

# Het gebruik van de normalen bij het bouwen van BSP acceleratiestructuren

Jesse Hoobergs

Thesis voorgedragen tot het behalen  
van de graad van Master of Science  
in de ingenieurswetenschappen: hoofdoptie  
computerwetenschappen, hoofdoptie  
Mens-machine communicatie

**Promotor:**  
Prof. dr. ir. Ph. Dutré

**Assessoren:**  
Dr. B. Verreet  
Prof. dr. R. Vandebriel

**Begeleiders:**  
Ir. M. Moulin  
Ir. P. Bartels

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail [info@cs.kuleuven.be](mailto:info@cs.kuleuven.be).

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

# Voorwoord

Dit is mijn dankwoord om iedereen te danken die mij bezig gehouden heeft. Hierbij dank ik mijn promotor, mijn begeleider en de voltallige jury. Ook mijn familie heeft mij erg gesteund natuurlijk.

*Jesse Hoobergs*

# Inhoudsopgave

<b>Voorwoord</b>	i
<b>Samenvatting</b>	iv
<b>Lijst van figuren</b>	v
<b>Lijst van tabellen</b>	vi
<b>Lijst van algoritmes</b>	vii
<b>Lijst van afkortingen en symbolen</b>	viii
<b>1 Inleiding</b>	1
1.1 Ray tracing . . . . .	1
1.2 Doelstelling . . . . .	1
1.3 Methodologie . . . . .	1
1.4 Contributie . . . . .	1
1.5 Overzicht . . . . .	1
<b>2 Voorgaand werk</b>	3
2.1 Basisconcepten . . . . .	3
2.2 <i>BSP</i> bomen . . . . .	4
2.3 <i>Kd</i> Bomen . . . . .	5
2.4 <i>RBSP</i> bomen . . . . .	7
2.5 Algemene <i>BSP</i> Bomen in de praktijk . . . . .	9
2.6 Vergelijking . . . . .	10
<b>3 <i>BSP<sub>SWEET</sub></i></b>	13
3.1 Probleemstelling . . . . .	13
3.2 Basisidee . . . . .	17
3.3 Gebaseerd op random richtingen . . . . .	17
3.4 Gebaseerd op normalen . . . . .	18
3.5 Vergelijking . . . . .	20
<b>4 Implementatie</b>	21
4.1 Hoog niveau beschrijving . . . . .	21
4.2 <i>Kd</i> boom . . . . .	25
4.3 <i>RBSP</i> boom . . . . .	27
4.4 <i>RBSP<sup>Kd</sup></i> boom . . . . .	28
4.5 <i>BSP<sub>SIZE</sub></i> boom . . . . .	29

## INHOUDSOPGAVE

---

4.6	$BSP_{IZE}^{Kd}$ boom . . . . .	31
4.7	$BSP_{SWEEP}$ boom . . . . .	33
<b>5</b>	<b>Resultaten</b>	<b>39</b>
5.1	Praktische aspecten . . . . .	39
5.2	Afhankelijkheid van aantal richtingen . . . . .	41
5.3	Vergelijking . . . . .	44
<b>6</b>	<b>Besluit</b>	<b>51</b>
	<b>Bibliografie</b>	<b>53</b>

# Samenvatting

In dit **abstract** environment wordt een al dan niet uitgebreide samenvatting van het werk gegeven. De bedoeling is wel dat dit tot 1 bladzijde beperkt blijft.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

# Lijst van figuren

2.1	Visuele voorstelling van <i>ray tracing</i> . . . . .	4
2.2	Splitsingskracht <i>BSP</i> boom . . . . .	5
2.3	Splitsingskracht <i>Kd</i> boom . . . . .	6
2.4	Splitsingskracht <i>RBSP</i> boom . . . . .	8
2.5	Splitsingsvlakken <i>Kd</i> , <i>RBSP</i> en <i>BSP<sub>I</sub>ZE</i> . . . . .	11
3.1	Invloed grootte bladknopen op aantal intersecties . . . . .	15
3.2	<i>False color</i> afbeeldingen van de Killeroo en Killeroo Been scene . . . . .	16
3.3	Splitsingsvlakken <i>BSP<sub>random</sub></i> . . . . .	18
3.4	Splitsingsvlakken <i>BSP<sub>random+Kd</sub></i> . . . . .	18
3.5	Splitsingsvlakken <i>BSP<sub>wn</sub></i> . . . . .	19
5.1	Testscenes . . . . .	40
5.2	Bouwtijd in functie van k . . . . .	41
5.3	<i>SAH</i> kost in functie van k . . . . .	42
5.4	Aantal inwendige knopen in functie van k . . . . .	43
5.5	Procentueel aantal <i>Kd</i> knopen . . . . .	44
5.6	Procentueel aantal <i>Kd</i> knopen per niveau . . . . .	45
5.7	Rendertijd in functie van k . . . . .	46
5.8	Straal-driehoek intersecties in functie van k . . . . .	47
5.9	Inwendige knoopdoorkruisingen in functie van k . . . . .	48
5.10	Aantal <i>Kd</i> doorkruisingen in functie van k . . . . .	50

# Lijst van tabellen

2.1	Vergelijking van de bestaande soorten <i>BSP</i> bomen.	11
3.1	Vergelijking van de nieuwe soorten <i>BSP</i> bomen.	20
4.1	Voorstelling <i>Kd</i> knoop	25
4.2	Voorstelling <i>RBSP</i> knoop	27
4.3	Voorstelling <i>BSP</i> knoop	29
4.4	Voorstelling <i>BSP<sup>Kd</sup></i> knoop	32
5.1	Statistieken Testscenes	40
5.2	Gebruikte parameterwaarden	40
5.3	Specificaties computersysteem	40
5.4	Vergelijking rendertijd en bouwtijd van <i>BSP</i> bomen	45
5.5	Vergelijking straal-driehoek intersecties van <i>BSP</i> bomen	46
5.6	Vergelijking inwendige knoopdoorkruisingen van <i>BSP</i> bomen	49
5.7	Vergelijking verhouding <i>Kd-BSP</i> knopen van <i>BSP</i> bomen	49

# Lijst van algoritmes

1	Bouwen van een BSP boom . . . . .	22
2	Intersecteren van een BSP boom . . . . .	24
3	Doorkruisen van een inwendige $Kd$ knoop. . . . .	25
4	Intersectie tussen een asgealigneerd vlak en een straal. . . . .	26
5	Beste split voor een bouwknoop b bij een $Kd$ boom. . . . .	26
6	Doorkruisen van een inwendige $RBSP$ knoop. . . . .	28
7	Intersectie tussen een vlak en een straal. . . . .	28
8	Doorkruisen van een inwendige $RBSP^{Kd}$ knoop. . . . .	29
9	Beste split voor een bouwknoop b bij een $RBSP^{Kd}$ boom. . . . .	30
10	Doorkruisen van een inwendige $BSP$ knoop. . . . .	31
11	Beste split voor een bouwknoop b bij een $BSP_{IZE}$ boom. . . . .	32
12	Doorkruisen van een inwendige $BSP^{Kd}$ knoop. . . . .	33
13	Beste split voor een bouwknoop b bij een $BSP_{IZE}$ boom. . . . .	34
14	Beste split voor een bouwknoop b bij een $BSP_{SWEEP(+)}$ boom. . . . .	35
15	Generatie richtingen voor de $BSP_{random}$ boom. . . . .	35
16	Generatie richtingen voor de $BSP_{wn}$ boom. . . . .	35
17	Generatie richtingen voor de $BSP_{cn}$ boom. . . . .	36
18	Beste split voor een bouwknoop b bij een $BSP^{Kd}$ boom. . . . .	37

# Lijst van afkortingen en symbolen

## Afkortingen

<i>BSP</i>	Binary Space Partitioning
<i>BVH</i>	Bounding Volume Hierarchy
$k - DOP$	Discrete Oriented Polytope
<i>RBSP</i>	Restricted Binary Space Partitioning Tree
$\mathcal{SA}$	Surface Area
$SAH$	Surface Area Heuristiek

---

## Symbolen

$BSP_{wn}$	$BSP$ boom die per knoop random normalen als splitsrichtingen gebruikt.
$BSP_{wn+}$	$BSP_{wn}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de splitsrichtingen.
$BSP_{wn+}^{Kd}$	$BSP_{wn+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$BSP_{cn}$	$BSP$ boom die per knoop een clustering van de normalen berekend en de centrums van deze clusters als splitsrichtingen gebruikt.
$BSP_{cn+}$	$BSP_{cn}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de splitsrichtingen.
$BSP_{cn+}^{Kd}$	$BSP_{cn+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$BSP_{random}$	$BSP$ boom die per knoop random richtingen als splitsrichtingen gebruikt.
$BSP_{random+}$	$BSP_{random}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de random splitsrichtingen.
$BSP_{random+}^{Kd}$	$BSP_{random+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$\mathcal{K}_{d,BSP}$	De kost om een $BSP$ knoop te doorkruisen.
$\mathcal{K}_{d,Kd}$	De kost om een $Kd$ knoop te doorkruisen.
$RBSP^{Kd}$	$RBSP$ waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.



# **Hoofdstuk 1**

## **Inleiding**

In dit hoofdstuk wordt het werk ingeleid. Het doel wordt gedefinieerd en er wordt uitgelegd wat de te volgen weg is (beter bekend als de rode draad).

Als je niet goed weet wat een masterproef is, kan je altijd Wikipedia eens nakijken.

### **1.1 Ray tracing**

**Stralen volgen** Door elke pixel één (of meerdere stralen), kleur intersectiepunt is kleur pixel -> path tracing.

**Acceleratiestructuren** Doel: aantal straal-driehoek intersecties verminderen.

### **1.2 Doelstelling**

Betere Acceleratiestructuur bouwen door een algemene BSP te maken die gebruik maakt van de geometrische normalen bij het splitsen. Aantal intersecties nog doen dalen, traversals stijgen, rendertijd dalen.

### **1.3 Methodologie**

### **1.4 Contributie**

### **1.5 Overzicht**



# Hoofdstuk 2

## Voorgaand werk

Dit hoofdstuk start met een korte uitleg over enkele basisconcepten: *ray tracing* en acceleratiestructuren. *Ray tracing* vereist acceleratiestructuren om efficiënt driehoeken in de scène te kunnen zoeken. Daarna wijdt dit hoofdstuk uit over één specifieke acceleratiestructuur: de *Binary Space Partitioning (BSP)* boom. De varianten van de *BSP* boom worden één voor één besproken.

### 2.1 Basisconcepten

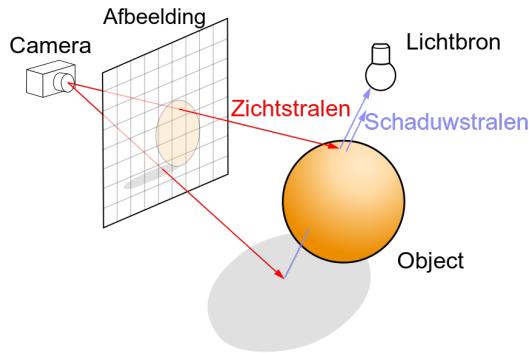
**Ray tracing** *Ray tracing* is een computergrafiek techniek voor fysisch gebaseerd renderen. Het vormt een beschrijving van een 3D scène om tot een fotorealistische 2D afbeelding. Een camera wordt op een bepaalde positie in de scène geplaatst en een afbeeldingsvlak, opgedeeld in pixels, wordt ervoor geplaatst. Door elke pixel worden één (of meerdere stralen) gestuurd. Deze stralen worden zichtstralen genoemd en de kleur van hun dichtste intersectiepunt met de scène, bepaalt de kleur van de pixel. Figuur 2.1 toont dit visueel.

Om realistische afbeeldingen te maken, wordt belichting in rekening gebracht. De eenvoudigste vorm van belichting is directe belichting waarbij een punt donker is als er niet rechtstreeks licht van een lichtbron op valt. Om dit te ondersteunen in *ray tracing* wordt gebruikt gemaakt van schaduwstralen. Dit zijn stralen van het punt naar de lichtbron. Deze schaduwstralen worden geïntersecteerd met de scène, als een intersectiepunt tussen het punt en de lichtbron gevonden wordt, is de lichtbron niet zichtbaar. Bij elke intersectie wordt aan de hand van schaduwstralen gekeken of het punt belicht wordt of niet, als het niet belicht wordt, is de kleur zwart. Voor indirecte belichting is *path tracing* een veelgebruikte techniek. *Path tracing* reflecteert stralen in het intersectiepunt afhankelijk van het materiaal en als deze straal na een aantal botsingen een lichtbron raakt, krijgt het intersectiepunt een belichting van die lichtbron.

**Acceleratiestructuren** Het doel van acceleratiestructuren is om het aantal straal-driehoek intersecties te verminderen. De simpelste acceleratiestructuur bestaat uit

## 2. VOORGAAND WERK

---



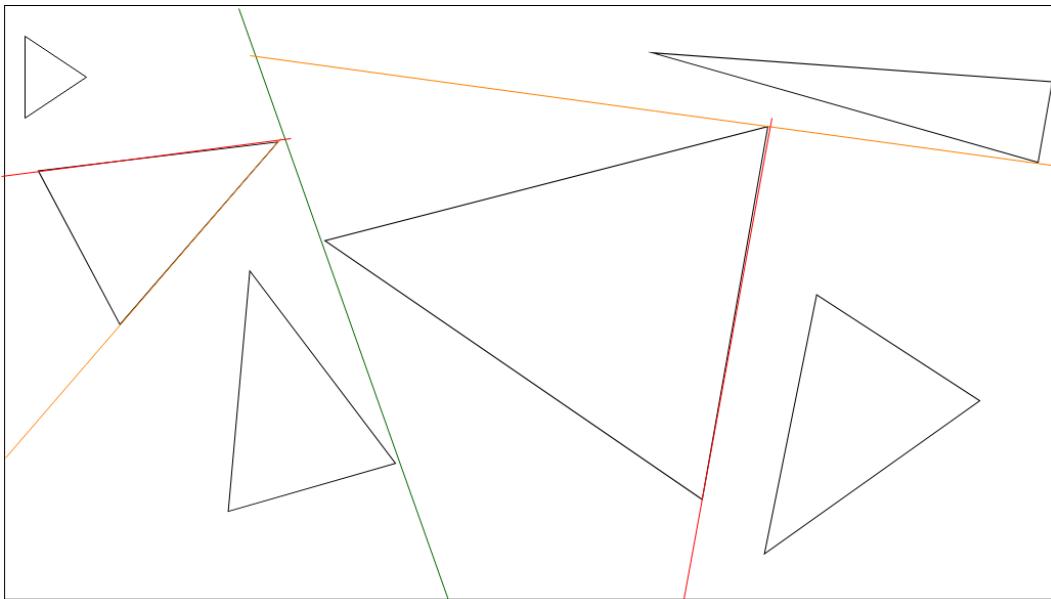
Figuur 2.1: Visuele voorstelling van *ray tracing* - Door elke pixel worden één of meerdere zichtstralen gestuurd. Schaduwstralen worden gebruikt om de belichting in de scène realistisch te maken. Deze afbeelding is een aangepaste versie van een afbeelding op [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))

het omhullend volume van de scène. Testen op intersectie met de driehoeken in de scène gebeurt dan enkel als dit omhullend volume intersecteert met de straal. Deze acceleratiestructuur kan worden uitgebreid tot een boomstructuur door dit volume recursief op te delen in kindvolumes. Binaire bomen delen elk omhullend volume op in twee nieuwe volumes, andere acceleratiestructuren zoals bijvoorbeeld octrees, delen het volume op in meer dan twee volumes.

Het volume kan worden opgedeeld op twee manieren: volgens objecten of volgens ruimte. Bij opdeling volgens objecten worden de objecten binnen het volume opgedeeld in meerdere disjuncte groepen en de kindvolumes zijn de omhullende volumes van deze groepen. Na deze opdeling zit elk object in exact één van deze nieuwe volumes, maar de volumes kunnen overlappen. Een voorbeeld van een acceleratiestructuur waarbij de opdeling volgens objecten gebeurt is de *Bounding Volume Hierarchy (BVH)*. Opdeling volgens ruimte betekent dat de ruimte in het volume wordt opgedeeld in meerdere delen. Na deze opdeling overlappen deze nieuwe volumes niet, maar een object ligt nu in minstens één (en mogelijk in meerdere) kindvolume(s). De *BSP* boom deelt de ruimte van het volume steeds op in twee kindvolumes.

## 2.2 *BSP* bomen

De *BSP* boom deelt de ruimte van het omhullend volume recursief op door te splitsen volgens een willekeurig georiënteerd vlak totdat aan een bepaalde stopconditie voldaan wordt. Het feit dat de *BSP* boom volgens willekeurig georiënteerde vlakken splitst, is zowel een voor- als nadeel. Het zorgt ervoor dat de *BSP* boom zich heel goed kan aanpassen aan de scène en alle niet-intersecerende driehoeken in principe kan scheiden (zie figuur 2.2). Het zorgt er echter ook voor dat het heel moeilijk is om deze goede splitsingsvlakken te vinden. In de praktijk wordt vaak een specifieke soort *BSP* boom gebruikt: de *Kd* boom.



Figuur 2.2: Splitsingskracht *BSP* boom - Een 2D voorbeeld dat toont dat alle (niet-intersecerende) driehoeken gesplitst kunnen worden door een *BSP* boom.

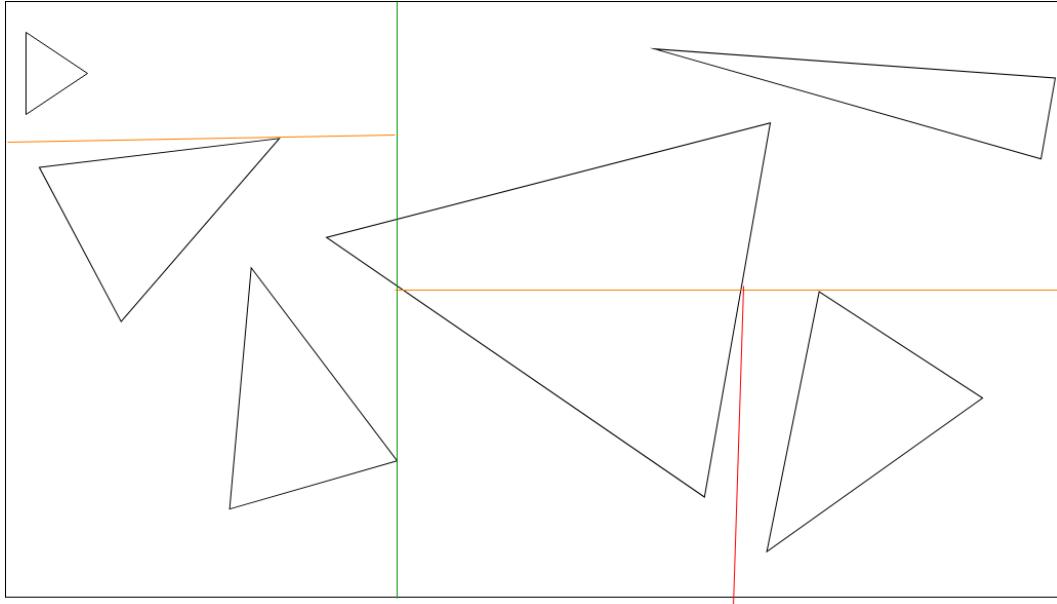
**Het bouwen van de boom** Het splitsingsvlak dat een knoop opdeelt in twee delen, kan een grote impact hebben op het aantal doorkruisstappen en het aantal driehoekintersecties. Het bouwalgoritme moet voor elke knoop het beste splitsingsvlak bepalen en als splitsen volgens dat vlak voordelig is, de knoop opsplitsen in twee kindknopen. Heuristieken voorspellen hoe goed een splitsing volgens een bepaald splitsingsvlak is. Het algoritme stopt met het splitsen van de knoop als een stopconditie bereikt wordt. Deze stopconditie kan een maximale diepte zijn, of het feit dat er geen nuttig splitsingsvlak gevonden wordt of dat het aantal driehoeken in de knoop lager is dan een vastgelegde limiet, bijvoorbeeld als er maar één driehoek in de knoop zit.

## 2.3 *Kd* Bomen

De *Kd* boom is een *BSP* boom waarbij alle splitsingsvlakken asgealigneerd zijn. Hiermee wordt bedoeld dat de splitsingsrichting (de normaal op het splitsingsvlak) evenwijdig is aan één van de drie hoofdassen. Dit zorgt ervoor dat elke knoop van de boom een asgealigneerde balk voorstelt. Het doorkruisen van een knoop uit een *Kd* boom is daardoor goedkoper dan het doorkruisen van een knoop uit een algemene *BSP* boom. De reden hiervoor is dat het goedkoper is om het intersectiepunt van een straal en een asgealigneerd vlak te vinden (verschil en vermenigvuldiging) dan het intersectiepunt van een straal en een willekeurig vlak (twee scalaire producten en deling). Door de beperking op mogelijke splitsingsvlakken kunnen *Kd* bomen zich minder goed aanpassen aan de scène. Ze kunnen bijvoorbeeld niet alle niet-intersecerende driehoeken scheiden. Figuur 2.3 toont een situatie waarin de *Kd*

## 2. VOORGAAND WERK

---



Figuur 2.3: Splitsingskracht  $Kd$  boom - Een 2D voorbeeld dat toont dat niet alle (niet-intersecerende) driehoeken gesplitst kunnen worden door de asgealigneerde vlakken van de  $Kd$  boom. De twee driehoeken linksonder (en rechtsboven) kunnen niet gesplitst worden omdat zowel hun projecties op de x-as, als hun projecties op de y-as overlappen.

boom niet alle driehoeken kan scheiden.

Ondanks dit nadeel, worden in de praktijk  $Kd$  bomen verkozen boven algemene  $BSP$  bomen. Ize et al [IWP08] geven drie (volgens hun foute) ruimverspreide aannames over algemene  $BSP$  bomen die ervoor zorgen dat  $Kd$  bomen hoger ingeschat worden. Ten eerste wordt er aangenomen dat algemene  $BSP$  bomen nooit sneller kunnen zijn dan  $Kd$  bomen omdat het doorkruisen van een  $BSP$  boom beduidend duurder is. De beperkte precisie van vloottende komma getallen wordt gezien als het tweede probleem omdat het  $BSP$  bomen numeriek onstabiel zou maken. De derde aannname is dat door de grotere flexibiliteit (= grotere verzameling van mogelijke splitsingsvlakken) het veel moeilijker is om een  $BSP$  boom te bouwen dan om een  $Kd$  boom te bouwen.

Voor een  $Kd$ -boom is de Surface Area Heuristiek ( $SAH$ ) de beste gekende methode om bomen met minimale verwachte kost te bouwen. De  $SAH$  is oorspronkelijk ontwikkeld door Goldsmith en Salmon [GS87] voor de  $BVH$  en later aangepast door MacDonald en Booth [MB90] voor de  $Kd$  boom. De  $SAH$  schat de kost van een splitsingsvlak door te veronderstellen dat beide kindknopen, bladknopen worden en dat de kost om een bladknoop te intersecteren afhankelijk is van het aantal driehoeken en de oppervlakte van het omhullend volume. De verwachte kost  $\mathcal{K}$  om een knoop  $p$  te splitsen in kindknopen  $l$  en  $r$  is dan:  $\mathcal{K}_p = \frac{\mathcal{SA}(l)}{\mathcal{SA}(p)} * n_l * \mathcal{K}_i + \frac{\mathcal{SA}(r)}{\mathcal{SA}(p)} * n_r * \mathcal{K}_i + \mathcal{K}_d$  met kost  $\mathcal{K}$ , oppervlakte  $\mathcal{SA}()$  en aantal driehoeken  $n$ . De subscripts  $i$  en  $d$  staan

voor respectievelijk intersectie en doorkruising. De kost van het beste splitsingsvlak (laagste kost) wordt dan vergeleken met de kost voor de knoop als die niet gesplitst zou worden:  $n * \mathcal{K}_i$ . Als splitsen voordelig is, wordt de knoop opgesplitst volgens dit splitsingsvlak, anders wordt de knoop een bladknoop.

Alle mogelijke asgealigeneerde splitsingsvlakken testen is ondoenbaar. Havran [Hav00] toonde aan dat de kost van de *SAH* langs een richting ofwel lineair stijgt ofwel lineair daalt tussen eindpunten (de minimale en maximale waarde) van de driehoeken langs die richting. Dit impliceert dat het beste splitsingsvlak door een eindpunt van een driehoek gaat. Het is dus voldoende om enkel de splitsingsvlakken te bekijken die door de eindpunten van de driehoeken gaan. Hieruit volgt dat er voor elke richting in een knoop met  $n$  driehoeken,  $2n$  splitsingsvlakken bekeken moeten worden. Aangezien een *Kd* boom drie richtingen bekijkt, is het bij een *Kd* boom voldoende om  $6n$  splitsingsvlakken te bekijken. Om de *SAH* kost te berekenen voor een splitsingsvlak moet het aantal driehoeken dat (deels) aan de ene kant van het splitsingsvlak ligt ( $n_l$ ) en het aantal driehoeken dat (deels) aan de andere kant ligt ( $n_r$ ) gekend zijn. Om deze aantallen efficiënt te kunnen berekenen, worden de eindpunten van de driehoeken in de knoop gesorteerd volgens hun projectie op de splitsingsrichting. Een veegbeweging (*sweep*) over deze gesorteerde lijst kan dan in lineaire tijd de *SAH* kost van alle splitsingsvlakken langs de splitsingsrichting berekenen.

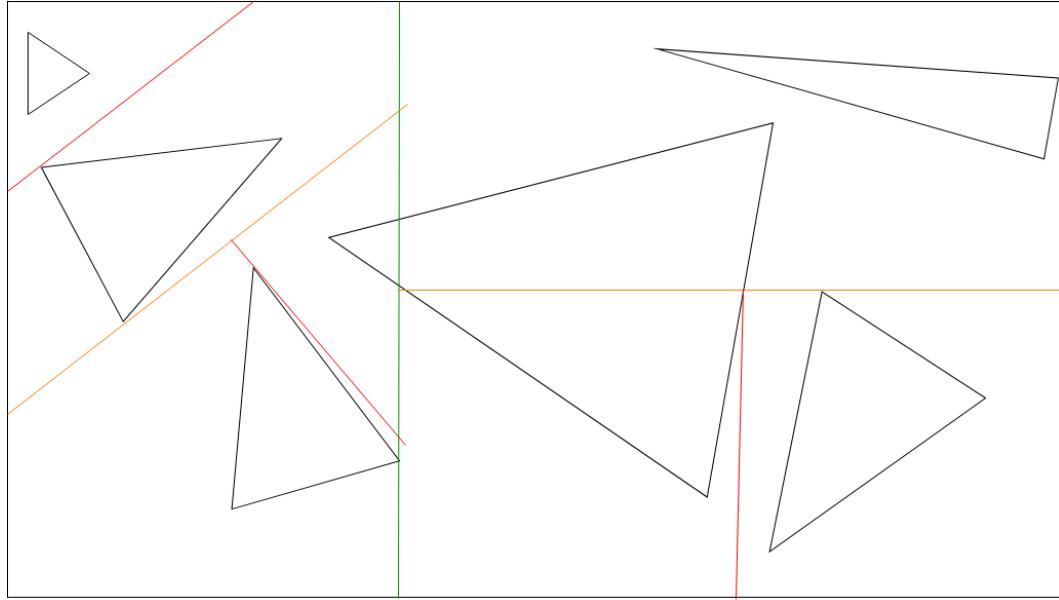
## 2.4 RBSP bomen

De Restricted Binary Space Partitioning (*RBSP*) boom is een uitbreiding van de *Kd* boom die toelaat om knopen op te splitsen volgens splitsingsrichtingen uit een vaste verzameling van  $k$  richtingen. Het omhullend volume van een *RBSP* knoop is hierdoor geen balk maar een Discreet Geöriënteerde Polytoop met  $k$  richtingen: een  $k$ -DOP. Zowel Kłosowski et al [KHM<sup>+</sup>98] als Zachmann [Zac98] hebben hiërarchiën van  $k$ -DOPs gebruikt binnen het domein van botsherkenning (*collision detection*). De *RBSP* boom is voor het eerst gebruikt in *ray tracing* door Kammaje en Mora [KM07] en verder onderzocht door Budge et al [BCNJ08]. Aangezien de splitsingsrichtingen niet asgealigeneerd moeten zijn, lijkt de *RBSP* boom meer op de algemene *BSP* boom dan de *Kd* boom. Net als de *Kd* boom kan de *RBSP* boom niet alle niet-intersecerende driehoeken scheiden. Figuur 2.4 toont een situatie waarin de *RBSP* boom niet alle driehoeken kan scheiden.

Een belangrijke ontwerpbeslissing bij *RBSP* bomen is het kiezen van de verzameling splitsingsrichtingen. Volgens Budge et al [BCNJ08] werkt in principe elke verzameling met minstens drie niet-evenwijdige richtingen, maar is het gewenst om richtingen te kiezen die samen de eenheidsbol goed bedekken. Kammaje en Mora [KM07] genereren punten langs een spiraal op gelijk verdeelde breedtegraden volgens de gouden ratio. Budge et al [BCNJ08] gebruiken de verzamelingen die ze de standaard richtingen van Kłosowski et al [KHM<sup>+</sup>98] noemen. Deze richtingen

## 2. VOORGAAND WERK

---



Figuur 2.4: Splitsingskracht *RBSP* boom - Een 2D voorbeeld dat toont dat niet alle (niet-intersecerende) driehoeken gesplitst kunnen worden door de vlakken van de *RBSP* boom. De *RBSP* boom kan de driehoeken wel beter splitsen dan de *Kd* boom in figuur 2.3. De twee driehoeken rechtsboven kunnen niet gesplitst worden omdat de discrete verzameling van k richtingen, geen richting bevat waارlangs deze driehoeken gesplitst kunnen worden.

gebruiken enkel waarden uit de verzameling  $\{-1, 0, 1\}$  als  $x$ ,  $y$  en  $z$  coördinaat.

Kammaje en Mora [KM07] toonden aan dat de SAH ook gebruikt kan worden voor *RBSP* bomen. Net zoals bij de *Kd* boom moeten voor elke splitsingsrichting de eindpunten van de driehoeken gesorteerd worden en  $2n$  splitsingsvlakken bekeken worden. In totaal bekijkt de *RBSP* boom dus  $2kn$  splitsingsvlakken. Het bouwen van de *RBSP* boom is computationeel duurder dan het bouwen van de *Kd* boom door het grotere aantal richtingen en het feit dat het splitsen en het berekenen van de oppervlakte van een k-DOP computationeel duurder is dan het splitsen en het berekenen van de oppervlakte van een balk. Budge et al [BCNJ08] vonden een oplossing voor dit tweede probleem door een methode te ontwikkelen waarmee de kind k-DOPs en hun oppervlaktes incrementeel berekend kunnen worden tijdens het sweepen.

De huidige *RBSP* implementaties zijn superieur ten opzichte van de *Kd* boom in termen van het aantal driehoekintersecties en het aantal doorkruisingen, maar moeten onderdoen in termen van rendertijd. Ize et al [IWP08] menen dat de *RBSP* boom de slechtste eigenschappen van zowel de *Kd* boom als de algemene *BSP* boom overneemt. Net als de *Kd* boom kan het geen rekening houden met de lokale geometrie en zich dus niet aanpassen aan complexe geometrie. Net als bij de algemene *BSP* boom moet de intersectie met een willekeurig georiënteerd vlak

berekend worden om knopen te doorkruisen. Budge et al [BCNJ08] tonen aan dat deze tweede eigenschap minder erg is bij de *RBSB* boom dan bij de algemene *BSP* boom omdat het mogelijk is om alle scalaire producten op voorhand (en dus maar één keer) te berekenen.

## 2.5 Algemene *BSP* Bomen in de praktijk

Ize et al [IWP08] ontwikkelden de eerste algemene *BSP* boom voor *ray tracing*. In tegenstelling tot de *Kd* en *RBSB* boom kan *BSP<sub>IZE</sub>* wel splitsingsvlakken kiezen afhankelijk van de geometrie. De *BSP<sub>IZE</sub>* boom gebruikt de splitsingsvlakken van de *Kd* boom en elke driehoek in de knoop bepaalt nog vier extra splitsingsvlakken. Deze vier vlakken zijn: het vlak van de driehoek zelf (autopartitie) en de drie vlakken loodrecht op de driehoek door elk van de drie zijden. Merk op dat deze laatste vier vlakken rekening houden met de geometrie en niet mogelijk zouden zijn bij een *RBSB* boom. Het omhullend volume van de knopen bij een *BSP* boom is een convex veelvlak. Voor de eenvoud wordt de omhullende balk gebruikt als omhullend volume voor de wortelknop. Dit heeft als extra voordeel dat er een zeer snelle test is voor stralen die niets raken.

De *SAH* kan ook gebruikt worden voor algemene *BSP* bomen. De *BSP<sub>IZE</sub>* boom controleert de  $6n$  *Kd* splitsingsvlakken door te sweepen zoals bij de *Kd* en *RBSB* boom. Het splitsen in kindknopen en de *SA* berekening is duurder aangezien het omhullend volume een convex veelvlak is. De *BSP<sub>IZE</sub>* boom controleert ook vier extra vlakken per driehoek. Voor deze vlakken is het berekenen van de *SAH* kost moeilijker omdat het aantal driehoeken links en rechts van het vlak bepaald moet worden. Om dit efficiënt te bepalen, gebruiken Ize et al [IWP08] de *BVH* als hulpstructuur. In elke knoop wordt een *BVH* boom gebouwd en die wordt bij elke niet-*Kd* splitsing gebruikt om het aantal driehoeken in beide kindknopen te bepalen. Hierdoor is het bouwen van de *BSP<sub>IZE</sub>* boom computationeel beduidend duurder dan het bouwen van *RBSB* en *Kd* bomen. De inwendige knopen van de boom kunnen worden opgedeeld in twee groepen: de knopen die gesplitst worden door een *Kd* vlak en de knopen die gesplitst worden door een geometrie-afhankelijk *BSP* vlak. Met *Kd* knopen worden de knopen uit de eerste groep bedoeld, met *BSP* knopen die uit de tweede groep.

Ize et al [IWP08] verminderen het probleem van de tragere *BSP* boom doorkruising door *Kd* knopen apart te behandelen bij het doorkruisen. Als een *Kd* knoop doorkruist wordt, kan de intersectie met het splitsingsvlak berekend worden als de intersectie van de straal met een asgealigneerd vlak. De boom die deze optimalisatie toepast, wordt aangeduid met *BSP<sub>IZE</sub><sup>Kd</sup>*. Het feit dat *Kd* knopen goedkoper zijn om te doorkruisen dan *BSP* knopen, zorgt ervoor dat er voor deze twee soorten knopen een aparte doorkruiskost gebruikt moet worden in de *SAH*:  $\mathcal{K}_{d,BSP}$  en  $\mathcal{K}_{d,Kd}$ . Deze kosten rechtstreeks in de *SAH* gebruiken, zorgt ervoor dat voornamelijk *BSP* knopen gecreëerd worden. De reden hiervoor is dat de intersectieterm van

de *SAH* lineair varieert in het aantal driehoeken en die intersectieterm domineert snel de constante doorkruisterm waardoor *BSP* splitsingen, die de driehoeken beter splitsen, gekozen worden. Deze lineaire afhankelijkheid is geen probleem als de *SAH* enkel moet beslissen of een knoop gesplitst wordt of niet, maar wel als het moet bepalen of een knoop al dan niet gesplitst wordt en of dit best door een goedkoop *Kd* vlak of door een duur *BSP* vlak gedaan wordt. Ize et al [IWP08] lossen dit probleem op door ook  $\mathcal{K}_{d,BSP}$  lineair te laten variëren in het aantal driehoeken:  $\mathcal{K}_{d,BSP} = \alpha * \mathcal{K}_i * (n - 1) + \mathcal{K}_{d,Kd}$  waarbij  $\alpha$  een instelbare parameter is. Als er na het testen van alle splitsingsvlakken geen splitsingsvlak gevonden is dat zorgt dat de kost om te splitsen lager is dan de kost om een bladknoop te creëren, worden alle *BSP* vlakken opnieuw getest, maar deze keer met een constante  $\mathcal{K}_{d,BSP}$ . Dit blijkt veel beter te werken dan een constante kost. Merk op dat deze optimalisatie gebruikt kan worden bij elke *BSP* boom die de *Kd* richtingen bekijkt. Aangezien de *Kd* richtingen bij alle huidige *RBSP* implementaties steeds in de verzameling splitsingsrichtingen zitten, kan de optimalisatie ook gebruikt worden om een  $RBSP^{Kd}$  boom te bouwen.

De  $BSP_{IZE}^{Kd}$  boom is in veel scenes even snel of zelfs sneller dan de *Kd* boom. De niet geoptimaliseerde *BSP<sub>IZE</sub>* boom is voor sommige scenes ook al beter dan de *Kd* boom. De winst wordt behaald door het lagere aantal straal driehoek intersecties. Het aantal knoopdoorkruisingen stijgt echter, waardoor de totale verbetering wordt afgezwakt. De  $BSP_{IZE}^{Kd}$  boom toont veel minder variatie in rendertijd per pixel dan de *Kd* boom die duidelijke hotspot regio's heeft. Dit toont aan dat de *BSP* boom beter kan omgaan met complexe geometrie.

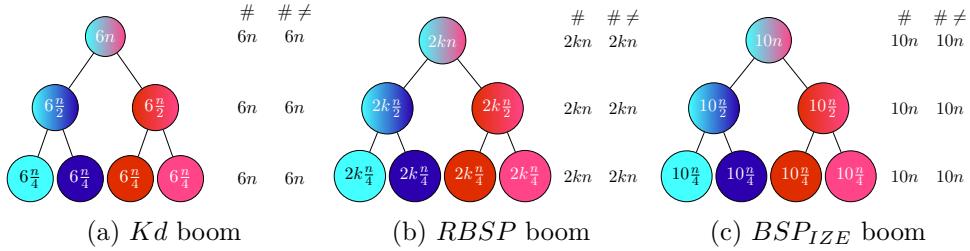
## 2.6 Vergelijking

**Bouwen en intersecteren** Tabel 2.1 vergelijkt de hierboven besproken *BSP* bomen. Hoe algemener de *BSP* boom, hoe complexer het omhullend volume en hoe nauwer de boom kan aansluiten aan de scene. Als voor elke driehoek in de knoop, dezelfde vlakken bekeken worden, kan sweeping toegepast worden om efficiënt de *SAH* kosten te berekenen. Als sweeping niet kan, zoals bij de geometrie-afhankelijke vlakken bij de *BSP<sub>IZE</sub>* boom, is het berekenen van de *SAH* kosten moeilijker en moet een hulpstructuur gebruikt worden. Voor elk van deze soorten *BSP* boom kan de optimalisatie met de snelle *Kd* knoop doorkruising gebruikt worden. Voor de *RBSP* boom is dit nog nooit gedaan.

**Aantal verschillende splitsingsvlakken** Eén van de belangrijkste eigenschappen van een *BSP* algoritme is zijn vermogen om zich aan te passen aan complexe geometrie. Het totaal aantal verschillende splitsingsvlakken dat bekeken wordt tijdens het bouwen van de boom, draagt bij tot dit vermogen. Voor een scene met  $n$  driehoeken, bekijkt een *Kd* boom  $6n$  verschillende splitsingsvlakken. Elk niveau van de boom bekijkt exact dezelfde splitsingsvlakken. Figuur 2.5a toont dit visueel. Een *RBSP* boom met  $k$  discrete richtingen doet hetzelfde, maar bekijkt  $2kn$  verschillende splitsingsvlakken. Figuur 2.5b toont dit visueel. De *BSP<sub>IZE</sub>* boom gebruikt  $10n$

	<i>Kd</i>	<i>RBSP</i>	<i>BSPIZE</i>
Omhullend volume	Asgealigneerde balk	$k - DOP$	Convex veelvlak
Sweeping	Ja	Ja	Deels
Geometrie afhankelijke vlakken	Nee	Nee	Ja
Snelle Kd doorkruising	$Kd$	$RBSP^{Kd}$ <sup>1</sup>	$BSP_{IZE}^{Kd}$
# splitsingsvlakken per niveau	$6n$	$2kn$	$10n$
totaal # ≠ splitsingsvlakken	$6n$	$2kn$	$10n$

Tabel 2.1: Vergelijking van de bestaande soorten *BSP* bomen - Deze tabel vat een aantal belangrijke eigenschappen van de bestaande soorten *BSP* bomen samen.



Figuur 2.5: Splitsingsvlakken *Kd*, *RBSP* en *BSPIZE* - Per niveau het aantal (#) splitsingsvlakken en het totaal aantal verschillende (# ≠) splitsingsvlakken gebruikt in bovenliggende niveaus.

verschillende splitsingsrichtingen,  $6n$  voor de *Kd* richtingen en  $n$  voor elk van de vier andere richtingen. De *BSPIZE* boom bekijkt op elk niveau ook exact dezelfde splitsingsvlakken en is dus niet zo algemeen als een *BSP* boom kan zijn. Figuur 2.5c toont dit visueel. De reden hiervoor is dat de gebruikte *BSP* splitsingsvlakken enkel afhankelijk zijn van de driehoeken zelf en niet van welke driehoeken samen in een knoop zitten. Geen van bovenstaande bomen gebruikt de volledige vrijheid van een *BSP* boom om op elk niveau andere splitsingsvlakken te nemen en op die manier beperken ze hun vermogen om zich aan te passen aan complexe geometrie. Driehoeken die in de wortelknoop door geen enkel vlak van elkaar kunnen worden gesplitst, kunnen nooit van elkaar gesplitst worden. Het volgende hoofdstuk introduceert nieuwe *BSP* bomen die deze vrijheid wel benutten.

<sup>1</sup>De  $RBSP^{Kd}$  boom is nog nooit geïmplementeerd.



# Hoofdstuk 3

## BSP<sub>SWEET</sub>

In dit hoofdstuk wordt een nieuwe soort *BSP* boom besproken: de *BSP<sub>SWEET</sub>* boom. Eerst wordt het nut van het opsplitsen van bladknopen in kleinere bladknopen wiskundig besproken en aangetoond met behulp van een voorbeeld. Het algemene idee van de *BSP<sub>SWEET</sub>* boom wordt dan besproken en daarna worden een aantal specifieke versies van de *BSP<sub>SWEET</sub>* (*BSP<sub>random</sub>*, *BSP<sub>wn</sub>* en *BSP<sub>cn</sub>*) besproken.

### 3.1 Probleemstelling

Het doel van acceleratiestructuren is om de totale tijd nodig om een scene te renderen, te minimaliseren. Deze sectie toont aan dat de totale rendertijd afhankelijk is van het aantal driehoeken in de geïntersecteerde bladknopen. In de eerste subsectie wordt wiskundig aangetoond dat het (onder bepaalde aannames) altijd voordelig is om bladknopen met meer dan twee driehoeken op te splitsen in kleinere bladknopen. De tweede subsectie toont een voorbeeld waarbij de *Kd* en *BSP<sub>IZE</sub>* boom vergeleken worden op basis van de grootte van de bladknopen.

#### 3.1.1 Wiskundige basis

Deze rendertijd  $T_{render}$  bestaat uit twee grote factoren: de tijd gespendeerd aan het intersecteren met driehoeken (de intersectietijd) en de tijd gespendeerd aan het doorkruisen van de boom (de doorkruistijd). De totale intersectietijd  $T_{i,totaal}$  is afhankelijk van het aantal driehoeken  $n_b$  in de geïntersecteerde bladknopen  $b$  en het aantal keer  $D_b$  dat elk van deze bladknopen doorkruist wordt. De totale doorkruistijd  $T_{d,totaal}$  is afhankelijk van het aantal doorkruisingen  $D_{inwendig}$  van inwendige knopen. Formule 3.1 beschrijft dit wiskundig met  $T_i$  de tijd nodig voor één straal-driehoek intersectie,  $T_d$  de tijd nodig om één knoop te doorkruisen en  $B$  het aantal geïntersecteerde bladknopen.

$$T_{render} \sim T_{i,totaal} + T_{d,totaal} = T_i * \sum_b^B n_b * D_b + T_d * D_{inwendig} \quad (3.1)$$

### 3. BSP<sub>SWEET</sub>

---

Stel dat één kindknoop  $b_j$  uit de boom wordt opgesplitst in twee kleinere kindknopen  $b_{j1}$  en  $b_{j2}$  die elk de helft van de driehoeken krijgen. Deze opsplitsing is voordelig als aan voorwaarde 3.2 voldaan is. Deze voorwaarde drukt uit dat knoop  $b_j$  nu een inwendig knoop wordt en dus niet meer zorgt voor een intersectietijd en wel voor een doorkruistijd en dat de nieuwe kindknopen zorgen voor een intersectietijd. De voorwaarde is equivalent aan voorwaarden 3.3 en 3.4.

$$T_{\text{render}} - T_i * n_{b_j} * D_{b_j} + T_d * D_{b_j} + \frac{n_{b_j}}{2} * (D_{b_{j1}} + D_{b_{j2}}) * T_i \leq T_{\text{render}} \quad (3.2)$$

$$\Leftrightarrow \frac{n_{b_j}}{2} * (D_{b_{j1}} + D_{b_{j2}}) * T_i \leq (T_i * n_{b_j} - T_d) * D_{b_j} \quad (3.3)$$

$$\Leftrightarrow D_{b_{j1}} + D_{b_{j2}} \leq 2D_{b_j} * \left(1 - \frac{T_d}{n_{b_j} * T_i}\right) \quad (3.4)$$

De som in het linkerlid van voorwaarde 3.4 is minstens gelijk aan  $D_{b_j}$  aangezien elke doorkruising van  $b_j$  voor minstens één doorkruising door een kindknoop zorgt. Analoog kan worden ingezien dat de maximale waarde voor deze som gelijk is aan  $2D_{b_j}$  aangezien elke doorkruising van  $b_j$  voor maximaal twee doorkruisingen door een kindknoop kan zorgen. Hieruit volgt ongelijkheid 3.5.

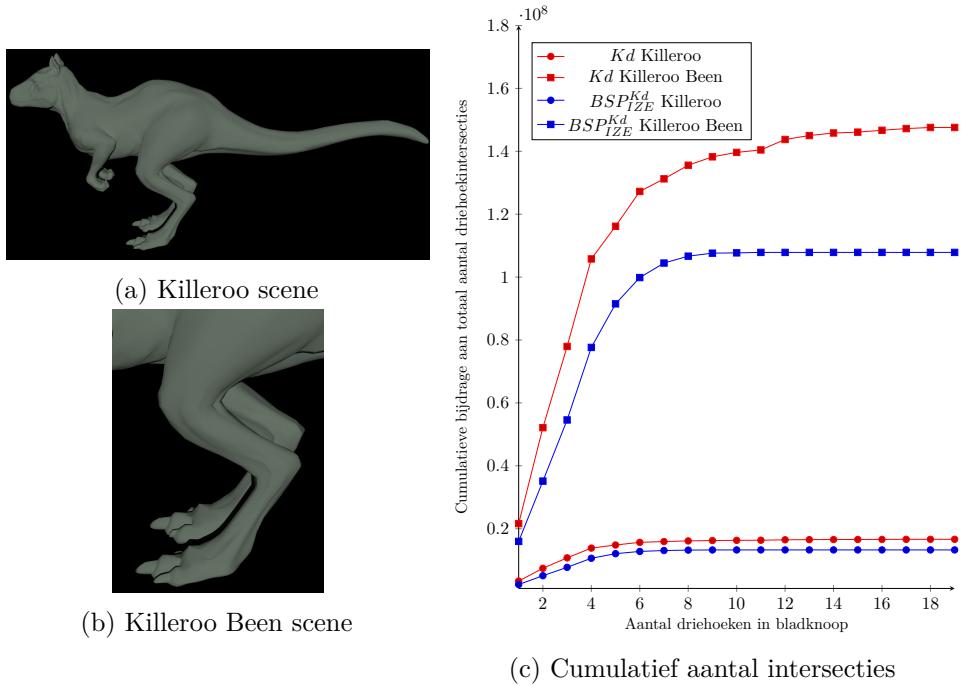
$$D_{b_j} \leq D_{b_{j1}} + D_{b_{j2}} \leq 2D_{b_j} \quad (3.5)$$

In het algemeen geval geldt dat  $D_{b_{j1}} + D_{b_{j2}} = (2 * (1 - \alpha) + \alpha)D_{b_j}$  waarbij  $\alpha$  het procentueel aantal doorkruisingen is dat door slechts één van de twee kindknopen gaat. In dit geval leidt voorwaarde 3.4 tot voorwaarde 3.6.

$$(2 * (1 - \alpha) + \alpha)D_{b_j} \leq 2D_{b_j} - \frac{2D_{b_j}T_d}{n_{b_j}T_i} \Leftrightarrow 2 - \alpha \leq 2 - \frac{2T_d}{n_{b_j}T_i} \Leftrightarrow T_d \leq \frac{\alpha n_{b_j}}{2} T_i \quad (3.6)$$

In het ideale geval ( $\alpha = 1$ ) leidt voorwaarde 3.6 tot  $T_d \leq \frac{n_{b_j}}{2} T_i$ . Het opsplitsen van een knoop met twee elementen, kan hierdoor pas voordelig zijn als de doorkruistijd kleiner is dan de intersectietijd. Aangezien de doorkruistijd in de realiteit beduidend kleiner is dan de intersectietijd, is het in dit ideale geval altijd voordelig om een kindknoop op te splitsen, ongeacht het aantal driehoeken in de knoop. In het slechtste geval ( $\alpha = 0$ ) kan aan voorwaarde 3.6 enkel voldaan zijn als de doorkruistijd gelijk is aan nul. Dit is onmogelijk waardoor opsplitsen nooit voordelig kan zijn in dit geval.

Het is moeilijk om de waarde van  $\alpha$  te voorspellen, deze is namelijk afhankelijk van de exacte stralen die tijdens het renderen gevuld worden, de specifieke driehoeken in de knoop en het splitsingsvlak. Voorwaarde 3.6 toont dat de kans dat splitsen voordelig is, lineair stijgt met  $n_{b_j}$ . De voorwaarde toont ook dat het splitsen van een knoop met twee driehoeken, voordelig is wanneer de doorkruistijd  $\alpha$  keer kleiner is dan de intersectietijd. Intuïtief lijkt het logisch dat hier in het algemeen aan voldaan is. Hieruit kan worden afgeleid dat het altijd beter is om bladknopen met meer dan één driehoek op te splitsen in twee kleinere bladknopen.



Figuur 3.1: Invloed grootte bladknopen op aantal intersecties - (a) toont de Killeroo scene, (b) toont de scène waarbij ingezoomd is op de benen van de Killeroo, (c) toont het totaal aantal intersecties als een cumulatieve som over het aantal driehoeken in de bladknopen. Dit betekent dat de waarde bij 2 overeenkomt met de som van het aantal intersecties in bladknopen met 1 driehoek en het aantal intersecties in bladknopen met 2 driehoeken.

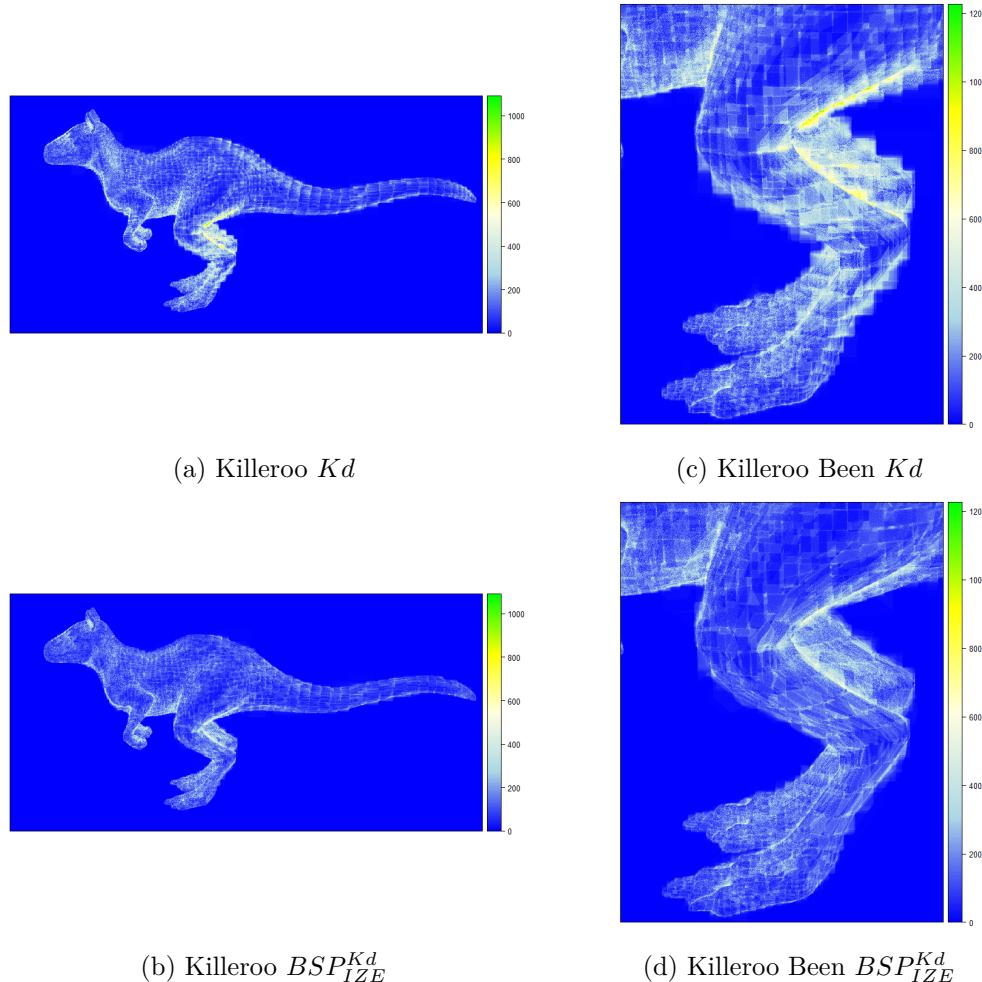
### 3.1.2 Voorbeeld

Figuur 3.1a toont de Killeroo scene die als voorbeeld scene dient bij de pbrt *ray tracer*. Voor deze scène zijn de *Kd* boom en de *BSP*<sub>IZE</sub><sup>*Kd*</sup> boom aan elkaar gewaagd. De *BSP*<sub>IZE</sub><sup>*Kd*</sup> boom neemt de bovenhand als er wordt ingezoomd op de benen (figuur 3.1b) van de Killeroo, waar de *Kd* boom veel bladknopen met meerdere driehoeken bevat. Figuur 3.1c toont het cumulatief aantal intersecties per groep van bladknopen met hetzelfde aantal driehoeken. Voor de gewone Killeroo scène, heeft de *Kd* boom slechts een beperkt aantal extra intersecties nodig. Bij de ingezoomde scène, komt een groot deel van de intersecties van de *Kd* boom door intersecties met bladknopen met een groot aantal driehoeken. Dit toont aan dat *BSP* bomen nuttig kunnen zijn omdat ze minder tot geen last hebben van regio's die ze minder goed aankunnen. Het aantal knoopdoorkruisingen bij de *BSP*<sub>IZE</sub><sup>*Kd*</sup> boom is zelfs lager dan bij de *Kd* boom voor beide scènes. Dit komt omdat de *BSP* knopen nauwer aansluiten aan het object waardoor minder stralen de knopen raken.

Figuur 3.2 toont *false color* afbeeldingen van het aantal zichtstraalintersecties van de Killeroo en Killeroo been scène voor zowel de *Kd* boom als de *BSP*<sub>IZE</sub><sup>*Kd*</sup> boom. Voor de Killeroo scène zijn de *false color* afbeeldingen van beide bomen zeer

### 3. BSP<sub>SWEET</sub>

---



Figuur 3.2: *False color* afbeeldingen van de Killeroo en Killeroo Been scene - Deze afbeeldingen tonen dat de  $Kd$  boom slecht presteert bij sommige delen van een scène, in dit geval de benen van de Killeroo. Bij de  $BSP_{IZE}^{Kd}$  boom is dit fenomeen veel minder uitgesproken en het valt ook op dat de  $BSP_{IZE}^{Kd}$  boom nauwer aansluit aan de scène.

gelijkaardig, maar aan de benen heeft de  $Kd$  boom duidelijk meer intersecties nodig. Dit wordt bevestigd door de *false color* afbeeldingen van de Killeroo been scène waarop duidelijk zichtbaar is dat de  $BSP_{IZE}^{Kd}$  boom zich beter kan aanpassen aan complexe geometrie. Het is ook zichtbaar dat de  $BSP_{IZE}^{Kd}$  boom nauwer aansluit aan de scène dan de  $Kd$  boom. Merk op dat, zoals besproken in het vorige hoofdstuk, de  $BSP_{IZE}^{Kd}$  boom nog niet alle vrijheden van een algemene  $BSP$  boom gebruikt.

De volgende secties bespreken nieuwe  $BSP$  bomen die gebruik maken van de vrijheid om op elk niveau van de boom andere splitsingsvlakken te gebruiken en op die manier meer bladknopen kunnen opsplitsen in kleinere bladknopen en nog nauwer kunnen aansluiten aan de scène.

## 3.2 Basisidee

De  $BSP_{SWEEP}$  boom is een algemene  $BSP$  boom waarbij in elke knoop  $k$  richtingen bepaald worden en alle  $2n$  splitsingsvlakken langs elk van deze richtingen worden bekeken door te sweepen. Deze  $k$  richtingen kunnen verschillend zijn voor elke knoop en kunnen gekozen worden afhankelijk van de lokale geometrie. De  $RBSP$  boom is een  $BSP_{SWEEP}$  boom waarbij de gekozen richtingen in elke knoop hetzelfde zijn. De  $BSP_{SWEEP}$  boom heeft drie belangrijke ontwerpbeslissingen. De belangrijkste ontwerpbeslissing bij de  $BSP_{SWEEP}$  boom is de methode die gebruikt wordt om de  $k$  richtingen te bepalen. Een tweede belangrijke ontwerpbeslissing is de waarde van  $k$ . De derde belangrijke ontwerpbeslissing sluit aan bij de eerste en gaat over het al dan niet gebruiken van de  $Kd$  richtingen als de eerste drie van de  $k$  richtingen.

## 3.3 Gebaseerd op random richtingen

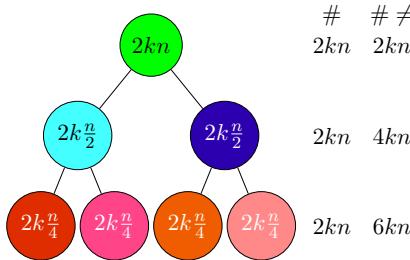
De simpelste  $BSP_{SWEEP}$  boom, de  $BSP_{random}$  boom, bepaalt in elke knoop  $k$  random richtingen onafhankelijk van de geometrie. De richtingen worden uniform op de hemisfeer gegenereerd. Het idee achter deze boom is dat het nuttiger kan zijn om driehoeken via veel verschillende vlakken te proberen splitsen, dan om ze steeds met dezelfde vlakken te proberen splitsen. Als de driehoeken in de scène uniform verdeeld zijn, dan is de kans dat twee driehoeken volgens een willekeurige richting gesplitst kunnen worden, even groot als de kans dat ze door een  $Kd$  richting gesplitst kunnen worden.

Als de  $BSP_{random}$  boom perfect gebalanceerd is, worden in elk niveau  $2kn$  verschillende splitsingsvlakken bekeken. Deze splitsingsvlakken zijn verschillend op elk niveau, zodat in totaal  $2kn\log(n)$  verschillende splitsingsvlakken bekeken worden. Figuur 3.3 toont dit visueel. De  $BSP_{random}$  boom probeert elke driehoek via gemiddeld  $2k\log(n)$  ( $\mathcal{O}(\log(n))$ ) vlakken te splitsen van de andere driehoeken, in tegenstelling tot de bestaande bomen die dit maximaal met  $\mathcal{O}(1)$  vlakken proberen.

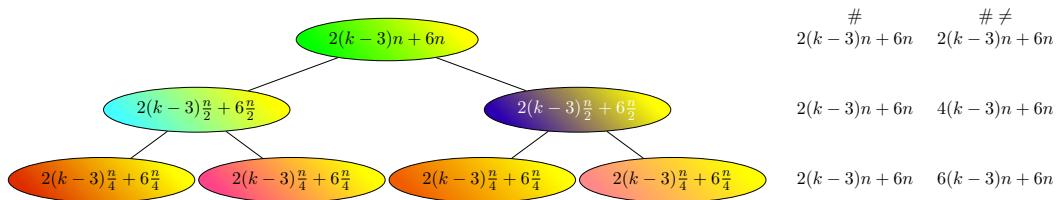
De  $Kd$  richtingen zijn in praktische scenes vaak beter dan willekeurige richtingen omdat ze loodrecht op elkaar staan waardoor ze de hemisfeer goed bedekken en omdat scenes die door de mens gemaakt worden, vaak asgealigneerde delen bevatten. Dit geeft aanleiding tot een boom die als eerste drie richtingen steeds de  $Kd$  richtingen kiest en enkel de overige  $k - 3$  richtingen random genereert: de  $BSP_{random+}$  boom. Een extra voordeel is dat de snellere  $Kd$  knoop doorkruising gebruikt kan worden. Een  $BSP_{random}$  boom die altijd de  $Kd$  richtingen gebruikt en de doorkruising van  $Kd$  knopen optimaliseert, wordt aangeduid als  $BSP_{random+}^{Kd}$ . Als de  $BSP_{random+}^{(Kd)}$  boom perfect gebalanceerd is, worden in elk niveau  $2kn$  splitsingsvlakken bekeken. Van deze  $2kn$  zijn er  $6n$  die hergebruikt worden, de vlakken volgens de  $Kd$  richtingen.

### 3. BSP<sub>SWEEP</sub>

---



Figuur 3.3: Splitsingsvlakken  $BSP_{random}$  - Per niveau het aantal ( $\#$ ) splitsingsvlakken en het totaal aantal verschillende ( $\# \neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{random}$  boom.



Figuur 3.4: Splitsingsvlakken  $BSP_{random+}^{(Kd)}$  - Per niveau het aantal ( $\#$ ) splitsingsvlakken en het totaal aantal verschillende ( $\# \neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{random+}^{(Kd)}$  boom.

Dit zorgt voor  $2(k-3)n$  verschillende splitsingsvlakken per niveau, zodat in totaal  $2(k-3)n \log(n) + 6n$  verschillende splitsingsvlakken bekeken worden. Figuur 3.4 toont dit visueel. De  $BSP_{random+}^{(Kd)}$  boom probeert elke driehoek via gemiddeld  $\mathcal{O}(\log(n))$  vlakken te splitsen van de andere driehoeken, net als de  $BSP_{random}$  boom.

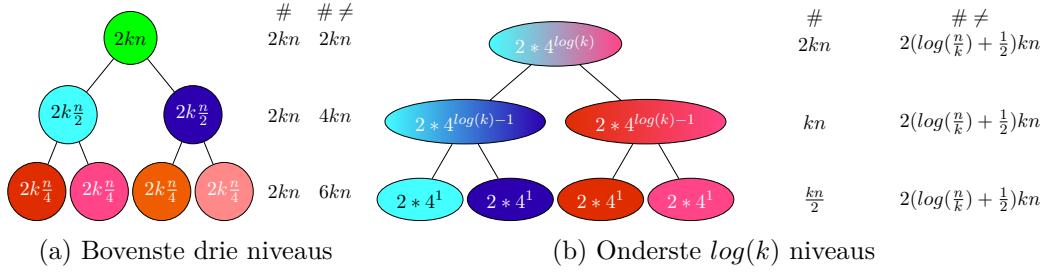
## 3.4 Gebaseerd op normalen

De kracht van de  $BSP_{SWEEP}$  boom ligt in het feit dat er gebruik gemaakt kan worden van de lokale geometrie om de splitsingsrichtingen te bepalen. De normalen van de driehoeken in een knoop bevatten informatie over de oriëntatie van de driehoeken. De autopartitie vlakken van Ize et al maken ook gebruik van de normalen, maar de normaal van een driehoek wordt enkel voor die driehoek zelf gebruikt.

### 3.4.1 Willekeurige normaal

De  $BSP_{wn}$  boom is een  $BSP_{SWEEP}$  boom waarbij in elke knoop de normalen van k willekeurige driehoeken gekozen worden als splitsingsrichtingen. Als er k of minder driehoeken in de knoop zitten, worden alle normalen als splitsingsrichting genomen. In het algemeen zijn er n driehoeken met n verschillende normalen waardoor er in totaal  $2n^2$  mogelijke splitsingsvlakken zijn. Figuur 3.5a toont het aantal splitsingsvlakken gebruikt per knoop in de bovenste niveaus van de boom. Deze figuur is identiek aan

### 3.4. Gebaseerd op normalen



Figuur 3.5: Splitsingsvlakken  $BSP_{wn}$  - Per niveau het aantal ( $\#$ ) splitsingsvlakken en het totaal aantal verschillende ( $\# \neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{random}$  boom.

de figuur voor de  $BSP_{random}$  boom.

Voor de onderste  $\log(k)$  niveaus is er echter een verschil, omdat er dan  $k$  of minder driehoeken in elke knoop zitten. In de onderste  $\log(k)$  niveaus worden er in elke knoop  $2n_m^2$  splitsingsvlakken bekeken, met  $n_m \leq k$ . Een knoop op het  $m^{de}$  laagste niveau (met  $m \leq \log(k)$ ) bevat  $\frac{n}{2^{\log(n)-m}}$  driehoeken waardoor er  $2n_m^2 = 2 * (\frac{n}{2^{\log(n)-m}})^2 = 2 * (2^m)^2 = 2 * 4^m$  splitsingen gebeuren in deze knoop. Formule 3.7 toont dat het totaal aantal splitsingsvlakken bekeken op het  $\log(k) - j^{de}$  onderste niveau gelijk is aan  $\frac{2kn}{2^j}$ . Dit aantal is berekend als het aantal splitsingsvlakken per knoop ( $2 * 4^{\log(k)-j}$ ) vermenigvuldigd met het aantal knopen ( $2^{\log(n)-\log(k)+j}$ ) op het niveau.

$$2 * 4^{\log(k)-j} * 2^{\log(n)-\log(k)+j} = 2 * 2^{\log(k)+\log(n)-j} = \frac{2 * 2^{\log(kn)}}{2^j} = \frac{2kn}{2^j} \quad (3.7)$$

In de bovenste  $\log(n) - \log(k) = \log(\frac{n}{k})$  niveaus worden  $2\log(\frac{n}{k})kn$  verschillende splitsingsvlakken bekeken. Het niveau eronder gebruikt nog  $kn$  nieuwe splitsingsvlakken zodat in totaal  $2(\log(\frac{n}{k}) + \frac{1}{2})kn$  verschillende splitsingsvlakken worden gebruikt. Figuur 3.5b toont dit visueel. Als de boom perfect gebalanceerd is, probeert de  $BSP_{wn}$  boom elke driehoek via gemiddeld  $2k\log(\frac{n}{k} + \frac{1}{2})$  ( $\mathcal{O}(\log(n))$ ) vlakken te splitsen van de andere driehoeken. Dit aantal is minder dan bij de  $BSP_{random}$  boom, die ook in de lagere niveaus steeds  $k$  splitsingrichtingen genereert.

Net als bij de  $BSP_{random}$  boom kan een versie van de  $BSP_{wn}$  boom gemaakt worden die als eerste drie richtingen steeds de  $Kd$  richtingen kiest en enkel voor de overige  $k - 3$  richtingen willekeurige normalen selecteert: de  $BSP_{wn+}$  boom. Een  $BSP_{wn}$  boom die altijd de  $Kd$  richtingen gebruikt en de doorkruising van  $Kd$  knopen optimaliseert, wordt aangeduid als  $BSP_{wn+}^{Kd}$ . Als de  $BSP_{wn+}^{Kd}$  boom perfect gebalanceerd is, worden in totaal  $2(\log(\frac{n}{k-3}) + \frac{1}{2})(k-3)n + 6n$  verschillende splitsingsvlakken bekeken. De analyse hiervoor is analoog aan de analyse voor  $BSP_{random+}^{Kd}$ .

### 3. BSP<sub>SWEET</sub>

---

	$BSP_{random}$	$BSP_{wn}$	$BSP_{cn}$
Omhullend volume	Convex veelvlak	Convex veelvlak	Convex veelvlak
Sweeping	Ja	Ja	Ja
Geometrie afhankelijke vlakken	Nee	Ja	Ja
Snelle Kd doorkruising	$BSP_{random+}^{Kd}$	$BSP_{wn+}^{Kd}$	$BSP_{cn+}^{Kd}$
# splitsingsvlakken per niveau totaal # ≠ splitsingsvlakken	$2kn$	$2kn^*$	$2kn^*$
	$2kn \log(n)$	$2(\log(\frac{n}{k}) + \frac{1}{2})kn$	$2(\log(\frac{n}{k}) + \frac{1}{2})kn$

Tabel 3.1: Vergelijking van de nieuwe soorten  $BSP$  bomen - Deze tabel vat een aantal belangrijke eigenschappen van de nieuwe soorten  $BSP$  bomen samen. \*De onderste niveaus van  $BSP_{wn}$  en  $BSP_{cn}$  bekijken minder dan  $2kn$  splitsingsvlakken.

#### 3.4.2 Geclusterde normalen

Bovenstaande  $BSP_{SWEET}$  bomen bevatten een grote niet-deterministische factor. De  $BSP_{cn}$  boom gebruikt het K-means clustering algoritme om deterministischere richtingen te bepalen. De normalen van de driehoeken in een knoop worden geclusterd in  $k$  clusters en de centra van deze clusters worden gebruikt als richtingen voor de  $BSP_{SWEET}$  boom. Als er  $k$  of minder driehoeken in de knoop zitten, worden alle normalen als splitsingsrichting genomen, net zoals bij de  $BSP_{wn}$  boom. In tegenstelling tot de  $BSP_{wn}$  is het voor de  $BSP_{cn}$  moeilijker om het totaal aantal mogelijke splitsingsvlakken te bepalen. In het algemeen kan elke clustering verschillende richtingen geven waardoor er oneindig veel mogelijke splitsingsrichtingen mogelijk zijn. De  $BSP_{cn}$  boom is zeer gelijkaardig aan de  $BSP_{wn}$  boom waardoor het totaal aantal verschillende splitsingsvlakken die gebruikt worden, gelijk is. Analoog bestaan ook de  $BSP_{cn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen.

## 3.5 Vergelijking

Tabel 3.1 vergelijkt de hierboven besproken  $BSP_{SWEET}$  bomen. Alle drie bomen zijn algemene  $BSP$  bomen die een convex veelvlak hebben als omhullend volume. De  $BSP_{SWEET}$  bomen zijn ontworpen om sweeping te ondersteunen en zo efficiënt de  $SAH$  kosten te kunnen berekenen, zonder hulpstructuren. De  $BSP_{random}$  boom is onafhankelijk van de lokale geometrie, de twee andere bomen bepalen hun splitsingsvlakken afhankelijk van de lokale geometrie. De  $BSP_{SWEET}$  bomen zijn makkelijk uit te breiden naar  $BSP_{SWEET}^{Kd}$  bomen die de optimalisatie met de snelle Kd knoop doorkruising gebruiken. Per niveau gebruiken de  $BSP_{SWEET}$  bomen niet altijd meer splitsingsvlakken dan de bestaande bomen, maar de  $BSP_{SWEET}$  bomen gebruiken in totaal duidelijk meer verschillende splitsingsvlakken dan de bestaande bomen en zouden zich hierdoor beter moeten kunnen aanpassen aan complexe scenes.

## Hoofdstuk 4

# Implementatie

In dit hoofdstuk wordt de implementatie van volgende *BSP* bomen besproken: *Kd* boom, *RBSP* boom, *RBSP<sup>Kd</sup>* boom, *BSP<sub>IZE</sub>* boom, *BSP<sub>IZE</sub><sup>Kd</sup>* boom, *BSP<sub>random</sub>* boom, *BSP<sub>random+</sub>* boom, *BSP<sub>random+</sub><sup>Kd</sup>* boom, *BSP<sub>wn</sub>* boom, *BSP<sub>wn+</sub>* boom, *BSP<sub>wn+</sub><sup>Kd</sup>* boom, *BSP<sub>cn</sub>* boom, *BSP<sub>cn+</sub>* boom en *BSP<sub>cn+</sub><sup>Kd</sup>* boom. Deze implementaties zijn toegevoegd aan de pbrt-v3 *ray tracer* [PM19] en zijn beschikbaar op <https://github.com/jhoobergs/Thesis-pbrt-v3>.

Het hoofdstuk start met een hogenniveau beschrijving van het algoritme om de boom te bouwen en het algoritme om de boom te intersecteren. Daarna wordt voor elke type aparte besproken hoe hun knopen voorgesteld worden in het geheugen, hoe inwendige knopen doorkruist worden en hoe het beste splitsingsvlak bepaald wordt.

### 4.1 Hoog niveau beschrijving

#### 4.1.1 Bouwalgoritme

Het bouwalgoritme voor *BSP* bomen heeft een aantal parameters:

- $\mathcal{K}_i$ : De intersectiekost voor primitieven. Deze parameter is nodig voor de *SAH* heuristiek.
- $\mathcal{K}_{d,Kd}$ : De doorkruiskost voor *Kd* knopen. Deze parameter is nodig voor de *SAH* heuristiek.
- $\mathcal{K}_{d,BSP}$ : De aparte doorkruiskost voor *BSP* knopen. Deze parameter is nodig voor de *SAH* heuristiek.
- $n_{max}$ : Elke knoop met  $n_{max}$  of minder primitieven wordt direct een bladknoop, er wordt niet geprobeerd om de knoop te splitsen.
- $d_{max}$ : De maximale diepte van de boom.

Algoritme 1 toont de algemene vorm van het bouwalgoritme voor *BSP* bomen. Elk type *BSP* boom wordt bepaald door de specifieke implementatie van de volgende functies:

#### 4. IMPLEMENTATIE

---



---

**Algoritme 1** Bouwen van een BSP boom

---

```

stack  $\leftarrow \emptyset$ 
Voeg een bouwknoop met alle primitieven toe aan de stack
while  $stack \neq \emptyset$  do
     $b \leftarrow \text{POP}(stack)$ 
    if  $b_n \leq n_{max}$  or  $b_d = d_{max}$  then
        MAAK_BLAD_KNOOP( $b$ )
        continue
    end if
     $besteSplit \leftarrow \text{BEPAAL_BESTE_SPLIT}(b)$ 
     $nietSplitKost \leftarrow b_n * \mathcal{K}_i$ 
    if  $besteSplit_{kost} > nietSplitKost$  then
         $b_{slechteAanpassingen} \leftarrow b_{slechteAanpassingen} + 1$ 
    end if
    if ( $besteSplit_{kost} > 4 * nietSplitKost$  and  $b_n < 16$ ) or  $besteSplit = None$  or
 $b_{slechteAanpassingen} = 3$  then
        MAAK_BLAD_KNOOP( $b$ )
        continue
    end if
    MAAK_INWENDIGE_KNOOP( $b$ )
    Plaats de kindknopen als twee nieuwe bouwknopen op de stack
end while

```

---

1. BEPAAL\_BESTE\_SPLIT(bouwknoop): Deze functie bepaalt de beste splitsing voor de knoop. Het resultaat bevat het splitsingsvlak en de bijhorende *SAH* kost.
2. MAAK\_BLAD\_KNOOP(bouwknoop): Deze functie maakt een bladknoop van de huidige bouwknoop.
3. MAAK\_INWENDIGE\_KNOOP(bouwknoop): Deze functie maakt een inwendige knoop van de huidige bladknoop.

De eerste functie omvat de specifieke eigenschappen van het type *BSP* boom en bepaalt de kracht van dat type *BSP* boom. Het resultaat van de tweede en derde functie is een specifieke representatie voor bladknopen / inwendige knopen die nuttig is voor het specifieke type *BSP* boom. In de volgende secties wordt bij elk type *BSP* boom hun specifieke knooprepresentaties besproken. De exacte implementaties van de functies om deze representaties in te vullen, zijn triviaal en worden niet expliciet beschreven.

Samengevat gaat Algoritme 1 voor elke knoop, die nog voldoende primitieven bevat en niet op de maximale diepte ligt, de beste splitsing bepalen en afhankelijk van de *SAH* waarde bepalen of een splitsing moet gebeuren. Aangezien een slechte splitsing op een bepaald niveau, kan leiden tot een nuttige split op een lager niveau,

wordt een knoop toch gesplitst als splitsen nadrukkelijk is volgens de *SAH* heuristiek. Zulke *slechte aanpassingen* worden niet gedaan als de kost van de beste split vier keer hoger is dan de kost om niet te splitsen en als er bovendien minder dan 16 primitieven in de knoop zitten. Elk lusvrij pad van de wortelknoop naar elke andere knoop, mag maximaal twee zulke *slechte aanpassingen* bevatten. Dit komt neer op het feit dat een *slechte aanpassing* enkel mag gebeuren als er minder dan twee voorouderknopen een *slechte aanpassing* gedaan hebben. Dit concept is overgenomen van de originele implementatie van de *Kd* boom in *pbrt-v3*.

#### 4.1.2 Intersectie-algoritme

Het intersectie-algoritme bepaalt het intersectiepunt tussen een straal en de *BSP* boom. Een straal wordt voorgesteld met de parametervoorstelling  $\vec{s} = \vec{o} + t\vec{d}$ . Algoritme 2 toont de algemene vorm van het intersectie-algoritme voor *BSP* bomen. Dit intersectie-algoritme is ontworpen voor zichtstralen en bepaalt dus het dichtste intersectiepunt. De aanpassing naar een algoritme dat controleert of er een intersectiepunt bestaat, wat gebruikt kan worden voor schaduwstralen, is triviaal en wordt niet verder besproken.

Het algoritme maakt gebruik van de volgende functies:

1. DOORKRUIS\_INWENDIGE\_KNOOP(knoop, straal): Deze functie bepaalt de intersectie van de gegeven straal met het splitsingsvlak van de gegeven inwendige knoop. Het resultaat bevat tVlak, de t waarde waarvoor de straal intersecteert met het vlak en linksEerst, een boolean die aangeeft of de straal eerst door de linkerkindknoop gaat of niet.
2. INTERSECTEER\_BLAD\_KNOOP(knoop, straal): Deze functie bepaalt de intersectie van de primitieven in de gegeven bladknoop met de gegeven straal.
3. INTERSECTEER(volume, straal): Deze functie bepaalt de intersectie van het omhullend volume van de boom en de straal. Het resultaat bevat een booleaans waarde, die aanduidt of het volume geraakt wordt door de straal. Als het volume geraakt wordt, bevat het ook de waarde tMin, de t waarde waarvoor de straal het volume binnengaat, en tMax, de t waarde waarvoor de straal het volume verlaat.

De eerste functie is afhankelijk van de specifieke representatie voor bladknopen / inwendige knopen die gebruikt worden voor het specifieke type *BSP* boom. Bij de besprekking van de knooprepresentaties in de volgende secties, wordt steeds de implementatie van deze functie voor die knooprepresentatie getoond. De tweede functie is hetzelfde voor alle knopen die besproken worden, elke driehoek in de bladknoop wordt op intersectie met de straal getest. De derde functie bepaalt de intersectie van het omhullend volume van de *BSP* boom met de straal. Bij alle besproken bomen wordt dit voorgesteld door een asgealigneerde balk. Intersectie berekenen met een asgealigneerde balk is triviaal.

#### 4. IMPLEMENTATIE

---



---

**Algoritme 2** Interseceren van een BSP boom

---

```

function INTERSECTEER(Boom b, Straal s)
    geraaktomhullendVolume, tMin, tMax  $\leftarrow$  INTERSECTEER( $b_{omhullendVolume}$ , s)
    if not geraaktomhullendVolume then
        return false
    end if
    geraakt  $\leftarrow$  false
    k  $\leftarrow$  bwortelKnoop
    stack  $\leftarrow$   $\emptyset$ 
    while k  $\neq$  None do
        if smaxT  $<$  tMin then
            break
        end if
        if kisInwendig then
            tVlak, linksEerst  $\leftarrow$  DOORKRUIS_INWENDIGE_KNOOP(k,s)
            if linksEerst then
                k1  $\leftarrow$  klinkerKind
                k2  $\leftarrow$  krechterKind
            else
                k1  $\leftarrow$  krechterKind
                k2  $\leftarrow$  klinkerKind
            end if
            if tVlak  $>$  tMax or tVlak  $\leq$  0 then
                k  $\leftarrow$  k1
            else if tVlak  $<$  tMin then
                k  $\leftarrow$  k2
            else
                ADD(stack, {k2, tMin : tVlak, tMax : tMax})
                k  $\leftarrow$  k1
                tMax  $\leftarrow$  tVlak
            end if
        else
            if INTERSECTEER_BLADKNOOP(k, s) then
                geraakt  $\leftarrow$  true
            end if
            if stack  $\neq$   $\emptyset$  then
                k, tMin, tMax  $\leftarrow$  POP(stack)
            else
                break
            end if
        end if
    end while
    return geraakt
end function

```

---

Inwendig	bits	Blad	bits
tSplit	32	primitief Offset	32
flags	2	flags	2
tweedeKindIndex	30	$n$	30
	64		64

Tabel 4.1: Voorstelling *Kd* knoop -

## 4.2 *Kd* boom

De implementatie van de *Kd* boom is gebaseerd op de *Kd* boom implementatie van pbrt. De *Kd* boom maakt gebruik van *Kd* knopen. Tabel 4.1 toont de representatie van zowel inwendige *Kd* knopen als blad *Kd* knopen. Zowel inwendige knopen als bladknopen worden voorgesteld met 64 bits en bevatten de *flags* variabele. Als de *flags* variabele gelijk is aan 3, is het een bladknoop. Bij inwendige knopen stelt die variabele de as voor waارlangs gesplitst wordt: 0 is x, 1 is y en 2 is z.

De inwendige knopen bevatten extra informatie over hun splitsingsvlak via *tSplit*. Deze *tSplit* variabele bepaalt de locatie van het vlak langs de as. De knopen van een boom worden opgeslagen in een lijst, bij een inwendige knoop wordt zijn linkerkindknoop opgeslagen op de volgende index in de lijst. De index van de rechterkindknoop, wordt opgeslagen in de *tweedeKindIndex* variabele.

De bladknopen bevatten informatie over hun primitieven via *n* en *primitiefOffset*. De *n* variabele stelt het aantal primitieven in de bladknoop voor. De *primitiefOffset* variabele verwijst naar de primitieven in de bladknoop. Als er maar één primitief in de knoop zit, wijst de variabele rechtstreeks naar het primitief. Als er meerdere primitieven in de knoop zitten, stelt het een index voor in een lijst. Elk element in die lijst met een index tussen *primitiefOffset* en *primitiefOffset + n* wijst dan naar een primitief in de bladknoop.

---

**Algoritme 3** Doorkruisen van een inwendige *Kd* knoop.

---

```
function DOORKRUIS__INWENDIGE__KNOOP(Kd Knoop k, Straal s)
    as < -ksplitsAs
    tVlak ← KD_VLAK_AFSTAND(ksplitPos, s,  $\frac{1}{s_d}$ , as)
    linksEerst ←  $\vec{s}_o[as] < k_{splitPos}$  or ( $\vec{s}_o[as] = k_{splitPos}$  and  $s_d[as] \leq 0$ )
    return tVlak, linksEerst
end function
```

---

Algoritme 3 toont het algoritme om inwendige *Kd* knopen te doorkruisen. Het gebruikt de functie KD\_VLAK\_INTERSECTIE (algoritme 4) om te intersecteren met een asgealigneerd vlak. De inverse van de richting van de straal ( $\frac{1}{s_d}$ ) moet voor

#### 4. IMPLEMENTATIE

---

**Algoritme 4** Intersectie tussen een asgealigneerd vlak en een straal.

---

```

function KD_VLAK_AFSTAND(splitPositie, s, inverseRichting, as)
    return ( $splitPos - \vec{s}_o[as]$ ) *  $inverseRichting[as]$ 
end function

```

---

elke straal slechts één keer berekend worden.

Algoritme 5 toont het algoritme om het beste splitsingsvlak te bepalen bij een  $Kd$  boom. Het gebruikt de SAH functie die de SAH kost berekent voor het splitsingsvlak. Deze functie gebruikt de oppervlaktes  $\mathcal{SA}_{rechts}$  en  $\mathcal{SA}_{links}$  van de gesplitste volumes en de aantallen primitieven  $n_{links}$  en  $n_{rechts}$  in de nieuwe knopen samen met de oppervlakte  $bouwknoop_{SA}$  van de huidige knoop. Het splitsen in kindvolumes en het berekenen van deze oppervlaktes is triviaal omdat de omhullende volumes asgealigneerde balken zijn. De waarden voor  $n_{links}$  en  $n_{rechts}$  worden efficiënt berekend door te sweepen.

**Algoritme 5** Beste split voor een bouwknoop b bij een  $Kd$  boom.

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    for as  $\in \{0, 1, 2\}$  do
         $\forall i : randen[2i] \leftarrow LINKERRAND(b_{primitieven}[i], as)$ 
         $\forall i : randen[2i + 1] \leftarrow RECHTERRAND(b_{primitieven}[i], as)$ 
        SORTEER(randen)
         $n_{links}, n_{rechts} \leftarrow 0, bouwknoop_n$ 
        for rand  $\in$  randen do
            if rand $_isRechts$  then
                 $n_{rechts} \leftarrow n_{rechts} + 1$ 
            end if
            kost  $\leftarrow$  SAH( $\mathcal{K}_d, \mathcal{K}_i, \mathcal{SA}_{links}, \mathcal{SA}_{rechts}, n_{links}, n_{rechts}, bouwknoops_{SA}$ )
            if kost  $<$  besteKost then
                besteKost, besteAs, besteT  $\leftarrow$  kost, as, randt
            end if
            if rand $_isLinks$  then
                 $n_{links} \leftarrow n_{links} + 1$ 
            end if
        end for
    end for
    return besteAs, besteT, besteKost
end function

```

---

Inwendig	bits	Blad	bits
tSplit	32	primitief	32
flags	$\lceil \log_2(k + 1) \rceil$	Offset	$\lceil \log_2(k + 1) \rceil$
tweedeKindIndex	$32 - \lceil \log_2(k + 1) \rceil$	flags	$32 - \lceil \log_2(k + 1) \rceil$
	64	n	64

Tabel 4.2: Voorstelling RBSP knoop -

### 4.3 RBSP boom

De implementatie van de *RBSP* boom is gebaseerd op de papers van Kammaje en Mora [KM07] en Budge et al [BCNJ08]. De *RBSP* boom maakt gebruik van *RBSP* knopen. Tabel 4.2 toont de representatie van zowel inwendige *RBSP* knopen als blad *RBSP* knopen. Deze representatie heeft dezelfde variabelen en totale grootte als de representatie van Kammaje en Mora [KM07], maar het aantal bits per variabele verschilt. Net als bij de *Kd* knoop worden zowel inwendige *RBSP* knopen als blad *RBSP* knopen voorgesteld met 64 bits en bevatten ze beide de *flags* variabele. Als de *flags* variabele gelijk is aan  $k$  (het aantal richtingen), is het een bladknoop. Bij inwendige knopen stelt die variabele de index voor van de richting waارlangs gesplitst wordt. De *flags* variabele heeft hierdoor  $\lceil \log_2(k + 1) \rceil$  bits nodig. Om te zorgen dat het totaal aantal bits op 64 blijft, wordt het aantal bits voor de *tweedeKindIndex* variabele bij de inwendige *RBSP* knoop en de *n* variable bij de blad *RBSP* knoop verlaagt tot  $32 - \lceil \log_2(k + 1) \rceil$ . Dit impliceert dat de *RBSP* boom  $2^{(\lceil \log_2(k+1) \rceil - 2)}$  keer minder knopen kan bevatten. Het maximaal aantal primitieven in een bladknoop daalt met dezelfde factor. Voor het aantal primitieven vormt dit geen probleem, aangezien bladknopen steeds kleine aantallen primitieven bevatten. Voor een *RBSP* boom met  $k = 13$  richtingen (maximaal  $2^{28}$  knopen), betekent dit dat de boom 4 keer minder knopen kan bevatten dan de *Kd* boom (maximaal  $2^{30}$  knopen).

Algoritme 6 toont het algoritme om inwendige *RBSP* knopen te doorkruisen. Het gebruikt de functie VLAK\_INTERSECTIE (algoritme 7) om te intersecteren met een willekeurig georiënteerd vlak. In vergelijking met de intersectie van een asgealigneerd vlak (algoritme 4), moeten twee extra scalaire producten en een deling uitgerekend worden. Aangezien het aantal richtingen waarmee deze scalaire producten berekend moeten worden, beperkt is, kunnen deze per straal éénmalig op voorhand berekend worden. Deze techniek is toegepast door Budge et al [BCNJ08]. Omdat het effect hiervan, zeker bij stijgende  $k$  waarden, niet altijd positief was, is dit niet toegepast.

Het algoritme om het beste splitsingsvlak te bepalen bij een *RBSP* boom is zeer gelijkaardig aan het algoritme voor de *Kd* boom. In plaats van te sweepen over de drie assen, wordt er geswept over alle  $k$  richtingen. Het bepalen van de linker- en rechterrand van de primitieven langs de as, vereist scalaire producten in tegenstelling tot bij de *Kd* boom. Het splitsen van de omhullende volumes en het berekenen van

#### 4. IMPLEMENTATIE

---

**Algoritme 6** Doorkruisen van een inwendige  $RBSP$  knoop.

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $RBSP$  Knoop k, Straal s)
    richtingId  $\leftarrow k_{splitsAs}$ 
     $tVlak, projOorsprong, invProjRichting \leftarrow VLAK_AFSTAND(richtingen[\vec{richtingId}],$ 
     $k_{splitPos}, s)$ 
     $linksEerst \leftarrow projOorsprong < k_{splitPos}$  or ( $projOorsprong = k_{splitPos}$  and
     $invProjRichting \leq 0$ )
    return  $tVlak, linksEerst$ 
end function
```

---

**Algoritme 7** Intersectie tussen een vlak en een straal.

---

```

function VLAK_AFSTAND( $\vec{normaal}$ , splitPositie, s)
     $projOorsprong \leftarrow normaal \cdot \vec{s_o}$ 
     $invProjRichting \leftarrow \frac{1}{normaal \cdot \vec{s_d}}$ 
    afstand  $\leftarrow (splitPos - projOorsprong) * invProjRichting$ 
    return projOorsprong, invProjRichting, afstand
end function
```

---

de oppervlaktes is complexer omdat met  $k$ -DOPs gewerkt moet worden in plaats van met asgealigneerde balken. Budge et al [BCNJ08] ontwikkelden een methode om deze splitsingsen en oppervlaktes incrementeel te berekenen tijdens het sweepen, deze methode wordt niet gebruikt.

#### 4.4 $RBSP^{Kd}$ boom

De  $RBSP^{Kd}$  boom is een nieuw concept dat in dit werk voor het eerst wordt voorgesteld. Het combineert het concept van de  $RBSP$  boom met de snelle  $Kd$  knoop doorkruistechniek van Ize et al [IWP08]. De  $RBSP^{Kd}$  knoop is identiek aan de  $RBSP$  knoop in termen van representatie (tabel 4.2), maar de doorkruismethode is aangepast om  $Kd$  knopen sneller te kunnen doorkruisen. Algoritme 8 toont de aangepaste versie van het doorkruisalgoritme.

Het algoritme om het beste splitsingsvlak te bepalen bij een  $RBSP^{Kd}$  boom wordt getoond in algoritme 9. De sweeping over de  $k$  richtingen wordt opgesplitst in twee delen: sweeping over de  $Kd$  richtingen en de sweeping over de  $BSP$  richtingen. Het eerste deel berekent de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  als doorkruiskost. Het tweede deel berekent twee  $SAH$  kosten, één met een  $\mathcal{K}_{d,BSP}$  kost die lineair afhankelijk is van het aantal driehoeken en één met een vaste  $\mathcal{K}_{d,BSP}$  kost. Deze eerste kost wordt rechtstreeks vergeleken met de beste kost van de  $Kd$  richtingen. De tweede kost wordt enkel gebruikt als er geen voordeelig  $Kd$  of  $BSP$  splitsingsvlak gevonden wordt via de eerste kost.

---

**Algoritme 8** Doorkruisen van een inwendige  $RBSP^{Kd}$  knoop.

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $RBSP^{Kd}$  Knoop k, Straal s)
    richtingId  $\leftarrow k_{splitsAs}$ 
    if richtingId < 3 then
        tVlak  $\leftarrow$  KD_VLAK_AFSTAND( $k_{splitPos}$ , s,  $\frac{1}{s_d}$ , richtingId)
        linksEerst  $\leftarrow \vec{s_o}[richtingId] < k_{splitPos}$  or ( $\vec{s_o}[richtingId] = k_{splitPos}$  and
         $s_d[richtingId] \leq 0$ )
        return tVlak, linksEerst
    else
        tVlak, projOorsprong, invProjRichting  $\leftarrow$ 
        VLAK_AFSTAND(richtingen[richtingId],  $k_{splitPos}$ , s)
        linksEerst  $\leftarrow$  projOorsprong <  $k_{splitPos}$  or (projOorsprong =  $k_{splitPos}$ 
        and invProjRichting  $\leq 0$ )
        return tVlak, linksEerst
    end if
end function

```

---

Inwendig	bits	Blad	bits
tSplit	32	primitief Offset	32
flags	1	flags	1
tweedeKindIndex	31	n	31
splitRichting	96		
	160		64

Tabel 4.3: Voorstelling  $BSP$  knoop -

## 4.5 $BSP_{IZE}$ boom

De  $BSP_{IZE}$  boom maakt gebruik van algemene  $BSP$  knopen. Tabel 4.3 toont de representatie van zowel inwendige  $BSP$  knopen als blad  $BSP$  knopen. De representatie van de  $BSP$  knopen is identiek aan de representatie van de paper van Ize et al [IWP08]. Net als bij de  $Kd$  en  $RBSP$  knopen bevatten zowel inwendige  $BSP$  knopen als blad  $BSP$  knopen de *flags* variabele. De *flags* variabele wordt bij de  $BSP$  boom enkel gebruikt om aan te geven of het een inwendige knoop (waarde 0) of een bladknoop (waarde 1) is. Om de oriëntatie van het splitsingsvlak voor te stellen, zijn drie vloottend komma getallen nodig. De *splitsRichting* variabele bij inwendige  $BSP$  knopen stelt de normaal van het splitsingsvlak voor. De nodige geheugenruimte voor de inwendige  $BSP$  knopen en de blad  $BSP$  knopen is verschillend, maar aangezien elke knoop evenveel geheugen moet innemen, neemt elke knoop 160 bits geheugen in. Dit is 2.5 keer meer dan de  $Kd$  en  $RBSP$  knopen en toont één van de nadelen van algemene  $BSP$  bomen.

Algoritme 10 toont het algoritme om inwendige  $BSP$  knopen te doorkruisen. Het

**Algoritme 9** Beste split voor een bouwknoop b bij een  $RBSP^{Kd}$  boom.

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    besteAsVast, besteTVast, besteKostVast  $\leftarrow$  None, None,  $\infty$ 
    for as  $\in \{0, 1, 2\}$  do
        Sorteer eindpunten langs de as
        for elk vlak langs de as do
            Bereken de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  als doorkruiskost
            Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak
        end for
    end for
    for as  $\in \{3, 4, \dots, k - 1\}$  do
        Sorteer eindpunten langs de as
        for elk vlak langs de as do
            Bereken de  $SAH$  kost met een  $\mathcal{K}_{d,BSP}$  die lineair afhankelijk is van het
            aantal primitieven als doorkruiskost
            Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak
            Bereken de  $SAH$  kost met een vaste  $\mathcal{K}_{d,BSP}$  als doorkruiskost
            Update  $besteAsVast$ ,  $besteTVast$ ,  $besteKostVast$  als dit splitsingsvlak
            beter is dan het huidige beste splitsingsvlak met vaste  $BSP$  doorkruiskost.
        end for
    end for
    if  $besteKost < bouwknoop_n * \mathcal{K}_i$  then
        return  $besteAs$ ,  $besteT$ ,  $besteKost$ 
    end if
    if  $besteKostVast < bouwknoop_n * \mathcal{K}_i$  then
        return  $besteAsVast$ ,  $besteTVast$ ,  $besteKostVast$ 
    end if
    return None, None,  $\infty$ 
end function

```

---

gebruikt, net als de  $RBSP$  knoop doorkruising, de functie VLAK\_INTERSECTIE (algoritme 7) om te interseceren met een willekeurig georiënteerd vlak.

---

**Algoritme 10** Doorkruisen van een inwendige  $BSP$  knoop.

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $BSP$  Knoop k, Straal s)
     $tVlak, projOorsprong, invProjRichting \leftarrow VLAK_AFSTAND(k.splitsRichting,$ 
     $k.splitPos, s)$ 
     $linksEerst \leftarrow projOorsprong < k.splitPos \text{ or } (projOorsprong = k.splitPos \text{ and}$ 
     $invProjRichting \leq 0)$ 
    return  $tVlak, linksEerst$ 
end function

```

---

Het algoritme om het beste splitsingsvlak te bepalen bij de  $BSP_{IZE}$  boom wordt getoond in algoritme 11. Voor de  $Kd$  richtingen kan sweeping gebruikt worden, om het efficiënt te berekenen. De  $BSP$  vlakken worden per driehoek bekeken en maken gebruik van een  $BVH$  boom om efficiënt te berekenen hoeveel driehoeken links en rechts van het splitsingsvlak liggen. Ize et al [IWP08] gebruiken een  $BVH$  boom met sferen als omhullende volumes, deze implementatie maakt gebruik van de bestaande  $BVH$  implementatie van pbrt die werkt met asgealigneerde balken als omhullende volumes. Bij de  $BSP_{IZE}$  boom is het splitsen van de omhullende volumes en het berekenen van de oppervlaktes complexer omdat met algemene convexe veelvlakken gewerkt moet worden in plaats van met asgealigneerde balken. Elke van deze convexe veelvlakken kan beschouwd worden als een k-DOP met andere richtingen en is ook op deze manier geïmplementeerd. Voor elke splitsingsrichting wordt gekeken of een voorouder van de huidige knoop al gesplitst is volgens een richting die binnen een hoek van  $0.5^\circ$  ligt. De gebruikte richtingen worden gecontroleerd in volgorde dat ze gebruikt zijn. Als een vorige richting gevonden wordt, wordt er opnieuw gesplitst volgens deze vorige richting in plaats van de nieuwe richting.

## 4.6 $BSP_{IZE}^{Kd}$ boom

De  $BSP_{IZE}^{Kd}$  boom maakt gebruik van  $BSP^{Kd}$  knopen. Deze  $BSP^{Kd}$  knopen zijn identiek aan  $BSP$  knopen, met uitzondering van het aantal bits gebruikt door de *flags* en *tweedeKindIndex* variabelen. De *flags* variabele heeft drie bits nodig, de waarden 0, 1 en 2 stellen de x,y en z as voor, waarde 3 betekent dat het een bladknoop is en waarde 4 betekent dat het een algemene  $BSP$  knoop is met willekeurig splitsingsvlak. Om het aantal bits een veelvoud van 32 te laten blijven, wordt de grootte van de *tweedeKindIndex* variabele met 2 verminderd. Hierdoor kan de  $BSP_{IZE}^{Kd}$  boom vier keer minder knopen bevatten dan de  $BSP_{IZE}$  boom. Tabel 4.4 toont de representatie van zowel inwendige  $BSP^{Kd}$  knopen als blad  $BSP^{Kd}$  knopen.

Het grootste verschil tussen  $BSP$  knopen en  $BSP^{Kd}$  knopen is dat de  $BSP^{Kd}$  knopen gebruik maken van de snellere  $Kd$  doorkruistechniek voor inwendige  $Kd$  knopen. Algoritme 12 toont de aangepaste versie van het doorkruisalgoritme voor  $BSP^{Kd}$  knopen.

#### 4. IMPLEMENTATIE

---

**Algoritme 11** Beste split voor een bouwknoop  $b$  bij een  $BSP_{SIZE}$  boom.

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    besteAsVast, besteTVast, besteKostVast  $\leftarrow$  None, None,  $\infty$ 
    for as  $\in \{0, 1, 2\}$  do
        Sorteer eindpunten langs de as
        for elk vlak langs de as do
            Bereken de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  als doorkruiskost
            Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak
        end for
    end for
    bvh  $\leftarrow$  BOUW_BVH( $b$ )
    for primitief  $\in$  bouwKnoopprimitieven do
        for elk splitsingsvlak bij het primitief do
            Bereken het aantal driehoeken links en rechts van het splitsingvlak met
            de  $BVH$  boom
            Bereken de  $SAH$  kost met een vaste  $\mathcal{K}_{d,BSP}$ 
            Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak.
        end for
    end for
    return  $besteAs, besteT, besteKost$ 
end function

```

---

Inwendig	bits	Blad	bits
tSplit	32	primitief	32
flags	3	Offset	3
tweedeKindIndex	29	$n$	29
splitRichting	96		
	160		64

Tabel 4.4: Voorstelling  $BSP^{Kd}$  knoop -

---

**Algoritme 12** Doorkruisen van een inwendige  $BSP^{Kd}$  knoop.

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $BSP^{Kd}$  Knoop k, Straal s)
    isKdKnoop  $\leftarrow k_{flags} < 4$ 
    if isKdKnoop then
        as  $\leftarrow k_{flags}$ 
        tVlak  $\leftarrow$  KD_VLAK_AFSTAND( $k_{splitPos}$ , s,  $\frac{1}{s_d}$ , as)
        linksEerst  $\leftarrow \vec{s}_o[as] < k_{splitPos}$  or ( $\vec{s}_o[as] = k_{splitPos}$  and  $s_d[as] \leq 0$ )
        return tVlak, linksEerst
    else
        tVlak, projOorsprong, invProjRichting  $\leftarrow$ 
        VLAK_AFSTAND( $k_{splitsRichting}$ ,  $k_{splitPos}$ , s)
        linksEerst  $\leftarrow$  projOorsprong  $< k_{splitPos}$  or (projOorsprong  $= k_{splitPos}$ 
        and invProjRichting  $\leq 0$ )
        return tVlak, linksEerst
    end if
end function

```

---

Het algoritme om het beste splitsingsvlak te bepalen bij de  $BSP_{IZE}^{Kd}$  boom wordt getoond in algoritme 13 en is zeer gelijkaardig aan het algoritme voor de  $BSP_{IZE}$  boom. Door de snelle  $Kd$  doorkruising is het nodig om aparte doorkruiskosten voor  $Kd$  en  $BSP$  te gebruiken. Zoals besproken in sectie 2.5 wordt gebruik gemaakt van een  $\mathcal{K}_{d,BSP}$  die lineair afhankelijk is van het aantal driehoeken om  $Kd$  knopen te bevoordelen. Als er geen nuttig  $Kd$  splitsingsvlak of  $BSP$  splitsingsvlak met deze  $\mathcal{K}_{d,BSP}$  gevonden wordt, worden de  $BSP$  splitsingsvlakken opnieuw bekeken met een vaste  $\mathcal{K}_{d,BSP}$ .

## 4.7 $BSP_{SWEEP}$ boom

De  $BSP_{SWEEP}$  en  $BSP_{SWEEP}^{Kd}$  bomen maken gebruik van respectievelijk de  $BSP$  en  $BSP^{Kd}$  knopen. Hierdoor zijn de doorkruismethodes voor inwendige knopen identiek aan die gebruikt bij de  $BSP_{IZE}$  en  $BSP_{IZE}^{Kd}$  bomen. Algoritme 14 toont het algoritme om het beste splitsingsvlak te bepalen bij een  $BSP_{SWEEP}$  boom. Dit algoritme gebruikt de BEPAAL\_RICHTINGEN functie die de specifieke soort  $BSP_{SWEEP}$  boom bepaalt. Zoals de naam suggereert wordt in elke knoop langs elk splitsrichting geswept om het beste splitsingsvlak op een efficiënte manier te vinden. Hierdoor is er geen nood aan een hulpstructuur zoals bij de  $BSP_{IZE}$  boom.

**BSP<sub>random</sub> boom** De  $BSP_{random}$  boom kiest op elk niveau k random richtingen. Algoritme 15 toont de implementatie van de BEPAAL\_RICHTINGEN functie voor de  $BSP_{random}$  boom. Deze functie gebruikt de UPDATE\_ZIN functie die zorgt dat zin van de richting zo gekozen wordt dat de x-component van de vector positief is. Dit is nodig omdat de vlakken langs een richting identiek zijn aan de vlakken langs zijn omgekeerde richting.

**Algoritme 13** Beste split voor een bouwknoop  $b$  bij een  $BSPIZE$  boom.

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    besteAsVast, besteTVast, besteKostVast  $\leftarrow$  None, None,  $\infty$ 
    for as  $\in \{0, 1, 2\}$  do
        Sorteer eindpunten langs de as
        for elk vlak langs de as do
            Bereken de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  als doorkruiskost
            Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak
            end for
        end for
        bvh  $\leftarrow$  BOUW_BVH( $b$ )
        for primitief  $f \in bouwKnoop_{primitieven}$  do
            for elk splitsingsvlak bij het primitief do
                Bereken het aantal driehoeken links en rechts van het splitsingvlak met
                de  $BVH$  boom
                Bereken de  $SAH$  kost met een  $\mathcal{K}_{d,BSP}$  die lineair afhankelijk is van het
                aantal primitieven als doorkruiskost
                Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is dan
                het huidige beste splitsingsvlak
                Bereken de  $SAH$  kost met een vaste  $\mathcal{K}_{d,BSP}$  als doorkruiskost
                Update  $besteAsVast$ ,  $besteTVast$ ,  $besteKostVast$  als dit splitsingsvlak
                beter is dan het huidige beste splitsingsvlak met vaste  $BSP$  doorkruiskost.
            end for
        end for
        if  $besteKost < bouwknop_n * \mathcal{K}_i$  then
            return  $besteAs, besteT, besteKost$ 
        end if
        if  $besteKostVast < bouwknop_n * \mathcal{K}_i$  then
            return  $besteAsVast, besteTVast, besteKostVast$ 
        end if
        return None, None,  $\infty$ 
    end function

```

---

**Algoritme 14** Beste split voor een bouwknoop b bij een  $BSP_{SWEEP(+)}$  boom.

```
function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost ← None, None, ∞
    richtingen ← BEPAAL_RICHTINGEN(bouwKnoop, k)
    for richting ∈ richtingen do
        Sorteer eindpunten volgens de richting
        for elk vlak volgens de richting do
            Bereken de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  of  $\mathcal{K}_{d,BSP}$  als doorkruiskost
            Update besteAs, besteT, besteKost als dit splitsingsvlak beter is dan
            het huidige beste splitsingsvlak
        end for
    end for
    return bestAs, besteT, besteKost
end function
```

---

**Algoritme 15** Generatie richtingen voor de  $BSP_{random}$  boom.

```
function BEPAAL_RICHTINGEN(b, k)
    richtingen ← ∅
    for i ∈ {0, 1, 2, ..., k - 1} do
        φ ← UNIFORM(0, 2π)
        θ ← acos(UNIFORM(-1, 1))
        richting ← {sin(θ)cos(φ), sin(θ)sin(φ), cos(θ)}
        richting ← UPDATE_ZIN(richting)
        ADD(richtingen, richting)
    end for
    return richtingen
end function
```

---

**BSP<sub>wn</sub> boom** De  $BSP_{wn}$  boom kiest op elk niveau de normalen van k driehoeken als richtingen. Als er minder dan k driehoeken zijn, worden de normalen van alle driehoeken gebruikt. Algoritme 16 toont de implementatie van de BEPAAL\_RICHTINGEN functie voor de  $BSP_{wn}$  boom.

**Algoritme 16** Generatie richtingen voor de  $BSP_{wn}$  boom.

```
function BEPAAL_RICHTINGEN(b, k)
    richtingen ← ∅
    while richtingenlengte < MIN(k - 1, bn - 1) do
        richting ← bprimtieven[RANDOM_INT(0, bn)].normaal
        richting ← UPDATE_ZIN(richting)
        ADD(richtingen, richting)
    end while
    return richtingen
end function
```

---

#### 4. IMPLEMENTATIE

---

**BSP<sub>cn</sub> boom** De  $BSP_{cn}$  boom kiest op elk niveau  $k$  richtingen door de normalen te clusteren in  $k$  groepen via K-means clustering. De centra van de clusters zijn de gebruikte richtingen. Algoritme 17 toont de implementatie van de BEPAAL\_RICHTINGEN functie voor de  $BSP_{cn}$  boom. Als er tijdens het bepalen van de clusters, een cluster leeg wordt, worden alle clusters opnieuw willekeurig gegenereerd.

---

**Algoritme 17** Generatie richtingen voor de  $BSP_{cn}$  boom.

---

```

function BEPAAL_RICHTINGEN( $b, k$ )
     $\forall i \in \{0, 1, \dots, b_n - 1\} : normalen[i] \leftarrow \text{UPDATE\_ZIN}(b_{\text{primitieven}}[i]_{\text{normaal}})$ 
    if  $b_n \leq k$  then
        return  $normalen$ 
    end if
     $clusterCentra \leftarrow \emptyset$ 
    while  $clusterCentra_{\text{lengte}} < k$  do
         $clusterCentrum \leftarrow normalen[\text{RANDOM\_INT}(0, b_n)]$ 
        ADD( $clusterCentra$ ,  $clusterCentrum$ )
    end while
     $iteratie \leftarrow 0$ 
     $oudeClusterCentra \leftarrow clusterCentra$ 
    while  $iteratie < maxIteraties$  and ( $iteraties = 0$  or  $oudeClusterCentra \neq clusterCentra$ ) do
         $\forall i \in \{0, 1..k - 1\} : clusters[i] \leftarrow \{\vec{n} | \vec{n} \in normalen, \neg \exists j : j \neq i, \angle(clusterCentra[i], \vec{n}) > \angle(clusterCentra[j], \vec{n})\}$ 
         $oudeClusterCentra \leftarrow clusterCentra$ 
        for  $i \in \{0, 1, 2..k - 1\}$  do
            if  $clusters[i] = \emptyset$  then
                 $clusterCentra \leftarrow \emptyset$ 
                while  $clusterCentra_{\text{lengte}} < k$  do
                     $clusterCentrum \leftarrow normalen[\text{RANDOM\_INT}(0, b_n)]$ 
                    ADD( $clusterCentra$ ,  $clusterCentrum$ )
                    break
                end while
            end if
             $clusterCentra[i] \leftarrow \text{GEMIDDELDE}(clusters[i])$ 
        end for
         $iteratie \leftarrow iteratie + 1$ 
    end while
    return  $clusterCentra$ 
end function

```

---

De implementaties van de  $BSP_{random+}$ ,  $BSP_{wn+}$  en  $BSP_{cn+}$  bomen zijn bijna identiek aan de implementaties van de  $BSP_{random}$ ,  $BSP_{wn}$  en  $BSP_{cn}$  bomen. Het enige verschil is dat de  $Kd$  richtingen steeds gebruikt worden en dat er maar  $k - 3$  richtingen gegenereert worden via de BEPAAL\_RICHTINGEN functies. De imple-

mentaties van de  $BSP_{random+}^{Kd}$ ,  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen hebben een aangepaste BEPAAL\_BESTE\_SPLIT methode die de aangepaste  $\mathcal{SA}$  heuristiek van de  $BSP_{SIZE}^{Kd}$  boom toepast. Algoritme 18 toont deze implementatie.

---

**Algoritme 18** Beste split voor een bouwknoop  $b$  bij een  $BSP^{Kd}$  boom.

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    besteAsVast, besteTVast, besteKostVast  $\leftarrow$  None, None,  $\infty$ 
    richtingen  $\leftarrow$  BEPAAL_RICHTINGEN( $b$ )
    for richting  $\in$  richtingen do
        Sorteer eindpunten volgens de richting
        if richting is  $KdRichting$  then
            for elk vlak volgens de richting do
                Bereken de  $SAH$  kost met  $\mathcal{K}_{d,Kd}$  als doorkruiskost
                Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is
                dan het huidige beste splitsingsvlak
            end for
        else
            for elk vlak volgens de richting do
                Bereken de  $SAH$  kost met een  $\mathcal{K}_{d,BSP}$  die lineair afhankelijk is van
                het aantal primitieven als doorkruiskost
                Update  $besteAs$ ,  $besteT$ ,  $besteKost$  als dit splitsingsvlak beter is
                dan het huidige beste splitsingsvlak
            end for
        end if
    end for
    if besteKost  $<$  bouwknoop $_n * \mathcal{K}_i$  then
        return  $besteAs$ ,  $besteT$ ,  $besteKost$ 
    end if
    if besteKostVast  $<$  bouwknoop $_n * \mathcal{K}_i$  then
        return  $besteAsVast$ ,  $besteTVast$ ,  $besteKostVast$ 
    end if
    return None, None,  $\infty$ 
end function

```

---



# Hoofdstuk 5

## Resultaten

Dit hoofdstuk bespreekt de resultaten van de nieuwe  $BSP_{SWEEP}$  bomen. In sectie 5.1 worden een aantal praktische aspecten omtrent de testopstelling besproken zoals de scenes, de gebruikte parameters en het gebruikte computersysteem. Daarna bespreken we de invloed van het aantal richtingen op de vorm (sectie 5.2.1) en kwaliteit (sectie 5.2.2) van de negen soorten  $BSP_{SWEEP}$  bomen. De laatste sectie (sectie 5.3) vergelijkt de beste  $BSP_{SWEEP}$  bomen met de bomen besproken in hoofdstuk 2.

### 5.1 Praktische aspecten

Alle  $BSP$  bomen zijn geïmplementeerd in het pbrt framework zoals besproken in hoofdstuk 4. Dit framework maakt geen gebruik van optimalisaties zoals SIMD instructies en de GPU. Hierdoor kan de kracht van de verschillende bomen objectief vergeleken worden.

Figuur 5.1 toont de gebruikte testscenes en tabel 5.1 toont informatie over deze scenes. De Killeroo Been scene is een scene waarbij algemene  $BSP$  bomen voordeel hebben ten opzichte van de  $Kd$  boom omwille van de complexe geometrie waarop is ingezoomd. De drie andere scenes zijn realistische indoor scenes. Alle scenes maken gebruik van Halton sampling met een specifiek aantal stralen per pixel  $spp$ . De globale belichting wordt berekend via *path tracing* waarbij steeds een maximale diepte van 5 gebruikt wordt, behalve bij de museum scene waar enkel directe belichting gebruikt wordt.

Tabel 5.2 toont de parameterwaarden die gebruikt zijn bij de testen. De formule voor de maximale diepte is afhankelijk van het aantal driehoeken  $n$  en is bedacht door Havran en Bittner [HB02] voor  $Kd$  bomen. Deze parameterwaarden zijn niet geoptimaliseerd en kunnen mogelijks beter worden afgesteld. De specificaties van het gebruikte computersysteem worden getoond in figuur 5.3.

## 5. RESULTATEN

---



(a) Killeroo Been scene (b) Sponza scene (c) Conference scene (d) Museum scene

Figuur 5.1: Testscenes - De testscenes gebruikt om de nieuwe *BSP* te testen

Scene	Aantal driehoeken	Resolutie	Sampling per pixel	Belichting
Killeroo Been	33264	5000x5000	Halton, 8spp	<i>path tracing</i> , diepte 5
Sponza	227309	700x700	Halton, 64spp	<i>path tracing</i> , diepte 5
Conference	123651	1000x676	Halton, 64spp	<i>path tracing</i> , diepte 5
Museum	1462840	700X700	Halton, 64spp	<i>path tracing</i> , diepte 1

Tabel 5.1: Statistieken Testscenes - Statistieken van de testscenes gebruikt om de nieuwe *BSP* te testen

Parameter	Waarde
$\mathcal{K}_i$	80
$\mathcal{K}_{d,Kd}$	1
$\mathcal{K}_{d,BSP}$	5
$n_{max}$	1
$d_{max}$	$k1\log_2(n) + k2$ met $k1 = 1.6$ en $k2 = 2$

Tabel 5.2: Gebruikte parameterwaarden - De waarden voor de parameters gebruikt om de nieuwe *BSP* te testen

Parameter	Waarde
CPU	Intel Core i7-6700HQ CPU @ 2.60GHz x 8
Geheugen	16 GB DDR4
Besturingssysteem	Ubuntu 18.04.2 LTS 64-bit

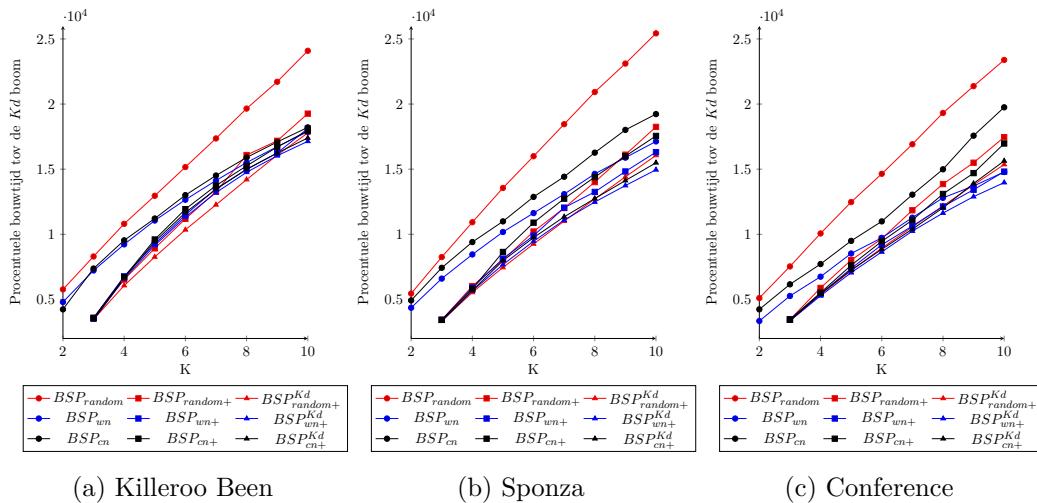
Tabel 5.3: Specificaties computersysteem - De specificaties van het gebruikte computersysteem.

## 5.2 Afhankelijkheid van aantal richtingen

In deze sectie wordt de afhankelijkheid van  $BSP_{SWEEP}$  bomen van het aantal gebruikte richtingen  $k$ , besproken. Alle negen varianten hebben bovenstaande scènes zeven keer gerenderd voor  $k$ -waarden van 2 tot en met 10. De zes varianten die gebruik maken van de  $Kd$  richtingen, moeten altijd een  $k$  waarde van minstens 3 hebben, dus deze zijn niet gerenderd voor  $k = 2$ . Voor elke combinatie van  $BSP$  boom en  $k$ -waarde wordt de uitvoering waarvan de rendertijd gelijk is aan de mediaan van de rendertijden van de zeven uitvoeringen, gebruikt als representatieve uitvoering. Alle onderstaande analyses maken gebruik van die uitvoeringen.

### 5.2.1 Bespreking bomen

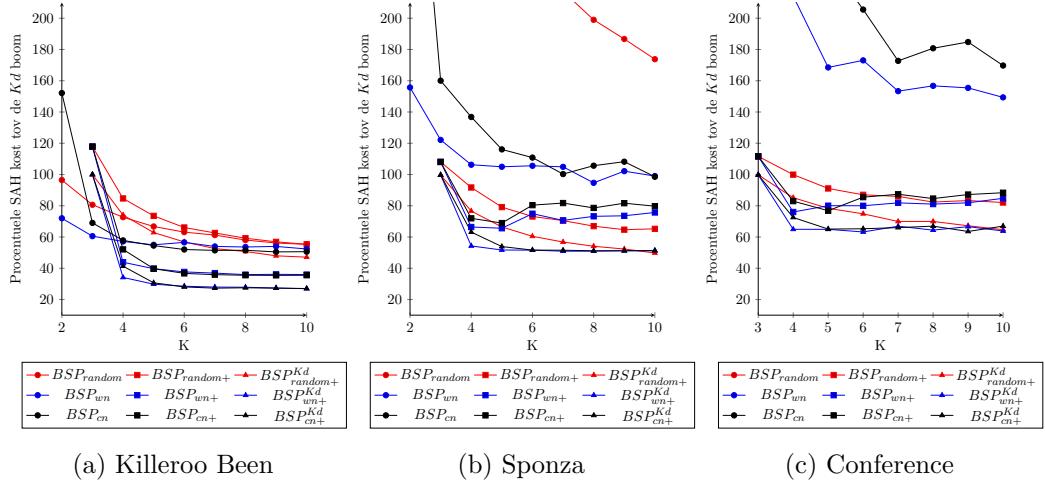
**Bouwtijd** Figuur 5.2 toont de bouwtijden ten opzichte van de bouwtijden van de  $Kd$  boom. De bouwtijden van de  $BSP_{SWEEP}$  bomen zijn twee ordergroottes groter dan die van de  $Kd$  boom. De grafieken voor de verschillende scènes (met een verschillend aantal driehoeken) zijn zeer gelijkaardig, dus is de complexiteit voor het bouwen van  $Kd$  en  $BSP_{SWEEP}$  bomen op dezelfde manier afhankelijk van het aantal driehoeken. De grafieken van de  $BSP_{random}$  bomen zijn alle drie bijna perfecte rechten. De bouwtijd is lineair afhankelijk van  $k$  omdat in elke knoop over  $k$  richtingen geswept wordt. Bij de  $BSP_{wn}$  en  $BSP_{cn}$  bomen is de afhankelijkheid sublineair omdat er in knopen met minder dan  $k$  driehoeken, over minder dan  $k$  richtingen geswept wordt. De  $BSP_{random+}^{Kd}$ ,  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen met  $k = 3$  zijn identiek aan de  $Kd$  boom, maar gebouwd met convexe veelvlakken in plaats van asgealigeneerde balken. Hieruit kunnen we afleiden dat het gebruik van convexe veelvlakken tijdens het bouwen ongeveer 25 (2500%) keer trager is dan het gebruik van asgealigeneerde balken.



Figuur 5.2: Bouwtijd in functie van  $k$  - Deze grafieken tonen voor elke scène de procentuele bouwtijd van de  $BSP_{SWEEP}$  bomen ten opzichte van de bouwtijd van de  $Kd$  boom in functie van  $k$ .

## 5. RESULTATEN

**SAH** De *SAH* wordt gebruikt om in elke knoop op een greedy manier het beste splitsingsvlak te bepalen. De totale *SAH* kost van de boom kan echter ook berekend worden. Bij de  $BSP_{SWEEP}$  bomen die gebruik maken van  $Kd$  richtingen, maar deze behandelen als *BSP* vlakken, wordt  $\mathcal{K}_{d,BSP}$  gebruikt als doorkruiskost. Bij de  $BSP_{SWEEP}^{Kd}$  bomen wordt de  $\mathcal{K}_{d,Kd}$  gebruikt als doorkruiskost voor de  $Kd$  knopen. Voor alle *BSP* splitsingsvlakken - ook degene die gekozen worden met de  $\mathcal{K}_{d,BSP}$  die lineair afhankelijk is van het aantal driehoeken - wordt de vaste  $\mathcal{K}_{d,BSP}$  gebruikt om de totale *SAH* kost te berekenen. Op deze manier kan de totale *SAH* kost van alle bomen vergeleken worden. Figuur 5.3 toont deze totale *SAH* kost van de  $BSP_{SWEEP}$  bomen ten opzichte van die van de  $Kd$  boom. Voor de Killeroo Been scene valt op dat de  $BSP_{random}^{k=3}$  boom, de  $BSP_{wn}^{k=3}$  boom en de  $BSP_{cn}^{k=3}$  boom een lagere *SAH* kost hebben dan de  $Kd$  boom. Bij de andere scenes zijn de bomen die geen  $Kd$  richtingen gebruiken, duidelijk ondergeschikt. Het valt ook op dat het toevoegen van één richtingen per knoop bovenop de  $Kd$  richtingen al zorgt voor een sterke vermindering van de *SAH* kost. Bij de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen is deze daling zelfs al 40%. De  $BSP_{random}$  bomen genereren bij stijgend k-waarden steeds bomen met een lagere *SAH* kost, terwijl de *SAH* kosten bij de  $BSP_{wn+}^{(Kd)}$  en  $BSP_{cn+}^{(Kd)}$  bomen nog maar amper dalen voor k waardes hoger dan 4. De  $BSP_{wn+}$  en  $BSP_{cn+}$  bomen genereren bij k-waardes groter dan 4 zelfs bomen met hogere *SAH* kosten.

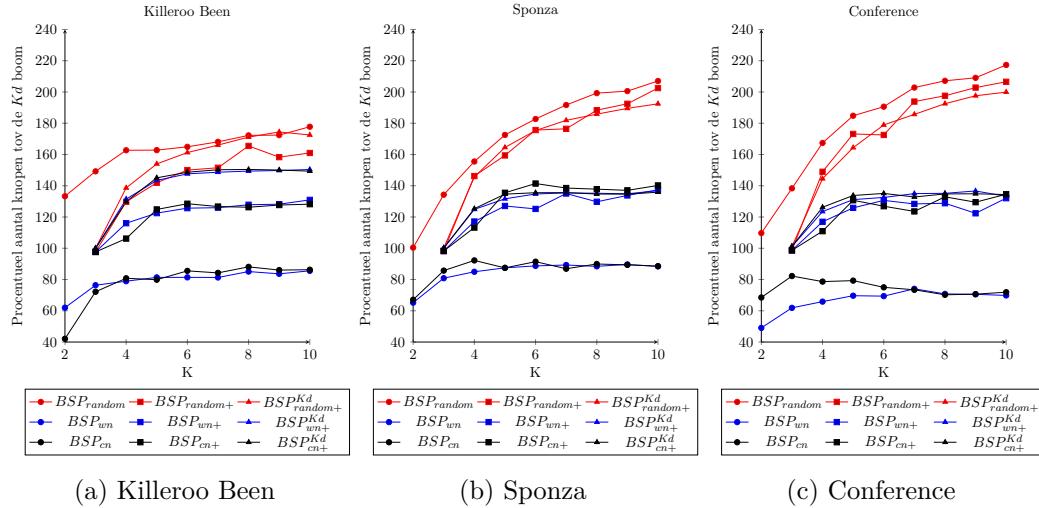


Figuur 5.3: *SAH* kost in functie van k - Deze grafieken tonen voor elke scene de procentuele *SAH* kost van de  $BSP_{SWEEP}$  bomen ten opzichte van de *SAH* kost van de  $Kd$  boom in functie van k. De  $BSP_{SWEEP}$  bomen die de  $Kd$  richtingen niet gebruiken, geven voor kleine waarden van k, hoge *SAH* kosten, deze worden niet getoond.

**Aantal knopen** Figuur 5.4 toont het aantal inwendige knopen bij de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal inwendige knopen bij de  $Kd$  boom. De  $BSP_{random(+)}^{(Kd)}$  bomen hebben duidelijk meer inwendige knopen dan de anderen. Een mogelijke verklaring hiervoor zou kunnen zijn dat deze bomen wel vaak een splitsingsvlak

## 5.2. Afhankelijkheid van aantal richtingen

vinden, maar dat deze van minder goede kwaliteit zijn, waardoor veel driehoeken in beide kindknopen zitten. De  $BSP_{wn(+)}^{(Kd)}$  en  $BSP_{cn(+)}^{(Kd)}$  bomen hebben een zeer gelijkaardig aantal knopen, zeker voor grotere k-waarden. Het aantal knopen stijgt voor stijgende k-waarden, maar deze stijging vlakt af vanaf een k-waarde van ongeveer 5.

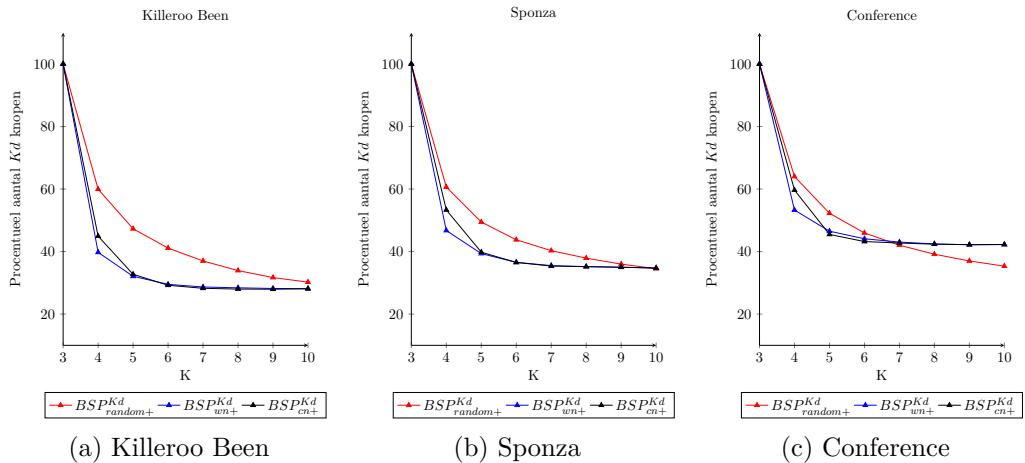


Figuur 5.4: Aantal inwendige knopen in functie van k - Deze grafieken tonen voor elke scène het procentueel aantal inwendige knopen van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal inwendige knopen van de  $Kd$  boom in functie van  $k$ .

**Aantal  $Kd$  knopen** Voor de  $BSP_{SWEEP}^{Kd}$  bomen is het interessant om te weten hoeveel inwendige knopen,  $Kd$  knopen zijn en hoeveel er  $BSP$  knopen zijn. Figuur 5.5 toont het procentuele aantal  $Kd$  knopen. Het procentueel aantal  $Kd$  knopen neemt zeer snel af met een stijgend aantal richtingen en convergeert naar een vast percentage. De drie varianten convergeren naar dezelfde waarde omdat de  $Kd$  knopen door de aangepaste *SAH* voornamelijk in de bovenste niveaus van de boom voorkomen, zoals getoond wordt in figuur 5.6. De bovenste niveaus van de drie varianten zijn hierdoor zeer gelijkaardig en op lagere niveaus worden bijna uitsluitend  $BSP$  splitsingen gebruikt. De  $BSP_{random+}^{Kd}$  boom gebruikt voor lagere k-waarden meer  $Kd$  vlakken omdat het geen goede  $BSP$  splitsingsvlakken vindt. Voor hogere k-waarden toont figuur 5.6 dat de  $BSP_{random+}^{Kd}$  boom procentueel meer  $BSP$  knopen heeft op hogere niveaus dan de twee andere varianten, hierdoor kan het dat de  $BSP_{random+}^{Kd}$  boom procentueel minder  $Kd$  knopen heeft. Dit gebeurt bijvoorbeeld bij de conference scène. Figuur 5.6 toont ook dat bij de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen de verdeling van  $Kd$  -  $BSP$  knopen voor verschillende dieptes niet meer verandert vanaf een k-waarde van ongeveer 6. Het kiezen van meer dan 6 richtingen lijkt niet nuttig.

## 5. RESULTATEN

---



Figuur 5.5: Procentueel aantal  $Kd$  knopen - Deze grafieken tonen voor elke scène het procentueel aantal  $Kd$  knopen van de  $BSP_{SWEEP}^{Kd}$  bomen ten opzichte van het totaal aantal inwendige knopen in functie van  $k$ .

### 5.2.2 Kwaliteit bomen

**Rendertijd** De rendertijd is de belangrijkste waarde om de kwaliteit van een boom te vergelijken. Figuur 5.7 toont de rendertijden ten opzichte van de rendertijden van de  $Kd$  boom.

**Aantal intersecties** Figuur 5.8

**Aantal doorkruisingen** Figuur 5.9

**$Kd$  doorkruisingen** Figuur 5.10

## 5.3 Vergelijking

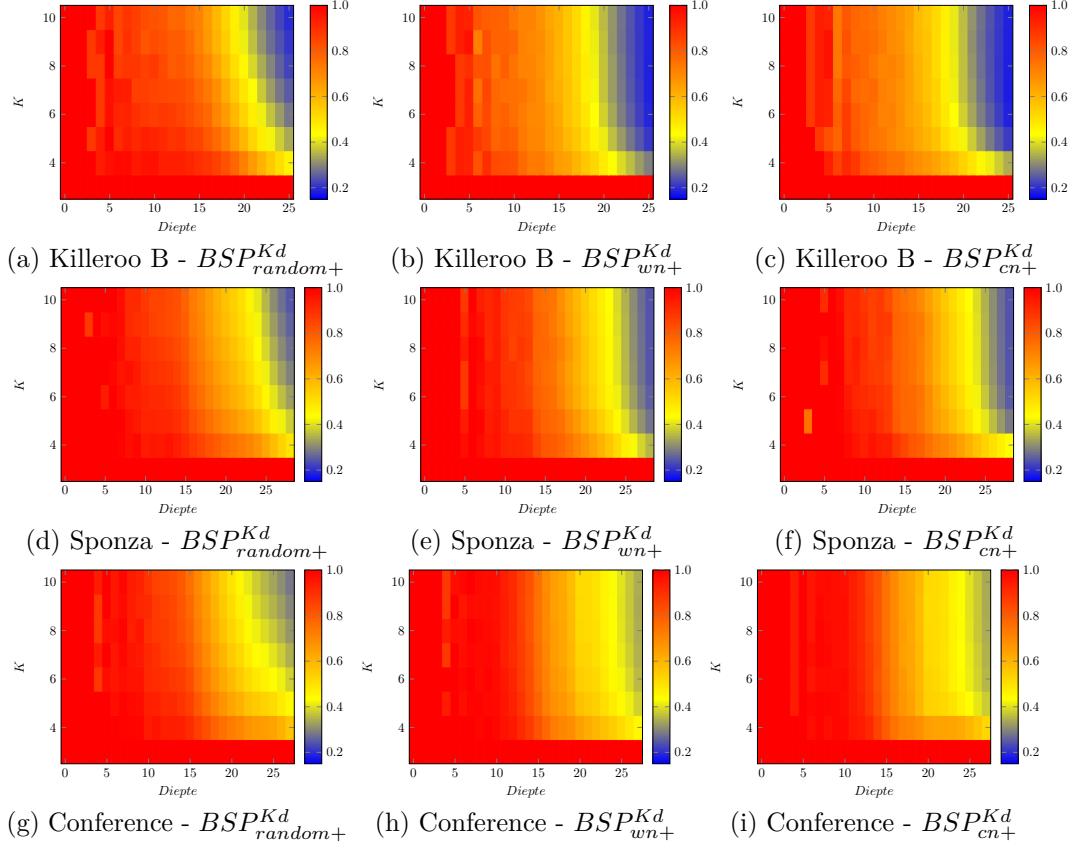
**Rendertijd** Tabel 5.4

**Aantal Intersecties** Tabel 5.5

**Aantal Doorkruisingen** Tabel 5.6

**$Kd$  knopen** Tabel 5.7

### 5.3. Vergelijking

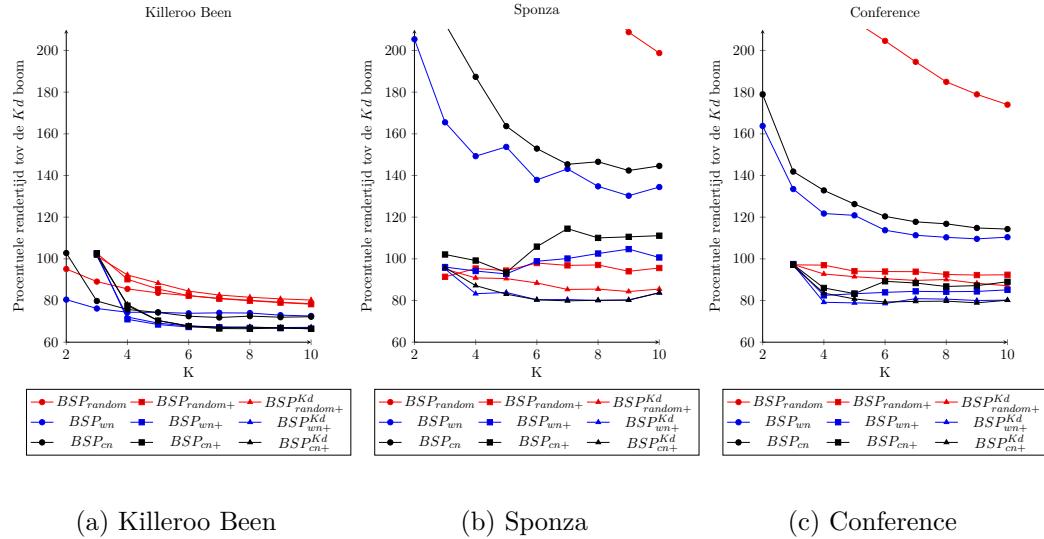


Figuur 5.6: Procentueel aantal  $Kd$  knopen per niveau - Deze grafieken tonen voor elke scène het procentueel aantal inwendige  $Kd$  knopen van de  $BSP_{SWEET}^{Kd}$  bomen ten opzichte van het totaal aantal inwendige knopen per niveau in functie van de diepte en  $k$ .

Boom	K	Killeroo Been		Sponza		Conference		Museum	
		R	B	R	B	R	B	R	B
$BSP_{wn+}^{Kd}$	6	67.7%	11 300%	80.4%	94 60%	78.6%	86 50%	63.4%	11 500%
$BSP_{cn+}^{Kd}$	6	67.8%	11 700%	80.3%	97 40%	79.3%	88 40%	65.9%	12 100%
$BSP_{IZE}^{Kd}$		94.1%	21 100%	165%	35 900%	105%	26 700%	88.2%	90 100%
$BSP_{IZE}^{Kd}$		95.0%	19 600%	109%	35 300%	92.5%	25 600%	84.7%	91 800%
$BSP_{random+}^{Kd}$	10	80.2%	17 800%	85.5%	16 100%	87.1%	15 400%	78.4%	21 800%
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$RBSP$	13	82.7%	25 100%	131%	23 700%	101%	22 300%	482%	25 000%
$RBSP^{Kd}$	13	83.5%	25 700%	94.2%	23 600%	92.9%	22 200%	78.1%	24 900%

Tabel 5.4: Vergelijking rendertijd en bouwtijd van  $BSP$  bomen - Deze tabel toont statistieken over de procentuele rendertijd en bouwtijd van  $BSP$  bomen ten opzichte van de rendertijd en bouwtijd van de  $Kd$  boom voor verschillende scenes.

## 5. RESULTATEN

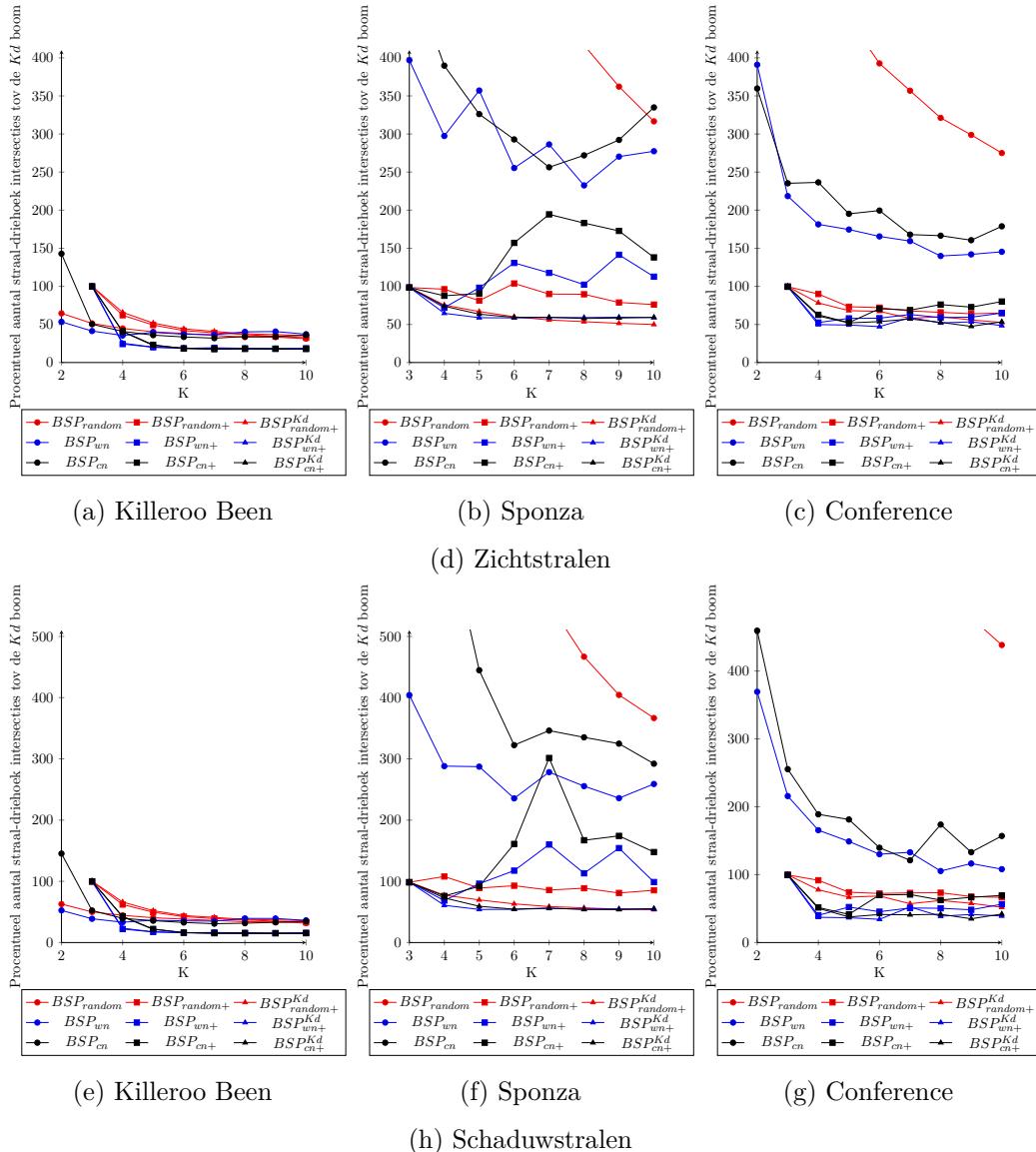


Figuur 5.7: Rendertijd in functie van  $k$  - Deze grafieken tonen voor elke scène de procentuele rendertijd van de  $BSP_{SWEEP}$  bomen ten opzichte van de rendertijd van de  $Kd$  boom in functie van  $k$ .

		Killeroo Been		Sponza		Conference		Museum	
Boom	K	ZI	SI	ZI	SI	ZI	SI	ZI	SI
$BSP_{wn+}^{Kd}$	6	19.0%	16.1%	58.6%	54.3%	47.1%	34.2%	27.0%	40.2%
$BSP_{cn+}^{Kd}$	6	18.7%	16.4%	59.3%	54.5%	53.7%	41.3%	28.5%	44.8%
$BSP_{IZE}^{Kd}$		73.1%	72.7%	390%	483%	128%	140%	73.1%	46.6%
$BSP_{IZE}^{Kd}$		76.2%	75.2%	157%	142%	83.4%	81.9%	57.5%	61.4%
$BSP_{random+}^{Kd}$	10	35.2%	35.8%	49.9%	53.8%	52.5%	52.0%	36.8%	52.3%
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$RBSP$	13	44.3%	45.2%	209%	239%	98.3%	111%	1380%	600%
$RBSP^{Kd}$	13	42.4%	43.0%	78.0%	75.8%	73.3%	78.4%	55.1%	33.1%

Tabel 5.5: Vergelijking straal-driehoek intersecties van *BSP* bomen - Deze tabel toont statistieken over het procentueel aantal straal-driehoek intersecties van *BSP* bomen ten opzichte van het aantal straal-driehoek intersecties van de *Kd* boom voor verschillende scenes. ZI staat voor zichtstaal intersecties en SI voor schaduwstraal intersecties.

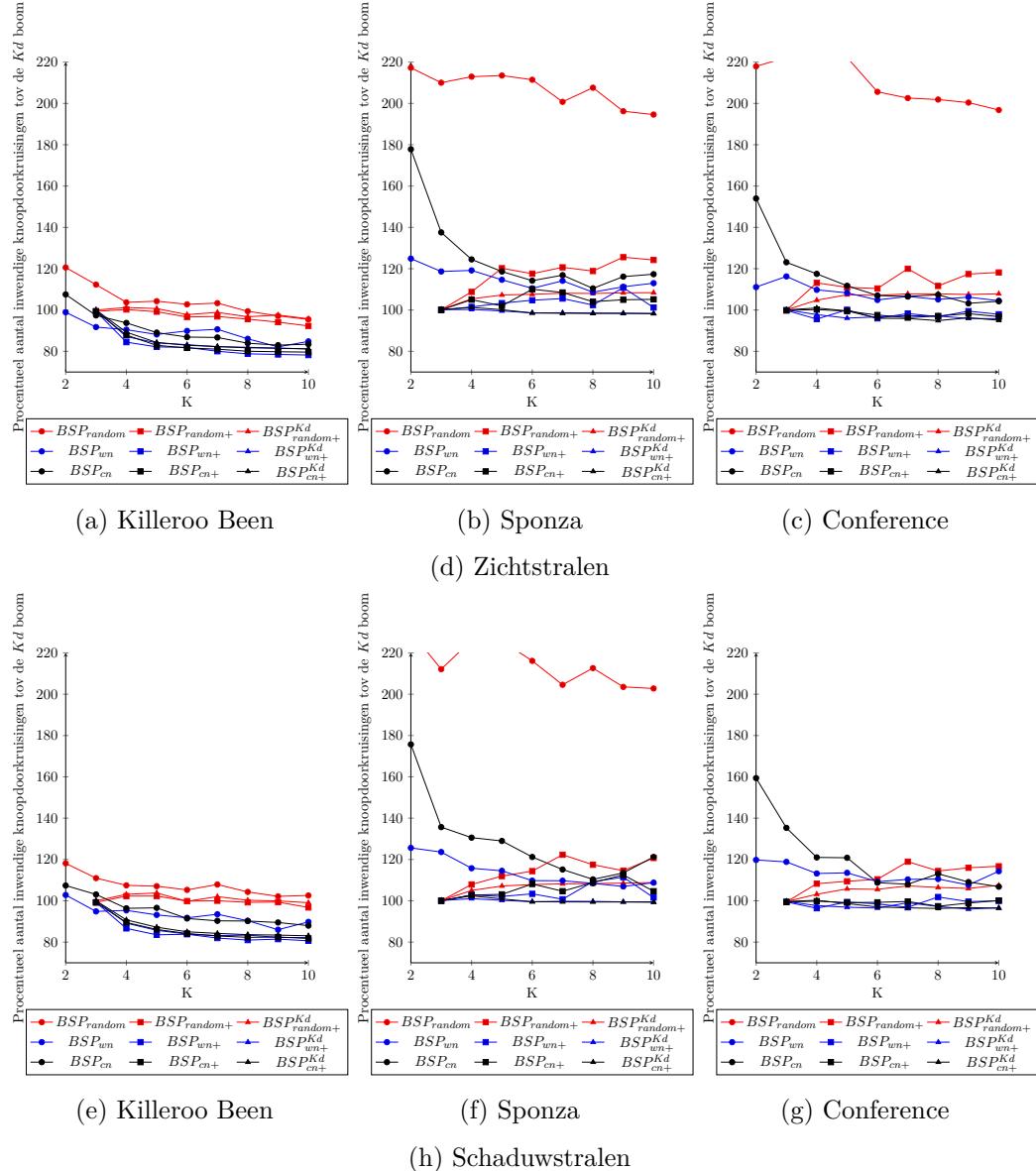
### 5.3. Vergelijking



Figuur 5.8: Straal-driehoek intersecties in functie van k - Deze grafieken tonen voor elke scène het procentueel aantal straal-driehoek intersecties van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal straal-driehoek intersecties van de Kd boom in functie van k. De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst en gebruiken een verschillende y-as.

## 5. RESULTATEN

---



Figuur 5.9: Inwendige knoopdoorkruisingen in functie van k - Deze grafieken tonen voor elke scène het procentueel aantal inwendige knoopdoorkruisingen van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal inwendige knoopdoorkruisingen van de  $Kd$  boom in functie van k. De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst en gebruiken een verschillende y-as.

### 5.3. Vergelijking

Boom	K	Killeroo Been		Sponza		Conference		Museum	
		ZD	SD	ZD	SD	ZD	SD	ZD	SD
$BSP_{wn+}^{Kd}$	6	83.2%	83.9%	98.6%	99.6%	96.7%	96.5%	94.1%	90.6%
$BSP_{cn+}^{Kd}$	6	83.0%	85.0%	98.6%	99.5%	96.0%	97.0%	95.0%	94.0%
$BSP_{IZE}$		97.3%	101%	130%	128%	117%	115%	109%	98.4%
$BSP_{IZE}^{Kd}$		100%	102%	117%	109%	107%	107%	107%	103%
$BSP_{random+}^{Kd}$	10	95.8%	99.1%	108%	109%	108%	108%	106%	105%
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$RBSP$	13	91.5%	96.1%	141%	138%	125%	127%	109%	97.1%
$RBSP^{Kd}$	13	95.3%	98.1%	117%	112%	110%	108%	105%	96.4%

Tabel 5.6: Vergelijking inwendige knoopdoorkruisingen van  $BSP$  bomen - Deze tabel toont statistieken over het procentueel aantal inwendige knoopdoorkruisingen van  $BSP$  bomen ten opzichte van het aantal inwendige knoopdoorkruisingen van de  $Kd$  boom voor verschillende scenes. ZD staat voor zichtstaal doorkruisingen en SD voor schaduwstraal doorkruisingen.

Boom	K	Killeroo Been				Sponza			
		$Kd$	ZD	$Kd$	SD	$Kd$	ZD	$Kd$	SD
$BSP_{wn+}^{Kd}$	6	29.5%	72.8%	72.3%	36.6%	88.8%	89.3%		
$BSP_{cn+}^{Kd}$	6	29.2%	72.8%	71.3%	36.5%	88.5%	89.0%		
$BSP_{IZE}$		49.4%	0%	0%	50.9%	0%	0%		
$BSP_{IZE}^{Kd}$		55.1%	88.3%	88.3%	55.6%	88.2%	90.9%		
$BSP_{random+}^{Kd}$	10	30.2%	74.6%	72.3%	34.5%	83.5%	84.9%		
$Kd$		100%	100%	100%	100%	100%	100%		
$RBSP$	13	22.0%	0%	0%	26.3%	0%	0%		
$RBSP^{Kd}$	13	29.8%	71.2%	70.9%	33.2%	77.9%	79.2%		

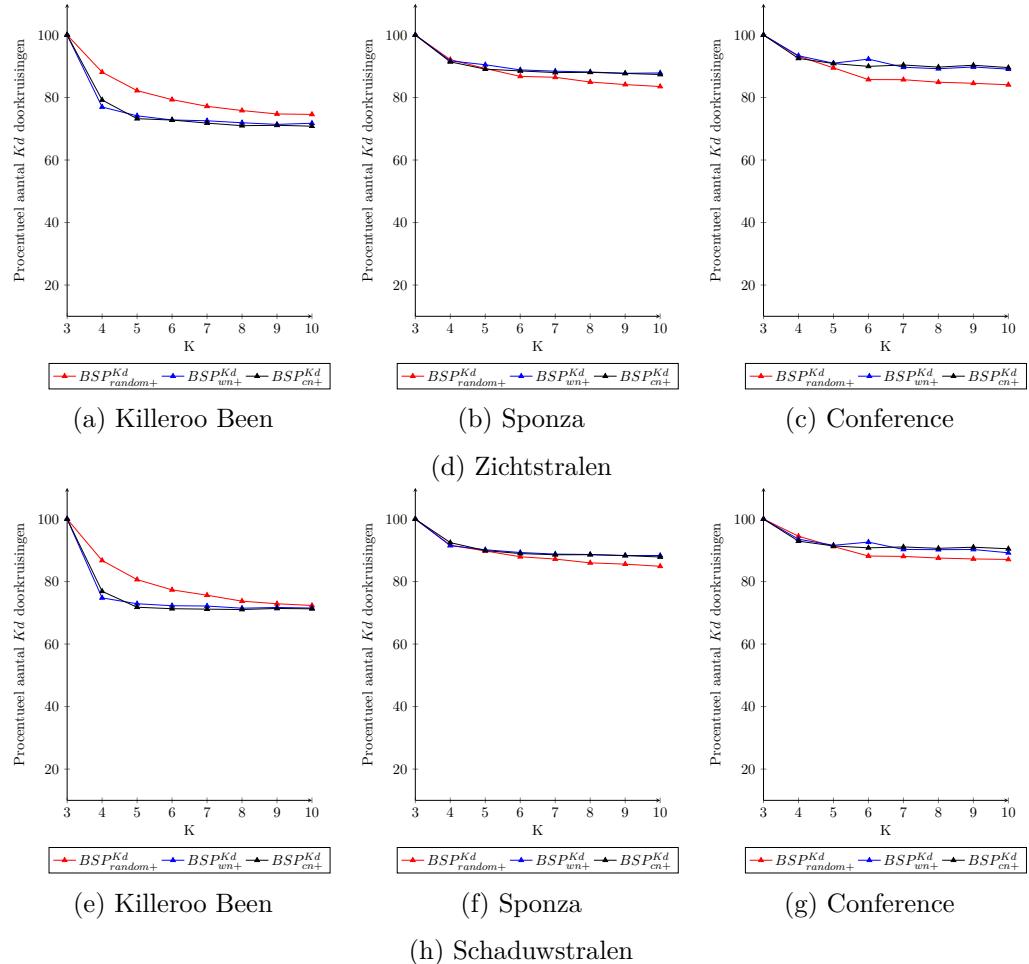
  

Boom	K	Conference				Museum			
		$Kd$	ZD	$Kd$	SD	$Kd$	ZD	$Kd$	SD
$BSP_{wn+}^{Kd}$	6	44.1%	92.3%	92.6%	32.0%	88.8%	87.3%		
$BSP_{cn+}^{Kd}$	6	43.2%	90.0%	90.8%	30.5%	87.1%	86.8%		
$BSP_{IZE}$		49.8%	0%	0%	38.5%	0%	0%		
$BSP_{IZE}^{Kd}$		58.4%	89.7%	91.0%	41.7%	87.1%	87.1%		
$BSP_{random+}^{Kd}$	10	35.3%	84.1%	87.1%	21.3%	83.1%	83.6%		
$Kd$		100%	100%	100%	100%	100%	100%		
$RBSP$	13	25.1%	0%	0%	18.3%	0%	0%		
$RBSP^{Kd}$	13	34.3%	80.4%	82.6%	22.2%	78.9%	78.4%		

Tabel 5.7: Vergelijking verhouding  $Kd$  knopen en  $Kd$  doorkruisingen knopen van  $BSP$  bomen - Deze tabel toont statistieken over het procentueel aantal inwendig  $Kd$  knopen, het procentueel aantal  $Kd$  zichtstraaldoorkruisngen (ZD) en het procentueel aantal  $Kd$  schaduwstraaldoorkruisngen (SD).

## 5. RESULTATEN

---



Figuur 5.10: Aantal  $Kd$  doorkruisingen in functie van  $k$  - Deze grafieken tonen voor elke scène het procentueel aantal  $Kd$  doorkruisingen van de  $BSP_{SWEEP}$  bomen in functie van  $k$ . De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst.

## **Hoofdstuk 6**

### **Besluit**



# Bibliografie

- [BCNJ08] B. C. Budge, D. Coming, D. Norpchen, and K. I. Joy. Accelerated building and ray tracing of restricted bsp trees. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 167–174, Aug 2008.
- [GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.
- [Hav00] Vlastimil Havran. *Heuristic ray shooting algorithms*. PhD thesis, Ph. d. thesis, Department of Computer Science and Engineering, Faculty of ..., 2000.
- [HB02] Vlastimil Havran and Jiří Bittner. On improving kd-trees for ray shooting. 2002.
- [IWP08] T. Ize, I. Wald, and S. G. Parker. Ray tracing with the bsp tree. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 159–166, Aug 2008.
- [KHM<sup>+</sup>98] James T Klosowski, Martin Held, Joseph SB Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.
- [KM07] R. P. Kammaje and B. Mora. A study of restricted bsp trees for ray tracing. In *IEEE/ EG Symposium on Interactive Ray Tracing 2007(RT)*, volume 00, pages 55–62, 09 2007.
- [MB90] J David MacDonald and Kellogg S Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.
- [PM19] WENZEL J. PHARR M., HUMPHREYS G. Pbrt v3. <http://www.pbrt.org/>, 2019.
- [Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned dop-trees. In *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No. 98CB36180)*, pages 90–97. IEEE, 1998.

## Fiche masterproef

*Student:* Jesse Hoobergs

*Titel:* Het gebruik van de normalen bij het bouwen van BSP acceleratiestructuren

*Engelse titel:* Using the normals to build BSP acceleration structures.

*UDC:* 621.3

*Korte inhoud:*

Hier komt een heel bondig abstract van hooguit 500 woorden. L<sup>A</sup>T<sub>E</sub>X commando's mogen hier gebruikt worden. Blanco lijnen (of het commando \par) zijn wel niet toegelaten!

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Mens-machine communicatie

*Promotor:* Prof. dr. ir. Ph. Dutré

*Assessoren:* Dr. B. Verreet

Prof. dr. R. Vandebri

*Begeleiders:* Ir. M. Moulin

Ir. P. Bartels