Katholieke
Universiteit
Leuven
Faculteit Ingenieurswetenschappen
Departement
Computerwetenschappen

INTERACTIEVE RAY TRACING VAN SKELETGEBASEERDE ANIMATIES

Thomas LOOCKX

Masterproef aangeboden tot het behalen van de graad van Master in de ingenieurswetenschappen: computerwetenschappen

2008–2009

Promotor : Prof. dr. ir. PH. DUTRÉ

Faculteit Ingenieurswetenschappen
Departement Computerwetenschappen
Celestijnenlaan 200A
3001 Leuven
(016) 32 77 00

K.U. Leuven
Academiejaar 2008–2009

Voornaam en naam student : Thomas Loockx

Titel :

Interactieve Ray Tracing van Skeletgebaseerde Animaties

Engelse vertaling :

Interactive Ray Tracing of Skeleton Based Animations

ACM Classificatie: I.3.7.

Korte inhoud :

Ray tracing is een actief onderwerp in computer graphics sedert verschillende decennia. Dankzij de wet van Moore en efficiënte datastructuren is het vandaag mogelijk om met standaard pc's complexe statische scenes te renderen aan interactieve framerates.

Animaties voegen extra complexiteit toe aan het ray tracing probleem omwille van de veranderende geometrie elk frame. Waar vroeger alleen effectieve rendertijd belangrijk was, is nu ook de bouwtijd van datastructuren een beperkende factor. Skeletgebaseerde animaties vormen een belangrijke klasse animaties en zijn de standaard voor het animeren van karakters in computerspelletjes en animatiefilms.

In deze masterproef bespreken we interactieve ray tracing en specifieke technieken voor (skeletgebaseerde) animaties. Van deze technieken maken we een grondige vergelijking op basis van een eigen implementatie.

Masterproef aangeboden tot het behalen van de graad van
Master in de ingenieurswetenschappen: computerwetenschappen

Promotor : Prof. dr. ir. Ph. Dutré

Assessoren : Ir. J. Laurijssen
Prof. Dr. ir. R. Cools

Begeleider : Dr. A. Lagae

Dankwoord

De thesis die voor u ligt zou nooit tot stand zijn gekomen zonder de hulp van verschillende mensen. Eerst en vooral zou ik mijn begeleider Dr. Ares Lagae willen bedanken. Zijn advies, kritiek en ervaring waren onmisbaar gedurende dit intensieve jaar. Ook dank aan mijn promoter Prof. Dr. ir. Ph. Dutré, hij toonde me een fantastische wereld genaamd *computer graphics*.

Mijn vriendin Sarah, die niet alleen mijn thesis heeft nagelezen maar me ook altijd gesteund en aangemoedigd heeft bij het schrijven van deze tekst.

Bedankt aan mijn familie voor het nalezen en verbeteren. Hoewel jullie niet altijd begrepen waarmee ik mee bezig was hebben jullie ongelooflijk bijgedragen aan de kwaliteit van deze tekst.

Tot slot bedankt Johannes Günther voor de hulp bij het begrijpen en implementeren van de *motion decomposition* technieken.

Inhoudsopgave

.

Hoofdstuk 1

Inleiding

Volgens *Wikipedia* [Wik] wordt computer graphics gedefinieerd als: *"Computer graphics is een discipline uit de informatica die zich bezighoudt met weergeven van beelden met behulp van computers."*. Een saaie definitie die het domein weinig eer aan doet. Computer graphics is een fascinerend, multidisciplinair domein. Sedert het ontstaan begin jaren '60 is er veel veranderd. Er zijn nieuwe technieken, snellere algoritmes en betere hardware. Één ding is door de jaren heen constant gebleven en dat is het plezier. Computer graphics is een domein waar werken nooit echt werken lijkt en waar iedereen constant plezier heeft in zijn onderzoek. We zijn dan ook zeer dankbaar voor de kans om ons te verdiepen in een fantastisch deelgebied van computer graphics namelijk *ray tracing*.

Sinds de 'uitvinding' van ray tracing door Arthur Appel in 1968 [App68] is het algoritme een actief onderzoeksonderwerp in computer graphics. Sindsdien zijn er vele wijzigingen aangebracht waardoor ray tracing ondertussen in staat is om fotorealistische afbeeldingen te genereren. Ray tracing is zo alomtegenwoordig in industriën zoals bijvoorbeeld de auto-industrie waar het mogelijk is om levensechte afbeeldingen te genereren van prototypes zonder dat die ook gebouwd worden. Een ander toepassingsdomein is animatiefilms, niet verwonderlijk dat bedrijven als *Pixar* stuwende krachten zijn in het onderzoek naar ray tracing.

Een domein waar ray tracing zijn stempel nog niet helemaal heeft kunnen drukken is in interactieve applicaties. Hoewel het dankzij de wet van Moore mogelijk is om steeds snellere processoren te bouwen, is interactieve ray tracing nog altijd geen gemeengoed. Het is al jaren duidelijk dat ray tracing een eenvoudig en elegant alternatief is voor de huidige renderalgoritmes. De grootste beperking is echter dat er nog altijd geen performantie kan worden bereikt zoals die nodig voor bijvoorbeeld computerspelletjes. Als de rekenkracht van de huidige hardware blijft stijgen zal er uiteindelijk een moment in de (nabije?) toekomst komen waarin interactieve ray tracing mogelijk wordt.

Een subdomein van interactieve ray tracing is het omgaan met animaties. Anima-

ties voegen extra complexiteit toe aan het ray tracing probleem en zijn een belangrijk onderwerp in het huidig onderzoek naar ray tracing. In onze thesis vernauwen we het domein van animaties naar dat van skeletgebaseerde animaties. Deze animaties zijn de standaard voor het modelleren van karakters in animatiefilms en computerspelletjes. We hopen met deze thesis een bescheiden bijdrage te leveren in het huidige onderzoek. We hopen ook dat de lezer evenveel plezier heeft met het lezen van deze thesis als wij plezier hebben beleefd aan het tot stand brengen ervan.

1.1 Doelstellingen

De doelstelling van deze thesis is het onderzoeken van technieken voor interactieve ray tracing van skeletgebaseerde animaties. Meer concreet legt deze thesis de focus op de volgende punten:

1. Onderzoek van de technieken om ray tracing interactief te maken en daaraan gekoppeld een implementatie van een interactieve ray tracer.
2. Onderzoeken en implementeren van specifieke technieken voor ray tracing van animaties in het algemeen en skeletgebaseerde animaties in het bijzonder.
3. Een grondige vergelijking maken van de onderzochte technieken op basis van onze eigen implementatie.

1.2 Overzicht

We beginnen met een inleiding tot animaties en meer specifiek skeletgebaseerde animaties in hoofdstuk 2. In hoofdstuk 3 bespreken we het ray tracing algoritme samen met technieken voor interactieve ray tracing. Specifieke algoritmes voor animaties en skeletgebaseerde animaties worden behandeld in hoofdstuk 4 en een grondige vergelijking hiervan in hoofdstuk 5. Ons besluit en aanwijzingen voor verder onderzoek formuleren we in onze eindconclusie (hoofdstuk 6). Ter verduidelijking hebben we achteraan ook een verklarende woordenlijst samengesteld.

Hoofdstuk 2

Animaties

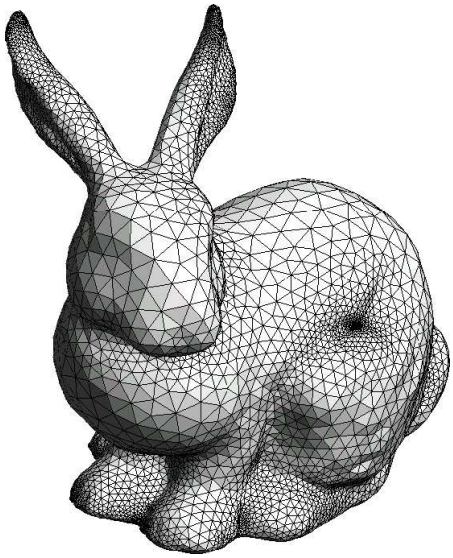
> Animation offers a medium of story telling and visual entertainment which can bring pleasure and information to people of all ages everywhere in the world. - Walt Disney

Iedereen weet ongetwijfeld wat animaties zijn. In onze jeugd hebben we uren gekeken naar tekenfilms zoals 'Mickey Mouse' of animatiefilms zoals 'Bambi'. Velen spelen ontelbare uren computerspelletjes zoals 'Quake', 'World of Warcraft', ... Animaties zijn zo vanzelfsprekend dat we eigenlijk nooit nadenken over deze fascinerende visuele illusies. In dit hoofdstuk proberen we daar verandering in te brengen. We bespreken het principe en de geschiedenis in sectie 2.1. Skeletgebaseerde animatie in sectie 2.2. De classificatie van animaties in sectie 2.3. Tenslotte bespreken we Cal3d een raamwerk voor skeletgebaseerde animaties in sectie 2.4.

2.1 Klassieke Animatie

Klassieke animatie bestaat uit een aantal individuele plaatjes, *frames* genoemd. Frames worden achter elkaar aan een versneld tempo getoond om de illusie van beweging te creëren (meestal 25 frames per seconde). Opeenvolgende frames mogen niet te sterk verschillen want anders ontstaan schokkerige beelden. Voor klassieke animaties werd elk frame met de hand getekend door een animator op een transparant cellofaan, een *cel* genaamd. Op de cel werden de verschillende poses van de animatiefiguurtjes (karakters) getekend. De cel werd dan op een statische achtergrond gelegd. Hierdoor moest niet telkens de achtergrond opnieuw getekend worden en konden verschillende mensen aan verschillende delen van de animatie werken. Uiteindelijk werden verschillende *cel's* op elkaar gelegd om een frame te bekomen.

Het arbeidsintensieve tekenen van alle frames is in het computertijdperk vervangen door gespecialiseerde softwarepakketten. Animators kunnen veel sneller meer frames



Figuur 2.1: Het Stanford Bunny is een klassieke mesh in computer graphics. De mesh is opgebouwd uit 69451 driehoeken en is een 3D-scan van een keramiek beeldje. Bron afbeelding: [Ryp].

produceren en makkelijker wijzigingen aanbrengen. Objecten worden niet meer gemodelleerd met Chinese inkt en penlijnen maar door 3D volumes te vervormen tot de gewenste vorm. Deze 3D volumes noemt men *meshes* en zijn meestal samengesteld uit polygonen. Voor de eenvoud zijn de polygonen vaak driehoeken (zie figuur 2.1). Bovendien laten technieken zoals *key framing* toe om slechts enkele frames te tekenen en de tussenliggende frames door de computer te laten berekenen.

Sinds enkele jaren zijn 3D animatiefilms zeer populair. Een animatie wordt nog steeds gemodelleerd door een animator maar wanneer een frame klaar is wordt het *gerendered*. Renderen transformeert een scene naar een fotorealistische afbeelding of naar een speciale stijl zoals bijvoorbeeld cartoonstijl (niet-fotorealistisch renderen). Ook gewone films maken hiervan gebruik. Bijvoorbeeld virtuele acteurs worden in echte scenes geplakt en gerendered onder de belichtingscondities van de scene waardoor het onderscheid tussen wat echt en wat virtueel is helemaal vervaagd. Een populaire rendertechniek in animatiefilms is *ray tracing*. Ray tracing bespreken we uitgebreid in het volgende hoofdstuk.

2.2 Skeletgebaseerde Animatie

Hoewel de klassieke technieken ook gebruikt werden voor het animeren van animatiefiguurtjes (karakters), is er in de jaren '80 een meer natuurlijke techniek ontwikkeld voor het animeren van karakters. Een karakter wordt voorgesteld door een skelet en

Figuur 2.2: Links: Frame uit 'Steamboat Willie' (1928), één van de eerste Disney animaties. Midden: Frame uit 'Toy Story' (1995), de eerste volledig computergeanimeerde animatiefilm. Rechts: animatiefiguur uit een computerspel.

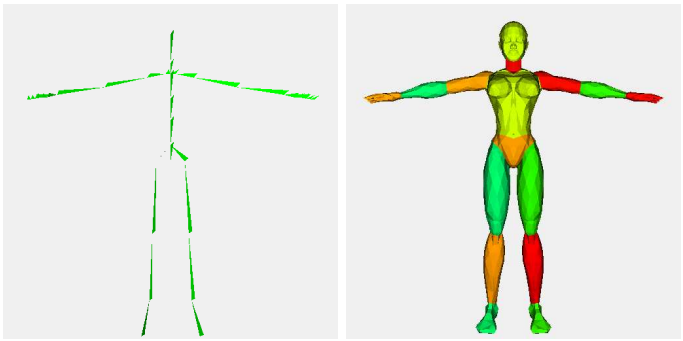
huid (skin). Deze techniek staat bekend als *skinning* en werd voor het eerst beschreven in [MTLT88]. Skinning is tegenwoordig de standaard techniek voor het animeren van karakters in animatiefilms en computerspelletjes. Skinning zoals hier uitgelegd komt in de literatuur ook voor als *lineair blend skinning*, *vertex blending* of *skeletal subspace deformation*.

2.2.1 Techniek

Elk karakter bestaat uit een onderliggend skelet. Het skelet is hiërarchisch opgebouwd uit botten en vormt een boomstructuur. Aan elk bot is een *transformatiematrix* gekoppeld die de vervorming beschrijft t.o.v. een rustpose. Een transformatiematrix is een 4×4 matrix die een combinatie van affiene transformaties beschrijft zoals bijvoorbeeld rotatie of herschaling. Voor een inleiding tot transformatiematrices verwijzen we naar [SAG+05]. Elk frame wordt het skelet vervormd om de gewenste pose te bekomen (zie figuur 2.4).

Over het skelet wordt een verzameling meshes (de skin) gedefinieerd die het uitzicht van het karakter bepalen (bijvoorbeeld de ledematen, het hoofd, de romp, ...). Elke vertex [1] van een mesh wordt beïnvloed door één of meerdere botten. Bijvoorbeeld de vertices van de arm bewegen mee met het overeenkomstige bot. Rond de gewrichten is er vaak invloed van meerdere botten om scheuren in de mesh te voorkomen. Meshes worden aan een skelet gekoppeld in de rusthouding. Om de meshes te animeren vervormt de animator het skelet t.o.v. de rusthouding. De meshes zullen op een natuurlijke wijze mee vervormen.

[1] Zoals hierboven reeds gezegd wordt een mesh vaak gemodelleerd door een verzameling (convexe) polygonen. Voor de eenvoud worden meestal driehoeken als polygonen gekozen. Met vertices worden de vertices of hoekpunten van de driehoeken bedoeld. Andere manieren om een mesh te modelleren zijn spline oppervlakken, impliciete functies, bézier-curven, ... In deze thesis veronderstellen we steeds dat meshes zijn opgebouwd uit driehoeken.



Figuur 2.3: Links: het skelet in de rusthouding. Rechts: meshes over het skelet in de rusthouding. Het skelet wordt in de animatie vervormd t.o.v. de rusthouding waardoor de overeenkomstige meshes mee vervormen.

De positie van een vertex op tijdstip t wordt berekend met een gewogen gemiddelde:

$$v_i(t) = \sum_{i=1}^{N} w_i M_i(t) \tilde{v}_i \qquad \text{met} \sum_{i=1}^{N} w_i = 1$$

$v_i(t)$ is de positie van vertex v_i op tijdstip t. N is het aantal botten die vertex v_i beïnvloeden. $M_i(t)$ is de transformatiematrix van bot i op tijdstip t. w_i bepaalt de invloed van bot i op de vertex en $\tilde{v}_i$ is de positie van de vertex in de rusthouding. Bij elk nieuw frame wordt de skinning operatie uitgevoerd op elke vertex.

Het creëren van skinned animations is het vakgebied van creatieve animatoren. Ze gebruiken hiervoor uitgebreide 3D paketten zoals *3ds Max*, *Blender*, ...

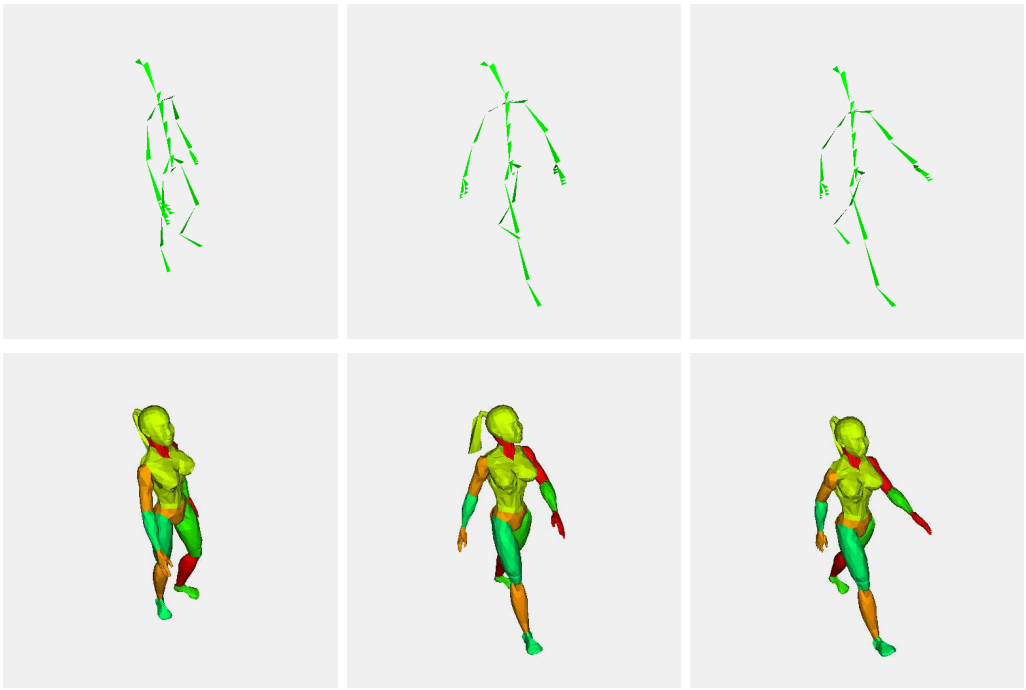
Skinning zoals hier gepresenteerd is niet in staat om alle soorten karakter animaties realistisch weer te geven. Er bestaan complexere skinning methoden of fysisch gebaseerde methoden die deze problemen geheel of gedeeltelijk oplossen.

2.3 Classificatie van Animaties

Animaties is een ruim begrip en moet daarom onderverdeeld worden in verschillende klassen. Hiervoor gebruiken we de classificatie zoals voorgesteld in [WMG+07] doorheen deze thesis.

Statische Scene Een scene waarbij de geometrie ongewijzigd blijft van frame tot frame. De camerapositie en de belichtingscondities mogen veranderen.

Partieel Statische Scene Een subset van de geometrie is in beweging terwijl de rest van de scene statisch blijft. Dit is vaak het geval bij computerspelletjes waarin helden en monsters bewegen in een statische wereld.



Figuur 2.4: Boven: Het onderliggende skelet wordt elk frame vervormd. Onder: De bovenliggende meshes worden onder invloed van het skelet vervormd door skinning.

Dynamische Scene Hier kan in principe alle geometrie wijzigen.

Ook de beweging van de polygonen moet onderverdeeld worden in klassen:

Hiërarchische beweging De polygonen van een scene kunnen onderverdeeld worden in groepen waarbij elke groep dezelfde beweging ondergaat. Een groep wordt een *object* of *cluster* genoemd. Vaak is de beweging van een cluster affien en kan ze beschreven worden door transformatiematrices.

Incoherente beweging De polygonen bewegen onafhankelijk van elkaar.

Semi-Hiërarchische beweging De scene kan worden gepartitioneerd in clusters waarbij elke cluster hiërarchisch beweegt met een kleine hoeveelheid residuele beweging. Een voorbeeld hiervan is een school vissen.

Een scene waarbij de connectiviteit van de polygonen wordt behouden over verschillende frames noemt men een *deformable scene*. Sommige scenes behouden de connectiviteit niet, een voorbeeld hiervan is een explosie.

Skeletgebaseerde animaties kunnen we onderbrengen bij de klasse semi-hiërarchische beweging. De beweging van het skelet is volledig hiërarchisch maar de invloed van

meerdere botten op een vertex zorgt voor een residuele component in de beweging van de vertex. Bovendien is de geometrie in skeletgebaseerde animaties deformable.

2.4 Cal3d

Cal3d [cal] is een open source C++ bibliotheek voor skeletgebaseerde animaties ontwikkeld voor het spel *Worldforge* [wor]. We gebruiken deze bibliotheek in onze implementatie voor het beheer van karakters en animaties. Het skinning algoritme van de bibliotheek hebben we opnieuw geïmplementeerd in onze eigen code omdat de implementatie van de technieken hierdoor eenvoudiger wordt. De bibliotheek bevat 3 voorgedefinieerde karakters (zie tabel 2.1).

Karakter	#Botten	#Driehoeken
Cally	37	3342
Skeleton	23	4604
Paladin	58	3422

Tabel 2.1: De 3 karakters die worden meegeleverd met de Cal3d bibliotheek.

Om later experimenten te kunnen uitvoeren hebben we vier verschillende skeletgebaseerde animaties gedefinieerd op basis van de 3 verschillende karakters:

Cally Walking Cally die rustig wandelt en wuift naar de camera.

Cally Freestyle Cally die een 'karate kick' uitvoert. Dit is een animatie met 'veel beweging', de animatie wijzigt sterk van frame tot frame.

Skeleton Freestyle Skeleton doet een dansje. Merk op dat elk bot van Skeleton wordt beïnvloedt door slechts 1 vertex. Dit is een zuiver hiërarchische animatie.

Paladin Jogging Paladin (een ridder) aan het joggen terwijl hij een pijl afvuurt uit zijn 'virtuele' boog.

Fragmenten uit de animaties zijn te zien in figuur 2.5.

2.5 Besluit

We hebben animaties kort besproken en zijn dieper ingegaan op skeletgebaseerde animaties. Het skinning principe is essentieel omdat het de koppeling maakt tussen het skelet en de vervormingen van de meshes zoals waargenomen in de animatie. De classificatie van animaties laat ons toe om een vooraf vastgelegde terminologie te gebruiken in de rest van deze thesis. Met behulp van Cal3d hebben we vier testanimaties gedefinieerd die we verder gebruiken voor het onderzoeken van verschillende technieken.

Figuur 2.5: Enkele frames uit de voorgedefinieerde animaties. Van boven naar onder: Cally Walking, Cally Freestyle, Skeleton Freestyle en Paladin Jogging.

Hoofdstuk 3

Ray Tracing

> Because of the nature of Moore's law, anything that an extremely clever graphics programmer can do at one point can be replicated by a merely competent programmer some number of years later. - John Carmack

Ray tracing is een centraal thema van deze thesis, daarom wijden we er een volledig hoofdstuk aan. Sectie 3.1 beschrijft het klassieke ray tracing algoritme. Omdat ray tracing in zijn eenvoudigste vorm niet interactief is bespreken we technieken om ray tracing te versnellen in sectie 3.2. Als laatste in sectie 3.3 bespreken we de specifieke moeilijkheden van ray tracing voor animaties.

3.1 Klassieke Ray Tracing

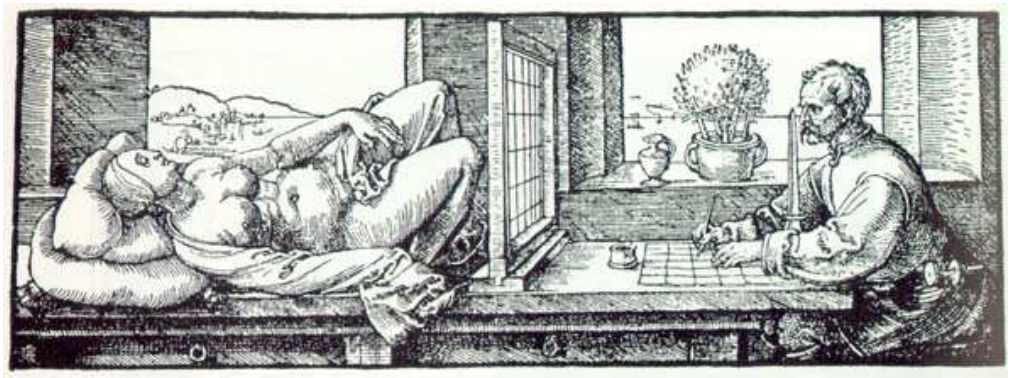
Ray tracing is een eenvoudige methode om vanuit een geometrische beschrijving van een scene een 3-dimensionale afbeelding te genereren. Het is een pixelgebaseerde methode die toelaat om fotorealistische beelden te genereren (zie figuur 3.1).

3.1.1 Algoritme

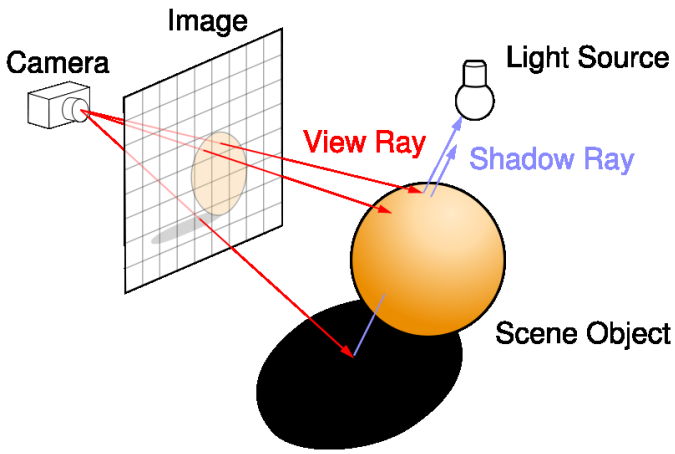
Het ray tracing algoritme heeft als invoer een scene die gemodelleerd is door een artiest. De scene bestaat uit verschillende objecten (vaak opgebouwd uit driehoeken) samengesteld uit verschillende materialen. Het principe is het omgekeerde van de werking van het menselijke oog en is al eeuwen bekend (zie figuur 3.2).

Stralen (*rays*) worden vanuit de camera (ook wel het oogpunt genoemd) door een scherm in de scene *geschoten* (*ray casting*). Het scherm is onderverdeeld in rechthoeken

Figuur 3.1: Fotorealistische afbeeldingen gerendered met het ray tracing algoritme. Bron afbeeldingen: [pov09].



Figuur 3.2: Het principe van ray tracing is al enkele eeuwen bekend. Albrecht Duerer, *Underweysung der Messung mit dem Zirkel und Richtscheyt* (Nurenberg, 1525), Book 3, figure 67. Bron afbeelding: [Dut06].



Figuur 3.3: Stralen worden vanuit de camera door elke pixel van het scherm in de scene geschoten. De pixels krijgen de kleur van het dichtstbijzijnde object. Bron afbeelding: [wik08].

of vierkanten die *pixels* worden genoemd. Door elke pixel gaat exact één straal[1]. De straal moet op intersectie getest worden met elk object in de scene. Alleen de intersectie met het dichtstbijzijnde object is belangrijk. De respectievelijke pixel wordt ingekleurd met de kleur van het dichtstbijzijnde object[2]. Stralen vanuit de camera worden *primaire* stralen genoemd. Het principe wordt getoond in figuur 3.3 en het algoritme in pseudo-code staat in listing 1.

Algorithm 1 Ray tracing algoritme.

```
for all pixels do
  Create primary ray through pixel.
  for all objects do
    Find closest ray/object intersection.
  end for
  if Ray/object intersection found. then
    Fill pixel with object color.
  else
    Fill pixel with background color.
  end if
end for
```

[1]Dit is niet altijd het geval, bijvoorbeeld om effecten zoals *anti-aliasing* te bekomen worden er meerdere stralen door een pixel geschoten. Dit valt echter buiten het bestek van deze thesis dus we veronderstellen altijd slechts één straal per pixel

[2]Algemeen wordt het kleuren van pixels *shading* genoemd. De kleur van de pixel wordt dan bepaald door het materiaal van het object (mat, reflecterend, ...) en de belichting van de scene. In onze ray tracer gebruiken we *diffuse shading* wat overeenkomt met het gebruik van matte materialen.

Het mooie aan ray tracing is dat het algoritme eenvoudig kan worden uitgebreid voor het renderen van schaduwen, reflecties, refracties, ... Bijvoorbeeld om te bepalen of een punt in de schaduw ligt wordt vanuit dat punt een straal geschoten naar de lichtbron. Als de straal wordt onderbroken door een object dan ligt het punt in de schaduw en wordt de respectievelijke pixel zwart gekleurd. Zo een straal wordt een *schaduwstraal* genoemd. In het algemeen worden stralen die vanuit een intersectiepunt worden geschoten *secundaire* stralen genoemd. Voor de lezer die een realistische ray tracer zelf wil bouwen verwijzen we naar [SM03].

3.1.2 Vergelijking met Rasterizatie

Ray tracing is niet het enige render algoritme. De tegenhanger van ray tracing is *rasterizatie*. In tegenstelling tot ray tracing is rasterizatie een objectgebaseerde methode. Een iteratie gaat over elk object van de scene en kijkt door welke pixels het object zichtbaar is om die vervolgens in te kleuren.

Het nadeel van rasterizatie is dat er verschillende kunstgrepen nodig zijn om realistische afbeeldingen te genereren. Bijvoorbeeld welk object het dichtstbijzijnde is gezien vanuit een bepaalde pixel is niet duidelijk en vereist speciale datastructuren zoals een *z-buffer*. Bij ray tracing is perspectief een natuurlijk gevolg van het algoritme. Bij rasterizatie is een *perspectieftransformatie* nodig om perspectief correct weer te geven. Ook het weergeven van schaduwen, reflecties, refracties, ... is niet voor de hand liggend.

Het voordeel van rasterizatie is dat het extreem snel is in vergelijking met ray tracing. Alle huidige grafische kaarten bevatten een implementatie van het rasterizatie-algoritme. Een bekende *API (Application Programmers Interface)* voor rasterizatie is *OpenGL*. De commerciële tegenhanger van Microsoft is *DirectX*.

3.2 Interactieve Ray Tracing

Het probleem bij ray tracing is het hoge aantal straal-driehoek intersectie testen. Een rekenvoorbeeldje om deze stelling te illustreren. In een scene met N driehoeken zijn N straal-driehoek intersectie testen nodig per straal! Stel een gemiddeld complexe scene met 10^5 driehoeken. Voor een afbeelding met een resolutie van 1024^2 pixels zijn er $10^5 \times 1024^2 = 104857600000$ straal-driehoek intersecties nodig. Bovendien is een straal-driehoek intersectie een relatief dure berekening zelfs op huidige architecturen. Het renderen van een scene neemt al gauw enkele minuten tot verschillende uren in beslag met het naïeve algoritme zoals hierboven besproken. Daarom bespreken we in deze sectie enkele technieken om ray tracing te versnellen zodat interactieve rendertijden mogelijk worden.

3.2.1 Versnellingsstructuren

Het zoeken naar de dichtstbijzijnde straal-driehoek intersectie is essentieel een zoekprobleem. De meeste zoekproblemen kunnen in de praktijk opgelost worden in *sub-lineaire* tijd (beter dan $O(N)$, in de praktijk $O(log(N))$) dankzij *verdeel-en-heers* technieken. Een nodige voorwaarde voor efficiënt zoeken is een gesorteerde invoerverzameling (cfr. zoeken in een rij gesorteerde getallen). *Versnellingsstructuren* zijn datastructuren die gebouwd worden in een *preprocessing* stap. Een versnellingsstructuur 'sorteert' de driehoeken van een scene en is analoog aan een gesorteerde lijst bij het zoeken in een rij getallen. Eenmaal een versnellingsstructuur gebouwd is het mogelijk om zeer snel de dichtstbijzijnde straal-driehoek intersectie te vinden voor een gegeven straal.

De twee meest courante manieren om een versnellingsstructuur op te bouwen zijn:

Spatial Subdivision Hierbij wordt de ruimte van de scene onderverdeeld in verschillende ruimtelijke cellen, *voxels* genaamd. *Traversal* van de straal gaat van cel tot cel volgens het pad van de straal. Als een cel een driehoek bevat wordt een straal-driehoek intersectietest uitgevoerd. Voorbeelden hiervan zijn *grids*, *kd-trees* en *bsp-trees*.

Object Hierarchies Hier wordt niet de ruimte onderverdeeld maar worden de objecten hiërarchisch verdeeld in een boomstructuur. Overlap is mogelijk tussen de verschillende delen van de versnellingsstructuur. Een voorbeeld hiervan is een *bounding volume hierarchie (BVH)*.

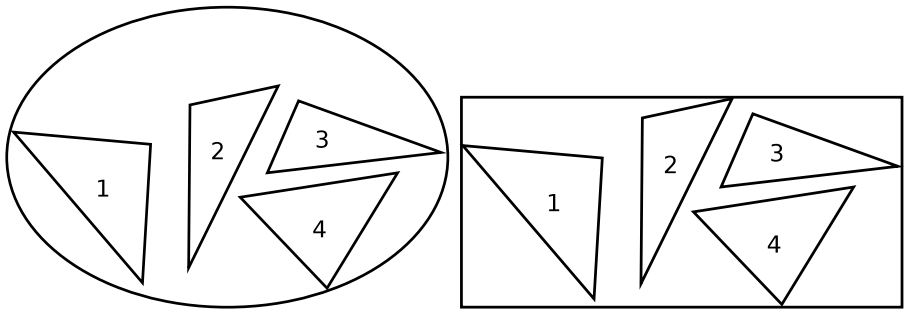
Voor een rigoureuze behandeling van het verschil tussen spatial subdivision en object hierarchies en een bewijs van hun computationele equivalentie verwijzen we naar [Hav07]. De BVH bespreken we uitgebreid in 3.2.2. Voor de andere technieken verwijzen we naar [SAG+05].

Op de vraag *"welke versnellingsstructuur is de beste?"* is het antwoord: dat is afhankelijk van het type scene. [Hav00] maakt een grondige vergelijking van de verschillende technieken. Alle technieken hebben voor- en nadelen maar het is algemeen aanvaard dat gemiddeld gezien de beste performantie voor statische scenes wordt bereikt met een kd-tree, gebouwd met de *surface area heuristic* (SAH). Voor ray tracing van animaties is de zoektocht naar de beste versnellingsstructuur nog altijd een onderwerp van onderzoek.

3.2.2 Bounding Volume Hierarchy

> If I were trapped on a desert island with only one data structure allowed I would bring my trusty BVH - Peter Shirley

De enige versnellingsstructuur die we grondig bespreken is de BVH. In ons onderzoek



Figuur 3.4: Links: Een ellipsoïde als bounding volume rond een verzameling driehoeken. Rechts: Een AABB als bounding volume rond een verzameling driehoeken. Meestal wordt voor driehoeken een AABB gebruikt als bounding volume omwille van zijn eenvoud.

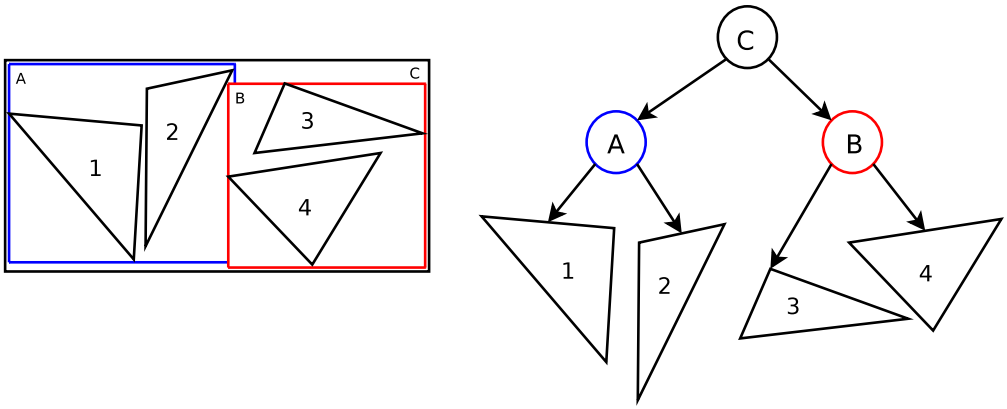
hebben we vanaf dag één gebruik gemaakt van de BVH omwille van zijn robuustheid en simpliciteit.

Principe

Een BVH is simpelweg een zoekboom van bounding volumes. Een bounding volume is een gesloten volume dat volledig een driehoek of verzameling van driehoeken omsluit. Dit bounding volume kan in principe elk soort gesloten volume zijn (zie figuur 3.4). Meestal wordt een as-gealigneerde bounding box (AABB) gebruikt omwille van de eenvoud. Een AABB is een balk met de zes vlakken evenwijdig aan de coördinaatassen. Een BVH versnelt ray tracing omdat het computationeel veel goedkoper is om een straal-AABB intersectietest te doen dan een straal-driehoek intersectietest. Een nodige voorwaarde voor straal-driehoek intersectie is dat de straal ook het bounding volume van de driehoek intersecteert. Het is echter geen voldoende voorwaarde, daarom moeten er bij het doorlopen van een BVH ook nog straal-driehoek intersectietesten worden uitgevoerd. Het aantal straal-driehoek intersectietesten is wel enkele ordegrootte minder dan bij het naïve ray tracing algoritme.

Een bladknoop in een BVH bevat een referentie naar één of meerdere driehoeken en de omsluitende AABB van de driehoeken. Een interne knoop bevat een referentie naar zijn kinderen. Meestal heeft een interne knoop twee kinderen (binaire boom), elk kind kan een bladknoop of een interne knoop zijn. De AABB van een interne knoop is de unie van de AABB van zijn kinderen. Het principe vatten we nog eens samen in figuur 3.5.

Doorlopen van de BVH gebeurt depth-first vanaf de wortelknoop. Als een straal een knoop mist (de straal intersecteert niet de AABB van de knoop) dan zal de straal ook alle kinderen van de knoop missen en gaan we verder met de zusterknoop. Als de



Figuur 3.5: Opbouw van een BVH. Driehoeken 1 en 2 zitten in knoop A die als AABB de blauwe rechthoek heeft. Driehoeken 3 en 4 zitten in knoop B die als AABB de rode rechthoek heeft. De wortelknoop C heeft als kinderen knopen A en B. De AABB van C is de unie van de AABBs van A en C namelijk de zwarte rechthoek. Merk op dat de AABBs van A en C overlappen. Omdat AABBs kunnen overlappen in een BVH is het onmogelijk om een BVH strikt *front-to-back* te doorlopen.

straal een knoop raakt testen we ook de kinderknopen. Bij een bladknoop wordt getest of de driehoeken worden geraakt door de straal. Dankzij dit *traversal* algoritme wordt in de praktijk een straal-driehoek intersectie gevonden in logaritmische tijd. Voor een voorbeeld in C++ zie 3.1.

Listing 3.1: C++ traversal code voor een BVH.

```
bool BVH::traverse(const Node& node, const Ray& ray) const
{
    if (node.boundingBox().hit(ray)) {
        if (node.isLeaf()) {
            return checkTriangles(node, ray);
        } else {
            bool lhit = traverse(node.leftChild(), ray);
            bool rhit = traverse(node.rightChild(), ray);
            if (lhit || rhit)
                return true;
        }
    }
    return false;
}
```

Bouwalgoritme

Voor het bouwen van een BVH zijn er twee verschillende aanpakken namelijk:

Bottom-up De individuele driehoeken worden telkens gegroepeerd in een bounding box. Recursief worden de bounding boxes of driehoeken verder samengenomen. Er wordt gestopt wanneer er nog één bounding box overblijft.

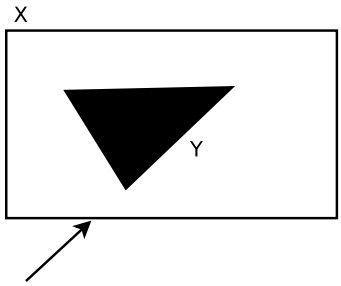
Top-down Hier wordt er gestart met een bounding box die alle driehoeken van de scene omsluit. Recursief wordt de bounding box verder opgedeeld in kleinere bounding boxes. Er wordt gestopt wanneer elke bounding box minder dan een vooraf bepaald aantal primitieven bevat.

Onderzoek heeft aangetoond dat top-down aanpak veel betere resultaten oplevert dan bottom-up. Daarom bespreken we hier top-down aanpakken. In theorie zijn er $2^N - 1$ mogelijke manieren om N driehoeken te verdelen in twee verzamelingen. In praktische algoritmes worden niet alle mogelijkheden beschouwd.

Het eenvoudigste bouwalgoritme sorteert de driehoeken volgens een coördinaatas. De gesorteerde driehoeken verdelen we over twee verzamelingen, elke verzameling bevat de helft van de driehoeken. De AABB van deze driehoeken is de AABB van een interne knoop. De twee verzamelingen driehoeken worden opnieuw recursief opgedeeld bij de bouw van de twee kinderknopen. Wanneer nog een klein aantal driehoeken overblijft wordt een bladknoop gemaakt. Een andere methode is om de driehoeken te partitioneren in twee groepen volgens een splitvlak in de ruimte. Dit noemt men de *median split* methode. Het bouwen van een BVH is in principe een sorteerprobleem. Een verzameling van N elementen (driehoeken) sorteren heeft tijdscomplexiteit $O(N \times log(N))$.

Het is algemeen aanvaard dat de beste methode om een versnellingsstructuur te bouwen is volgens de *surface area heuristic* (SAH). Het idee werd voor het eerst gelanceerd door MacDonald en Booth [MB90]. Deze methode is de standaard voor het bouwen van kd-trees en wordt nog maar recent gebruikt voor BVHs. De SAH zoekt de positie van het beste splitvlak door het minimum van een kostfunctie te bepalen. De kostfunctie geeft de verwachte traversal kost voor een bepaalde deelboom. De heuristiek maakt gebruik van een stelling uit de statistische geometrie: *De kans dat een willekeurige straal die een convex oppervlak X raakt ook een convex oppervlak Y raakt $p(Y|X)$ wordt gegeven door de verhouding van de oppervlaktes van de oppervlakken (Surface Area of kortweg SA)* $p(Y|X) = \frac{SA(Y)}{SA(X)}$. De stelling wordt geïllustreerd in figuur 3.7. Voor AABBs is het zeer makkelijk om het oppervlak te berekenen.

Stel een volume V met N driehoeken wordt door een splitvlak j verdeeld in 2 halfruimten V_L en V_R. De verzameling driehoeken wordt hierdoor gepartitioneerd in 2 partities L en R met N_R en N_L driehoeken. De kost om het volume V te doorlopen als



Figuur 3.6: De kans dat de straal Y raakt als ze ook X raakt wordt gegeven door de verhouding van de oppervlaktes van de oppervlakken of meer formeel $P(Y|X) = \frac{SA(Y)}{SA(X)}$.

j het splitvlak is wordt gegeven door:

$$Kost_j = K_T + K_I(\frac{SA(V_L)}{SA(V)}N_L) + (\frac{SA(V_R)}{SA(V)}N_R))$$

Met K_T en K_I vastgelegde constanten. Meestal worden die 1 gekozen zodat de kostfunctie vereenvoudigd kan worden naar:

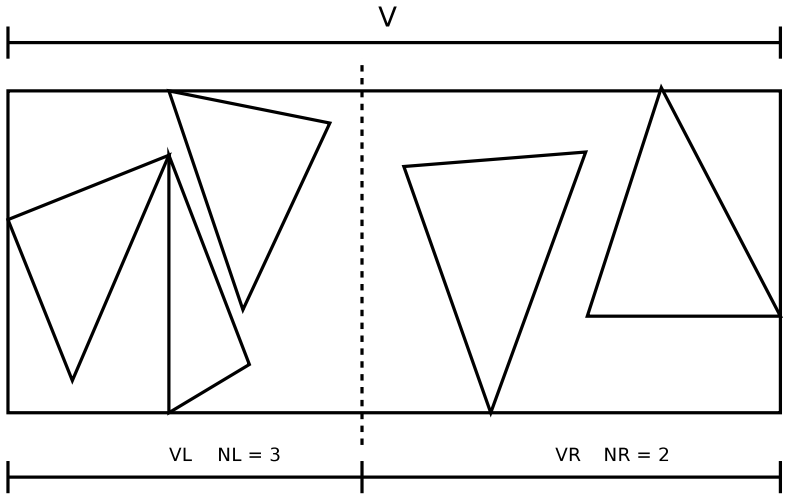
$$Kost_j = \frac{SA(V_L)}{SA(V)}N_L + \frac{SA(V_R)}{SA(V)}N_R$$

Met deze kostfunctie kan de kost van elke interessant splitvlak worden onderzocht. Het splitvlak met de laagste kost partitioneert de driehoeken in 2 partities die opnieuw recursief worden opgedeeld. Het splitsen stopt als de kost om op te delen hoger is dan de kost om elke driehoek afzonderlijk te testen. Wij bouwen onze BVH volgens een variant van de SAH die niet de SAH kostfunctie evalueert maar een discretisatie ervan (zie sectie 4.1).

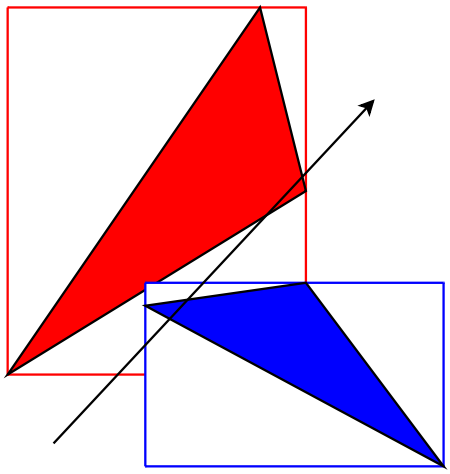
Optimalisaties

Omdat de performantie van onze ray tracer grotendeels afhangt van de kwaliteit van onze BVH hebben we enkele optimalisaties geïmplementeerd. We lichten deze even toe.

[Wal04] verdedigt het gebruik van cache efficiënte layouts voor knopen van een kd-tree. De kost voor het overlopen van een versnellingsstructuur bestaat grotendeels uit geheugen toegangen. Als het aantal bytes nodig om een knoop voor te stellen een deler is van de lengte van een cache lijn kunnen er verschillende knopen tegelijk in de cache geplaatst worden met slechts 1 *cache miss*. We hebben onze BVH knopen 32 bytes groot gemaakt zodat er 4 passen in een 128 byte cache lijn ([Wal04] comprimeert kd-tree knopen tot 8 byte!). Deze techniek geeft gemiddelde speed-ups van 5%. [YL06] gaat zelfs nog verder en stelt een algoritme voor om de efficiënste layout te berekenen voor een BVH.



Figuur 3.7: Het splitvlak partitioneert het volume V in 2 halfruimten V_L en V_R. De traversal kost als het splitvlak gekozen wordt kan nu berekend worden.



Figuur 3.8: De rode bounding box wordt eerst getest en hierin wordt een eerste straal-driehoek intersectie gevonden. Later wordt de blauwe bounding box getest en ook hierin wordt ook een straal-driehoek intersectie gevonden die dichterbij is. Omwille van de mogelijke overlap tussen verschillende delen van de BVH kan er in principe pas gestopt worden wanneer alle intersecties zijn gevonden en dus is strikt front-to-back doorlopen onmogelijk.

Een nadeel van BVHs is dat het niet mogelijk is om ze *front-to-back* te doorlopen zoals bijvoorbeeld kd-trees. Dit omdat bounding boxes van knopen uit verschillende delen van de boom elkaar kunnen overlappen (zie figuur 3.8). Als een intersectie gevonden is in de linker deelboom, dan moet altijd de rechter deelboom nog doorlopen worden omdat daar misschien nog een intersectie mogelijk is die dichterbij is (t.o.v. het oogpunt). Front-to-back traversal kan benaderd worden door telkens de afstand tot de voorlopig dichtstbijzijnde straal-AABB intersectie bij te houden. Als daarna een knoop een grotere intersectieafstand heeft moeten zijn kinderen niet meer doorlopen worden. De keuze eerst linkerkind dan rechterkind heeft ook een grote impact op de performantie. Als elke knoop weet volgens welke as hij gesplitst is (x, y of z) dan kan a.d.h.v. het teken van de richting van de straal bepaald worden welke node eerst doorlopen moet worden. Bijvoorbeeld een knoop die gesplitst is volgens de x-as, als de straal een positief teken heeft in de x-richting doorlopen we eerst het linker kind dan het rechter (in de veronderstelling dat alle driehoeken links van het splitvlak in het linker kind terechtkomen). Dit is slechts een *pseudo* front-to-back traversal maar we hebben in onze implementatie toch voor sommige scenes een halvering van het aantal straal-driehoek testen kunnen waarnemen.

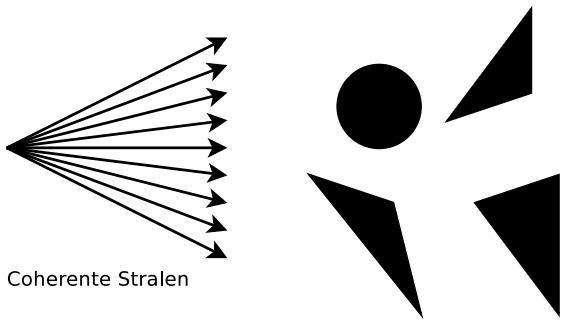
Omdat het traversal algoritme van een BVH recursief is kan dat tot wat extra werk leidden door de vele recursieve functie aanroepen. Daarom hebben we het traversal algoritme iteratief geïmplementeerd en houden we onze eigen stack bij. Een bijkomend voordeel is een beetje minder geheugengebruik.

3.2.3 Pakket Ray Tracing

Een belangrijke voorwaartse sprong in performantie van ray tracing werd gedaan met het uitbuiten van coherentie van stralen. Vooral primaire stralen worden gebundeld in een pakket (zie figuur 3.9) en gaan allemaal samen door de versnellingsstructuur. Het doorlopen van een versnellingsstructuur met een pakket stralen is in principe gelijk aan het doorlopen van een versnellingsstructuur met één straal. Pakket traversal leent zich wel tot enkele belangrijke optimalisaties.

Het bundelen van stralen in een pakket verdeeld de kost van sommige operaties over een volledig pakket. Pakketten voor primaire stralen zijn altijd een vierkante bundel stralen die een tegel vormen op het scherm. Het aantal is altijd een veelvoud van vier. De standaard pakketgroottes zijn $2 \times 2, 4 \times 4, 8 \times 8, 32 \times 32$ en 64×64. Het veelvoud van vier is nodig zodat de pakketten kunnen verwerkt worden door *SIMD* (Single Instruction, Multiple Data) instructies. Dit zijn instructies die parallel eenzelfde bewerking kunnen uitvoeren op meerdere operanden. In populaire architecturen zoals de *Intel* architectuur is het mogelijk om operaties uit te voeren op vier getallen in parallel, dit verklaart het veelvoud van vier voor ray pakket groottes. Het idee voor pakket ray tracing werd het eerst aangehaald in [WBWS01]. Een methode om te controleren of een pakket een knoop mist tijdens traversal werd uitgewerkt in [RSH05].

[WBS07] en [LeYM06] bespreken pakket traversal specifiek voor de BVH en intro-



Figuur 3.9: Coherente stralen kunnen makkelijk samengenomen worden tot één pakket. De stralen worden dan als pakket in de scene *geschoten*.

duceren enkele optimalisaties:

Early Hit Test Bij een interne knoop is het voldoende als een enkele straal de AABB van de knoop raakt. Onmiddellijk kan er afgedaald worden in de deelboom zonder de andere stralen van het pakket te testen. Hiermee kost het testen van een interne knoop vaak slechts één straal-AABB intersectie.

Early Miss Exit Als het pakket de AABB mist moeten in principe alle stralen uit het pakket getest worden op overlap met de AABB. [RSH05] introduceert een pakket/AABB overlap test die ineens berekent of het volledige pakket de AABB mist. Het efficiëntste is om eerst een straal te testen op intersectie en als deze test faalt een overlap test te doen.

First Active Ray Tracking In principe kan er afgedaald worden in de kinderen van een knoop als een straal de AABB van de knoop raakt. Dit is daarom niet de eerste. Door de index van de eerste straal die raakt (*first active ray*) bij te houden kunnen we AABB straal intersecties vermijden in alle knopen onder de huidige knoop. Omdat een straal die een knoop mist gegarandeerd ook alle onderliggende knopen zal missen.

Leaf Traversal In een bladknoop wordt eerst nog een straal-AABB test gedaan voor de dure straal-driehoek test. Dit omdat door *early hit test* te gebruiken er waarschijnlijk stralen in het pakket zitten die de driehoeken in de bladknoop niet raken. Door alle stralen vanaf de first active ray in het pakket nog eens te testen met de AABB van de knoop kunnen nog een aantal straal-driehoek testen vermeden worden.

Pakket traversal hebben we ook in onze ray tracer geïmplementeerd. Early miss exit en SIMD extensions hebben we niet geïmplementeerd. De performantie van onze ray tracer heeft een flinke *boost* gekregen door het gebruik van ray pakketten. Afhankelijk van de scene zijn onze rendertijden een factor 2 tot een factor 4 verbeterd.

3.3 Interactieve Ray Tracing van Animaties

Animaties voegen een extra dimensie toe aan ray tracing namelijk het omgaan met geanimeerde scenes. Voor interactieve ray tracing zijn versnellingsstructuren nodig. Bij een statische scene is de bouwtijd geen probleem, de verwachte performantiewinst bij ray tracing zal altijd veel groter zijn dan de tijd nodig om de versnellingsstructuur te bouwen.

Bij een dynamische scene verandert de geometrie elk frame zodat in principe de oude versnellingsstructuur van vorig frame ongeldig is. De bouwtijd van de versnellingsstructuur is nu echter een integraal onderdeel van de frametijd. Voor animaties is niet alleen snel renderen een probleem, ook het bouwen van een versnellingsstructuur moet vanaf nu snel gaan. Vanaf nu moeten bouwalgoritmes voor versnellingsstructuren streven naar de beste rendertijd - bouwtijd verhouding.

In het huidige onderzoek naar interactieve ray tracing van animaties zijn er verschillende denkpistes voor het omgaan met versnellingsstructuren in animaties:

Rebuilding Herbouwen van de versnellingsstructuur elk frame. Als de bouwtijd beperkt is, is het onmogelijk om een optimale versnellingsstructuur te bouwen voor elk frame.

Update In plaats van elk frame te herbouwen wordt de versnellingsstructuur aangepast om te anticiperen op de veranderende geometrie van de scene. Meestal is de versnellingsstructuur optimaal voor het frame waarvoor hij gebouwd werd maar verslechterd de rendertijd bij de volgende frames.

One fits all Proberen om een versnellingsstructuur vooraf te bouwen die geldig blijft voor elk frame. Deze versnellingsstructuur is dan een gemiddelde voor alle frames en zal nooit zo goed kunnen zijn als een versnellingsstructuur speciaal gebouwd voor een specifiek frame.

Al deze technieken offeren rendertijd op voor bouwtijd of omgekeerd. Volgend hoofdstuk onderzoeken we technieken uit elk van bovenstaande denkpistes. Ook het aanpassen van de geometrie elk frame is een kost die voor complexe scenes niet kan verwaarloosd worden.

3.4 Implementatie

Als onderdeel van deze thesis hebben we zelf geprobeerd een interactieve ray tracer te bouwen. Onze ray tracer is volledig in C++ geïmplementeerd en maakt gebruik van een BVH als versnellingsstructuur en ray pakketten voor efficiënte traversal. SIMD extensies

Scene	# Driehoeken	Rendertijd (s)	pakket grootte
Ulm Box	492	0.59	16x16
Classroom	9k	0.98	16x16
Office	34k	0.82	16x16
Cabin	220k	1.28	16x16
Armadillo	346k	0.65	8x8
Atrium	560k	1.35	8x8
Conference	987k	1.13	16x16
Cruiser	3.6M	1.9	4x4
Dragon	7.21M	0.83	4x4

Tabel 3.1: Rendertijden van onze ray tracer voor enkele veelgebruikte testscenes. De optimale performantie voor een bepaalde scene is sterk afhankelijk van het aantal stralen in een pakket. Alle afbeeldingen zijn gerendered op een resolutie van 1024x1024 pixels.

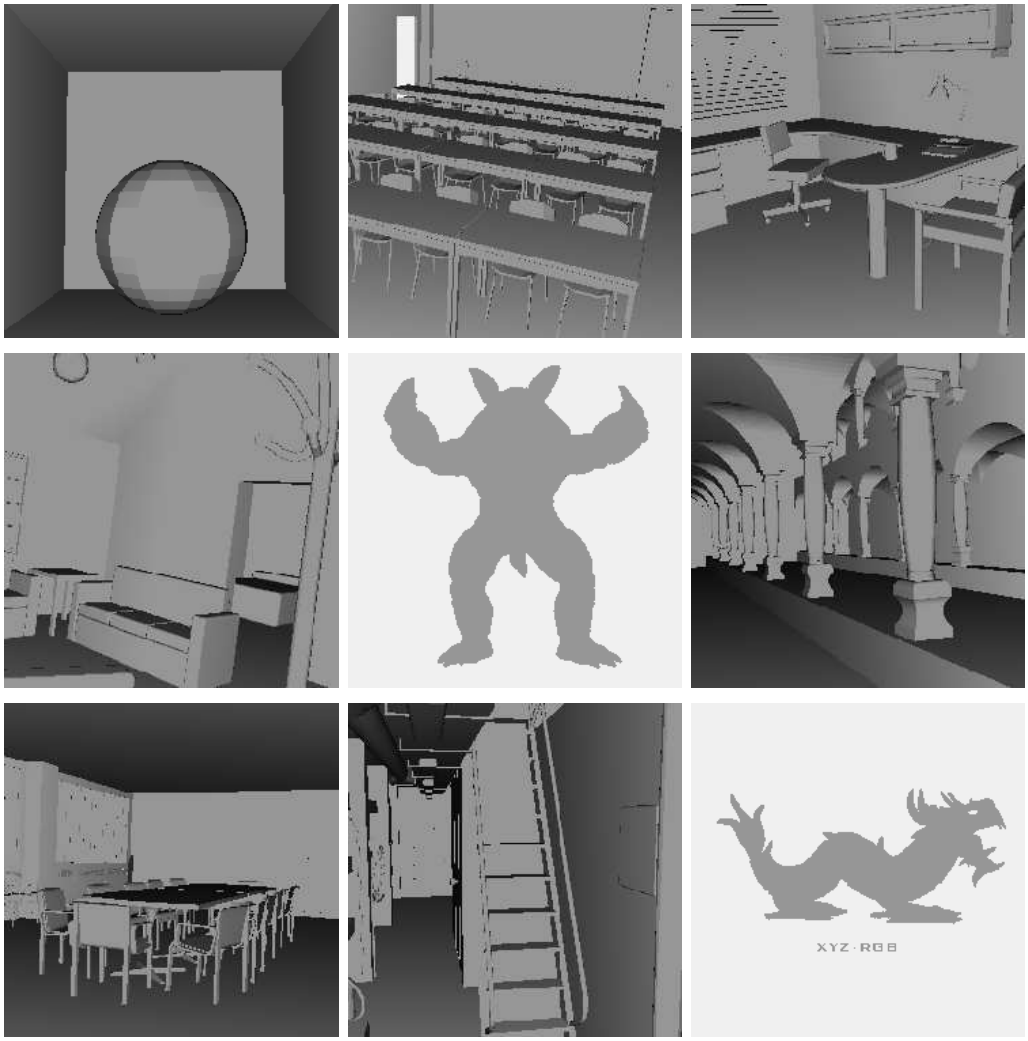
of andere architectuur specifieke optimalisaties hebben we niet gebruikt in onze ray tracer wegens te weinig tijd en te weinig ervaring. Spijtig, want alle *state-of-the-art* ray tracers maken hiervan gebruik.

Om de performantie van onze ray tracer te testen hebben we rendertijden gemeten voor enkele scenes uit het Stanford 3D Scan Repository [Sta] en uit de bwfirt benchmark suite [Uni]. De resultaten zijn samengevat in tabel 3.1 en de afbeeldingen zijn te zien in figuur 3.10. [3]

3.5 Besluit

Hoewel klassieke ray tracing een simpel en elegant algoritme is voor het renderen van fotorealistische afbeeldingen, is het veel te traag voor interactieve toepassingen omwille van de computationele kost en het hoge aantal straal-driehoek intersecties. Daarom zijn er specifieke versnellingsstructuren bedacht voor het versnellen van ray tracing die proberen het aantal straal-driehoek intersecties drastisch te verminderen. Concreet hebben we de BVH besproken omdat de BVH de structuur is die we zelf geïmplementeerd hebben voor ons onderzoek. Bovendien kunnen primaire stralen perfect gebundeld worden in zgn. pakketten zodat de traversal kost voor een versnellingsstructuur kan verdeeld worden over de verschillende stralen in het pakket. Bij ray tracing van animaties is niet alleen de rendertijd belangrijk maar wordt ook de tijd voor het bouwen van een versnellingsstructuur een beperkende factor.

[3] Al onze resultaten zijn verkregen op een Intel Core2 E6600 machine met een kloksnelheid van 2.40GHz, 4096KB cache en 4GB werkgeheugen. We hebben altijd slechts 1 core van de processor gebruikt.



Figuur 3.10: Afbeeldingen gerendered met onze eigen ray tracer: (v.l.n.r) Ulm Box, Classroom, Office, Cabin, Armadillo, Atrium, Conference, Cruiser en Dragon.

Hoofdstuk 4

Ray Tracing van Skeletgebaseerde Animaties

> If I have seen farther than others, it is because I was standing on the shoulders of giants. - Isaac Newton

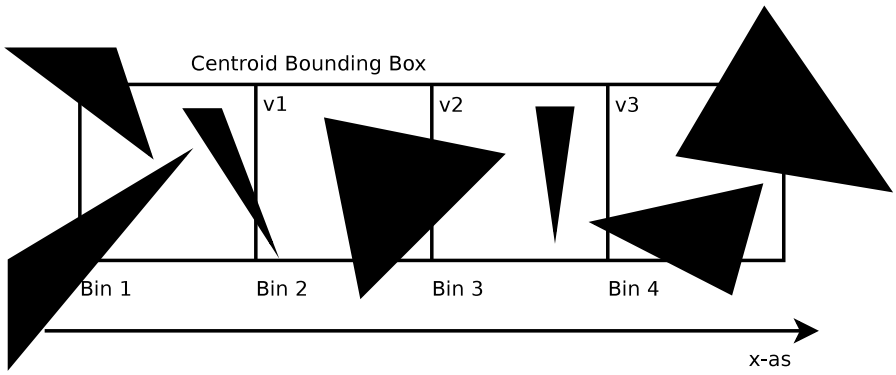
In dit hoofdstuk bespreken we enkele technieken voor ray tracing van skeletgebaseerde animaties. We beginnen met *fast rebuilds* (4.1) een algoritme dat algemeen toepasbaar is voor alle klassen animaties. Vervolgens bespreken we *refitting* (4.2), toepasbaar voor deformable scenes. Als laatste introduceren we een techniek die speciaal ontwikkeld is voor skeletgebaseerde animaties (4.3) en onze eigen variant hiervan (4.4). Voor al deze implementaties gebruiken we uiteraard de BVH als versnellingsstructuur.

4.1 Fast Rebuilds

De meest algemene techniek voor ray tracing van animaties is om elk frame de versnellingsstructuur opnieuw te bouwen. Het voordeel is dat deze techniek werkt voor alle klassen animaties. Het nadeel is dat een versnellingsstructuur opbouwen veel tijd kost. [Wal07] introduceert een techniek voor het snel bouwen van BVH versnellingsstructuren op basis van de surface area heuristic (zie 3.2.2). Het idee is om de kostfunctie slechts in een beperkt aantal discrete punten te berekenen.

4.1.1 Methode

Eerst wordt er een bounding box berekend die de centra van de bounding boxes van elke driehoek omvat. Deze bounding box wordt onderverdeeld in K gelijke delen volgens een bepaalde as (als as wordt meestal de as gekozen waar de bounding box het grootste is).



Figuur 4.1: Binning van een scene. De bounding box wordt onderverdeeld in 4 bins waarover de driehoeken worden verdeeld. Het verdelen van de driehoeken gebeurt op basis van het middelpunt van de AABB (van de driehoek). De vlakken tussen de bins (v1, v2 en v3) zijn de mogelijke splitvlakken.

Dit zijn de *bins* $B_1 \dots B_K$ Het *bin domain* is de ruimte die een bin bepaalt (zie figuur 4.1).

De verdeling over de bins gebeurt op basis van het middelpunt van de bounding box van de driehoek. Elke driehoek valt in de bin die het middelpunt van zijn bounding box bevat.

De scheidingsvlakken tussen de bins zijn de mogelijke *splitvlakken.* Zo zijn er $K - 1$ manieren om de driehoeken te verdelen in twee partities. Als splitvlak j gekozen wordt dan worden de partities:

$$Driehoeken_{Links,j} = \{B_1 \dots B_j\}$$
$$Driehoeken_{Rechts,j} = \{Bj + 1 \dots B_K\}$$

Stel dat er in bin B_i n_i driehoeken zitten dan wordt het aantal driehoeken links en rechts van splitvlak:

$$N_{Links,j} = \sum_{i=1}^{j} n_i \qquad\qquad N_{Rechts,j} = \sum_{i=j+1}^{K} n_i$$

Voor elke bin wordt de bounding box berekend die de driehoeken in de bin omsluit. Er is geen verband tussen deze bounding box en het bin domein. Met deze bounding

boxes kan het oppervlak links en rechts van de split berekend worden met de formules:

$$A_{L,j} = SurfaceArea(\bigcup_{i=1}^{j} bbox_i)$$
$$A_{R,j} = SurfaceArea(\bigcup_{i=j+1}^{K} bbox_i)$$

De vereenvoudigde kostfunctie voor split j wordt dan:

$$Kost_j = A_{L,j}N_{L,j} + A_{R,j}N_{R,j} \qquad j = 1 \ldots K-1$$

Merk op dat dit een discretisatie is van de echte SAH kostfunctie, de kostfunctie hier wordt slechts geëvalueerd in K punten.

Om de kost van alle K mogelijke splitvlakken te bereken is eerst een iteratie nodig van links naar rechts over de bins. Deze iteratie accumuleert de bounding boxes en het aantal driehoeken links. Daarna gebeurt een analoge iteratie van rechts naar links. Na deze iteraties is het mogelijk om voor elk vlak de kost te berekenen. Het splitvlak met de laagste kost partitioneert de driehoeken in twee partities die vervolgens opnieuw recursief worden gepartitioneerd volgens hetzelfde principe. Er wordt recursief verder gepartitioneerd tot een bepaalde stopconditie bereikt is.

De asymptotische complexiteit voor het bouwen van een BVH is $O(N \times log(N))$. Dit algoritme is in praktijk veel sneller dan de meest gebruikte technieken. [Wal07] rapporteert een ordegrootte sneller dan de standaard technieken. Dit terwijl de render performantie maar een paar procenten slechter is.

4.1.2 Resultaten

Omdat deze techniek werkt voor elk soort animatie gebruiken wij de techniek voor skeletgebaseerde animaties. We hebben de techniek geïmplementeerd zonder gebruik te maken van SIMD extensies. Dit leidt tot een minder performante implementatie. Het aantal bins is 16 omdat dit volgens [Wal07] de beste afweging is qua rendertijd-bouwtijd. We stoppen met recursief partitioneren als het aantal driehoeken twee is.

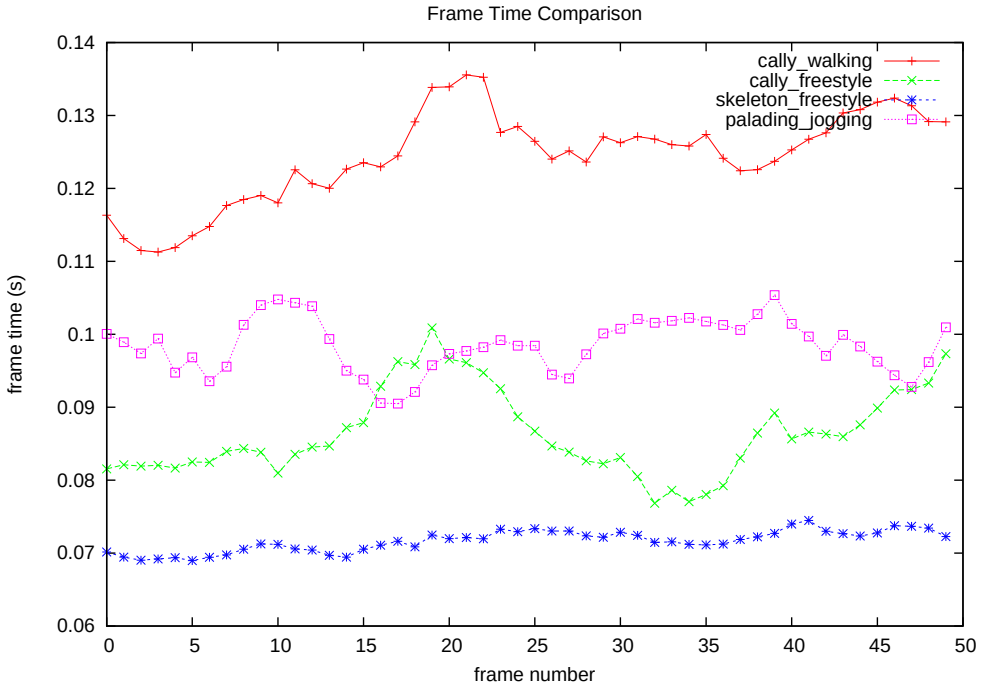
De BVH wordt herbouwd elk frame met als basis een reeks driehoeken zonder extra informatie (*polygon soup*). Alle parameters zijn gelijk voor elk type animatie.

Alle resultaten in dit hoofdstuk zijn verkregen door te renderen aan een resolutie van 512 x 512 pixels. De *time-to-image* schommelt tussen 0.08s en 0.13s[1] (zie figuur 4.2). De time-to-image bestaat uit het skinnen van de animatie, het bouwen van de BVH en het

[1]Alle gerapporteerde tijden zijn iets slechter dan in werkelijkheid. Dit omdat om de statistieken te verzamelen er extra code wordt uitgevoerd in de meest performantie gevoelige delen van onze ray tracer.

Animatie	Frames Per Seconde (FPS)
Cally Walking	11.55
Cally Freestyle	11.21
Skelet Freestyle	13.07
Paladin Jogging	9.86

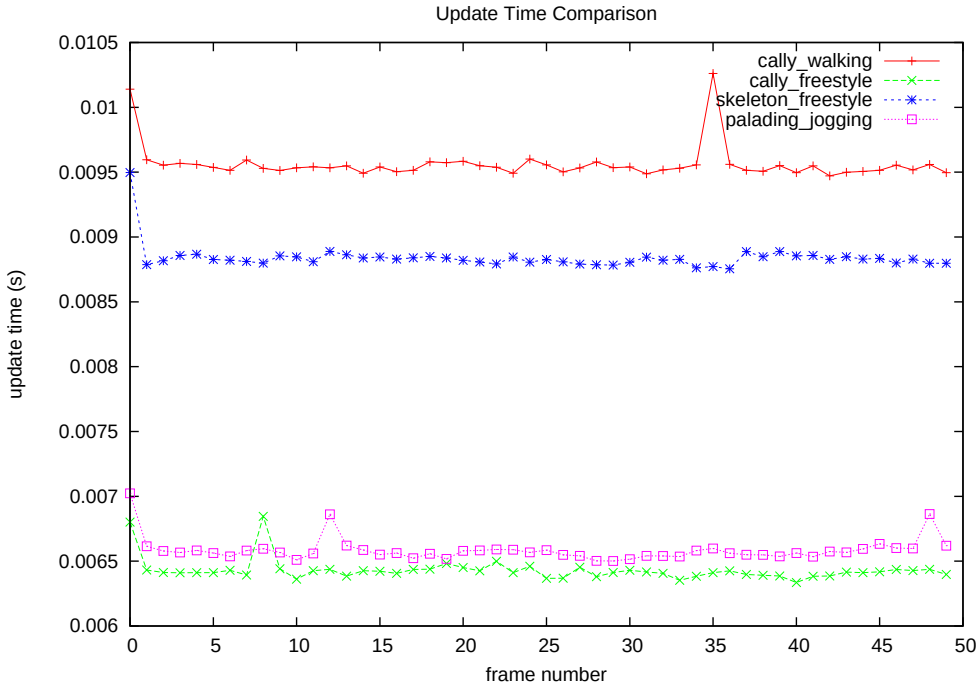
Tabel 4.1: Framerates voor de verschillende animaties met de fast rebuild techniek.



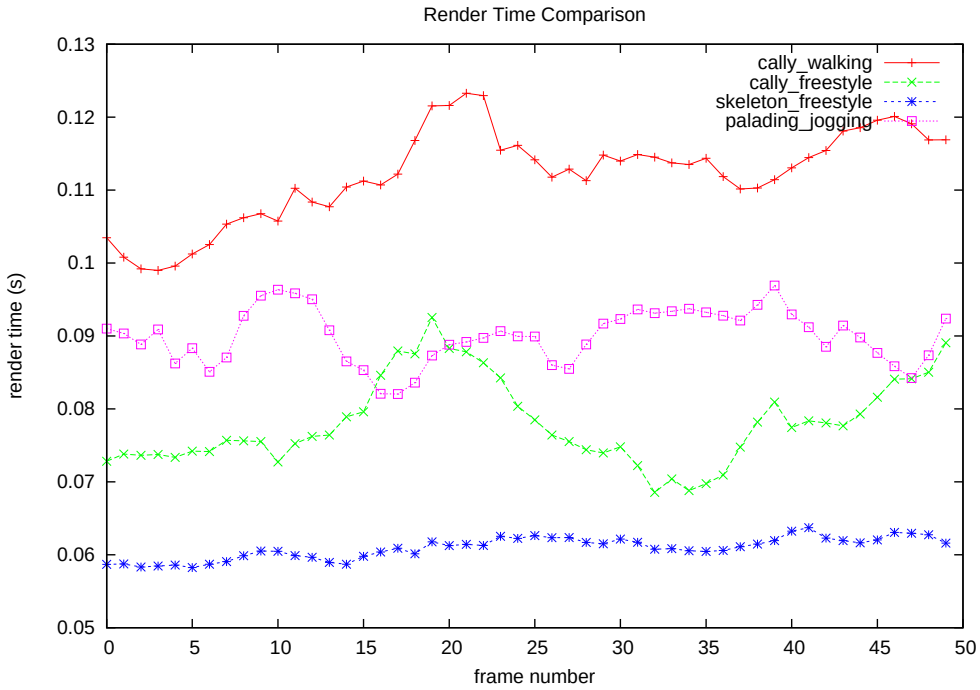
Figuur 4.2: Time-to-image voor de 4 testanimaties met de fast rebuild techniek.

renderen van het frame (met shading). De tijd nodig om een frame te encoderen naar het PNG bestandsformaat hebben we niet meegerekend. De gemiddelde framerates voor elke animatie zijn samengevat in tabel 4.1.

De bouwtijden (zie figuur 4.3) fluctueren niet sterk van frame tot frame. De bouwtijden zijn enkel afhankelijk van de animatie. Na analyse van onze implementatie hebben we gemerkt dat het gros van de tijd wordt besteed aan het accumuleren van de bounding boxes. Niet toevallig de code die [Wal07] aanraadt om in SIMD te implementeren.



Figuur 4.3: Bouwtijden voor de 4 testanimaties met de fast rebuild techniek.



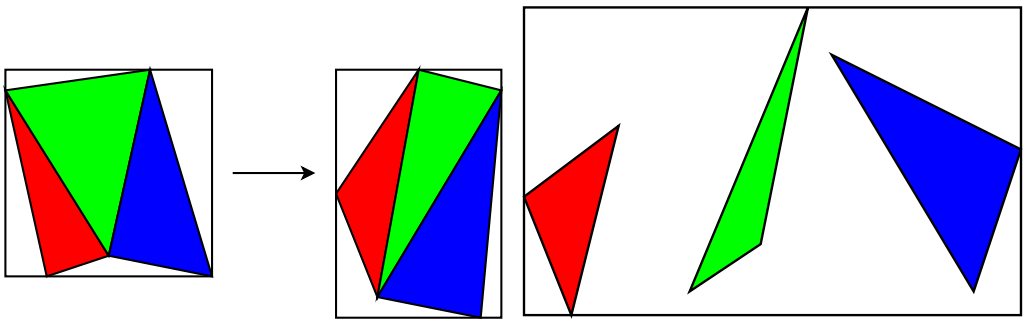
Figuur 4.4: Rendertijden voor de 4 testanimaties met de fast rebuild techniek.

4.2 Refitting

Een alternatief voor alles herbouwen elk frame is refitting. Deze techniek is voorgesteld door Wald en Shirley [WBS07] voor interactieve ray tracing van deformable scenes. Bij refitting wordt er maar één datastructuur gebouwd en deze vervormen we elk frame. Deze techniek is algemeen toepasbaar voor elk type deformable scenes. Een voordeel is dat de animatie niet op voorhand moet gekend te zijn.

4.2.1 Methode

De techniek start met het bouwen van een BVH over de geometrie van één van de frames. Het eerste frame wordt meestal gebruikt en geeft in de praktijk goede resultaten. Voor een skeletgebaseerde animatie is de rustpose een goede kandidaat (zie tabel 4.2). Dit omdat alle andere poses een vervorming zijn van de rustpose en dus de rustpose een soort 'gemiddelde' is. Wanneer de vertex posities wijzigen wordt voor elke driehoek een nieuwe bounding box berekent (zie figuur 4.5(a)), dit zijn de bladknopen van de BVH. Daarna wordt de BVH recursief doorlopen om de bounding box van de interne knopen aan te passen. Ook de traversal informatie van elke knoop moet worden aangepast zodat de BVH altijd front-to-back kan worden doorlopen.



Figuur 4.5: Links: Elk frame worden de bounding boxes herberekend om de veranderende geometrie te blijven omsluiten. Rechts: Veel beweging of niet-deformable scenes geeft aanleiding tot het 'opblazen' van bounding boxes waardoor de traversal performantie sterk daalt.

Deze techniek kan leiden tot zwaar performantieverlies wanneer de beweging in de animatie te incoherent is of niet deformable (zie figuur 4.5(b)). Gelukkig is de beweging van skeletgebaseerde animaties semi-hiërarchisch en deformable dus goede resultaten zijn te verwachten.

Refitting van een BVH zal voor niet triviale scenes altijd sneller zijn dan rebuilding. De reden hiervoor is dat refitting tijdscomplexiteit $O(N)$ heeft en rebuilding tijdscom-

	BVH over frame		BVH over rustpose	
Animatie	BBox tests/hit	Tri tests/hit	BBox tests/hit	Tri tests/hit
Cally Walking	95.92	7.6	77.12	6.53
Cally Jogging	89.6	7.2	75.67	6.9
Cally Freestyle	93.54	9.78	78.45	8.27
Skelet Walking	137.81	11.93	125.72	11.97
Skelet Jogging	127.91	11.86	121.25	12.16
Skelet Freestyle	161.15	11.63	143.11	12.3
Paladin Walking	78.23	7.29	62.44	6.91
Paladin Jogging	77.06	8.31	65.9	7.71
Paladin Freestyle	66.56	7.61	69.74	10.1

Tabel 4.2: Vergelijking van de renderperformantie van refitting voor een BVH gebouwd over het eerste frame en een BVH gebouwd over de rustpose. Uit onze resultaten is duidelijk te zien dat de performantie iets beter is voor een BVH gebouwd over de rustpose voor het gros van de scenes.

Animatie	Frames Per Seconde (FPS)
Cally Walking	12.44
Cally Freestyle	11.37
Skelet Freestyle	14.36
Paladin Jogging	10.35

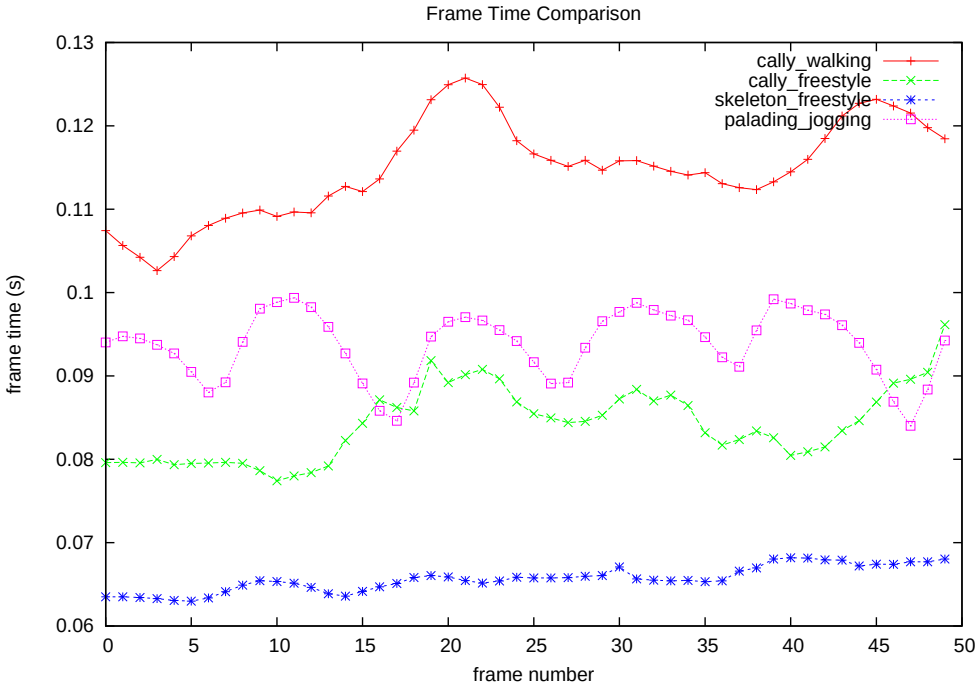
Tabel 4.3: Framerates voor de verschillende animaties met de refitting techniek.

plexiteit $O(N \times log(N))$ met N het aantal knopen in de BVH tree. Of er ook een winst is in traversal performantie voor skeletgebaseerde animaties zullen we verder onderzoeken.

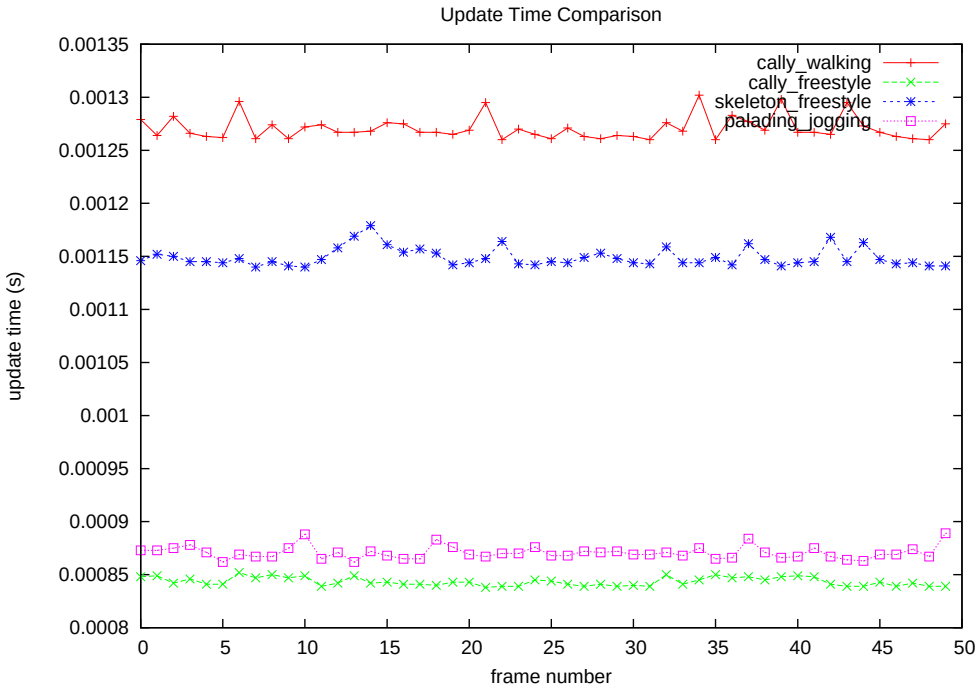
4.2.2 Resultaten

Voor het bouwalgoritme gebruiken we hetzelfde als besproken bij fast rebuilds maar met 1024 bins. Door het groot aantal bins neemt het bouwen van de BVH over de rustpose al gauw enkele seconden in beslag.

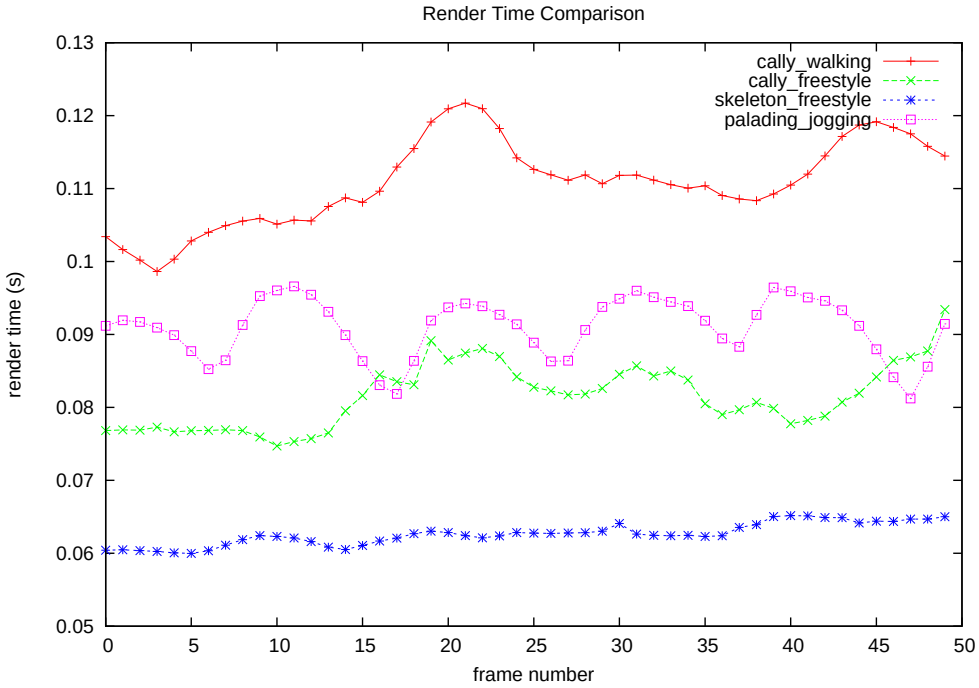
Voor de frame rates zie tabel 4.3. De time-to-image voor de eerste vijftig frames is gegeven in figuur 4.6. Grafiek 4.7 geeft de tijden nodig om de BVH structuur te refitten. De renderperformantie is af te lezen in grafiek 4.8.



Figuur 4.6: Time-to-image voor de 4 testanimaties met de refitting techniek.



Figuur 4.7: Bouwtijden voor de 4 testanimaties met de refitting techniek.



Figuur 4.8: Rendertijden voor de 4 testanimaties met de refitting techniek.

4.3 Motion Decomposition & Fuzzy Versnellingsstructuren

Het gebruik van *fuzzy* versnellingsstructuren voor skeletgebaseerde animaties is voorgesteld door [GFSS06]. Het idee is om het herbouwen of aanpassen van een versnellingsstructuur grotendeels te vermijden. De beweging wordt gecompenseerd door de straal te transformeren in plaats van de geometrie.

4.3.1 Methode

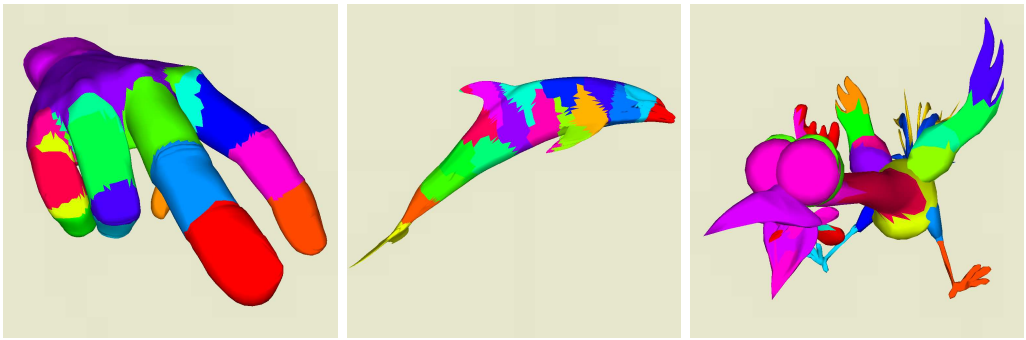
Motion Decomposition

In beweging van deformable meshes kunnen twee componenten onderscheiden worden:

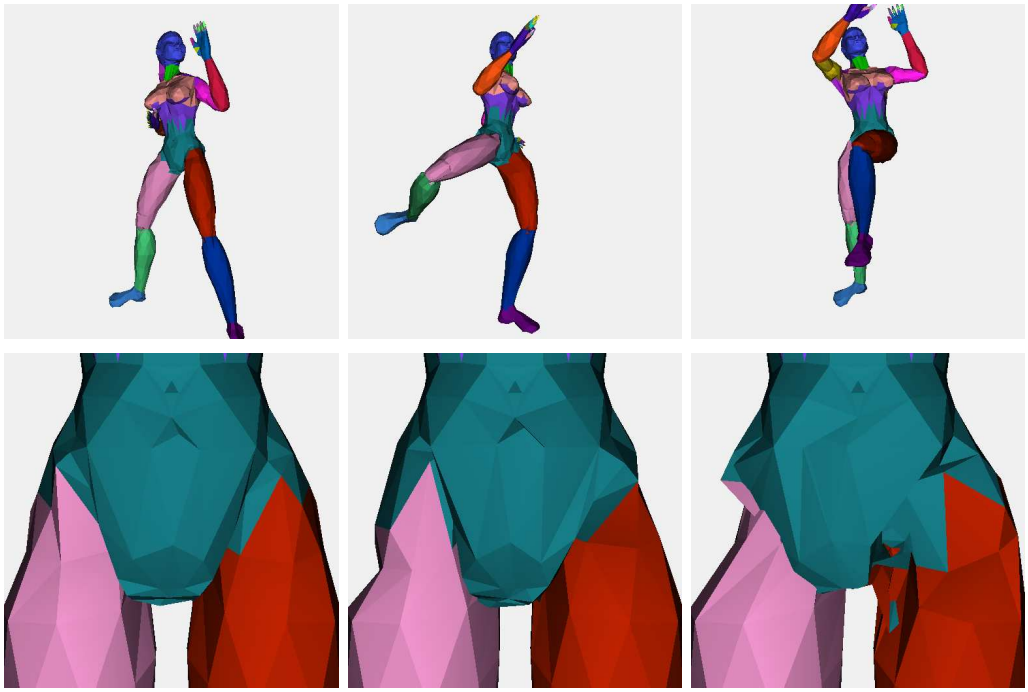
Coherente Beweging Beweging die kan beschreven worden door affiene transformaties. Coherente beweging wordt in de literatuur soms affiene beweging genoemd.

Residuele beweging Beweging die overblijft na het aftrekken van de affiene component. Dit is vaak een kleine lokale beweging.

Dit principe noemt *motion decomposition* en wordt beschreven in [GFW+06]. In de paper wordt een algoritme beschreven om de driehoeken van een deformable mesh te groeperen in clusters (zie figuur 4.9). Alle driehoeken van dezelfde cluster ondergaan dezelfde affiene transformatie. Het doel is om clusters te vinden zodanig dat de residuele beweging van elke driehoek minimaal is.



Figuur 4.9: Het clusteringalgoritme van [GFW+06] probeert de driehoeken van een deformable mesh te groeperen in clusters van coherente beweging. (bron afbeeldingen: [GFW+06])



Figuur 4.10: Boven: Enkele frames uit de Cally Freestyle animatie. Onder: De overeenkomstige residuele beweging in het lokaal assenstelsel rond het bekken van Cally. Het is duidelijk dat hoewel de globale vervorming groot is, de vervorming in het lokale assenstelsel van de botten relatief klein is.

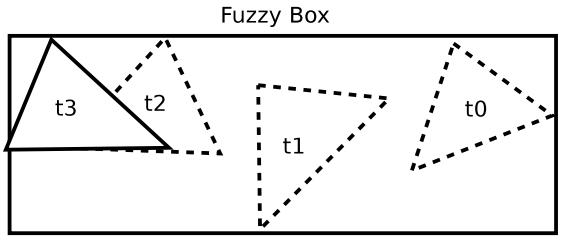
Skeletgebaseerde animaties.

Bij skeletgebaseerde animaties is het partitioneren in clusters makkelijker. Elk bot suggereert een cluster door delen van de mesh die door hetzelfde bot worden beïnvloed samen te nemen. Als alle vertices van een skeletgebaseerde animatie slechts beïnvloed worden door slechts één bot dan is er enkel affiene beweging.

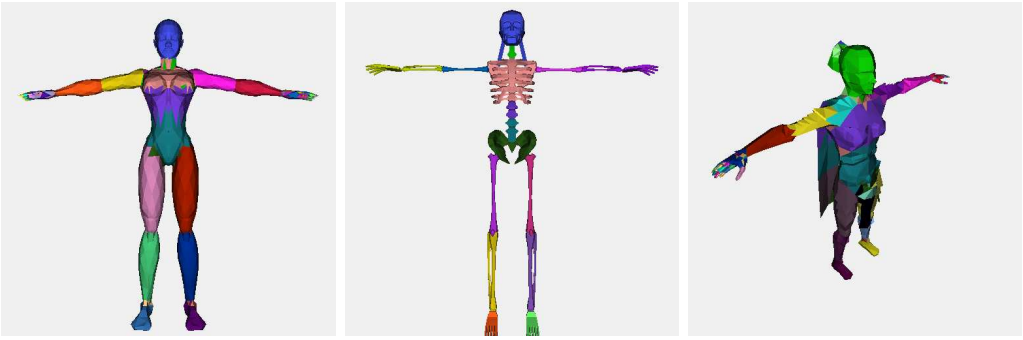
[GFSS06] merkt op dat zelfs als vertices worden beïnvloed door meerdere botten (zoals in standaard skinning), de beweging van een vertex in het lokale assenstelsel van een *goed* gekozen bot zeer klein is (zie figuur 4.10). De beweging van een vertex in het lokaal assenstelsel van een bot is de residuele beweging van de vertex.

Omdat de lokale beweging van een vertex zeer klein is, is het mogelijk om hierover een *fuzzy box* te bouwen. Dat is een bounding box die alle posities van de vertex omsluit voor de hele animatie (zie figuur 4.3.1) . De fuzzy box blijft dus geldig voor de hele animatie. De fuzzy box van een driehoek is de unie van de de fuzzy boxes van zijn hoekpunten.

Fuzzy boxes zijn een uitstekende kandidaat om een versnellingsstructuur over te bou-



Figuur 4.11: De driehoek valt binnen zijn fuzzy box op alle mogelijke tijdstippen van de animatie.



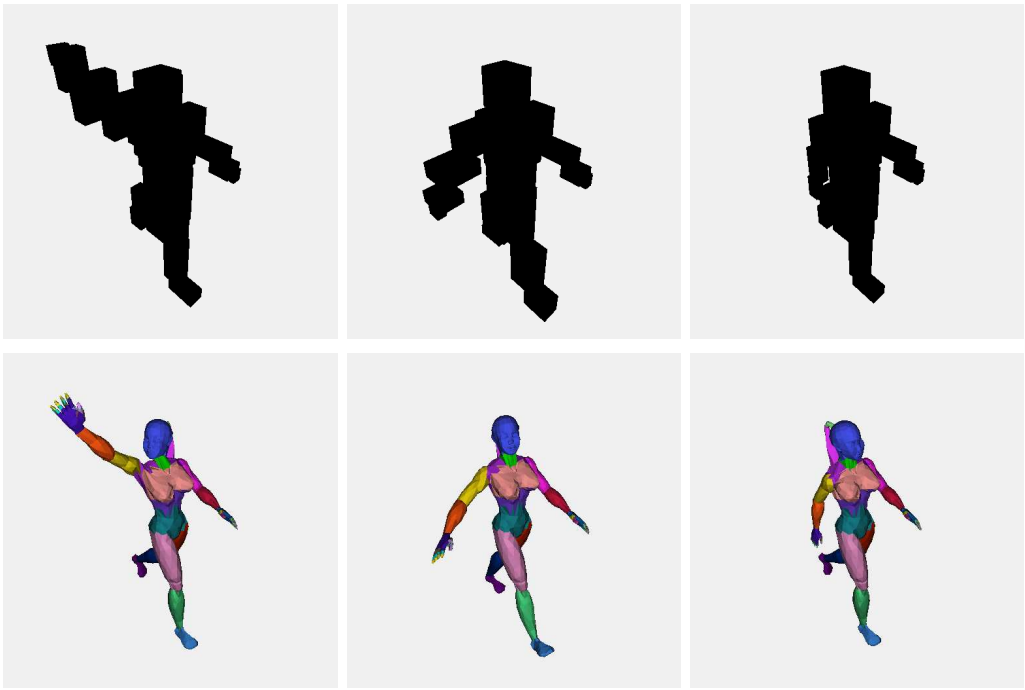
Figuur 4.12: De rustposes van onze testkarakters. De verschillende kleuren duiden de verschillende clusters aan.

wen die geldig blijft gedurende de volledige animatie. We noemen deze structuur de *fuzzy structuur* of *lokale-structuur*. Een nadeel van het bouwen van een versnellingsstructuur over fuzzy boxes is dat een fuzzy box zijn driehoek niet nauw omsluit. Hierdoor worden de surface areas van de structuur groter wat een verhoogde traversal kost met zich meebrengt. Er is één fuzzy structuur per bot van het skelet. De fuzzy structuren vatten de residuele beweging van de animatie.

Een *cluster* noemen we de verzameling driehoeken die zijn toegewezen aan hetzelfde bot (zie figuur 4.12). Elk frame wordt voor elke cluster de bounding box berekent. Over die bounding boxes bouwen we elk frame een nieuwe versnellingsstructuur, de *top-level versnellingsstructuur* (zie figuur 4.13). Het bouwen van de top-level structuur is niet rekenintensief omdat het aantal clusters maximaal gelijke is aan het aantal botten. Het aantal botten is voor de meeste animaties kleiner dan honderd[2]. De top-level structuur vat de affiene beweging van de animatie.

De top-level structuur en de fuzzy structuren vormen samen de basis voor ray tracing van de animatie. Een straal doorloopt eerst de top-level structuur. Voor elke cluster die geraakt wordt door de straal, transformeren we de straal naar het lokale

[2]Als we bijvoorbeeld mensen fysisch correct zouden modelleren dan is er nog geen probleem. Een volwassen mens heeft 206 botten en een baby 231.



Figuur 4.13: Boven: de bounding boxes van elke cluster, over deze bounding boxes wordt de top-level versnellingsstructuur gebouwd. Onder: de overeenkomstige frames uit de Cally Walking animatie.

coördinaatsysteem van de cluster. Deze transformatie is de affiene transformatie die het bot ondergaat waaraan de cluster is toegewezen. Vervolgens doorloopt de getransformeerde straal de fuzzy structuur. Dit traversal algoritme is veel complexer dan bij de andere methodes en wordt *two-level ray tracing* genoemd in de literatuur. Het concept werd voor het eerst geïntroduceerd in [WBS03].

In een preprocessing stap worden de fuzzy boxes berekend voor elke driehoek. Het is belangrijk dat de fuzzy boxes de driehoeken omsluiten gedurende de volledige animatie. Niet *conservatieve* fuzzy boxes kan leiden tot render artefacten (zie figuur 4.14) . Als de fuzzy boxes te groot zijn verlaagd de ray tracing performantie. Om de fuzzy box van elke vertex te bepalen worden verschillende poses van de animatie *gesampled*. De fuzzy box van een vertex is de unie van de fuzzy box van alle *samples*. De fuzzy box van elke driehoek is de unie van de fuzzy boxes van zijn vertices. Om het optimale bot te vinden voor elke driehoek wordt een exhaustieve zoektocht gedaan. De fuzzy box wordt berekend voor elke driehoek in het lokale coördinaatsysteem van elk bot. Een driehoek wordt toegekend aan het bot waarvoor zijn fuzzybox een minimale oppervlakte heeft.

4.3.2 Resultaten

Het berekenen van de fuzzy boxes doen we op twee manieren:

Pose Space Sampling Elk bot kan een willekeurige rotatie maken rond zijn ouder bot. Dit *sampled* de volledige pose ruimte van een animatie. Om kleinere fuzzy boxes te bekomen beperken we de rotatie van een bot rond zijn ouder bot tot een hoek van 90 graden.

Animation Cycle Sampling Hier nemen we een aantal samples uit alle mogelijke animaties sequenties (bijvoorbeeld wandelen, lopen, wuiven, ...).

Animation Cycle Sampling geeft betere resultaten dan Pose Space Sampling (zie tabel 4.4). De reden hiervoor is dat Pose Space Sampling gebruik maakt van samples die waarschijnlijk nooit voorkomen in de eigenlijke animatie en dus de fuzzy boxes overschat. Het nadeel van Animation Cycle Sampling is dat de verschillende animaties op voorhand moeten gekend zijn.

[GFSS06] maakt gebruik van kd-trees bij het implementeren van deze methode, wij hebben gebruik gemaakt van BVHs. Voor de fuzzy structuren bouwen we eenmalig BVHs voor elke cluster met de fast rebuild techniek. We gebruiken 1024 bins zodat het berekenen van de lokale structuren enkele seconden in beslag neemt.

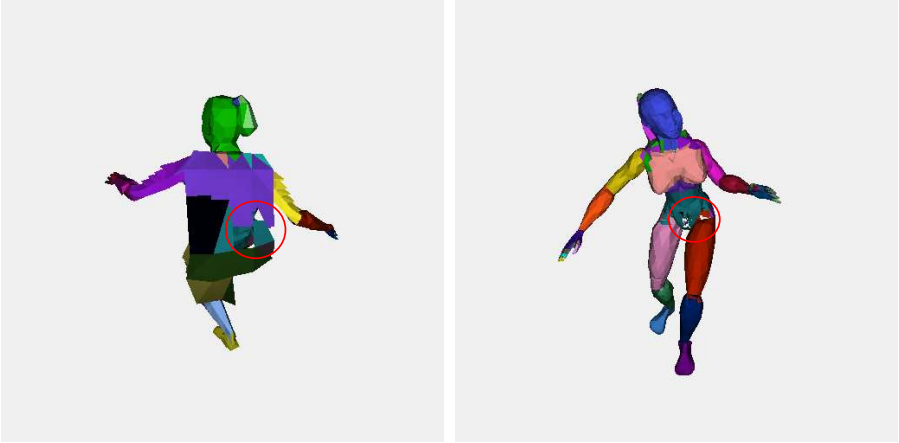
De top-level structuur is ook een BVH die we opbouwen met de median split techniek. Deze techniek is sneller dan de fast rebuilds techniek maar minder performant bij traversal. Dat vormt geen probleem omdat de top-level BVH weinig knopen heeft zodat de traversaltijd te verwaarlozen is t.o.v. de traversal van de fuzzy structuren.

Een kleine optimalisatie die we zelf bedacht hebben is het sorteren van de lokale structuren. Nadat de top-level structuur doorlopen is en we weten welke lokale structuren we vervolgens moeten doorlopen sorteren we eerst de lokale structuren. De structuur die het dichtst bij de oorsprong van de stralen ligt doorlopen we eerst.

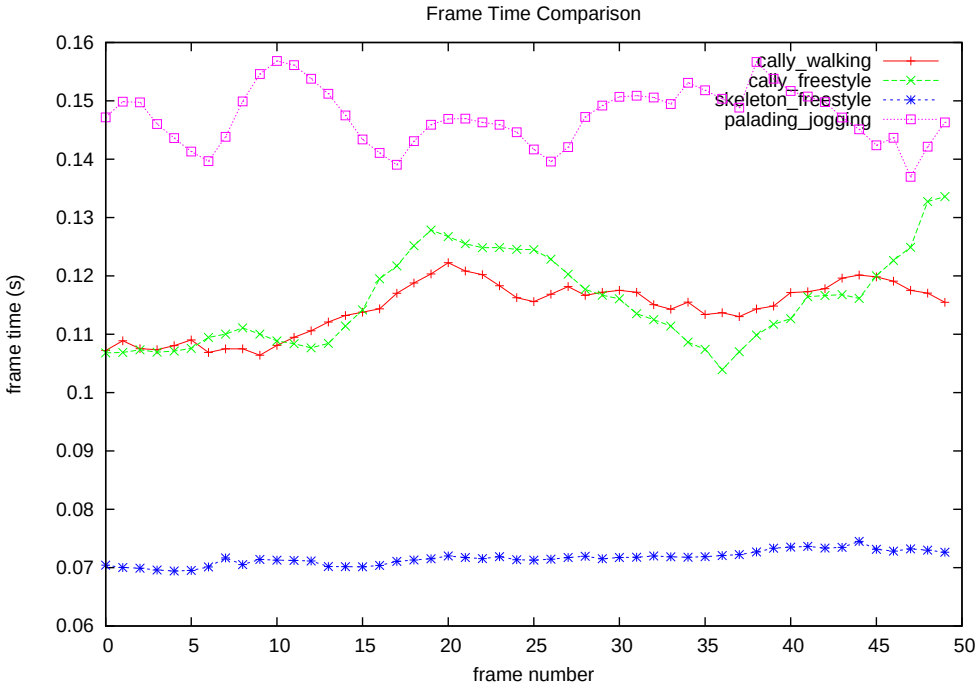
De tijden zijn af te lezen in grafieken 4.15, 4.16, 4.17. De framerates zijn samengevat in tabel 4.4. De fuzzy boxes zijn gesampled met animation cycle sampling omdat dit de beste performantie geeft.

Animatie	FPS (Pose space sampling)	FPS (Animation cycle sampling)
Cally Walking	8.28	8.76
Cally Freestyle	8.76	8.78
Skelet Freestyle	13.77	13.81
Paladin Jogging	6.11	6.97

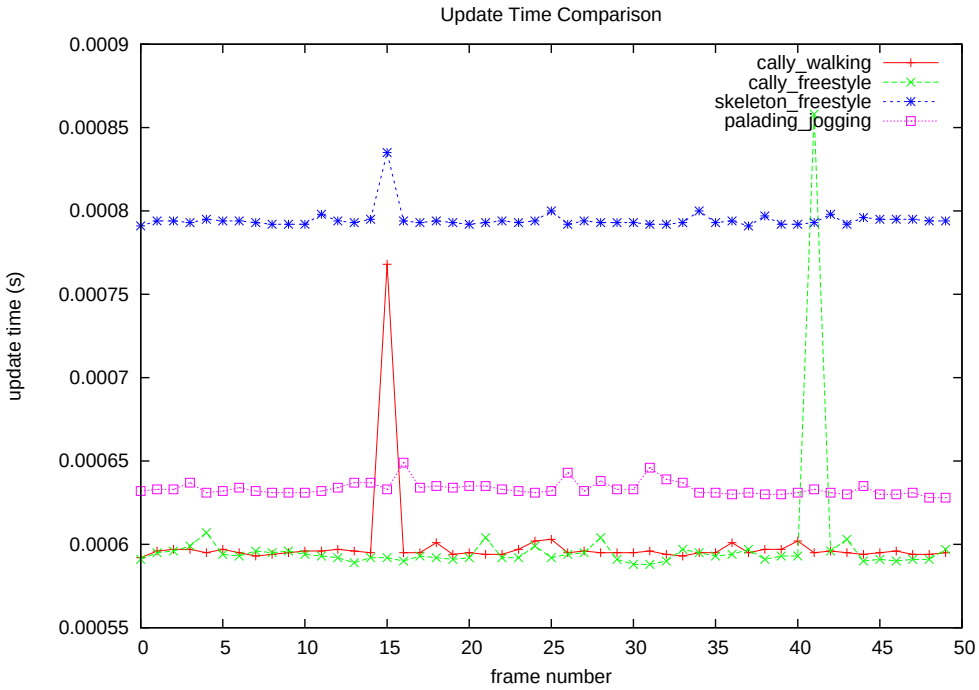
Tabel 4.4: Framerates voor de verschillende animaties met de motion decomposition & fuzzy techniek. De waarden tonen aan dat Animation cycle sampling betere resultaten geeft dan Pose space sampling.



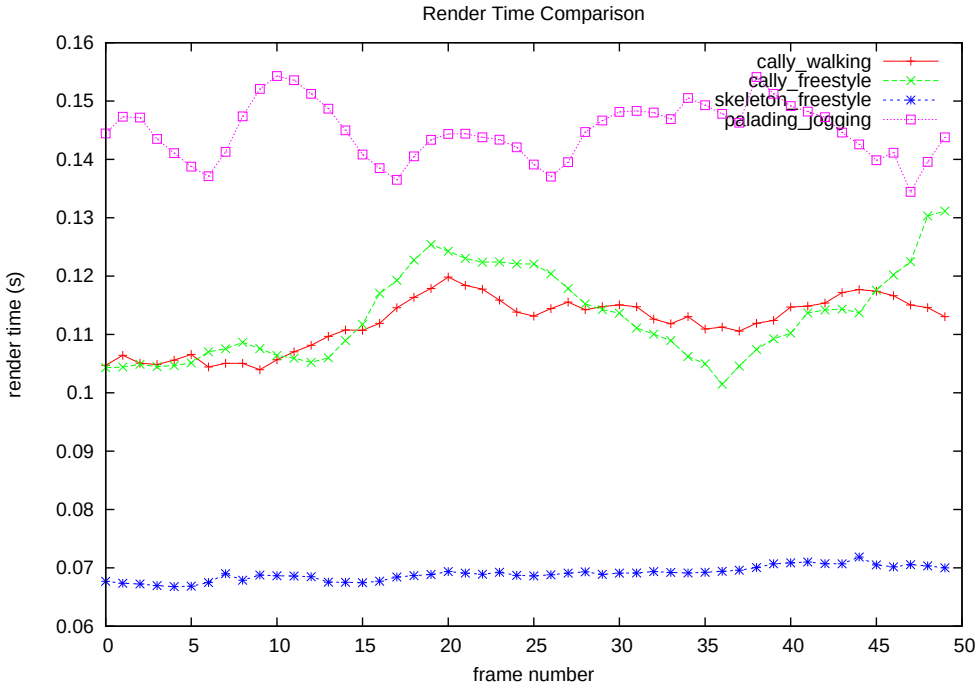
Figuur 4.14: Door te weinig samples te nemen zijn de fuzzy boxes niet meer conservatief. Hierdoor kunnen render artefacten ontstaan. In onze experimenten ontstaan render artefacten door te samplen met minder dan 10 samples per vertex.



Figuur 4.15: Time-to-image voor de 4 testanimaties met de fuzzy techniek.



Figuur 4.16: Bouwtijden voor de 4 testanimaties met de fuzzy techniek.



Figuur 4.17: Rendertijden voor de 4 testanimaties met de fuzzy techniek.

Animatie	Frames Per Seconde (FPS)
Cally Walking	9.50
Cally Freestyle	9.22
Skelet Freestyle	13.50
Paladin Jogging	7.72

Tabel 4.5: Framerates voor de verschillende animaties met de motion decomposition & refitting techniek.

4.4 Motion Decomposition & Refitting

De laatste methode hebben we zelf bedacht en is een combinatie van de fuzzy techniek en refitting.

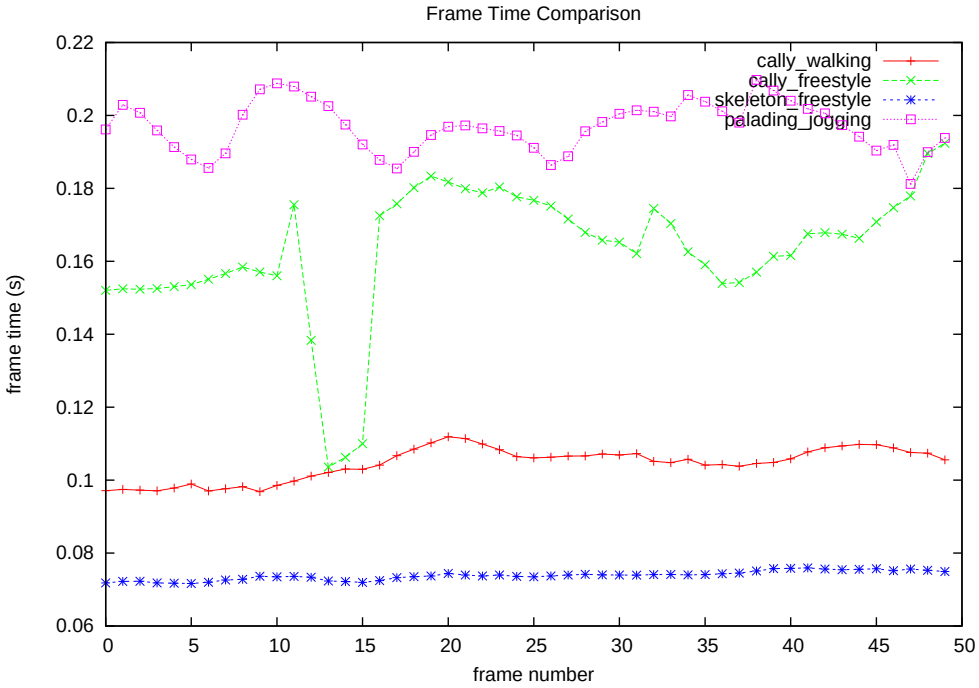
4.4.1 Methode

Het probleem met motion decomposition en fuzzy versnellingsstructuren zijn de fuzzy boxes. Door de fuzzy boxes heeft de BVH veel overlap tussen zijn knopen waardoor de traversal performantie gevoelig daalt. Bij refitting kan er een probleem ontstaan wanneer de vervorming t.o.v. de rustpose te sterk is. De topologie van de BVH wijzigt wat ook leidt tot verlaagde traversal performantie.

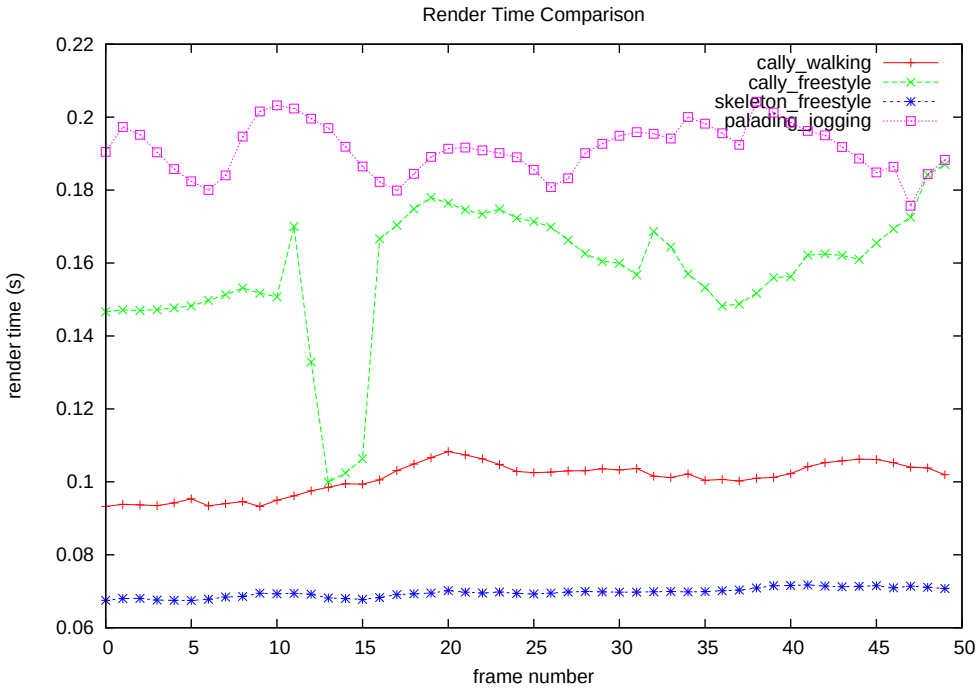
Het principe is analoog aan de vorige methode maar we vervangen de fuzzy boxes door nauw aansluitende bounding boxes. Deze bounding boxes moeten nu elk frame herberekend worden om de residuele beweging van elke driehoek correct te vatten. In vergelijking met refitting is de vervorming van frame tot frame veel kleiner. Dit omdat de residuele beweging van een willekeurige driehoek zeer gering is. We verwachten dan ook dat de kwaliteit van de BVHs niet sterk zal verschillen van frame tot frame. Het probleem van de fuzzy boxes is hiermee opgelost en de topologie van de BVH blijft zeer goed behouden. De prijs de we hiervoor betalen is de extra refitting stap en het complexere traversal algoritme.

4.4.2 Resultaten

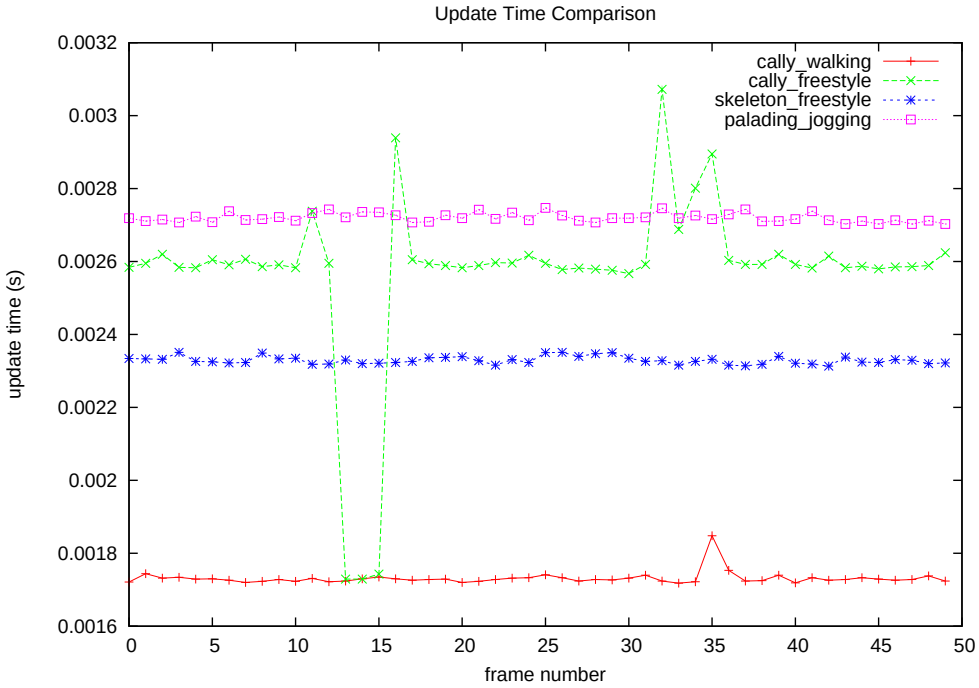
De resultaten zijn samengevat in de grafieken 4.18, 4.20, 4.19 en tabel 4.5.



Figuur 4.18: Time-to-image motion decomposition & refitting techniek.



Figuur 4.19: Rendertijden motion decomposition & refitting techniek.



Figuur 4.20: Bouwtijden decomposition & refitting techniek.

4.5 Besluit

In dit hoofdstuk hebben we vier verschillende technieken besproken namelijk fast rebuilds, refitting, motion decomposition & fuzzy versnellingsstructuren en motion decomposition & refitting. Uit de resultaten blijkt duidelijk dat alle technieken in staat zijn om interactieve ray tracing performantie te behalen. In het volgende hoofdstuk maken we een grondige vergelijking van de vier technieken.

Hoofdstuk 5

Vergelijking van de Technieken

> However beautiful the strategy, you should occasionally look at the results.
> - Winston Churchill

Nu we verschillende technieken hebben besproken om skeletgebaseerde animaties te renderen wordt het tijd voor een grondige vergelijking. In sectie 5.1 bespreken we de time-to-image van onze test scenes. Om de resultaten te verklaren bespreken we specifiekere metrieken in secties 5.2 en 5.3. Onze conclusies formuleren we op het einde in sectie 5.4.

5.1 Time-to-Image

Uiteindelijk is de belangrijkste metriek de time-to-image. Wanneer ray tracing bijvoorbeeld gemeengoed wordt voor computerspelletjes dan is dit de prijs die elk frame betaald moet worden. Net zoals in vorig hoofdstuk bestaat de time-to-image uit de volgende tijden:

1. De tijd nodig voor het berekenen van de huidige pose. Dit komt neer op het skinning algoritme zoals besproken in sectie 2.2.
2. De bouw- of updatetijd van de versnellingsstructuur.
3. Het effectief renderen van het huidige frame.

In de praktijk hebben we gemerkt dat de tijd die het skinning algoritme vergt te verwaarlozen is t.o.v. de rendertijd en bouwtijd (minder dan 1%).

Voor de Cally Walking animatie (zie figuur 5.1) is er een strikte ordening tussen de verschillende technieken. Refitting is de snelste techniek en motion decomposition &

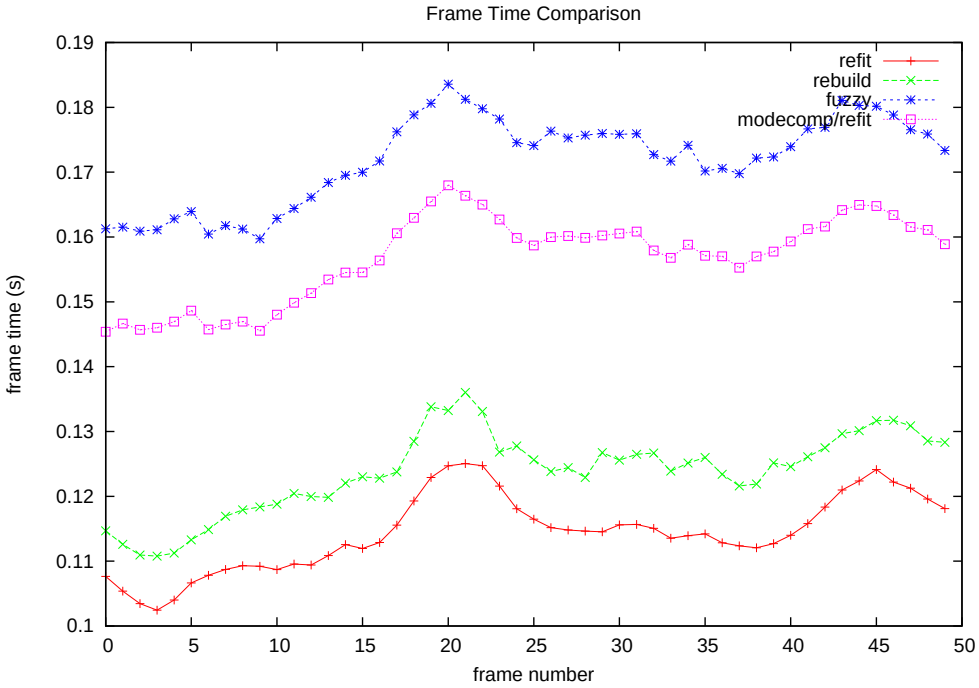
fuzzy versnellingsstructuren de traagste. De piek tussen frame 15 en 25 is wanneer Cally haar arm omhoog brengt om te beginnen wuiven. Deze piek is omdat hier de surface area van de bounding box van Cally maximaal is waardoor het renderen duurder wordt.

De Cally Freestyle animatie is complexer en dat is duidelijk te zien aan de sterke stijging van de time-to-image (zie figuur 5.2) tussen frame 10 en 30. Dit is het begin van de *karate kick* die Cally uitvoert. Vanaf frame 25 is rebuild sneller dan refitting. Vermoedelijk omdat de BVH zo sterk vervormd wordt waardoor de traversal performantie sterk afneemt.

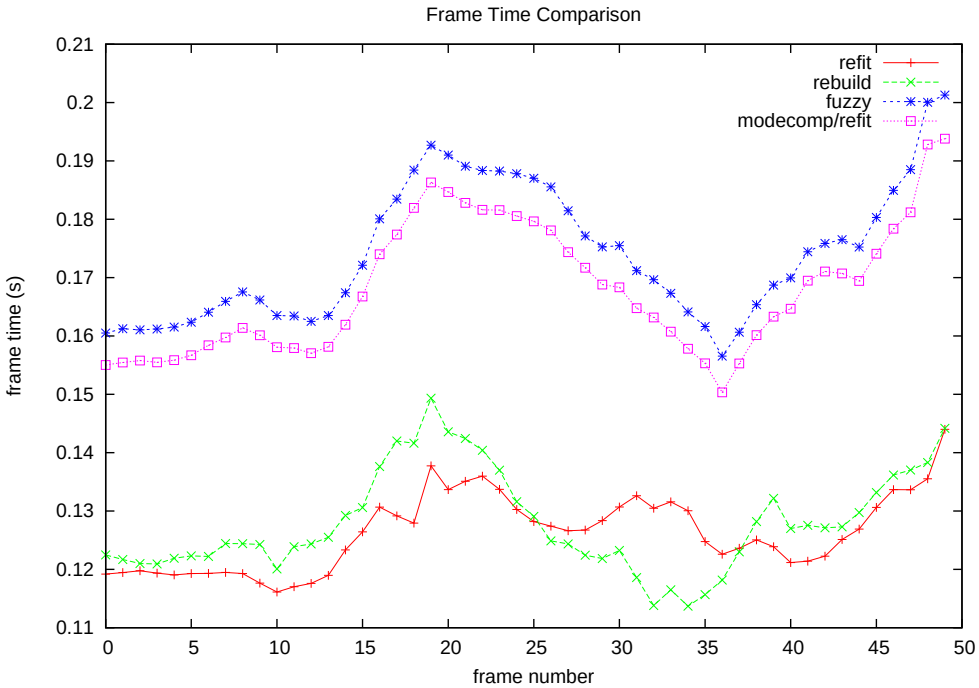
In grafiek 5.3 is te zien hoe motion decomposition & fuzzy versnellingsstructuren fast rebuilds en motion decomposition & refitting voorbij steekt. De reden hiervoor is dat voor de Skeleton Freestyle animatie de residuele beweging nihil is. Hierdoor is de BVH niet opgebouwd uit fuzzy boxes waardoor de rendertijd gevoelig stijgt. Motion decomposition & refitting moet nog een refitting doen elk frame en fast rebuilds een rebuild. Hierdoor steekt motion decomposition & fuzzy versnellingsstructuren de twee technieken voorbij omwille van de goedkope update. Refitting is de meest performante methode.

Refitting is iets sneller bij de Paladin Jogging methode (zie figuur 5.4). De motion decomposition technieken zijn ook voor deze scene minder performant.

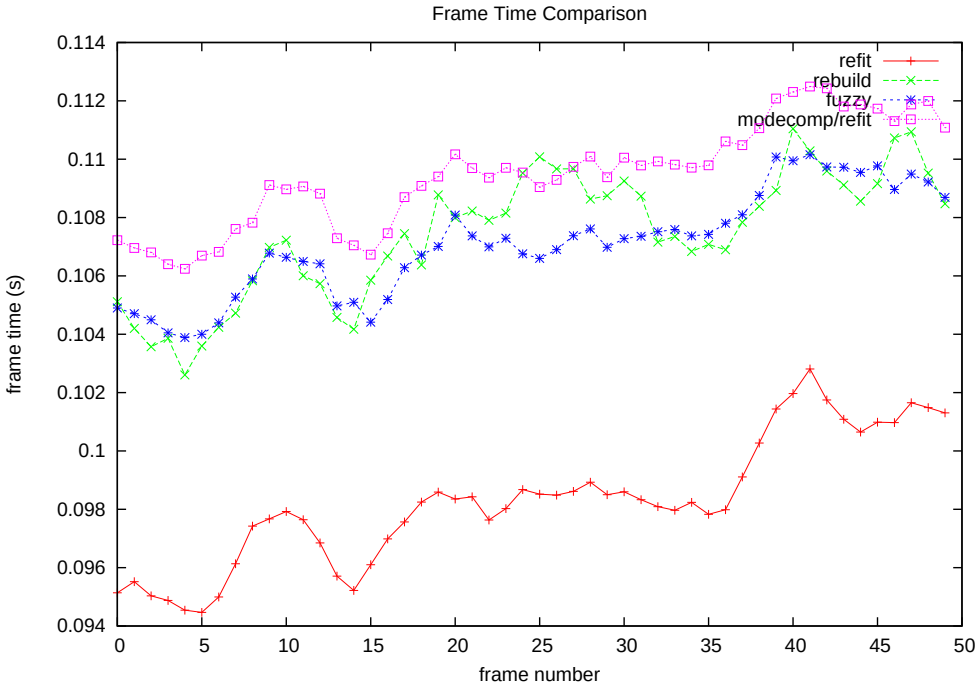
Op de Skeleton Freestyle animatie na is refitting en fast rebuilds niet te kloppen in time-to-image. De motion decomposition technieken vragen meestal 0.04 a 0.05 seconden extra per frame.



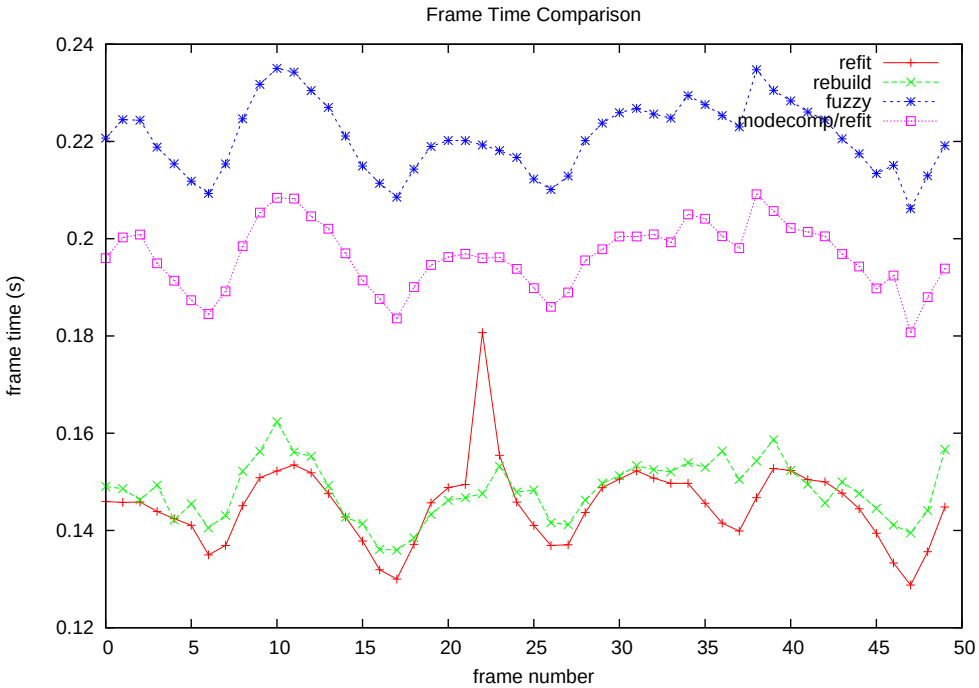
Figuur 5.1: Time-to-image voor de Cally Walking animatie.



Figuur 5.2: Time-to-image voor de Cally Freestyle animatie.



Figuur 5.3: Time-to-image voor de Skeleton Freestyle animatie.



Figuur 5.4: Time-to-image voor de Paladin Jogging animatie.

5.2 Rendertijden

Een deel van de time-to-image bestaat uit de rendertijd. Daarom geven we een overzicht van de rendertijden voor de vier verschillende methodes.

Voor de Cally Walking animatie zien we dat de slechtste rendertijd is voor de motion decomposition & fuzzy versnellingsstructuren techniek. Zeer logisch omdat deze techniek rendertijd opoffert om herbouwen van de versnellingsstructuur te vermijden. Motion decomposition & refitting presteert beter maar betaalt ook een performantiekost omwille van de straaltransformaties en de top-level BVH. Refitting en fast rebuild hebben ongeveer dezelfde rendertijden.

Refitting rendert bijna over de volledige Cally Freestyle animatie slechter dan rebuilds. Waarschijnlijk door de zware vervorming van de BVH (zie figuur 5.6). Ook hier zijn de technieken gebaseerd op motion decomposition slechter omwille van de complexere traversal algoritmes.

Bij de Skeleton Freestyle animatie overlappen de rendertijden van de motion decomposition technieken (zie figuur 5.7). Omdat er geen residuele beweging is zijn de fuzzy boxes gelijk aan de bounding boxes van de driehoeken. Hierdoor gebruiken de twee technieken dezelfde BVH met als gevolg dezelfde rendertijden. Fast rebuilds rendert iets sneller dan refitting.

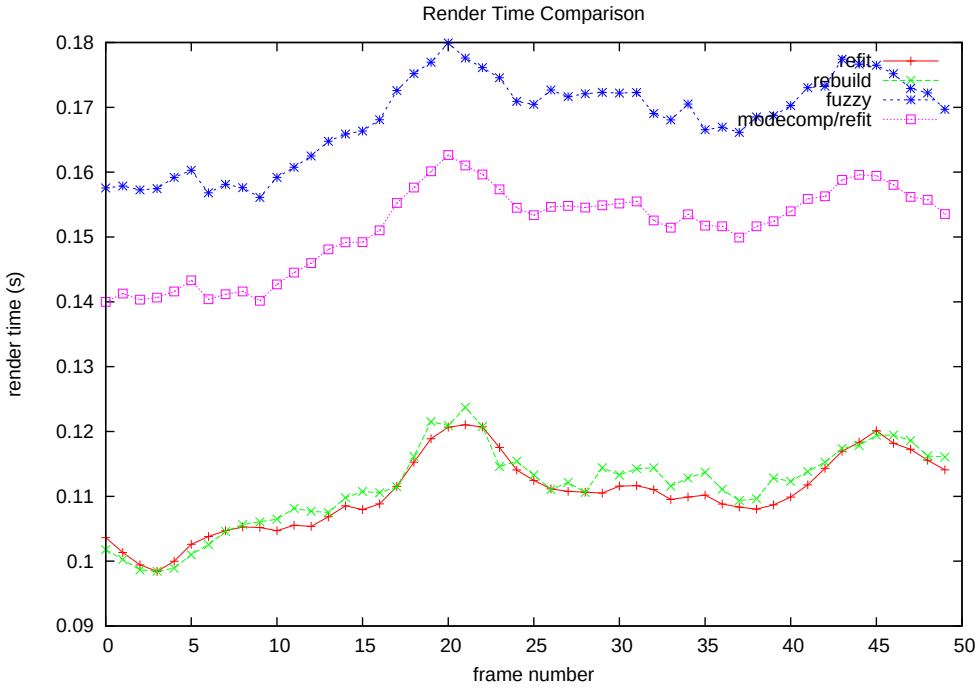
Bij de Paladin Jogging animatie (zie figuur 5.8) is fast rebuilds de snelste gevolgd door refitting.

Uit de rendertijden kunnen we besluiten dat voor elke animatie de fast rebuild de snelste rendertijden geeft. Een BVH geoptimaliseerd voor elk frame geeft algemeen de beste rendertijd.

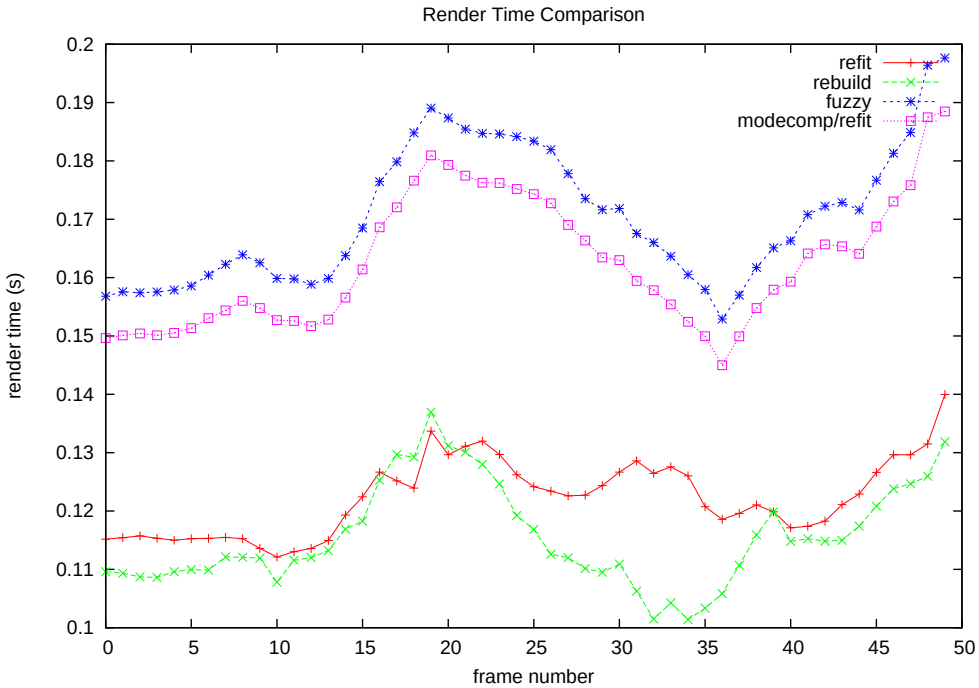
5.2.1 Aantal Straal-Driehoek Intersecties

Om dieper in te gaan op de rendertijden analyseren we het aantal straal-driehoek intersecties en verder het aantal straal-AABB intersecties (sectie 5.2.2). De rendertijden kunnen onderhevig zijn aan externe factoren (belasting van de computer, overblijvende data in de cache, ...). Het tellen van het aantal intersectietesten is volledig onafhankelijk van externe factoren.

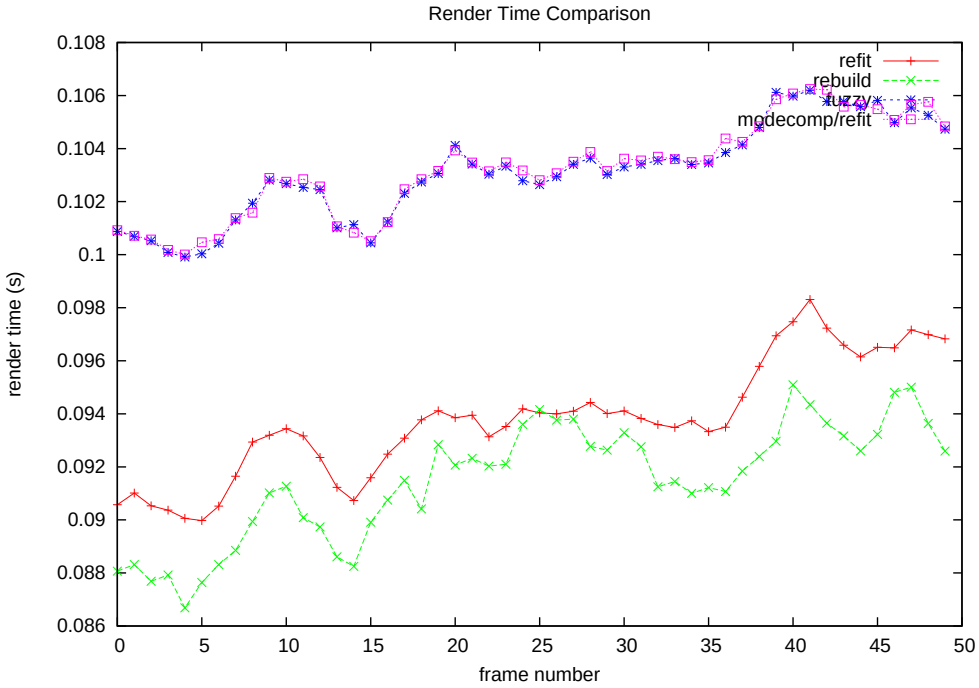
De motion decomposition technieken doen minder straal-driehoek intersecties dan refitting en fast rebuilds (zie figuur 5.9) voor de Cally Walking animatie. Dit komt door een implementatiedetail: omdat de fuzzy boxes niet nauw aansluiten bij de driehoeken is de kans dat een straal die de fuzzy box van een bladknoop raakt ook één van de ingesloten driehoeken raakt klein. Daarom berekenen we eerst de effectieve bounding box van de driehoeken en testen deze eerst nog met de straal voordat we effectief de driehoeken testen. Zo kunnen we veel straal-driehoek testen vermijden door een paar



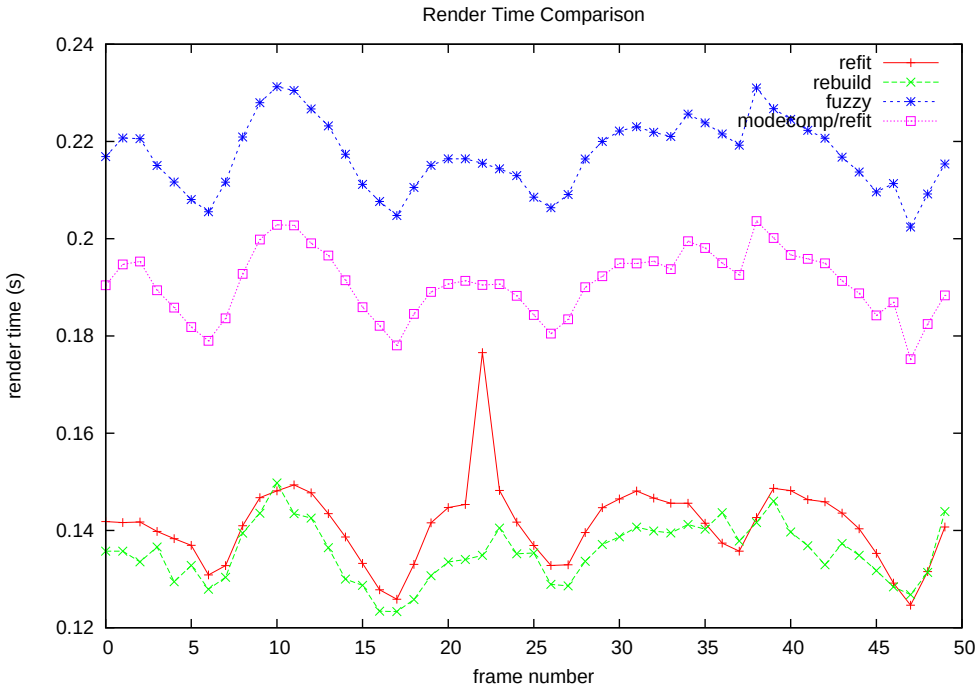
Figuur 5.5: De render tijden voor de Cally Walking animatie.



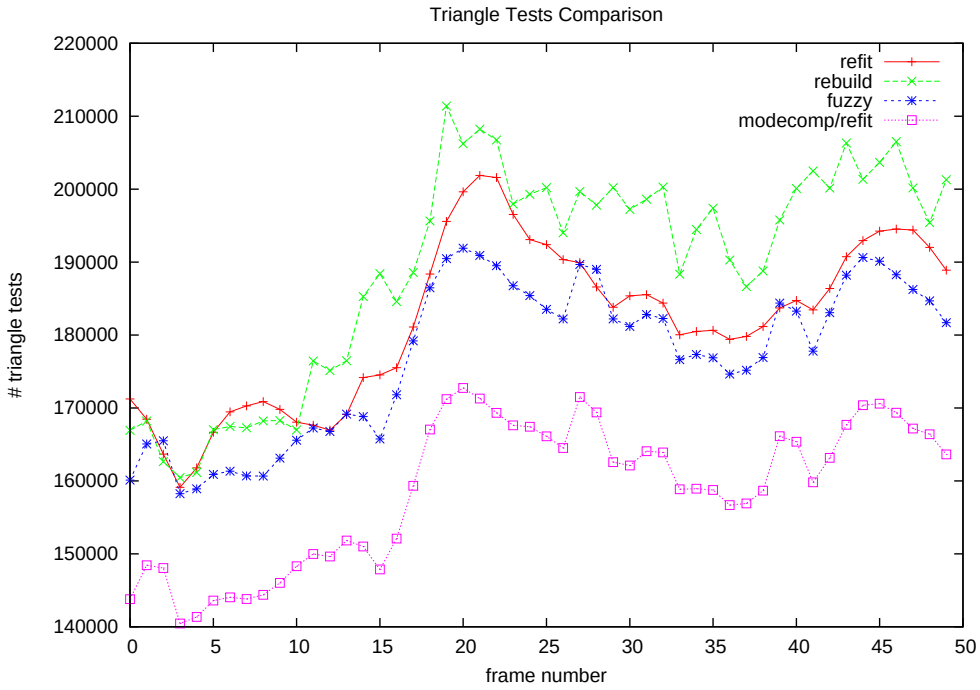
Figuur 5.6: De render tijden voor de Cally Freestyle animatie.



Figuur 5.7: De render tijden voor de Skeleton Freestyle animatie.



Figuur 5.8: De render tijden voor de Paladin Jogging animatie.



Figuur 5.9: Het aantal straal-driehoek intersecties voor de Cally Walking Animatie.

extra straal-AABB testen te doen. Refitting en rebuilds liggen dicht bij elkaar alleen doet refitting iets minder straal-driehoek testen.

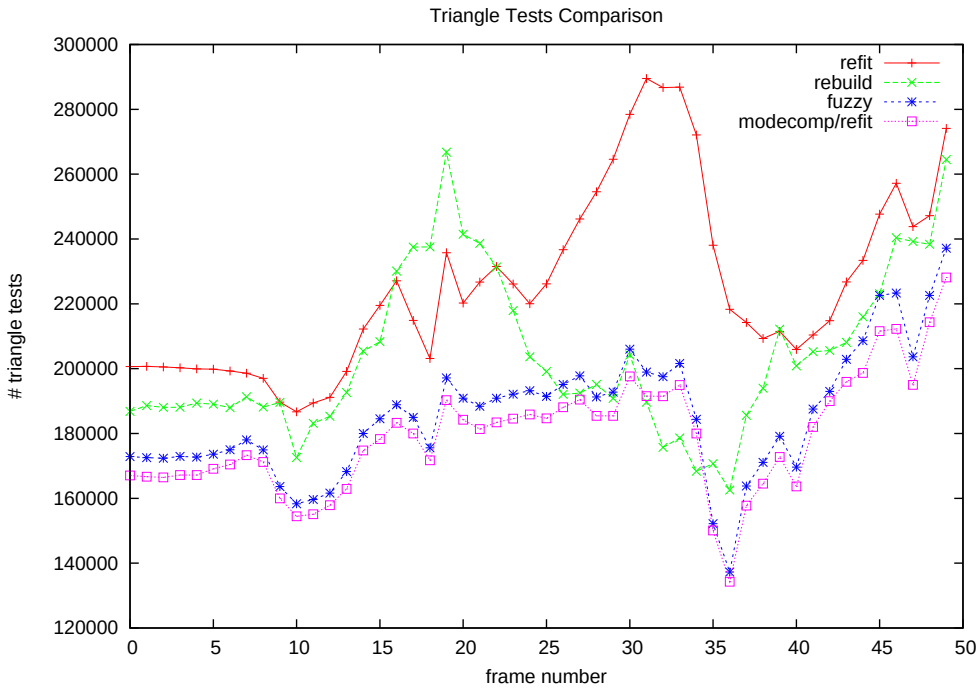
Ons vermoeden dat het performantieverlies van de refitting techniek voor de Cally Freestyle animatie te wijten is aan een sterk vervormde BVH wordt bevestigd in grafiek 5.10. De hoge piek in het aantal straal-driehoek intersecties wijst op een sterk gedegradeerde BVH. De motion decomposition technieken doen het minste aantal straal-driehoek intersecties.

Grafiek 5.11 bevestigt nogmaals dat de motion decomposition technieken dezelfde BVH gebruiken voor de Skeleton Freestyle animatie want het aantal straal-driehoek intersecties is exact gelijk. Het verloop voor het aantal straal-driehoek intersecties voor refitting en fast rebuilds is ongeveer gelijk.

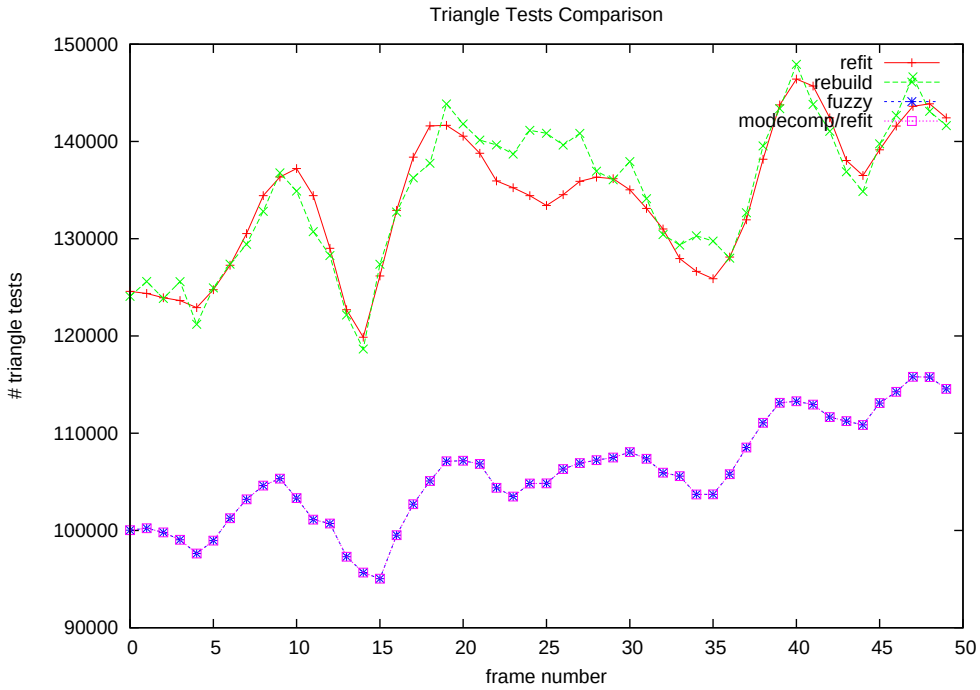
Voor de Paladin Jogging animatie (zie figuur 5.12) is het verloop van de grafieken ongeveer analoog aan dat van de Cally Walking animatie.

5.2.2 Aantal Straal-AABB Intersecties

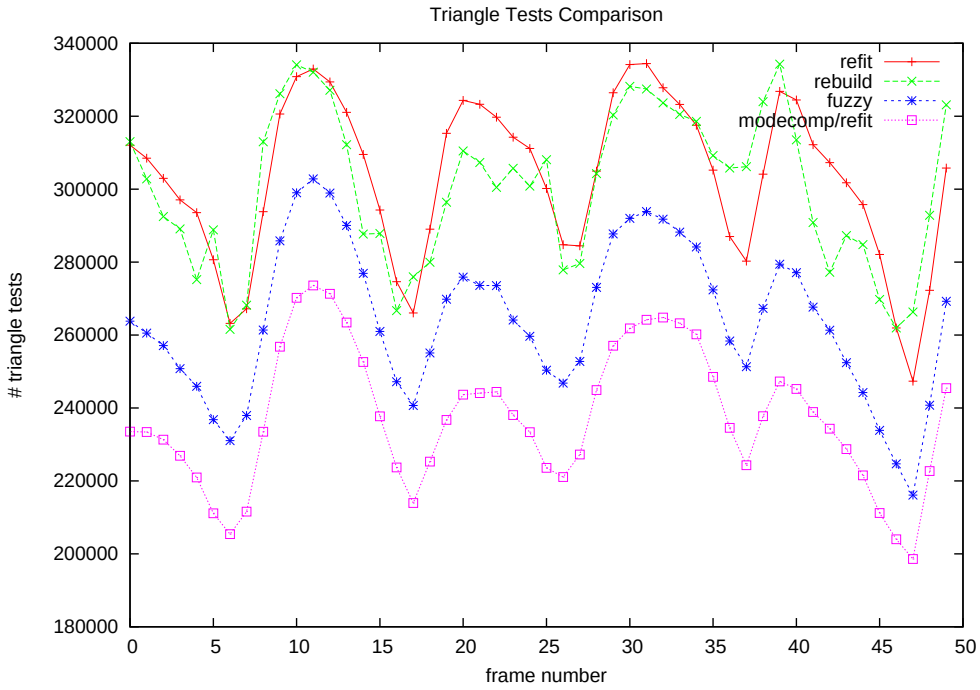
Ook het aantal straal-AABB intersecties is een metriek onafhankelijk van externe factoren die we hier in detail bekijken voor de vier testanimaties.



Figuur 5.10: Het aantal straal-driehoek intersecties voor de Cally Freestyle Animatie.



Figuur 5.11: Het aantal straal-driehoek intersecties voor de Skeleton Freestyle Animatie.



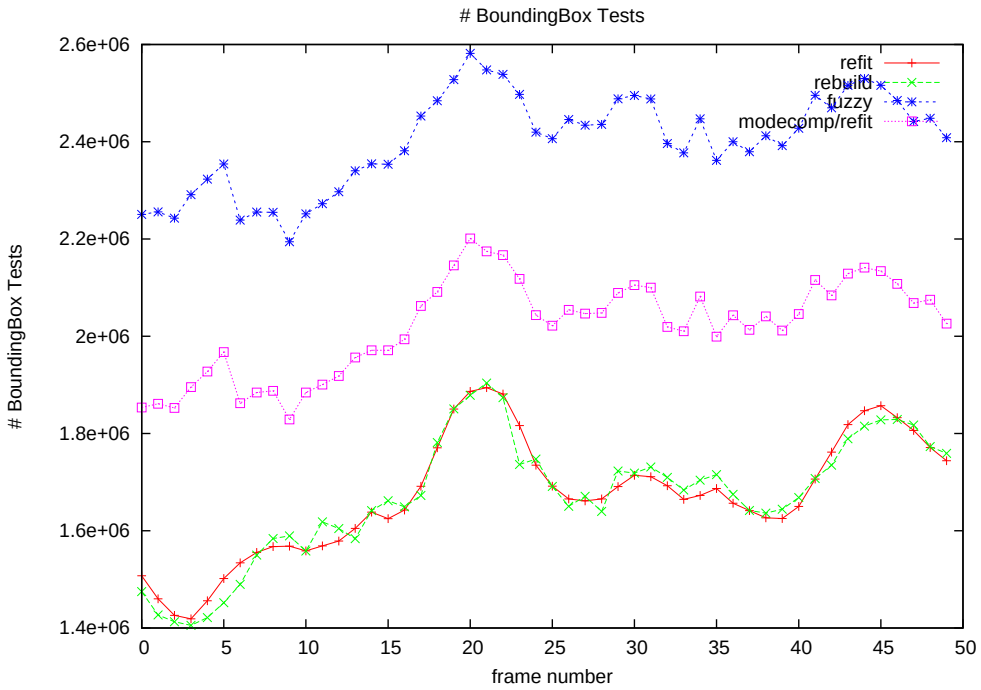
Figuur 5.12: Het aantal straal-driehoek intersecties voor de Paladin Jogging Animatie.

Het is duidelijk dat motion decomposition & fuzzy versnellingsstructuren het meest aantal testen doet (zie figuur 5.13) voor de Cally Walking animatie. De reden hiervoor is de fuzzy boxes en extra testen in de top-level structuur. Motion decomposition & refitting doet het beter maar doet nog altijd meer testen omwille van het complexere traversal algoritme. Refitting en fast rebuilds doen grofweg hetzelfde aantal testen.

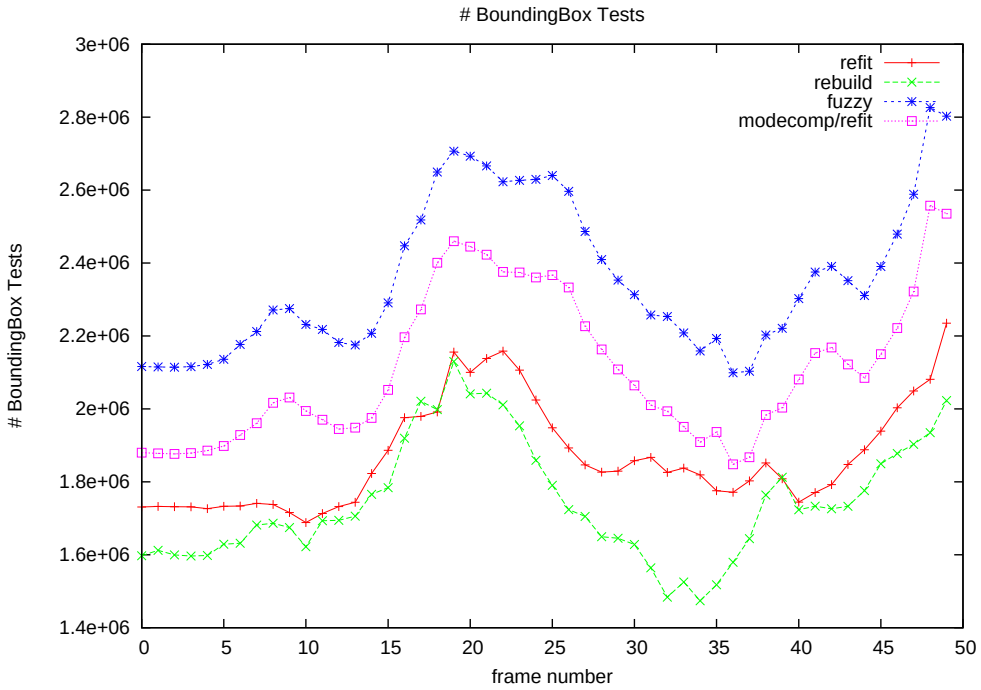
Ook voor de Cally Freestyle animatie (zie figuur 5.14) is het gedrag zoals verwacht. Refitting doet hier altijd meer straal-AABB testen omwille van de zwaar vervormde BVH als gevolg van de hevige beweging in de animatie.

Het bewijs dat de motion decomposition technieken exact dezelfde BVH gebruiken voor de Skeleton Freestyle animatie wordt ook geleverd door grafiek 5.15. Het aantal testen voor de 2 methoden is exact gelijk. Fast rebuilds doet hier minder testen dan refitting. Merk op dat hoewel de motion decomposition technieken minder straal-driehoek en straal-AABB testen doen, de rendertijd toch hoger is dan voor de andere technieken. Dit minieme verschil van ongeveer 10 ms is te wijten aan de tijd nodig voor de straaltransformaties.

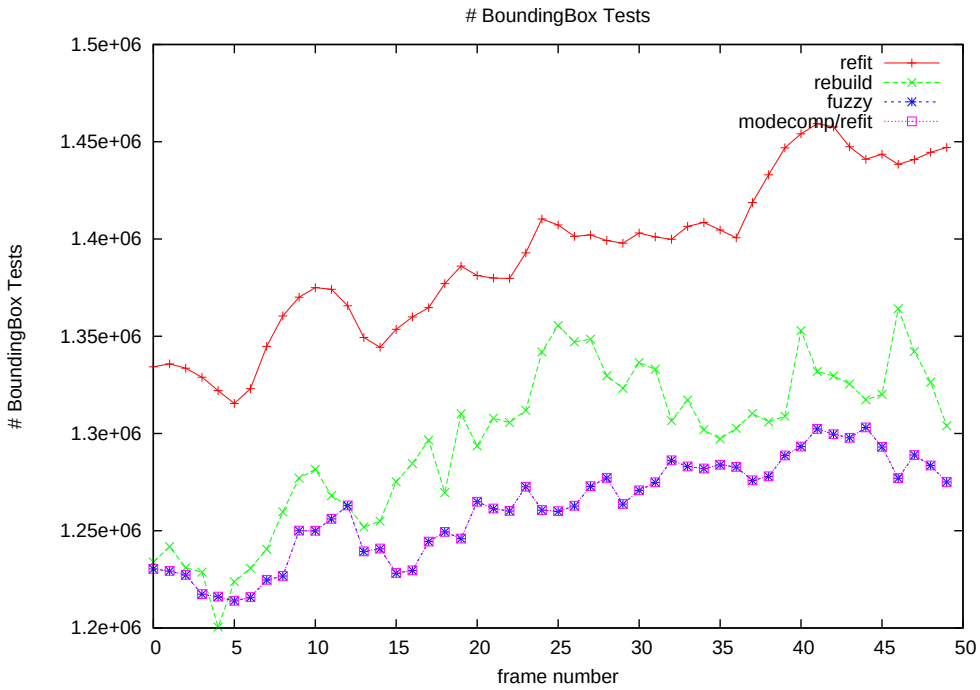
Voor de Paladin Jogging scene (zie figuur 5.16) doet motion decomposition & fuzzy versnellingsstructuren het meest aantal testen. Fast rebuilds en refitting zijn ongeveer aan elkaar gewaagd.



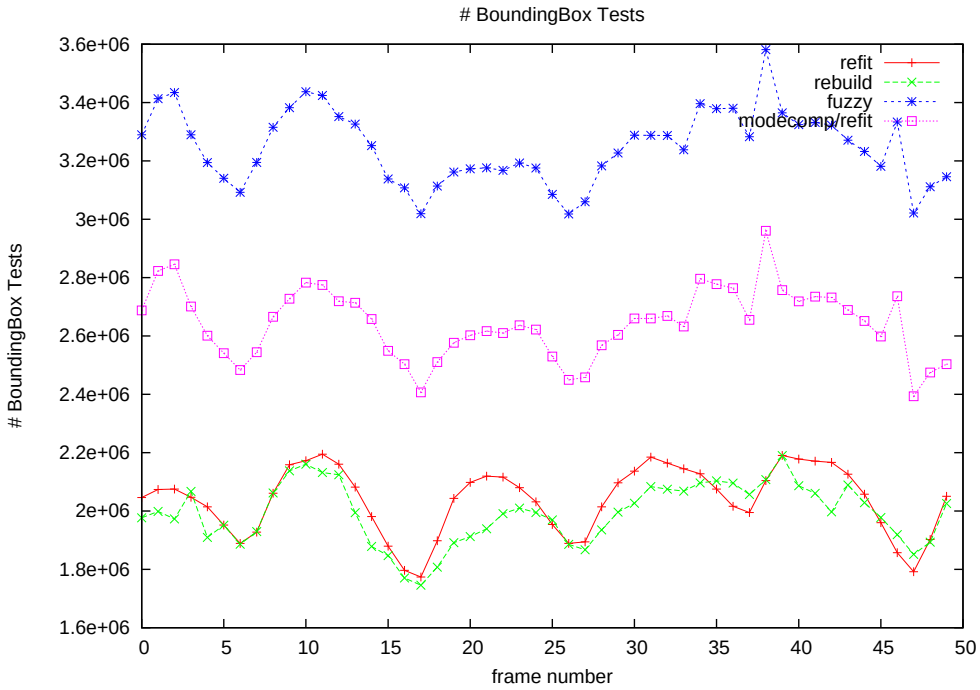
Figuur 5.13: Het aantal straal-AABB intersecties voor de Cally Walking Animatie.



Figuur 5.14: Het aantal straal-AABB intersecties voor de Cally Freestyle Animatie.



Figuur 5.15: Het aantal straal-AABB intersecties voor de Skeleton Freestyle Animatie.



Figuur 5.16: Het aantal straal-AABB intersecties voor de Paladin Jogging Animatie.

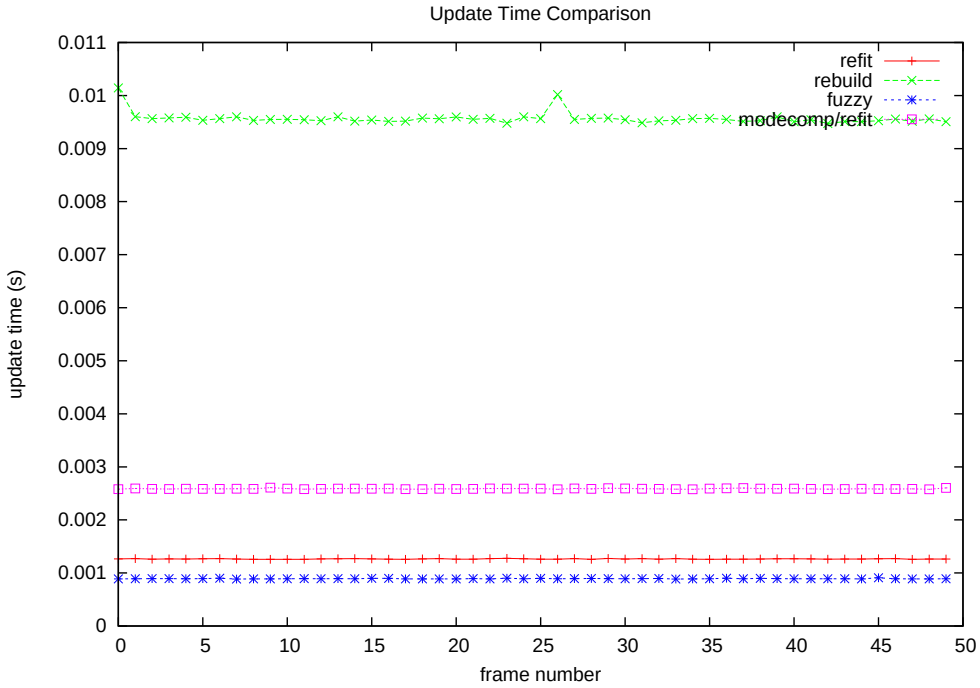
5.3 Update Tijden

De andere component van de time-to-image is de update tijd. Hiermee bedoelen we het skinnen van een animatie en het eventueel refitten of herbouwen van een BVH. Een bepaalde techniek mag nog zo snel renderen, als de update tijd te hoog is wordt hiermee kostbare tijd verloren.

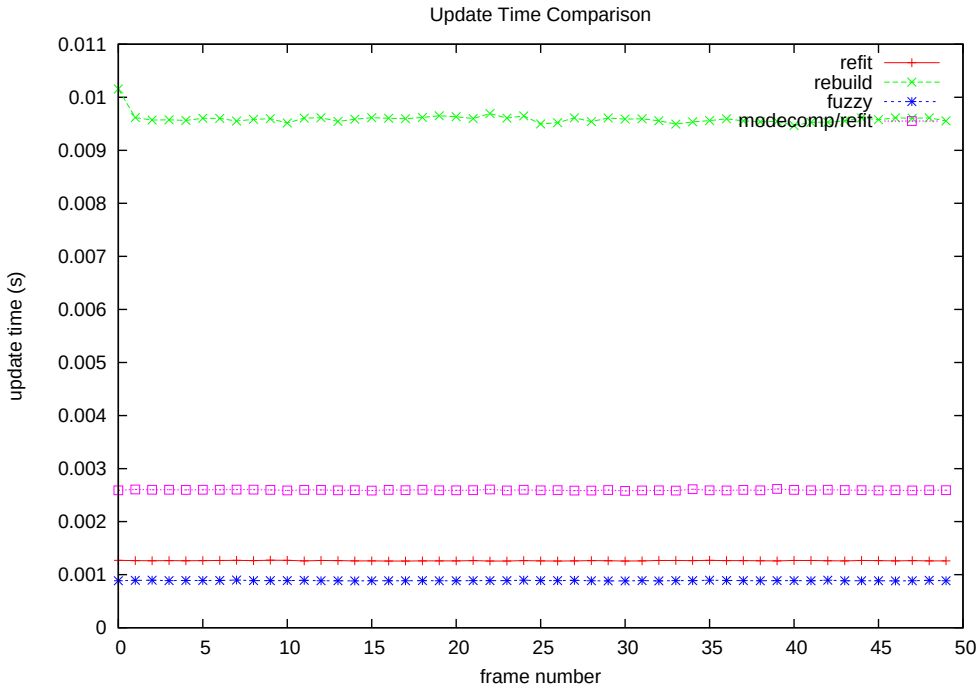
De update tijden zijn samengevat in grafieken 5.17, 5.18, 5.19 en 5.20. De tijden geven exact weer wat we theoretisch zouden verwachten:

1. Motion decomposition & fuzzy versnellingsstructuren is het snelste. De top-level structuur wordt herbouwd over een klein aantal bounding boxes (< 60 voor onze animaties). De lokale fuzzy structuren worden niet aangepast.
2. Refitting moet de BVH aanpassen. Dit heeft tijdscomplexiteit $O(N)$.
3. Motion decomposition & refitting moet een update doen van de lokale fuzzy structuren. Dit heeft een tijdscomplexiteit van $O(N)$. Ook moet de top-level structuur opnieuw worden gebouwd.
4. Fast rebuilds is veel trager omwille van de volledig herbouw. Een volledige herbouw heeft tijdscomplexiteit $O(N \times log(N))$. De tijden zijn een ordegrootte slechter dan die van de snelste techniek.

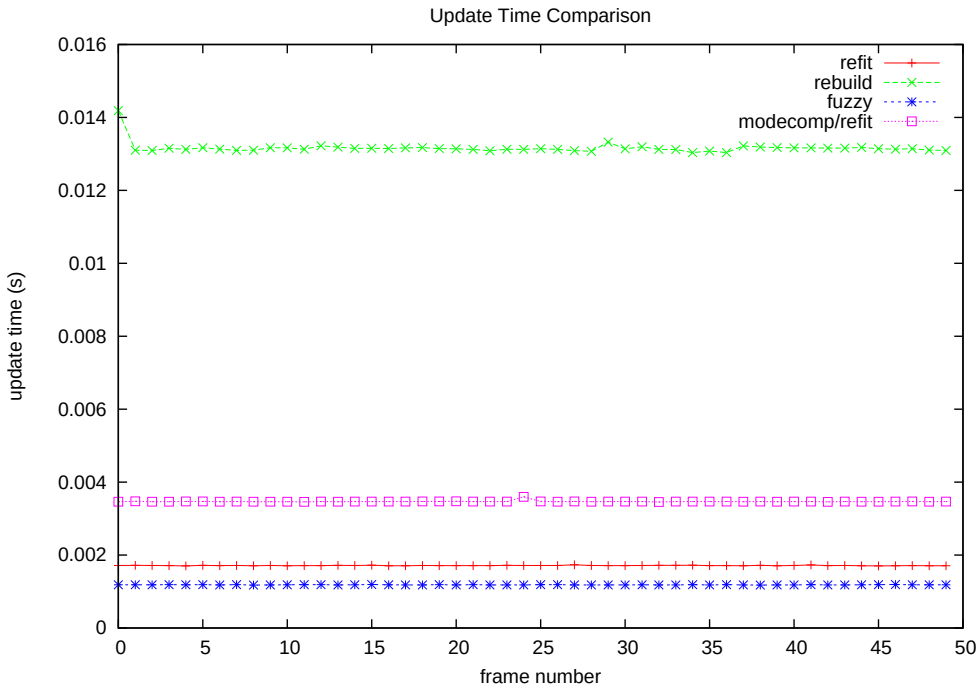
Afhankelijk van de techniek is de verhouding update tijd versus time-to-image minder dan 1% tot meer dan 10%. Zeer verwonderlijk is dat de update tijd van de refitting techniek zeer dicht tegen de update tijd van de motion decomposition & fuzzy versnellingsstructuren ligt.



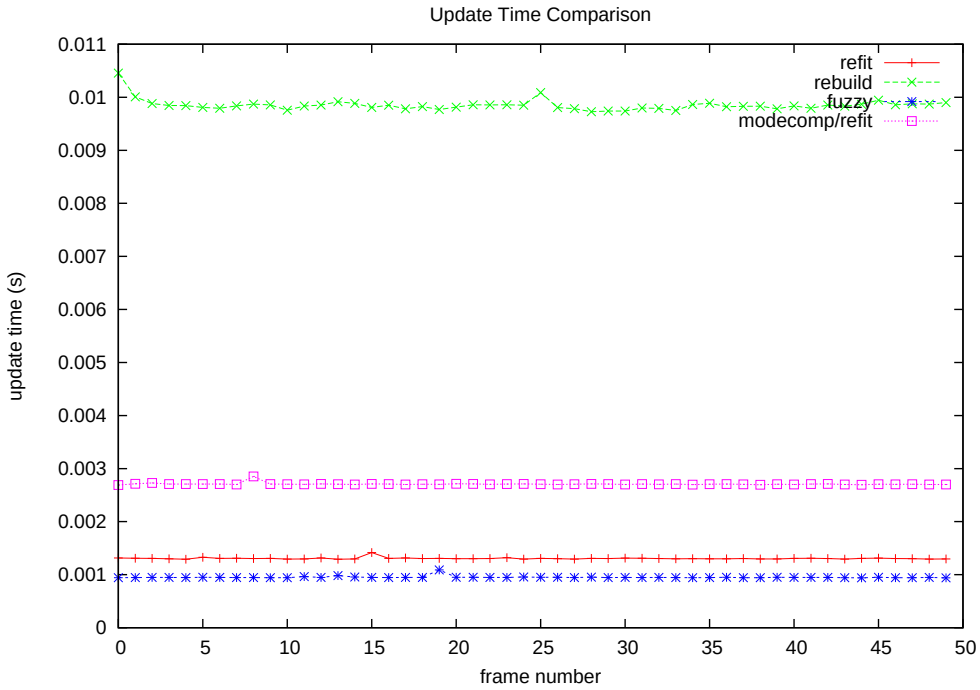
Figuur 5.17: De updatetijden voor de Cally Walking animatie.



Figuur 5.18: De updatetijden voor de Cally Freestyle animatie.



Figuur 5.19: De updatetijden voor de Skeleton Freestyle animatie.



Figuur 5.20: De updatetijden voor de Paladin Jogging animatie.

5.4 Conclusie

Na een grondige vergelijking van de technieken zijn we klaar voor een afsluitend oordeel. Alle technieken hebben voor- en nadelen dus is het bijna onmogelijk om de beste techniek aan te duiden. We zullen proberen een leidraad te geven.

Qua time-to-image is refitting altijd de iets betere techniek. De rendertijden zijn goed en vergelijkbaar met die van een BVH die elk frame wordt herbouwd. De updatetijd is zeer laag. In het geval van animaties met veel beweging zoals de Cally Freestyle animatie daalt de kwaliteit van de BVH zeer duidelijk. In geval van twijfel zouden we eerst deze techniek aanraden omwille van zijn simpliciteit.

Ook fast rebuilds is zeer performant. De rendertijden zijn snel maar de bouwtijd bepaalt een redelijke fractie van de time-to-image. Het is niet altijd nodig om voor elk frame van de animatie opnieuw te bouwen. Een betere techniek is een hybride techniek: de refitting techniek gebruiken en periodiek herbouwen. Ideaal zou zijn om niet periodiek te herbouwen maar adaptief. Ons voorstel is om elk frame bepaalde metrieken van de versnellingsstructuur te meten (vb. aantal intersecties, surface area van deelbomen, ...) en dan de wanneer een drempelwaarde wordt bereikt de structuur herbouwen of gedeeltelijk aanpassen [1]. Dit zou een interessante onderzoekspiste zijn voor een thesis.

Motion decomposition & fuzzy versnellingsstructuren haalt ook interactieve framerates maar zeer verrassend minder goed dan refitting en fast rebuilds. Het probleem zit in het complexe traversal algoritme. Eerst moet een top-level structuur doorlopen worden, daarna een straaltransformatie waarna verschillende fuzzy structuren moeten doorlopen worden. Deze kost kan bij onze scenes niet terugverdiend worden door de goedkope update stap bij elk frame.

Ook onze eigen techniek motion decomposition & refitting kan niet concurreren met refitting en fast rebuilding. Het vermijden van de fuzzy boxes geeft betere rendertijden maar niet spectaculair beter. Het essentiële probleem bij de motion decomposition technieken is het complexere traversal algoritme.

Als het aantal driehoeken in een scene gevoelig verhoogt is er geen andere keuze dan de motion decomposition & fuzzy versnellingsstructuren. Het updaten van de versnellingsstructuren wordt dan een grotere kost dan renderen.

[1] Een voorbeeld van gedeeltelijk aanpassen is *tree rotations*. Wanneer een BVH effectief gebouwd is, is het mogelijk om de echte kost te berekenen met de SAH. Kensler stelt voor om die kost te berekenen en dan knopen in de BVH van plaats te wisselen om zo de kost proberen te verminderen. Voor meer uitleg zie [Ken08].

Hoofdstuk 6

Besluit

6.1 Overzicht

In deze thesis hebben we verschillende technieken bestudeerd voor interactieve ray tracing van skeletgebaseerde animaties. Hoewel ray tracing in staat is om fotorealistische afbeeldingen te genereren is het klassieke algoritme veel te traag voor gebruik in interactieve toepassingen.

We zijn begonnen met het uiteenzetten van het concept animatie en meer specifiek skeletgebaseerde animatie, de standaard techniek voor het modelleren van karakters. Door skinning kan een skelet de bovenliggende geometrie op een natuurlijke wijze vervormen.

Daarna hebben we het ray tracing algoritme besproken en enkele technieken om het te versnellen. Het essentiële probleem is het grote aantal straal-driehoek intersecties. Versnellingsstructuren proberen het aantal straal-driehoek intersecties sterk te verminderen door de driehoeken te 'sorteren'. Specifiek hebben we de BVH als versnellingsstructuur besproken omdat we deze hebben gebruikt in onze implementatie. Een bijkomende optimalisatie van ray tracing is het bundelen van stralen en deze samen in de scene schieten als een ray pakket.

In hoofdstuk 4 passen we specifieke technieken voor ray tracing van animaties toe op skeletgebaseerde animaties. Concreet bespreken we *fast rebuilds*, *refitting*, *motion decomposition & fuzzy versnellingsstructuren* en *motion decomposition & refitting*. Onze implementatie is in staat om framerates te halen tussen 7 fps en 14 fps voor een resolutie van 512x512 pixels afhankelijk van de techniek en animatie.

Een grondige vergelijking van de technieken presenteren we in hoofdstuk 5. We zijn de eerste auteurs die een vergelijking en analyse doen van deze verschillende technieken. Bovendien zijn onze vergelijkingen volledig 'objectief' omdat elke techniek geïmplementeerd

is op dezelfde onderliggende code en de resultaten verkregen zijn onder exact dezelfde omstandigheden (zelfde hardware, ...).

6.2 Conclusie

Eerst en vooral zijn we er in geslaagd om een interactieve ray tracer te bouwen. Bovenop deze ray tracer hebben we specifieke technieken geïmplementeerd voor animaties in het algemeen en skeletgebaseerde animaties in het bijzonder.

Uit een vergelijking van de verschillende technieken in onze implementatie kunnen we besluiten dat refitting de beste keuze is voor skeletgebaseerde animaties. Verrassend genoeg is motion decomposition & fuzzy versnellingsstructuren niet de meest performante techniek alhoewel die speciaal ontwikkeld is voor skeletgebaseerde animaties. Het algoritme heeft als doel het herbouwen van een versnellingsstructuur compleet te vermijden ten koste van een minder performante versnellingsstructuur en een complexer traversal algoritme. Uit onze resultaten is het duidelijk dat het complexe traversal algoritme te zwaar weegt op de methode. Met een eigen techniek hebben we motion decomposition & fuzzy structuren en refitting gecombineerd. Dit leidt tot betere resultaten maar toch kan ook onze eigen techniek niet concurreren met refitting.

Uiteindelijk zijn we zeer tevreden met onze resultaten. Onze resultaten tonen aan dat interactieve ray tracing mogelijk is voor skeletgebaseerde animaties. Bovendien zijn we de eersten om een vergelijking te maken tussen de verschillende technieken.

6.3 Toekomstig onderzoek

Ray tracing van animaties is een actief onderzoeksdomein en daarom willen ook wij op basis van onze ervaringen enkele suggesties doen voor toekomstig onderzoek.

Uit ons onderzoek is gebleken dat eenvoudigere, algemene methodes zoals rebuilding en refitting betere resultaten geven voor skeletgebaseerde animaties dan meer gespecialiseerde methoden. Daarom lijkt het ons aangewezen om het onderzoek te concentreren op algemene ray tracing technieken voor animaties. Deze technieken zullen dan hoogstwaarschijnlijk ook makkelijk toepasbaar zijn op een gespecialiseerde niche zoals skeletgebaseerde animaties.

Omdat rebuilding en refitting de beste resultaten geven suggereert dat een betere oplossing door combinatie van de 2 technieken. Een hybride oplossing past refitting toe op een versnellingsstructuur en herbouwd deze wanneer de kwaliteit te sterk is afgenomen. Dit vereist een metriek voor het bepalen van de kwaliteit van versnellingsstructuren die liefst snel te berekenen is. Een mogelijkheid is een metriek op basis van de SAH. Wanneer deze waarde onder een vooraf vastgelegde drempelwaarde valt is het tijd voor het herbouwen van de versnellingsstructuur.

De vraag is of de SAH überhaupt de beste heuristiek is voor animaties. De SAH wordt beschouwd als de beste heuristiek voor statische scenes en is daarna volledig overgenomen voor dynamische scenes. Volgens ons moet een goede heuristiek voor animaties niet naar de geometrie kijken van slechts één frame. Bijvoorbeeld een ideale heuristiek voor refitting kan een versnellingsstructuur bouwen die optimaal is voor meerdere frames (met refitting tussen de verschillende frames). Bovendien kan de heuristiek ook op voorhand voorspellen na hoeveel frames het beter is om de versnellingsstructuur volledig te herbouwen. Nieuwe heuristieken onderzoeken is volgens ons een interessante denkpiste.

De focus in onze thesis lag sterk op de BVH als versnellingsstructuur. Dit wil zeker niet zeggen dat de BVH de beste structuur is voor animaties. Verder onderzoek moet zich daarom niet alleen concentreren op de 'klassieke' versnellingsstructuren zoals grids, kd-trees en BVHs maar actief op zoek gaan naar nieuwe mogelijkheden. Bijvoorbeeld [WK06] introduceert de *Bounding Interval Hierarchy*, [WMS06] combineert de BVH en kd-tree in de *B-KD tree* en [LD08] maakt gebruik van *constrained tetrahedralizations*. Net zoals Havran een vergelijking heeft gemaakt van ray tracing technieken voor statische scenes [Hav00], is er ook nood aan een grondige vergelijking van de mogelijkheden voor dynamische scenes.

Met de opkomst van multi-core architecturen zijn multi-threaded aanpakken een interessante denkpiste. Een thread kan de versnellingsstructuur voor het volgende frame bouwen terwijl een ander thread het huidige frame rendert. Een voorwaarde is dat de geometrie van het volgende frame reeds beschikbaar is in het huidige frame. Ook traversal van een versnellingsstructuur leent zich tot multi-threading omdat traversal de versnellingsstructuur niet aanpast zijn *concurrency* problemen onbestaand. Verschillende threads kunnen bijvoorbeeld elk verantwoordelijk zijn voor de verwerking van een ray pakket.

Ook het onderzoek naar ray tracing hardware kan in de toekomst potentieel belangrijk worden. Prototypes ontwikkeld met FPGAs (Field Programmable Gate Arrays) [SWW+04] zijn in staat om rendertijden te halen die competitief zijn met de state-of-the-art software ray tracers bij kloksnelheden van slechts 90MHz! Het bouwen van versnellingsstructuren is nog altijd een probleem op hardware en vraagt nog veel verder onderzoek. [1]

[1] In april 2009 heeft de firma Caustic een commerciële ray tracing versnellingskaart gelanceerd waarmee interactieve framerates te halen zijn voor zeer complexe scenes [Gra].

Verklarende Woordenlijst

AABB Afkorting voor Axis Aligned Bounding Box. Een bounding box met de 6 vlakken parallel aan de coördinaatassen.

Affiene transformatie Meetkundige transformatie waarbij de meetkundige structuur (punten blijven punten, rechten blijven rechten en vlakken blijven vlakken) en parallellisme behouden blijven.

Bin Datastructuur gebruikt bij het implementeren van de fast rebuild techniek.

Bounding box Omsluitend volume in de vorm van een balk.

Bounding sphere Omsluitend volume in de vorm van een bol.

BVH Afkorting voor Bounding Volume Hierarchy. Een specifieke versnellingsstructuur voor ray tracing.

Cal3d C++ bibliotheek voor het beheer van skeletgebaseerde animaties.

Cluster Verzameling driehoeken die dezelfde affiene beweging ondergaan.

Deformable scene Scene waarbij de connectiviteit van de polygonen behouden blijft over de frames.

Fast Rebuilds Techniek voor het snel herbouwen van een versnellingsstructuur. Wordt gebruikt om een versnellingsstructuur te herbouwen voor elk frame van een animatie.

FPS Frames per seconden, metriek voor de framerate in animaties.

Frame Afbeelding in een animatie. Frames worden versneld achter elkaar getoond om de illusie van beweging op te wekken.

Front-to-back Traversal Optimale manier om een versnellingsstructuur voor het ray tracing algoritme te doorlopen.

Fuzzy Box Bounding box die de positie van een primitief omsluit gedurende een volledige animatie.

Fuzzy Structuur Versnellingsstructuur gebouwd over fuzzy boxes.

Median split Heuristiek voor het bouwen van een versnellingsstructuur

Mesh Voorstelling van een volume in computer graphics. Meshes worden vaak gemodelleerd als een verzameling aaneengesloten driehoeken.

Motion Decomposition Principe waarbij beweging wordt gesplits in coherente beweging en residuele beweging.

Primaire straal Straal in het ray tracing algoritme die begint in de camera. Ook wel camerastraal of oogstraal genoemd.

Rasterizatie Algoritme voor het renderen van scenes, is het standaard algoritme voor grafische kaarten.

Ray Packet Een bundel van coherente stralen.

Ray tracing Populair algoritme voor het renderen van fotorealistische afbeeldingen.

Refitting Techniek voor het snel aanpassen van een versnellingsstructuur. Is alleen geschikt voor deformable scenes.

Rustpose Pose waarin een karakter wordt gemodelleerd voor het wordt geanimeerd, 'Da Vinci' code.

SAH Afkorting voor Surface Area Heuristic. Een heuristiek voor het bouwen van een versnellingsstructuur. Wordt beschouw als de beste heuristiek voor het bouwen van versnellingsstructuren in huidig onderzoek.

Secundaire straal Straal in het ray tracing algoritme die begint in het snijpunt van een primaire straal en een object. Een voorbeeld van een secundaire straal is een schaduwstraal.

SIMD Single Instruction, Multiple Data, speciale machine-instructies die een operand toepassen op verschillende getallen in parallel. Deze instructie zijn architectuurspecifiek.

Skeletgebaseerde animatie Animatie op basis van een onderliggend skelet.

Skinning Techniek voor het animeren van karakters in computeranimaties of computerspelletjes.

Splitvlak Vlak dat bij het bouwen van een versnellingsstructuur een verzameling driehoeken partioneert in 2 disjuncte partities.

Time-to-image Totale tijd nodig voor het genereren van een afbeelding of frame.

Top-level Versnellingsstructuur Versnellingsstructuur die gebouwd worden over andere versnellingsstructuren.

Transformatiematrix Matrix die veel gebruikt wordt voor het formaliseren van affiene transformaties.

Traversal Het doorlopen van een versnellingsstructuur.

Versnellingsstructuur Specifieke datastructuur voor het versnellen van het ray tracing algoritme.

Vertex Hoekpunt van 1 of meerdere driehoeken in een mesh opgebouwd uit driehoeken.

Bibliografie

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[cal] Cal3d. 3d character animation library. `http://home.gna.org/cal3d/`.

[Dut06] Phil Dutré. Global illumination compendium, 2006. `http://www.cs.kuleuven.ac.be/ phil/GI/`.

[GFSS06] Johannes Günther, Heiko Friedrich, Hans-Peter Seidel, and Philipp Slusallek. Interactive ray tracing of skinned animations. *The Visual Computer*, 22(9):785–792, September 2006. (Proceedings of Pacific Graphics).

[GFW+06] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum*, 25(3):517–525, September 2006. (Proceedings of Eurographics).

[Gra] Caustic Graphics. Caustic graphics :: Realtime raytracing :: Causticrt. `http://www.caustic.com/`.

[Hav00] Vlastimil Havran. *Heuristic Ray Shooting Algorithms.* Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[Hav07] Vlastimil Havran. About the relation between spatial subdivisions and object hierarchies used in ray tracing. In Mateu Sbert, editor, *23rd Spring Conference on Computer Graphics (SCCG 2007)*, pages 55–60, Budmerice, Slovakia, May 2007. ACM SIGGRAPH and EUROGRAPHICS, ACM.

[Ken08] Andrew Kensler. Tree Rotations for Improving Bounding Volume Hierarchies. In *Proceedings of the 2008 IEEE Symposium on Interactive Ray Tracing*, pages 73–76, Aug 2008.

[LD08] Ares Lagae and Philip Dutré. Accelerating ray tracing using constrained tetrahedralizations. *Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering)*, 27(4):1303–1312, June 2008.

[LeYM06] Christian Lauterbach, Sung eui Yoon, and Dinesh Manocha. Rt-deform: Interactive ray tracing of dynamic scenes using bvhs. In *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

[MB90] David J. MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166, 1990.

[MTLT88] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[pov09] Pov-ray hall of fame, 2009. `http://hof.povray.org/index-lb.html`.

[RSH05] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1176–1185, New York, NY, USA, 2005. ACM.

[Ryp] Daniel Rypl. Approaches to discretization of 3d surfaces. `http://mech.fsv.cvut.cz/ dr/papers/Habil/habil.html`.

[SAG+05] Peter Shirley, Michael Ashikhmin, Michael Gleicher, Stephen Marschner, Erik Reinhard, Kelvin Sung, William Thompson, and Peter Willemsen. *Fundamentals of Computer Graphics, Second Ed.* A. K. Peters, Ltd., Natick, MA, USA, 2005.

[SM03] Peter Shirley and R. Keith Morley. *Realistic Ray Tracing.* A. K. Peters, Ltd., Natick, MA, USA, 2003.

[Sta] Stanford. Stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

[SWW+04] Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek. Realtime ray tracing of dynamic scenes on an fpga chip. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 95–106, New York, NY, USA, 2004. ACM.

[Uni] Ulm University. Benchmarking ray tracing for realistic light transport algorithms. `http://bwfirt.sourceforge.net/`.

[Wal04] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination.* PhD thesis, Computer Graphics Group, Saarland University, 2004.

[Wal07] Ingo Wald. On fast construction of sah-based bounding volume hierarchies. *Symposium on Interactive Ray Tracing*, 0:33–40, 2007.

[WBS03] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In Alan Chalmers and Theresa-Marie Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001. available at http://graphics.cs.uni-sb.de/ wald/Publications.

[Wik] Wikipedia. Computergraphics. `http://nl.wikipedia.org/wiki/Computergraphics`.

[wik08] Ray tracing (graphics) - wikipedia, 2008. `http://en.wikipedia.org/wiki/Ray_tracing_(graphics)`.

[WK06] Carsten Wchter and Er Keller. Instant ray tracing: The bounding interval hierarchy. In *In Rendering Techniques 2006 Proceedings of the 17th Eurographics Symposium on Rendering*, pages 139–149, 2006.

[WMG+07] Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In Dieter Schmalstieg and Jirí Bittner, editors, *STAR Proceedings of Eurographics 2007*, pages 89–116, Prague, Czech Republic, September 2007. Eurographics Association.

[WMS06] Sven Woop, Gerd Marmitt, and Philipp Slusallek. B-kd trees for hardware accelerated ray tracing of dynamic scenes. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 67–77, New York, NY, USA, 2006. ACM.

[wor] Worldforge, the original open source mmo project. `http://www.worldforge.org/`.

[YL06] Sung-Eui Yoon and Peter Lindstrom. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1213–1220, 2006.

Interactive Ray Tracing of Skeleton Based Animations

Thomas Loockx

***Abstract*—Current state-of-the-art ray tracers are capable of rendering static scenes at interactive framerates. The case for animated scenes is still a topic of current research. Because of the changing geometry, data structures built for the current frame are invalid the next frame. In this paper we discuss and compare four approaches for ray tracing skeleton based animations, typically used in computer games. All techniques use the BVH as data structure. Our implementation is able to render scenes at frame rates up to 14 frames per second at 512x512 resolution.**

***Index Terms*—interactive ray tracing, animations, skeleton based animations, skinning, BVH, fuzzy data structures, motion decomposition, refitting, fast rebuilds**

I. Introduction

Since Appel [App68] introduced ray tracing a lot has changed in the field of computer graphics. Ray tracing is a popular method for offline rendering and has become fairly interactive with the advent of ray tracing data structures. Today the dominating data structures are BVHs, grids and kd-trees. With the introduction of coherent ray tracing, another significant step forward was made. Rays traverse the data structure together in *packets* to amortize the traversal cost. Ray packets can be processed by the use of SIMD extensions very easily which makes them a good choice on current architectures.

Traditionally interactive rendering is dominated by rasterisation algorithms. A significant advantage of ray tracing over these algorithms is the ease of adding techniques like shadows, reflections, refractions... Rasterisation algorithms suffer from the fact that adding new techniques is getting more and more complex.

Ray tracing animated scenes is still a moving target in current research. The problem is that a data structure built for the current frame is invalid for the next frame because of the changing geometry. The time spent on building or updating the acceleration structure is a significant chunk in the total framerate of an animation. Current techniques for dynamic scenes can be divided into three categories: fast rebuilding the data structure every frame, updating the data structure or trying to build a data structure that is valid for every frame.

In our paper we explore some current techniques for ray tracing animated scenes and use them for skeleton based animations. First we will discuss related work in section II. In section III we discuss skeleton based animations. The different methods we used are discussed in section IV. A thorough comparison of the methods is made in section V. We formulate our conclusion in section VI.

II. Related Work

Although ray tracing is a topic of active research in computer graphics since decades, the field of interactive ray tracing is fairly young. In 2001 Wald et al. [WBWS01] proposed the idea of exploiting ray coherence by bundling rays together in *ray packets*. The implementation of the OpenRT ray tracing engine proved that it was possible ray tracing complex scenes at interactive frame rates [Wal04] on common desktop pc's.

More recently some interesting approaches emerged for ray tracing animated scenes using grids [WIK$^+$06], BVHs [WBS07] [LeYM06] and kd-trees [GFW$^+$06]. Günther et al. were the first to present a ray tracing approach specifically for skeleton based animations [GFW$^+$06]. For a complete overview of the research in the field of interactive ray tracing we highly recommend [WMG$^+$07].

III. Skeleton Based Animations

Skeleton based animation is the standard technique for animating characters in computer games and movies. A character is represented by a skin and underlying skeleton. The character is animated by animating the underlying skeleton. This technique, also known as *skinning*, *vertex blending* or *skeletal subspace deformation*, was first introduced in [MTLT88].

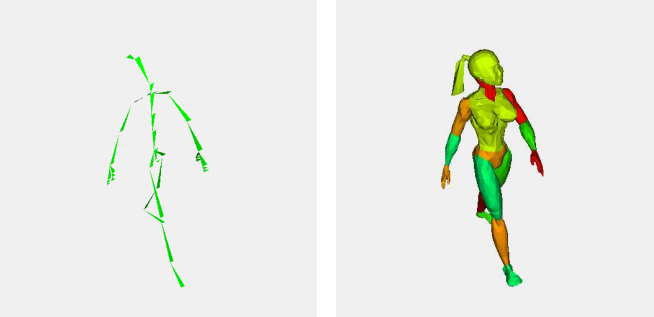


Fig. 2. The character is animated by animating the underlying skeleton.

Each character has an underlying skeleton composed of bones. Each bone has a transformation matrix associated with it describing the affine transformation of the bone. Over the skeleton a set of meshes is defined, these meshes are the visible parts of the character (head, arms, torso, ...). Each vertex of the meshes is influenced by one or several bones. To animate the character the skeleton is deformed w.r.t. a rest pose, due to the coupling between mesh and bone(s) the meshes are deformed as well (see figure 2). The position of a vertex at time t is simply a weighted sum:

$$v_i(t) = \sum_{i=1}^{N} w_i M_i(t) \tilde{v}_i \qquad \text{with} \sum_{i=1}^{N} w_i = 1$$

$v_i(t)$ is the position of vertex v_i at time t. N is the number of bones influencing vertex v_i. $M_i(t)$ is the transformation matrix associated with bone i at time t. Weight w_i is the influence of bone i on the vertex and $\tilde{v}_i$ is the position of the vertex in the rest pose. This operation is called *skinning* and is applied to each vertex every frame.

(a) Cally Walking (b) Cally Freestyle (c) Skeleton Freestyle (d) Paladin Jogging

Fig. 1. Some fragments from our four test animations.

[WMG+07] proposes a classification for animations. Skeleton based animations fall in the category semi-hierarchical. The movement of the skeleton is fully hierarchical (affine) but the influence of multiple bones on a vertex introduces a residual component in the motion of each vertex.

IV. Used Methods for Ray Tracing Skeleton Based Animations

As stated before the problem in ray tracing animated scenes are the data structures. Three approaches can be distinguished in handling data structures:

Rebuilding The data structure is rebuild each frame. A trade-off is made between build-time and build-quality.

Refitting The data structure is modified every frame. The data structure is optimized for one frame and deteriorates the next frames.

One fits all A data structure is built that is correct for every frame. Obviously the data structure is a mean for all the frames.

We discuss four methods, one from each category and our own hybrid method. In our implementation all the methods use the BVH as data structure. Although the kd-tree is considered the best data structure for offline rendering [Hav00], a search for a 'best' structure for dynamic scenes is still a topic in current research.

Recently Wald and Shirley showed that BVHs can be built with very good rendering performance [WBS07]. Because of it's simplicity and robustness we chose to use a BVH for all our implementations.

A. Fast Rebuilds

Wald [Wal07] introduced a fast build algorithm for BVHs based on the surface area heuristic. The high performance is achieved by the use of binning, a technique first explored for the fast building of kd-trees [WH06]. The scene is divided into K bins of equal width by subdividing the bounding box of the scene. The primitives are distributed over the bins. Each primitive goes in the bin that contains the center of the primitives bounding box. Each plane between two bins is a possible split for partitioning the set of triangles into two halves. The cost for using plane j as split plane is formulated as:

$$Cost_j = A_{L,j}N_{L,j} + A_{R,j}N_{R,j} \qquad j = 1 \ldots K-1$$

$A_{L,j}$ and $A_{R,j}$ are the accumulated areas of the bounding boxes left and right, $N_{L,j}$ and $N_{R,j}$ are the triangle counts left and right of the split plane. These metrics are calculated by a left sweep followed by a right sweep over the bins. Wald determined $K = 16$ as the optimal number of bins. Compared to standard techniques for building BVHs, Wald's technique is one order of magnitude faster while rendering at almost the same speed (Wald reports a 7% performance penalty in the worst case.).

This technique is suitable for all animation categories so we can easily apply it to skeleton based animations.

B. Refitting

Another approach is updating the data structure every frame. [WBS07] suggests refitting a BVH every frame for deformable scenes. The approach builds a high quality BVH over one of the frames. Wald et al. builds the BVH over the first frame. Our experiments show that building the BVH over the rest pose yields better results for skeleton based animations. When the vertex positions of the primitives change, a new bounding box is calculated for each primitive, this corresponds to the leaf nodes of the BVH. Afterwards the BVH is traversed recursively to update the bounding box of the internal nodes. So the same BVH topology is reused for the subsequent frames.

This approach can lead to a performance penalty in the case of 'heavy motion' or non-deformable scenes. Luckily the motion of skeleton based animations is semi-hierarchical so good results can be expected.

For non-trivial scenes this approach will always yield a faster update step for each frame. Reason is the asymptotic complexity of refitting ($O(N)$) versus rebuilding ($O(N \times log(N))$). Whether there's a gain in traversal performance will be examined in section V.

C. Motion Decomposition & Fuzzy Data Structures

Günther et al. introduces an interesting approach specifically for skeleton based animations [GFSS06] that almost completely avoids the update of data structures. The approach is based on

the observation that the motion of a vertex in the local coordinate system of a carefully chosen bone is small, this is the residual motion.

Because of this small motion in the local coordinate system, it's possible to build a *fuzzy box* over the vertex. The fuzzy box is an axis-aligned bounding box that captures all the positions of the vertex during the whole animation. The fuzzy box of a triangle is the union of the fuzzy boxes of it's vertices. Over these fuzzy boxes a data structure is built, one for each bone. Günther et al. uses a kd-tree as data structure, in our implementation we used a BVH.

A cluster is a set of triangles assigned to the same bone. Each frame the bounding box is calculated for each cluster. Over these bounding boxes a top-level data structure is built. Because of the low number of bounding boxes (< 60 for our animations) this can be done very fast.

Ray traversal is done in two steps. The ray starts traversing the top-level structure. For each hitted cluster, the ray is transformed to the local coordinate system of the cluster and traverses the local data structure. The transformation is the affine transformation of the bone associated with the cluster.

In a preprocessing step the fuzzy boxes are calculated by exhaustive search. The fuzzy boxes are calculated for each triangle by assigning each triangle to each bone. The bone is selected that yields the smallest area for the fuzzy box. We sample the fuzzy boxes by taking samples from all the possible animation sequences. Günther argues that this gives good results and our experiments confirm this.

D. Motion Decomposition & Refitting

This method is a new approach to the problem and combines the ideas of motion decomposition & fuzzy data structures with refitting.

The problem with motion decomposition & fuzzy data structures are the fuzzy boxes. Using fuzzy boxes gives rise to a lot of overlap between the different nodes of the data structure resulting in lowered traversal performance. With refitting problems arise when the deformation with respect to the rest pose is severe, this results in a traversal performance as well.

Our approach uses the same principle as motion decomposition & fuzzy data structures but replaces the fuzzy boxes with tight bounding boxes. These bounding boxes need to be refit every frame to capture the small residual motion. The problem with the fuzzy boxes is solved and the topology of the local BVHs is conserved very well because the relative small residual motion of the triangles. The price we pay is the complexer traversal algorithm and the extra refitting step in each frame.

V. Comparison

We have implemented all the techniques on top of our own C++ ray tracer. The ray tracer uses the BVH as data structure with the same packet traversal scheme as described in [WBS07] but without the inverse frustum culling. The packets size is 8×8. We don't use SIMD instructions. Cal3d, a framework for skeleton based animations [cal], is used for handling the animations. Cal3d includes three characters and we use them to define four different animations (see figure 1):

Cally Walking The Cally character walking and waving to the camera.

Cally Freestyle The Cally character doing a 'karate kick'. This is an animation with 'heavy' motion.

Skeleton Freestyle The Skeleton character doing a little dance performance. Note that each vertex of the skeleton animation is influenced by only one vertex. This is a pure hierarchical (affine) animation.

Palading Jogging The Paladin character jogging while shooting an arrow from his 'virtual' bow.

The frames rate for the first 50 frames for each technique are shown in table I. All timings are obtained on a single core of a 2.4GHz Intel E6600 CPU.

A. Time-to-Image

Time-to-image is a very important metric. For example when ray tracing is used for computer games this is the price that has to be paid every frame. Locally high time-to-image will lead to 'shaky' animations, something we wish to avoid in computer games. In general this is the metric that needs to be minimized.

Figure 3 shows a comparison of the time-to-image for the different animations. In general, refitting is always the fastest technique, only for the Cally Freestyle animation (figure 3(b)) fast rebuilds becomes faster between frames 25 and 37. Reason for this is the severe motion in the animation.

The greatest time-to-image is for the motion decomposition & fuzzy data structures technique. Only for the Skeleton Freestyle animation the technique outperforms fast rebuilds and motion decomposition & refitting (see figure 3(c)). This due the fact that there's no residual motion in the animation so the fuzzy boxes are tight fitting boxes that are valid for the entire animation. This leads to higher traversal performance of the local data structures.

B. Render Time

A significant part of the time to image is the render time. An overview is given in figure 5.

For all the animations the refitting technique has the best render times. Only for the Cally Freestyle (see figure 4(b) animation fast rebuilds performs better due to the severe deformation of the BVH for the refitting technique.

Motion decomposition techniques perform the worst for all animations. Figure 4(c) shows that the render times for the two techniques are exactly the same for the Skeleton Freestyle animation. As further results will prove, this is because they use the same BVH.

C. # Ray-Triangle Intersections

To analyze the render times more in depth we take a look at the number of ray-triangle intersections and in the next section the number of ray-AABB (axis aligned bounding box) intersections. Counting the number of intersection tests is an implementation independent metric.

The motion decomposition techniques perform less ray-triangle tests. This is due to our implementation. Before actually doing the ray-triangle test an extra ray-AABB is performed with the current AABB of the triangle. This optimization saves

Frames Per Second (FPS)				
	Cally Walking	Cally Freestyle	Skelet Freestyle	Paladin Jogging
Fast Rebuilds	11.55	11.21	13.07	9.86
Refitting	12.44	11.37	14.36	10.35
Modecomp & Fuzzy Data Structures	8.76	8.78	13.81	6.97
Modecomp & Refitting	9.50	9.22	13.50	7.72

TABLE I

FRAMERATES FOR THE FOUR DIFFERENT TECHNIQUES FOR THE FIRST 50 FRAMES OF EACH ANIMATION.

us a lot of costly ray-triangle tests by doing one extra ray-AABB test.

The high peak in the number of ray-triangle tests for the Cally freestyle animation (figure 5(b)) confirms the heavy deterioration of the BVH.

Figure 5(c) confirms that the two motion decomposition techniques use exactly the same BVH, because the number of ray-triangle tests is exactly the same.

D. # Ray-AABB Intersections

The number of Ray-AABB intersection tests is another important part of the render time. This metric is like # ray-triangle tests implementation independent.

It's clear that the motion decomposition & fuzzy data structures approach does far more ray-AABB tests than all the other approaches. The reason is twofold. First there's a significant traversal penalty for using fuzzy boxes. Second there are some extra tests in the top-level data structure. Together with the extra time spent transforming rays to the local coordinate system of the bone (a matrix-vector multiplication), this is the reason for the worse render timings compared to the other techniques. Our own approach eliminates the fuzzy boxes and lowers the number of ray-AABB tests but still isn't able to outperform refitting and fast rebuilds.

E. Update Times

The update times are shown in figure 7. These timings reflect what we theoretically would expect from the different techniques:

1. Motion decomposition & fuzzy data structures is the fastest. The local structures are not updated and the top-level structure is rebuilt over a small number of bounding boxes.
2. Refitting must update the BVH, this has time complexity $O(N)$.
3. Motion decomposition & refitting must refit each local data structure and must rebuild the top-level structure which makes it slightly slower than refitting.
4. Fast rebuilds is about one magnitude slower because of the full rebuild. Rebuilding has time complexity $O(N \times log(N))$.

For the techniques the ratio update time versus time-to-image ranges from less than 1% up to more than 10%. Surprisingly, the update time of the refitting technique is very close to the update time of the motion decomposition & fuzzy data structures.

VI. CONCLUSION

After comparing several aspects of the techniques it's time for our final conclusion. All techniques have some advantages and disadvantages so it's difficult to pick one winner for all situations. Although we will try to give some pointers.

Refitting is overall the best performer. The render times are good and comparable with those of a BVH rebuilt every frame. Because of the low update cost it outperforms fast rebuilds.

Although motion decomposition & fuzzy data structures achieves interactive frame rates it's surprisingly less performant than fast rebuilds and refitting for skeleton based animations. The problem is the costly traversal algorithm. This cost cannot be gained back by the cheap update step in each frame.

Our own approach suffers from the same problem as the motion decomposition & fuzzy data structures technique. Not using fuzzy boxes buys some traversal performance but not as much as expected initially. Also in this approach the complexer traversal scheme is too costly to outperform refitting an fast rebuilds.

When the number of triangles in the models increases significantly there's no way around motion decomposition & fuzzy data structures. The cost of updating the acceleration structure now dominates the rendering cost for the other techniques.

Some remark has to be made about the motion decomposition & fuzzy data structures technique, Günther et al. proposed this technique to be used with kd-trees. We believe that this implementation would outperform our BVH based implementation but we don't think that it will outperform the other techniques. This because it suffers from the same complexer traversal scheme.

VII. FUTURE WORK

We think that future research shouldn't focus on a specific niche like skeleton based animations but on more general classes of animations. The solution to the problems lies for the most part in finding suitable data structures that are cheap to build or update and provide reasonable traversal performance.

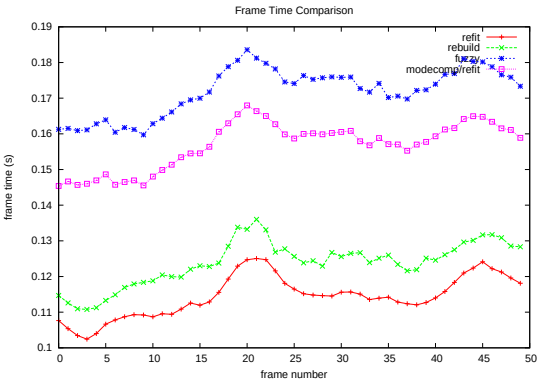
Interesting new data structures are emerging like the *B-KD tree* [WMS06], the *Bounding Interval Hierarchy (BIH)* [WK06] or *constrained tetrahedralizations* [LD08]. A comparison of these methods with the classic data structures (grids, kd-trees and BVHs) could give some new insights in current research. Ideally a full comparison should be done for dynamic data structures the same way Havran did for static data structures [Hav00].

The surface area heuristic works very good for static scenes and is used for dynamic scenes as well. The heuristic concen-

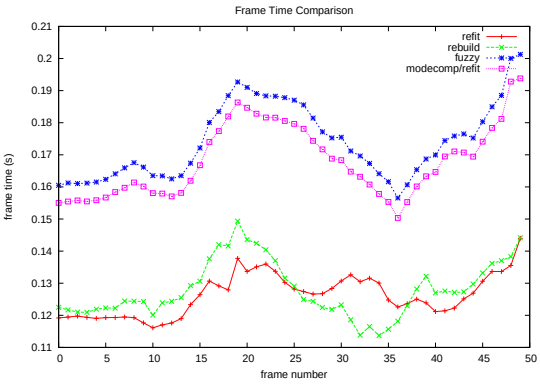
trates on building a high quality data structure for one triangle configuration. This suggest a heuristic for animations that combines triangle information from several frames. For example a heuristic for refitting could be used for building a data structure that is optimal for several frames under the refitting operation. The heuristic is able to calculate how many frames the data structure can 'stretch' without rebuilding. With this heuristic we could combine refitting and rebuilding like proposed in [LeYM06]. The heuristic predicts in advance in which frames rebuilding is necessary instead of calculating this every frame.

VIII. Acknowledgments

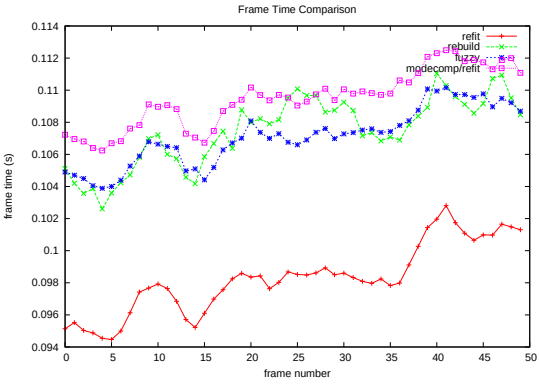
We would like to thank Ares Lagae and Philip Dutré for their guidance and advice. Special thanks to Johannes Günther for his help with understanding and implementing the motion decomposition & fuzzy data structures approach.



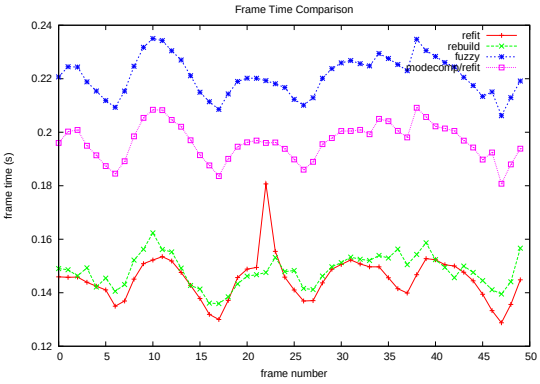
(a) Time-to-image comparison Cally Walking animation



(b) Time-to-image comparison Cally Freestyle animation

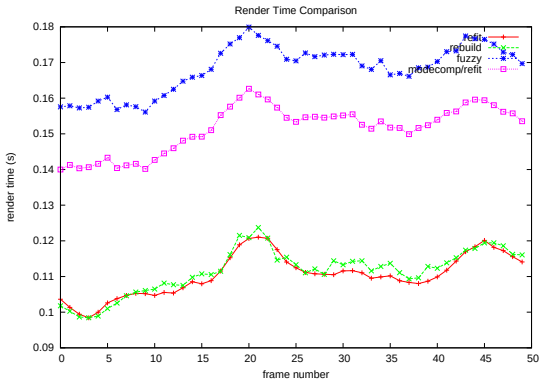


(c) Time-to-image comparison Skeleton Freestyle animation

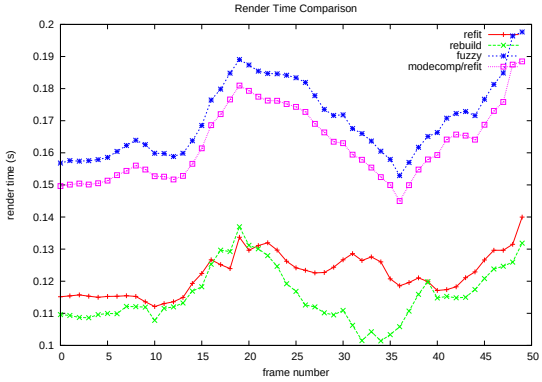


(d) Time-to-image comparison Paladin Jogging animation

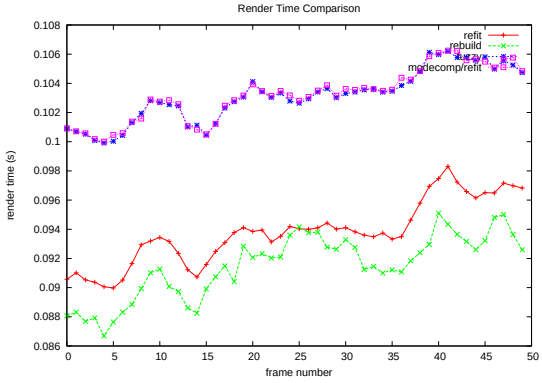
Fig. 3. Time-to-image comparison for each technique.



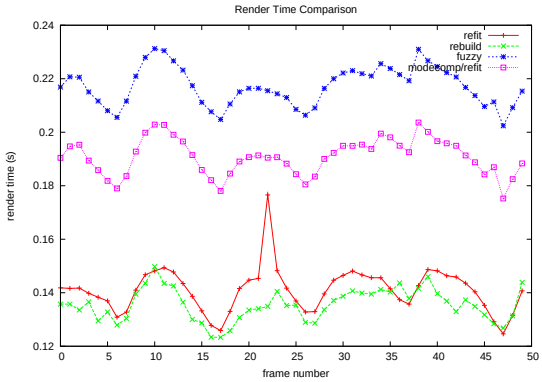
(a) Render time comparison Cally Walking animation



(b) Render time comparison Cally Freestyle animation

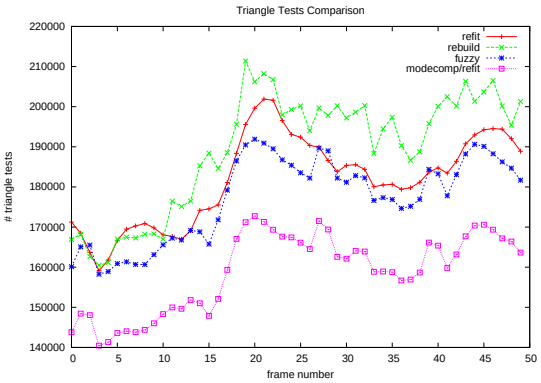


(c) Render time comparison Skeleton Freestyle animation

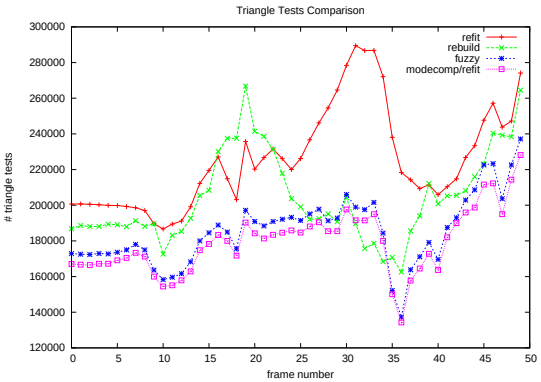


(d) Render time comparison Paladin Jogging animation

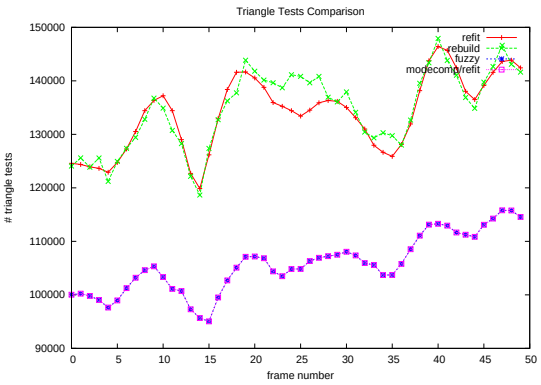
Fig. 4. Render time comparison for each technique.



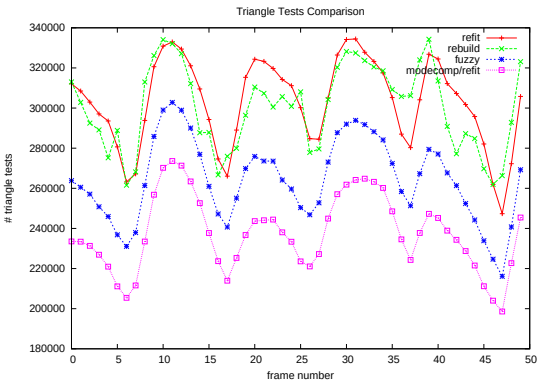
(a) # Ray-Triangle Intersections comparison Cally Walking animation



(b) # Ray-Triangle Intersections comparison Cally Freestyle animation

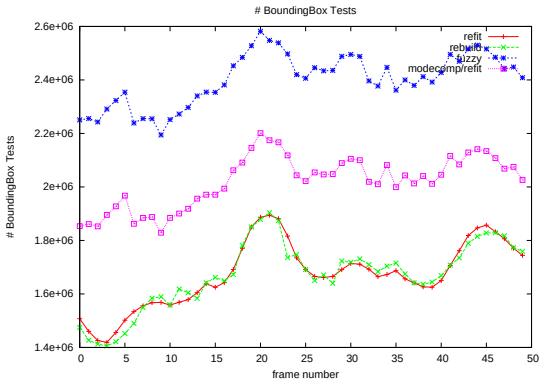


(c) # Ray-Triangle Intersections comparison Skeleton Freestyle animation

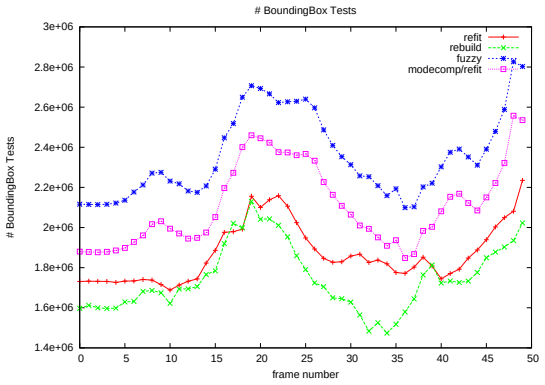


(d) # Ray-Triangle Intersections comparison Paladin Jogging animation

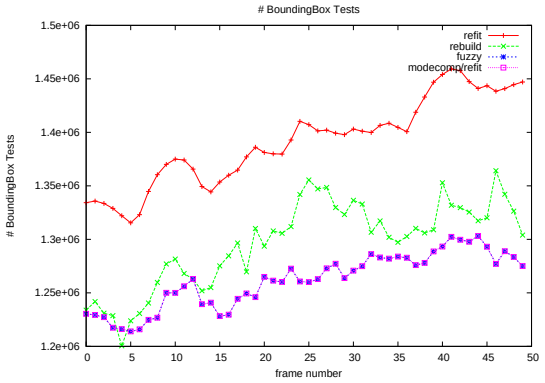
Fig. 5. # Ray-Triangle test comparison for each technique.



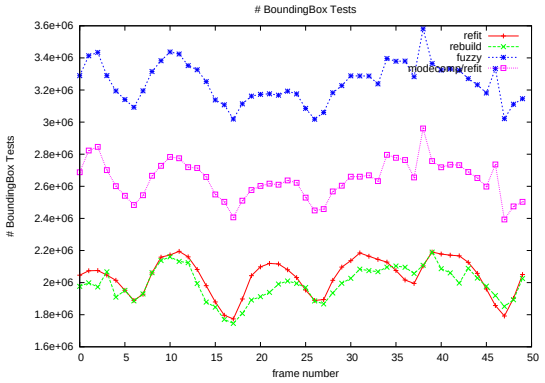
(a) # Ray-AABB Intersections comparison Cally Walking animation



(b) # Ray-AABB Intersections comparison Cally Freestyle animation

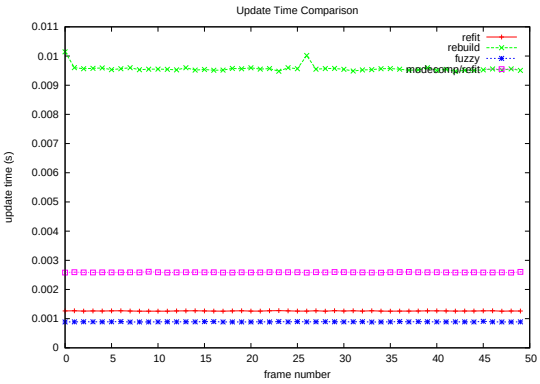


(c) # Ray-AABB Intersections comparison Skeleton Freestyle animation

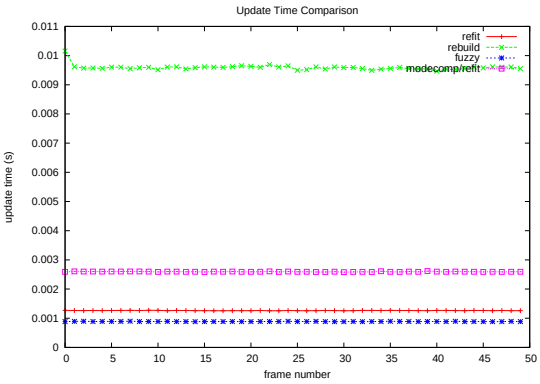


(d) # Ray-AABB Intersections comparison Paladin Jogging animation

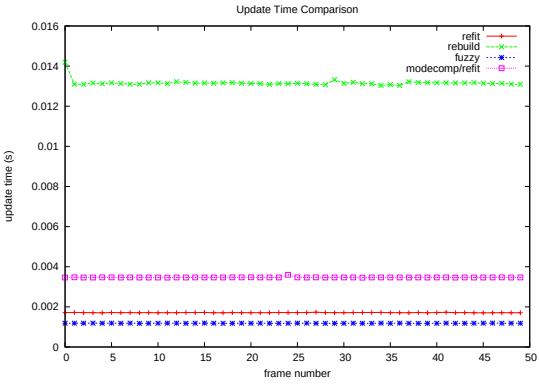
Fig. 6. # Ray-AABB test comparison for each technique.



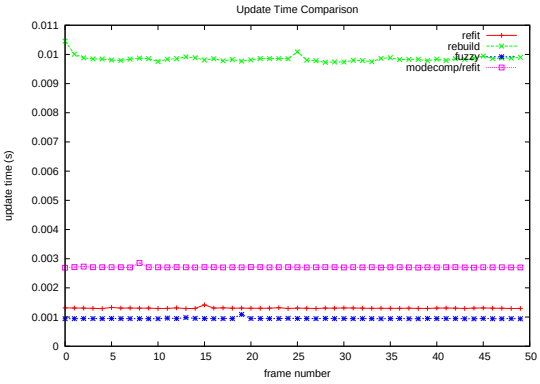
(a) Update time comparison Cally Walking animation



(b) Update time comparison Cally Freestyle animation



(c) Update time comparison Skeleton Freestyle animation



(d) Update time comparison Paladin Jogging animation

Fig. 7. Update time comparison for each technique.

REFERENCES

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[cal] Cal3d. 3d character animation library. `http://home.gna.org/cal3d/`.

[GFSS06] Johannes Günther, Heiko Friedrich, Hans-Peter Seidel, and Philipp Slusallek. Interactive ray tracing of skinned animations. *The Visual Computer*, 22(9):785–792, September 2006. (Proceedings of Pacific Graphics).

[GFW+06] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum*, 25(3):517–525, September 2006. (Proceedings of Eurographics).

[Hav00] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[LD08] Ares Lagae and Philip Dutré. Accelerating ray tracing using constrained tetrahedralizations. *Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering)*, 27(4):1303–1312, June 2008.

[LeYM06] Christian Lauterbach, Sung eui Yoon, and Dinesh Manocha. Rt-deform: Interactive ray tracing of dynamic scenes using bvhs. In *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

[MTLT88] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[Wal04] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[Wal07] Ingo Wald. On fast construction of sah-based bounding volume hierarchies. *Symposium on Interactive Ray Tracing*, 0:33–40, 2007.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In Alan Chalmers and Theresa-Marie Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001. available at http://graphics.cs.uni-sb.de/ wald/Publications.

[WH06] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). pages 61–69, September 2006.

[WIK+06] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, pages 485–493, 2006. (Proceedings of ACM SIGGRAPH 2006).

[WK06] Carsten Wchter and Er Keller. Instant ray tracing: The bounding interval hierarchy. In *In Rendering Techniques 2006 Proceedings of the 17th Eurographics Symposium on Rendering*, pages 139–149, 2006.

[WMG+07] Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In Dieter Schmalstieg and Jirí Bittner, editors, *STAR Proceedings of Eurographics 2007*, pages 89–116, Prague, Czech Republic, September 2007. Eurographics Association.

[WMS06] Sven Woop, Gerd Marmitt, and Philipp Slusallek. B-kd trees for hardware accelerated ray tracing of dynamic scenes. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 67–77, New York, NY, USA, 2006. ACM.

Interactieve Ray Tracing van Skeletgebaseerde Animaties

KATHOLIEKE UNIVERSITEIT LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Master Computer-wetenschappen

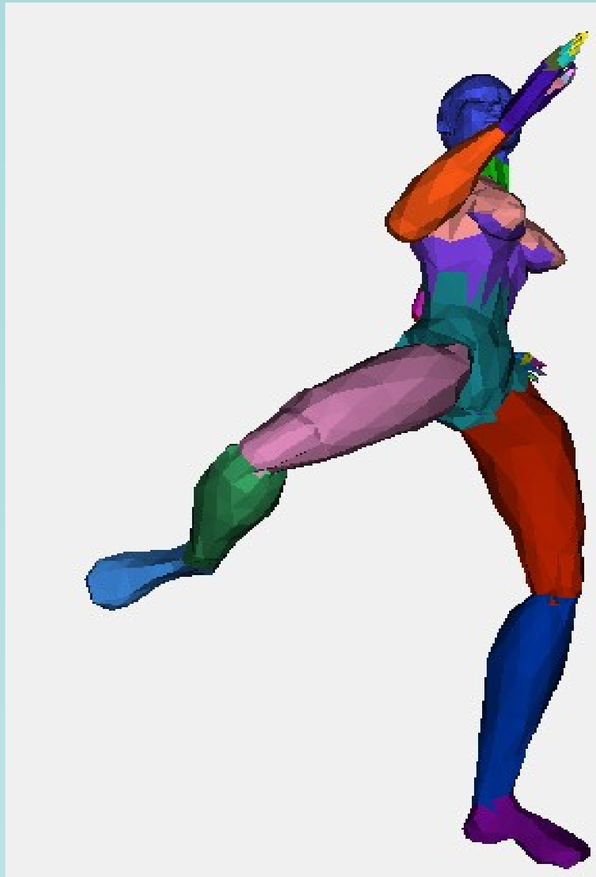
Masterproef *Thomas Loockx*

Promotor *dr.ir.Ph.Dutré*

Academiejaar 2008-2009

Situering

Ray tracing is een belangrijk rendering algoritme in het domein van computer graphics. Lange tijd werd ray tracing veel te traag beschouwd voor interactieve toepassingen. Dankzij nieuwe ontwikkelingen is de snelheid van ray tracing spectaculair gestegen waardoor het algoritme kan toegepast worden in nieuwe domeinen zoals dat van skeletgebaseerde animaties.



Doelstelling

- Implementeren van een interactieve ray tracer.
- Onderzoek van verschillende technieken voor skeletgebaseerde animaties.
- Grondige vergelijking van de verschillende technieken.

Resultaten

- Interactieve ray tracing is mogelijk op hedendaagse hardware.
- Eigen implementatie haalt snelheden tot 14 frames per seconde voor resoluties van 512x512 pixels.
- Speciale technieken voor skeletgebaseerde animaties zijn niet noodzakelijk beter dan algemenere technieken.

Toepassingen

- Computer games, animatiefilms, modellering, ...