

Avaruusjakoon perustuvat tietorakenteet  
tietokonegrafiikassa

Timo Heinonen  
kandidaatintutkielma  
marraskuu 2016

Informaatioteknologian laitos  
tietojenkäsittelytiede  
Turun yliopisto

# Tiivistelmä

TURUN YLIOPISTO

Informaatioteknologian laitos

HEINONEN, TIMO: Avaruusjakoon perustuvat tietorakenteet tietokone-  
grafikassa

kandidaatintutkielma, xx s., yy liites.

Tietojenkäsittelytiede

Marraskuu 2016

---

Asiasanat: Tietokonegrafikka, Ray Tracing, binääripuut.

# Sisällys

<b>1</b>	<b>Johdanto</b>	<b>2</b>
<b>2</b>	<b>Kolmiulotteisen tietokonegrafiikan peruskäsitteitä</b>	<b>4</b>
2.1	Määritelmiä . . . . .	4
2.2	Ray Tracing -tekniikka . . . . .	5
<b>3</b>	<b>Avaruusjakopuut</b>	<b>8</b>
3.1	Binäärinen avaruusjako . . . . .	8
3.1.1	BSP-puu . . . . .	8
3.1.2	kd-puu . . . . .	11
3.1.3	quad- ja oct-puu . . . . .	14
3.2	Rajaavien tilojen hierarkia . . . . .	14
<b>4</b>	<b>Avaruusjakopuiden vertailua</b>	<b>16</b>
<b>5</b>	<b>Yhteenveto</b>	<b>16</b>
	<b>Viitteet</b>	<b>17</b>

# 1 Johdanto

Kolmiulotteisen tietokonegrafiikan tutkimuksella on ollut merkittävä vaikutus viihdeteollisuuteen, kuten animaatioelokuvaan, peleihin ja virtuaalitodellisuuteen, sekä tietokoneavusteiseen suunnitteluun, esimerkiksi arkkitehtuurin ja teollisuuden alalla. [Wald, 2004] Tietokonegrafiikka on osin jopa syrjäyttämässä perinteistä valokuvaustyötä: huonekalujätti Ikea on siirtynyt käyttämään myyntikuvastoissaan valtaosin tietokoneella generoituja kuvia valokuvien sijaan [CGSociety, 2014]. Tietokonegrafiikan sovelluskohdeet 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 laske-  
malla suoran yhtälöitä kuvasta maisemaan ja siitä valonlähteisiin. Tämän tekniikan avulla voitiin piirtää yksinkertaisia varjostuksia. [Appel, 1968.]. Myöhemmin tästä Ray Tracing -nimisestä tekniikasta tuli erittäin suosittu.

Jo Appel totesi Ray Tracing -tekniikan olevan erittäin laskennallisesti raskasta [Appel, 1968]. 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 [VentureBeat, 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 and Whitted, 1980.] Henry Fuchs et al. kehittivät toisen metodin, johon kuului myös esiprosessointivaihe, tällä kertaa tietokoneen suorittamana. Maiseman objektit oli jaettu pienempiin osiin, polygoneihin, joista valittiin binääripuun juureksi mahdollisimman keskellä maisemaa oleva.

Tässä tutkielmassa esitellään avaruuden jakamiseen perustuvia tietorakenteita, joilla kolmiulotteisten kuvien hahmontamista voidaan nopeuttaa. Luvussa 2 määritellään joitakin grafiikan peruskäsitteitä sekä esitetään algoritmi Ray Tracing -tekniikalle. Luvussa 3 tutkitaan binääristä avaruusjakoa, kd-puuta ja rajaavien tilojen hierarkiaa sekä niiden rakentamiseen ja läpikäyntiin liittyviä algoritmeja. Luvussa 4 vertaillaan edellä mainittuja tietorakenteita ja niiden soveltuvuutta Ray Tracing -algoritmin optimoimiseen.

## 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  $P = (x, y, z) \in \mathbb{R}^3$ : esimerkiksi kolmiota voidaan kuvata kolmella pisteellä ja palloa kahdella pisteellä, jotka esittävät sen keskipistettä ja yhtä pistettä sen pinnalla. [Angel and Shreiner, 2014.]

Objektit jaetaan lähes kaikissa ei-triviaaleissa tapauksissa *polygoneihin*. Polygoni eli monikulmio  $\gamma = \diamond P_1 P_2 \dots P_n$ ,  $n > 2$ , on samassa tasossa olevien kärkien  $P_1, \dots, P_n$  muodostaman murtoviivan rajaama alue, jonka kärkien muodostamat janat  $P_i P_{i+1}$  eivät leikkaa toisiaan muualla kuin kärjissä [Harju, 2015]. Useimmiten grafiikkasovelluksissa ja -rajapinnoissa valitaan polygonien muodoksi kolmiot, sillä niiden kolme kärkeä muodostavat aina tason, ja grafiikkaprosessorit osaavat operoida kolmioilla erittäin nopeasti [Angel and Shreiner, 2014].

Objektien sisäpuoli halutaan yleensä jättää huomioimatta, joten polygoneille on määriteltävä, kummalla puolella on niiden etupuoli. Kolmiot ovat tässäkin suhteessa hyvä valinta polygonien muodoksi, sillä kolmiolle  $\triangle P_1 P_2 P_3$  voidaan helposti laskea etupuolen määrittävä normaali  $n$  ristitu-  
lolla  $(P_2 - P_1) \times (P_3 - P_1)$ . 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.

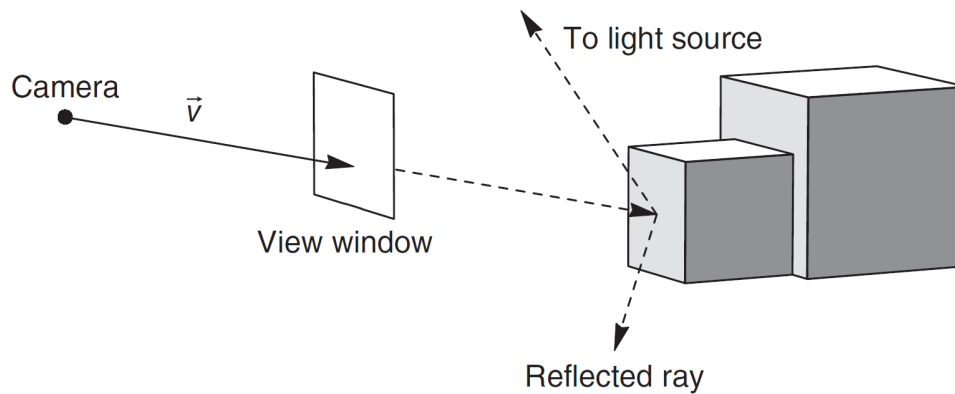
*Maailmakoordinaatisto* kuvaa koko avaruutta ja sisältää tietoa siitä, mihin objektien lokaalikoordinaatistojen origot on sijoitettu. Lopuksi tämä maisema kuvataan virtuaalisella kameralla, jolla on oma *kamerakoordinaatistonsa*. Koordinaatistosta toiseen siirtyminen, koordinaatistojen skaalaus ja rotaatio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

## 2.2 Ray Tracing -tekniikka

Ray Tracing on hahmontamistekniikka, jolla voidaan piirtää erittäin fotorealistisia kuvia. Ray Tracing -tekniikka 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, Ray Tracing -algoritmi ottaa huomioon vain ne säteet, jotka todella osuvat kameraan. Valonsäteitä seurataan siis käänteisessä järjestyksessä, kamerasta objekteihin, ja niistä valonlähteisiin. Ray Tracing -tekniikkaa on havainnollistettu kuvassa 1. [Janke, 2015.]

Ray Tracing -algoritmi muodostaa sille syötteenä annetusta kolmiulotteisesta maisemasta kamerasijainnin perusteella kaksiulotteisen kuvan. Jokaisen kuvatason pikselin läpi ammutaan säde  $\vec{R} = O + t\vec{D}$ , missä  $t \in \mathbb{R}$ ,  $O$  on kamerasijainti 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. Osumakohdasta ammutaan uusi, varjostussädeksi kutsuttu säde. Jos varjostussäde osuu suoraan tai kimmoten muista objekteista valonlähteeseen, lankeaa objektin pinnalle valoa. [Janke, 2015.] Jos polygonien muodoksi on valittu kolmiot, voidaan säteen ja kolmion leikkaus määrittää esimerkiksi nopealla Möllerin-Trumboren algoritmilla [ks. Möller and Trumbore, 1997]. Ray Tracing -tekniikan pseudokoodi on esitetty algoritmossa 1.

Algoritmin suoritusnopeutta rajoittaa se, että jokaista sädettä kohti on



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

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

Ray Tracing -tekniikkaa on perinteisesti sen hitauden vuoksi hyödynnetty hahmontamisessa, jossa käytetty laskenta-aika saakin venyä pitkäksi. Peleissä, virtuaalitodellisuudessa ja muissa reaaliaikaista hahmontamista vaativissa sovelluksissa on käytetty pitkälti niinkutsuttua kolmioiden rasterointitekniikkaa (engl. *rasterization*). Ray Tracing -tekniikalla hahmonnetut kuvat ovat kuitenkin huomattavasti realistisempia kuin rasteroinnilla saavutetut, joten reaaliaikainen hahmontaminen Ray Tracing -tekniikalla on suosittu tutkimuskohde. [Wald, 2004.]



**Input:**

kuvataso:  $x * y$  kokoinen taulukko pikseleitä

maisema: joukko valonlähteitä ja polygoneihin jaettuja objekteja

**Output:**

kolmiulotteinen maisema projisoituna kuvatasolle

RAY\_TRACING(kuvataso, maisema):

```
1 foreach pikseli  $(x, y) \in$  kuvataso do
2   | etaisyys  $\leftarrow \infty$ 
3   | foreach polygoni  $\in$  maisema do
4   |   | Ammu säde  $\vec{R} = O + t\vec{D}$  kamerasta pikselin läpi maisemaan
5   |   | if säde  $R$  osui polygoniin pisteessä  $P$  and  $t < \text{etaisyys}$  then
6   |   |   | etaisyys  $\leftarrow t$ 
7   |   |   | Valon määrä  $V \leftarrow 0$ 
8   |   |   | foreach valonlähde  $L$  do
9   |   |   |   | Ammu varjostussäde  $\vec{R}_s = L - P$  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 väriksi
15  |   | end
16  | end
17 end
18 return kuvataso
```

**Algoritmi 1: RAY\_TRACING**

## 3 Avaruusjakopuut

### 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 polygonijoukosta  $\Gamma$  yksi polygoni  $\gamma_k$ , joka asetetaan puun juureksi. Polygonin  $\gamma_k$  muodostama taso jakaa maiseman, ja siten polygonijoukon  $\Gamma$ , kahteen osaan  $\Gamma_{k,+}$  ja  $\Gamma_{k,-}$ . Joukko  $\Gamma_{k,+}$  sisältää polygonin  $\gamma_k$  positiivisella puolella olevat polygonit, ja siten ne asetetaan BSP-puuhun juuren oikeaksi lapseksi. Vastaavasti joukko  $\Gamma_{k,-}$  sisältää negatiivisella puolella olevat polygonit, ja kuuluvat polygonin  $\gamma_k$  vasemmaksi lapseksi. Tämä jakavan polygonin 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 polygoneja. [Samet, 2005.] BSP-puun rakentamisen pseudokoodi on esitelty algoritmossa 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
    BSP_Node* vasen_lapsi
    Polygon Polygonit[]    //joukko polygoneja, josta alipuut
                           //muodostetaan
}
```

**Input:**  
 BSP\_Node solmu: juurisolmu, josta puu rakennetaan  
 Polygon polygonit[]: polygonijoukko, josta alipuut rakennetaan

**Output:**  
 BSP-puu, jonka juurena on solmu node

RAKENNA\_BSP\_PUU(solmu, polygonit):

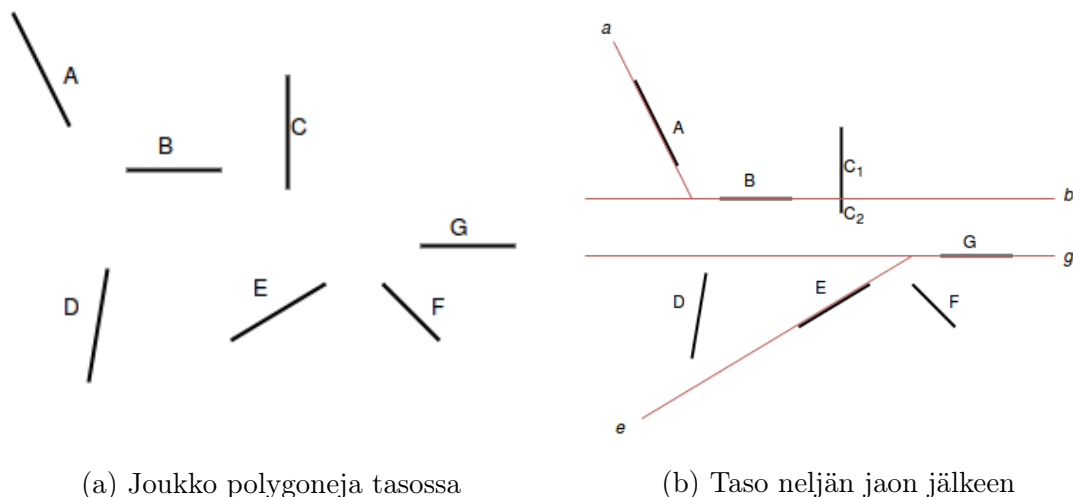
```

1 jakaja ← VALITSE_JAKAVA_POLYGOINI(polygonit)
2 positiivinen_joukko ← ∅
3 negatiivinen_joukko ← ∅
4 foreach  $\gamma \in$  polygonit do
5   sijainti ← VERTAA(jakaja,  $\gamma$ )
6   if sijainti = jakajan edessä then
7     positiivinen_joukko = positiivinen_joukko  $\cup$   $\gamma$ 
8   else if sijainti = jakajan takana then
9     negatiivinen_joukko = negatiivinen_joukko  $\cup$   $\gamma$ 
10  else if sijainti = leikkaa jakajan määrittämää tasoa then
11    JAA_POLYGOINI( $\gamma$ , jakaja,  $\gamma_i$ ,  $\gamma_j$ )
12    positiivinen_joukko = positiivinen_joukko  $\cup$   $\gamma_i$ 
13    negatiivinen_joukko = negatiivinen_joukko  $\cup$   $\gamma_j$ 
14  end
15 end
16 if positiivinen_joukko  $\neq$  ∅ then
17   RAKENNA_BSP_PUU(solmu.oikea_lapsi, positiivinen_joukko)
18 end
19 if negatiivinen_joukko  $\neq$  ∅ then
20   RAKENNA_BSP_PUU(solmu.vasen_lapsi, negatiivinen_joukko)
21 end

```

**Algoritmi 2:** RAKENNA\_BSP\_PUU

Kuvissa 2-3 on esitetty esimerkki BSP-puun muodostamisesta. Kuvassa 2a on yksinkertaisuuden vuoksi esitetty polygonit  $\Gamma = \{A, B, C, D, E, F, G\}$  sisältävä maisema kaksiulotteisena. Kuvassa 2b ensimmäiseksi jakopolygoniksi on valittu  $G$ , jonka positiiviselle puolelle jäävät polygonit  $\Gamma_{g,+} = \{A, B, C\}$ , ja negatiiviselle puolelle  $\Gamma_{g,-} = \{D, E, F\}$ . Jaon  $g$  negatiivinen puoli saadaan jaettua loppuun asti ongelmitta valitsemalla jakopolygoniksi  $E$ , mutta jos positiivisella puolella valitaan jakopolygoniksi  $B$ , joudutaan polygoni  $C$  jakamaan osiin  $C_1$  ja  $C_2$ .

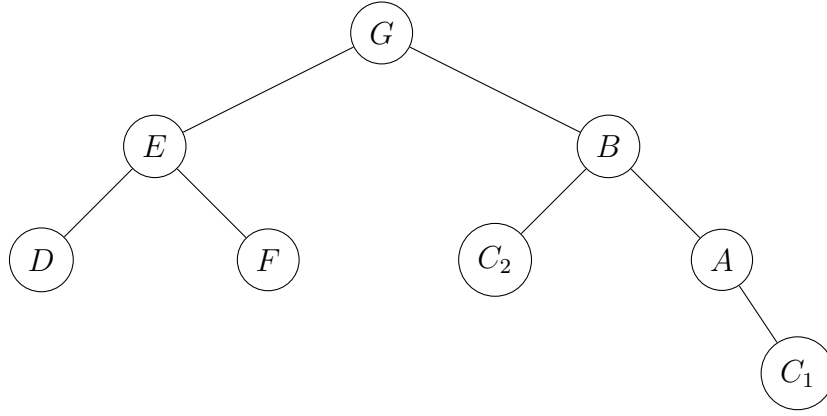


Kuva 2: Tason jakaminen

Jakolinjan  $b$  negatiiviselle puolelle jää vain yksi polygoni  $C_2$ , joten jaettavaksi jää vain  $b$ :n positiivinen puoli. Valitsemalla viimeiseksi jakopolygoniksi  $A$  syntyy kuvan 3 mukainen BSP-puu.

BSP-puun kokoon ja muotoon vaikuttaa suuresti avaruuden jakavan polygonin valinta. Pahimmassa tapauksessa kaikki polygonit  $\Gamma \setminus \{\gamma_k\}$  jäävät polygonin  $\gamma_k$  positiiviselle tai negatiiviselle puolelle jokaisella jaolla  $k$ . Tällöin puusta muodostuu pikemminkin ketjun muotoinen. Sama polygoni voi myös kuulua moneen BSP-puun alipuuhun, jos jonkun ylemmällä tasolla avaruuden jakavan polygonin  $\gamma_k$  muodostama jakolinja leikkaa tätä polygonia. [Samet, 2005.] Toinen lähestymistapa jakolinjalla oleviin polygoneihin on halkaista ne kahtia. Tämäkin tapa on epäedullinen, sillä se lisää maisemassa olevien polygonien määrää. [Ranta-Eskola, 2001.]

BSP-puuta rakennettaessa tavoitteena on muodostaa mahdollisimman tasapainoinen binääripuu valitsemalla jokaisella jakokerralla jakajaksi sellainen polygoni  $\gamma_k$ , jonka positiivisella ja negatiivisella puolella on likimain yhtä paljon polygoneja, eli  $|\Gamma_{k,+}| \approx |\Gamma_{k,-}|$ . Tällöin  $n$ :n polygonin joukosta muodostetun BSP-puun syvyys olisi  $O(\log n)$ . Koska puun solmuja syntyy lisää, kun jakolinjan leikkaavat polygonit jaetaan kahtia, tai jakolinjalla oleva polygoni sisällytetään useaan alipuuhun, voidaan puun logaritmisuutta tavoitesyvyyttä pitää vain alarajana. [Hughes et al., 2013.]  $n$  polygonia sisältävästä maisemasta rakennetun BSP-puun syvyys on siis  $\Omega(\log n)$ .



Kuva 3: Tasosta muodostettu BSP-puu

BSP-puu voidaan rakentaa ennen hahmontamista esiprosessointivaiheessa. Hahmontamisvaiheessa sitä voidaan käyttää vähentämään säde-polygoni leikkaustestien määrää. Kamerasta ammuttua sädettä verrataan ensin BSP-puun juurena toimivaan polygoniin. Jos säde leikkaa polygonin  $\gamma_k$  muodostaman avaruuden jakavan tason, säde voi osua johonkin polygoniin molemmissa joukoissa  $\Gamma_{k,+}$  ja  $\Gamma_{k,-}$ , eli joudutaan tarkastelemaan molempia alipuuta. Jos säde ei leikkaa jakotasoa, siirrytään tarkastelemaan vain toista alipuuta. Säteen  $\vec{R} = O + t\vec{D}$  osumista tasoon  $T$  voidaan testata laskemalla etäisyys  $t = \frac{-\vec{n} \cdot (O - Q_0)}{\vec{n} \cdot \vec{D}}$ , 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 polygoneihin.[Ranta-Eskola, 2001.]

### 3.1.2 kd-puu

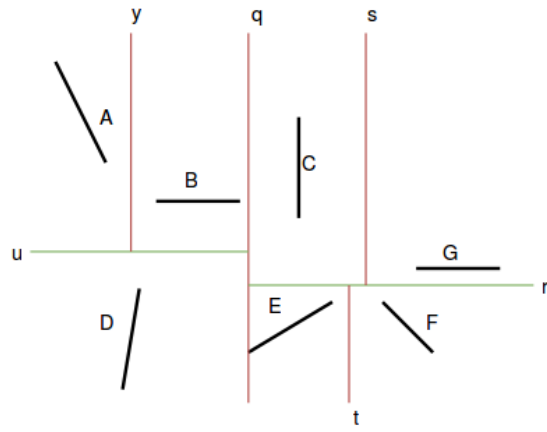
BSP-puun erikoistapaus on *kd-puu*, eli k-ulotteinen puu (engl. *k-dimensional tree*). kd-puussa avaruuden jakavaa tasoa ei valita jakavan polygonin muodostaman sivun mukaan, vaan jaot tehdään siten, että jakotaso on kohtisuorassa jotakin koordinaattiakselia vasten. Yleisin tapa muodostaa kd-puu kolmiulotteisen maiseman polygoneista on valita jakotaso vuorotellen maailmakoordinaatiston  $x$ -,  $y$ - ja  $z$ -akseleiden vastaiseksi. kd-puun jokaiseen solmuun tallennetaan jakavan polygonin sijaan jakava taso. Tällöin polygonien jakaminen solmun lapsiin onnistuu helposti

vertaamalla niiden sijaintia jakotasoon. Esimerkiksi jos avaruuden jakava taso valitaan kohtisuoraksi  $x$ -akselia vasten, tallennetaan solmun vasempaan lapseen polygonit, joiden sijainnin  $x$ -koordinaatin arvo on pienempi kuin jakotason. Oikeaan lapseen taas tallennetaan polygonit, jotka ovat jakotason positiivisella puolella  $x$ -akselin suhteen. [Samet, 2005.] Avaruutta jaetaan rekursiivisesti osiin kunnes lehtisolmuissa on jokin ennaltamäärätty määrä polygoneja. kd-puu muodostetaan vastaavalla algoritmilla kuin BSP-puun rakentamista kuvaava algoritmi 2.

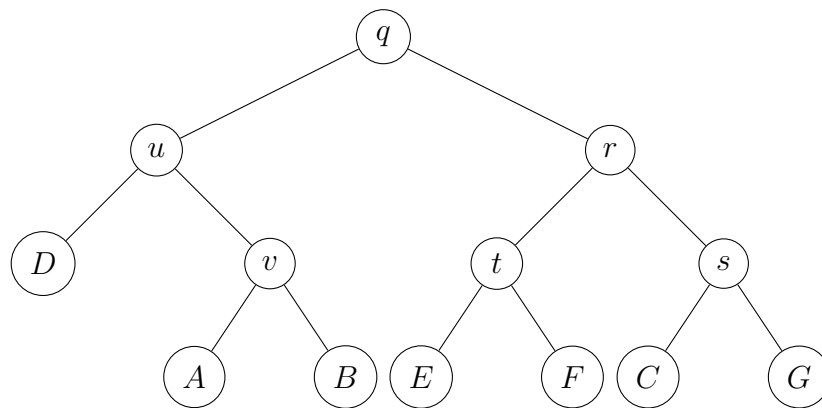
Yleisen BSP-puun tapaan myös kd-puuta rakennettaessa avaruuden jakavien tasojen valinta vaikuttaa siihen, kuinka tehokkaasti puuta voidaan hyödyntää hahmontamisessa. Jakotaso voidaan valita aina jakamalla avaruuden osa tasan kahtia, jolloin samalla syvyydellä puussa olevat solmut vastaavat saman kokoista laatikkoa. Tämä tapa ei takaa sitä, että jakotason molemmille puolille jäisi saman verran polygoneja, joten puusta ei tule tasapainoista. Hahmontamisen kannalta parempi tapa on käyttää esiprosessointivaiheessa enemmän laskenta-aikaa ja valita jakotaso siten, että sen molemmille puolilla jää likimain yhtä paljon polygoneja. Sellaiset polygonit, jotka leikkaavat avaruuden jakavaa tasoa, täytyy sisällyttää puuhun useaan solmuun tai jakaa ne pienempiin osiin. [Havran, 2001.]

Eräs mahdollinen tapa jakaa taso osiin koordinaattiakselien suuntaisesti on esitetty kuvassa 4, jossa tarkastellaan kuvan 2a tapaan polygonit  $\Gamma = \{A, B, C, D, E, F, G\}$  sisältävää kaksiulotteista tasoa. Ensin taso jaetaan puoliksi  $x$ -akselin vastaisella suoralla  $q$ . Jaon positiiviselle puolelle jäävät polygonit  $\Gamma_{q,+} = \{C, G, E, F\}$ . Kun jaetaan suoran  $q$  positiivinen puoli  $y$  akselin vastaisella jaolla  $r$  ja edelleen  $x$  akselin vastaisilla jakosuorilla  $s$  ja  $t$ , on polygonit  $C, G, E$  ja  $F$  saatu omiin solmuihinsa. Vastaava jako suoritetaan suoran  $q$  negatiivisella puolella oleville polygoneille  $\Gamma_{q,-} = \{A, B, D\}$  jakosuorilla  $u$  ja  $v$ , jolloin tason polygoneista saadaan muodostettua kuvan 5 mukainen kd-puu.

Muodostamalla esiprosessointivaiheessa maiseman polygoneista BSP-puun sijaan kd-puu voidaan nopeuttaa säteiden ja avaruuden osien yhteentörmäystestejä. Koska kd-puussa avaruuden jakavat tasot ovat kohtisuorassa koordinaattiakselia  $a$  vasten,  $a \in (x, y, z)$ , 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, 2001.]



Kuva 4: Kuvan 2a taso jaettuna kuusi kertaa



Kuva 5: Tasosta muodostettu kd-puu

### 3.1.3 quad- ja oct-puu

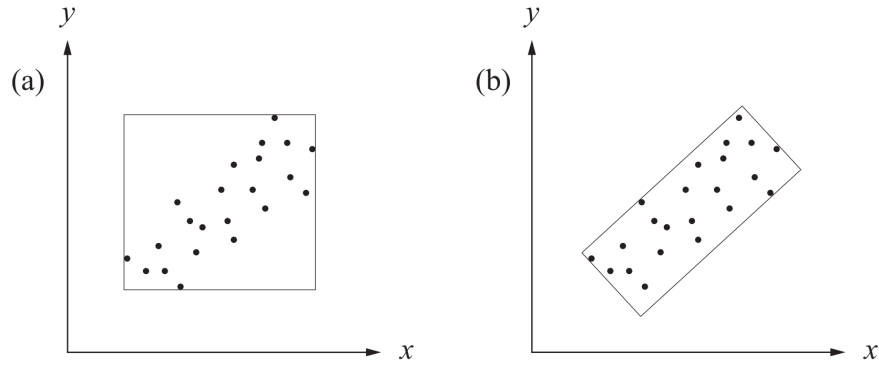
## 3.2 Rajaavien tilojen hierarkia

Eräs objektilähtöinen tapa jakaa kolmiulotteista avaruutta osiin on määrittää jokaiselle objektille *rajaava tila* (engl. *bounding volume*). Objektiin  $O$  rajava tila on jokin yksinkertainen kolmiulotteinen muoto  $V$ , jolle  $O \cap V \equiv O$  [Havran, 2001]. Useimmiten rajaavan tilan muodoksi valitaan pallo tai koordinaattiakselien suuntainen laatikko, sillä ne rajaavat objekteja yleensä 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 rajaavia tiloja. Puun solmuihin on tallennettu tieto rajaavan tilan muodosta ja koosta, sekä osoittimet lapsisolmuihin. Lehtisolmut ovat pienimpiä rajaavia tiloja, jotka sisältävät yhden objektin tai 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 kaikkien objektien polygonien 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, jona sisään kaikki objektin polygonit jäävät. [Janke, 2015.]

BVH:n hyödyntäminen hahmontamisessa Ray Tracing -tekniikalla 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 polygoneja. [Janke, 2015.]





Kuva 6: Kaksi vaihtoehtoa pistejoukon rajaavaksi tilaksi [Lengyel, 2012]

Kuten BSP- ja kd-puun tapauksissa, myös BVH:n tuoma etu hahmontamisessa riippuu siitä, miten hyvin jako rajaaviin tiloihin onnistuu. Kuvassa 6 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 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  $d = \sqrt{a^2 - (\frac{\vec{a} \cdot \vec{v}}{|\vec{v}|})^2}$  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 polygoneihin. [Janke, 2015.]

4 Avaruusjakopuiden vertailua

5 Yhteenveto

# Viitteet

- Angel, E. and 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), pages 37–45, New York, NY, USA. ACM.
- CGSociety (2014). Building 3d with ikea. [http://www.cgsociety.org/index.php/cgsfeatures/cgsfeaturespecial/building\\_3d\\_with\\_ikea](http://www.cgsociety.org/index.php/cgsfeatures/cgsfeaturespecial/building_3d_with_ikea). Luettu: 28.10.2016.
- Harju, T. (1989–2015). Geometria, lyhyt kurssi.
- Havran, V. (2001). *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, Praha, Tšekki.
- Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., and 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.
- Lengyel, E. (2012). *Mathematics for 3D Game Programming and Computer Graphics*. Course Technology PTR.
- Möller, T. and Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28.
- Ranta-Eskola, S. (2001). Binary space partitioning trees and polygon removal in real time 3d rendering. Master’s thesis, Uppsalan yliopisto, Uppsala, Ruotsi.
- Rubin, S. M. and 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.

VentureBeat (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.

Wald, I. (2004). *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, Saarbrücken, Saksa.

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

## Lista algoritmeista

1	RAY_TRACING . . . . .	7
2	RAKENNA_BSP_PUU . . . . .	9