

# 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:  
computerwetenschappen, hoofdoptie  
Mens-machine communicatie

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

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

**Begeleider:**  
Ir. P. Bartels

© Copyright KU Leuven

Without written permission of the thesis supervisor and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email [info@cs.kuleuven.be](mailto:info@cs.kuleuven.be).

A written permission of the thesis supervisor is also required to use the methods, products, schematics and programmes described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

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.

# Preface

Het maken van een masterproef is een uitdaging die je niet alleen aangaat. Ik zou daarom dit voorwoord in de eerste plaats willen gebruiken om mijn promotor Philip Dutré te bedanken om dit onderwerp ter beschikking te stellen, voor de vrijheid waarmee ik dit onderwerp mocht aanpakken en voor de nuttige feedback en discussies tijdens de tussentijdse presentaties. Verder wil ik Matthias Moulin bedanken voor de uitstekende begeleiding in het eerste semester en Pieterjan Bartels om deze taak in het tweede semester met glans over te nemen. Ze hebben me met hun feedback geholpen bij het vormgeven van de ontwikkelde methoden en deze thesistekst. Hiernaast wil ik nog graag de voltallige jury bedanken voor het lezen van deze thesis.

Tot slot wil ik graag mijn familie en vrienden bedanken voor hun steun tijdens dit volledige academiejaar. In het bijzonder wil ik Jens Claes en Jorik Jooken bedanken om deze thesistekst na te lezen.

*Jesse Hoobergs*

# Contents

Preface	i
Abstract	iv
List of Figures	v
List of Tables	vi
Lijst van algoritmen	vii
Lijst van afkortingen en symbolen	viii
<b>1 Inleiding</b>	<b>1</b>
1.1 Fysisch gebaseerd renderen . . . . .	1
1.2 Doelstelling . . . . .	2
1.3 Structuur van de tekst . . . . .	3
<b>2 BSP bomen</b>	<b>5</b>
2.1 Concept . . . . .	5
2.2 $Kd$ Bomen . . . . .	7
2.3 $RBS\!P$ bomen . . . . .	9
2.4 Algemene $BSP$ Bomen in de praktijk . . . . .	11
2.5 Samenvatting . . . . .	12
<b>3 <math>BSP_{SWEEP}</math></b>	<b>14</b>
3.1 Probleemstelling . . . . .	14
3.2 Concept . . . . .	18
3.3 Gebaseerd op random richtingen . . . . .	18
3.4 Gebaseerd op normalen . . . . .	19
3.5 Samenvatting . . . . .	21
<b>4 Implementatie</b>	<b>23</b>
4.1 Hoog niveau beschrijving . . . . .	23
4.2 $Kd$ boom . . . . .	27
4.3 $RBS\!P$ boom . . . . .	28
4.4 $RBS\!P^{Kd}$ boom . . . . .	30
4.5 $BSP_{IZE}$ boom . . . . .	31
4.6 $BSP_{IZE}^{Kd}$ boom . . . . .	34
4.7 $BSP_{SWEEP}$ boom . . . . .	35
<b>5 Resultaten</b>	<b>41</b>

## CONTENTS

---

5.1	Praktische aspecten . . . . .	41
5.2	Afhankelijkheid van het aantal richtingen . . . . .	43
5.3	Vergelijking met bestaande bomen . . . . .	51
<b>6</b>	<b>Conclusie</b>	<b>61</b>
6.1	Conclusies $BSP_{SWEEP}$ . . . . .	61
6.2	Toekomstig onderzoek . . . . .	62
<b>7</b>	<b>English Commentary</b>	<b>63</b>
	<b>Bibliography</b>	<b>66</b>

# Abstract

*Ray tracing* is a computer graphics technique where rays are sent through a virtual 3D scene to generate a realistic 2D image. To reduce the amount of ray-triangle intersections, acceleration structures are used. These acceleration structures ensure that rays are only intersected with triangles that could possibly intersect with the ray. One of the most often used acceleration structures are *Binary Space Partitioning (BSP)* trees. The *BSP* tree recursively splits the bounding volume of the scene in child volumes by using splitting planes. In practice, only *Kd* trees are used, these are a specific kind of *BSP* trees which only split by using axis-aligned splitting planes.

In this thesis, we introduce a new kind of general *BSP* tree: the *BSP<sub>SWEET</sub>* tree. The *BSP<sub>SWEET</sub>* tree allows to determine a number of directions in each node depending on the triangles in that node. The best plane of all planes whose normal is one of these directions, is used to split the node. Three types of *BSP<sub>SWEET</sub>* trees are designed: one that generates random directions in each nodes (*BSP<sub>random</sub>*) and two types that generate directions based on the normals of the triangles in the node (*BSP<sub>wn</sub>* and *BSP<sub>cn</sub>*). The *BSP<sub>wn</sub>* tree chooses the normals of a few random triangles in the node and the *BSP<sub>cn</sub>* tree calculates a clustering of the normals of the triangles in the node and uses the cluster centra as directions. An optimised version of the *BSP<sub>SWEET</sub>* tree has also been worked out: the *BSP<sub>SWEET+</sub><sup>Kd</sup>* tree which favors axis-aligned splits and makes use of their computational advantages.

The three types of *BSP<sub>SWEET+</sub><sup>Kd</sup>* trees are clearly better than the existing *BSP* trees. The two types that make use of the local geometry (by using the normals) are significantly better than the type that uses the random directions. In all test scenes, they reduce the number of ray-triangle intersections by more than 40% and the render time by more than 20% with respect to the *Kd* tree. The build time is two orders of magnitude greater than that of the *Kd* tree, but smaller than that of any other general *BSP* tree.

# List of Figures

1.1	Visuele voorstelling van <i>ray tracing</i> . . . . .	1
1.2	Visuele voorstelling opdeling volume volgens object en ruimte . . . . .	3
2.1	Intersecteren van een <i>BSP</i> boom . . . . .	6
2.2	Splitsingskracht <i>BSP</i> boom . . . . .	7
2.3	Splitsingskracht <i>Kd</i> boom . . . . .	8
2.4	Splitsingskracht <i>RBSP</i> boom . . . . .	10
2.5	Splitsingsvlakken <i>Kd</i> , <i>RBSP</i> en <i>BSP<sub>I</sub>ZE</i> . . . . .	13
3.1	Invloed grootte bladknopen op het aantal straal-driehoekintersecties . .	16
3.2	<i>False color</i> afbeeldingen van de Killeroo en Killeroo Been scene . . . . .	17
3.3	Splitsingsvlakken <i>BSP<sub>random</sub></i> . . . . .	19
3.4	Splitsingsvlakken <i>BSP<sub>random+Kd</sub></i> . . . . .	19
3.5	Splitsingsvlakken <i>BSP<sub>wn</sub></i> en <i>BSP<sub>cn</sub></i> . . . . .	21
5.1	Testscenes . . . . .	42
5.2	Bouwtijd in functie van $k$ . . . . .	44
5.3	<i>SAH</i> kost in functie van $k$ . . . . .	44
5.4	Aantal inwendige knopen in functie van $k$ . . . . .	45
5.5	Procentueel aantal <i>Kd</i> knopen . . . . .	46
5.6	Procentueel aantal <i>Kd</i> knopen per niveau . . . . .	47
5.7	Rendertijd in functie van $k$ . . . . .	48
5.8	Straal-driehoekintersecties in functie van $k$ . . . . .	49
5.9	Inwendige knoopdoorkruisingen in functie van $k$ . . . . .	50
5.10	Aantal <i>Kd</i> doorkruisingen in functie van $k$ . . . . .	51
5.11	Cumulatieve procentuele bijdrage aan het totaal aantal straal-driehoekintersecties in functie van de bladknooppootte . . . . .	55
5.12	Procentueel aantal <i>Kd</i> knopen per niveau . . . . .	59

# List of Tables

2.1	Vergelijking van de bestaande soorten <i>BSP</i> bomen.	13
3.1	Vergelijking van de nieuwe soorten <i>BSP</i> bomen.	22
4.1	Voorstelling <i>Kd</i> knoop	27
4.2	Voorstelling <i>RBSB</i> knoop	29
4.3	Voorstelling <i>BSP</i> knoop	33
4.4	Voorstelling <i>BSP<sup>Kd</sup></i> knoop	35
5.1	Statistieken Testscenes	42
5.2	Gebruikte parameterwaarden	42
5.3	Specificaties computersysteem	42
5.4	Vergelijking rendertijd en bouwtijd van <i>BSP</i> bomen	53
5.5	Vergelijking straal-driehoekintersecties van <i>BSP</i> bomen	54
5.6	Vergelijking inwendige knoopdoorkruisingen van <i>BSP</i> bomen	54
5.7	<i>False color</i> afbeeldingen straal-driehoekintersecties met gelijke kleurwaarden	56
5.8	<i>False color</i> afbeeldingen straal-driehoekintersecties met verschillende kleurwaarden	57
5.9	Vergelijking proportie <i>Kd</i> knopen en proportie <i>Kd</i> knoopdoorkruisingen van <i>BSP</i> bomen	60

# Lijst van algoritmen

1	Bouwen van een BSP boom . . . . .	24
2	Intersecteren van een BSP boom . . . . .	26
3	Doorkruisen van een inwendige $Kd$ knoop. . . . .	28
4	Intersectie tussen een asgealigneerd vlak en een straal. . . . .	28
5	Beste splitsing voor een bouwknoop b bij een $Kd$ boom. . . . .	29
6	Doorkruisen van een inwendige $RBSP$ knoop. . . . .	30
7	Intersectie tussen een vlak en een straal. . . . .	30
8	Doorkruisen van een inwendige $RBSP^{Kd}$ knoop. . . . .	31
9	Beste splitsing voor een bouwknoop b bij een $RBSP^{Kd}$ boom. . . . .	32
10	Doorkruisen van een inwendige $BSP$ knoop. . . . .	33
11	Beste splitsing voor een bouwknoop b bij een $BSP_{IZE}$ boom. . . . .	34
12	Doorkruisen van een inwendige $BSP^{Kd}$ knoop. . . . .	35
13	Beste splitsing voor een bouwknoop b bij een $BSP_{IZE}^{Kd}$ boom. . . . .	36
14	Beste splitsing voor een bouwknoop b bij een $BSP_{SWEEP(+)}$ boom. . . . .	37
15	Generatie richtingen voor de $BSP_{random}$ boom. . . . .	37
16	Generatie richtingen voor de $BSP_{wn}$ boom. . . . .	38
17	Generatie richtingen voor de $BSP_{cn}$ boom. . . . .	39
18	Beste splitsing voor een bouwknoop b bij een $BSP_{SWEEP+}^{Kd}$ boom. . . . .	40

# Lijst van afkortingen en symbolen

## Afkortingen

<i>BSP</i>	Binary Space Partitioning
<i>BVH</i>	Bounding Volume Hierarchy
$k - DOP$	Discrete Oriented Polytope met $k$ richtingen
<i>RBSP</i>	Restricted Binary Space Partitioning
<i>SA</i>	Surface Area
<i>SAH</i>	Surface Area Heuristiek

---

## Symbolen

$\alpha$	Een parameter om $\mathcal{K}_{d,BSP}$ lineair te laten variëren met het aantal driehoeken.
$\beta$	Het procentueel aantal doorkruisingen door een inwendige knoop dat door slechts één van de twee kindknopen gaat.
$BSP_{wn}$	$BSP$ boom die per knoop willekeurige normalen als splitsingsrichtingen gebruikt.
$BSP_{wn+}$	$BSP_{wn}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de splitsingsrichtingen.
$BSP_{wn+}^{Kd}$	$BSP_{wn+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$BSP_{cn}$	$BSP$ boom die per knoop een clustering van de normalen berekent en de centra van deze clusters als splitsingsrichtingen gebruikt.
$BSP_{cn+}$	$BSP_{cn}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de splitsingsrichtingen.
$BSP_{cn+}^{Kd}$	$BSP_{cn+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$BSP_{random}$	$BSP$ boom die per knoop random richtingen als splitsingsrichtingen gebruikt.
$BSP_{random+}$	$BSP_{random}$ boom waarbij de drie richtingen volgens de hoofdassen altijd deel uitmaken van de random splitsingsrichtingen.
$BSP_{random+}^{Kd}$	$BSP_{random+}$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.
$k$	Het aantal splitsingsrichtingen dat in elke knoop bekeken wordt.
$\mathcal{K}_d$	De kost om een inwendige knoop te doorkruisen.
$\mathcal{K}_{d,BSP}$	De kost om een inwendige $BSP$ knoop te doorkruisen.
$\mathcal{K}_{d,Kd}$	De kost om een inwendige $Kd$ knoop te doorkruisen.
$\mathcal{K}_i$	De kost te intersecteren met een primitief.
$n$	Het aantal primitieven
$T_d$	De tijd nodig om één inwendige knoop te doorkruisen.
$T_{d,totaal}$	De totale tijd gespendeerd aan het doorkruisen van een boom.
$T_i$	De tijd nodig om te intersecteren met één primitief.
$T_{i,totaal}$	De totale tijd gespendeerd aan het intersecteren met primitieven.
$T_{render}$	De rendertijd
$RBSP^{Kd}$	$RBSP$ boom waarbij $Kd$ knopen efficiënter doorkruist worden dan $BSP$ knopen.

# Chapter 1

## Inleiding

### 1.1 Fysisch gebaseerd renderen

**Ray tracing** *Ray tracing* [App68] is een computergrafiek techniek voor fysisch gebaseerd renderen. Het genereert afbeeldingen (= renderen) door stralen te sturen door een scène [Suf07]. Een camera wordt op een bepaalde positie in de scène geplaatst en een afbeeldingsvlak, opgedeeld in pixels, wordt er voor 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 1.1 toont dit visueel.

Om realistische afbeeldingen te genereren, wordt belichting in het intersectiepunt 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

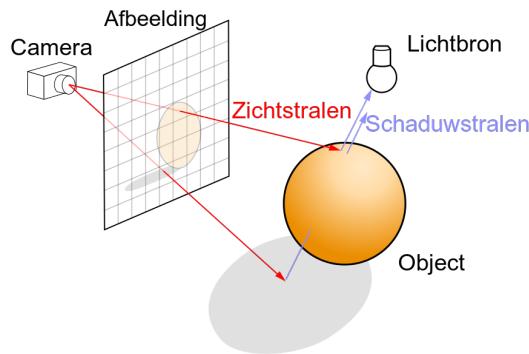


Figure 1.1: Visuele voorstelling van *ray tracing* - Door elke pixel worden één of meerdere zichtstralen gestuurd. Schaduwstralen worden gebruikt om de belichting in de scène realistisch te maken. Deze afbeelding is een aangepaste versie van een afbeelding op [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))

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 - licht dat via reflecties en refracties punten belicht - is *path tracing* [Kaj86] 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 de totale tijd nodig om een scene te renderen, te minimaliseren. Eén van de meest gebruikte representaties van 3D objecten is de *triangle mesh* waarbij het object wordt voorgesteld door een verzameling driehoeken. De intersectie berekenen tussen een straal en een object komt in dat geval neer op het testen op intersectie tussen de straal en alle driehoeken. Zelfs een eenvoudige scene kan meer dan honderdduizend driehoeken bevatten en tijdens het renderen kunnen meer dan tien miljoen stralen gestuurd worden. Het is onhaalbaar om voor elk van deze stralen, elke driehoek op intersectie te testen. Acceleratiestructuren verminderen de rendertijd door het aantal straal-driehoekintersecties te verminderen.

De simpelste acceleratiestructuur bestaat uit het omhullend volume van de scene. Testen op intersectie met de driehoeken in de scene 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. Figuur 1.2a toont dit visueel. Een voorbeeld van een acceleratiestructuur waarbij de opdeling volgens objecten gebeurt is de *Bounding Volume Hierarchy (BVH)* [GS87]. 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 mogelijks in meerdere) kindvolume(s). Figuur 1.2b toont dit visueel. De *Binary Space Partitioning (BSP)* boom deelt de ruimte van het volume steeds op in twee kindvolumes. Deze masterproef focust op *BSP* bomen.

## 1.2 Doelstelling

In de praktijk wordt altijd een specifiek soort *BSP* boom gebruikt: de *Kd* boom. Deze boom splitst knopen enkel op via asgealigeneerde splitsingsvlakken waardoor

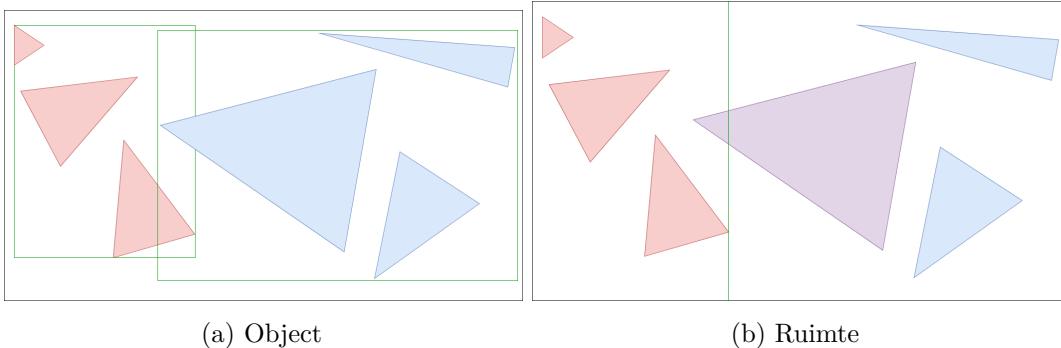


Figure 1.2: Visuele voorstelling opdeling volume volgens object en ruimte - (a) toont de opdeling volgens object, elk driehoek behoort tot slechts één kindvolume, maar de kindvolumes overlappen. (b) toont de opdeling volgens volume, de paarse driehoek behoort tot beide kindvolumes, maar de kindvolumes zelf zijn disjunct.

hij een aantal computationele voordelen heeft tijdens het bouwen en intersecteren van de boom. Het nadeel hiervan is dat de *Kd* boom zich niet goed kan aanpassen aan complexe niet-asgealigeneerde geometrie waardoor bepaalde regio's veel straal-driehoekintersecties nodig hebben.

Algemene *BSP* bomen kunnen hiervoor theoretisch gezien een oplossing bieden. Het ontwerpen van een algemene *BSP* boom die efficiënter is dan de *Kd* boom bij het *ray traceen*, is een uitdagend probleem. Eén van de moeilijkheden bij het bouwen van een algemene *BSP* boom is het kiezen van goede splitsingsvlakken uit het grote aantal mogelijke splitsingsvlakken. In tegenstelling tot de *Kd* boom, waarbij enkel asgealigeneerde vlakken mogelijk zijn, zijn alle vlakken mogelijk bij een algemene *BSP* boom. In deze masterproef wordt onderzocht of het mogelijk is om goede splitsingsvlakken te bepalen afhankelijk van de normalen van de driehoeken in de scène.

De oplossing die in deze tekst naar voren geschoven wordt, is een nieuw soort algemene *BSP* boom: de *BSP<sub>SWEET</sub>* boom. De *BSP<sub>SWEET</sub>* boom laat toe om in elke knoop een aantal richtingen te bepalen afhankelijk van de driehoeken in die knoop. Het beste vlak van alle vlakken met als normaal één van die richtingen, wordt gebruikt om de knoop te splitsen.

### 1.3 Structuur van de tekst

Hoofdstuk 2 bespreekt de bestaande soorten *BSP* bomen. Hoofdstuk 3 start met het beschrijven van het algemene concept van de nieuwe *BSP<sub>SWEET</sub>* boom. Daarna worden drie varianten van dit type boom besproken, één die random richtingen gebruikt en twee die gebruik maken van de normalen. Hoofdstuk 4 beschrijft de implementaties, inclusief gelijkenissen en verschillen, van de verschillende bestaande en nieuwe *BSP* bomen. Hoofdstuk 5 bespreekt de resultaten van de nieuwe *BSP<sub>SWEET</sub>*.

### 1.3. Structuur van de tekst

---

bomen in functie van het aantal richtingen en vergelijkt de kwaliteit van de nieuwe bomen met de bestaande *BSP* bomen.

# Chapter 2

## BSP bomen

Dit hoofdstuk start met een uitleg over het concept van *BSP* bomen. Daarna wijdt dit hoofdstuk uit over de verschillende varianten van de *BSP* boom: de *Kd* boom, de *RBSB* boom en de algemene *BSP* boom van Ize et al. [IWP08] (*BSPIZE*). Het hoofdstuk eindigt met een vergelijking van deze verschillende varianten.

### 2.1 Concept

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 bouwen van de boom** Om een *BSP* boom te bouwen moet eerst het omhullend volume van de scène bepaald worden. Het bepalen van de omhullende asgealigneerde balk is het eenvoudigste. Dit omhullend volume wordt dan in twee volumes (de kindvolumes) gesplitst volgens een vlak (het splitsingsvlak). De wortelknoop - die de hele scène voorstelt - krijgt dan twee kindknopen, één voor elk kindvolume. Deze kindknopen bevatten alle driehoeken die (deels) in hun volume liggen. Deze kindknopen worden dan recursief opgesplitst op dezelfde manier. Figuur 2.1 toont een scène waarbij de wortelknoop opgesplitst is in twee kindknopen  $k_1$  en  $k_2$  door het oranje splitsingsvlak. Deze twee kindknopen zijn elk ook opgesplitst volgens een groen splitsingsvlak.

**Het intersecteren van de boom** De eerste stap bij het bepalen van de intersectie tussen een straal en een boom, is het bepalen van de intersectie tussen de straal en het volume van de wortelknoop. Als er geen intersectie is, is er geen intersectie tussen de straal en de hele boom. Als er wel een intersectie is, wordt de wortelknoop doorkruist. Bij het doorkruisen van een inwendige knoop wordt de intersectie tussen de straal en het splitsingsvlak van de inwendige knoop bepaald. Aan de hand van dat intersectiepunt kan bepaald worden door welke kindknopen de straal gaat en in welke volgorde. Deze kindknopen worden dan op hun beurt in volgorde doorkruist. Het doorkruisen van een bladknoop komt neer op het berekenen van de intersectie

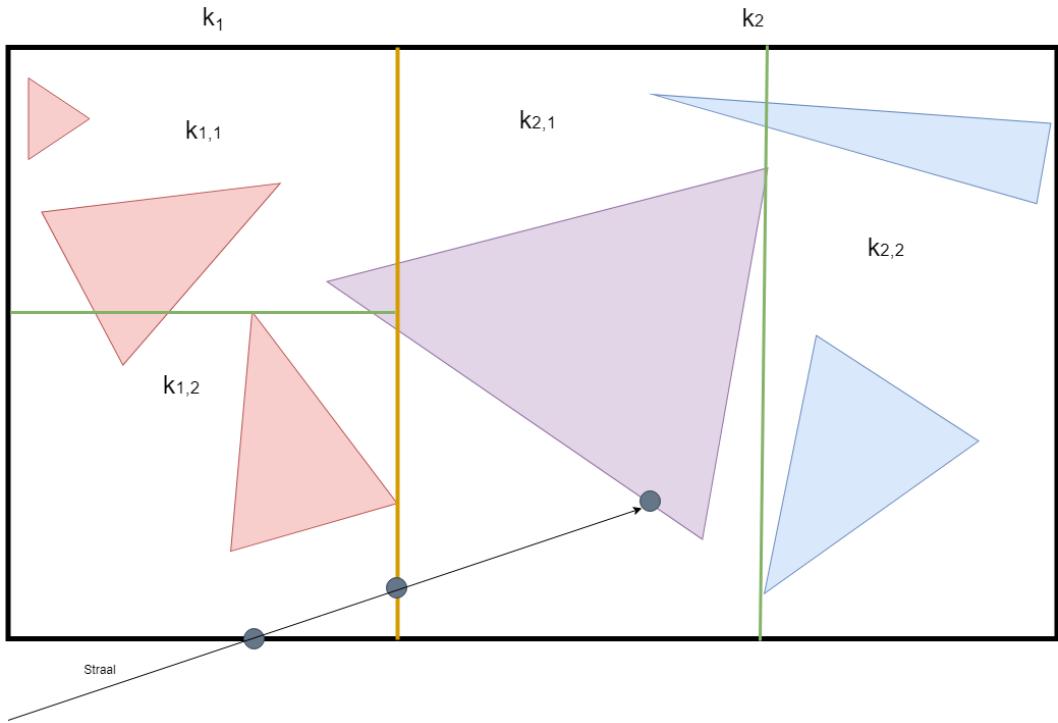


Figure 2.1: Intersecteren van een *BSP* boom - Een 2D voorbeeld dat toont hoe de intersectie tussen een straal en een *BSP* boom wordt bepaald. De straal bepaalt eerst de intersectie met het volume van de wortelknoop. Aangezien er een intersectie is, doorkruist de straal de wortelknoop door te intersecteren met het oranje splitsingsvlak. De  $k_1$  knoop wordt doorkruist waardoor de intersectie met het linkse groene splitsingsvlak berekend wordt. Dan worden alle driehoeken in bladknoop  $k_{1,2}$  getest op intersectie, gevolgd door alle driehoeken in bladknoop  $k_{1,1}$ . Omdat er nog geen intersectie gevonden is, wordt dan de  $k_2$  knoop doorkruist waardoor intersectie met het rechtse groene splitsingsvlak berekend wordt. Bladknoop  $k_{2,1}$  wordt dan doorkruist waardoor het intersectiepunt gevonden wordt.

met alle driehoeken in die bladknoop. Figuur 2.1 toont een straal die een *BSP* boom intersecteert.

**Splitsingsvlakken** Het splitsingsvlak dat een knoop opdeelt in twee delen, kan een grote impact hebben op het aantal doorkruisstappen en het aantal straal-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.

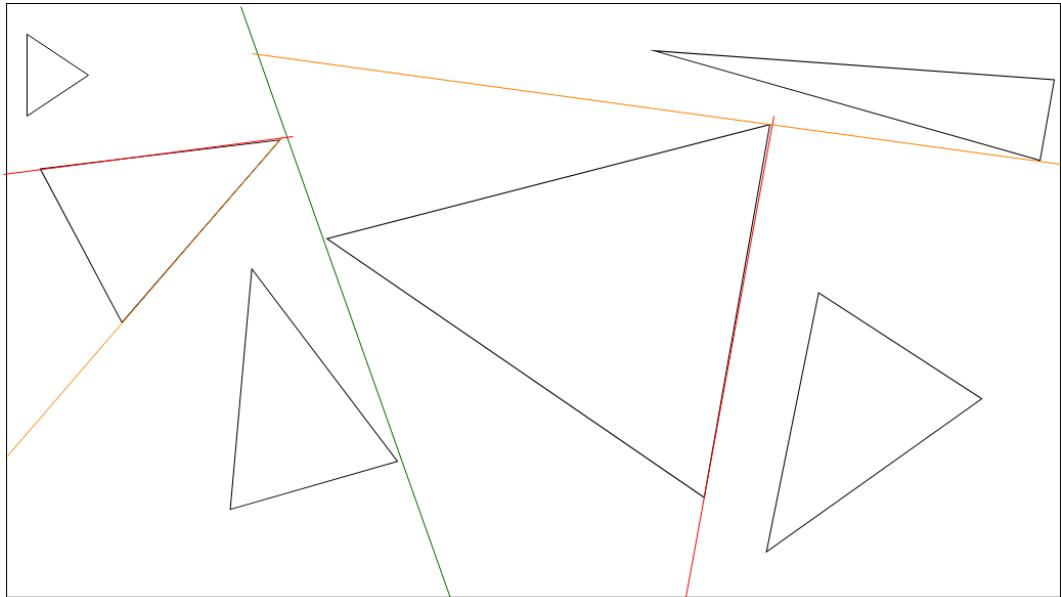


Figure 2.2: Splitsingskracht *BSP* boom - Een 2D voorbeeld dat toont dat alle (niet-intersecerende) driehoeken gesplitst kunnen worden door een *BSP* boom.

Het feit dat de *BSP* boom volgens willekeurig georiënteerde vlakken splitst, is zowel een voor- als nadeel [IWP08]. Het zorgt ervoor dat de *BSP* boom zich heel goed kan aanpassen aan de scene en alle niet-intersecerende driehoeken in principe kan scheiden (zie figuur 2.2). Het zorgt er echter ook voor dat het moeilijk is om deze goede splitsingsvlakken te vinden in de grote verzameling van mogelijke splitsingsvlakken. In de praktijk wordt vaak een specifieke soort *BSP* boom gebruikt: de *Kd* boom.

## 2.2 *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 [IWP08]. 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 [IWP08]. Ze kunnen bijvoorbeeld niet alle niet-intersecerende driehoeken scheiden. Figuur 2.3 toont een situatie waarin de  $Kd$  boom niet alle driehoeken kan scheiden.

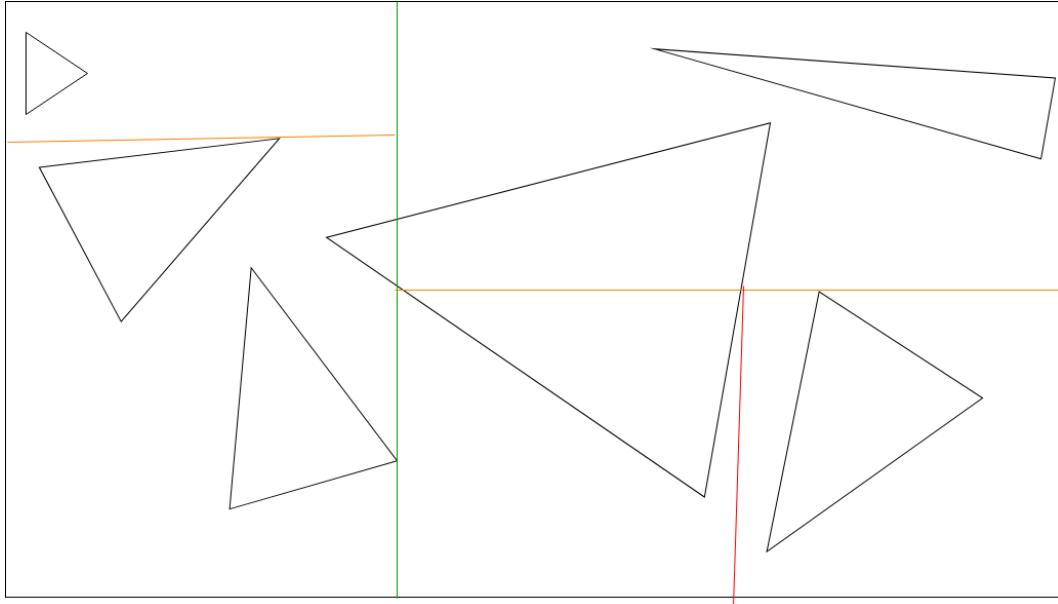


Figure 2.3: Splitsingskracht *Kd* boom - Een 2D voorbeeld dat toont dat niet alle (niet-intersecterende) driehoeken gesplitst kunnen worden door de asgealigneerde vlakken van de *Kd* boom. De twee driehoeken linksonder (en rechtsboven) kunnen niet gesplitst worden omdat zowel hun projecties op de x-as,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

Ondanks dit nadeel, worden in de praktijk *Kd* bomen verkozen boven algemene *BSP* bomen. Ize et al. [IWP08] geven drie ruimverspreide foute aannames over algemene *BSP* bomen die ervoor zorgen dat *Kd* bomen hoger ingeschatt 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 aansname is dat door de grotere flexibiliteit (= grotere verzameling van mogelijke splitsingsvlakken) het veel moeilijker is om een goede *BSP* boom te bouwen dan om een goede *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 [IWP08]. 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  $\mathcal{K}_p$  om een knoop  $p$  te splitsen in kindknopen  $l$  en  $r$  is dan gelijk aan formule 2.1 met kost  $\mathcal{K}$ , oppervlakte  $SA()$  en aantal driehoeken  $n$ . De subscripts  $i$  en  $d$  staan voor

respectievelijk intersectie en doorkruising.

$$\mathcal{K}_p = \frac{SA(l)}{SA(p)} * n_l * \mathcal{K}_i + \frac{SA(r)}{SA(p)} * n_r * \mathcal{K}_i + \mathcal{K}_d \quad (2.1)$$

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 onhaalbaar. 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.

## 2.3 *RBS<sub>P</sub>* bomen

De Restricted Binary Space Partitioning (*RBS<sub>P</sub>*) 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 *RBS<sub>P</sub>* knoop is hierdoor geen balk maar een Discreet Geöriënteerde Polytoop met  $k$  richtingen: een *k-DOP*. Zowel Klosowski et al. [KHM<sup>+</sup>98] als Zachmann [Zac98] hebben hiërarchiën van *k-DOPs* gebruikt binnen het domein van botsherkenning (*collision detection*). De *RBS<sub>P</sub>* 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 *RBS<sub>P</sub>* boom meer op de algemene *BSP* boom dan de *Kd* boom. Net als de *Kd* boom kan de *RBS<sub>P</sub>* boom niet alle niet-intersecerende driehoeken scheiden. Figuur 2.4 toont een situatie waarin de *RBS<sub>P</sub>* boom niet alle driehoeken kan scheiden.

Een belangrijke ontwerpbeslissing bij *RBS<sub>P</sub>* 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

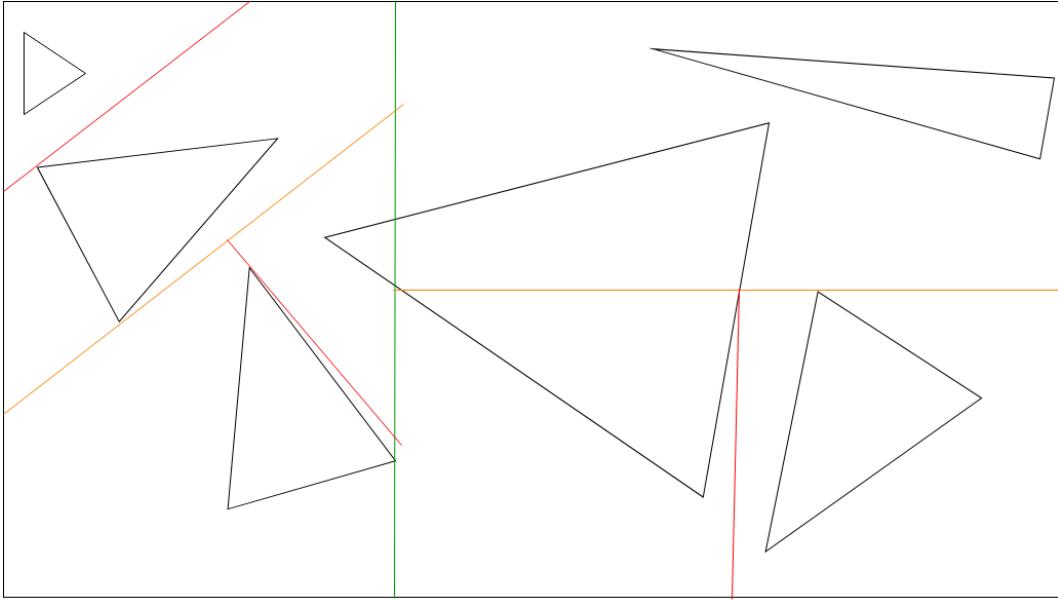


Figure 2.4: Splitsingskracht *RBSP* boom - Een 2D voorbeeld dat toont dat niet alle (niet-intersecerende) driehoeken gesplitst kunnen worden door de vlakken van de *RBSP* boom. De *RBSP* boom kan de driehoeken wel beter splitsen dan de *Kd* boom in figuur 2.3. De twee driehoeken rechtsboven kunnen niet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

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 standaardrichtingen van Klosowski et al. [KHM<sup>+</sup>98] noemen. Deze richtingen 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 de sweepbeweging.

De huidige *RBSP* implementaties zijn superieur ten opzichte van de *Kd* boom in termen van het aantal straal-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.

## 2.4 Algemene *BSP* Bomen in de praktijk

Ize et al. [IWP08] ontwikkelden de eerste (en enige) 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 [IWP08]. Het omhullend volume van de knopen bij een *BSP* boom is een convex veelvlak. Voor de eenvoud wordt de asgealigneerde omhullende balk gebruikt als omhullend volume voor de wortelknoop. 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. Het bepalen van deze aantallen gebeurt in logaritmische tijd in tegenstelling tot de constante tijd bij de vorige bomen. 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 doorkruistterm waardoor *BSP* splitsingen, die de driehoeken beter splitsen, gekozen worden [IWP08]. 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 met 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 blad-knoop 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-driehoekintersecties. 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 algemene *BSP* bomen beter kunnen omgaan met complexe geometrie.

## 2.5 Samenvatting

**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 meerdere driehoeken in de knoop, vlakken volgens dezelfde richting bekeken worden, kan sweeping toegepast worden om efficiënt de *SAH* kosten te berekenen. Als sweeping niet kan, zoals bij de geometrie-afhankelijke vlakken van 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* knoopdoorkruising gebruikt worden. Voor de *RBSP* boom is dit nog nooit gedaan.

**Aantal verschillende splitsingsvlakken** Eén van de belangrijkste eigenschappen van een *BSP* boom is zijn vermogen om zich aan te passen aan complexe geometrie. Het totaal aantal verschillende splitsingsvlakken dat bekeken wordt tij-

	<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	<i>Kd</i>	$RBSP^{Kd}$	$BSP_{IZE}^{Kd}$
# splitsingsvlakken per niveau	$6n$	$2kn$	$10n$
totaal # ≠ splitsingsvlakken	$6n$	$2kn$	$10n$

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

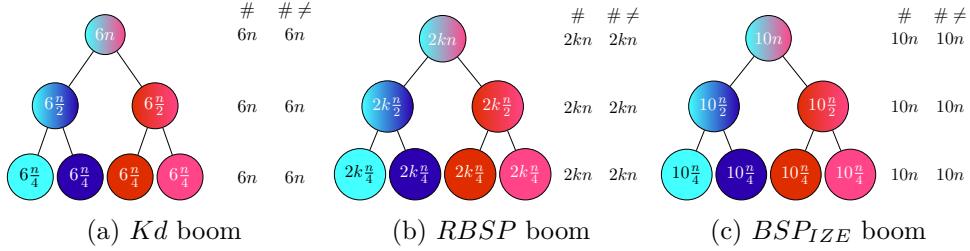


Figure 2.5: Splitsingsvlakken *Kd*, *RBSP* en *BSPIZE* - Per niveau het aantal (#) splitsingsvlakken en het totaal aantal verschillende (# ≠) splitsingsvlakken gebruikt in bovenliggende niveaus.

dens 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 *BSPIZE* boom gebruikt  $10n$  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 op geen enkel niveau 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.

# Chapter 3

## BSP<sub>SWEET</sub>

In dit hoofdstuk wordt een nieuw 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 scène te renderen, te minimaliseren. Deze sectie toont aan dat de totale rendertijd afhankelijk is van het aantal driehoeken in de geïnterseerde bladknopen. In de eerste subsectie wordt wiskundig aangetoond dat het (onder bepaalde aannames) altijd voordelig is om bladknopen met meer dan één driehoek op te splitsen in kleinere bladknopen. De tweede subsectie toont een voorbeeld waarbij de *Kd* en *BSP<sub>SIZE</sub>* boom vergeleken worden op basis van de grootte van de bladknopen.

#### 3.1.1 Wiskundige basis

De rendertijd  $T_{render}$  bevat 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 geïnterseerde 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-driehoekintersectie,  $T_d$  de tijd nodig om één knoop te doorkruisen en  $B$  het aantal geïnterseerde bladknopen.

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

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 inwendige 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  $D_{b_{j1}} + D_{b_{j2}} = (2 * (1 - \beta) + \beta)D_{b_j}$  waarbij  $\beta$  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.

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

In het ideale geval ( $\beta = 1$ ) leidt voorwaarde 3.6 tot  $T_d \leq \frac{n_{b_j}}{2} T_i$ . Het opsplitsen van een knoop met twee driehoeken, 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 ( $\beta = 0$ ) 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  $\beta$  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  $\beta$  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.

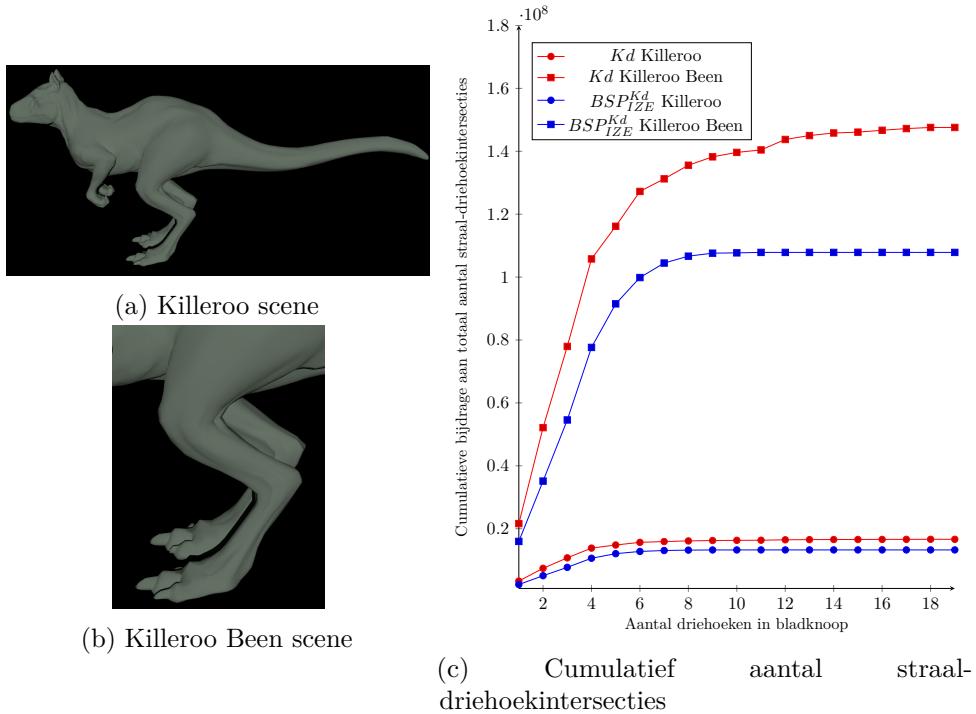


Figure 3.1: Invloed grootte bladknopen op het aantal straal-driehoekintersecties - (a) toont de Killeroo scene, (b) toont de scene waarbij ingezoomd is op de benen van de Killeroo, (c) toont het totaal aantal intersecties als een cumulatieve som over het aantal driehoeken in de bladknoop. Dit betekent dat de waarde bij  $x$  overeenkomt met de som van het aantal intersecties in bladknopen met  $x$  of minder driehoeken.

### 3.1.2 Voorbeeld

Figuur 3.1a toont de Killeroo scene die als voorbeeldscene dient bij de pbrt [PM19] *ray tracer*. Voor deze scene zijn de  $Kd$  boom en de  $BSP_{IZE}^{Kd}$  boom aan elkaar gewaagd. De  $BSP_{IZE}^{Kd}$  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 scene, heeft de  $Kd$  boom slechts een beperkt aantal extra intersecties nodig. Bij de ingezoomde scene, komt een groot deel van de intersecties van de  $Kd$  boom door intersecties met bladknopen met een groot aantal driehoeken. Dit toont aan dat algemene  $BSP$  bomen nuttig kunnen zijn omdat ze in principe alle niet-intersecerende driehoeken van elkaar kunnen scheiden en er dus minder regio's zijn met veel grote bladknopen. Het aantal knoopdoorkruisingen bij de  $BSP_{IZE}^{Kd}$  boom is zelfs lager dan bij de  $Kd$  boom voor beide scenes. Dit komt omdat de algemene  $BSP$  bomen 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 scene voor zowel de  $Kd$  boom als de  $BSP_{IZE}^{Kd}$

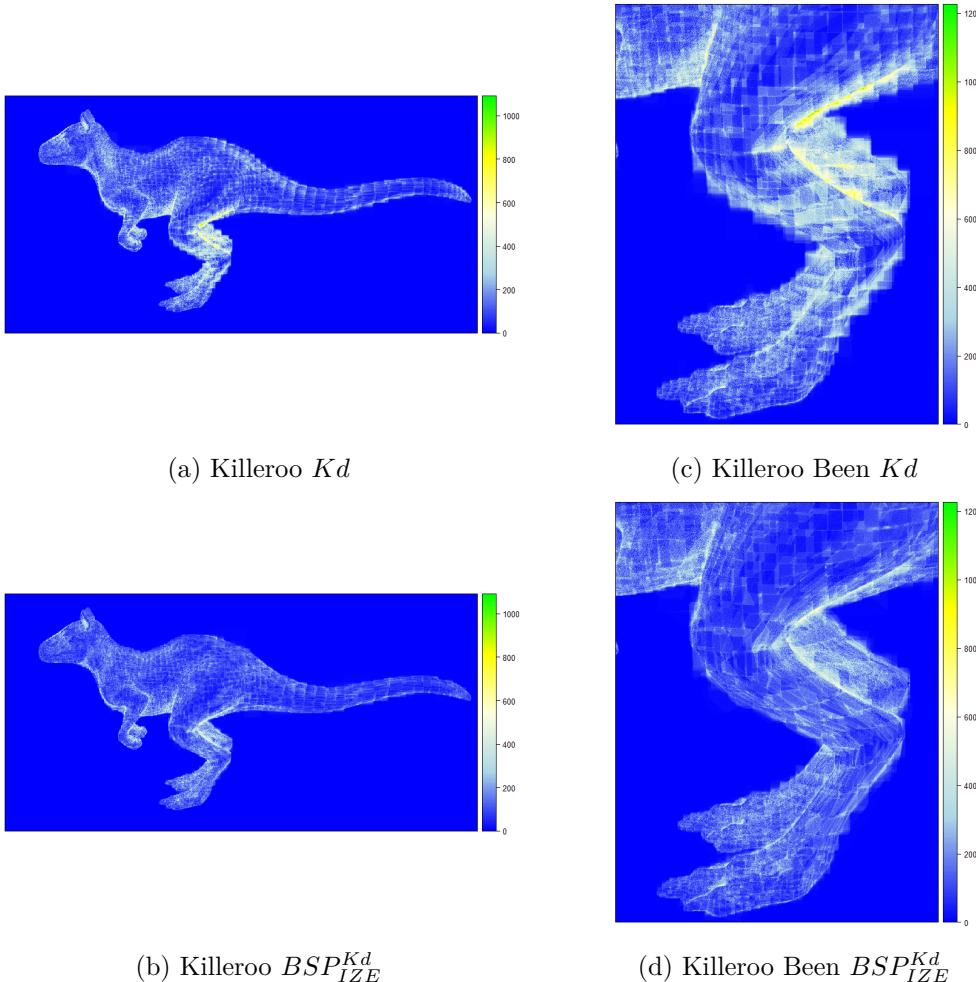


Figure 3.2: *False color* afbeeldingen van de Killeroo en Killeroo Been scene - Deze afbeeldingen tonen dat de  $Kd$  boom slecht presteert bij sommige delen van een scène, in dit geval de benen van de Killeroo. Bij de  $BSP_{IZE}^{Kd}$  boom is dit fenomeen veel minder uitgesproken en het valt ook op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

boom. Voor de Killeroo scène zijn de *false color* afbeeldingen van beide bomen zeer 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. Op die manier kunnen ze meer bladknopen opsplitsen in kleinere bladknopen en nog

nauwer aansluiten aan de scene.

## 3.2 Concept

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 bekeken worden 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 te proberen om driehoeken te splitsen via veel verschillende (mogelijks slechte) vlakken, dan om ze steeds met dezelfde vlakken te proberen te 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 scènes vaak beter dan willekeurige richtingen omdat ze loodrecht op elkaar staan waardoor ze de hemisfeer goed bedekken en omdat scènes 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  $Kd$  knoopdoorkruising 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  $BSP_{random+}$  boom perfect gebalanceerd is, worden in elk niveau  $2kn$  splitsingsvlakken bekeken.

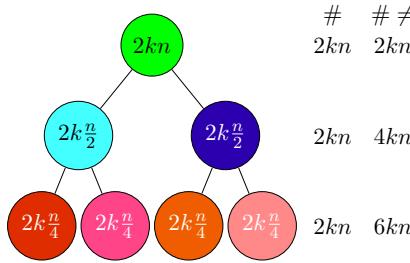


Figure 3.3: Splitsingsvlakken  $BSP_{random}$  - Per niveau het aantal ( $\#$ ) splitsingsvlakken en het totaal aantal verschillende ( $\# \neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{random}$  boom.

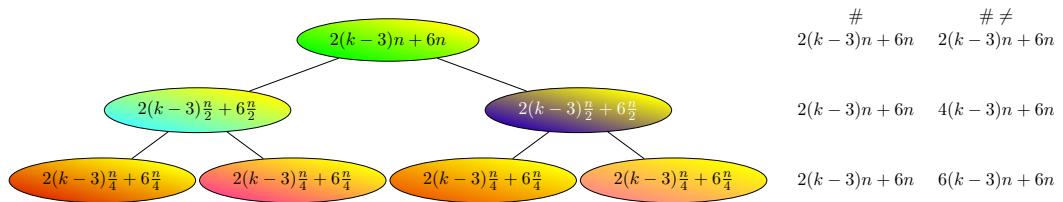


Figure 3.4: Splitsingsvlakken  $BSP_{random+}^{(Kd)}$  - Per niveau het aantal ( $\#$ ) splitsingsvlakken en het totaal aantal verschillende ( $\# \neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{random+}^{(Kd)}$  boom.

Van deze  $2kn$  zijn er  $6n$  die hergebruikt worden, de vlakken volgens de  $Kd$  richtingen. Dit zorgt voor  $2(k-3)n$  verschillende splitsingsvlakken per niveau, zodat in totaal  $2(k-3)n \log(n) + 6n$  verschillende splitsingsvlakken bekijken 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 autopartitievvlakken 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 splitsingsrichtingen 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 splits-

ingsvlakken gebruikt per knoop in de bovenste niveaus van de boom. Deze figuur is identiek aan de figuur voor de  $BSP_{random}$  boom.

Voor de onderste  $\log(k)$  niveaus waarop gesplitst wordt - dit zijn de onderste  $\log(k) + 1$  niveaus behalve het onderste niveau - is er echter een verschil. Dit verschil volgt uit het feit dat er in die niveaus  $k$  of minder driehoeken in elke knoop zitten. In deze 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 van deze  $\log(k)$  onderste niveaus bevat  $n_m = \frac{n}{2^{\log(n)-m}}$  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. Dit kan worden ingezien als volgt: de knopen op het niveau erboven bevatten  $2k$  driehoeken en in die knoop worden  $k$  normalen gekozen. De twee kindknopen van elk van deze knopen gebruiken elk  $k$  verschillende richtingen en gebruiken dus de volledige  $2k$  richtingen waaruit de ouderknoop kon kiezen. Dit betekent dat ze  $k$  nieuwe richtingen gebruiken, elk voor de helft van het aantal driehoeken:  $2 * k \frac{n}{2} = kn$ . In totaal worden er  $2(\log(\frac{n}{k}) + \frac{1}{2})kn$  verschillende splitsingsvlakken 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  $k$  splitsingsrichtingen 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  $BSP_{random+}^{(Kd)}$ .

### 3.4.2 Geclusterde normalen

Bovenstaande  $BSP_{SWEEP}$  bomen bevatten een grote niet-deterministische factor. De  $BSP_{cn}$  boom gebruikt het K-means clustering algoritme om deterministischere richtingen te bepalen. Kim et al. [KCK02] gebruikten een K-means clustering van de normalen om de bestanden van 3D modellen te comprimeren. Bij de  $BSP_{cn}$  boom worden de normalen van de driehoeken in een knoop geclusterd in  $k$  clusters en de

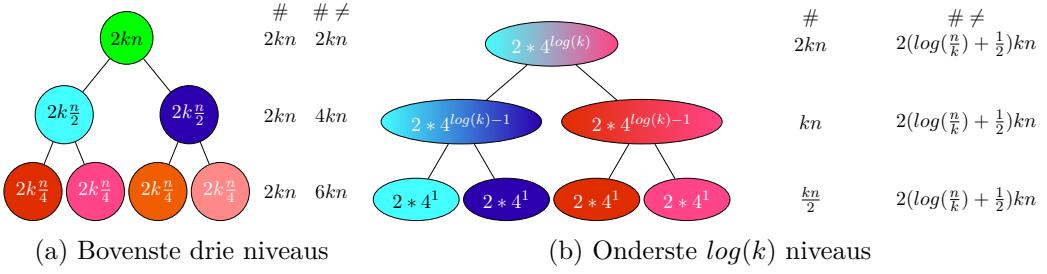


Figure 3.5: Splitsingsvlakken  $BSP_{wn}$  en  $BSP_{cn}$  - Per niveau het aantal (#) splitsingsvlakken en het totaal aantal verschillende (# ≠) splitsingsvlakken gebruikt in bovenliggende niveaus bij de  $BSP_{wn}$  en  $BSP_{cn}$  bomen.

centra van deze clusters worden gebruikt als richtingen voor de  $BSP_{SWEEP}$  boom. Als er  $k$  of minder driehoeken in de knoop zitten, worden, net als bij de  $BSP_{wn}$  boom, alle normalen als splitsingsrichting genomen. 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  $BSP_{cn+}$  en  $BSP_{cn+}^{Kd}$  bomen.

## 3.5 Samenvatting

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

	$BSP_{random}$	$BSP_{wn}$	$BSP_{cn}$
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	$2kn$	$2kn^*$	$2kn^*$
totaal # ≠ splitsingsvlakken	$2kn\log(n)$	$2(\log(\frac{n}{k}) + \frac{1}{2})kn$	$2(\log(\frac{n}{k}) + \frac{1}{2})kn$

Table 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.

# Chapter 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 *Github* [Hoo19].

Het hoofdstuk start met een hoog niveau beschrijving van het algoritme om een *BSP* boom te bouwen en het algoritme om deze bomen te intersecteren. Daarna wordt voor elke type apart 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.
- $maxIterations$ : Het maximale aantal iteraties bij de K-means clustering van de *BSP<sub>cn</sub>* boom.

- $\alpha$ : Een parameter om  $\mathcal{K}_{d,BSP}$  lineair te laten variëren met het aantal driehoeken. Deze parameter wordt gebruikt bij het bouwen van bomen met de snelle  $Kd$  knoopdoorkruising.

---

**Algoritme 1** Bouwen van een BSP boom

---

```

stack ← ∅
Voeg een bouwknoop met alle primitieven toe aan de stack
while stack ≠ ∅ do
    b ← POP(stack)
    if  $b_n \leq n_{max}$  or  $b_d = d_{max}$  then
        MAAK_BLAD_KNOOP(b)
        continue
    end if
    besteSplit ← BEPAAL_BESTE_SPLITSING(b)
    nietSplitKost ←  $b_n * \mathcal{K}_i$ 
    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

```

---

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:

1. BEPAAL\_BESTE\_SPLITSING(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 bouwknoop.

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 splitsing op een lager niveau, wordt een knoop toch gesplitst als splitsen nadelig is volgens de *SAH* heuristiek. Zulke *slechte aanpassingen* worden niet gedaan als de kost van de beste splitsing 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  $\vec{s} = \vec{o} + t\vec{d}$  met  $\vec{o}$  het startpunt en  $\vec{d}$  de richtingsvector. 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, in plaats van te zoeken naar het dichtstbijzijnde, is triviaal en wordt niet verder besproken. Deze aanpassing is nodig voor de intersectie van schaduwstralen.

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 booleaanse waarde 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 booleaanse 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

---

**Algoritme 2** Intersecceren van een BSP boom

---

```
function INTERSECTEER(Boom b, Straal s)
    geraaktomhullendVolume, tMin, tMax ← INTERSECTEER( $b_{omhullendVolume}$ , s)
    if not geraaktomhullendVolume then
        return false
    end if
    geraakt ← false
    k ← bwortelKnoop
    stack ← ∅
    while k ≠ None do
        if  $s_{maxT} < tMin$  then
            break
        end if
        if  $k_{isInwendig}$  then
            tVlak, linksEerst ← DOORKRUIS__INWENDIGE__KNOOP(k,s)
            if linksEerst then
                k1 ← klinkerKind
                k2 ← krechterKind
            else
                k1 ← krechterKind
                k2 ← klinkerKind
            end if
            if  $tVlak > tMax$  or  $tVlak \leq 0$  then
                k ← k1
            else if  $tVlak < tMin$  then
                k ← k2
            else
                ADD(stack, {k2, tMin : tVlak, tMax : tMax})
                k ← k1
                tMax ← tVlak
            end if
        end if
    else
        if INTERSECTEER__BLADKNOOP(k, s) then
            geraakt ← true
        end if
        if stack ≠ ∅ then
            k, tMin, tMax ← POP(stack)
        else
            k ← None
        end if
    end if
end while
return geraakt
end function
```

---

Inwendig	bits	Blad	bits
<i>tSplit</i>	32	<i>primitiefOffset</i>	32
<i>flags</i>	2	<i>flags</i>	2
<i>tweedeKindIndex</i>	30	<i>n</i>	30
	64		64

Table 4.1: Voorstelling *Kd* knoop - Deze tabel toont de nodige variabelen voor zowel inwendige als blad *Kd* knopen.

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 volledige *BSP* boom (= omhullend volume wortelknoop) met de straal. Bij alle besproken bomen wordt dit voorgesteld door een asgealigneerde balk. Intersectie berekenen met een asgealigneerde balk is triviaal.

## 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 toont het algoritme om inwendige *Kd* knopen te doorkruisen. Het gebruikt de functie KD\_VLAK\_INTERSECTIE (algoritme 4) om te intersecteren

---

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

---

```
function DOORKRUIS_INWENDIGE_KNOOP(Kd Knoop k, Straal s)
    as  $\leftarrow k_{flags}$ 
    tVlak  $\leftarrow$  KD_VLAK_AFSTAND(kSplit, s,  $\frac{1}{s_d}$ , as)
    linksEerst  $\leftarrow \vec{s}_o[as] < k_{tSplit}$  or ( $\vec{s}_o[as] = k_{tSplit}$  and  $\vec{s}_d[as] \leq 0$ )
    return tVlak, linksEerst
end function
```

---

---

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

---

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

---

met een asgealigneerd vlak. De vector waarvan elke component gelijk is aan de inverse van de overeenkomstige component van de richting van de straal ( $\frac{1}{s_d}$ ) moet voor elke straal slechts één keer berekend worden. Daarnaast bepaalt het algoritme aan de hand van de oorsprong en de richting van de straal of deze straal eerst door de linkse of rechtste kindknoop gaat.

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 *SA<sub>rechts</sub>* en *SA<sub>links</sub>* van de gesplitste volumes en de aantalen primitieven *n<sub>links</sub>* en *n<sub>rechts</sub>* in de nieuwe knopen samen met de oppervlakte *bouwKnoopsA* 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<sub>links</sub>* en *n<sub>rechts</sub>* worden efficiënt berekend door te sweepen.

### 4.3 *RBS*P boom

De implementatie van de *RBS*P boom is gebaseerd op de papers van Kammaje en Mora [KM07] en Budge et al. [BCNJ08]. De *RBS*P boom maakt gebruik van *RBS*P knopen. Tabel 4.2 toont de representatie van zowel inwendige *RBS*P knopen als blad *RBS*P 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 *RBS*P knopen als blad *RBS*P 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 64 blijft, wordt het aantal bits voor de *tweedeKindIndex* variabele bij de inwendige *RBS*P knoop en de *n* variable bij de blad *RBS*P knoop verlaagt tot  $32 - \lceil \log_2(k + 1) \rceil$ . Dit impliceert dat de *RBS*P boom  $2^{(\lceil \log_2(k+1) \rceil - 2)}$  keer minder knopen kan bevatten. Het maximaal aantal primitieven in een bladknoop

---

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

---

```

function BEPAAL_BESTE_SPLITSING(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    for as  $\in \{0, 1, 2\}$  do
         $\forall i : \text{randen}[2i] \leftarrow \text{LINKERRAND}(b_{\text{primitieven}}[i], \text{as})$ 
         $\forall i : \text{randen}[2i + 1] \leftarrow \text{RECHTERRAND}(b_{\text{primitieven}}[i], \text{as})$ 
        SORTEER(randen)
        nlinks, nrechts  $\leftarrow 0$ , bouwKnoop
        for rand  $\in$  randen do
            if randisRechts then
                nrechts  $\leftarrow$  nrechts - 1
            end if
            SAlinks, SArechts  $\leftarrow$  SPLIT(bouwKnoopvolume, as, rand)
            kost  $\leftarrow$  SAH( $\mathcal{K}_d, \mathcal{K}_i, SA_{\text{links}}, SA_{\text{rechts}}, n_{\text{links}}, n_{\text{rechts}}, bouwKnoop_{SA}$ )
            if kost  $<$  besteKost then
                besteKost, besteAs, besteT  $\leftarrow$  kost, as, rand
            end if
            if randisLinks then
                nlinks  $\leftarrow$  nlinks + 1
            end if
        end for
    end for
    return besteAs, besteT, besteKost
end function

```

---

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

Table 4.2: Voorstelling RBSP knoop - Deze tabel toont de nodige variabelen voor zowel inwendige als blad RBSP knopen.

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 VLAKE\_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, bij een  $RBSP$  boom, 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.

---

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

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $RBSP$  Knoop k, Straal s)
    richtingId  $\leftarrow k.flags$ 
     $tVlak, projOorsprong, invProjRichting \leftarrow VLAK_AFSTAND(richtingen[\vec{richtingId}],$ 
     $k_{tSplit}, s)$ 
     $linksEerst \leftarrow projOorsprong < k_{tSplit} \text{ or } (projOorsprong = k_{tSplit} \text{ and}$ 
     $invProjRichting \leq 0)$ 
    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 afstand, projOorsprong, invProjRichting
end function

```

---

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 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 splitsingen 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  $Kd$  knoopdoorkruistechniek 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.

---

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

---

```

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

```

---

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: de 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.

## 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 bladknop (waarde 1) is. Om de oriëntatie van het splitsingsvlak voor te stellen, zijn drie vloottende 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 - in de huidige implementatie - evenveel geheugen moet innemen, neemt elke knoop 160 bits geheugen in. Dit is 2.5 keer zoveel als de  $Kd$  en  $RBSP$  knopen

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

```
function BEPAAL_BESTE_SPLITSING(bouwKnoop)
    besteAs, besteT, besteKost ← None, None, ∞
    besteAsVast, besteTVast, besteKostVast ← None, None, ∞
    for as ∈ {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 ∈ {3, 4, ..., 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 < bouwKnoopn *  $\mathcal{K}_i$  then
        return besteAs, besteT, besteKost
    end if
    if besteKostVast < bouwKnoopn *  $\mathcal{K}_i$  then
        return besteAsVast, besteTVast, besteKostVast
    end if
    return None, None, ∞
end function
```

---

Inwendig	bits	Blad	bits
tSplit	32	primitiefOffset	32
flags	1	flags	1
tweedeKindIndex	31	$n$	31
splitsRichting	96		
	160		64

Table 4.3: Voorstelling  $BSP$  knoop - Deze tabel toont de nodige variabelen voor zowel inwendige als blad  $BSP$  knopen.

en toont één van de nadelen van algemene  $BSP$  bomen.

Algoritme 10 toont het algoritme om inwendige  $BSP$  knopen te doorkruisen. Het gebruikt, net als de  $RBSP$  knoopdoorkruising, de functie VLAK\_INTERSECTIE (algoritme 7) om te intersecteren met een willekeurig georiënteerd vlak.

---

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

---

```

function DOORKRUIS_INWENDIGE_KNOOP( $BSP$  Knoop k, Straal s)
     $tVlak, projOorsprong, invProjRichting \leftarrow VLAK\_AFSTAND(k_{splitsRichting},$ 
     $k_{tSplit}, s)$ 
     $linksEerst \leftarrow projOorsprong < k_{tSplit} \text{ or } (projOorsprong = k_{tSplit} \text{ and}$ 
     $invProjRichting \leq 0)$ 
    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 de volgorde waarin ze gebruikt zijn. Als een vorige richting gevonden wordt, wordt er opnieuw gesplitst volgens deze vorige richting in plaats van de nieuwe richting.

**Algoritme 11** Beste splitsing voor een bouwknop  $b$  bij een  $BSP_{IZE}$  boom.

```
function BEPAAL_BESTE_SPLITSING(bouwKnoop)
    besteAs, besteT, besteKost ← None, None, ∞
    for as ∈ {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 ← BOUW_BVH(b)
    for primitief ∈ bouwKnoop.primitieven do
        for elk splitsingsvlak bij het primitief do
            Bereken het aantal primitieven links en rechts van het splitsingvlak met
            behulp van 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 bestAs, besteT, besteKost
end function
```

---

## 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 blad-knoop 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 bits 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  $Kd$  doorkruisttechniek voor inwendige  $Kd$  knopen. Algoritme 12 toont de aangepaste versie van het doorkruisalgoritme voor  $BSP^{Kd}$  knopen.

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

Inwendig	bits	Blad	bits
tSplit	32	primitiefOffset	32
flags	3	flags	3
tweedeKindIndex	29	$n$	29
splitsRichting	96		
	160		64

Table 4.4: Voorstelling  $BSP^{Kd}$  knoop - Deze tabel toont de nodige variabelen voor zowel inwendige als blad  $BSP^{Kd}$  knopen.

---

**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
         $as \leftarrow k.flags$ 
         $tVlak \leftarrow KD\_VLAK\_AFSTAND(k_{tSplit}, s, \frac{1}{s_d}, as)$ 
         $linksEerst \leftarrow \vec{s}_o[as] < k_{tSplit}$  or ( $\vec{s}_o[as] = k_{tSplit}$  and  $\vec{s}_d[as] \leq 0$ )
        return  $tVlak, linksEerst$ 
    else
         $tVlak, projOorsprong, invProjRichting \leftarrow$ 
        VLAK_AFSTAND( $k_{splitsRichting}, k_{tSplit}, s$ )
         $linksEerst \leftarrow projOorsprong < k_{tSplit}$  or ( $projOorsprong = k_{tSplit}$  and
         $invProjRichting \leq 0$ )
        return  $tVlak, linksEerst$ 
    end if
end function

```

---

$Kd$  en  $BSP$  te gebruiken. Zoals besproken in sectie 2.4 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 bekijken 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 van de boom suggereert wordt in elke knoop langs elk splitsingsrichting geswept om het beste splitsingsvlak op een efficiënte manier te vinden. Hierdoor is er geen nood aan een hulpstructuur zoals bij

---

**Algoritme 13** Beste splitsing voor een bouwknop  $b$  bij een  $BSP_{SIZE}^{Kd}$  boom.

---

```

function BEPAAL_BESTE_SPLITSING(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  $\in bouwKnoop_{primitieven}$  do
            for elk splitsingsvlak bij het primitief do
                Bereken het aantal primitieven links en rechts van het splitsingvlak met
                behulp van 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  $besteKost < bouwKnoop_n * \mathcal{K}_i$  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

```

---

de  $BSP_{IZE}$  boom.

---

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

```

function BEPAAL_BESTE_SPLITSING(bouwKnoop)
    besteAs, besteT, besteKost  $\leftarrow$  None, None,  $\infty$ 
    richtingen  $\leftarrow$  BEPAAL_RICHTINGEN(bouwKnoop, k)
    for richting  $\in$  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

```

---

**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 de zin van de richting zo gekozen wordt dat de eerste niet-nul component van de vector positief is. Als de x- en y-component bijvoorbeeld beiden nul zijn, wordt gezorgd dat de z-component positief is. Dit is nodig omdat de vlakken langs een richting identiek zijn aan de vlakken langs zijn omgekeerde richting.

---

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

```

function BEPAAL_RICHTINGEN(b, k)
    richtingen  $\leftarrow$   $\emptyset$ 
    for i  $\in \{0, 1, 2, \dots, k - 1\}$  do
         $\phi \leftarrow$  UNIFORM(0,  $2\pi$ )
         $\theta \leftarrow$   $\text{acos}(\text{UNIFORM}(-1, 1))$ 
         $\vec{\text{richting}} \leftarrow \{\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta)\}$ 
         $\vec{\text{richting}} \leftarrow$  UPDATE_ZIN( $\vec{\text{richting}}$ )
        ADD(richtingen,  $\vec{\text{richting}}$ )
    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 ← bprimitieven[RANDOM_INT(0, bn)].normaal
        richting ← UPDATE_ZIN(richting)
        ADD(richtingen, richting)
    end while
    return richtingen
end function
```

---

**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.

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

---

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

---

```

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

```

---

---

**Algoritme 18** Beste splitsing voor een bouwknoop b bij een  $BSP_{SWEEP+}^{Kd}$  boom.

---

```

function BEPAAL_BESTE_SPLITSING(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  $<$  bouwKnoopn *  $\mathcal{K}_i$  then
        return besteAs, besteT, besteKost
    end if
    if besteKostVast  $<$  bouwKnoopn *  $\mathcal{K}_i$  then
        return besteAsVast, besteTVast, besteKostVast
    end if
    return None, None,  $\infty$ 
end function

```

---

# Chapter 5

## Resultaten

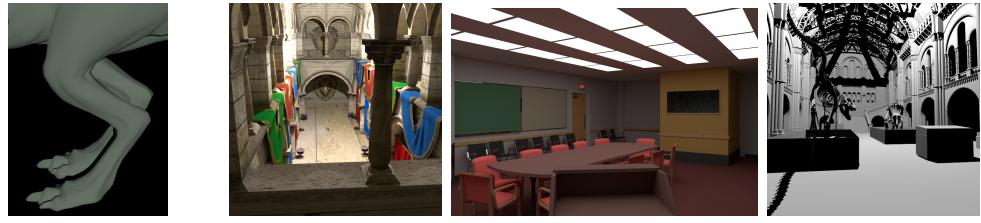
Dit hoofdstuk bespreekt de resultaten van de nieuwe  $BSP_{SWEEP}$  bomen. In sectie 5.1 worden een aantal praktische aspecten omtrent de testopstelling besproken zoals de scenes, de gebruikte parameterwaarden en het gebruikte computersysteem. Daarna bespreken we de invloed van het aantal richtingen 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 [PM19] 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 omdat van de complexe geometrie waarop is ingezoomd. De drie andere scenes zijn realistische indoor scenes. Alle scenes maken gebruik van de in pbrt ingebouwde Halton sampling [PM19] 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.



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

Figure 5.1: Testscenes - De testscenes gebruikt om de nieuwe *BSP* bomen 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

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

Parameter	Waarde
$\mathcal{K}_i$	80
$\mathcal{K}_{d,Kd}$	1
$\mathcal{K}_{d,BSP}$	5
$n_{max}$	1
$d_{max}$	$[k_1 \log_2(n) + k_2]$ met $k_1 = 1.6$ en $k_2 = 2$
$maxIteraties$	500
$\alpha$	0.1

Table 5.2: Gebruikte parameterwaarden - De waarden van de parameters gebruikt bij het testen van de nieuwe *BSP* bomen.

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

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

## 5.2 Afhankelijkheid van het aantal richtingen

In deze sectie wordt de afhankelijkheid van  $BSP_{SWEEP}$  bomen van het aantal gebruikte richtingen  $k$ , besproken. Alle negen varianten hebben de Killeroo Been, Sponza en Conference scenes 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 ordegroottes groter dan die van de  $Kd$  boom. De grafieken voor de verschillende scenes (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 asgealigneerde balken. Hieruit kunnen we afleiden dat het gebruik van convexe veelvlakken tijdens het bouwen ongeveer 25 (2500%) keer trager is dan het gebruik van asgealigneerde balken.

**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 richting 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 minstens 40% bij alle scenes. De  $BSP_{random}$  bomen genereren bij

## 5.2. Afhankelijkheid van het aantal richtingen

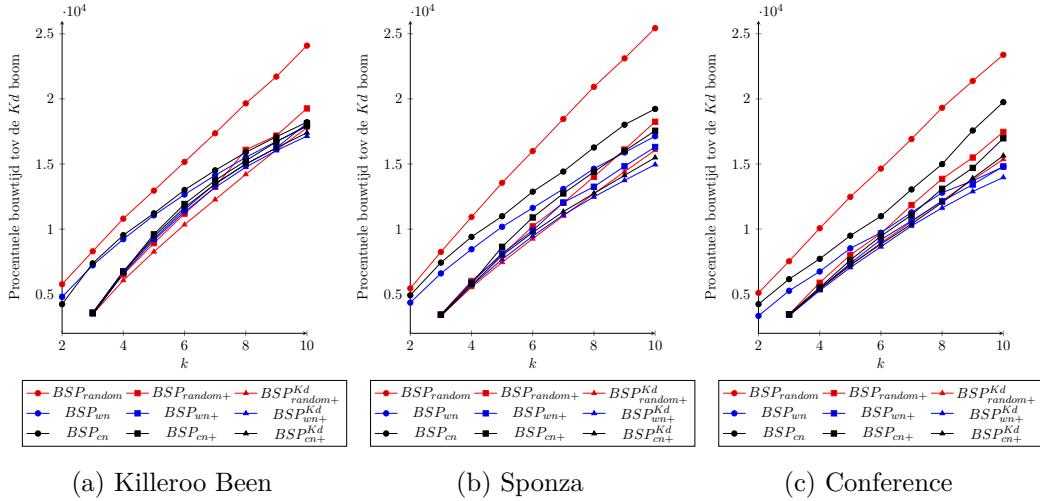


Figure 5.2: Bouwtijd in functie van  $k$  - Deze grafieken tonen voor elke scène de procentuele bouwtijd van de  $BSP_{SWEEP}$  bomen ten opzichte van de bouwtijd van de  $Kd$  boom, in functie van  $k$ .

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  $k$ -waarden hoger dan 4. De  $BSP_{wn+}$  en  $BSP_{cn+}$  bomen genereren bij  $k$ -waarden groter dan 4 zelfs bomen met hogere  $SAH$  kosten.

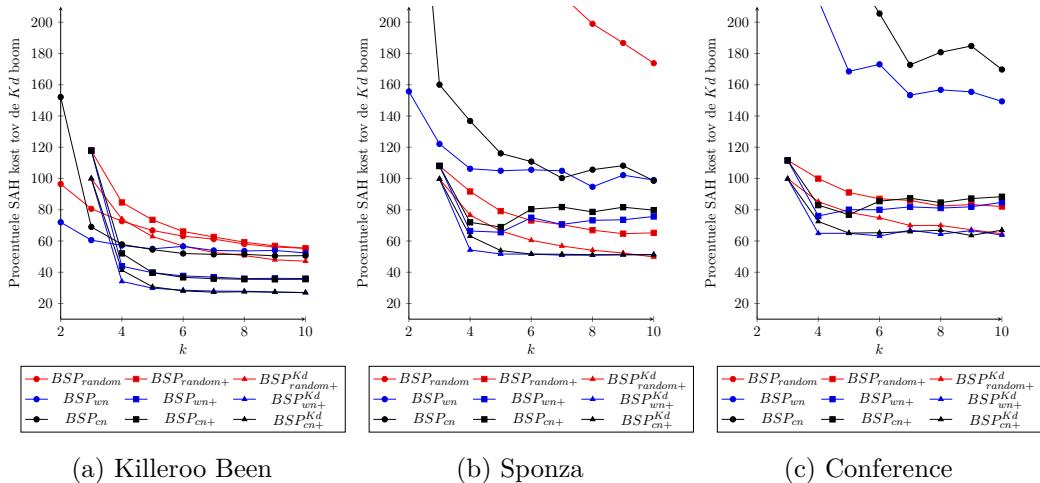


Figure 5.3: *SAH* kost in functie van  $k$  - Deze grafieken tonen voor elke scène de procentuele *SAH* kost van de  $BSP_{SWEEP}$  bomen ten opzichte van de *SAH* kost van de  $Kd$  boom, in functie van  $k$ . De  $BSP_{SWEEP}$  bomen die de  $Kd$  richtingen niet gebruiken, geven voor kleine waarden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

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

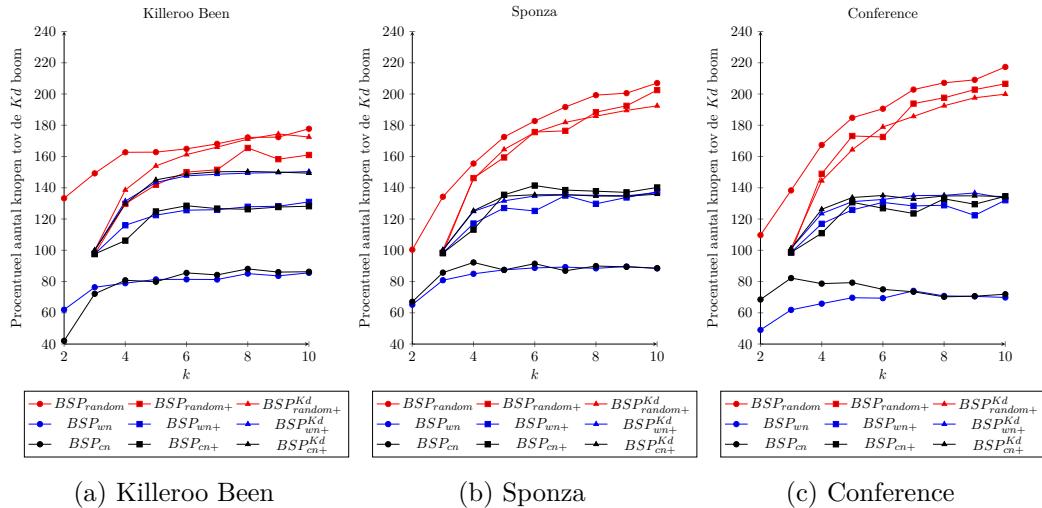


Figure 5.4: Aantal inwendige knopen in functie van  $k$  - Deze grafieken tonen voor elke scène het procentueel aantal inwendige knopen van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal inwendige knopen van de  $Kd$  boom, in functie van  $k$ .

**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 scene. 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

## 5.2. Afhankelijkheid van het aantal richtingen

vanaf een  $k$ -waarde van ongeveer 6. Het kiezen van meer dan 6 richtingen lijkt niet nuttig.

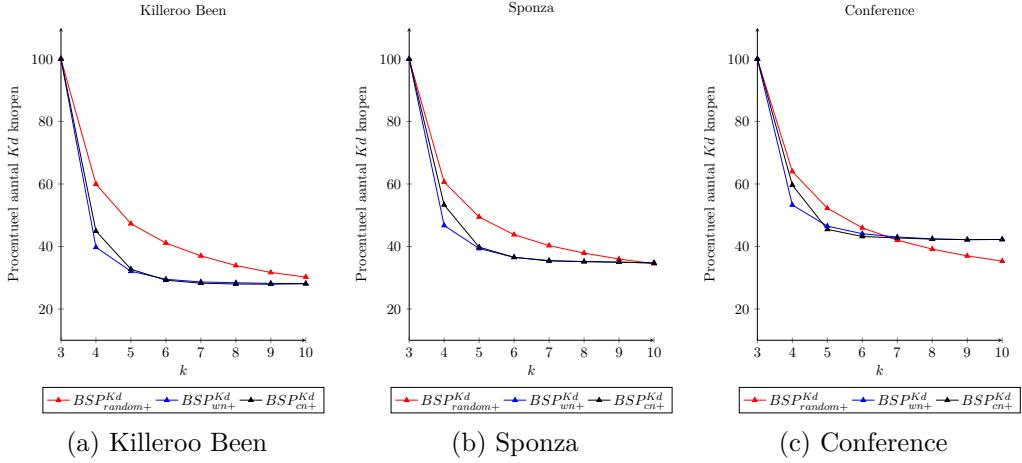


Figure 5.5: Procentueel aantal  $Kd$  knopen - Deze grafieken tonen voor elke scène het procentueel aantal  $Kd$  knopen van de  $BSP_{SWEEP+}^{Kd}$  bomen ten opzichte van het totaal aantal inwendige knopen, in functie van  $k$ .

### 5.2.2 Kwaliteit bomen

**Rendertijd** De rendertijd is de belangrijkste waarde om de kwaliteit van de  $BSP_{SWEEP}$  bomen te vergelijken. Figuur 5.7 toont de rendertijden van de  $BSP_{SWEEP}$  bomen ten opzichte van de rendertijden van de  $Kd$  boom. De rendertijd van de  $BSP_{SWEEP}$  bomen zonder de  $Kd$  richtingen daalt sterk met stijgende  $k$ -waarde, maar enkel bij de Killeroo Been scene worden rendertijden onder die van de  $Kd$  boom bereikt. De  $BSP_{random}$  boom is beduidend trager dan de  $BSP_{wn}$  en  $BSP_{cn}$  bomen. Elk type  $BSP_{SWEEP+}$  boom heeft voor elke scène minstens één  $k$ -waarde waarvoor de rendertijd lager is dan die van de  $Kd$  boom. De minimale rendertijd ligt meestal bij een  $k$ -waarde van ongeveer 5, voor hogere waarden blijft de rendertijd gelijk of stijgt hij zelfs. De  $BSP_{random+}$  boom is vaak trager dan de  $BSP_{wn+}$  en  $BSP_{cn+}$  bomen, maar het verschil is kleiner dan bij de variant zonder de  $Kd$  richtingen. De  $BSP_{SWEEP+}^{Kd}$  bomen zijn altijd sneller dan de  $Kd$  boom en het toevoegen van maar één extra richting ( $k = 4$ ) zorgt al voor een reductie van de rendertijd met minstens 15% bij alle scenes. Bij de  $BSP_{random+}^{Kd}$  boom blijft de rendertijd monotoon dalen met stijgende  $k$ . De rendertijden van de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  varianten daarentegen hebben vaak een minimum voor een  $k$ -waarde van ongeveer 6 en stijgen nadien terug lichtjes.

**Aantal intersecties** De grootste kracht van algemene  $BSP$  bomen zit in het feit dat ze minder straal-driehoekintersecties nodig hebben dan de  $Kd$  boom. Figuur 5.8 toont voor zowel zichtstralen als schaduwstralen het aantal straal-driehoekintersecties van de  $BSP_{SWEEP}$  boom ten opzichte van het aantal straal-driehoekintersecties

## 5.2. Afhankelijkheid van het aantal richtingen

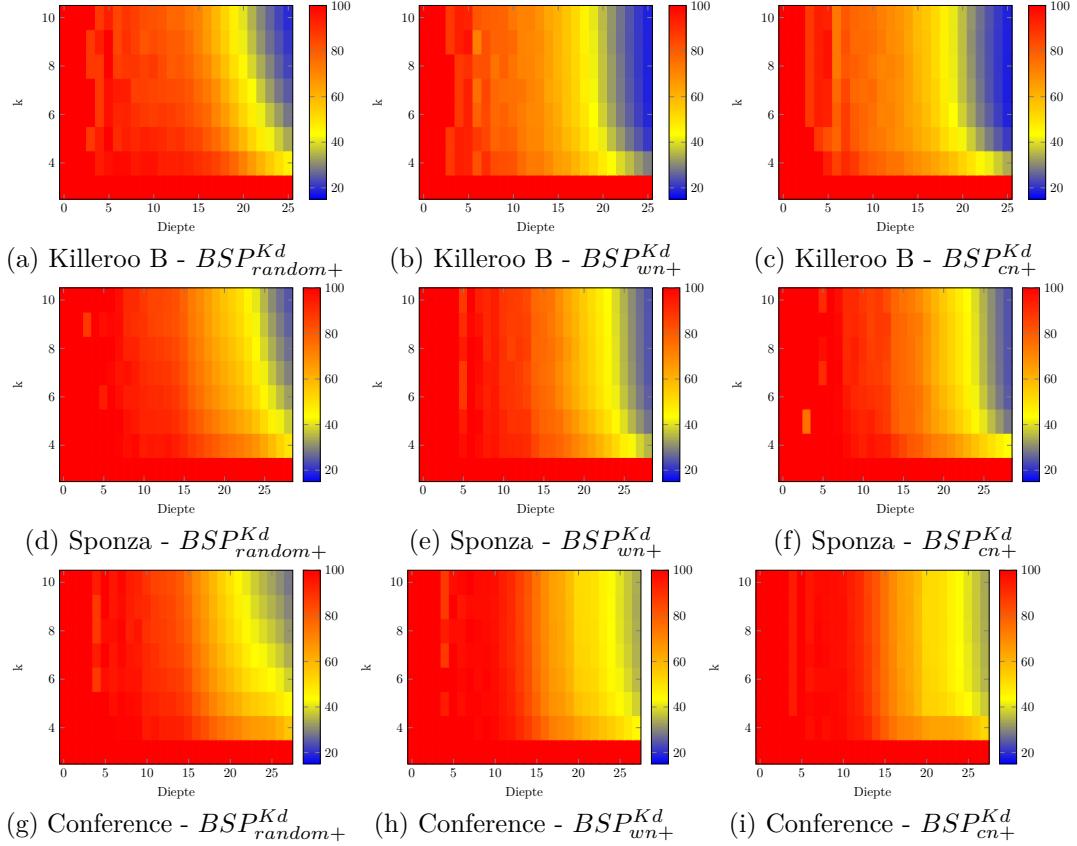


Figure 5.6: Procentueel aantal  $Kd$  knopen per niveau - Deze grafieken tonen voor elke scène het procentueel aantal  $Kd$  knopen van de  $BSP_{SWEEP+}^{Kd}$  bomen op een bepaald niveau ten opzichte van het totaal aantal inwendige knopen op dat niveau, in functie van de diepte en  $k$ .

van de  $Kd$  boom, in functie van  $k$ . Deze figuur vertoont een sterke overeenkomst met figuur 5.7. Dit is logisch aangezien het aantal straal-driehoekintersecties een grote invloed heeft op de rendertijd. Bij de Killeroo Been scene zorgt het toevoegen van één richting ( $k = 4$ ) bij de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen al voor een reductie van het aantal straal-driehoekintersecties met 80% ten opzichte van de  $Kd$  boom. Bij de  $BSP_{random+}^{Kd}$  met  $k = 4$  is deze reductie slechts 35%. Bij de varianten die de normalen gebruiken, stijgt de reductie niet hard voor stijgende  $k$ -waarden, in tegenstelling tot de variant met random richtingen waar de reductie stijgt tot 65%, wat beduidend minder is dan de varianten met normalen. Bij de Sponza scene valt op dat de  $BSP_{random+}^{Kd}$  boom voor een  $k$ -waarde van 10 een grotere reductie doet dan de twee andere varianten. Dit toont aan dat het nuttig zou kunnen zijn om een klein aantal richtingen te bepalen aan de hand van de normalen, en een ander klein aantal richtingen random te genereren. Op die manier zou de reductie snel groot worden en langer blijven stijgen in functie van  $k$ .

## 5.2. Afhankelijkheid van het aantal richtingen

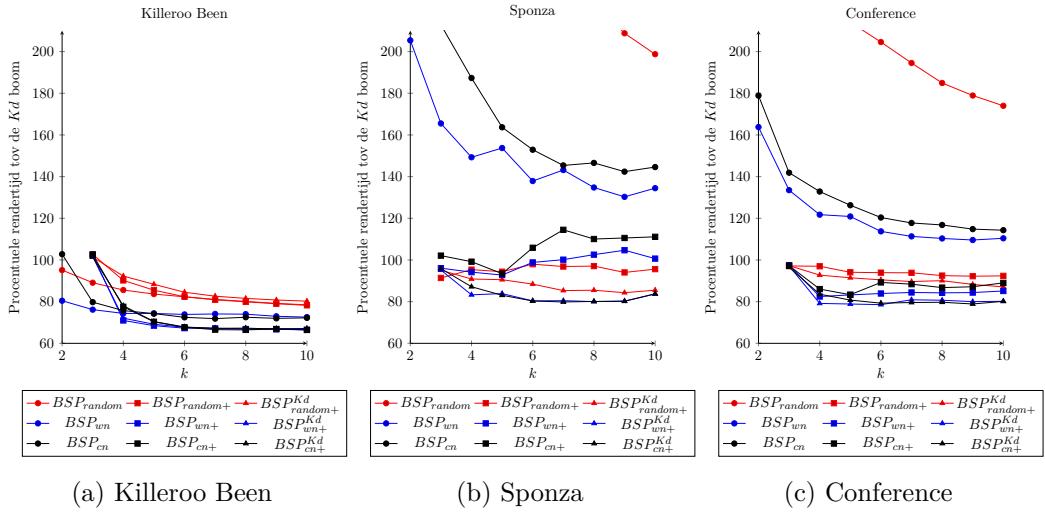


Figure 5.7: Rendertijd in functie van  $k$  - Deze grafieken tonen voor elke scène de procentuele rendertijd van de  $BSP_{SWEEP}$  bomen ten opzichte van de rendertijd van de  $Kd$  boom, in functie van  $k$ . De  $BSP_{SWEEP}$  bomen die de  $Kd$  richtingen niet gebruiken, geven voor kleine waarden van  $k$ , hoge rendertijden, deze worden niet getoond.

**Aantal doorkruisingen** Het nadeel van het beter opsplitsen van bladknopen is het stijgend aantal doorkruisingen van inwendige knopen. Deze stijging wordt deels tegengegaan door het nauwer aansluiten van de  $BSP$  boom aan de scène dan de  $Kd$  boom. Figuur 5.9 toont voor zowel zichtstralen als schaduwstralen het aantal doorkruisingen van inwendige knopen van de  $BSP_{SWEEP}$  boom ten opzichte van het aantal doorkruisingen van inwendige knopen van de  $Kd$  boom, in functie van  $k$ . Het gaat hierbij over het totaal aantal doorkruisingen van inwendige knopen waarbij doorkruisingen van inwendige  $Kd$  en  $BSP$  knopen hetzelfde behandeld worden. Bij de Killeroo Been scène daalt het aantal doorkruisingen tot 80%, dit komt voornamelijk door het nauwer aansluiten van de boom aan de scène. De  $BSP_{random+}^{Kd}$  boom zorgt voor een lichtjes stijgend aantal doorkruisingen in functie van  $k$ , het stijgt tot ongeveer 10% boven het aantal van de  $Kd$  boom. Bij de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen is het aantal doorkruisingen ongeveer gelijk aan het aantal bij de  $Kd$  boom en onafhankelijk van  $k$ .

**$Kd$  doorkruisingen** Bij de  $BSP_{SWEEP+}^{Kd}$  bomen is er een onderscheid tussen het doorkruisen van inwendig  $Kd$  knopen en het doorkruisen van inwendige  $BSP$  knopen. Figuur 5.5 toonde dat het grootste deel van de inwendige knopen,  $BSP$  knopen zijn en figuur 5.6 toonde dat dit enkel voor de lagere niveaus het geval was en dat de hogere niveaus uit bijna uitsluitend  $Kd$  knopen bestaan. De vraag is nu hoeveel procent van de doorkruisingen van inwendige knopen, door de goedkope  $Kd$  knopen gaan en hoeveel procent er door de dure  $BSP$  knopen gaan. Figuur 5.10 toont dat het grootste deel van de doorkruisingen door de goedkope  $Kd$  knopen gaan. Dit komt omdat de bovenste niveaus uit bijna enkel  $Kd$  knopen bestaan. Bij grote  $k$ -waarden kreeg de  $BSP_{random+}^{Kd}$  boom meer  $BSP$  knopen op de hogere niveaus, hierdoor daalt

## 5.2. Afhankelijkheid van het aantal richtingen

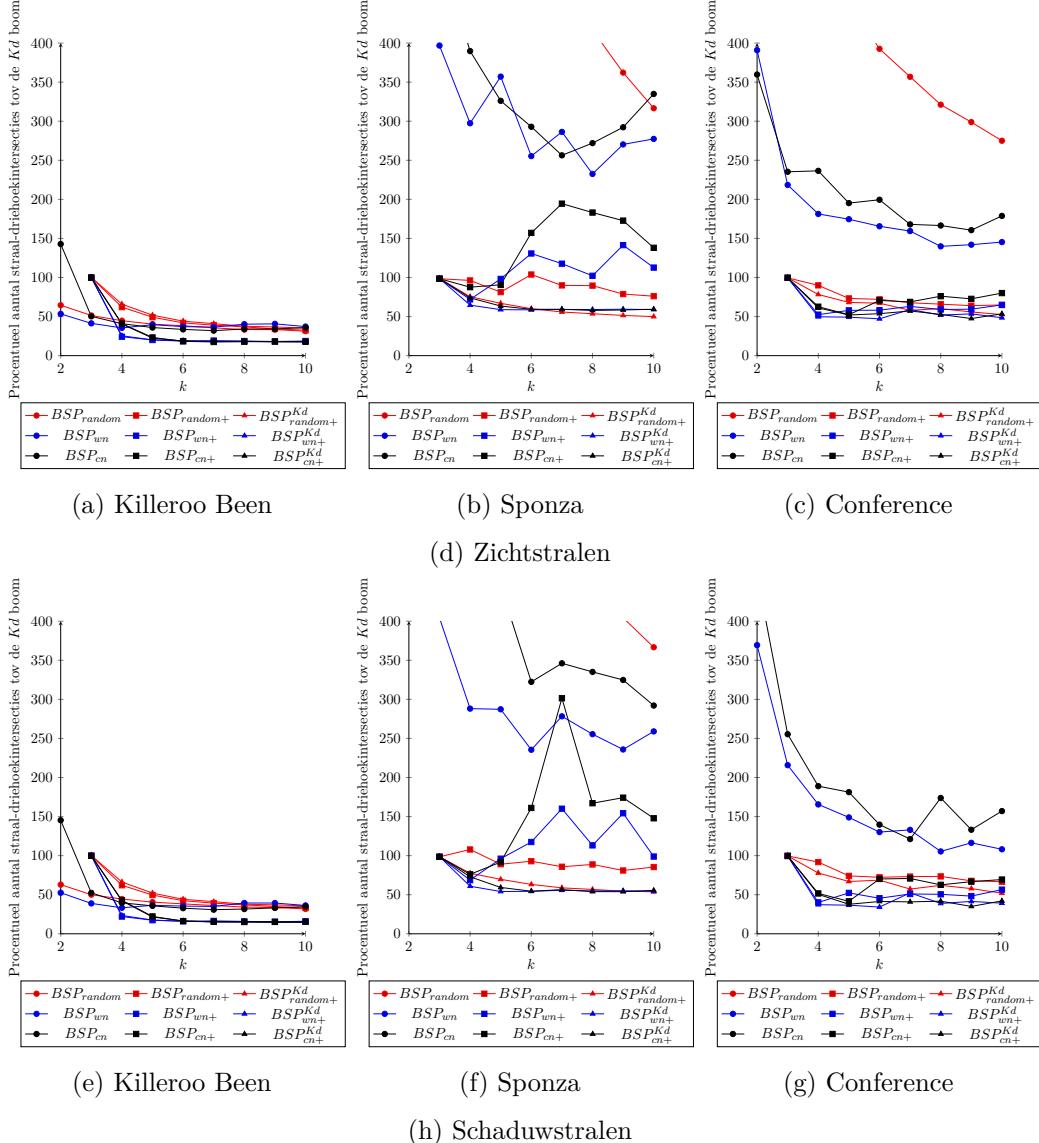


Figure 5.8: Straal-driehoekintersecties in functie van  $k$  - Deze grafieken tonen voor elke scène het procentueel aantal straal-driehoekintersecties van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal straal-driehoekintersecties van de  $Kd$  boom, in functie van  $k$ . De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst en gebruiken een verschillende y-as. De  $BSP_{SWEEP}$  bomen die de  $Kd$  richtingen niet gebruiken, geven voor kleine waarden van  $k$ , een groot aantal intersecties, deze worden niet getoond.

## 5.2. Afhankelijkheid van het aantal richtingen

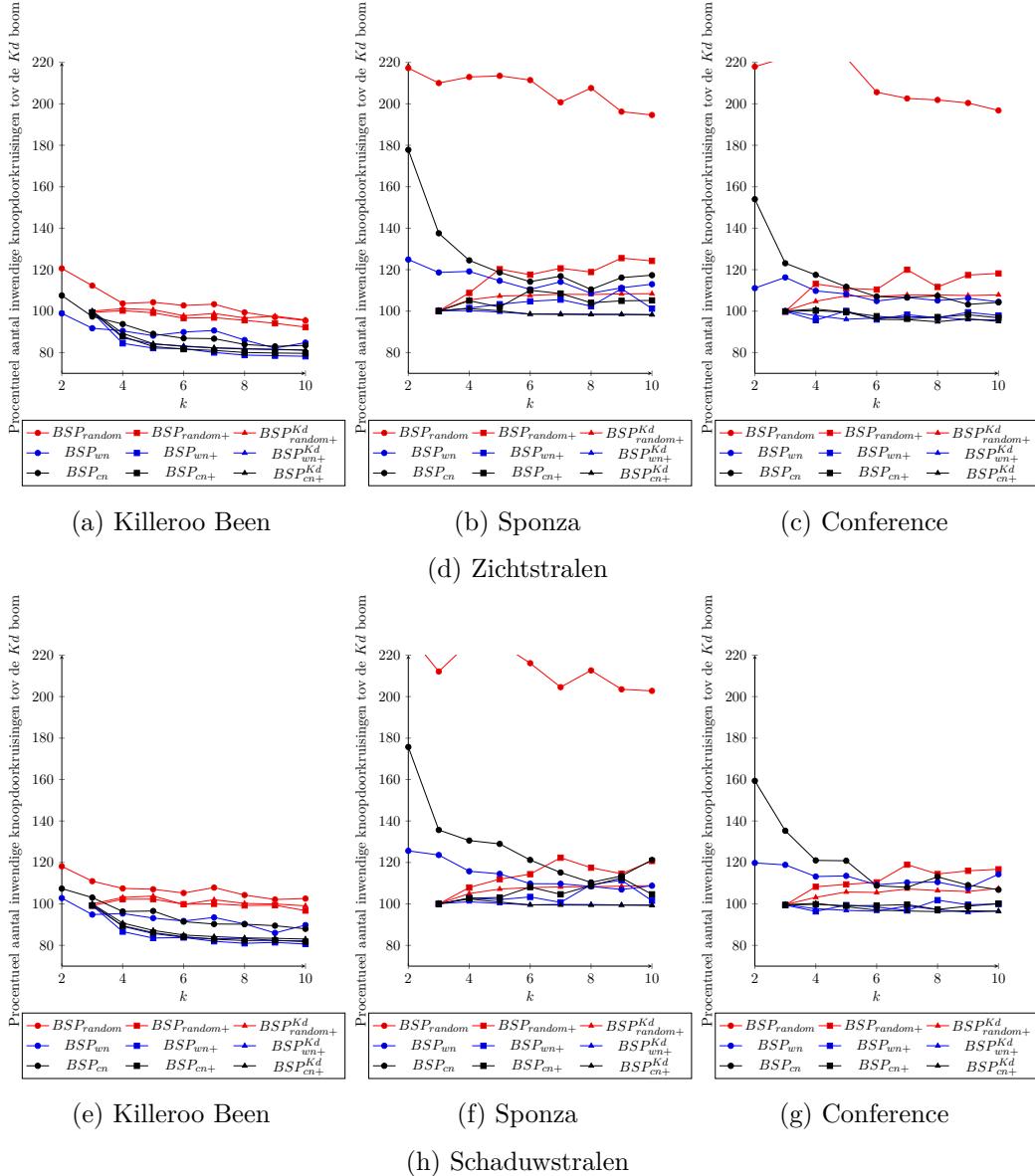


Figure 5.9: Inwendige knoopdoorkruisingen in functie van  $k$  - Deze grafieken tonen voor elke scène het procentueel aantal inwendige knoopdoorkruisingen van de  $BSP_{SWEEP}$  bomen ten opzichte van het aantal inwendige knoopdoorkruisingen van de  $K_d$  boom, in functie van  $k$ . De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst en gebruiken een verschillende y-as. De  $BSP_{SWEEP}$  bomen die de  $K_d$  richtingen niet gebruiken, geven voor kleine waarden van  $k$ , een groot aantal doorkruisingen, deze worden niet getoond.

### 5.3. Vergelijking met bestaande bomen

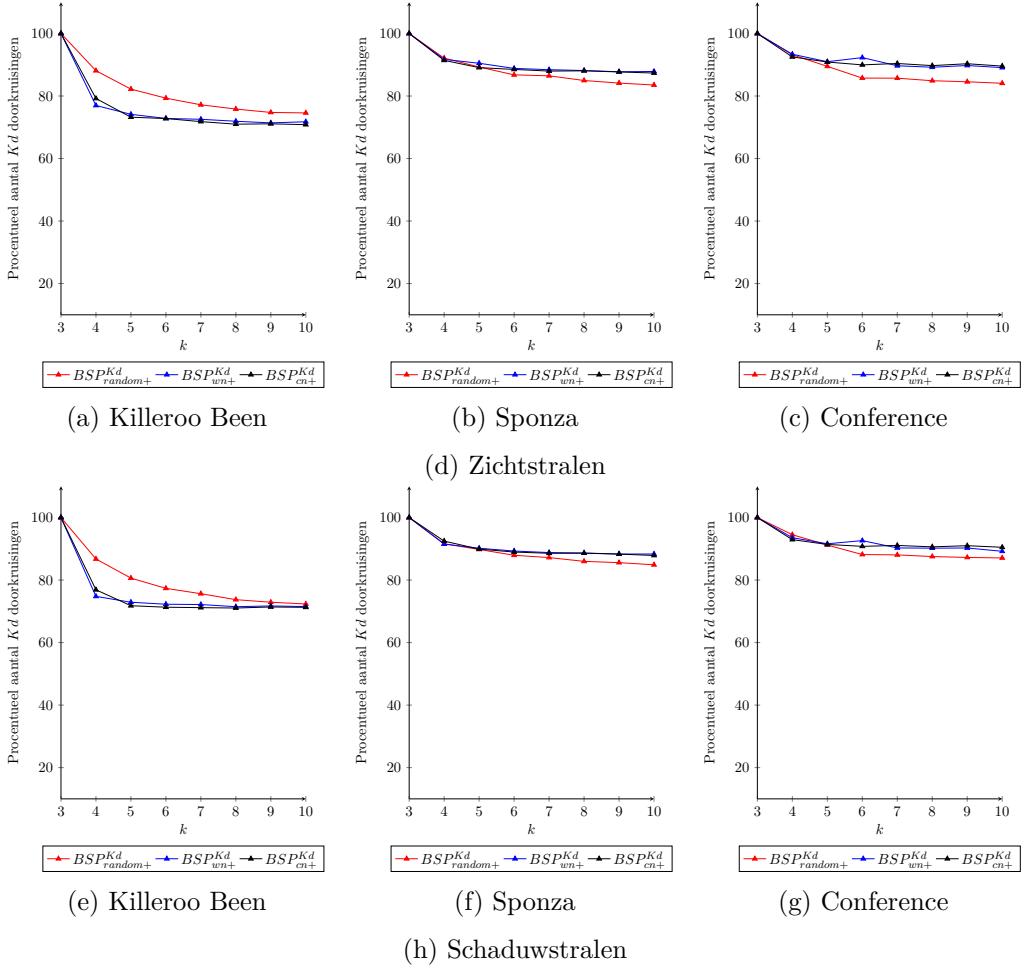


Figure 5.10: Aantal  $Kd$  doorkruisingen in functie van  $k$  - Deze grafieken tonen voor elke scène het procentueel aantal  $Kd$  doorkruisingen van de  $BSP_{SWEEP}$  bomen, in functie van  $k$ . De grafieken voor de zichtstralen en schaduwstralen zijn opgesplitst.

het procentueel aantal doorkruisingen door  $Kd$  knopen.

### 5.3 Vergelijking met bestaande bomen

In deze sectie worden de beste  $BSP_{SWEEP}$  bomen vergeleken met een aantal bestaande bomen en de nieuwe  $RBSP^{Kd}$  boom. De drie  $BSP_{SWEEP}$  bomen die gebruikt worden in de vergelijking zijn: de  $BSP_{random+}^{Kd}$  met  $k = 10$  en de  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen met  $k = 6$ . Voor de  $RBSP^{(Kd)}$  bomen wordt de versie met  $k = 13$  gebruikt. Elk type  $BSP$  boom heeft elke testscene zeven keer gerenderd en net zoals in 5.2 wordt de uitvoering met als rendertijd de mediaan van de zeven uitvoeringen, als representatieve uitvoering gekozen.

**Render- en bouwtijd** Tabel 5.4 toont de render- en bouwtijden van de verschillende bomen voor de vier scenes. De bouwtijd van de algemene *BSP* bomen is 2 à 3 ordegroottes groter dan die van de *Kd* boom. De *BSP<sub>SWEET</sub>* bomen hebben kleinere bouwtijden dan de *BSP<sub>IZE</sub>* bomen. De bouwtijd van de *BSP<sub>SWEET</sub>* bomen stijgt ook trager met stijgende *n* dan de bouwtijd van de *BSP<sub>IZE</sub>* bomen. Voor de Killeroo Been scene duurt het bouwen van de *BSP<sub>IZE</sub>* boom 2 keer langer dan het bouwen van de *BSP<sub>SWEET</sub>* bomen met zes richtingen, voor de Museum scene is dit al 9 keer langer. De bouwtijd van de *RBSP* boom is even goed als die van de *BSP<sub>SWEET</sub>* bomen, de grotere waarde komt door het groter aantal richtingen. Dit is logisch omdat de *RBSP* boom een *BSP<sub>SWEET</sub>* boom is die steeds dezelfde richtingen kiest in elke knoop. De optimalisaties van Budge et al. [BCNJ08] voor het bouwen van *RBSP* bomen zouden ook deels kunnen worden toegepast op de *BSP<sub>SWEET</sub>* boom.

Op vlak van rendertijd presteren de *BSP<sub>wn+</sub><sup>Kd</sup>* en *BSP<sub>cn+</sub><sup>Kd</sup>* bomen het beste op alle scenes. Deze twee bomen hebben bij elke scene een zeer gelijkaardige rendertijd, met een licht voordeel voor de *BSP<sub>wn+</sub><sup>Kd</sup>* boom. Bij elke scene zijn ze minstens 20% sneller dan de *Kd* boom en bij de Museum scene is het voordeel (bijna 40%) zelfs nog groter dan bij de Killeroo Been scene. De *BSP<sub>random+</sub><sup>Kd</sup>* boom presteert op alle scenes het derde beste, behalve op de museum scene waar de *RBSP<sup>Kd</sup>* boom lichtjes beter presteert. Deze andere nieuwe boom, de *RBSP<sup>Kd</sup>* boom, presteert ook op alle scenes beter dan de bestaande *BSP* bomen, behalve bij de Conference scene waar het net moet onderdoen voor de *BSP<sub>IZE</sub><sup>Kd</sup>* boom. Alle algemene *BSP* bomen met de *Kd*-doorkruisoptimalisatie zijn voor alle scenes sneller dan de *Kd* boom, behalve de *BSP<sub>IZE</sub><sup>Kd</sup>* boom bij de Sponza scene.

Bij de Killeroo Been scene valt op dat de *BSP<sub>IZE</sub>* en *RBSP* bomen die geen gebruik maken van de snelle *Kd* doorkruising, sneller zijn dan de versies die wel gebruik maken van die snellere doorkruising. Dit fenomeen treedt ook op bij de *BSP<sub>SWEET</sub>* bomen en toont aan dat de aangepaste versie van de *SAH* in combinatie met de gebruikte parameterwaarden, niet optimaal is. De Sponza scene daarentegen is de scene waarbij de *Kd*-doorkruisoptimalisatie het meeste voordeel biedt. Bij de Museum scene valt het op dat de *RBSP* boom een zeer hoge rendertijd heeft. De reden hiervoor is dat hij een deel van het dak van het museum niet goed opgesplitst krijgt en enkele bladknopen met meer dan 1000 driehoeken maakt. De versie die de aangepaste *SAH* gebruikt, heeft hier geen last van.

**Aantal intersecties en doorkruisingen** De rendertijd wordt voornamelijk bepaald door twee zaken: het aantal straal-driehoekintersecties en het aantal knoopdoorkruisingen. Tabel 5.5 toont voor zowel zichtstralen als schaduwstralen het aantal straal-driehoekintersecties bij de verschillende bomen voor de vier scenes. Net als bij de rendertijd zorgen de *BSP<sub>wn+</sub><sup>Kd</sup>* en *BSP<sub>cn+</sub><sup>Kd</sup>* bomen voor de grootste daling in straal-driehoekintersecties. Bij elke scene daalt het aantal straal-driehoekintersecties met minstens 40% en bij de Killeroo Been scene zelfs met 80%. Bij de Sponza scene

### 5.3. Vergelijking met bestaande bomen

Boom	$k$	Killeroo Been		Sponza		Conference		Museum	
		R	B	R	B	R	B	R	B
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$BSP_{IZE}$		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%
$BSP_{wn+}^{Kd}$	6	67.7%	11 300%	80.4%	9460%	78.6%	8650%	63.4%	11 500%
$BSP_{cn+}^{Kd}$	6	67.8%	11 700%	80.3%	9740%	79.3%	8840%	65.9%	12 100%
$BSP_{random+}^{Kd}$	10	80.2%	17 800%	85.5%	16 100%	87.1%	15 400%	78.4%	21 800%
$RBSP$	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%

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

daalt het aantal straal-driehoekintersecties het minste, een mogelijke verklaring hiervoor is het feit dat die scene het sterkst gealigneerd is met de assen waardoor  $Kd$  richtingen goed werken. De  $BSP_{random+}^{Kd}$  boom heeft het derde minste aantal straal-driehoekintersecties maar het zijn er wel beduidend meer dan bij de twee andere varianten. Dit kan verklaard worden door het feit dat de random richtingen vaak minder goed zijn waardoor vaker de slechte  $Kd$  knopen of slechtsplitsende  $BSP$  knopen gebruikt worden. De  $BSP_{SWEEP}$  bomen die in elke knoop andere splitsingsvlakken bekijken, doen beduidend minder intersecties dan de  $BSP_{IZE}$  en  $RBSP$  bomen die steeds dezelfde bekijken. De  $RBSP^{Kd}$  boom doet het in elke scene beter dan de  $BSP_{IZE}^{Kd}$  boom en de  $Kd$  boom.

Figuur 5.11 toont dat bij de algemene  $BSP$  bomen, bladknopen met weinig driehoeken voor een grotere proportie van het totaal aantal straal-driehoekintersecties zorgen dan bij de  $Kd$  boom. Dit toont dat algemene  $BSP$  bomen bladknopen beter opsplitsen in kleinere bladknopen en zo het totaal aantal straal-driehoekintersecties verminderen. Tabel 5.6 toont voor zowel zichtstralen als schaduwstralen het aantal inwendige knoopdoorkruisingen bij de verschillend bomen voor de vier scenes. Door de betere opsplitsing van bladknopen in kleinere bladknopen, hebben de algemene  $BSP$  bomen meer interne knopen dan de  $Kd$  boom en dit leidt tot meer interne knoopdoorkruisingen. De  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen sluiten echter veel nauwer aan bij de scene, waardoor het totaal aantal interne knoopdoorkruisingen lager is dan bij de  $Kd$  boom voor alle scenes. De  $BSP_{random+}^{Kd}$ ,  $RBSP^{Kd}$  en  $BSP_{IZE}^{Kd}$  bomen sluiten minder goed aan waardoor het aantal doorkruising ongeveer 10% hoger ligt dan bij de  $Kd$  boom.

Tabellen 5.7 en 5.8 tonen de *false color* afbeeldingen van de straal-driehoekintersecties van zicht- en schaduwstralen samen. Bij tabel 5.7 gebruiken alle *false color* afbeelding van een scene dezelfde kleurschaal zodat verschillende bomen vergeleken kunnen wor-

### 5.3. Vergelijking met bestaande bomen

Boom	$k$	Killeroo Been		Sponza		Conference		Museum	
		ZI	SI	ZI	SI	ZI	SI	ZI	SI
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$BSP_{IZE}$		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%
$BSP_{wn+}^{Kd}$	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%
$BSP_{random+}^{Kd}$	10	35.2%	35.8%	49.9%	53.8%	52.5%	52.0%	36.8%	52.3%
$RBSP$	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%

Table 5.5: Vergelijking straal-driehoekintersecties van  $BSP$  bomen - Deze tabel toont statistieken over het procentueel aantal straal-driehoekintersecties van  $BSP$  bomen ten opzichte van het aantal straal-driehoekintersecties van de  $Kd$  boom voor verschillende scenes. ZI staat voor zichtstaalintersecties en SI voor schaduwstraalintersecties.

Boom	$k$	Killeroo Been		Sponza		Conference		Museum	
		ZD	SD	ZD	SD	ZD	SD	ZD	SD
$Kd$		100%	100%	100%	100%	100%	100%	100%	100%
$BSP_{IZE}$		97.3%	101%	130%	128%	117%	115%	109%	98.4%
$BSP_{IZE}^{Kd}$		100%	102%	117%	109%	107%	107%	107%	103%
$BSP_{wn+}^{Kd}$	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%
$BSP_{random+}^{Kd}$	10	95.8%	99.1%	108%	109%	108%	108%	106%	105%
$RBSP$	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%

Table 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 zichtstraaldoorkruisingen en SD voor schaduwstraaldoorkruisingen.

den, bij tabel 5.8 is de kleurschaal bij elke afbeelding anders zodat de pijnpunten van de bomen zichtbaar zijn. In de eerste tabel valt op dat de  $BSP_{SWEEP+}^{Kd}$  bomen over de hele scene minder straal-driehoekintersecties nodig hebben dan de andere bomen. De toevoeging van de  $Kd$  richtingen aan de  $RBSP$  en  $BSP_{IZE}$  bomen zorgt ook voor een duidelijke verbetering. Bij de tweede tabel valt het op dat de  $BSP_{SWEEP+}^{Kd}$  bomen minder last hebben van zogenaamde *hotspot* regio's zoals bijvoorbeeld de stoelen in de Conference scene.

### 5.3. Vergelijking met bestaande bomen

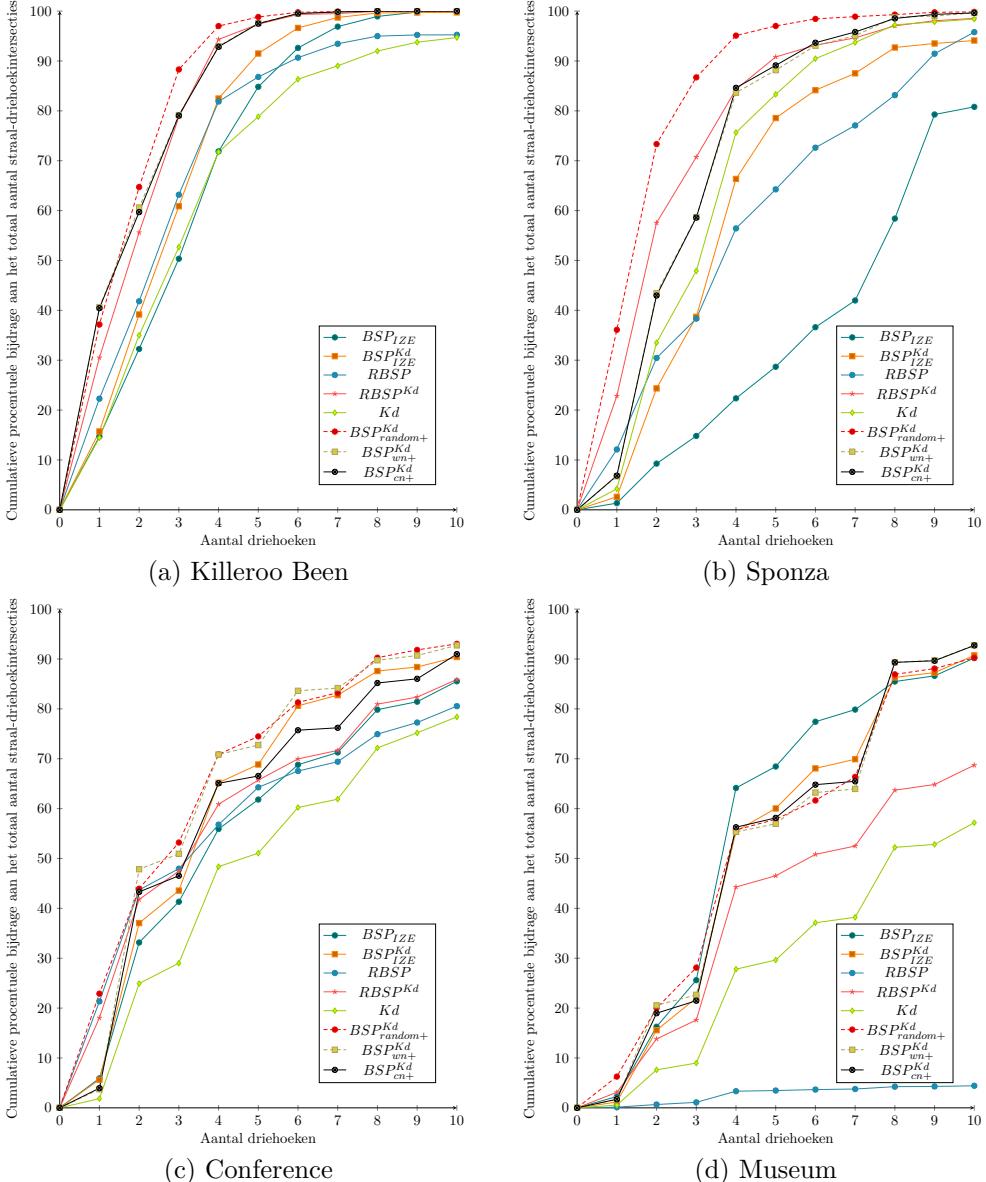


Figure 5.11: Cumulatieve procentuele bijdrage aan het totaal aantal straal-driehoekintersecties in functie van de bladknooppootte - Elk van deze grafieken toont voor een bepaalde scene voor elke  $BSP$  boom de cumulatieve procentuele bijdrage aan het totaal aantal straal-driehoekintersecties (met zicht- en schaduwstralen) in functie van de bladknooppootte. Voor een waarde van  $x$  op de x-as, wordt het procentueel aantal straal-driehoekintersecties dat gebeurt in bladknopen met  $x$  of minder driehoeken, getoond.

Boom	Killeroo Been	Sponza	Conference	Museum
$Kd$				
$BSP_{IZE}$				
$BSP_{IZE}^{Kd}$				
$BSP_{wn+}^{Kd}$				
$BSP_{cn+}^{Kd}$				
$BSP_{random+}^{Kd}$				
$RBSP$				
$RBSP^{Kd}$				

Table 5.7: *False color* afbeeldingen straal-driehoekintersecties met gelijke kleurwaarden - *False color* afbeeldingen van de vier scenes voor alle bomen. Voor alle scenes gebruiken alle bomen dezelfde kleurwaarden waarbij donkerblauw lage waarden voorstelt en geel hoge waarden. De *RBSP* boom is weggelaten bij de *Museum* scene omdat die zoveel meer intersecties doet, dat de *false color* afbeelding van alle andere bomen volledig blauw zouden zijn.

### 5.3. Vergelijking met bestaande bomen

---

Boom	Killeroo Been	Sponza	Conference	Museum
$Kd$				
$BSP_{IZE}$				
$BSP_{IZE}^{Kd}$				
$BSP_{wn+}^{Kd}$				
$BSP_{cn+}^{Kd}$				
$BSP_{random+}^{Kd}$				
$RBSP$				
$RBSP^{Kd}$				

Table 5.8: *False color* afbeeldingen straal-driehoekintersecties met verschillende kleurwaarden - *False color* afbeeldingen van de vier scenes voor alle bomen. Elke *false color* afbeelding gebruikt zijn eigen kleurwaarden waarbij donkerblauw lage waarden voorstelt en geel hoge waarden.

**$Kd$  knopen** De  $BSP_{SWEEP+}^{Kd}$  bomen hebben een gelijk aantal of lichtjes hoger aantal knoopdoorkruisingen dan de  $Kd$  boom. Dit aantal is de som van het aantal goedkope  $Kd$  knoopdoorkruisingen en het aantal dure  $BSP$  knoopdoorkruisingen. Tabel 5.9 toont voor elke scène voor elke  $BSP$  boom hoeveel procent van de inwendige knopen,  $Kd$  knopen zijn en voor zowel zichtstralen als schaduwstralen hoeveel procent van de knoopdoorkruisingen, doorkruisingen door deze  $Kd$  knopen zijn. Het valt op dat bij de  $BSP^{Kd}$  bomen het grootste deel (80 à 90%) van de doorkruisingen, door de goedkope  $Kd$  doorkruisingen gaat, terwijl het procentueel aantal  $Kd$  knopen veel lager ligt. In vergelijking met de  $BSP_{IZE}^{Kd}$  boom hebben de  $BSP_{SWEEP+}^{Kd}$  bomen procentueel minder  $Kd$  knopen, maar het procentueel aantal  $Kd$  doorkruisingen is ongeveer gelijk. Dit betekent dat de  $BSP_{SWEEP+}^{Kd}$  bomen elk van hun  $Kd$  knopen gemiddeld vaker doorkruisen dan de  $BSP_{IZE}^{Kd}$  boom en op die manier de snellere  $Kd$  doorkruising beter benutten. De  $BSP_{IZE}$  en  $RBSP$  bomen bevatten  $Kd$  knopen, maar deze worden behandeld als algemene  $BSP$  knopen, daarom zijn hun doorkruisingen in de tabel op 0 gezet. De tabel toont duidelijk dat de aangepaste heuristiek zorgt dat er meer  $Kd$  knopen in de  $BSP_{IZE}^{Kd}$  en  $RBSP^{Kd}$  bomen zitten. Figuur 5.12 toont dat dit het geval is op alle niveaus van de boom en dat de bovenste niveaus van de bomen met aangepaste heuristiek bijna uitsluitend uit  $Kd$  knopen bestaan.

### 5.3. Vergelijking met bestaande bomen

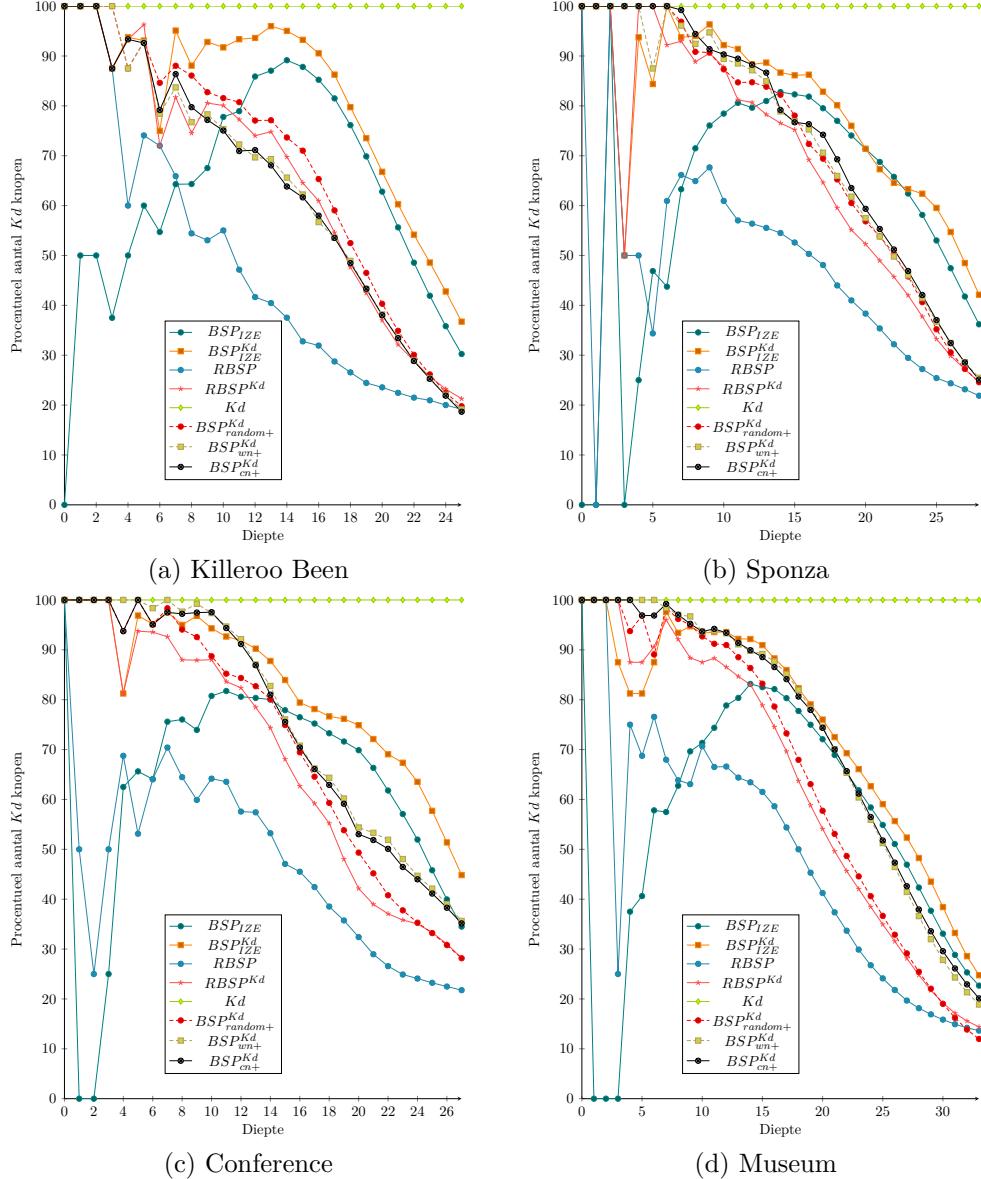


Figure 5.12: Procentueel aantal  $Kd$  knopen per niveau - Elk van deze grafieken toont voor een bepaalde scène voor elke  $BSP$  boom het procentueel aantal  $Kd$  knopen in functie van de diepte.

Boom	$k$	Killeroo Been			Sponza		
		$Kd$	ZD $Kd$	SD $Kd$	$Kd$	ZD $Kd$	SD $Kd$
$Kd$		100%	100%	100%	100%	100%	100%
$BSP_{IZE}$		49.4%	0%	0%	50.9%	0%	0%
$BSP_{IZE}^{Kd}$		55.1%	88.3%	88.3%	55.6%	88.2%	90.9%
$BSP_{wn+}^{Kd}$	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%
$BSP_{random+}^{Kd}$	10	30.2%	74.6%	72.3%	34.5%	83.5%	84.9%
$RBSP$	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 $Kd$	$Kd$	ZD $Kd$	SD $Kd$
$Kd$		100%	100%	100%	100%	100%	100%
$BSP_{IZE}$		49.8%	0%	0%	38.5%	0%	0%
$BSP_{IZE}^{Kd}$		58.4%	89.7%	91.0%	41.7%	87.1%	87.1%
$BSP_{wn+}^{Kd}$	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%
$BSP_{random+}^{Kd}$	10	35.3%	84.1%	87.1%	21.3%	83.1%	83.6%
$RBSP$	13	25.1%	0%	0%	18.3%	0%	0%
$RBSP^{Kd}$	13	34.3%	80.4%	82.6%	22.2%	78.9%	78.4%

Table 5.9: Vergelijking proportie  $Kd$  knopen en proportie  $Kd$  knoopdoorkruisingen van  $BSP$  bomen - Deze tabel toont statistieken over het procentueel aantal inwendige  $Kd$  knopen, het procentueel aantal  $Kd$  zichtstraaldoorkruisingen (ZD) en het procentueel aantal  $Kd$  schaduwstraaldoorkruisingen (SD).

# Chapter 6

## Conclusie

### 6.1 Conclusies $BSP_{SWEEP}$

De ontworpen  $BSP_{SWEEP}$  boom is een zeer nuttig en uitbreidbaar concept om algemene  $BSP$  bomen te bouwen. De lokale geometrie van de knoop kan zeer eenvoudig in rekening gebracht worden om goede splitsingsrichtingen te genereren. Door het sweepen blijft de bouwtijd relatief beperkt in verhouding met de enige andere algemene  $BSP$  boom: de  $BSP_{IZE}$  boom. De bouwtijd is wel twee ordegroottes groter dan die van de  $Kd$  boom. Voor tal van toepassingen is dit echter geen probleem aangezien de boom op voorhand één keer gebouwd kan worden.

De simpele  $BSP_{SWEEP}$  bomen die geen gebruik maken van de  $Kd$  richtingen, moeten bij normale scènes onderdoen voor de  $Kd$  boom en  $BSP_{IZE}$  bomen. Bij de speciale Killeroo Been scene tonen deze bomen wel hun mogelijkheden, aangezien ze daarop beter werken dan de  $Kd$  boom. De  $BSP_{SWEEP+}$  bomen, die steeds de  $Kd$  richtingen nemen als eerste drie richtingen, presteren voor bepaalde waarden van  $k$  (het aantal richtingen), beter dan de  $Kd$  en  $BSP_{IZE}$  bomen. Deze bomen zullen in de praktijk echter nooit gebruikt worden omdat ze als opstapje dienen naar de  $BSP_{SWEEP+}^{Kd}$  bomen, die altijd beter presteren dan de bestaande  $BSP$  bomen.

De  $BSP_{random+}^{Kd}$  boom toont dat het kiezen van andere splitsingsrichtingen in elke knoop, de boom veel beter maakt. Zelfs zonder rekening te houden met de lokale geometrie, kan dan een boom gebouwd worden die beter is dan de  $Kd$  boom of goed uitgedachte  $BSP_{IZE}^{Kd}$  boom. Dit komt voornamelijk door het grotere aantal verschillende splitsingsvlakken -  $O(n \log(n))$  in plaats van  $O(n)$  - dat bekijken wordt tijdens het bouwen. De  $BSP_{wn+}^{Kd}$  en  $BSP_{cn+}^{Kd}$  bomen tonen dat het kiezen van splitsingsrichtingen afhankelijk van de lokale geometrie deze boom nog beduidend beter kan maken. Ze verminderen bij alle testscènes het aantal straal-driehoekintersecties met meer dan 40% en de rendertijd met meer dan 20% ten opzichte van de  $Kd$  boom. Deze bomen tonen ook dat het toevoegen van slechts één richting bovenop de  $Kd$  richtingen, al een groot effect kan hebben op deze waarden. De clustering lijkt geen voordeel te bieden ten opzichte van het kiezen van willekeurige normalen.

## 6.2 Toekomstig onderzoek

Toekomstig onderzoek zou zich moeten toespitsen op een aantal zaken: het verbeteren van de bouwtijd, het bepalen van het beste aantal richtingen en het bepalen van betere richtingen. Het verbeteren van de bouwtijd zou kunnen door gebruik te maken van de methode van Budge et al. [BCNJ08] om de *SA* van de gesplitste volumes incrementeel te berekenen in plaats van steeds helemaal vanaf nul. Bepaalde delen van het bouwproces zijn ook goed paralleliseerbaar. Zo kan er in parallel over de verschillende splitsingsrichtingen geswept worden. Het aantal richtingen dat gegenereerd wordt zou afhankelijk gemaakt kunnen worden van de diepte. Op hogere niveaus willen we voornamelijk  $Kd$  knopen dus is het mogelijk om daar enkel de  $Kd$  richtingen of de  $Kd$  richtingen plus een klein aantal *BSP* richtingen te gebruiken. Op de lagere niveaus zouden dan meer *BSP* richtingen gebruikt worden.

Het is ook een mogelijkheid om te stoppen met zoeken naar het beste splitsingsvlak, als er al een splitsingsvlak gevonden is dat de driehoeken opdeelt in twee disjuncte delen. Dit laat toe om hogere  $k$ -waarden te kiezen, zonder een stijging in bouwtijd te krijgen. In sectie 5.2.2 viel het ook op dat het aantal straal-driehoekintersecties sterk daalde als er één richting bepaald door de normalen gebruikt werd, maar dat meer richtingen aan de hand van de normalen, dit aantal niet meer lieten dalen. Bij de  $BSP_{random+}^{Kd}$  boom bleef het aantal intersecties dalen met een stijgend aantal richtingen. Het lijkt ons dus een goed idee om een hybride oplossing te maken die een klein aantal richtingen bepaald aan de hand van de normalen en een groter aantal richtingen random genereert als er nog geen goede splitsing gevonden is.

## Chapter 7

# English Commentary

The first chapter gives a general overview of ray tracing and the concept of acceleration structures. It also specifies the goal of the thesis: 'Creating a general *BSP* tree that is superior to the *Kd* tree in terms of render time'.

The second chapter gives an overview of the existing types of *BSP* trees:

- The *Kd* tree is a *BSP* tree which only uses axis-aligned splitting planes. Figure 2.3 shows its splitting power.
- The *RBSP* tree is a *BSP* tree that - like the *Kd* tree - has a fixed set of splitting axes, but this set contains more axes than just the x-, y- and z-axis. Figure 2.4 shows its splitting power.
- The *BSPIZE* tree is a general *BSP* tree that uses that *Kd* splitting planes and four additional planes for each triangle: the plane of the triangle itself and the three planes that go through the sides of the triangle and are perpendicular to it. It was the first and only - before the *BSPSWEEP* tree - general *BSP* tree that could compete with the *Kd* tree in terms of render time.

Table 2.1 shows in its last two rows the amount of distinct splitting planes used in each level and the total amount of distinct splitting planes used for each of these types of *BSP* trees when there are n triangles in the scene. In this thesis it is noticed that the same splitting planes are tested in all levels of the tree. This means that any two triangles that can't be separated from each other in the root node, can't be separated in the entire tree.

The third chapter presents the new type of *BSP* tree: the *BSPSWEEP* tree. The chapter starts with a mathematical explanation as to why - under certain conditions - it is always advantageous to split a leaf node with more than one triangle into child nodes. This also implies that a tree that tries more distinct splitting planes - and thus has more splitting power - has the potential to be better. In section 3.2 the idea of the *BSPSWEEP* tree is explained. The *BSPSWEEP* tree is a general *BSP* tree in which k directions are determined in each node and all  $2n$  splitting planes

---

along each of these directions are checked by sweeping. These  $k$  directions can be different for every node and can be chosen dependant on the local geometry. The  $RBS\!P$  tree can now be seen as a  $BSP_{SWEEP}$  tree where the same  $k$  directions are chosen in every node.

Three versions of the  $BSP_{SWEEP}$  tree are designed:

- $BSP_{random}$  generates  $k$  random directions in each node. This tree doesn't make use of the possibility to use the local geometry to determine the optimal split directions and is therefore used as a baseline for  $BSP_{SWEEP}$  performance.
- $BSP_{wn}$  chooses the normals of  $k$  arbitrary triangles in the node as split directions.
- $BSP_{cn}$  calculates a K-means clustering of the normals in the node and uses the cluster centra as split directions.

Each of these versions has three variants:

- $BSP_{SWEEP}$ : Calculates  $k$  directions in each node,
- $BSP_{SWEEP+}$ : Calculates  $k - 3$  directions and uses the x-, y- and z-axis as other split axes
- $BSP_{SWEEP+}^{Kd}$ : Use the same split planes as  $BSP_{SWEEP+}$  but optimises and favors the use of  $Kd$  planes

Table 3.1 shows the same statistics as table 2.1 but now for the three types of  $BSP_{SWEEP}$  trees. It shows that the  $BSP_{SWEEP}$  trees use a total of  $O(n \log n)$  distinct split planes instead of  $O(n)$  for the  $Kd$  tree. This follows from the fact that the  $BSP_{SWEEP}$  trees use different split planes in each level of the tree.

The forth chapter explains the implementation details of all the  $BSP$  trees used in the comparisons. It shows that the general outlines of the build and intersect algorithms for each of these trees are the same and that they only differ in the implementation of some helper methods. It also explains the data structure of the nodes for each type of  $BSP$  tree and shows that the nodes of the general  $BSP$  trees need 2.5 times as much memory. This is due to the fact that they need to store three floating point values to identify the split axis.

The fifth chapter discusses the results gathered by rendering the four scenes in figure 5.1 with different configurations of the  $BSP_{SWEEP}$  trees. Section 5.2 shows the results for the 9  $BSP_{SWEEP}$  trees for increasing values of  $k$ . All plots are relative to the result of the  $Kd$  tree. There are a lot of conclusions to draw from the plots, but the most important one is that the versions without the  $Kd$  directions can't compete with the  $Kd$  tree, that the versions with the  $Kd$  directions have at least one  $k$  value for which the render time is lower than that of the  $Kd$  tree and the versions with the optimisation for  $Kd$  directions clearly beat the  $Kd$  tree in terms of render time for all values of  $k$ . It can also be noted that the  $BSP_{random}$  trees improve with

---

increasing  $k$  and that the other two versions seem to converge to an optimal solution for  $k$  equal to 6 or 7.

Section 5.3 compares the  $BSP_{SWEEP}$  trees with existing  $BSP$  trees. Table 5.4 shows for every scene the render times (R) and the build times (B) relative to that of the  $Kd$  tree. The three  $BSP_{SWEEP}$  trees do always have a better render time than the  $Kd$  tree. It is also worth noting that the  $RBSP^{Kd}$  tree - also a type of  $BSP_{SWEEP}$  tree - also outperforms the  $Kd$  tree. The build times of the  $BSP_{SWEEP}$  trees are also much better than the build times of the only other general  $BSP$  tree, the  $BSP_{SIZE}$  tree. The main reason for this is the reduction of ray-triangle intersections as shown by table 5.5 where ZI represent the intersections with primary rays and SI the intersection with secondary rays. Tables 5.7 and 5.8 show false color images for the amount of ray-triangle intersections for each pixel. In the first table, all trees use the same color scale. In the second table, they all use different color scales. The  $BSP_{SWEEP}$  trees are clearly much better at adapting to locale geometry and do therefore have less hotspot regions.

Table 5.6 shows that the amount of node traversals does also decrease when using the  $BSP_{SWEEP}$  trees. ZD represent the traversals caused by primary rays and SD the traversals caused by secondary rays. It is important to note that the traversals through a general  $BSP$  are computationally more expensive than the traversals through a  $Kd$  tree. This implies that the  $BSP_{SWEEP}$  trees might still spend more time traversing nodes than the  $Kd$  tree despite the decrease in node traversals. In the  $BSP_{SWEEP+}^{Kd}$  optimised trees, this problem is reduced a lot by favoring  $Kd$  nodes and traversing them with a different traversing algorithm. Table 5.9 shows the proportion of nodes that are  $Kd$  node (the  $Kd$  column), the proportion of primary ray traversals (ZD  $Kd$ ) that use  $Kd$  nodes and the proportion of secondary ray traversals (SD  $Kd$ ) that use  $Kd$  nodes. The table shows that almost all traversals can use the fast  $Kd$  node traversal algorithm even though less than half of the nodes are  $Kd$  nodes. The reason for this is that  $Kd$  nodes are used mainly in the top levels while the  $BSP$  nodes are mostly used in the lower levels of the tree. This can be seen in the heatmaps in figure 5.6 which show the proportion of  $Kd$  nodes for each depth, for multiple values of  $k$ .

# Bibliography

- [App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM, 1968.
- [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.
- [Hoo19] Jesse Hoobergs. Pbrt v3 met *BSPSWEEP* bomen. <https://github.com/jhoobergs/Thesis-pbrt-v3>, 2019.
- [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.
- [Kaj86] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986.
- [KCK02] Deok-Soo Kim, Youngsong Cho, and Donguk Kim. The compression of the normal vectors of 3d mesh models using clustering. In *International Conference on Computational Science*, pages 275–284. Springer, 2002.
- [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.
- [Suf07] Kevin Suffern. *Ray Tracing from the Ground Up*. A. K. Peters, Ltd., Natick, MA, USA, 2007.
- [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

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

*UDC:* 681.3

*Korte inhoud:*

In deze masterproef introduceren we een nieuw soort algemene Binary Space Partitioning (*BSP*) boom: de *BSP<sub>SWEET</sub>* boom. De *BSP<sub>SWEET</sub>* boom laat toe om in elke knoop een aantal richtingen te bepalen afhankelijk van de driehoeken in die knoop en het beste vlak van alle vlakken met als normaal één van die richtingen, wordt gebruikt om de knoop te splitsen. Drie soorten *BSP<sub>SWEET</sub>* boom zijn ontworpen: één soort die in elke knoop random richtingen genereert (*BSP<sub>random</sub>*) en twee soorten die richtingen genereren afhankelijk van de normalen van de driehoeken in de knoop (*BSP<sub>wn</sub>* en *BSP<sub>cn</sub>*). De *BSP<sub>wn</sub>* boom kiest de normalen van enkele willekeurige driehoeken in de knoop als richtingen en de *BSP<sub>cn</sub>* boom bepaalt een clustering van de normalen van de driehoeken in de knoop en gebruikt de centra van deze clusters als richtingen. Er is ook een optimalisatie van de *BSP<sub>SWEET</sub>* boom uitgewerkt: de *BSP<sub>SWEET+</sub><sup>Kd</sup>* boom die asgealigneerde splitsingsvlakken bevoordeeld omdat ze computationele voordelen hebben.

De drie *BSP<sub>SWEET+</sub><sup>Kd</sup>* soorten zijn duidelijk beter dan de bestaande *BSP* bomen. Van de drie soorten *BSP<sub>SWEET+</sub><sup>Kd</sup>* bomen zijn de twee soorten die gebruik maken van de normalen, beduidend beter dan de soort die random richtingen bepaalt. Ze verminderen bij alle testscenes het aantal straal-driehoekintersecties met meer dan 40% en de rendertijd met meer dan 20% ten opzichte van de *Kd* boom. De bouwtijd is twee ordegroottes groter dan die van de *Kd* boom, maar wel kleiner dan die van elke andere *BSP* boom.

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. Vandebriel

*Begeleider:* Ir. P. Bartels