhteydessä. Myös syvyysvertailuissa on muistettava käyttää jonkinlaista virhemarginaalia, koska syvyysarvot eivät pyöristysepätarkkuuksista johtuen ole yleensä täysin samoja, vaikka niiden pitäisikin olla.

Koska monien näytepisteiden käsitteleminen on hidasta, käyttivät Heidrich et al. [2000] toisenlaista lähestymistapaa. Heidän menetelmänsä käyttää ainoastaan kahta näytepistettä, ja pyrkii arvioimaan alueiden, joissa vain toinen piste on näkyvissä, varjostusta lineaarisesti. Menetelmä aloitetaan edellisen tapaan säätämällä valonlähteet samansuuntaisiksi kohtisuoraan niiden väliin

jäävää vektoria vasten, ja laskemalla tavallinen syvyyskartta molemmille pisteille (kuva 35a). Tämän jälkeen molemmista syvyyskartoista erotetaan siluettireunat jotain kuva-analyysimenetelmää käyttäen. Kummankin varjokartan rinnalle luodaan erilliset näkyvyyskanavapuskurit (visibility channel), jotka sisältävät näkyvyysarvoja välillä 0-1.

Vasemmanpuoleinen syvyyskartta käsitellään siirtämällä se kuvainnollisesti oikean päälle. Koska valonlähteet ovat olleet eri paikoissa, globaalissa koordinaatistossa samat ääriviivat (jotka nyt on siis erotettu kuvasta) ovat piirtyneet karttoihin eri kohtiin. Ääriviivat yhdistetään näkyvyyskanavassa täyttämällä niiden väliin jäävä alue esimerkiksi Gouraud-varjostetuilla monikulmioilla siten, että näkyvissä oleva reuna saa arvon 1 ja näkymättömissä oleva reuna arvon 0 (kuva 35b). Tämän jälkeen näkyvyyskartta sisältää lineaarisen siirtymän ääriviivalta toiselle. Se siis estimoi vasemman varjokartan pikselien näkyvyyttä oikeanpuoleisesta näytepisteestä. Sama käsittely toistetaan käänteisesti myös oikealle syvyyskartalle, ja näin molemmat näkyvyyskanavat on laskettu.



Kuva 35. Lineaarisen valonlähteen näytepisteiden syvyyskartat (a), näkyvyyskanavat (b) sekä lopullinen varjo (c) [Heidrich et al., 2000].

Piirtovaiheessa piste p siirretään molempien näytepisteiden avaruuteen, ja etsitään syvyyskartoista sitä vastaavat pikselit. Jos piste näkyy molemmista valoista, se on täysin valaistu, eikä näkyvyyskanavaa tarvita. Sama toistuu käänteisesti myös täydellisen varjostumisen tapauksessa. Jos piste on sen sijaan näkyvissä vain toisesta näytepisteestä, saadaan lopullinen valaistusarvo selvitetyksi laskemalla näkyvyyskanavien vastaavien pikseleiden arvot yhteen.

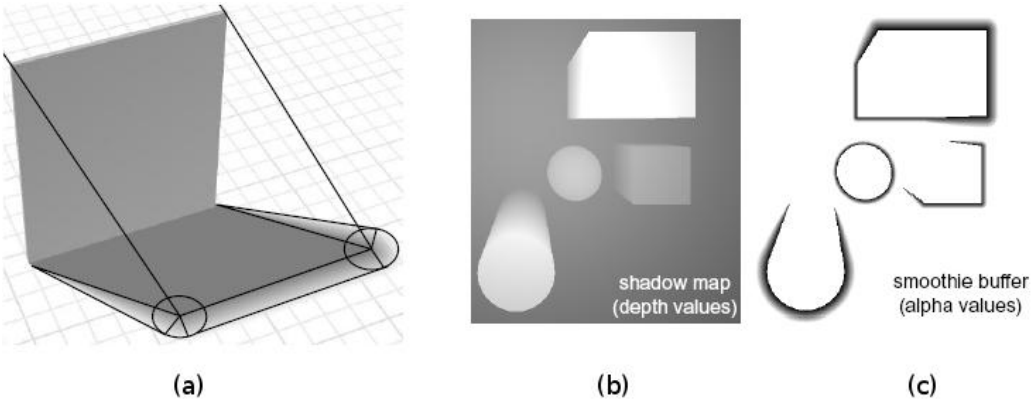
Menetelmä on siinä mielessä edellisiä tehokkaampi, ettei sen tarvitse luoda kuin kaksi syvyyskarttaa. Vastaavasti kuva-analyysiin käytettävä aika hidastaa menetelmän suorittamista, eikä tee siitäkään riittävän tehokasta dynaamisten varjojen käsittelyyn [Hasenfrantz et al., 2003]. Varsinkin monimutkaiset

näkymät ja mallit tuottavat menetelmälle vaikeuksia. Alkuperäisen menetelmän ongelma on myös sen rajoittuminen ainoastaan kahteen näytepisteeseen, jolloin tulee helposti vastaan tilanteita, joissa näytepisteiden katvealueeseen jäävät esineet eivät varjostu oikein [Heidrich et al., 2000]. Menetelmää on sittemmin laajennettu käyttämään myös monikulmion muotoisia valonlähteitä [Ying et al., 2002]. Tällöin näkyvyyskanava lasketaan monikulmion pisteitä yhdistäville reunoille, ja näkyvyysarvo lasketaan valomonikulmion alan suhteen. Useampien näytepisteiden käyttö kuitenkin myös hidastaa menetelmää samassa suhteessa.

6.2. Puolivarjon laskeminen pistevalosta

Useiden näytepisteiden käsitteleminen siis hidastaa ohjelman suorittamista niin paljon, ettei edellisiä menetelmiä voida käyttää dynaamisten varjojen luomiseen. Näin ollen onkin kehitetty menetelmiä, jotka pystyvät arvioimaan pehmeän varjoalueen käyttämällä ainoastaan yhtä pistemäistä valonlähdettä. Menetelmiä yhdistää se, etteivät ne pyri luomaan fysikaalisesti oikeaoppisia varjoja, vaan varjot ovat periaatteellisellakin tasolla aina virheellisiä. Täten tavoite onkin pyrkiä tekemään varjosta sellainen, että virheet ovat mahdollisimman vaikeasti havaittavissa. Haines [2001], Wyman ja Hansen [2003] sekä Chan ja Durand [2003] käyttävät puolivarjon estimointiin hybridimenetelmiä, joissa varjostuskartta luodaan varjomallin ja näkymän ylle projisoitavien varjokarttojen avulla.

Hainesin [2001] tasannemalli (plateaus) projisoi pehmeän varjostuskartan ennalta määritetylle tasaiselle pinnalle. Ideana on muodostaa varjostavasta esineestä varjotilavuusmenetelmän mukaisesti siluetti, jonka jokaista pistettä kohden luodaan kalteva kartio, jonka kärki on siluettipisteessä, ja alareuna venytetään valosta katsottuna varjostettavaa pintaa vasten (kuva 36a). Kamera asetetaan osoittamaan varjostettavaan tasoon kohtisuorassa, ja tämän jälkeen näkymään piirretään varjostettavan mallin pohjalta luotu perinteinen varjotilavuus. Syvyystarkastus säädetään siten, että seuraavat piirto-operaatiot jäävät aina varjomallin taakse. Tämän jälkeen piirretään kartiot sekä niitä yhdistävät monikulmiot siten, että reunat ovat vaaleita ja sisäalueet tummia. Tuloksena kovan varjotilavuuden ympärille syntyy asteittaisesti himmenevä alue. Varjon vaalenemista voidaan säätää erilaisia tekstikarttoja käyttämällä. Wymanin ja Hansenin [2001] esittelemän tehostuksen jälkeen menetelmä kykenee myös reaaliaikaiseen varjolaskentaan, ja kartioiden koko riippuu myös valonlähteen etäisyydestä.



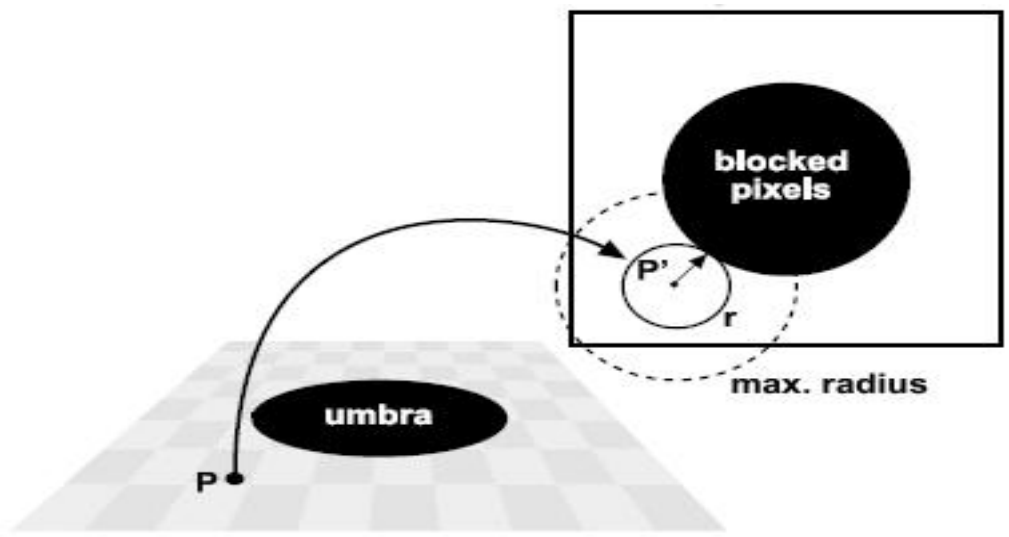
Kuva 36. Tasannemalli (a) [Haines, 2001] sekä syvyyskartta (b) ja sitä vastaavat pehmentimet (c) [Chan and Durand, 2003].

Kumpikaan edellisen menetelmän muunnelma ei tue itsevarjostuvuutta ja rajoittuu tasapintaiseen varjostettavaan esineeseen. Chan ja Durand [2003] kääntävät edellisten menetelmien ajatuksen toisinpäin, ja käyttävät ylimääräisiä monikulmioita laajentamaan varjokarttaan piirrettyä kovaa varjoa. Aluksi piirretään näkymästä tavallinen varjokartta. Tämän jälkeen varjostavan mallin siluettireunoihin kiinnitetään ylimääräiset nelikulmiot, sumentimet (smoothies), joista kukin on kohtisuorassa reunan normaaliin nähden. Sumentimet on päällystetty varjostavan esineen reunoilta poispäin vaalenevilla tekstuureilla, joiden arvot määrittävät sen, millainen puolivarjoalue syntyy. Käytännössä tekstuuri piirretään valon koon ja etäisyyden mukaan. Varjon piirtäminen tapahtuu kahdessa vaiheessa. Ensin piirretään kova varjo syvyyskarttaa käyttäen (kuva 36b), ja sen jälkeen piirretään sumentimet valon näkökulmasta toiseen tekstikarttaan (kuva 36c), joka lopuksi projisoidaan kameranäkymään.

Sumentimien käyttö on erittäin nopeaa, ja itsevarjostuvuuden ansiosta niiden käyttö sopii hyvin reaaliaikaiseen grafiikkaan. Se jakaa kuitenkin kahden aiemmin esitellyn menetelmän tapaan merkittävän ongelman: se ei mahdollista kuin ulomman puolivarjon laskemisen. Tämä voidaan antaa anteeksi pienillä valonlähteillä, mutta kookkaat täysvarjot tekevät näkymästä tummemman kuin sen pitäisi oikeasti olla [Hasenfrantz et al., 2003]. Ei siis riitä, että tummennetaan ulompaa puolivarjoa, vaan on lisäksi vaalennettava kovaan varjoon jäävää sisempää puolivarjoa.

Brabec ja Seidel [2002] ratkaisivat ongelman puhtaan bittikarttamenetelmän avulla. Pohjana he käyttivät Parkerin et al. [1998] ajatusta, joka on puhtaasti kolmessa ulottuvuudessa laskettava, niin ikään vain ulomman puolivarjon laskeva menetelmä, jossa varjostusaste määräytyy sen mukaan, kuinka läheltä varjostavaa esinettä valonsäde kulkee. Brabec ja Seidel [2002] siirsivät

tarkastelun tehtäväksi syvyyskartassa, jolloin on mahdollista laskea myös sisempi puolivarjo. Menetelmä toimii siten, että mikäli muunnettu piste p on varjossa, lasketaan etäisyys lähimpään valossa olevaan varjokartan pikseliin, ja lisätään p:n valaistusarvoa sen mukaisesti. Sama tehdään toisinpäin, mikäli piste p on valossa. Tällöin valaistusarvoa vähennetään suhteessa lähimpään varjossa olevaan pikseliin.



Kuva 37. Pistevarjon laajentamisen periaate [Brabec and Seidel, 2002].

Puolivarjolle asetetaan maksimikoko r_{max}, joka samalla myös rajoittaa etsintään käytettävän alueen. r_{max} määräytyy varjostettavan pinnan etäisyydellä valosta. Pisteen etäisyyttä varjon reunasta merkitään vastedes kirjaimella r. Valaistusaste lasketaan kaavalla, joka ottaa huomioon valon, varjostavan ja varjostettavan esineen etäisyydet toisistaan. Hasenfrantz et al. [2003] määrittelevät kaavan seuraavalla tavalla (R ja S ovat vapaasti määriteltäviä vakioita):

$$f = \frac{dist(p_{varjostaja}, p_{varjostettava})}{RS\, z_{varjostettava} \left| z_{varjostettava} - z_{varjostaja} \right|}.$$

Menetelmässä luodaan kaksi karttaa: tavallinen syvyyskartta sekä tunnistekartta [Hourace and Nicholas, 1985], jonka pikseleihin tallennetaan yksilöllisen tunnisteen avulla tieto siitä, mihin esineeseen syvyysarvo kuuluu. Tunnistekartta auttaa varjon reunan löytämisessä, koska sen avulla voidaan välttää pehmeiden varjojen vääränlainen itsevarjostuminen [Brabec ja Seidel, 2002]. Samalla se valitettavasti synnyttää myös menetelmän pahimman ongelman, eli itsevarjostuvuuden puuttumisen. Kun p ja siihen liittyvä tunniste on muunnettu valoavaruuteen, syvyys ja esinetunniste haetaan varjokartasta tavalliseen tapaan. Jos muunnettu piste on kauempana kuin syvyyskartan vastaava arvo, se on kovassa varjossa, ja voi kuulua vain sisempään

puolivarjoon. Muussa tapauksessa se voi olla osa ulompaa puolivarjoa. Puolivarjon koko lasketaan kaavalla $r_{max} = R * z_{varjostettava}$, missä R on vapaavalintainen parametri [Hasenfrantz et al., 2003].

Tämän jälkeen seuraa menetelmän raskain osa, missä r_{max}:n alueelta etsitään lähin varjossa tai valossa oleva pikseli. Mikäli p on kovan varjon sisällä, etsintä päättyy kun on löydetty lähin piste, jonka syvyysarvo on suurempi tai yhtäsuuri kuin p_z, ja pikseliin tallennettu esinetunniste on sama kuin kameranäkymästä välitetty. Vastaavasti p:n ollessa valossa on löydettävä piste, jonka syvyysarvo on pienempi, ja jonka esinetunniste on jokin muu kuin p:n mukana saapunut. Kun sopiva piste on löytynyt, saadaan p:n valaistusarvo laskettua aiemmin esitellyllä kaavalla. Sisävarjossa olevan pisteen varjostustaso on 0.5(1-f), ja muulloin 0.5(1+f). Mikäli etäisyydeltä r_{max} ei löytynyt ehdot täyttävää pistettä, varjostustaso on joko 0 (p ei ollut kovassa varjossa) tai 1.

Varjoreunan etäisyyden laskeminen on menetelmän tehokkuuden kannalta tärkein tekijä. Vaikka spiraalimaisesti etenevä etsiminen tuntuisikin loogisimmalta vaihtoehdolta, Brabec ja Seidel [2002] ovat havainneet lineaarisen, kaikki etsintäalueen rajaaman neliön pikselit tarkastavan menetelmän olevan tehokkain. Tämä johtuu lähinnä näytönohjainten tavasta käsitellä muistia. Lisäksi he esittelevät hieman binäärihakua muistuttavan etsintäalgoritmin, jonka kaltaisen hyödyntäminen on reaaliaikaisuuden kannalta välttämätöntä. Myös r_{max}:in koolla on luonnollisesti suuri vaikutus menetelmän nopeuteen. Kirsch ja Doellner [2003] ovat sittemmin tehneet menetelmästä tehokkaamman käyttämällä erillistä tekstikarttaa, jonka pikseleihin esilasketaan kunkin pisteen etäisyys lähimmästä valaistusta pisteestä. Tämä menetelmä kuitenkin mahdollistaa vain sisemmän puolivarjon muodostamisen.

Itsevarjostuvuuden puuttumisen lisäksi menetelmää vaivaa sama ongelma kuin kaikkia muitakin äärettömän pientä valonlähdettä käyttäviä algoritmeja. Joitakin äärimmäisen yksinkertaisia malleja lukuun ottamatta, esineen siluetti muuttuu näkökulmasta riippuen. Tämä koskee myös valon näytepisteitä, ja mikäli pehmeä valo luodaan vain yhtä pistettä käyttäen, varjon muoto ei valoalueen reunoilta katsottuna ole oikea. Tämä on ongelma myös takaisinprojisoinnin kanssa

6.3. Takaisinprojisointi

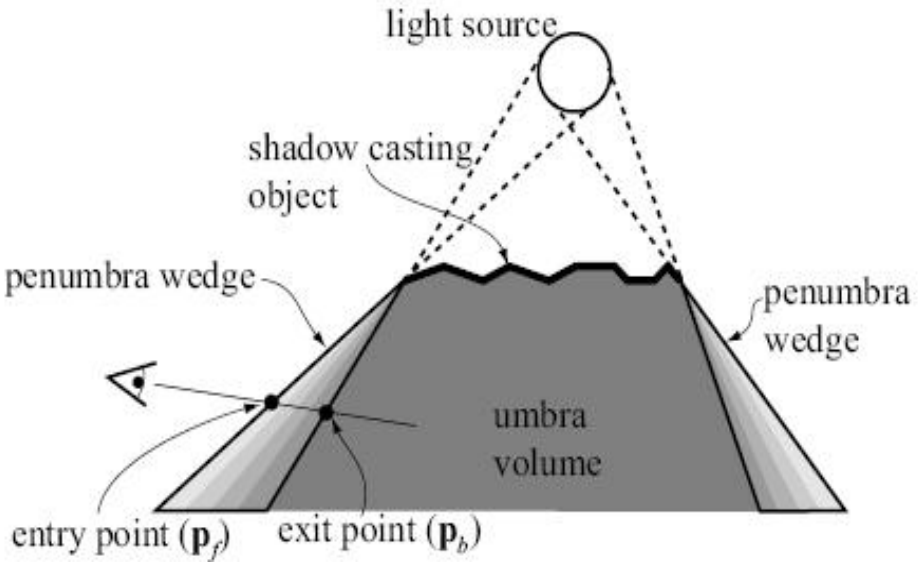
Useat tahot ovat havainneet takaisinprojisoinnin (backprojection) [Drettakis and Fiume, 1994] ratkaisuksi moniin aikaisempien algoritmien ongelmiin. Näin se on noussut viime vuosina merkittäväksi tutkimuskohteeksi reaaliaikaisen grafiikan yhteydessä. Takaisinprojisoinnin ajatus perustuu siihen, että varjostettavan pisteen ja tilallisen valonlähteen välille luodaan näkökenttä,

johon varjostava geometria projisoidaan. Tämän avulla voidaan yleispätevästi selvittää, kuinka suuri osa valonlähteestä on kustakin pisteestä katsottuna peitossa, mikä takaa myös fysikaalisesti oikeaoppisen varjostuksen. Menetelmä muistuttaa jossain määrin ZP+-algoritmia (ks. luku 4.1.2), missä peitettiin alueita kameran näkökentästä valonlähteestä katsottuna. Nyt sama tehdään varjostavasta pisteestä valoa vasten.

Varjostavien mallien projisointi jokaista pistettä kohti on luonnollisesti liian hidasta, joten lähestymistapaa on jouduttu yksinkertaistamaan. Tästä huolimatta takaisinprojisointi vaatii monimutkaisia pikselikohtaisia operaatioita, eikä ole täten ollut varteenotettava vaihtoehto ennen ohjelmoitavien näytönohjaimien tuloa markkinoille. Takaisinprojisointi-termiä käytetään yleensä nimenomaan varjokarttojen yhteydessä, mutta samaa perusajatusta sovellettiin varjotilavuuksiin jo paljon aikaisemmin.

6.3.1. Puolivarjokiilat

Akenine-Möller ja Assarsson [2002] laajentavat perinteistä varjotilavuusmenetelmää luomalla varjomallin reunalle kiilat (penumbra wedge), jotka kapseloivat sisäänsä sekä sisemmän että ulomman puolivarjon (kuva 38). Nämä mallit luodaan saman ajatuksen mukaan kuin kovatkin varjot, eli venyttämällä siluettireunaa valosta poispäin. Tämä venytys tehdään kuitenkin valon molemmilta puolilta, jolloin tuloksena on kaksi venytettyä nelikulmiota, joista toinen erottaa ulomman puolivarjon ja valon, ja toinen sisemmän puolivarjon ja täysvarjon. Eli kun valonlähteen koko lähestyy nollaa, puolivarjokiilat lähestyvät tavallista kovaa varjoa. Kiiloja luotaessa otetaan siis automaattisesti huomioon kaikki pehmeiden varjojen keskeiset elementit: esineiden ja valon etäisyydet toisistaan, sekä valon koko.



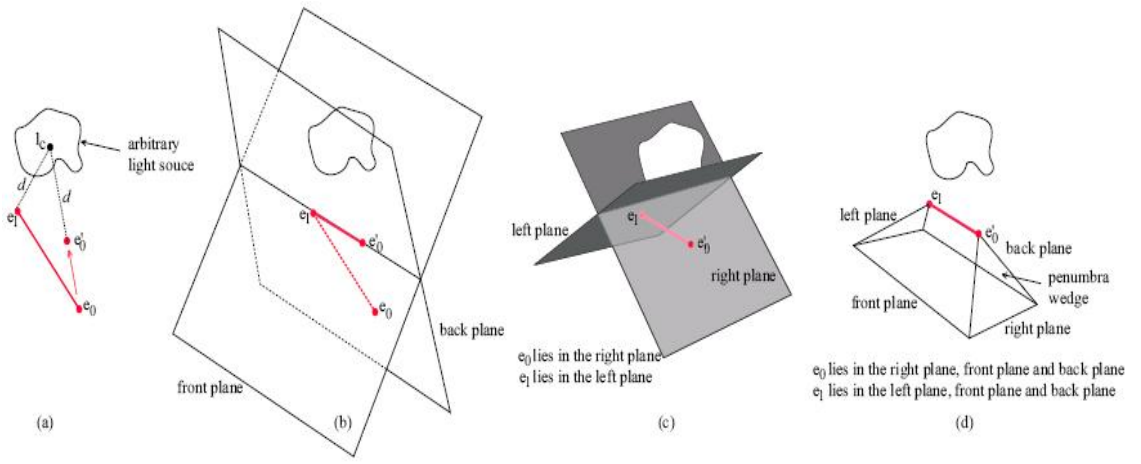
Kuva 38. Puolivarjokiilojen ajatus kaksiulotteisesti esitettynä [Akenine-Möller and Assarsson, 2002].

Varjokiilan luontia on helpointa havainnollistaa käyttämällä valonlähteenä palloa (tämä on myös Akenine-Möllerin ja Assarssonin [2002] oletus). Pistevalo, josta kovat varjot luodaan, sijaitsee pallon keskipisteessä, ja kiilojen reunoja venytetään pallon tangenttien ja siluettireunan suuntaisesti. Hieman Brabecin ja Seidelin [2002] tapaan, tila jaetaan osiin varjostustavan suhteen. Täysvarjossa oleva piste (joka on kovassa varjossa mutta ei kummassakaan puolivarjossa) mustataan lopullisesta kuvasta. Sisemmässä puolivarjossa olevan pisteen valaistustasoa nostetaan nollasta kovan varjon reunoja lähestyttäessä, ja ulommassa puolivarjossa valaistusta vastaavasti vähennetään. Varjojen ulkopuolella oleva piste jätetään varjostamatta. Kuten muissakin varjotilavuusmenetelmissä, valaistusasteet lasketaan kameranäkymässä.

Menetelmän ensimmäisessä versiossa [Akenine-Möller and Assarsson, 2002] valonlähde oli siis pallon muotoinen. Kiiloja luotaessa määritettiin ensin kaksi pistettä: $b = c + rn$ ja $f = x - rn$, missä c on valon keskipiste, r on sen säde, ja n on kovan varjoreunan normaali. Ulompi puolivarjoreuna venyttiin käyttämällä valonlähteenä pistettä f, sisempi puolestaan rakennettiin pisteen b kautta. Tämän lisäksi tarvittiin ylimääräinen reunakolmio sulkemaan kiilojen väliset reunat. Tämä menetelmä aiheutti lukuisia ongelmia [Assarsson and Akenine-Möller, 2003]. Ensinnäkin, mikäli reunan kärkipisteet olivat eri etäisyydellä valosta, tuloksena oli hyperbolinen kiila, jonka muuttaminen varjomalliksi on epäluontevaa. Laajan valonlähteen tapauksessa puolivarjot saattoivat lisäksi kohdata varjostavan esineen takana, jolloin ne piti leikata toisiaan vasten. Lähes valonsuuntaiset reunat aiheuttivat sisäkkäisiä puolivarjoja, jotka sotkivat varjostusta, ja kun tähän vielä lisättiin menetelmän kykenemättömyys huolehtia useampaan kuin kahteen muuhun ääriviivaan liittyvistä siluettireunoista, oli menetelmään tehtävä parannuksia.

Puolivarjokiilojen ensimmäinen versio ei käyttänyt takaisinprojisointia, vaan varjostusaste määritettiin laskemalla varjostettavan pisteen sijainti suhteessa kiilan reunoihin. Tasaisena jatkuvan varjostuksen aikaansaaminen tällä menetelmällä vaati, että reunakiilat olivat tietoisia toistensa ominaisuuksista [Akenine-Möller and Assarsson, 2002]. Takaisinprojisoinnissa tällä tiedolla ei kuitenkaan ole merkitystä, joten Assarsson ja Akenine-Möller [2003] ratkaisivat (takaisinprojisointiin siirtyessään) ongelmat muuttamalla reunakiilojen muodostusta siten, että ne voitiin luoda toisistaan riippumatta. Menetelmän avulla kiilan molemmat reunat saadaan lisäksi avautumaan tasaisesti, jolloin hyperbolisilta kiiloilta vältytään.

Oletetaan, että muodostetaan kiila reunasta, jonka kärkipisteet ovat e_0 ja e_1, ja että e_1 on lähempänä valon keskipistettä. Luodaan piste e'_0, joka saadaan liikuttamalla e_0:aa valon suuntavektorin vastakkaiseen suuntaan, kunnes se on yhtä kaukana valosta kuin e_1 (kuva 39a). Reunaa vastaava kiila muodostetaan nyt pisteiden e'_0 ja e_1 avulla. Ensin luodaan etu- ja takareuna muodostamalla kaksi tasoa, jotka sisältävät valon keskipisteet sekä pisteet e'_0 ja e_1. Molempia tasoja kierretään reunan ympäri vastakkaisiin suuntiin siten, että ne sivuavat valonlähdettä sen eri puolilta (kuva 39b). Sivureunat muodostetaan hieman samaan tapaan. Aluksi luodaan tasot, joista toinen (oikea sivureuna) kulkee e'0:n kautta ja toinen (vasen sivureuna) e_1:n kautta. Molempien tasojen normaali asetetaan osoittamaan reunan suuntaisesti varjostettavasta mallista poispäin. Tämän jälkeen sivutasoja käännetään siten, että nekin sivuavat valoa päinvastaisilta puolilta (kuva 39c). Kiilan lopulliset monikulmiot saadaan muodostettua käyttämällä hyväksi reunan pisteitä ja tasojen normaaleita (kuva 39d). Jos käytetään zfail-menetelmää, on kiilan alareuna muistettava vielä sulkea ylimääräisellä suorakulmiolla. Reunan toisen kärkipisteen siirtäminen valoa kohti suurentaa kiilaa, ja takaa varsinaisen puolivarjon mahtuvan sen sisään. Kiilaan kuuluva piste ei välttämättä kuulu puolivarjoon, mutta tämä ei ole ongelma lopullista varjostusarvoa määritettäessä. Kiiloja on mahdollista pienentää, jos valo on suorakulmion muotoinen. Tällöin kiila voidaan tehdä hyvin yksinkertaisesti heijastamalla valonlähde reunan kautta äärettömyyteen [Assarsson et al., 2003]. Tällöin myös alareunan sulkeminen muuttuu triviaaliksi.



Kuva 39. Reunakiilojen muodostaminen [Assarsson and Akenine-Möller, 2003].

Kiilojen piirtäminen tapahtuu Assarssonin et al. [2003] ohjeiden mukaisesti (esimerkissä käytetään zpass-menetelmää, mutta zfailia voidaan soveltaa kääntämällä operaatiot päinvastaisiksi, kuten luvussa 4.1.3 on kuvattu). Aluksi kameranäkymä piirretään väri- ja syvyyspuskuriin pitämällä käsiteltävä valo

päällä. Tämän jälkeen näkyvyyskartan pikselit asetetaan arvoon 1.0. Kovat varjot piirretään muuten tavalliseen tapaan, mutta kameraan päin osoittavien varjopintojen kohdalla näkyvyyspuskurin arvoa vähennetään yhdellä, ja poispäin osoittavan kohdalla sitä lisätään yhdellä. Nyt näkyvyyskartan nolla-arvot osoittavat kovan varjon sijainnin. Menetelmässä on mahdollista käyttää myös sapluunapuskuria, ja kopioida se sopivasti kerrottuna näkyvyyspuskuriin (mikäli laitteisto sen sallii) [Lengyel, 2005]. Tällöin voidaan hyödyntää laitteistokohtaisia optimointeja sapluunapuskureille.

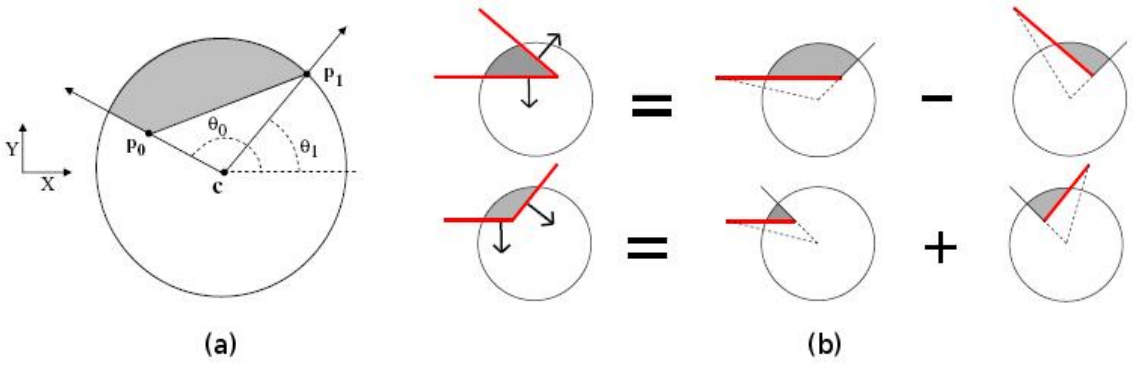
Seuraavaksi piirretään puolivarjokiilat yksi kerrallaan. Koska puolivarjon vaalennus ja himmennys vaatii melko raskaita pikselikohtaisia toimintoja, käytetään sapluunapuskuria rajaamaan pois sellaiset alueet, joissa puolivarjo ei ole näkyvä [Assarsson et al., 2003]. Ensin piirretään kiilojen etureunat ainoastaan sapluunapuskuriin, asettamalla pikselin arvoksi 1 aina syvyystestin onnistuessa. Tämän jälkeen asetetaan näkyvyyspuskuriin kirjoittaminen päälle (tämä aktivoi myös pikselivarjostimet), ja piirretään kiilojen takareunat. Piste piirretään vain, jos sen syvyysarvo on suurempi kuin aiemmin syvyyspuskuriin kirjoitettu arvo, ja sapluunapuskurin arvo on 1. Kukin puolivarjo on piirrettävä erikseen, koska muuten limittäin menevät sisemmät ja ulommat puolivarjoalueet sotkevat toistensa varjostuksen (Fauerby ja Kjær [2003] antavat tästä seikkaperäisen selvityksen). Kullekin piirrettävälle pisteelle ajetaan pikselivarjostin, joka tummentaa ulomman puolivarjon aluetta ja vaalentaa sisempää. Tämän jälkeen näkyvyyskartta voidaan sulauttaa alussa piirrettyyn väripuskuriin.

Vähennettävän tai lisättävän varjostusarvon laskeminen pikselivarjostimessa on menetelmän raskain vaihe. Akenine-Möller ja Assarsson [2002] laskivat varjostusarvon reunapisteiden normaalien ja kiilan etu- ja takareunan avulla. Tämä mahdollisti kuitenkin vain varjostusarvon lineaarisen approksimaation, eikä näin ollen tuottanut aina realistisen näköisiä tuloksia. Lisäksi menetelmä toimi väärin limittäisten varjokiilojen tapauksessa. Takaisinprojisointi [Assarsson and Akenine-Möller, 2003] korjaa nämä ongelmat.

Ennen kuin varjostusarvoa voidaan säätää, pikselivarjostimen on tiedettävä, pitääkö valoa vähentää vai lisätä. Tämän päättämiseen tarvitaan tieto siitä, sijaitseeko piste sisemmän vai ulomman puolivarjon alueella. Lengyel [2005] antaa pikselivarjostimelle vakiosyötteinä kiilan ja kovan varjoreunan tasojen neliulotteiset vektorit muunnettuina normalisoituun kamera-avaruuteen. Tämän jälkeen se, millä alueella piste on, voidaan laskea pikselikohtaisesti, käyttämällä pisteen kaksiulotteisia koordinaatteja ja syvyyspuskurin arvoa. Sijainnit suhteessa kuhunkin tasoon saadaan pistetulojen avulla. Assarsson et

al. [2003] kuitenkin varoittavat syvyysarvojen epätarkkuudesta ja ongelmista, joita pyöristysvirheet aiheuttavat varjostettavan pisteen ollessa lähellä kovaa varjoa. He pilkkovat puolivarjokiilat sisempään ja ulompaan osaan käyttämällä ylimääräisillä reunakolmioilla jatkettua kovaa varjoa. Tämän jälkeen puolivarjoalueet piirretään toistaan erillään, eikä tarkastusta tarvitse enää tehdä pikselikohtaisesti.

Takaisinprojisointi suoritetaan projisoimalla kaikki varjostavien mallien ääriviivat globaaliin koordinaatistoon muunnetun piirrettävän pisteen ja valonlähteen rajaamaan, normalisoituun avaruuteen. Tässä avaruudessa varjostettava piste sijaitsee origossa, ja valonlähde täyttää muotonsa puitteissa koko takakentän. Siluettireunoilla tarkoitetaan pisteistä e_0 ja e_1 koostuvia reunoja, ei niitä, joista kiilat on tehty. Näin myös oikean puolivarjoalueen ulkopuolella olevat pikselit varjostuvat oikein. Mallin peittämä alue saadaan laskettua seuraavalla tavalla [Assarsson and Akenine-Möller, 2003]. Ensin kukin reuna leikataan pisteen projisoidun näkökentän reunoja myöten, jolloin saadaan p:n näkökentässä olevat pisteet p_0 ja p_1. Tämän jälkeen lasketaan valon keskipisteen (joka on normalisoidun avaruuden takareunan keskellä) ja reunapisteiden välinen terävä kulma, joka saadaan vähentämällä atan2-funktion avulla lasketut kulmat toisistaan. Nämä funktiot ovat liian raskaita suoritettavaksi pikselikohtaisesti, joten ne suositellaan esilaskettavaksi erilliseen tekstuuriin, josta ne voidaan lukea reunan projisoitujen koordinaattien avulla [Assarsson and Akenine-Möller, 2003]. Myös kulmaa vastaava ala voidaan esilaskea, mutta varsinkin pallonmuotoisten (eli projisoituna ympyränmuotoisten) valonlähteiden kohdalla se saadaan nopeasti jakamalla radiaaneina esitetty kulma π:n arvolla. Tuloksesta on tämän jälkeen vähennettävä valon keskipisteen ja pisteiden rajaaman kolmion ala, joka saadaan kaavalla $0.5|(a-c)x(b-c)|$, missä c on valon keskipiste, ja a sekä b ovat reunan leikatut kärkipisteet. Lopputuloksena on alue, jonka mallin reuna valonlähteestä peittää. Suhteellinen varjostusaste saadaan jakamalla se valon kokonaispinta-alalla (kuva 40a).



Kuva 40. Yhden reunan varjostaminen (a) [Assarsson et al., 2003], ja reunojen yhteinen varjostus (b) [Fauerby and Kjær, 2003]. Harmaa kuvaa varjostettua aluetta.

Yhden reunan laskeminen ei luonnollisesti riitä, vaan näkökenttään on projisoitava koko siluetti. Assarsson et al. [2003] osoittavat, että jokainen reuna voidaan käsitellä erikseen, ja niiden aiheuttaman varjostuksen kokonaisvaikutus saadaan lisäämällä tai vähentämällä tulos aiemmin käsiteltyjen reunojen aiheuttamasta varjostuksesta. Reunan suunta suhteessa varjostettavaan pisteeseen saadaan selville monikulmioiden piirtosuunnan mukaisesti (vasta- tai myötäpäivään). Reunan etupuolella oleva piste on ulommassa puolivarjossa, ja täten reunan varjostama alue vähennetään pisteen kokonaisvalaistuksesta. Muussa tapauksessa piste on sisemmässä puolivarjossa, ja valaistusarvoa lisätään (kuva 40b). Koska laskenta tehdään reunakohtaisesti, pystytään käsittelemään myös tilanteet, joissa osa reunoista osoittaa pistettä kohti, ja osa poispäin.

Mikäli valonlähde on pallo tai suorakulmio, varjostusarvon laskenta on yleensä mahdollista suorittaa analyyttisin menetelmin [Assarsson et al., 2003]. Varjostusarvot voidaan kuitenkin myös esilaskea neliulotteiseen listaan, jonka kaksi ensimmäistä tasoa kuvastavat p_0:n kaksiulotteisia koordinaatteja, ja kaksi seuraavaa p_1:n vastaavia [Assarsson and Akenine-Möller, 2003]. Listan arvot ilmaisevat alan, jonka pisteistä muodostuva reuna varjostaa. Esilaskenta lisää tehokkuutta (varjostusarvojen tarkkuuden kustannuksella), ja mahdollistaa lisäksi kuvallisten valonlähteiden käyttämisen. Tämä onnistuu tallentamalla neliulotteisen tekstikartan väriarvoihin tieto siitä, kuinka paljon kutakin väriä varjostava alue peittää. Nämä arvot vähennetään sitten valaistusarvokartan (jonka on nyt tallennettava arvot kullekin värikanavalle erikseen) täysistä väriarvoista. Samalla menetelmä mahdollistaa myös tulen kaltaiset yksinkertaiset valoanimaatiot [Assarsson and Akenine-Möller, 2003].

Menetelmä tuottaa aidonnäköisiä varjoja, mutta sitä vaivaa pari hankalasti ratkaistavaa ongelmaa. Ensimmäinen on jo aiemmin mainittu, yhden valonäytteen käytöstä johtuva epätarkkuus ja toinen on se, että mikäli kahden eri mallin siluetit varjostavat samaa pistettä, varjo tummuu liikaa. Ensimmäistä ongelmaa on mahdollista korjata pilkkomalla valonlähde pienempiin osiin. Tämä hidastaa suoritusta jonkin verran, mutta ei kuinkaan samassa suhteessa kuin varjokarttojen kohdalla, sillä yksittäisten valonlähteiden kutistuessa jaon yhteydessä myös niiden langettamat puolivarjot kutistuvat, ja yhden kiilan piirtämiseen kuluva aika vähenee [Assarsson et al., 2003]. Päällekkäisten siluettireunojen aiheuttamaa ongelmaa voidaan helpottaa muodostamalla ääriviivoista yhtenäisiä ketjuja, ja piirtää jokainen ketju erikseen omaan

valaisuarvokarttaansa. Lopuksi nämä yhdistetään, jolloin päällekkäiset varjot eivät enää tummenna toisiaan liikaa [Assarsson et al., 2003]. Forest et al. [2006] korjaavat ongelmaa pilkkomalla valonlähdettä p:n avaruudessa, ja pyrkimällä kapseloimaan mahdollisia päällekkäisyyksiä. Menetelmä lienee kuitenkin liian raskas reaaliaikaiseen käyttöön [Schwarz and Stamminger, 2007].

Algoritmin tehokkuus riippuu mallin reunojen määrästä ja varjojen koosta. Lisäksi kiilat on piirrettävä erikseen, ja koska jokainen niistä vaatii monta piirto-operaatiota, ei menetelmä skaalaudu kovin hyvin monimutkaisten mallien esittämiseen. Kiilojen piirtämistä voidaan nopeuttaa mm. käyttämällä luvussa 4.3.1 esiteltyjä varjomallien optimointimenetelmiä [Lengyel, 2005]. Lisäksi Assarsson et al. [2003] esilaskevat piirrettävien pisteiden sijainnit globaalissa koordinaatistossa ylimääräiseen tekstikarttaan, koska se on heidän mukaansa tehokkaampaa kuin muunnoksen tekemisen pikselivarjostimessa (Lengyel [2005] toimii päinvastoin). Fauerby ja Kjær [2003] kehittivät rajoitetun menetelmän, jonka avulla reunakiiloja voidaan piirtää kerralla useampia. Algoritmi ei kuitenkaan ole yleispätevä vaan vaatii, että valonlähde on pyöreä, ja että siluetit ovat suljettuja ketjuja. He tehostivat myös varjostuksen laskemista pallonmuotoisille valonlähteille.

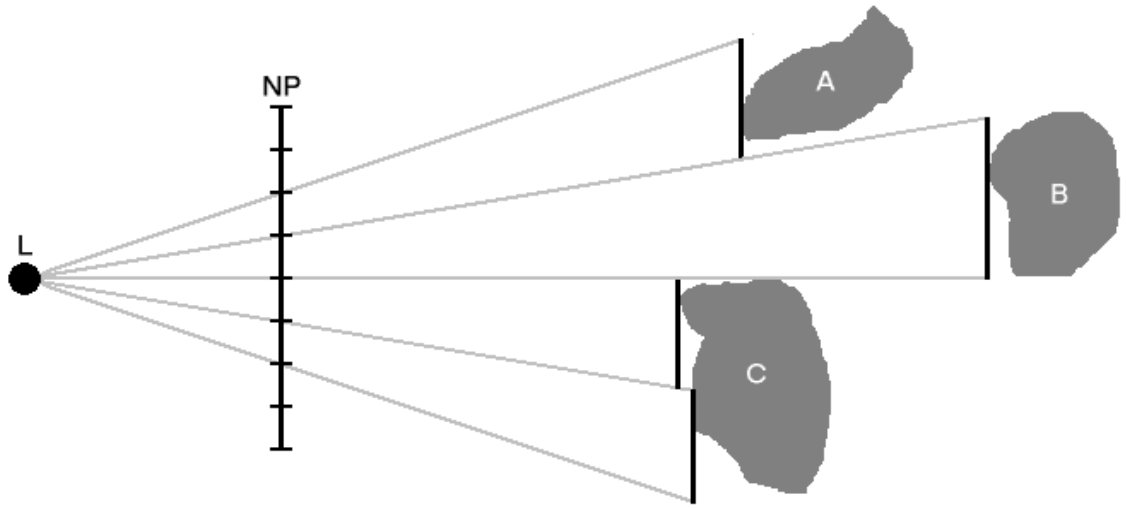
Varjokiilamenetelmän suurin ongelma kuitenkin on, ettei se tunnu kovin helposti taipuvan 3D-rajapintojen perusominaisuuksiin, vaan vaatii paljon toiminnallisuutta, joihin näytönohjaimia ei ole optimoitu. Esimerkiksi valaistuskartan käytännön toteuttamiseksi on kokeiltu monia eri menetelmiä (mm. [Assarsson et al., 2003], [Fauerby ja Kjær, 2003] ja [Lengyel, 2005]), mutta yhtäkään niistä ei voi kutsua kovin luontevaksi. Menetelmää ei viime vuosina olekaan liiemmin kehitetty, vaan huomio on keskittynyt vastaavaa varjostustapaa soveltaviin varjokartta-algoritmeihin

6.3.2. Bittikarttapeitot

Atty et al. [2006] ja Guennebaud et al. [2006] keksivät samaan aikaan toisistaan riippumatta soveltaa takaisinprojisoinnin ajatusta varjokarttoihin. Aivan kuten puolivarjokiilojen tapauksessa, kummankin ajatuksena on tehdä projektio piirrettävästä pisteestä valonlähteeseen, ja laskea varjostusarvo sen mukaan, kuinka suuri osa valonlähteestä on näkyvissä. Kuten edellisessäkin menetelmässä, valaistusasteiden arvot ovat välillä [0,1] ilmaisten prosentteina, kuinka paljon valonlähteestä on piirrettävän pisteen p näkökulmasta näkyvissä. Bittikarttapeittojen tapauksessa p:n ja valonlähteen väliin ei kuitenkaan projisoida geometriaa vaan syvyyskartta, joka lasketaan valon keskipisteestä. Täten menetelmä ei vaadi tietoa piirrettävästä geometriasta, eli se jakaa muiden bittikarttamenetelmien hyvät puolet, ja on varsinkin monimutkaisten mallien varjostamiseen paljon varjokiiloja elegantimpi

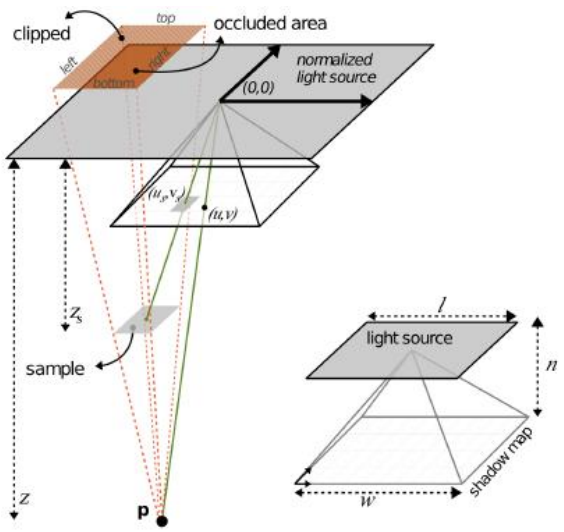
ratkaisu. Valonlähteiden on periaatteessa oltava suorakaiteen muotoisia, mutta sopivilla maskeilla voidaan luoda minkä muotoinen valo tahansa. Tekstikarttavaloja voidaan tukea samalla periaatteella kuin Assarsson ja Akenine-Möller [2003] tekivät varjomallien kohdalla

Menetelmä muuttaa syvyyskartan pikselit nk. peitoiksi (micropatch), jotka ovat tilallisen valonlähteen suuntaisia, kolmiulotteisia suorakulmioita. Valosta katsottuna yksi peitto on täsmälleen yhden varjokartan pikselin kokoinen, eli globaalissa koordinaatistossa niiden koko kasvaa perspektiivin mukaisesti suhteessa niiden etäisyyteen valosta (kuva 41). Kun varjokiilojen yhteydessä p:n näkökenttään projisoitiin varjostavan mallin siluettireunat, tässä menetelmässä siihen projisoidaan syvyyskartasta luodut peitot. Mitä suuremman osan valonlähteestä peitot peittävät, sitä vähemmän valonlähteestä on näkyvissä, ja sitä tummempi piste p on.



Kuva 41. Peitot esitettynä kaksiulotteisessa koordinaatistossa. L on valonlähteen keskipiste, NP on taso jolle syvyyskartta luodaan, ja A, B ja C ovat varjostavia esineitä.

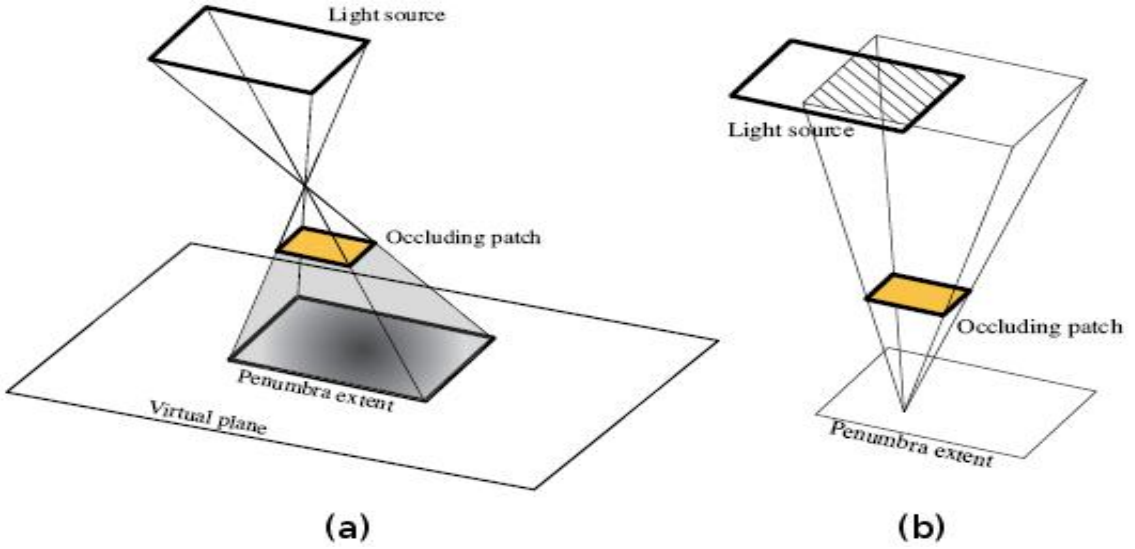
Koska kaikkien peittojen suunta ja suhdeluku on sama kuin valonlähteen, niiden peittämän alueen laskeminen on helppoa ja nopeaa: tarvitsee vain löytää kahden suorakaiteen leikkausalue. Guennebaud et al. [2006] antavat kaavan, jolla peiton kärkipisteet saadaan laskettua p:n muodostamassa avaruudessa (kuva 42). Kaavasta on huomattava se, että z-arvojen on oltava etäisyyksiä valosta globaalissa koordinaatistossa, ei pikseleiden normalisoituja syvyysarvoja. Peiton ääripisteet rajataan alueelle [-½,½], ja sen ala saadaan kertomalla sivujen pituudet. Tämä arvo voidaan vähentää valaistusarvosta ilman erillistä normalisointia, koska laskutoimitukset tehdään yksikköavaruudessa.



$$\mathbf{B} = \begin{bmatrix} b_{left} \\ b_{right} \\ b_{bottom} \\ b_{top} \end{bmatrix} = \begin{matrix} (u_s - u - 1/2) \\ (u_s - u + 1/2) \\ (v_s - v - 1/2) \\ (v_s - v + 1/2) \end{matrix} \frac{w}{n * r} z_s \frac{z}{z - z_s} \frac{1}{l}$$

Kuva 42. Kaava, jolla varjokartan piste muunnetaan peitoksi pisteen yksikköavaruuteen (r on varjokartan resoluutio) [Guennebaud et al., 2006]

Attyn et al. [2006] menetelmä luo pehmennetyn varjokartan, joka voidaan laskemisen jälkeen projisoida näkymän päälle. Menetelmä alkaa piirtämällä kaksi syvyyskarttaa: toiseen piirretään varjostavat esineet ja toiseen varjostettavat. Esineet siis jaetaan varjostaviin ja varjostettaviin esineisiin, eikä menetelmä täten tue itsevarjostuvuutta. Varjostavista esineistä tehdyn syvyyskartan pikselit muutetaan peitoiksi. Sitten lasketaan kunkin peiton langettama puolivarjoalue virtuaalisella tasolla, joka kulkee varjostettavien esineiden etureunalla (kuva 43a). Tämä on se taso, jolle pehmeä varjokartta luodaan. Jokaiselle peiton puolivarjossa olevalle pisteelle lasketaan tämän jälkeen varjostusarvo takaisinprojisointia käyttäen (kuva 43b). Pisteen näkökenttään projisoidaan vain se peitto, joka puolivarjoalueen on synnyttänyt. Pisteen koordinaatit saadaan varjostettavien esineiden syvyyskartasta. Tämän jälkeen varjostusarvo lisätään pehmennetyn varjokartan vastaavan pikselin arvoon. Kun jokaisen peiton osalta on käsitelty kaikki sen varjostamat pikselit, pehmeää varjokarttaa voidaan käyttää projisoimalla se näkymään.



Kuva 43. Attyn et al. [2006] menetelmän toiminta. Peiton puolivarjoalueen laskenta (a) ja takaisinprojisointi (b).

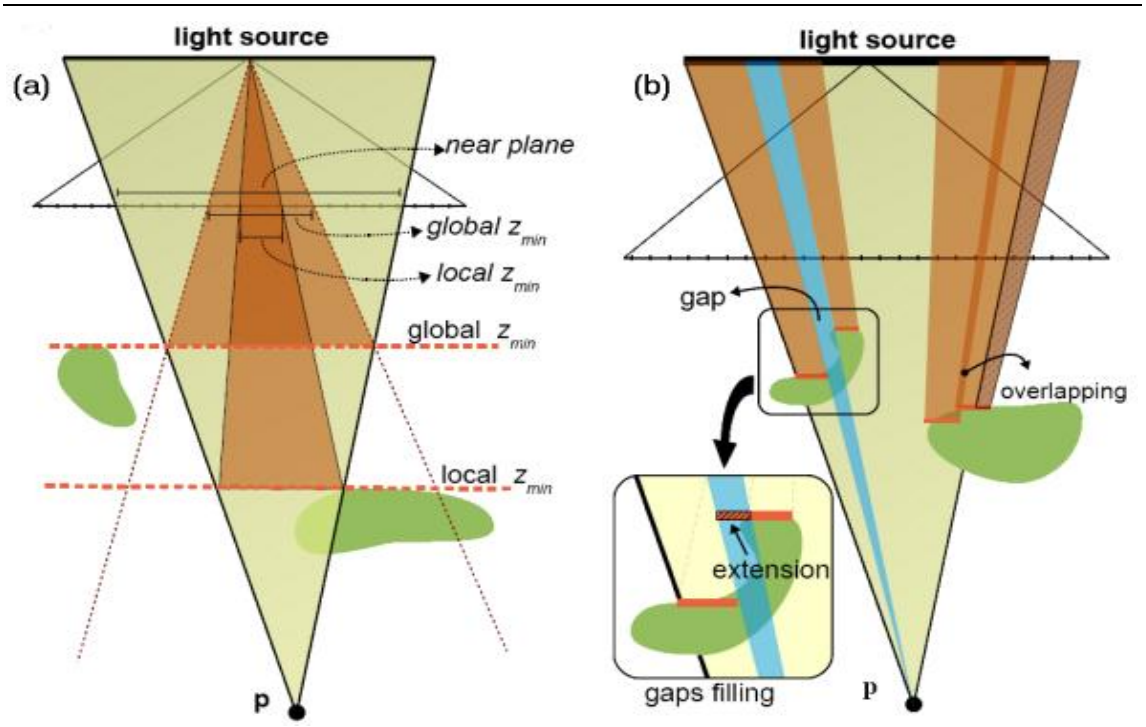
Periaatteessa menetelmä voitaisiin toteuttaa myös siten, että varjostus laskettaisiin kaikille varjostettavien esineiden kartan pisteille. Peittojen puolivarjoalueen laskennan avulla menetelmä on kuitenkin nopeampi, koska sellaiset pisteet, jotka eivät kuulu yhdenkään peiton varjoalueelle, ovat valossa, eikä niitä tarvitse käsitellä takaisinprojisoinnin avulla. Käytännössä menetelmä kykenee käyttämään vain melko pieniä tekstikarttoja [Atty et al., 2006], ja ongelmana on myös se, että koska varjostettavien karttaa estimoidaan virtuaalisella tasolla, tuloksena saattaa olla pahaa laskostumista [Guennebaud et al., 2006]. Suurin ongelma on kuitenkin se, että menetelmä ei tue itsevarjostuvuutta.

Guennebaudin et al. [2006] menetelmä korjaa itsevarjostuvuuden puutteen. Se on muuten samankaltainen kuin Attyn et al. [2006] näkemys, mutta varjostus lasketaan kameranäkymässä. Näin menetelmä tuottaa samanlaisen valaistusastekartan kuin varjokiilamenetelmäkin. Kameranäkymän käyttäminen tekee menetelmästä siinä mielessä edellistä hitaamman, että käsiteltäviä pikseleitä on paljon. Kukin pikseli kuitenkin varjostetaan vain kerran, käymällä samalla kertaa läpi kaikki peitot, jotka kuuluvat varjostettavan pisteen näkökenttään. Menetelmä siis käyttää vain yhtä syvyyskarttaa, joka äärellistää valoon päin osoittavan näkymän. Varjokartta muodostetaan tavalliseen tapaan, mutta myöhemmin esiteltävää optimointia ajatellen siitä täytyy pystyä muodostamaan hierarkia, joten sen resoluution pitäisi olla kahden potenssi. Toisessa vaiheessa luodaan valaistusastekartta, mikä tapahtuu muodostamalla jokaiselle kameranäkymän pisteelle näkökenttä, jonka kärki on piirrettävässä pisteessä (globaalissa koordinaatistossa), ja

takareuna valonlähteessä. Varjostusaste lasketaan samaan tapaan kuin Attyn et al. [2006] tapauksessa, käyttäen kuitenkin kaikkia syvyyskartasta luotuja peittoja. Aluksi valaistus on täysi, eli 1. Tämän jälkeen siitä vähennetään kunkin peiton aiheuttama varjostuksen määrä.

Kuten Attyn et al. [2006] menetelmässäkin, varjostuslaskennan tehokkuus riippuu paljolti siitä, kuinka monta peittoa kunkin pikselin kohdalla on käsiteltävä. Edellistä menetelmää pystyttiin tehostamaan rajoittamalla käsiteltävien varjokartan pikselien määrää, mutta koska Guennebaudin et al. [2006] algoritmissa on varjostettava kaikki kameranäkyvyyden pisteet, on erityisen tärkeää, että ainoastaan pisteen varjostukseen vaikuttavat peitot käsitellään. Luonnollisin tapa peittojen karsimiseksi on tietysti syvyyskartan resoluution pienentäminen, mutta se aiheuttaa laskostumista ja epätarkkoja varjoja.

Guennebaudin et al. [2006] tehostusmalli alkaa mipmap-tyyppisen, hierarkkisen syvyyskartan luomisella. Siinä epätarkempi taso sisältää tarkemman alueen pikseleiden suurimman ja pienimmän syvyysarvon siten, että tarkin taso on syvyyskartta sellaisenaan ja ylin, ainoastaan yhden pikselin sisältävä taso kertoo koko varjokartan syvyysintervallin. Tämän avulla pikselivarjostimessa voidaan nopeasti suodattaa syvyyskartasta pois sellaisia alueita, jotka eivät vaikuta piirrettävän pisteen varjostukseen. Seuraava tehtävä on poistaa sellaiset peitot, jotka sijaitsevat valosta katsottuna p:n takana, tai sen näkökentän ulkopuolella. Tämä tapahtuu rajaamalla syvyyskartasta alue, jonka ulkopuolella olevia pikseleitä ei oteta mukaan varjostustarkasteluun. Tämä alue voidaan rajoittaa globaalissa koordinaatistossa pyramidiin, jonka kärkipiste on valonlähteen keskipisteessä, ja takareuna rajoittuu p:n näkökentän reunoihin syvyydessä z_{min} (p:stä katsottuna). Se, minkä kokoinen tämä näkökenttä on valon todellisen näkökentän etutasolla (jossa syvyyskartta piirretään) ratkaisee sen, mistä pikseleistä luodut peitot vaikuttavat pisteen p varjostumiseen (kuva 44a). Tavoite on siis saada z_{min} mahdollisimman pieneksi.



Kuva 44. Pyramidi, joka rajoittaa käsiteltävien peittojen määrän (a). Peittojen väliin voi jäädä aukkoja tai ne voivat limittyä toistensa päälle (b) [Guennebaud et al., 2006].

Globaali z_{min} (joka on siis sama kaikille varjostettaville pisteille) saadaan hierarkkisen syvyyskartan ylimmän tason ainoasta pisteestä. Tämä arvo on samalla lähimpänä valoa olevan syvyyskarttapisteen arvo. Tämän jälkeen käsiteltävät pikselit on rajoitettu alueelle, joka saadaan seuraavalla kaavalla [Guennebaud et al., 2006] (kirjaimet viittaavat tekijöiden aiemmin esitettyyn kaavaan):

$$w_a = l\frac{n}{w}\left(\frac{1}{z_{min}} - \frac{1}{z}\right).$$

Nyt voidaan määrittää p:n paikallinen minimisyvyys (joka on siis määritettävä jokaiselle varjostettavalle pisteelle erikseen). Se saadaan käymällä hierarkkista syvyyskarttaa läpi rekursiivisesti aiemmin määritellystä alueesta alkaen. Alueelta etsitään lähimpänä p:tä oleva syvyysarvo, ja rajoitetaan hakualuetta joka kierroksella uudelleen (kaavaa käyttäen), kunnes on löydetty p:stä katsottuna kaukaisin piste, joka vaikuttaa varjostukseen. Guennebaud et al. [2006] toteavat yhden iteraatiokierroksen riittävän käytännössä.

Hierarkiaan tallennettuja alipuun suurimpia syvyysarvoja voidaan käyttää tunnistamaan tilanteet, joissa p on täysvarjossa. Näin käy silloin, jos etsittävän alueen (valosta katsottuna) kaukaisin piste on valonlähteen ja p:n välissä. Tämä tarkoittaa sitä, että kaikki alueen peitot ovat p:n ja valon välissä, ja näin ne peittävät valonlähteen kokonaan. Jos $p_z \leq z_{min}$, kaikki peitot ovat valosta

katsottuna p:n takana, ja p näin ollen kokonaan valossa. Myös sitä, kuinka tarkalle tasolle hierarkkista syvyyskarttaa käsitellään, voidaan säätää tietyn kynnysarvon mukaan. Näin syvyyskartan tarkkuutta voidaan muuttaa koska tahansa. Tämä mahdollistaa ajonaikaisesti säädettävän kompromissin varjon laadun ja algoritmin nopeuden välillä, jolloin esimerkiksi kameran liikkuessa voidaan panostaa nopeuteen, koska varjojen epätarkkuus on vaikeasti huomattavissa. Kun kamera pysähtyy, voidaan käyttää tarkempia varjoja, koska nopeudella ei ole enää niin suurta merkitystä.

Hierarkkinen kartta on tehokas menetelmä, jota näytönohjaimet tukevat suoraan mipmappien muodossa. Tässä tapauksessa se ei kuitenkaan ole ihanteellinen, sillä mikäli käsiteltävä suorakulmio leikkaa yhdenkin pisteen jonkin alipuun alueelta, koko alipuu joudutaan käymään tarkastelussa läpi. Tämä aiheuttaa monesti tehottomuutta, koska tarkasteltava alue harvoin rajoittuu juuri yhden alipuun alueeseen. Schwarz ja Stamminger [2007] käyttävätkin perinteisen hierarkian sijasta skaalautuvaa syvyyskarttaa, jonka kaikki tasot sisältävät yhtä suuren tekstikartan. Tason i piste sisältää suurimman ja pienimmän syvyysarvon alemman tason alueella $2^i x 2^i$, jonka keskipisteessä se on. Tämä mahdollistaa sen, että kutakin käsiteltävää aluetta voidaan tarkastella yksittäisesti.

Menetelmän hankaluutena on se, että koska kunkin peiton varjostama alue joudutaan laskemaan erikseen, p:n näkökentästä katsottuna päällekkäiset peitot aiheuttavat liiallista tummentumista, koska limittyvät alueet lisätään varjostusasteeseen monta kertaa. Vielä suurempi ongelma on se, että peittojen väliin voi myös jäädä aukkoja, jolloin varjoihin syntyy helposti havaittavia vaaleita kohtia (kuva 44b).

Välien täyttäminen on onneksi helppoa, koska voidaan tehdä oletus, että aukot sijaitsevat aina vierekkäisten peittojen välissä [Guennebaud et al., 2006]. Täten kutakin peittoa voidaan venyttää siten, että se kohtaa p:n kaksiulotteiseksi projisoidussa näkökentässä naapuripeiton rajat. Täyttäminen tehdään laskemalla sekä käsiteltävän peiton että sen vasemmalla ja yläpuolella olevien naapureiden ääripisteet. Tämän jälkeen peiton vasen reuna venytetään vasemman naapurin oikeaa reunaa vasten, ja yläreuna yläpuolella olevan naapurin alareunaa vasten. Jos naapuripeitto on kauempana valosta kuin käsiteltävä piste p, venytystä ei luonnollisestikaan tarvitse tehdä. Limittäin menevien peittojen kohdalla vastaavaa korjausmenetelmää ei ole, koska limittäin voi olla useita peittoja, jotka eivät välttämättä ole toistensa naapureita. Limittymisistä johtuva varjojen tummuminen (jota aukkojen täyttäminen jossain määrin pahentaa) ei ole kuitenkaan niin huomattava ongelma kuin aukot. Yleensä limittäisyys muuttuu suuremmaksi ongelmaksi vasta, kun yksi

erittäin lähellä valoa oleva peitto peittää monta valosta kaukana olevaa. Tällaiset tilanteet ovat kuitenkin hyvin harvinaisia [Guennebaud et al., 2006].

Menetelmään on lyhyessä ajassa annettu useita kehitysehdotuksia. Bavoil et al. [2006] käyttävät monitasoisia syvyyskarttoja (ks. luku 5.2.1) aukkojen täyttämiseksi, koska Guennebaudin et al. [2006] esittelemä malli ei toimi erityisen hyvin ohuiden esineiden varjostamisessa. He piirtävät näkymän useaan kertaan (kehittäjien mukaan kolme kerrosta riittää), ja luovat peitot lähimpänä p:tä olevista pisteistä. Menetelmä on luonnollisesti hitaampi kuin yksikerroksista syvyyskarttaa käyttävät, mutta toisaalta sen avulla voidaan korjata myös syvyysarvojen epätarkkuuteen liittyviä ongelmia.

Schwarz ja Stamminger [2007] korjaavat varjojen tummentumista pilkkomalla valon palasiksi, ja luomalla bittikartan, jossa yksi bitti vastaa yhtä valonlähteen osaa. Bitin arvo 0 tarkoittaa sitä, että kyseinen valon alue näkyy käsiteltävästä pisteestä. Muussa tapauksessa alue on varjostavan esineen peittämä. Pisteen p varjostaminen toimii asettamalla kaikki bitit ensin nolliksi. Tämän jälkeen kukin käsiteltävä peitto projisoidaan p:n näkökenttään ja määritetään se, mitkä jaetuista alueista jäävät sen taakse. Vastaavat bitit asetetaan tämän jälkeen ykkösiksi. Jos alue on jo jonkun toisen peiton peittämä, sen arvo ei muutu, eikä päällekkäisyydellä näin ole merkitystä varjostusarvoon. Lopullinen varjostusaste määräytyy sen mukaan, kuinka suuri osuus biteistä on ykkösiä. Menetelmän tuottamien varjojen tarkkuus riippuu siitä, kuinka moneen osaan valo on pilkottu. Schwarz ja Stamminger [2007] huomauttavat myös, että bittikentän avulla on mahdollista yhdistää useamman eri syvyyskartan tulokset. Tästä on apua esimerkiksi laajojen valonlähteiden kohdalla, koska sen avulla voidaan luoda useampi syvyyskartta eri näytepisteistä, ja laskea p:n näkyvyys niiden kaikkien suhteen. Tämä korjaisi pistevalosta tehtyjen pehmeiden varjojen perusongelmaa merkittävästi. Menetelmää voidaan myös käyttää tehostamaan teksturoituja valonlähteitä. Guennebaud et al. [2007] esittelevät viimeisimpänä ajatuksenaan varjotilavuusmenetelmistä tutun ääriviivalaskennan yhdistämistä takaisinprojisoituihin varjokarttoihin, sekä mukautuvien varjokarttojen kaltaista lähestymistapaa varjon tarkkuusvaatimusten arviointiin. Direct3D 10:n tarjoama tuki bittikohtaisille operaatioille pikselivarjostimissa tekee näistä menetelmistä varteenotettavan tutkimuskohteen tulevaisuutta ajatellen.

7. Yhteenveto

Reaaliaikaiset varjomallinnusmenetelmät ovat käyneet läpi mielenkiintoisen kehityskaaren. Vuoroin on pyritty kehittämään menetelmiä, jotka rajatuissa tilanteissa antavat fysikaalisesta virheellisyydestään huolimatta ohjelman

käyttäjälle illuusion todellisuudesta, ja vuoroin taas keskitytty kehittämään näitä menetelmiä eteenpäin, että vaativammatkin näkymät voidaan varjostaa niiden avulla yleispätevästi.

Yleisesti ottaen, yksinkertaisuus on kaunista myös varjoalgoritmien tapauksessa. Reaaliaikaiseen varjomallinnukseen on kehitetty lukemattomia menetelmiä, joista monimutkaisimmat tuntuvat poikkeuksetta sisältävän eniten epätarkkuuksia ja erikoistapauksia. Matemaattisesti oikeaoppinen lähestymistapa ei yleensä ole paras, vaikka laitteistojen laskentavoima kasvaakin kovaa tahtia. Pyöristysepätarkkuudet pahenevat laskujen monimutkaistuessa, ja niistä aiheutuvien virheiden korjaaminen on usein erittäin hankalaa. Reaaliaikamenetelmissä onkin usein kysymys enemmän ohjelmoinnillisista kuin matemaattisista oivalluksista, ja ennen kaikkea kyvystä löytää oleelliset asiat kaavojen takaa, ja jättää kaikki ylimääräinen pois.

Varjotilavuusmenetelmien hyvänä puolena on aina ollut muiden vektorigraafisten algoritmien tapaan se, että niiden avulla on voitu saavuttaa tarkkaa grafiikkaa suhteellisen pienillä muistivaatimuksilla. Tämä oli erityisen tärkeää silloin, kun tietokoneet eivät vielä kyenneet pitämään muistissaan kerralla useita suuria kuvia. Kuten vektoripohjaiset menetelmät yleensäkin, muistin määrää joudutaan kuitenkin kompensoimaan laskutoimituksilla, että mallit saadaan tuottamaan haluttua kuvaa. Viime vuosina varjotilavuuksien käyttö onkin selvästi vähentynyt, mikä johtuu siitä, että niiden tehokkuus laskee näkymien monimutkaistuessa. Toisaalta laitteiden muistimäärien nopea kasvu mahdollistaa tarkemmat tekstikartat, ja näin niihin liittyvät laskostumisongelmat helpottuvat vuosi vuodelta. Varjokarttamenetelmät toimivat myös varjomalleja yleisemmällä tasolla, ja ne voivat helpommin hyödyntää kaksiulotteiseen kuvankäsittelyyn kehitettyjä menetelmiä, joita tutkitaan paljon 3D-grafiikan ulkopuolella.

Tulevaisuudessa realismivaatimukset kasvavat, ja näin pehmeistä varjoista tulee jossain vaiheessa standardi. Riippuu hyvin pitkälle laitevalmistajista, minkälaisia ominaisuuksia menetelmien toteuttamiseksi saadaan. Tällä hetkellä kuitenkin vaikuttaa siltä, että varjokartat ovat seuraavina vuosina ottamassa tukevan niskalenkin varjomallimenetelmistä. Toisaalta geometriavarjostimien kehittyminen saattaa jossain vaiheessa yksinkertaistaa varjotilavuusalgoritmien toteuttamista, ja nostaa ne jälleen esiin. Karkeasti ottaen voikin sanoa, että varjotilavuudet hyötyvät nopeista matematiikkaprosessoreista, ja varjokartat taas kasvavasta muistikapasiteetista. Sillä, kumpi laitteissa kasvaa nopeammin, on ratkaiseva merkitys siihen, millaisia varjoalgoritmeja tulevaisuuden reaaliaikaisissa 3D-ohjelmistoissa käytetään.

Viiteluettelo

[Agrawala et al., 2000] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll, Efficient image-based methods for rendering soft shadows. *In Computer Graphics (SIGGRAPH 2000),* Annual Conference Series, pages 375–384. ACM SIGGRAPH, 2000.

[Akenine-Moeller and Assarsson, 2002] Akenine-Moeller, T., and Assarsson, U., Approximate soft shadows on arbitrary surfaces using penumbra wedges. *In 13th Eurographics Workshop on Rendering 2002*.

[Ambrož, 2006] David Ambrož, Planar shadows by J. Blinn. 2006. Available as *http://www.shadowstechniques.com/blinn.html*.

[Assarsson and Akenine-Möller, 2003] Ulf Assarsson and Tomas Akenine-Möller, A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics* (SIGGRAPH 2003), **22**(3), 2003.

[Assarsson et al., 2003] Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller, An optimized soft shadow volume algorithm with real-time performance. *In Graphics Hardware*, 2003.

[Atty et al., 2006] Atty L., Holzschuch N., Lapierre M., Hasenfratz J.-M., Hansen C. and Sillion F., Soft shadow maps: efficient sampling of light source visibility. Computer Graphics Forum (2006). (to appear).

[Barequet et al., 2001] G. Barequet, C.A. Duncan, M. T. Goodrich, W. Huang, S. Kuma and M. Pop, Efficient perspective-accurate silhouette computation. *Proc. 17th Ann. ACM Symp. on Computational Geometry (SoCG)*, Medford, MA, pp. 60-68, June 2001

[Batagelo and Júnior, 1999] Harlen Costa Batagelo and Ilaim Costa Júnior, Realtime shadow generation using BSP trees and stencil buffers. *In SIBGRAPI*, volume **12**, pages 93–102, October 1999.

[Bavoil et al., 2006] Louis Bavoil, Steven P. Callahan and Claudio T. Silva, Robust soft shadow mapping with depth peeling. SCI Institute Technical Report, No. **UUSCI-2006-028**, University of Utah, 2006.

[Bergeron, 1986] Philippe Bergeron, A general version of crow's shadow volumes. *IEEE Computer Graphics and Applications*, Sept. 1986, pp. 17-28.

[Bilodeau and Songy, 1999] Bill Bilodeau and Mike Songy, *Creative Labs sponsored game developer conference*, unpublished slides, Los Angeles, May 1999.

[Blinn, 1988] Blinn, James, Me and my (fake) shadow. *IEEE Computer Graphics and Applications*, January 1988. N. (eds.) ISBN 3-211-83213-0

[Blinn, 1993] Jim Blinn, A trip down the graphics pipeline: the homogeneous perspective transform. *IEEE Computer Graphics and Applications*, May 1993, pp. 75-88.

[Brabec and Seidel, 2002] Brabec, S., and Seidel, H.-P., Single sample soft shadows using depth maps. *In Proc. Graphics Interface*, 219–228, 2002.

[Brabec and Seidel, 2003] Stefan Brabec and Hans-Peter Seidel, Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics 2003)*, **25**(3), September 2003.

[Brabec et al., 2002a] Brabec S., Annen T. and Seidel H.-P., Practical shadow mapping. *Journal of Graphics Tools* **7**, 4 (2002), pp. 9-18.

[Brabec et al., 2002b] Brabec, S., Annen, T. and Seidel, H.-P., Shadow mapping for hemispherical and omnidirectional light sources. *In Computer Graphics International*, 2002.

[Brennan, 2002] Chris Brennan, Shadow volume extrusion using a vertex shader. In ShaderX: Vertex and Pixel Shaders Tips and Tricks, Wolfgang Engel editor, Wordware, May 2002.

[Carmack, 2000] John Carmack, Carmack on shadow volumes, correspondence between Mark Kilgard and John Carmack, 2000.

[Chan and Durand, 2003] Eric Chan and Fredo Durand, Rendering fake soft shadows with smoothies. *In Rendering Techniques 2003 (14th Eurographics Symposium on Rendering).* ACM Press, 2003.

[Chong and Gortler, 2004] Chong H. and Gortler S., A lixel for every pixel. *In Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 167-172.

[Crow, 1977] Crow, F. C., Shadow algorithms for computer graphics. *In Computer Graphics (Proceedings of SIGGRAPH 77)*, vol. **11**, 242–248.

[Diefenbach, 1996] Paul Diefenbach, Multi-pass pipeline rendering: interaction and realism through hardware provisions. Ph.D. thesis, University of Pennsylvania, tech report **MS-CIS-96-26**, 1996.

[Dietrich, 2001] Sim Dietrich, Shadow techniques. NVIDIA Corporation, 2001. PowerPoint Presentation available from *http://www.nvidia.com*.

[Donelly and Lauritzen, 2006] Donnelly W. and Lauritzen A., Variance shadow maps. *In SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006) ACM Press pp. 161-165

[Drettakis and Fiume, 1994] George Drettakis and Eugene Fiume, A fast shadow algorithm for area light sources using backprojection. *In Computer Graphics (SIGGRAPH 1994)*, Annual Conference Series, pages 223–230. ACM SIGGRAPH, 1994.

[Everitt et al., 2001] C. Everitt, A. Rege, and C. Cebenoyan, Hardware shadow mapping. White paper, nVIDIA, 2001.

[Everitt and Kilgard, 2002] Everitt, C., and Kilgard, M. J., Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Published online at *developer.nvidia.com*.

[Everitt and Kilgard, 2003] Cass Everitt and Mark J. Kilgard, Optimized stencil shadow volumes. 2003

[Fauerby and Kjær, 2003] Real-time Soft Shadows in a Game Engine. Master's thesis, December 2003.

[Fernando et al., 2001] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg, Adaptive shadow maps. *Proc. of SIGGRAPH 2001*, pages 387–390, 2001.

[Forest et al., 2006] Forest V., Barthe L. and Paulin M., Realistic soft shadows by penumbra-wedges blending. *In Proceedings of Graphics ardware 2006* (2006), pp. 39-47.

[Fournier and Fussell, 1988] Alain Fournier and Donald Fussell, On the power of the frame buffer. *ACM Transactions on Graphics*, April 1988, 103-128.

[Guennebaud et al., 2006] Gael Guennebaud, Loıc Barthe and Mathias Paulin, Realtime soft shadow mapping by backprojection. *In Eurographics Symposium on Rendering*, pp. 227–234, 2006.

[Guennebaud et al., 2007] Gaël Guennebaud, Loïc Barthe and Mathias Paulin, High-quality adaptive soft shadow mapping. In *Eurographics*, 2007.

[Guinot, 2005] Jérôme Guinot, Stencil shadow volumes. 2005. Available as *http://www.ozone3d.net/tutorials/stencil_shadow_volumes.php*.

[Haines, 2001] Eric Haines, Soft planar shadows using plateaus. *Journal of Graphics Tools*, **6**(1):19–27, 2001.

[Hasenfrantz et al., 2003] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch and François Sillion, A survey of real-time soft shadows algorithms. *In Computer Graphics Forum*, Volume **22**, Number 4 - 2003.

[Heckbert and Herf, 1997] P. S. Heckbert and M. Herf, Simulating Soft Shadows with Graphics Hardware. Technical Report **CMU-CS-97-104**, Carnegie Mellon University, Januar 1997

[Heidmann, 1991] Heidmann, T., Real shadows real time. *IRIS Universe*, **18**, 28–31.

[Heidrich and Seidel, 1998] Wolfgang Heidrich and Hans-Peter Seidel, View-independent environment maps. *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 39–46, August 1998. Held in Lisbon, Portugal.

[Heidrich et al., 2000] Wolfgang Heidrich, Stefan Brabec and Hans-Peter Seidel, Soft shadow maps for linear lights high-quality. *In Rendering Techniques 2000 (11th Eurographics Workshop on Rendering)*, pages 269–280. Springer-Verlag, 2000. Slides available as *http://www.mpi-inf.mpg.de/~brabec/*.

[Herf, 1997] Michael Herf, Efficient generation of soft shadow textures. Technical Report **CMU-CS-97-138**, Carnegie Mellon University, 1997.

[Hornus et al., 2005] Hornus S., Hoberock J., Lefebvre S. and Hart J., ZP+: correct z-pass stencil shadows. *In SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, pp. 195–202.

[Hourcade and Nicolas, 1985] J.-C. Hourcade and A. Nicolas, Algorithms for antialiased cast shadows. *Computers & Graphics*, **9**(3):259–265, 1985.

[Isard et al., 2002] M. Isard, M. Shand and A. Heirich, Distributed rendering of interactive soft shadows. In *4th Eurographics Workshop on Parallel Graphics and Visualization*, pages 71–76. Eurographics Association, 2002.

[Kilgard, 1999] M. J. Kilgard, Improving shadows and reflections via the stencil buffer. Advanced Game Development course notes, *Game Development Conference*, pp. 204-253, March 16, 1999

[Kilgard, 2001a] Mark Kilgard, Robust stencil volumes. *CEDEC 2001* presentation, Tokyo, Sept. 4, 2001.

[Kilgard, 2001b] Kilgard, M. J., Shadow mapping with today's opengl hardware. Published online at *developer.nvidia.com*.

[Kim and Neumann, 2001] Kim T.-Y., Neumann U., Opacity shadow maps. *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer-Verlag, pp. 177–182.

[King, 2004] Gary King, Shadow mapping algorithms. Technical report, NVIDIA Corporation, 2004.

[Kirsch and Doellner, 2003] Florian Kirsch and Juergen Doellner, Real-time soft shadows using a single light sample. *Journal of WSCG* (Winter School on Computer Graphics 2003), **11**(1), 2003.

[Lefohn et al., 2005] Aaron Lefohn, Shubhabrata Sengupta, Joe Kniss, Robert Strzodka and John D. Owens, Dynamic adaptive shadow maps on graphics hardware. *International Conference on Computer*, 2005

[Lefohn et al., 2007] Aaron E. Lefohn, Shubhabrata Sengupta and John D. Owens, Resolution-matched shadow maps. *ACM Transactions on Graphics*, Vol. **26**, No. 4, October 2007, Pages 1-23.

[Lengyel, 2002] Eric Lengyel, The mechanics of robust stencil shadows. Available in *http://www.gamasutra.com* (October 2002).

[Lengyel, 2005] Lengyel E., Advanced stencil shadow and penumbral wedge rendering. Presentation at *Game Developers Conference 2005*, available as *http://www.terathon.com/gdc_lengyel.ppt*.

[Lokovich and Veach, 2000] Tom Lokovic and Eric Veach, Deep shadow maps. *In Computer Graphics (SIGGRAPH 2000),* Annual Conference Series, pages 385–392. ACM SIGGRAPH, 2000.

[Lloyd et al., 2004] Lloyd B., Wendt J., Govindaraju N. K. and Manocha D., CC shadow volumes. *In Proceedings of the Eurographics Symposium on Rendering* (2004).

[Lloyd et. al., 2006] Brandon Lloyd, David Tuft, Sung-eui Yoon and Dinesh Manocha, Warping and partitioning for low error shadow maps. *In proceedings of the Eurographics Symposium on Rendering 2006*, p. 215-226.

[Mamassian et al., 1998] Pascal Mamassian, David C. Knill, and Daniel Kersten, The perception of cast shadows. *Trends in Cognitive Sciences*, **2**(8):288–295, 1998.

[Markosian et al., 1997] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein and John F. Hughes, Real-time nonphotorealistic rendering. *Computer Graphics (Proceedings of SIGGRAPH '97)*, August, 1997.

[Martin and Tan, 2004] Martin, T. and Tan, T.-S., Anti-aliasing and continuity with trapezoidal shadow maps. *In Proc. Eurographics Symposium on Rendering 2004*, 153--160.

[McCool, 2000] McCool, M. D., Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics* **19**, 1 (January), 1–26, 2000, ISSN 0730-0301.

[McGuire et al., 2003] McGuire M., Hugues J. F., Egan K. T., Kilgard M. and Everitt C., Fast, practical and robust shadows. Tech. Rep. **CS03-19**, Brown University, October 2003.

[Mertens et al., 2004] Mertens T., Kautz J., Bekaert P., Van Reeth F., A selfshadow algorithm for dynamic hair using clustered densities. *In Proceedings of Eurographics Symposium on Rendering 2004* (June 2004), pp. 173–178.

[Morein, 2000] Steve Morein, ATI Radeon hyperZ technology. *In Workshop on Graphics Hardware, Hot3D Proceedings*. ACM SIGGRAPH/Eurographics, August 2000.

[Parker et al., 1998] Steven Parker, Peter Shirley and Brian Smits, Single sample soft shadows. Technical Report **UUCS-98-019**, Computer Science Department, University of Utah, October 1998.

[Reeves et al., 1987] W. T. Reeves, D. H. Salesin and R. L. Cook, Rendering antialiased shadows with depth maps. *Computer Graphics (Proc. of SIGGRAPH 87)*, **21**(4):283–291, 1987.

[Roettger et al., 2002] Stefan Roettger, Alexander Irion and Thomas Ertl, Shadow volumes revisited. *In Winter School on Computer Graphics*, 2002.

[Scherzer, 2005] Daniel Scherzer, Robust shadow maps for large environments. *Proceedings of the Central European Seminar on Computer Graphics 2005*, ISSN, 2005

[Schwarz and Stamminger, 2007] Michael Schwarz and Marc Stamminger, Bitmask soft shadows. *Computer Graphics Forum* **26**:3 (2007), 515–524.

[Segal et al., 1992] Segal M., Korobkin C., Van Widenfelt R., Foran J. and Haeberli P., Fast shadows and lighting effects using texture mapping. In *Proceedings of SIGGRAPH* (1992), pp. 249-252.

[Sen et al., 2003] Sen P., Cammarano M. and Hanrahan P., Shadow silhouette maps. *In Proceedings of SIGGRAPH* (2003), pp. 521–526.

[Stamminger and Drettakis, 2002] Stamminger, M. and Drettakis, G., Perspective shadow maps. *In Proceedings of ACM SIGGRAPH 2002*, ACM Press/ ACM SIGGRAPH, J. Hughes, Ed., Annual Conference Series.

[Sutherland, 1973] Sutherland, I. E., Polygon sorting by subdivision: a solution to the hidden Surface Problem. Unpublished, 1973.

[Tadamura et al., 2001] Tadamura, K., Qin, X., Jiao, G. and Nakamae, E., Rendering optimal solar shadows with plural sunlight depth buffers. *The Visual Computer* **17**, 2, 76-90, 2001.

[Upstill, 1990] S. Upstill, *The RenderMan Companion*. Addison-Wesley, 1990.

[van Waveren, 2005] J.M.P. van Waveren, Shadow Volume Construction. 2005. Available as *http://cache-www.intel.com/cd/00/00/29/37/293752_293752.pdf*

[Weiskopf and Ertl, 2003] D. Weiskopf and T. Ertl, Shadow mapping based on dual depth layers. *In Procceedings of Eurographics '03* Short Papers, pages 53–60, 2003.

[Wang and Molnar, 1994] Yulan Wang and Steven Molnar, Second-depth shadow mapping. Technical report, University of North Carolina at Chapel Hill, 1994.

[Williams, 1978] Williams, L., Shadow algorithms for computer graphics. *Computer Graphics*, vol. **12**, no. 3, pp270-4, 1978.

[Williams, 1983] L. Williams, Pyramidal parametrics. *Computer Graphics* (Proceedings of SIGGRAPH 83), **17**(3):1–11, July 1983. P. Tanner, editor.

[Wimmer et al., 2004] Michael Wimmer, Daniel Scherzer and Werner Purgathofer, Light space perspective shadow maps. *In Proceedings of the Eurographics Symposium on Rendering 2004*, pp. 143–152.

[Woo et al., 1990] Andrew Woo, Pierre Poulin and Alain Fournier, A survey of shadow algorithims. *IEEE Computer Graphics and Applications*, **10**(6):13-32, November 1990

[Woo, 1992] A. Woo. The shadow depth map revisited. In D. Kirk, editor, *Graphics Gems III*, pages 338-342. AP Professional, Boston, 1992.

[Wyman and Hansen, 2003] Chris Wyman and Charles Hansen, Penumbra maps: approximate soft shadows in real-time. *In Rendering Techniques 2003 (14th Eurographics Symposium on Rendering)*. ACM Press, 2003.

[Ying et al., 2002] Zhengming Ying, Min Tang and Jinxiang Dong, soft shadow maps for area light by area approximation. *In 10th Pacific Conference on Computer Graphics and Applications*, pages 442–443. IEEE, 2002.

[Yuksel and Keyser, 2007] Cem Yuksel and John Keyser, Deep opacity maps. Technical report **tamu-cs-tr-2007-6-1**, Department of Computer Science, Texas A&M University.

[Zhang et al., 2006a] Zhang, F., Sun, H., Xu, L., and Lun, L. K., Parallel-split shadow maps for large-scale virtual environments. *In Proceedings of the 2006 ACM international Conference on Virtual Reality Continuum and Its Applications* (Hong Kong, China). VRCIA '06. ACM, New York, NY, 311-318.

[Zhang et al., 2006b] Zhang, F., Xu, L., Tao, C. and Sun, H., Generalized linear perspective shadow map reparameterization. *In Proc. of the 2006 ACM international conference on Virtual reality continuum and its applications*, pp. 339–342. ACM Press, New York (2006)