Avaruusjakoon perustuvat tietorakenteet tietokonegrafiikassa

TURUN YLIOPISTO
Informaatioteknologian laitos
tietojenkäsittelytieteet
4. joulukuuta 2016
kandidaatintutkielma
Timo Heinonen

Tiivistelmä

TURUN YLIOPISTO
Informaatioteknologian laitos

HEINONEN, TIMO: Avaruusjakoon perustuvat tietorakenteet tietokonegrafiikassa
kandidaatintutkielma, 25 s.
Tietojenkäsittelytieteet
4. joulukuuta 2016

Tässä tutkielmassa tutustutaan kolmiulotteisten kuvien hahmontamiseen kaksiulotteisiksi kuviksi säteenseurannaksi kutsutulla tekniikalla. Säteenseurannalla saadaan muodostettua realistisia kuvia, mutta se on laskennollisesti erittäin raskasta. Tästä syystä hahmonnuksen kohteena olevasta maisemasta muodostetaan ennen hahmonnusta avaruuden osiin jakamiseen perustuva tietorakenne, jota läpikäymällä voidaan vähentää tarvittavien laskutoimitusten määrää.

Avaruusjakorakenteista esitetään BSP-puu, sen erikoistapaus kd-puu ja rajaavien tilojen hierarkia BVH. Oikein muodostettuna näitä rakenteita käyttämällä voidaan vähentää hahmontamisessa tarvittavia säteiden ja monikulmioiden leikkausta testaavia operaatioita logaritmiseen määrään.

Lopuksi vertaillaan tietorakenteita keskenään ja yritetään löytää niistä sellainen, joka nopeuttaisi säteenseurantaa eniten. Tulokset hahmontamisen nopeudessa vaihtelevat kuitenkin kuvattavien maisemien, käytettyjen implementaatioiden sekä laitteistojen välillä niin paljon, ettei tehokkainta tietorakennetta pystytä osoittamaan.

Asiasanat: tietokonegrafiikka, säteenseuranta, avaruusjako, BSP-puu, kd-puu, rajaava tila.

Sisällys

1 Johdanto

Kolmiulotteisen tietokonegrafiikan tutkimuksella on ollut merkittävä vaikutus viihdeteollisuuteen, kuten animaatioelokuviin, peleihin ja virtuaalitodellisuuteen, sekä tietokoneavusteiseen suunnitteluun, esimerkiksi arkktehtuurin ja teollisuuden alalla. Tietokonegrafiikka on osin jopa syrjäyttämässä perinteistä valokuvaustyötä: huonekalujätti Ikea on siirtynyt käyttämään myyntikuvastoissaan valtaosin tietokoneella generoituja kuvia valokuvien sijaan [Parkin, 2014]. Tietokonegrafiikan sovelluskohteet lisääntyvät jatkuvasti. Eräs aktiivinen tutkimuskohde on esimerkiksi tietokonegrafiikan tekniikoiden soveltaminen konenäköön. [Hughes et al., 2013.]

Grafiikan piirtämistä kolmiulotteisista malleista kaksiulotteisiksi kuviksi kutsutaan hahmontamiseksi (engl. *rendering*). Hahmontamisen lähtökohtana on kuvattava maisema (engl. *scene*), joka sisältää objekteja ja valonlähteitä. Objektit ja valonlähteet on voitava mallintaa matemaattisesti, jotta niille voidaan määrittää sijainti ja suuntaus ja jotta niiden välisiä etäisyyksiä ja suhteita voidaan laskea. Hahmontaminen tapahtuu aina jostakin kuvakulmasta, ja tätä varten määritellään virtuaalinen kamera, jolla on oma sijaintinsa ja suuntauksensa maisemassa. Tämän jälkeen on selvitettävä, mitkä objektit kamera näkee, miten objekteihin osuvat valonsäteet vaikuttavat niiden väriin ja kuvan varjostukseen. Lopuksi lasketaan mitkä värit projisoidaan kuvatason mihinkin pikseliin. [Janke, 2015.]

1960-luvulla tietokonegrafiikkaa käytettiin lähinnä teollisuuden komponenttisuunnittelussa ja arkkitehtuurissa. Tietokoneella osattiin piirtää objektien ääriviivoja (engl. *wireframe*), mutta varjostustekniikoita ei tunnettu. IBM:n tutkija Arthur Appel esitteli algoritmin, joka mallinsi valonsäteitä laskemalla suoran yhtälöitä kuvasta maisemaan ja siitä valonlähteisiin. Tämän tekniikan avulla voitiin piirtää yksinkertaisia varjostuksia. [Appel, 1968.]. Myöhemmin tästä säteenseurannaksi nimetystä tekniikasta tuli erittäin suosittu.

Jo Appel totesi säteenseurannan olevan laskennollisesti erittäin raskasta. Vaikka tietokoneiden ja varsinkin grafiikkaprosessoreiden laskentateho

kasvaa jatkuvasti, ei grafiikan tuottaminen ole vieläkään halpaa tai nopeaa. Kuvista halutaan jatkuvasti realistisempia, ja yksityiskohtaisemmat kuvattavat mallit ja monimutkaiset valaisutekniikat vaativat erittäin paljon laskentatehoa. Esimerkiksi elokuvastudio Pixarin Monsterit-yliopisto -animaatioelokuvan piirtäminen vaati yli sata miljoonaa prosessorituntia [Takahashi, 2013]. Tämän takia tutkimuksen kohteena on ollut jo pitkään entistä nopeampien hahmontamistekniikoiden kehittäminen.

1980-luvulla kehitettiin menetelmiä, joilla voitiin nopeuttaa hahmontamista vähentämällä valonsäteiden ja maiseman osumatarkasteluiden määrää. Steven Rubin ja Turner Whitted esittelivät tekniikan, jossa maisema ositetaan esiprosessointivaiheessa manuaalisesti hierarkisiin laatikoihin. Säteiden ja laatikoiden osumia tarkastelemalla voitiin vähentää operaatioiden kokonaismäärää. [Rubin ja Whitted, 1980.] Henry Fuchs et al. kehittivät toisen metodin, johon kuului myös esiprosessointivaihe, tällä kertaa tietokoneen suorittamana. Maiseman objektit oli jaettu pienempiin monikulmioihin, joista valittiin binääripuun juureksi mahdollisimman keskellä maisemaa oleva. [Fuchs et al., 1980.]

Tässä tutkielmassa esitellään avaruuden jakamiseen perustuvia tietorakenteita, joilla kolmiulotteisten kuvien hahmontamista voidaan nopeuttaa. Luvussa 2 määritellään joitakin grafiikan peruskäsitteitä sekä esitetään algoritmi säteenseurannalle. Luvussa 3 tutkitaan binääristä avaruusjakoa, kd-puuta ja rajaavien tilojen hierarkiaa sekä niiden rakentamiseen liittyviä algoritmeja. Luvussa 4 selvitetään, mitä eroja on tietorakenteiden alustamisessa ja niiden hyödyntämisessä säteenseurannassa, sekä yritetään löytää niistä parhaiten hahmontamisen optimoimiseen soveltuva. Lopuksi luodaan myös lyhyt katsaus muihin optimointitekniikoihin ja reaaliaikaiseen säteenseurantaan.

2 Kolmiulotteisen tietokonegrafiikan peruskäsitteitä

2.1 Määritelmiä

Kolmiulotteisten kuvien hahmontamisen kohteena ovat *objektit*, jotka mallintavat jotakin esinettä tai muotoa avaruudessa $\mathbb{R}^3$. Objektit voidaan esittää tietokoneen muistissa taulukkona pisteitä $P = (x, y, z)\ \epsilon\mathbb{R}^3$: esimerkiksi kolmiota voidaan kuvata kolmella pisteellä ja palloa kahdella pisteellä, jotka esittävät sen keskipistettä ja yhtä pistettä sen pinnalla. [Angel ja Shreiner, 2014.]

Objektit jaetaan lähes kaikissa ei-triviaaleissa tapauksissa monikulmioihin (engl. *Polygon*). Monikulmio $\gamma = \diamond P_1P_2...P_n,\ n > 2$, on samassa tasossa olevien kärkien $P_1, ..., P_n$ muodostaman murtoviivan rajaama alue, jonka kärkien muodostamat janat P_iP_{i+1} eivät leikkaa toisiaan muualla kuin kärjissä [Harju, 2015]. Useimmiten grafiikkasovelluksissa ja -rajapinnoissa valitaan monikulmioiden kärkien lukumääräksi kolme, sillä kolmioiden kärjet muodostavat aina tason, ja grafiikkaprosessorit osaavat operoida kolmioilla erittäin nopeasti [Angel ja Shreiner, 2014].

Objektien sisäpuoli halutaan yleensä jättää huomioimatta, joten monikulmioille on määriteltävä, kummalla puolella on niiden etupuoli. Kolmiot ovat tässäkin suhteessa hyvä valinta monikulmioiden muodoksi, sillä kolmiolle $\triangle P_1P_2P_3$ voidaan helposti laskea etupuolen määrittävä normaali n ristitulolla $(P_2 - P_1) \times (P_3 - P_2)$. Huomioitavaa on, että normaalin suunta riippuu siitä, missä järjestyksessä kärjet P_1, P_2 ja P_3 on määritelty. [Hughes et al., 2013.]

Jotta voitaisiin tarkastella objektien, valonlähteiden ja kuvakulman eli *kameran* välisiä suhteita ja suuntauksia avaruudessa, valitaan kolme koordinaatistoa, jotka on määritelty kolmella toisiinsa nähden kohtisuoralla kantavektorilla $(\vec{i}, \vec{j}, \vec{k})$. Jokaisella objektilla on *lokaalikoordinaatisto*, joka sisältää objektin geometrian. Useimmiten origo sijoitetaan objektin keskipisteeseen.

Maailmakoordinaatisto kuvaa koko avaruutta ja sisältää tietoa siitä, mihin objektien lokaalikoordinaatistojen origot on sijoitettu. Lopuksi tämä maisema kuvataan virtuaalisella kameralla, jolla on oma *kamerakoordinaatistonsa*. Koordinaatistosta toiseen siirtyminen, koordinaatistojen skaalaus ja rotaatio voidaan toteuttaa lineaarikuvauksilla. [Janke, 2015.]

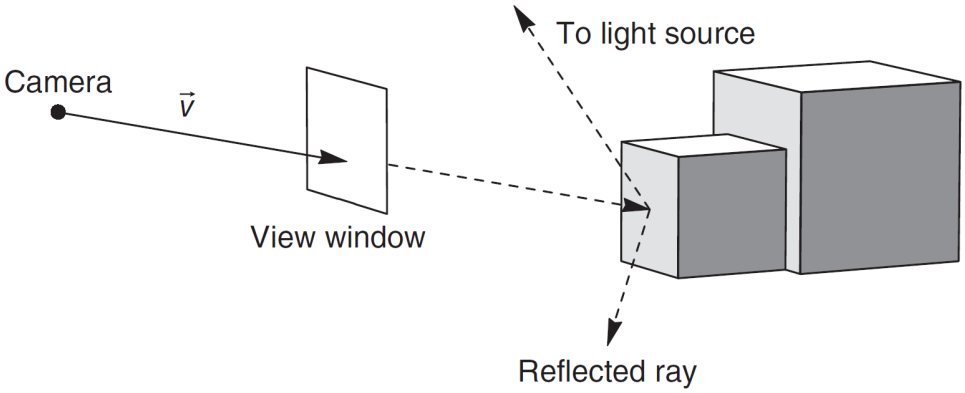
2.2 Säteenseuranta

Säteenseuranta (engl. *Ray Tracing*) on hahmontamistekniikka, jolla voidaan piirtää erittäin fotorealistisia kuvia. Säteenseuranta pyrkii mallintamaan valonsäteitä, jotka saavat alkunsa valonlähteistä, kulkevat avaruudessa ja osuvat objekteihin, jolloin ne valaisevat niitä, kimpoavat niistä toisiin objekteihin ja muodostavat varjoja. Jotkut valonsäteet löytävät lopulta tiensä katsojan silmiin eli kameraan. Koska olisi mahdotonta selvittää jokaisen valonsäteen kulkua avaruudessa, säteenseuranta-algoritmi ottaa huomioon vain ne säteet, jotka todella osuvat kameraan. Valonsäteitä seurataan siis käänteisessä järjestyksessä, kamerasta objekteihin, ja niistä valonlähteisiin. [Janke, 2015.] Säteenseurantatekniikkaa on havainnollistettu kuvassa 1.

Säteenseuranta-algoritmi muodostaa sille syötteenä annetusta kolmiulotteisesta maisemasta kameran sijainnin perusteella kaksiulotteisen kuvan. Jokaisen kuvatason pikselin läpi ammutaan säde $\vec{R} = O + t\vec{D}$, missä $t \epsilon \mathbb{R}$, O on kameran sijainti maailmakoordinaatistossa ja normalisoitu vektori $\vec{D}$ kuvaa säteen kulkusuuntaa. Säteellä etsitään törmäyspistettä lähimmän objektin kanssa eli sellaista mahdollisimman pientä arvoa t, että piste $P = O + t\vec{D}$ on jonkin objektin pinnalla. Tällöin osuman saaneen objektin piste P voi näkyä kameraan, mikäli siihen osuu valoa.[1] Osumakohdasta ammutaan uusi, varjostussäteeksi kutsuttu säde. Jos varjostussäde osuu suoraan tai kimmoten muista objekteista valonlähteeseen, lankeaa objektin pinnalle valoa.[2] [Janke, 2015.] Jos monikulmioiden muodoksi on valittu

[1] Tämä ratkaisee niin kutsutun *näkyvyysongelman*: onko piste P näkyvissä pisteestä O katsottuna?

[2] Tässä tutkielmassa jätetään huomiomatta varjostus, joka on oleellinen osa hahmontamista. Hyvä lähtöpiste realistiseen varjostukseen ja valaistukseen tutustumiseen on esimerkiksi James Kajiyan hahmontamisyhtälö [ks. Kajiya, 1986].



Kuva 1: Säteen ampuminen kuvan läpi maisemaan [Janke, 2015]

kolmiot, voidaan säteen ja kolmion leikkaus määrittää esimerkiksi nopealla Möllerin-Trumboren algoritmilla [ks. Möller ja Trumbore, 1997]. Säteenseurantatekniikan pseudokoodi on esitetty algoritmissa 1.

Algoritmin suoritusnopeutta rajoittaa se, että jokaista sädettä kohti on käytävä läpi kaikki maiseman monikulmiot ja testattava, osuuko säde niihin. Säteiden ja monikulmioiden leikkauksien määrittämiseen joudutaan joissain tapauksissa käyttämään jopa 95 % koko laskenta-ajasta [Whitted, 1980]. Algoritmia saataisiin siis nopeutettua huomattavasti, jos testattavien monikulmioiden määrää jokaista sädettä kohti saataisiin vähennettyä. Yleisesti käytetty tapa leikkaustestien vähentämiseksi on muodostaa maisemasta hierarkinen tietorakenne ennen varsinaista hahmontamista. Tätä tietorakennetta läpikäymällä löydetään nopeasti monikulmio, jonka pinnalla säteen ja objektin leikkauspiste P on. [Rubin ja Whitted, 1980.]

Säteenseurantaa on perinteisesti sen hitauden vuoksi hyödynnetty hahmontamisessa, jossa käytetty laskenta-aika saakin venyä pitkäksi (engl. *Offline Rendering*). Peleissä, virtuaalitodellisuudessa ja muissa reaaliaikaista hahmontamista vaativissa sovelluksissa on käytetty pitkälti niinkutsuttua kolmioiden rasterointitekniikkaa (engl. *rasterization*). Säteenseurannalla hahmonnetut kuvat ovat kuitenkin huomattavasti realistisempia kuin rasteroinnilla saavutetut, joten reaaliaikainen hahmontaminen säteenseurantatekniikalla

kalla on suosittu tutkimuskohde. [Wald, 2004.]

Input:
`kuvataso`: $x * y$ kokoinen taulukko pikseleitä
`maisema`: joukko valonlähteitä ja monikulmioihin jaettuja objekteja
Output:
kolmiulotteinen maisema projisoituna kuvatasolle

`RAY_TRACING(kuvataso, maisema)`:

```
1  foreach pikseli (x, y) ϵ kuvataso do
2      etaisyys ← ∞
3      foreach monikulmio ϵ maisema do
4          Ammu säde R⃗ = O + tD⃗ kamerasta pikselin läpi maisemaan
5          if säde R⃗ osui monikulmioon pisteessä P and t < etaisyys then
6              etaisyys ← t
7              Valon määrä V ← 0
8              foreach valonlähde L do
9                  Ammu varjostussäde R⃗_s = L − P valonlähdettä kohti
10                 Kasvata valosummaa V
11             end
12             Aseta pikselin (x, y) väri valosumman V mukaisesti
13         else
14             Aseta pikseli (x, y) taustan väriseksi
15         end
16     end
17 end
18 return kuvataso
```

Algoritmi 1: `RAY_TRACING`

3 Avaruusjakopuut

Tässä luvussa esitellään kaksi tekniikkaa avaruuden ositukseen säteenseurannan nopeuttamiseksi. Binääriseksi avaruusjaoksi kutsutaan tekniikkaa, jossa avaruutta jaetaan rekursiivisesti kahteen pienempään osaan. Binääriseen avaruusjakoon liittyvistä tietorakenteista esitellään BSP-puu ja sen erikoistapaus kd-puu. Toinen keino jakaa maisema nopeammin käsiteltävään muotoon on hahmotella objekteja ryhmiin ja muodostaa jokaiselle objektille niin kutsuttu rajaava tila. Tällä tekniikalla voidaan muodostaa puun muotoinen rajaavien tilojen hierarkia.

3.1 Binäärinen avaruusjako

3.1.1 BSP-puu

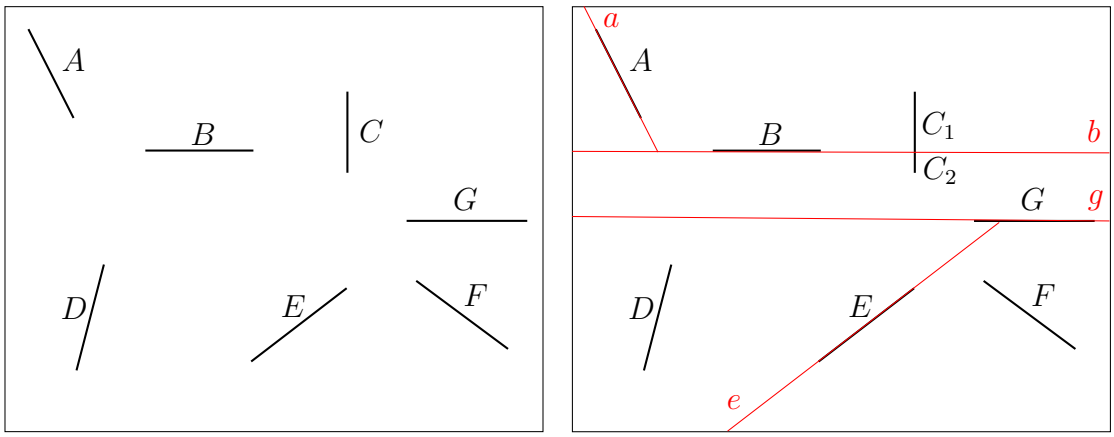
Eräs suosittu avaruusjakoon perustuva tietorakenne on binäärinen avaruusjakopuu, eli *BSP-puu* (engl. *Binary Space Partitioning*). BSP-puu luodaan valitsemalla kolmiulotteisen maiseman monikulmiojoukosta Γ yksi monikulmio γ_k, joka asetetaan puun juureksi. Monikulmion γ_k muodostama taso jakaa maiseman, ja siten monikulmiojoukon Γ, kahteen osaan $\Gamma_{k,+}$ ja $\Gamma_{k,-}$. Joukko $\Gamma_{k,+}$ sisältää monikulmion γ_k positiivisella puolella olevat monikulmiot, ja siten ne asetetaan BSP-puuhun juuren oikeaksi lapseksi. Vastaavasti joukko $\Gamma_{k,-}$ sisältää negatiivisella puolella olevat monikulmiot, ja kuuluvat monikulmion γ_k vasemmaksi lapseksi. Tämä jakavan monikulmion valinta ja avaruuden jako niin sanotuiksi *vokseleiksi* (engl. *voxel, volume element*) suoritetaan rekursiivisesti BSP-puun lehdille, kunnes jokaisessa lehdessä on vain yksi tai ennaltamäärätty määrä monikulmioita. [Samet, 2005.] BSP-puun rakentamisen pseudokoodi on esitelty algoritmissa 2.

BSP-puuta ja sen solmuja voidaan esittää grafiikkasovelluksessa seuraavasti:

```
class BSP_Tree
{
  BSP_Node juuri
}
```

```
class BSP_Node
{
   Polygon jakaja
   BSP_Node* oikea_lapsi    //Osoitin oikeean lapseen
   BSP_Node* vasen_lapsi    //Osoitin vasempaan lapseen
   Polygon monikulmiot[]    //joukko monikulmioita, josta alipuut
                            //muodostetaan
}
```

Kuvissa 2 ja 3 on esitetty esimerkki BSP-puun muodostamisesta. Kuvassa 2a on yksinkertaisuuden vuoksi esitetty monikulmiot $\Gamma = \{A, B, C, D, E, F, G\}$ sisältävä maisema kaksiulotteisena tasona. Kuvassa 2b ensimmäiseksi jakomonikulmioksi on valittu G, jonka positiiviselle puolelle jäävät monikulmiot $\Gamma_{g,+} = \{A, B, C\}$, ja negatiiviselle puolelle $\Gamma_{g,-} = \{D, E, F\}$. Jaon g negatiivinen puoli saadaan jaettua loppuun asti ongelmitta valitsemalla jakomonikulmioksi E, mutta jos positiivisella puolella valitaan jakomonikulmioksi B, joudutaan monikulmio C jakamaan osiin C_1 ja C_2. Jakolinjan b negatiiviselle puolelle jää vain yksi monikulmio C_2, joten jaettavaksi jää vain b:n positiivinen puoli. Valitsemalla viimeiseksi jakomonikulmioksi A syntyy kuvan 3 mukainen BSP-puu.



(a) Joukko monikulmioita tasossa

(b) Taso neljän jaon jälkeen

Kuva 2: Tason jakaminen

BSP-puun kokoon ja muotoon vaikuttaa suuresti avaruuden jakavan monikulmion valinta. Pahimmassa tapauksessa kaikki monikulmio $\Gamma \backslash \{\gamma_k\}$ jäävät monikulmion γ_k positiiviselle tai negatiiviselle puolelle jokaisella jaolla k. Tällöin puusta muodostuu pikemminkin ketjun muotoinen. Sama monikulmio voi myös kuulua moneen BSP-puun alipuuhun, jos jonkun ylemmällä tasolla avaruuden jakavan

Input:

`BSP_Node solmu`: juurisolmu, josta puu rakennetaan

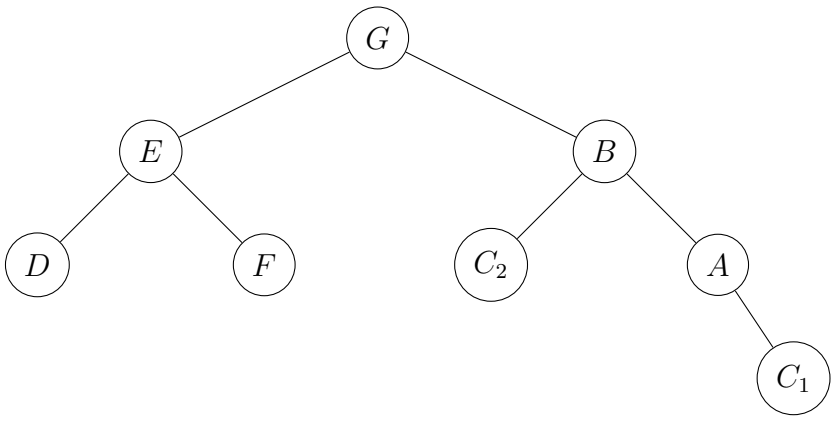
`Polygon monikulmiot[]`: monikulmiojoukko, josta alipuut rakennetaan

Output:

BSP-puu, jonka juurena on solmu `node`

```
   RAKENNA_BSP_PUU(solmu, monikulmiot):
 1 jakaja ← VALITSE_JAKAVA_MONIKULMIO(monikulmiot)
 2 positiivinen_joukko ← ∅
 3 negatiivinen_joukko ← ∅
 4 foreach γ ϵ monikulmiot do
 5 |   sijainti ← VERTAA(γ, jakaja)
 6 |   if sijainti = jakajan edessä then
 7 |   |   positiivinen_joukko = positiivinen_joukko ∪ γ
 8 |   else if sijainti = jakajan takana then
 9 |   |   negatiivinen_joukko = negatiivinen_joukko ∪ γ
10 |   else if sijainti = leikkaa jakajan määrittämää tasoa then
11 |   |   JAA_MONIKULMIO(γ, jakaja, Γ_{jakaja,+}, Γ_{jakaja,−})
12 |   |   positiivinen_joukko = positiivinen_joukko ∪ Γ_{jakaja,+}
13 |   |   negatiivinen_joukko = negatiivinen_joukko ∪ Γ_{jakaja,−}
14 |   end
15 end
16 if positiivinen_joukko ≠ ∅ then
17 |   RAKENNA_BSP_PUU(solmu.oikea_lapsi, positiivinen_joukko)
18 end
19 if positiivinen_joukko ≠ ∅ then
20 |   RAKENNA_BSP_PUU(solmu.vasen_lapsi, negatiivinen_joukko)
21 end
```

Algoritmi 2: `RAKENNA_BSP_PUU` [Ranta-Eskola, 2001]



Kuva 3: Tasosta muodostettu BSP-puu

monikulmion γ_k muodostama jakolinja leikkaa tätä monikulmiota. [Samet, 2005.] Toinen lähestymistapa jakolinjalla oleviin monikulmioihin on halkaista ne osiin. Tämäkin tapa on epäedullinen, sillä se lisää maisemassa olevien monikulmioiden määrää. [Ranta-Eskola, 2001.]

BSP-puuta rakennettaessa tavoitteena on muodostaa mahdollisimman tasapainoinen binääripuu valitsemalla jokaisella jakokerralla jakajaksi sellainen monikulmio γ_k, jonka positiivisella ja negatiivisella puolella on likimain yhtä paljon monikulmioita, eli $|\Gamma_{k,+}| \approx |\Gamma_{k,-}|$. Tällöin n:n monikulmion joukousta muodostetun BSP-puun syvyys olisi $O(\log n)$, mikäli rekursiivista jakoa jatketaan kunnes jokainen monikulmio on omassa lehdessään. Koska puun solmuja syntyy lisää, kun jakolinjan leikkaavat monikulmiot jaetaan osiin, tai jakolinjalla oleva monikulmio sisällytetään useaan alipuuhun, voidaan puun logaritmista tavoitesyvyyttä pitää vain alarajana. [Hughes et al., 2013.] n monikulmiota sisältävästä maisemasta rakennetun BSP-puun syvyys on siis $\Omega(\log n)$.

BSP-puu voidaan rakentaa ennen hahmontamista esiprosessointivaiheessa. Hahmontamisvaiheessa sitä voidaan käyttää vähentämään säde-monikulmio leikkaustestien määrää. Kamerasta ammuttua sädettä verrataan ensin BSP-puun juurena toimivaan monikulmioon. Jos säde leikkaa monikulmion γ_k muodostaman avaruuden jakavan tason, säde voi osua johonkin monikulmioon molemmissa joukoissa $\Gamma_{k,+}$ ja $\Gamma_{k,-}$, eli joudutaan tarkastelemaan molempia alipuita. Jos säde ei leikkaa jakotasoa, siirrytään tarkastelemaan vain toista alipuuta. Säteen $\vec{R} = O + t\vec{D}$ osumista tasoon T voidaan testata laskemalla etäisyys $t = \dfrac{-\vec{n} \cdot (O - Q_0)}{\vec{n} \cdot \vec{D}}$, missä $\vec{n}$ on tason T normaalivektori ja Q_0 on jokin tason T piste. Säde $\vec{R}$ osuu tasoon T

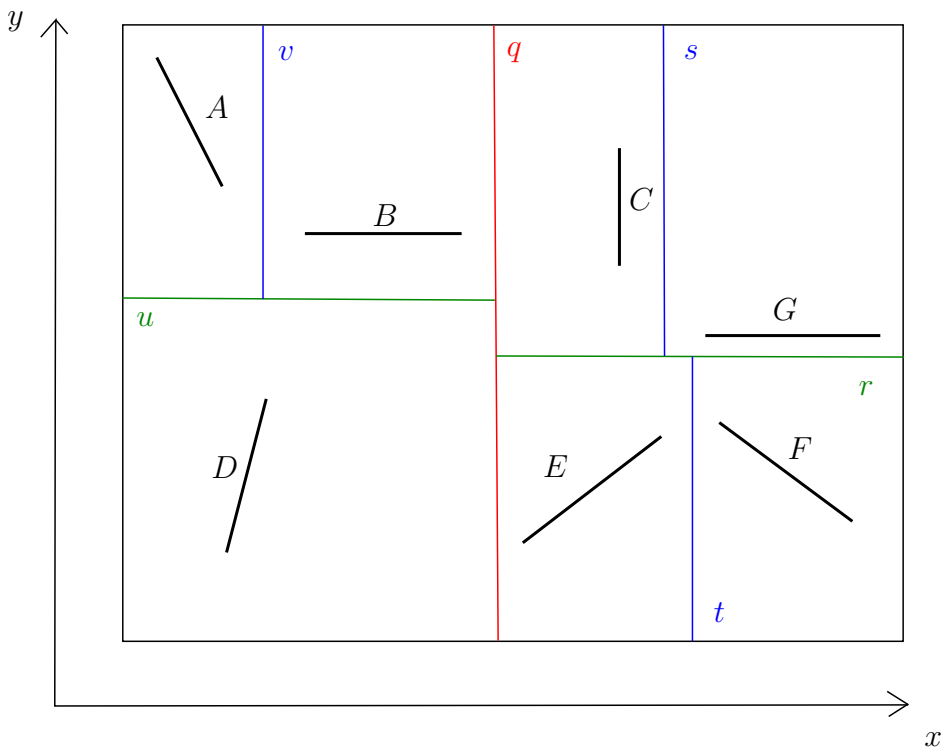
jos sijoittamalla t säteen yhtälöön, on piste $Q = O + t\vec{D}$ tasossa T. Tämä pitää paikkansa jos $(Q - Q_0) \cdot \vec{n} = 0$. [Hughes et al., 2013.] Toistamalla säteiden ja avaruuden jakavien tasojen leikkauksia rekursiivisesti, päädytään lopulta BSP-puun juureen ja voidaan testata, osuuko säde monikulmioihin. [Ranta-Eskola, 2001.]

3.1.2 kd-puu

BSP-puun erikoistapaus on *kd-puu*, eli k-ulotteinen puu (engl. *k-dimensional tree*). kd-puussa avaruuden jakavaa tasoa ei valita jakavan monikulmion muodostaman sivun mukaan, vaan jaot tehdään siten, että jakotaso on samansuuntainen jonkin koordinaattiakselin kanssa. Yleisin tapa muodostaa kd-puu kolmiulotteisen maiseman monikulmioista on valita jakotaso vuorotellen maailmakoordinaatiston x-, y- ja z-akseleiden suuntaiseksi. kd-puun jokaiseen solmuun tallennetaan jakavan monikulmion sijaan jakava taso. Tällöin monikulmioiden jakaminen solmun lapsiin onnistuu helposti vertaamalla niiden sijaintia jakotasoon. Esimerkiksi jos avaruuden jakava taso valitaan samansuuntaiseksi z-akselin kanssa, tallennetaan solmun vasempaan lapseen monikulmiot, joiden sijainnin x-koordinaatin arvo on pienempi kuin jakotason. Oikeaan lapseen taas tallennetaan monikulmiot, jotka ovat jakotason positiivisella puolella x-akselin suhteen. [Samet, 2005.] Avaruutta jaetaan rekursiivisesti osiin kunnes lehtisolmuissa on jokin ennaltamäärätty määrä monikulmioita. kd-puu muodostetaan vastaavalla algoritmilla kuin BSP-puun rakentamista kuvaava algoritmi 2.

Yleisen BSP-puun tapaan myös kd-puuta rakennettaessa avaruuden jakavien tasojen valinta vaikuttaa siihen, kuinka tehokkaasti puuta voidaan hyödyntää hahmontamisessa. Jakotaso voidaan valita aina jakamalla avaruuden osa tasan kahtia, jolloin samalla syvyydellä puussa olevat solmut vastaavat saman kokoista laatikkoa. Tämä tapa ei takaa sitä, että jakotason molemmille puolille jäisi saman verran monikulmioita, jolloin puusta voi tulla epätasapainoinen. Hahmontamisen kannalta parempi tapa on käyttää esiprosessointivaiheessa enemmän laskenta-aikaa ja valita jakotaso siten, että sen molemmille puolillä jää likimain yhtä paljon monikulmioita. Sellaiset monikulmiot, jotka leikkaavat avaruuden jakavaa tasoa, täytyy sisällyttää puuhun useaan solmuun tai jakaa ne pienempiin osiin. [Havran, 2000.]

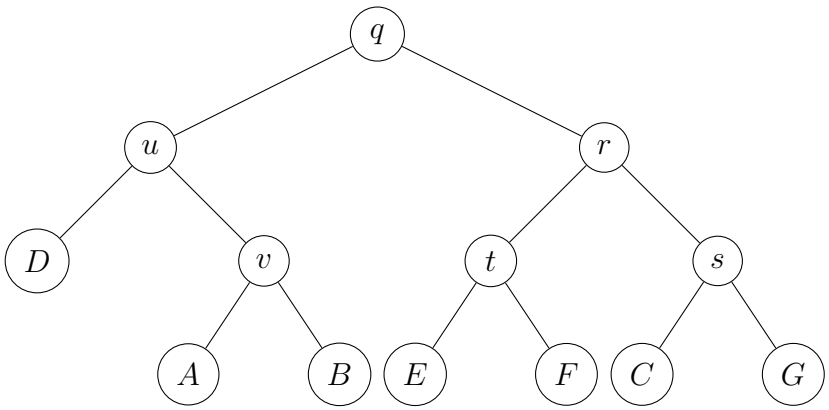
Eräs mahdollinen tapa jakaa taso osiin koordinaattiakselien suuntaisesti



Kuva 4: Kuvan 2a taso jaettuna kuusi kertaa koordinaattiakselien suuntaisesti

on esitetty kuvassa 4, jossa tarkastellaan kuvan 2a tapaan monikulmiot $\Gamma = \{A, B, C, D, E, F, G\}$ sisältävää kaksiulotteista tasoa. Ensin taso jaetaan puoliksi y-akselin suuntaisella suoralla q. Jaon positiiviselle puolelle jäävät monikulmiot $\Gamma_{q,+} = \{C, G, E, F\}$. Kun jaetaan suoran q positiivinen puoli x akselin suuntaisella jaolla r ja edelleen y akselin suuntaisilla jakosuorilla s ja t, on monikulmiot C, G, E ja F saatu omiin lehtiinsä. Vastaava jako suoritetaan suoran q negatiivisella puolella oleville monikulmioille $\Gamma_{q,-} = \{A, B, C\}$ jakosuorilla u ja v, jolloin tason monikulmioista saadaan muodostettua kuvan 5 mukainen kd-puu.

Muodostamalla esiprosessointivaiheessa maiseman monikulmioista BSP-puun sijaan kd-puu voidaan nopeuttaa säteiden ja avaruuden osien yhteentörmäystestejä. Koska kd-puussa avaruuden jakavat tasot ovat kohtisuorassa koordinaattiakselia a vasten, $a \epsilon (x, y, z)$, ja jakosuunta on joka solmussa tiedossa, voidaan jakotaso esittää vain yhdellä arvolla a_p. Tällöin säteen ja jakotason yhteentörmäyksen testaaminen on noin kolme kertaa nopeampaa kuin satunnaisesti suunnatulle tasolle $ax + by + cz + d = 0$. [Havran, 2000.]



Kuva 5: Tasosta muodostettu kd-puu

Muita BSP-puun erikoistapauksia ovat *quad-* ja *oct-puut*. Quad-puuta muodostettaessa avaruuden osa jaetaan aina neljään yhtä suureen kuutioon kunnes jokin pysähtymisehto saavutetaan. Oct-puun tapauksessa kuutioiden määrä on kahdeksan [Samet, 2005.]. Lukuunottamatta ennaltamäärättyä avaruusjakojen määrää quad- ja oct-puut ovat kuten yllä määritellyt kd-puut.

3.2 Rajaavien tilojen hierarkia

BSP-puuta objektilähtöisempi tapa jakaa kolmiulotteista avaruutta osiin on määrittää jokaiselle objektille *rajaava tila* (engl. *bounding volume*). Objektin O rajaava tila on jokin yksinkertainen kolmiulotteinen muoto V, jolle $O \cap V \equiv O$ [Havran, 2000]. Useimmiten rajaavan tilan muodoksi valitaan pallo tai koordinaattiakseleiden suuntainen laatikko, sillä säteen osumista niihin on helppo testata.[3] Rajaavat tilat tulisi valita siten, että ne rajaavat objekteja tarkasti, jättämättä liikaa tyhjää tilaa itsensä ja objektin väliin. [Hughes et al., 2013]

Rajaavien tilojen hierarkia, eli *BVH* (engl. *Bounding Volume Hierarchy*) on puu, jonka juurena on koko maiseman tilavuus ja solmuissa pienempiä, yhden tai useamman objektin rajaavia tiloja. Puun solmuihin on tallennettu tieto rajaavan tilan muodosta, koosta ja sijainnista, sekä osoittimet lapsisolmuihin. Lehtisolmut ovat pienimpiä rajaavia tiloja, jotka sisältävät yhden objektin tai joissakin tapauksissa osan siitä. BVH voidaan muodostaa rakentamalla ensin maisemasta BSP-puu ja sen jälkeen määrittää ympäröiviä tiloja objekteille rekursiivisesti lehtisolmuista ylöspäin [Hughes et al., 2013]. Toinen tapa on käydä esiprosessointivaiheessa läpi

[3]Säteen osumatesti kummankin muodon kanssa vaatii noin kymmenen laskutoimitusta [Goldsmith ja Salmon, 1987].

Input:
`monikulmiot`: monikulmiojoukko, josta alipuut rakennetaan
Output:
`BVH_Node`: BVH:n sisäsolmu

`RAKENNA_BVH(monikulmiot)`:

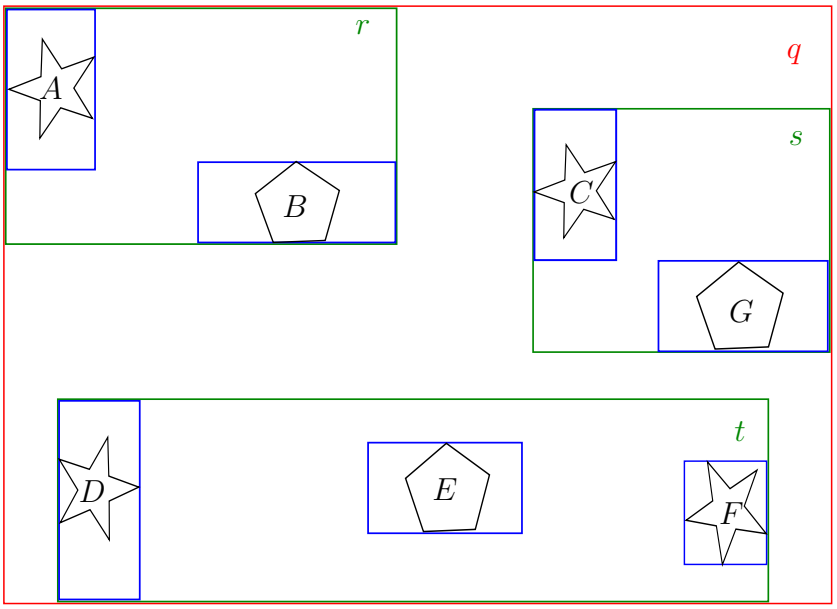
```
1  if monikulmiot sisältää vain yhteen objektiin kuuluvia monikulmioita, tai
     ennalta määrätyn vähimmäismäärän monikulmoita then
2  |  return lehtisolmu, joka sisältää monikulmiot
3  else
4  |  Selvitä taso joka jakaa objektit kahteen, likimain yhtä suureen osaan
5  |  BVH_Node solmu
6  |  solmu.vasen_lapsi ← RAKENNA_BVH(monikulmiot jaon vasemmalla
   |    puolella)
7  |  solmu.oikea_lapsi ← RAKENNA_BVH(monikulmiot jaon oikealla
   |    puolella)
8  |  solmu.rajaava_tila ← pienin rajaava tila, joka sisältää kaikki
   |    monikulmiot
9  |  return solmu
10 end
```

Algoritmi 3: `RAKENNA_BVH [Thrane ja Simonsen, 2005]`

kaikkien objektien monikulmioiden kärjet ja ottaa talteen maksimi- ja minimiarvot x_{max}, x_{min}, y_{max}, y_{min}, z_{max} ja z_{min}. Näiden arvojen avulla voidaan muodostaa rajaava tila, jonka sisään kaikki objektin monikulmiot jäävät. [Janke, 2015.] Algoritmissa 3 on esitetty eräs tapa alustaa BVH.

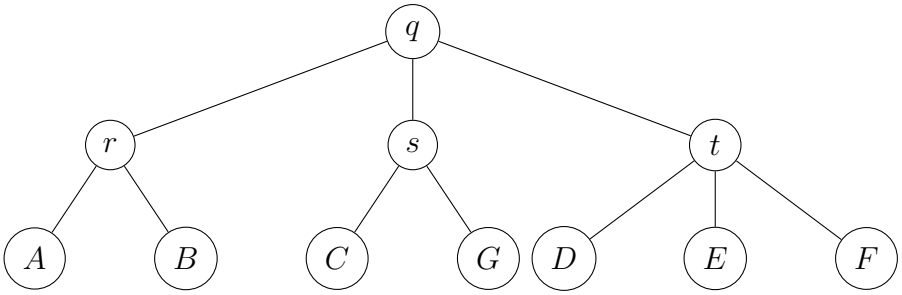
BVH:n hyödyntäminen hahmontamisessa säteenseurantatekniikalla on yksinkertaista. Kamerasta ammuttavien säteiden osumia testataan ensin BVH:n juureen, minkä jälkeen siirrytään rekursiivisesti molempiin lapsisolmuihin aina kun säde osuu solmuun. Jos käytössä on koordinaattiakselien suuntaiset laatikot, osuu säde $\vec{R} = O + t\vec{D}$ laatikkoon kun se leikkaa laatikon kaksi sivua. Säteen ja tason osumatestiä on kuvailtu luvussa 3.1. Vertailemalla säteen $\vec{R}$ leikkauspisteitä rajaavan tilan sivujen kanssa saadaan etäisyydelle t välit $[t_{x_0}, t_{x_1}]$, $[t_{y_0}, t_{y_1}]$ ja $[t_{z_0}, t_{z_1}]$, joissa alarajat ovat säteen ensimmäisiä ja ylärajat sen toisia leikkauspisteitä laatikon kanssa. Jos jokin arvo t kuuluu kaikkiin näihin väleihin, lävistää säde $\vec{R}$ laatikon, ja siten siirrytään tarkastelemaan joko laatikon sisällä olevia rajaavia tiloja tai sen

sisältämiä monikulmioita. [Janke, 2015.]



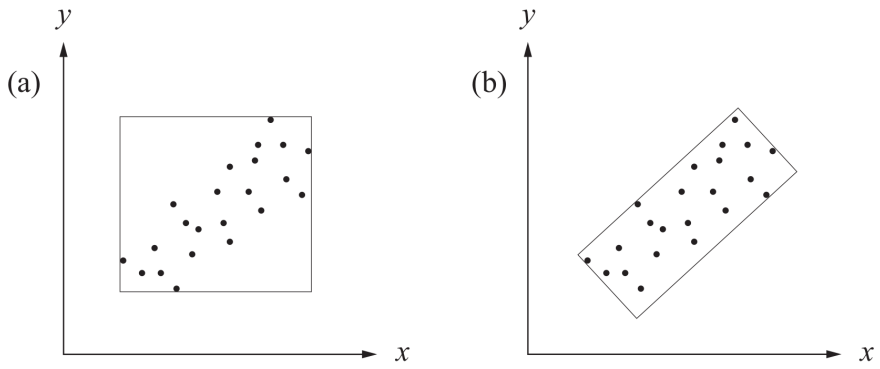
Kuva 6: Objektijoukko jaettuna rajaaviin tiloihin

Kuvassa 6 on esimerkki objektijoukosta $\{A, B, C, D, E, F, G\}$ tasossa. Tasolle on ensin määritelty rajaava tila q, joka sisältää kaikki objektit. Tilan q sisällä on pienempiä objektiosajoukkoja sisältävät tilat r,s ja t. Lopuksi jokaiselle objektille on määritelty omat rajaavat tilansa. Objektijoukosta muodostettu BVH-puu on esitetty kuvassa 7.



Kuva 7: Objektijoukosta muodostettu BVH-puu

Kuten BSP- ja kd-puun tapauksissa, myös BVH:n tuoma etu hahmontamisessa riippuu siitä, miten hyvin jako rajaaviin tiloihin onnistuu. Kuvassa 8 havainnollistetaan rajaavan tilan valinnan merkitystä. Kuvassa on pisteistä kaksiulotteisessa tasossa ja kaksi vaihtoehtoa niitä rajaavaksi tilaksi. Koordinaattiakseleiden suuntainen laatikko (a) ei ole tässä tapauksessa optimaalinen, vaan jättää pisteiden ja



Kuva 8: Kaksi vaihtoehtoa pistejoukon rajaavaksi tilaksi [Lengyel, 2012]

laatikoiden väliin liikaa tyhjää tilaa. Tällöin monet laatikon lävistävät säteet eivät osu itse pisteisiin. Tiiviimpi laatikko on esitetty vieressä (b).

Säteen $\vec{R} = O + t\vec{D}$ ja pallon S leikkausta voidaan tarkastella määrittämällä ensin vektori $\vec{a} = C - O$, missä C on pallon S keskipiste. Tällöin saadaan suorakulmainen kolmio, jossa $\vec{a}$ on hypotenuusa ja $\frac{\vec{a} \cdot \vec{v}}{|\vec{v}|}$ on kateetti. Laskemalla Pythagoraan lauseella toisen kateetin pituus $d = \sqrt{\vec{a}^2 - \left(\frac{\vec{a} \cdot \vec{v}}{|\vec{v}|}\right)^2}$ saadaan selville mikä on pienin säteen ja pallon keskipisteen välinen etäisyys. Jos $|d| < r$, missä r on pallon S säde, lävistää säde R pallon ja voi osua sen sisältämiin monikulmioihin. [Janke, 2015.]

4 Avaruusjakopuiden vertailua säteenseurannan kannalta

Tässä luvussa vertaillaan luvussa 3 esiteltyjä tietorakenteita ja selvitetään, mitä eroja niissä on säteenseurannan nopeuttamisen kannalta. Lisäksi selvitetään mitä ongelmia tietorakenteiden alustamiseen liittyy ja miten ongelmat voidaan selvittää. Vertailussa keskitytään binääriseen avaruusjakoon kd-puun avulla ja objektien ryhmittelyyn BVH:n avulla. Yleisen BSP-puun sijaan tarkastellaan kd-puuta sen yksinkertaisuuden ja sitä käsittelevän tutkimuksen määrän vuoksi.

4.1 Tietorakenteiden alustaminen

Rakennettaessa kd-puuta on otettava kantaa, mitä tehdään monikulmioille, jotka ulottuvat avaruuden jakavan tason molemmille puolille. Sekä monikulmion jakaminen osiin, että sen sisällyttäminen useaan solmuun kasvattavat kd-puuta turhaan. kd-puusta saatava hyöty on suurin silloin, kun avaruusjakoja on tehty mahdollisimman paljon ja lehtisolmuissa on mahdollisimman vähän monikulmioita. Tällöin todennäköisyys sille, että monikulmioita jää jakotasojen molemmin puolin on suuri ja tietorakenteen vaatima tallennustila kasvaa merkittävästi. [Wald, 2004.]

BVH:ta muodostettaessa ei synny samankaltaista ongelmaa kuin kd-puiden kanssa, sillä monikulmio voi kuulua vain yhteen objektiin ja objektin rajaava tila sisältää kaikki sen monikulmiot. Myös BVH-puu kannattaa muodostaa syväksi. Jos BVH sisältäisi vain koko maiseman kattavan juuren, jonka lapsina olisi kaikkien objektien rajaavat tilat, ei tietorakenteen tuoma hyöty olisi paras mahdollinen. Puuhun kannattaa siis sijoittaa useita rajaavia tiloja, jotka sisältävät useita objekteja. Objektijoukon jakaminen kahtia BVH:n jokaisella tasolla vaikuttaa olevan hyvä valinta. [Goldsmith ja Salmon, 1987.]

Kuten aikaisemmin mainittu, avaruusjakorakenteen tuoma hyöty riippuu siitä, miten hyvin avaruuden ositus on onnistunut. Sekä kd-puun, että BVH:n alustamisessa voidaan hyödyntää samankaltaista hintafunktiota, jolla voidaan minimoida puun läpikäymisen viemä aika. Hintafunktiota voidaan kuvata seuraavasti: olkoon alipuussa N monikulmiota, jotka peittävät alueen V. Tämä alipuu jaetaan osiin L ja R, joissa on N_L ja N_R monikulmiota, jotka peittävät alueen V_L ja V_R. Tällöin alipuun läpikäymisen hinta on

$$Hinta(V \to \{L, R\}) = K_T + K_I \left(\frac{A(V_L)}{A(V)} N_L + \frac{A(V_R)}{A(V)} N_R \right),$$

missä $A(V_i)$ on alueen V_i pinta-ala, ja arvot K_T ja K_I ovat puun läpikäynnistä ja säde-monikulmio-yhteentörmäystesteistä johtuvia vakioita. Minimoimalla tämä funktio puun alustuksen jokaisessa vaiheessa, saadaan muodostettua tehokas avaruuden jako. [Wald, 2007.]

Edellä mainittua, hintafunktiota käyttävää tekniikkaa kutsutaan *pinta-alaheuristiikaksi* (engl. *Surface Area Heuristics*) ja sillä saadaan muodostettua hyvä avaruuden ositus sekä kd-puulle, että BVH:lle. Ingo Waldin tekemässä tutkimuksessa BVH:n muodostaminen onnistui jopa 10 kertaa nopeammin kuin kd-puun. Tämä johtuu muun muassa kd-puun alustamisessa esiintyvistä, jakotason leikkaavista monikulmioista. [Wald, 2007.]

4.2 Tietorakenteiden hyödyntäminen hahmontamisessa

Avaruusjakorakenteiden hyödyntäminen säteenseurannassa on hyvin samankaltaista riippumatta siitä, millä periaatteella tietorakenne on muodostettu. Algoritmissa 4 on kuvattu yksinkertaistettu pseudokoodi, jolla voidaan testata säteen osumista avaruusjakopuuna kuvattuun maisemaan.

Vlastimil Havran vertaili väitöskirjassaan eri tietorakenteita säteenseurannan nopeuttamisen kannalta. Havranin testeissä pinta-alaheuristiikan avulla muodostettu kd-puu osoittautui yleisesti tehokkaimmaksi tietorakenteeksi, vaikka joissakin tapauksissa muut tekniikat tuottivat nopeamman hahmonnuksen. BVH:n avulla suoritettu säteenseuranta tuotti pienimmän hyödyn kaikista vertailluista rakenteista. Havran kuitenkin myöntää, että yhtä optimaalista tietorakennetta kaikkien maisemien hahmontamiseen tuskin löytyy. [Havran, 2000.]

Niels Thrane ja Lars Simonsen vertailivat pro gradu -työssään eri avaruusjakorakenteita grafiikkaprosessoreilla suoritetussa säteenseurannassa. Koska säteenseurannassa jokainen säde on riippumaton muista, voidaan säteenseuranta-algoritmi suorittaa helposti rinnakkaislaskentana. Tämä onnistuu grafiikkaprosessoreilta erittäin nopeasti. Toisin kuin Havranin testeissä, Thranen ja Simonsenin vertailussa BVH:n käytöllä saatiin yhdeksänkertainen nopeutus kd-puuhun verrattuna. [Thrane ja Simonsen, 2005.]

Input:
$\vec{R}$: kamerasta ammuttu säde
`Solmu s`: jonkin avaruusjakopuun solmu
Output:
Kolmio, johon säde $\vec{R}$ osui

```
SÄDE_PUU_YHTEENTÖRMÄYS(R, s):
if R osui solmun s kuvaamaan avaruuden osaan then
    if s on lehtisolmu then
        return SÄDE_MONIKULMIO_YHTEENTÖRMÄYS (R, s.monikulmiot)
    end
    SÄDE_PUU_YHTEENTÖRMÄYS(R, s.oikea_lapsi)
    SÄDE_PUU_YHTEENTÖRMÄYS(R, s.vasen_lapsi)
else
    return NULL
end
```

Algoritmi 4: `SÄDE_PUU_YHTEENTÖRMÄYS`

Yksiselitteisesti ei voida siis osoittaa, mikä avaruusjakoon perustuvista tietorakenteista soveltuisi parhaiten säteenseurannan nopeuttamiseen. Asymptoottisesti avaruusjakopuiden käyttö säteenseurassa vähentää tarvittavien yhteentörmäystestien määrää luokasta $O(n)$ keskimäärin luokkaan $O(\log n)$, riippumatta käytetystä tietorakenteesta. Tulokset hahmontamisen nopeudessa voivat vaihdella käytännössä eri maisemien, implementaatioiden, laitteistojen ja sovelluksien välillä. [Wald, 2004.]

5 Muita nopeutuskeinoja

5.1 Reaaliaikainen säteenseuranta

Kuten luvussa 2.2 mainittiin, nopeaa hahmontamista vaativiin sovelluksiin käytetään yleensä rasterointitekniikkaa. Säteenseuranta on kuitenkin ylivoimainen hahmontamistekniikka johtuen sen kyvystä mallintaa realistista valon liikettä ja heijastumista. Reaaliaikaista säteenseurantaa on pitkään pidetty mahdottomana, sillä se vaatii vähintään 30 kuvan hahmontamista sekunnissa, jottei ihmissilmä erottaisi yksittäisiä kuvia. Tähän sisältyy jokaista pikseliä vastaavan säteen ampuminen, yhteentörmäysten etsiminen, varjostus ja kuvan tulostaminen ruudulle.

Ingo Wald osoittaa väitöskirjassaan staattisille maisemille suoritetun reaaliaikaisen säteenseurannan olevan mahdollista hyvin muodostetun avaruusjakotietorakenteen ja lukuisten muiden optimointien avulla [Wald, 2004]. Jos objektit liikkuvat hahmonnettaessa reaaliaikaisesti maisemaa, joudutaan avaruusjakorakennetta muokkaamaan kuvien välissä. Wald ja Havran ovat näyttäneet artikkelissaan, että uutta maisemaa vastaava kd-puu voidaan muodostaa ajassa $O(n \log n)$, missä n on monikulmioiden määrä [Wald ja Havran, 2006]. Tämä ei kuitenkaan ole tarpeeksi nopeaa. Wald et al. on myös esittänyt käytettäväksi topologialtaan staattista BVH:ta, jonka rajaavien tilojen koordinaatteja muutetaan objektien liikkuessa [Wald et al., 2007]. Vaikka tulokset ovat olleet lupaavia, on reaaliaikaiseen, liikkuvien maisemien hahmontamiseen säteenseurannalla vielä matkaa.

5.2 Joitakin optimointitekniikoita

Monimutkaisen maiseman reaaliaikainen hahmontaminen vaatii toki muitakin optimointitekniikoita kuin avaruusjakorakenteet onnistuakseen kuluttajahintaisella laitteistolla. Luodaan lopuksi katsaus muutamaan tekniikkaan.

Säteenseurannassa säde $\vec{R}$ voi osua useaan monikulmioon, mutta kiinnostuneita ollaan vain kameraa lähinnä olevasta leikkauspisteestä. Jos avaruuden osat eli vokselit on esiprosessointivaiheessa järjestetty kasvavaan järjestykseen suhteessa niiden etäisyyteen kamerasta, voidaan säteen ja monikulmioiden leikkauspisteiden etsintä lopettaa heti ensimmäisen leikkauksen löytyessä (engl. *Early Ray Termination*). Tämä tekniikka vaatii kuitenkin enemmän laskentaa esiprosessointivaiheessa. [Wald, 2004]

Pullonkaulaksi säteenseurannassa voi muodostua monikulmioiden ja muun datan hakeminen keskusmuistista. Datan hakeminen cache-muistista on moninkertaisesti nopeampaa kuin keskusmuistista, joten usein tarvittua dataa, kuten säde-monikulmio-yhteentörmäystesteihin vaadittuja laskutoimituksia, olisi hyvä hakea cache-muistiin etukäteen ja pitää se siellä niin kauan kuin tarvitaan. Tämä vaatii erittäin matalan tason optimointia. [Wald et al., 2001.]

Toinen tapa välttää muistihausta aiheutuvaa latenssia on ampua maisemaan yksittäisten säteiden sijaan niin sanottuja *sädepaketteja* (engl. *Ray Packets*). Monet nykyaikaiset prosessorit tukevat SIMD-komentoja (*Single Instruction, Multiple Data*), joiden avulla voidaan suorittaa laskutoimituksia samanaikaisesti kaikille paketin säteille. [Wald et al., 2001.]

6 Yhteenveto

Kolmiulotteisten kuvien hahmontaminen on tärkeä tutkimuskohde ja viihdeteollisuuden ala. Tässä tutkielmassa on esitelty säteenseurannaksi kutsuttu hahmontamistekniikka, jolla pystytään luoda erittäin realistisia, kolmiulotteisesta maisemasta muodostettuja kuvia. Säteenseuranta-algoritmi ampuu virtuaalisesta kamerasta valon kulkua mallintavia säteitä geometrisesti määriteltyyn maisemaan ja selvittää, mitkä maiseman objektit näkyvät kameraan ja minkä värisinä. Säteenseuranta on laskennollisesti erittäin raskasta, joten sen nopeuttamista on tutkittu paljon.

Suosittu tapa optimoida säteenseuranta-algoritmia on luoda esiprosessointivaiheessa maisemasta hierarkinen tietorakenne. Avaruutta voidaan jakaa osiin tilan suhteen muodostamalla siitä esimerkiksi BSP- tai kd-puu. Toinen tapa on ryhmitellä maiseman objekteja yhteen, ja muodostaa niiden rajaavista tiloista hierarkinen puurakenne, BVH. Näiden tietorakenteiden avulla laskentatyötä saadaan siirrettyä itse hahmontamisvaiheesta esiprosessointivaiheeseen.

Avaruuden jakaminen tulee suorittaa huolellisesti, jotta esiprosessointivaiheessa muodostetusta tietorakenteesta olisi hyötyä. Laskemalla hintafunktio jokaista potentiaalista puuhun lisättävää alipuuta kohden voidaan valita avaruusjakopuuhun aina sellainen alipuu, jonka läpikäyminen on mahdollisimman nopeaa. Oikein muodostettuna avaruusjakorakenne vähentää tarvittavien säde-monikulmioyhteentörmäystestien määrää luokasta $O(n)$ luokkaan $O(\log n)$, mikä on merkittävä parannus maisemassa, jossa voi olla satoja tuhansia tai miljoonia monikulmioita.

Eri tutkimuksissa on saatu ristiriitaisia tuloksia siitä, mikä tietorakenne olisi tehokkain säteenseurannan nopeuttamiseen. Voidaan kuitenkin sanoa kd-puun ja BVH:n olevan soveltuvia tähän tarkoitukseen ja yhdistettynä grafiikkaprosessorien tarjoamaan tehokkaaseen rinnakkaisuuteen ja muihin optimointitekniikoihin, niillä voidaan saavuttaa reaaliaikaiseen hahmontamiseen vaadittava nopeus. Säteenseuranta tulee todennäköisesti tuomaan erittäin realistista grafiikkaa myös sovelluksiin, jotka on aiemmin hahmonnettu rasterointitekniikalla, esimerkiksi videopeleihin.

Viitteet

Angel, E. ja Shreiner, D. (2014). *Interactive Computer Graphics with WebGL.* Addison-Wesley Professional, 7th edition.

Appel, A. (1968). Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), sivut 37–45, New York, NY, USA. ACM.

Fuchs, H., Kedem, Z. M., ja Naylor, B. F. (1980). On visible surface generation by a priori tree structures. *SIGGRAPH Comput. Graph.*, 14(3):124–133.

Goldsmith, J. ja Salmon, J. (1987). Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, 7(5):14–20.

Harju, T. (1989–2015). Geometria, lyhyt kurssi, Turun yliopisto.

Havran, V. (2000). *Heuristic Ray Shooting Algorithms.* väitöskirja, Czech Technical University, Praha, Tšekki.

Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., ja Akeley, K. (2013). *Computer graphics: principles and practice (3rd ed.).* Addison-Wesley Professional, Boston, MA, USA.

Janke, S. J. (2015). *Mathematical Structures for Computer Graphics.* John Wiley & Sons, Inc., New York, NY, USA.

Kajiya, J. T. (1986). The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150.

Lengyel, E. (2012). *Mathematics for 3D Game Programming and Computer Graphics.* Course Technology PTR.

Möller, T. ja Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28.

Parkin, K. (2014). Building 3d with ikea. `http://www.cgsociety.org/index.php/cgsfeatures/cgsfeaturespecial/building_3d_with_ikea`. Luettu: 28.10.2016.

Ranta-Eskola, S. (2001). Binary space partitioning trees and polygon removal in real time 3d rendering. pro gradu -työ, Uppsalan yliopisto, Uppsala, Ruotsi.

Rubin, S. M. ja Whitted, T. (1980). A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph.*, 14(3):110–116.

Samet, H. (2005). *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Takahashi, D. (2013). How pixar made monsters university, its latest technological marvel. `http://venturebeat.com/2013/04/24/the-making-of-pixars-latest-technological-marvel-monsters-university/view-all/`. Luettu 30.10.2016.

Thrane, N. ja Simonsen, L. O. (2005). A comparison of acceleration structures for gpu assisted ray tracing. pro gradu -työ, Aarhusin yliopisto, Aarhus, Tanska.

Wald, I. (2004). *Realtime Ray Tracing and Interactive Global Illumination*. väitöskirja, Saarlandin yliopisto, Saarbrücken, Saksa.

Wald, I. (2007). On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, sivut 33–40, Washington, DC, USA. IEEE Computer Society.

Wald, I., Boulos, S., ja Shirley, P. (2007). Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1).

Wald, I. ja Havran, V. (2006). On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *2006 IEEE Symposium on Interactive Ray Tracing*, sivut 61–69.

Wald, I., Slusallek, P., Benthin, C., ja Wagner, M. (2001). Interactive rendering with coherent ray tracing. In *Computer Graphics Forum*, sivut 153–164.

Whitted, T. (1980). An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349.