

# 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. Vandebril

**Begeleiders:**

Ir. M. Moulin

Ir. P. Bartels

© Copyright KU Leuven

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

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

# Voorwoord

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

*Jesse Hoobergs*

# Inhoudsopgave

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

4.6	$BSP_{IZE}^{Kd}$ boom	31
4.7	$BSP_{SWEEP}$ boom	33
<b>5</b>	<b>Resultaten</b>	<b>39</b>
5.1	Praktische aspecten	44
5.2	Afhankelijkheid van aantal richtingen	44
5.3	Vergelijking	53
<b>6</b>	<b>Besluit</b>	<b>55</b>
	<b>Bibliografie</b>	<b>57</b>

# Samenvatting

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

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

# Lijst van figuren

2.1	Visuele voorstelling van <i>ray tracing</i> . . . . .	4
2.2	Splitsingskracht <i>BSP</i> boom . . . . .	5
2.3	Splitsingskracht <i>Kd</i> boom . . . . .	6
2.4	Splitsingskracht <i>RBSP</i> boom . . . . .	8
2.5	Splitsingsvlakken <i>Kd</i> , <i>RBSP</i> en <i>BSP<sub>IZE</sub></i> . . . . .	11
3.1	Invloed grootte bladknopen op aantal intersecties . . . . .	15
3.2	<i>False color</i> afbeeldingen van de Killeroo en Killeroo Been scene . . . . .	16
3.3	Splitsingsvlakken <i>BSP<sub>random</sub></i> . . . . .	18
3.4	Splitsingsvlakken <i>BSP<sub>random+</sub><sup>(Kd)</sup></i> . . . . .	18
3.5	Splitsingsvlakken <i>BSP<sub>wn</sub></i> . . . . .	19

# Lijst van tabellen

2.1	Vergelijking van de bestaande soorten <i>BSP</i> bomen. . . . .	11
3.1	Vergelijking van de nieuwe soorten <i>BSP</i> bomen. . . . .	20
4.1	Voorstelling <i>Kd</i> knoop . . . . .	25
4.2	Voorstelling <i>RBSP</i> knoop . . . . .	27
4.3	Voorstelling <i>BSP</i> knoop . . . . .	29
4.4	Voorstelling <i>BSP<sup>Kd</sup></i> knoop . . . . .	32
5.1	Rendertijd <i>BSP<sub>random</sub></i> in functie van <i>K</i> . . . . .	39
5.2	Rendertijd <i>BSP<sub>random+</sub></i> in functie van <i>K</i> . . . . .	40
5.3	Rendertijd <i>BSP<sub>random+</sub><sup>Kd</sup></i> in functie van <i>K</i> . . . . .	40
5.4	Rendertijd <i>BSP<sub>wn</sub></i> in functie van <i>K</i> . . . . .	41
5.5	Rendertijd <i>BSP<sub>wn+</sub></i> in functie van <i>K</i> . . . . .	41
5.6	Rendertijd <i>BSP<sub>wn+</sub><sup>Kd</sup></i> in functie van <i>K</i> . . . . .	42
5.7	Rendertijd <i>BSP<sub>cn</sub></i> in functie van <i>K</i> . . . . .	42
5.8	Rendertijd <i>BSP<sub>cn+</sub></i> in functie van <i>K</i> . . . . .	43
5.9	Rendertijd <i>BSP<sub>cn+</sub><sup>Kd</sup></i> in functie van <i>K</i> . . . . .	43
5.10	. . . . .	53



# Lijst van algoritmes

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

# Lijst van afkortingen en symbolen

## Afkortingen

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

---

## Symbolen

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



# Hoofdstuk 1

## Inleiding

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

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

### 1.1 Ray tracing

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

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

### 1.2 Doelstelling

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

### 1.3 Methodologie

### 1.4 Contributie

### 1.5 Overzicht



## Hoofdstuk 2

# Voorgaand werk

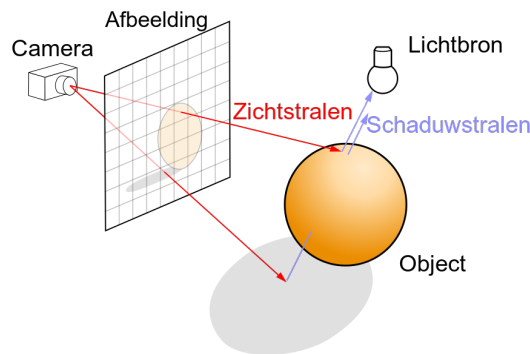
Dit hoofdstuk start met een korte uitleg over enkele basisconcepten: *ray tracing* en acceleratiestructuren. *Ray tracing* vereist acceleratiestructuren om efficiënt driehoeken in de scene te kunnen zoeken. Daarna wijdt dit hoofdstuk uit over één specifieke acceleratiestructuur: de *Binary Space Partitioning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

### 2.1 Basisconcepten

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

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

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



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

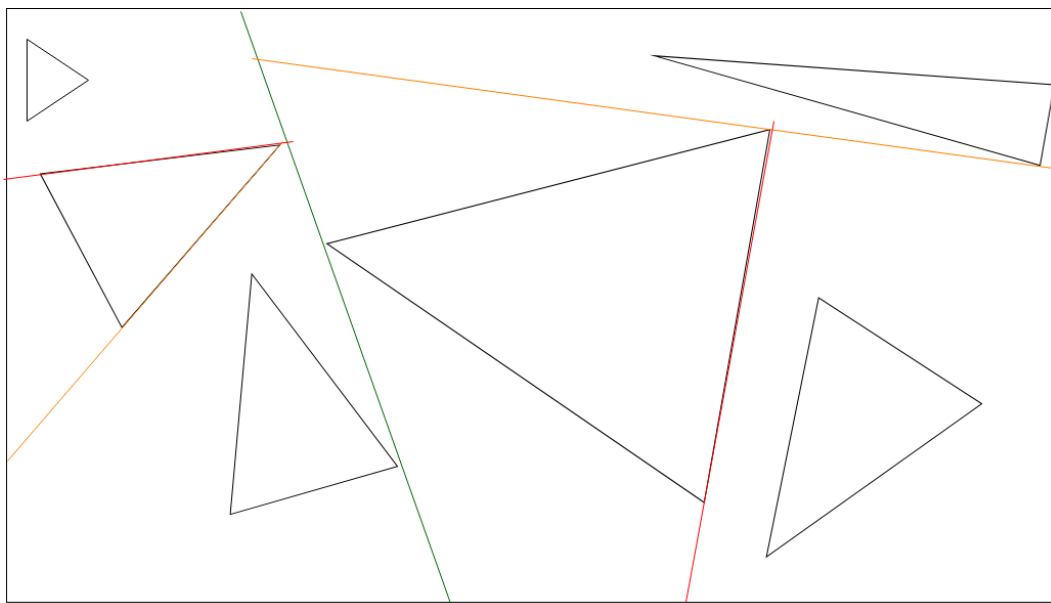
het omhullend volume van de scene. Testen op intersectie met de driehoeken in de scene gebeurt dan enkel als dit omhullend volume intersekteert met de straal. Deze acceleratiestructuur kan worden uitgebreid tot een boomstructuur door dit volume recursief op te delen in kindvolumes. Binaire bomen delen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

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

## 2.2 *BSP* bomen

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



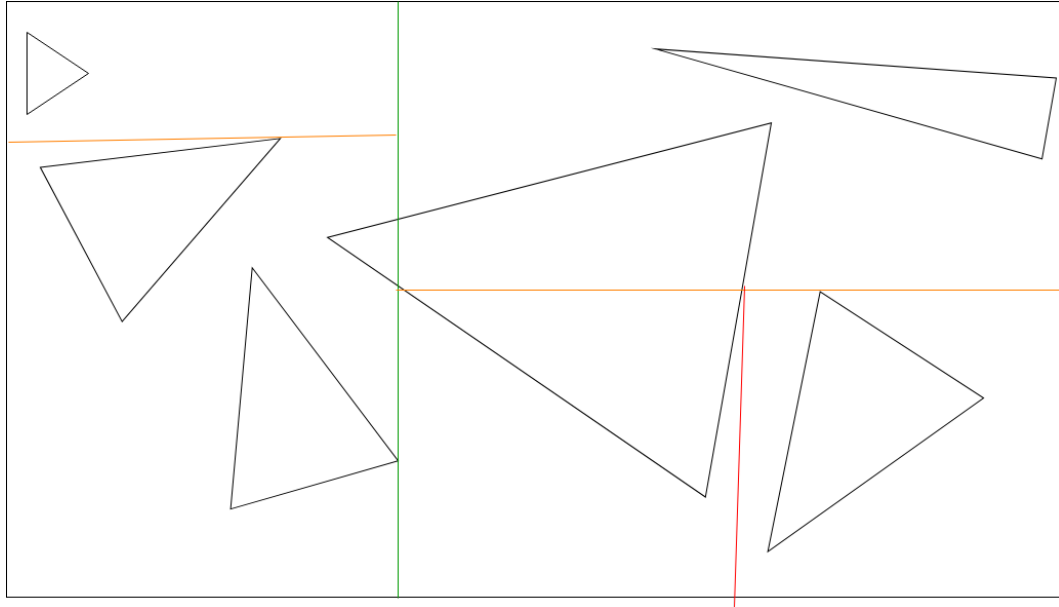


Figuur 2.2: Splittingskracht *BSP* boom - Een 2D voorbeeld dat toont dat alle (niet-intersecterende) driehoeken gesplitst kunnen worden door een *BSP* boom.

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

## 2.3 *Kd* Bomen

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



Figuur 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 links onder (en rechts boven) kunnen niet gesplitst worden omdat zowel hun projecties op de x-as, als hun projecties op de y-as overlappen.

boom niet alle driehoeken kan scheiden.

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

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

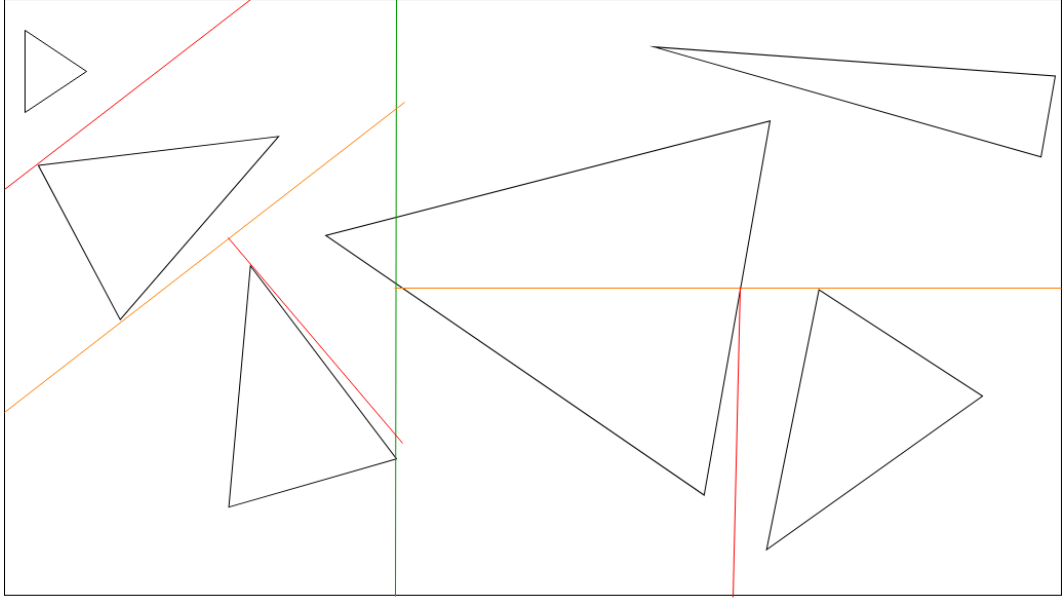
voor respectievelijk intersectie en doorkruising. De kost van het beste splitsingsvlak (laagste kost) wordt dan vergeleken met de kost voor de knoop als die niet gesplitst zou worden:  $n * \mathcal{K}_i$ . Als splitsen voordelig is, wordt de knoop opgesplitst volgens dit splitsingsvlak, anders wordt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

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

## 2.4 *RBSP* bomen

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

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



Figuur 2.4: Splitsingskracht *RBSP* boom - Een 2D voorbeeld dat toont dat niet alle (niet-intersecterende) 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 waarlangs deze driehoeken gesplitst kunnen worden.

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

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

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

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

## 2.5 Algemene *BSP* Bomen in de praktijk

Ize et al [IWP08] ontwikkelden de eerste algemene *BSP* boom voor *ray tracing*. In tegenstelling tot de *Kd* en *RBSP* boom kan  $BSP_{IZE}$  wel splitsingsvlakken kiezen afhankelijk van de geometrie. De  $BSP_{IZE}$  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 *RBSP* boom. Het omhullend volume van de knopen bij een *BSP* boom is een convex veelvlak. Voor de eenvoud wordt de omhullende balk gebruikt als omhullend volume voor de 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  $BSP_{IZE}$  boom controleert de  $6n$  *Kd* splitsingsvlakken door te sweeppen zoals bij de *Kd* en *RBSP* boom. Het splitsen in kindknopen en de *SA* berekening is duurder aangezien het omhullend volume een convex veelvlak is. De  $BSP_{IZE}$  boom controleert ook vier extra vlakken per driehoek. Voor deze vlakken is het berekenen van de *SAH* kost moeilijker omdat het aantal driehoeken links en rechts van het vlak bepaald moet worden. Om dit efficiënt te bepalen, gebruiken Ize et al [IWP08] de *BVH* als hulpstructuur. In elke knoop wordt een *BVH* boom gebouwd en die wordt bij elke niet-*Kd* splitsing gebruikt om het aantal driehoeken in beide kindknopen te bepalen. Hierdoor is het bouwen van de  $BSP_{IZE}$  boom computationeel beduidend duurder dan het bouwen van *RBSP* 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  $BSP_{IZE}^{Kd}$ . Het feit dat *Kd* knopen goedkoper zijn om te doorkruisen dan *BSP* knopen, zorgt ervoor dat er voor deze twee soorten knopen een aparte doorkruiskost gebruikt moet worden in de *SAH*:  $\mathcal{K}_{d,BSP}$  en  $\mathcal{K}_{d,Kd}$ . Deze kosten rechtstreeks in de *SAH* gebruiken, zorgt ervoor dat voornamelijk *BSP* knopen gecreëerd worden. De reden hiervoor is dat de intersectieterm van

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

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

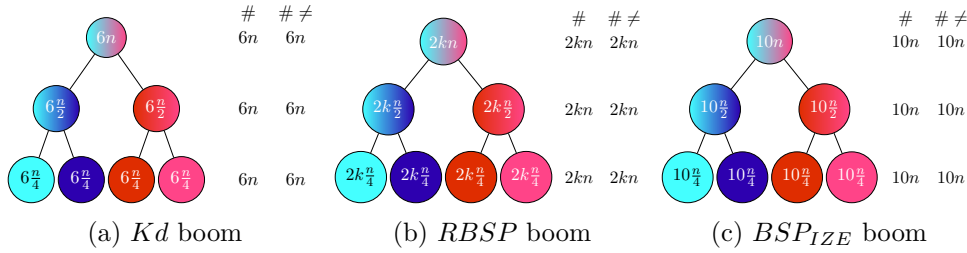
## 2.6 Vergelijking

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

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

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

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



Figuur 2.5: Splitsingsvlakken  $Kd$ ,  $RBSP$  en  $BSP_{IZE}$  - Per niveau het aantal (#) splitsingsvlakken en het totaal aantal verschillende (#  $\neq$ ) splitsingsvlakken gebruikt in bovenliggende niveaus.

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

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





# Hoofdstuk 3

## BSP<sub>SWEEP</sub>

In dit hoofdstuk wordt een nieuwe soort *BSP* boom besproken: de *BSP<sub>SWEEP</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>SWEEP</sub>* boom wordt dan besproken en daarna worden een aantal specifieke versies van de *BSP<sub>SWEEP</sub>* (*BSP<sub>random</sub>*, *BSP<sub>wn</sub>* en *BSP<sub>cn</sub>*) besproken.

### 3.1 Probleemstelling

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

#### 3.1.1 Wiskundige basis

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

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

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

---

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

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

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

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

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. Analooog kan worden ingezien dat de maximale waarde voor deze som gelijk is aan  $2D_{b_j}$  aangezien elke doorkruising van  $b_j$  voor maximaal twee doorkruisingen door een kindknoop kan zorgen. Hieruit volgt ongelijkheid 3.5.

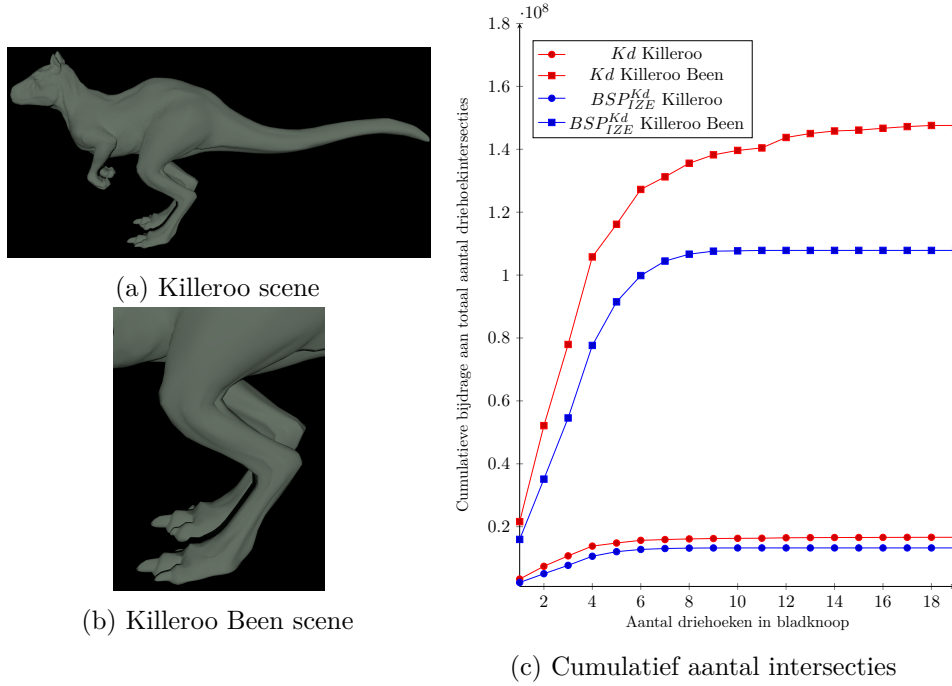
$$D_{b_j} \leq D_{b_{j1}} + D_{b_{j2}} \leq 2D_{b_j} \quad (3.5)$$

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

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

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

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

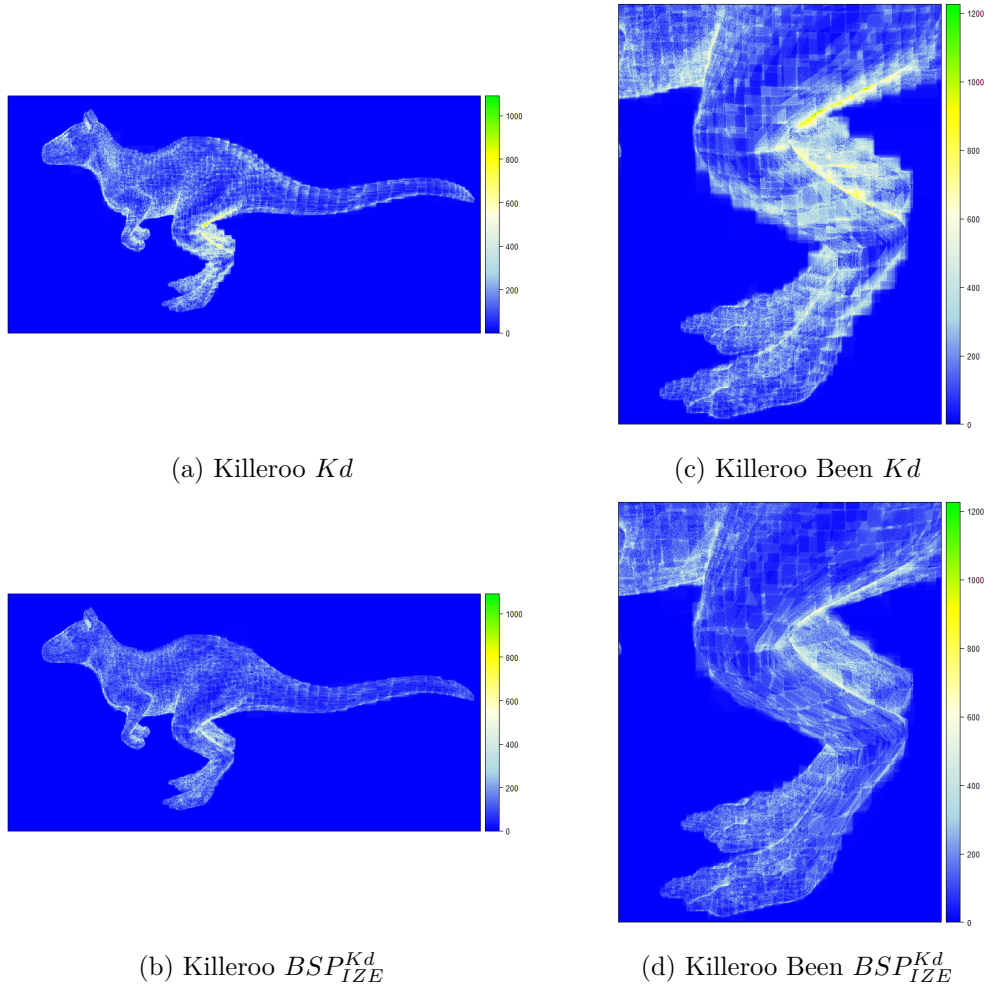


Figuur 3.1: Invloed grootte bladknopen op aantal intersecties - (a) toont de Killeroo scene, (b) toont de scene waarbij ingezoomd is op de benen van de Killeroo, (c) toont het totaal aantal intersecties als een cumulatieve som over het aantal driehoeken in de bladknoop. Dit betekent dat de waarde bij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

### 3.1.2 Voorbeeld

Figuur 3.1a toont de Killeroo scene die als voorbeeld scene dient bij de pbrt *ray tracer*. Voor deze 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  $BSP$  bomen nuttig kunnen zijn omdat ze minder tot geen last hebben van regio's die ze minder goed aankunnen. Het aantal knoopdoorkruisingen bij de  $BSP_{IZE}^{Kd}$  boom is zelfs lager dan bij de  $Kd$  boom voor beide scenes. Dit komt omdat de  $BSP$  knopen nauwer aansluiten aan het object waardoor minder stralen de knopen raken.

Figuur 3.2 toont *false color* afbeeldingen van het aantal zichtstraalintersecties van de Killeroo en Killeroo been scene voor zowel de  $Kd$  boom als de  $BSP_{IZE}^{Kd}$  boom. Voor de Killeroo scene zijn de *false color* afbeeldingen van beide bomen zeer



Figuur 3.2: *False color* afbeeldingen van de Killeroo en Killeroo Been scene - Deze afbeeldingen tonen dat de  $Kd$  boom slecht presteert bij sommige delen van een scene, 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 scene.

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 scene 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 scene dan de  $Kd$  boom. Merk op dat, zoals besproken in het vorige hoofdstuk, de  $BSP_{IZE}^{Kd}$  boom nog niet alle vrijheden van een algemene  $BSP$  boom gebruikt.

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

### 3.2 Basisidee

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

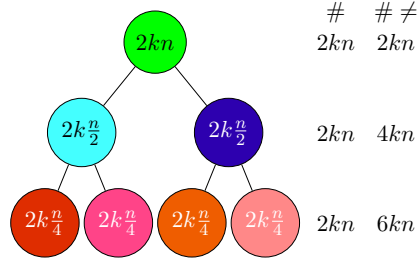
### 3.3 Gebaseerd op random richtingen

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

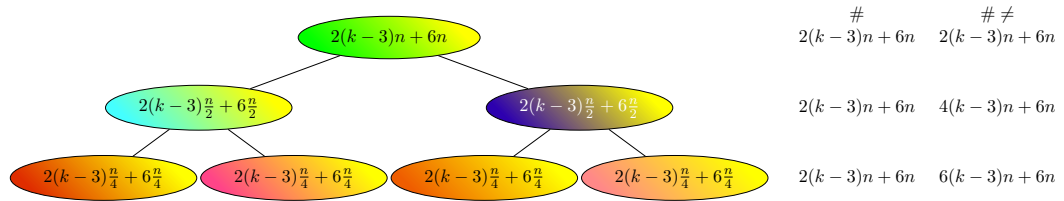
Als de  $BSP_{random}$  boom perfect gebalanceerd is, worden in elk niveau  $2kn$  verschillende splitsingsvlakken bekeken. Deze splitsingsvlakken zijn verschillend op elk niveau, zodat in totaal  $2kn \log(n)$  verschillende splitsingsvlakken bekeken worden. Figuur 3.3 toont dit visueel. De  $BSP_{random}$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

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

### 3. $BSP_{SWEEP}$



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



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

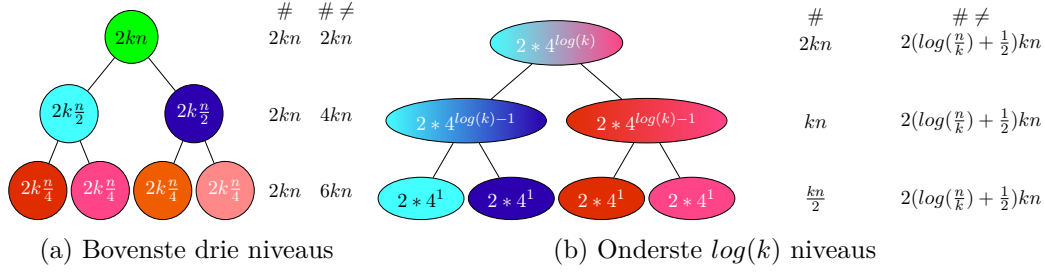
Dit zorgt voor  $2(k-3)n$  verschillende splitsingsvlakken per niveau, zodat in totaal  $2(k-3)n \log(n) + 6n$  verschillende splitsingsvlakken bekeken worden. Figuur 3.4 toont dit visueel. De  $BSP_{random+}^{(Kd)}$  boom probeert elke driehoek via gemiddeld  $\mathcal{O}(\log(n))$  vlakken te splitsen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

## 3.4 Gebaseerd op normalen

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

### 3.4.1 Willekeurige normaal

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



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

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

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

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

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

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

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

#### 3.4.2 Geclusterde normalen

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

### 3.5 Vergelijking

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 de  $SAH$  kosten te kunnen berekenen, zonder hulpstructuren. De  $BSP_{random}$  boom is onafhankelijk van de lokale geometrie, de twee andere bomen bepalen hun splitsingsvlakken afhankelijk van de lokale geometrie. De  $BSP_{SWEEP}$  bomen zijn makkelijk uit te breiden naar  $BSP_{SWEEP}^{Kd}$  bomen die de optimalisatie met de snelle  $Kd$  knoop doorkruising 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.



# Hoofdstuk 4

## Implementatie

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

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

### 4.1 Hoog niveau beschrijving

#### 4.1.1 Bouwalgoritme

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

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

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

**Algoritme 1** Bouwen van een BSP boom

---

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

---

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

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

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

#### 4.1.2 Intersectie-algoritme

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

Het algoritme maakt gebruik van de volgende functies:

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

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

**Algoritme 2** Intersecteren van een BSP boom

---

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

---

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

Tabel 4.1: Voorstelling *Kd* knoop -

## 4.2 *Kd* boom

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

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

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

---

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

---

```

function DOORKRUIS_INWENDIGE_KNOOP(Kd Knoop k, Straal s)
     $as < -k_{splitsAs}$ 
     $tVlak \leftarrow KD\_VLAK\_AFSTAND(k_{splitPos}, s, \frac{1}{s_d}, as)$ 
     $linksEerst \leftarrow \vec{s_o}[as] < k_{splitPos}$  or  $(\vec{s_o}[as] = k_{splitPos}$  and  $s_d[as] \leq 0)$ 
    return  $tVlak, linksEerst$ 
end function

```

---

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

---

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

---

```

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

```

---

elke straal slechts één keer berekend worden.

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

---

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

---

```

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

```

---

Inwendig	bits	Blad	bits
tSplit	32	primitief Offset	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

Tabel 4.2: Voorstelling *RBSP* knoop -

### 4.3 *RBSP* boom

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

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

Het algoritme om het beste splitsingsvlak te bepalen bij een *RBSP* boom is zeer gelijkaardig aan het algoritme voor de *Kd* boom. In plaats van te sweeppen over de drie assen, wordt er gesweept over alle  $k$  richtingen. Het bepalen van de linker- en rechterrands 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

---

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

---

```

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

```

---



---

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

---

```

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

```

---

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

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

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

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



---

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

---

```

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

```

---

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

Tabel 4.3: Voorstelling  $BSP$  knoop -

## 4.5 $BSP_{IZE}$ boom

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

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

---

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

---

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

---

gebruikt, net als de  $RBSP$  knoop doorkruising, de functie `VLAKE_INTERSECTIE` (algoritme 7) om te intersecteren met een willekeurig geörienteerd vlak.

---

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

---

```

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

```

---

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

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

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

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

---

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

---

```

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

```

---

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

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

---

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

---

```

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

```

---

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

## 4.7 $BSP_{SWEEP}$ boom

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

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

---

**Algoritme 13** Beste split voor een bouwknoop  $b$  bij een  $BSP_{I\mathcal{Z}E}$  boom.

---

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

---

---

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

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
   $besteAs, besteT, besteKost \leftarrow None, None, \infty$ 
   $richtingen \leftarrow BEPAAL\_RICHTINGEN(bouwKnoop, k)$ 
  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  $besteAs, besteT, besteKost$ 
end function

```

---

**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
     $\phi \leftarrow \text{UNIFORM}(0, 2\pi)$ 
     $\theta \leftarrow \text{acos}(\text{UNIFORM}(-1, 1))$ 
     $richting \leftarrow \{\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta)\}$ 
     $richting \leftarrow \text{UPDATE\_ZIN}(richting)$ 
     $\text{ADD}(richtingen, richting)$ 
  end for
  return  $richtingen$ 
end function

```

---

**$BSP_{wn}$  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)
   $richtingen \leftarrow \emptyset$ 
  while  $richtingen_{lengte} < \text{MIN}(k-1, b_n-1)$  do
     $richting \leftarrow b_{primitieven}[\text{RANDOM\_INT}(0, b_n)]_{normaal}$ 
     $richting \leftarrow \text{UPDATE\_ZIN}(richting)$ 
     $\text{ADD}(richtingen, richting)$ 
  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.

---

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

---

```

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

```

---

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



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

---

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

---

```

function BEPAAL_BESTE_SPLIT(bouwKnoop)
   $besteAs, besteT, besteKost \leftarrow None, None, \infty$ 
   $besteAsVast, besteTVast, besteKostVast \leftarrow None, None, \infty$ 
   $richtingen \leftarrow BEPAAL\_RICHTINGEN(b)$ 
  for  $richting \in richtingen$  do
    Sorteer eindpunten volgens de richting
    if  $richting_{isKdRichting}$  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
        Bereken de  $SAH$  kost met een vaste  $\mathcal{K}_{d,BSP}$  als doorkruiskost
        Update  $besteAsVast, besteTVast, besteKostVast$  als dit splitsings-
        vlak beter is dan het huidige beste splitsingsvlak met vaste  $BSP$  doorkruiskost.
      end for
    end if
  end for
  if  $besteKost < bouwknop_n * \mathcal{K}_i$  then
    return  $besteAs, besteT, besteKost$ 
  end if
  if  $besteKostVast < bouwknop_n * \mathcal{K}_i$  then
    return  $besteAsVast, besteTVast, besteKostVast$ 
  end if
  return  $None, None, \infty$ 
end function

```

---



# Hoofdstuk 5

## Resultaten

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

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

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
2	94.263	95.144	95.837	454.741	437.608	454.138	298.405	300.392	310.614
3	88.062	89.091	89.322	364.787	345.522	368.295	261.261	266.944	268.196
4	85.441	85.581	85.62	316.678	294.67	305.886	225.13	232.307	239.816
5	83.558	83.651	84.418	289.664	272.468	278.195	211.855	214.093	216.993
6	81.93	82.325	82.287	255.474	248.777	263.412	200.431	204.593	205.686
7	81.117	81.057	81.357	249.249	231.221	232.563	193.014	194.539	194.655
8	79.914	80.121	80.089	235.069	223.647	223.241	183.805	184.94	184.278
9	79.39	79.174	79.477	224.679	208.786	212.783	177.842	178.935	178.442
10	78.257	78.481	78.198	219.107	198.755	199.18	171.722	173.994	178.498

Tabel 5.1: Rendertijd  $BSP_{random}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{random}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de *Kd* boom.

## 5. RESULTATEN

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	102.896	102.681	102.697	100.902	91.383	89.77	97.219	97.129	102.432
4	90.292	90.191	89.904	102.357	95.368	100.245	95.658	96.988	97.714
5	85.014	85.512	87.979	100.715	94.368	98.783	93.773	94.099	99.636
6	82.523	82.349	86.901	100.973	97.991	99.783	91.391	93.91	98.434
7	80.748	80.892	80.888	104.466	96.864	101.065	90.272	93.843	94.656
8	79.55	79.911	79.513	105.823	97.068	99.078	92.536	92.504	92.933
9	78.837	79.001	78.943	101.183	94.022	97.615	92.26	92.212	93.991
10	78.225	78.24	78.501	103.721	95.61	98.762	91.082	92.371	92.354

Tabel 5.2: Rendertijd  $BSP_{random+}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{random+}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de  $Kd$  boom.

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	101.563	101.346	101.579	105.439	95.087	99.882	98.029	97.286	100.595
4	91.987	92.271	95.524	100.225	90.867	92.592	92.05	92.736	93.297
5	88.29	88.335	88.449	97.599	90.537	90.244	90.222	91.448	92.026
6	84.518	84.519	84.819	95.31	88.422	88.965	88.878	90.463	95.608
7	82.944	82.626	82.843	94.765	85.315	85.011	89.03	89.65	93.787
8	81.29	81.566	83.309	93.903	85.485	88.245	87.747	90.072	94.933
9	80.581	80.799	81.702	93.203	84.301	83.876	87.304	88.272	89.006
10	80.028	80.234	79.716	93.666	85.503	88.406	86.135	87.09	90.935

Tabel 5.3: Rendertijd  $BSP_{random+}^{Kd}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{random+}^{Kd}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de  $Kd$  boom.

---

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
2	78.381	80.439	80.06	189.64	205.401	299.886	143.656	163.78	352.315
3	75.41	76.171	77.975	166.622	165.521	193.158	127.188	133.497	170.861
4	74.31	74.369	74.782	157.165	149.284	154.059	115.946	121.746	128.461
5	73.644	74.334	74.324	151.537	153.732	157.101	112.462	120.877	134.361
6	73.125	73.867	76.303	148.679	137.888	162.36	109.783	113.731	124.092
7	72.715	74.099	74.861	143.61	143.17	150.795	107.986	111.324	114.767
8	72.301	73.992	73.77	144.221	134.784	142.216	108.338	110.33	115.036
9	72.375	72.973	74.384	135.611	130.295	141.337	108.666	109.587	119.766
10	71.807	72.579	73.052	137.872	134.463	134.933	107.595	110.402	112.553

---

Tabel 5.4: Rendertijd  $BSP_{wn}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{wn}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de *Kd* boom.

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	102.555	102.405	103.09	104.655	96.063	100.144	97.549	97.503	97.794
4	70.873	70.982	70.946	96.228	94.165	95.941	81.85	82.461	86.085
5	67.85	68.482	68.37	94.684	92.765	99.968	84.101	83.239	87.39
6	67.419	67.283	68.196	101.928	98.831	107.121	82.223	83.912	87.486
7	66.999	67.177	67.03	96.528	100.152	113.112	83.9	84.41	88.507
8	66.752	66.861	66.809	106.938	102.555	110.824	83.37	84.228	88.147
9	66.831	66.729	66.616	107.749	104.674	107.536	83.752	84.364	88.829
10	66.414	66.491	66.772	107.362	100.657	110.953	83.12	85.119	89.107

---

Tabel 5.5: Rendertijd  $BSP_{wn+}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{wn+}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de *Kd* boom.

## 5. RESULTATEN

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	101.652	101.544	101.482	105.459	95.15	98.877	97.689	97.169	96.844
4	71.907	72.068	71.865	91.903	83.277	88.974	79.777	79.16	87.162
5	68.89	69.158	69.3	90.929	83.87	82.445	79.208	78.864	86.424
6	67.788	67.723	67.484	88.438	80.364	79.45	77.673	78.562	83.647
7	67.372	67.182	67.237	88.779	80.472	80.619	78.101	80.896	86.181
8	67.092	67.012	67.01	89.004	80.036	78.671	78.746	80.682	80.576
9	67.163	67.043	66.863	88.714	80.296	80.374	78.44	80.007	86.294
10	67.15	67.126	66.711	90.26	83.683	86.547	78.839	80.188	80.407

Tabel 5.6: Rendertijd  $BSP_{wn+}^{Kd}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{wn+}^{Kd}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de  $Kd$  boom.

K	Been			Sponza			Conference Hall		
	Min	Med	Max	Min	Med	Max	Min	Med	Max
2	101.213	102.797	104.333	370.413	370.346	406.212	176.209	178.901	186.517
3	79.02	79.765	81.037	232.447	212.866	224.591	138.359	141.885	146.04
4	74.735	75.708	76.117	200.78	187.343	185.482	128.482	132.851	134.903
5	72.416	74.268	75.84	173.423	163.685	169.425	125.084	126.273	133.541
6	72.482	72.469	73.301	159.539	152.916	166.353	116.994	120.382	130.53
7	71.614	71.852	72.965	151.17	145.378	153.251	113.379	117.766	120.444
8	72.276	72.52	72.391	150.888	146.582	158.044	112.247	116.81	134.788
9	71.515	71.992	73.25	147.545	142.395	153.013	112.446	114.814	121.813
10	71.857	72.229	72.821	153.516	144.594	150.238	104.583	114.291	118.389

Tabel 5.7: Rendertijd  $BSP_{cn}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{cn}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de  $Kd$  boom.

---

	Been			Sponza			Conference Hall		
K	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	102.827	102.606	102.73	104.446	102.09	99.832	97.57	97.246	98.053
4	77.713	77.536	77.276	101.442	99.184	97.47	85.942	86.035	91.19
5	70.206	70.343	70.382	101.233	93.449	105.114	80.749	83.323	87.966
6	67.618	67.694	67.306	110.861	105.866	115.97	84.898	89.229	90.635
7	66.694	66.572	66.722	122.426	114.48	123.15	84.767	88.392	90.639
8	66.655	66.451	66.718	117.905	110.056	116.398	84.719	86.746	88.584
9	66.618	66.879	66.583	112.357	110.56	114.611	83.749	87.156	88.896
10	66.52	66.411	66.117	117.148	111.116	111.114	87.258	89.005	94.316

---

Tabel 5.8: Rendertijd  $BSP_{cn+}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{cn+}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de *Kd* boom.

---

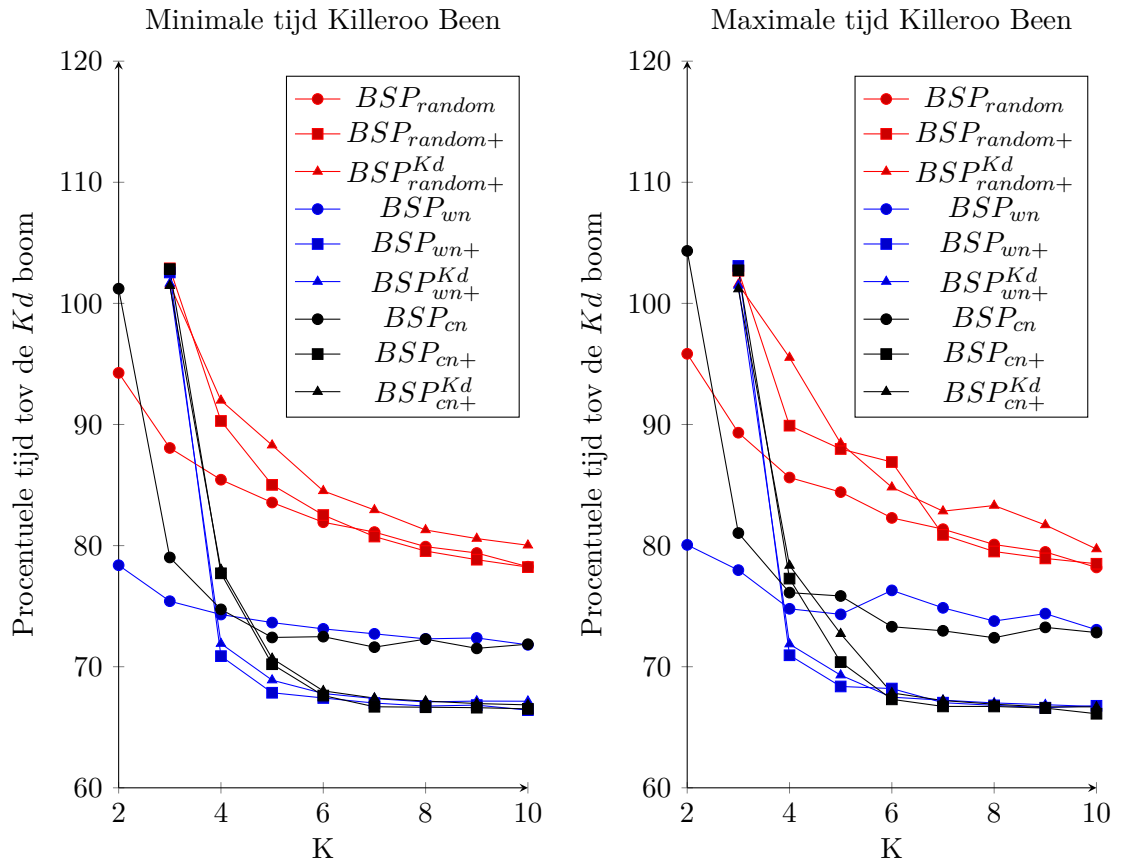
	Been			Sponza			Conference Hall		
K	Min	Med	Max	Min	Med	Max	Min	Med	Max
3	101.454	101.571	101.18	105.169	95.846	99.905	96.645	96.491	95.757
4	78.023	78.065	78.356	96.472	87.161	94.116	84.088	83.582	91.343
5	70.681	70.429	72.718	91.49	83.064	85.978	79.583	80.756	88.038
6	68.006	67.828	67.837	89.127	80.339	80.096	78.349	79.272	86.204
7	67.408	67.208	67.208	88.377	79.844	78.395	79.853	79.577	85.83
8	67.165	67.291	66.905	88.68	80.107	79.958	79.798	79.712	86.478
9	66.958	66.847	66.683	89.005	80.173	79.666	77.809	78.885	79.792
10	66.849	66.899	66.686	89.448	83.764	83.188	79.45	80.151	87.728

---

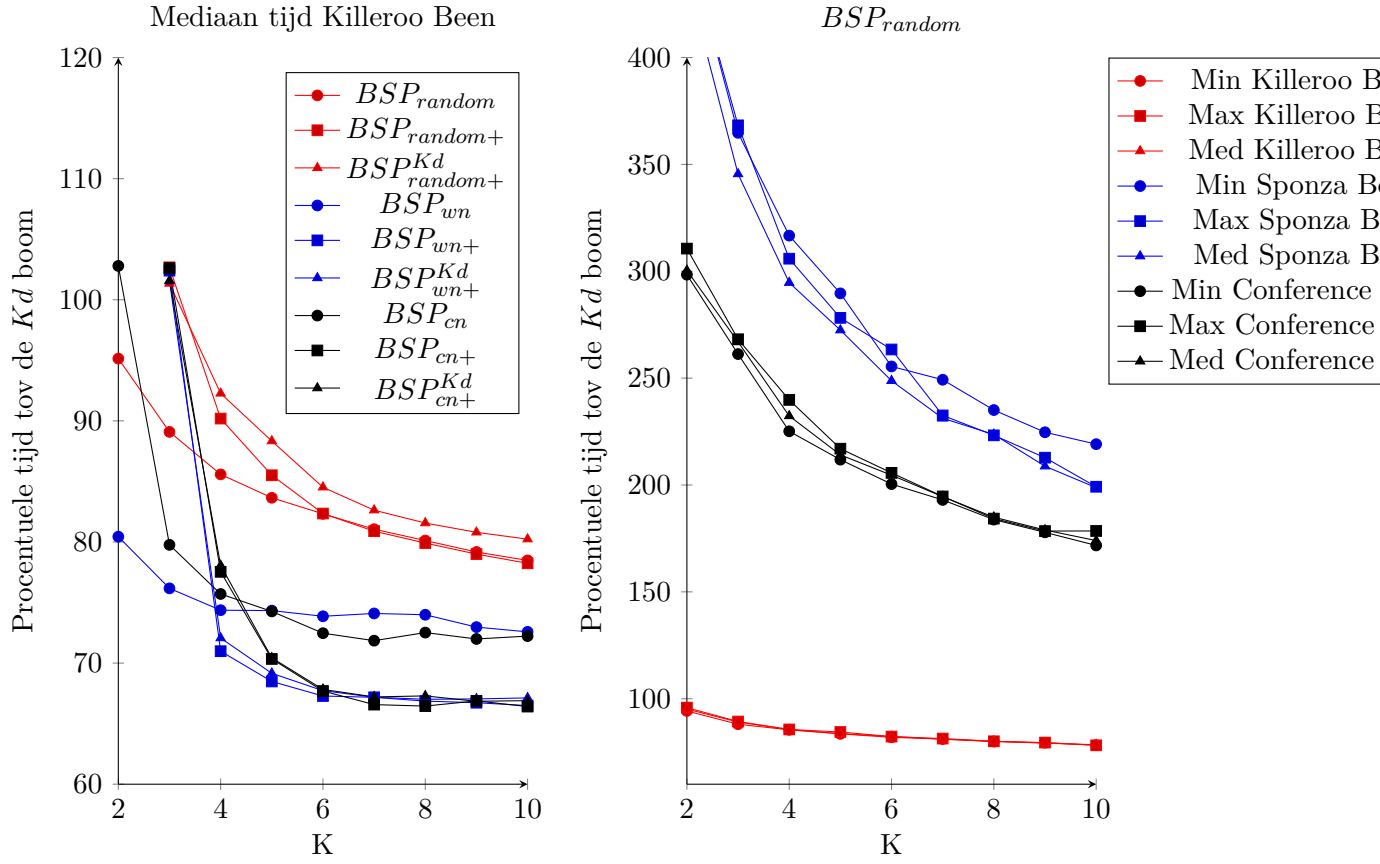
Tabel 5.9: Rendertijd  $BSP_{cn+}^{Kd}$  in functie van K - Deze tabel toont statistieken over de rendertijd voor  $BSP_{cn+}^{Kd}$  voor verschillende waarden van K. Voor elke waarde van K is het algoritme zeven keer uitgevoerd. De *Min* kolom staat voor de minimale rendertijd, de *Max* kolom voor de maximale rendertijd en de *Med* kolom voor de mediaan van de zeven tijd. De (min, max, med) rendertijd is procentueel uitgedrukt ten opzichte van de (min, max, med) rendertijd van de *Kd* boom.

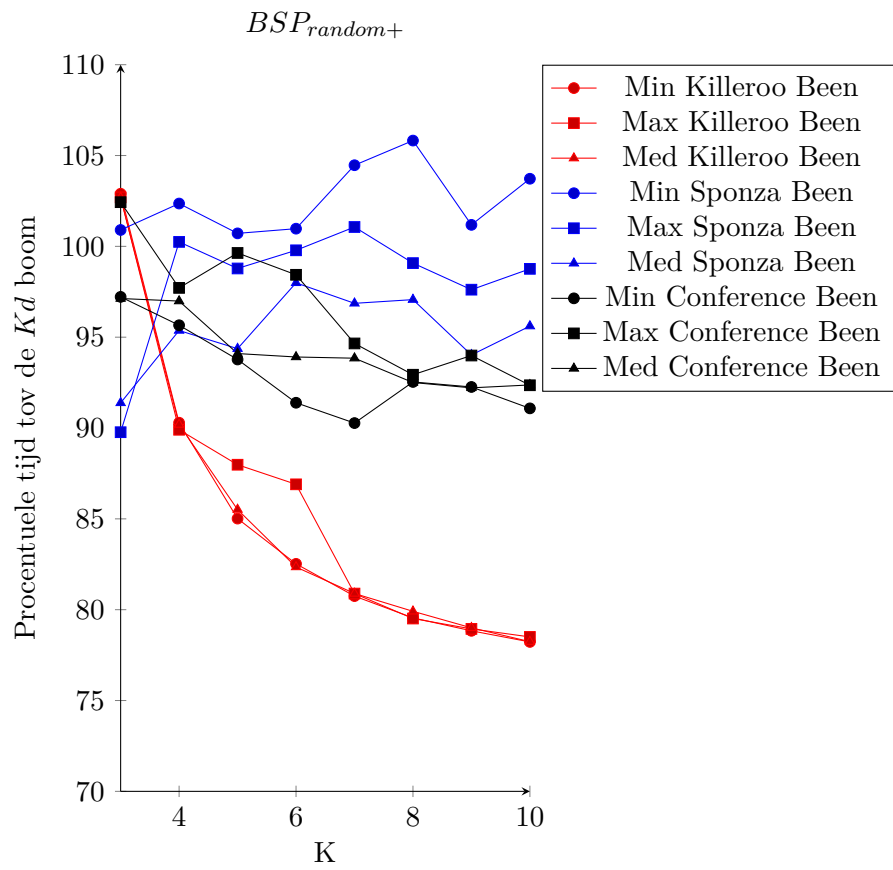
## 5.1 Praktische aspecten

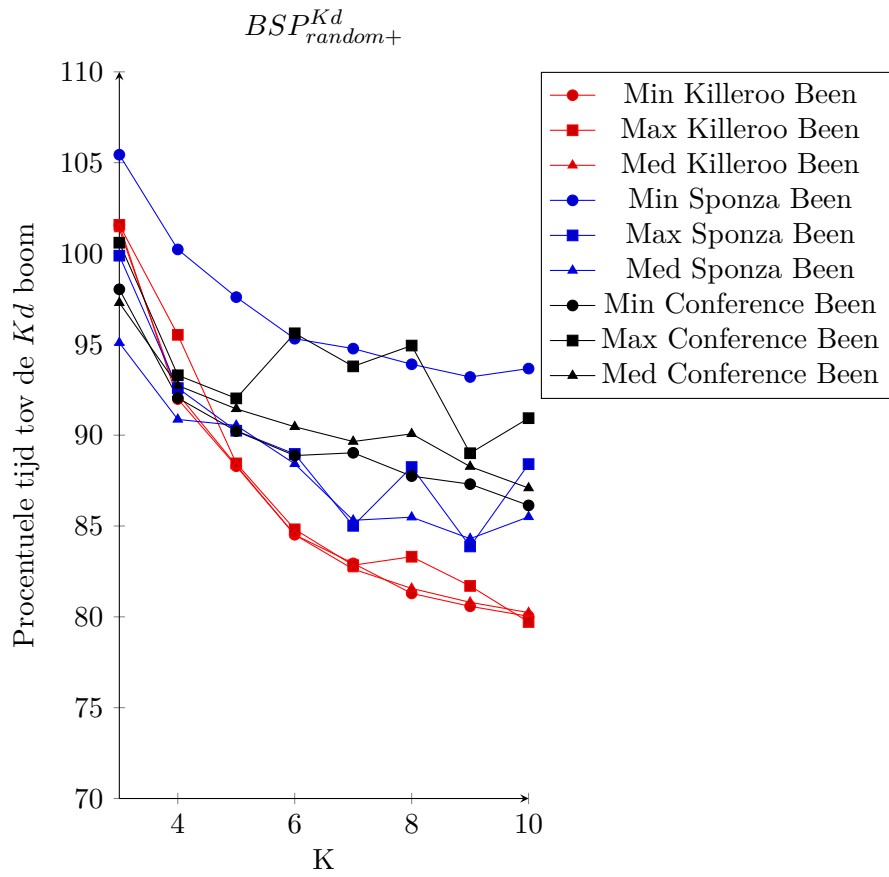
## 5.2 Afhankelijkheid van aantal richtingen

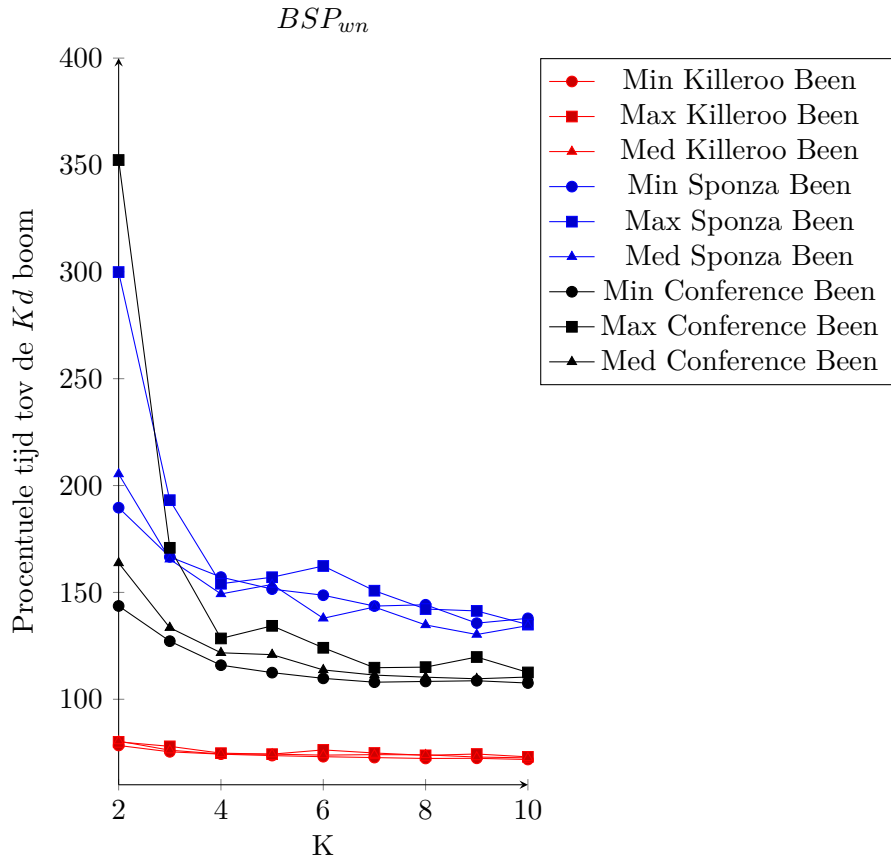


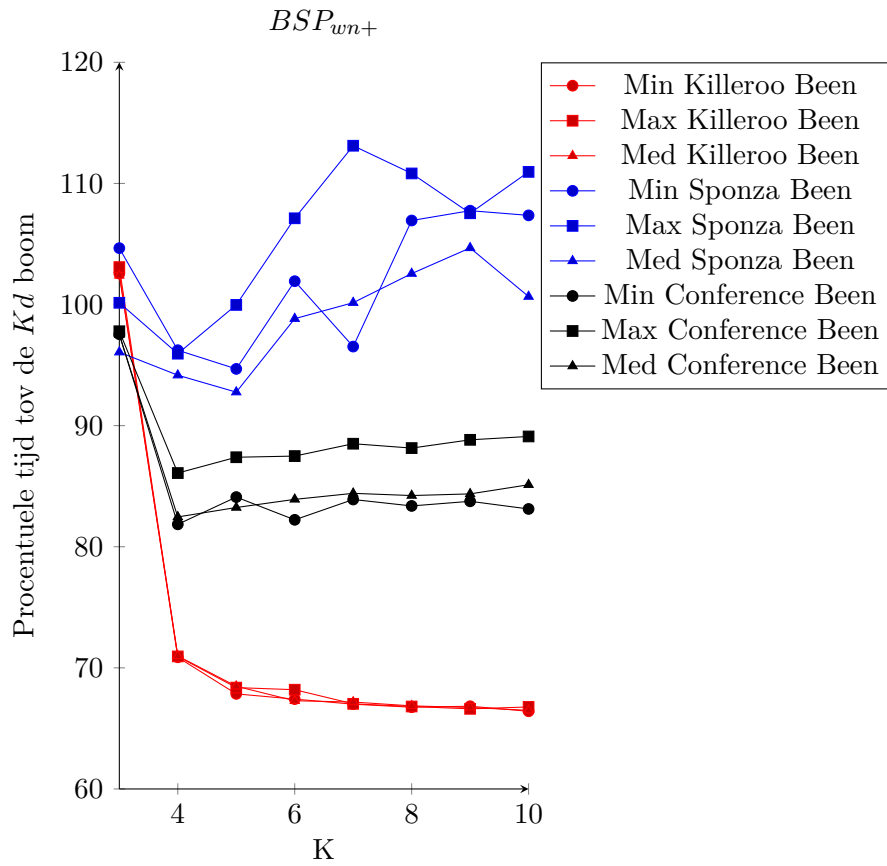


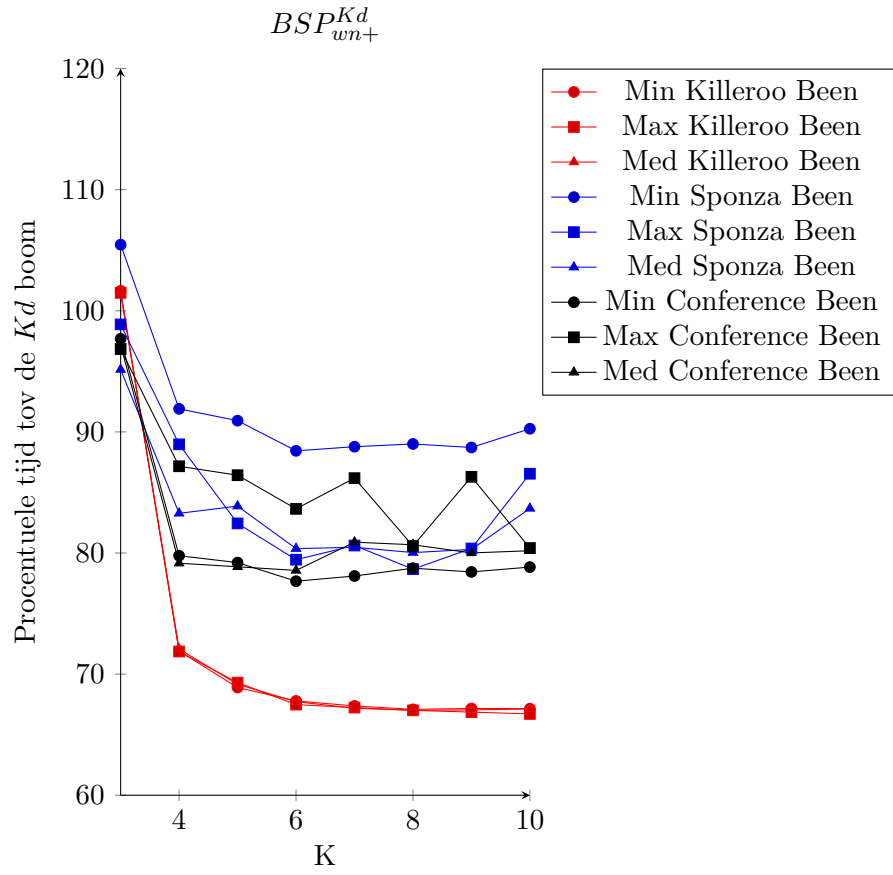


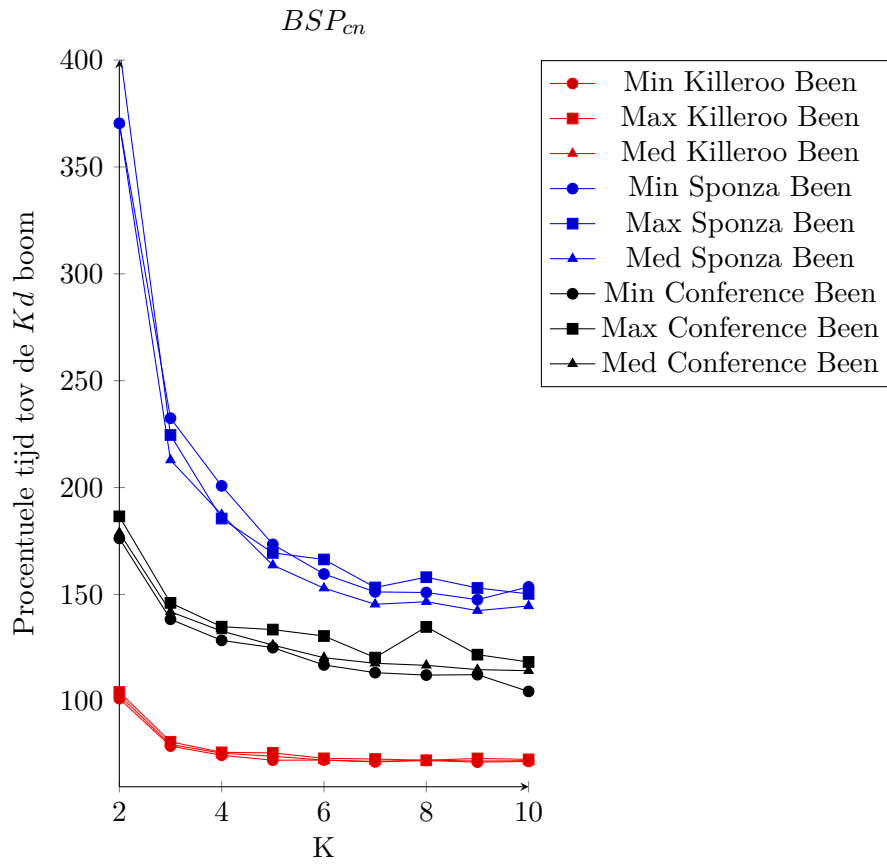


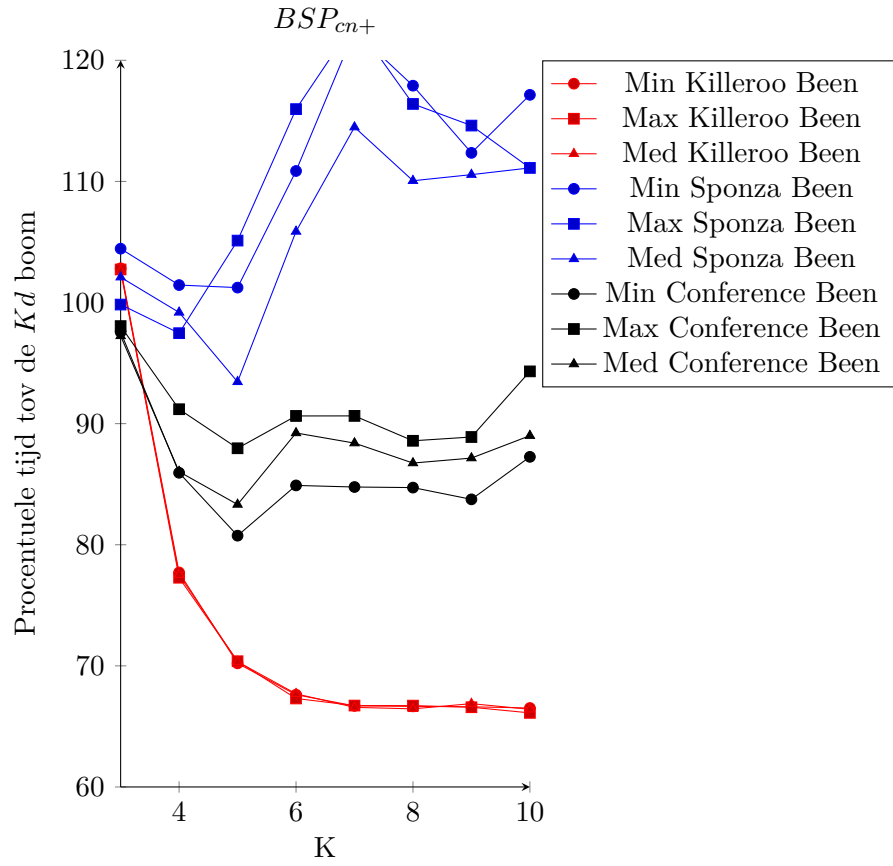








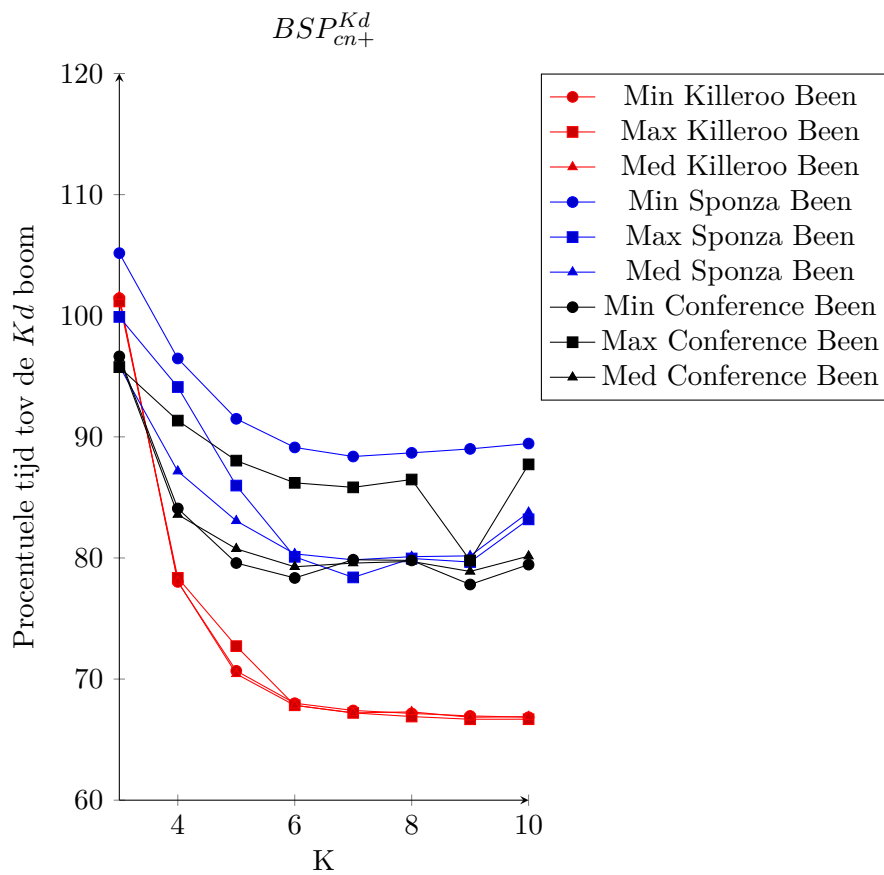






K	Feet			Sponza			Conference Hall			Museum		
	<i>Kd</i>	<i>BSP</i>	Totaal	<i>Kd</i>	<i>BSP</i>	Totaal	<i>Kd</i>	<i>BSP</i>	Totaal	<i>Kd</i>	<i>BSP</i>	Totaal
2	0	642081	642081	0	2473542	2473542	0	954708	954708			
3	0	718759	718759	0	3307722	3307722	0	1203867	1203867			
4	0	783752	783752	0	3831201	3831201	0	1456774	1456774			
5	0	784195	784195	0	4250026	4250026	0	1608240	1608240			
6	0	794381	794381	0	4500849	4500849	0	1658489	1658489			
7	0	809726	809726	0	4721923	4721923	0	1765114	1765114			
8	0	829596	829596	0	4908790	4908790	0	1802687	1802687			
9	0	830428	830428	0	4940333	4940333	0	1819335	1819335			
10	0	856250	856250	0	5099947	5099947	0	1891082	1891082			

Tabel 5.10



### 5.3 Vergelijking



## Hoofdstuk 6

### Besluit



# Bibliografie

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

## Fiche masterproef

*Student:* Jesse Hoobergs

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

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

*UDC:* 621.3

*Korte inhoud:*

Hier komt een heel bondig abstract van hooguit 500 woorden.  $\text{\LaTeX}$  commando's mogen hier gebruikt worden. Blanco lijnen (of het commando `\par`) zijn wel niet toegelaten!

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

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

*Promotor:* Prof. dr. ir. Ph. Dutré

*Assessoren:* Dr. B. Verreet

Prof. dr. R. Vandebril

*Begeleiders:* Ir. M. Moulin

Ir. P. Bartels