Neopica | Uhasselt

Dynamic Mapping for 3D Mesh Creation

Masterproef Gunter Schreurs

Gunter Schreurs
Academiejaar 2014-2015

I. Inhoudsopgave

II. Voorwoord

Als eerste wil ik mijn promotor en begeleider Professor Quax bedanken voor de begeleiding doorheen het volledige project. Daarnaast wil ik ook de heer Filip Hautekeete bedanken voor de constante hulp bij het verwerken van de thesis. Zonder de begeleiding en hulp van beide had mijn thesis nooit dit niveau gehaald. Daarnaast wil ik ook alle andere mensen bedanken die rechtstreeks of onrechtstreeks hebben geholpen tijdens de verwerking van mijn thesis. Daarbij horen in het bijzonder mijn ouders die altijd bereid waren hun kennis te delen en die mij telkens gesteund hebben.

1.Introductie

1.1 Masterproef Beschrijving

De masterproef focust op Augmented Reality (hierna afgekort als AR), waarbij specifiek aandacht wordt besteed aan toepassingen van deze technologie zoals serious games, educational games, training etc. Het is voor gamebedrijven (voor deze thesis wordt specifiek samengewerkt met Neopica) interessant om snel en eenvoudig content aan te kunnen maken. Hierbij is het belangrijk dat deze content in een 3D wereld kan worden geplaatst. Er moet vooral aandacht worden geschonken aan reeds gekende technieken die bruikbare resultaten geven met enkele artefacten. De bedoeling is te kijken op welke manier deze artefacten kunnen opgelost worden om zo een volwaardig 3D model te creëren. Hiervoor wordt er een analyse gemaakt over de state of the art technologieën en processen voor AR-content te creëren. De focus ligt hierbij op captatie technieken die zorgen voor het snel en eenvoudig aanmaken van 3D modellen vanuit objecten in de wereld. Alle concepten die hier van toepassing zijn worden besproken. Er is een pipeline die beschrijft hoe het probleem mogelijk wordt aangepakt. Deze pipeline beschrijft het volledige verloop van het digitaliseren van het object (captatie) tot het tonen van een 3D representatie op het beeldscherm (visualisatie). De volledige pipeline wordt overlopen doorheen de thesis. Er wordt een iteratief proces gebruikt voor het tot stand komen van deze thesis. Initieel wordt een high-level analyse gemaakt van de verschillende stappen die nodig zijn om een dergelijk systeem te bouwen (end-to-end, dus van captatie tot visualisatie en interactie). Doorheen deze analyse worden verschillende elementen van het proces vergeleken met elkaar. Hierbij wordt er vooral aandacht gelegd op elementen van het proces die niet optimaal zijn voor het genereren van game-gerelateerde objecten. Voor deze objecten is het namelijk belangrijk dat ze dynamisch zijn. Hiermee wordt bedoeld dat het eenvoudig moet zijn ze te kunnen animeren en vrij geplaatst kunnen worden in een 3D omgeving. Daarnaast moet het ook eenvoudig zijn om verschillende modellen te kunnen creëren op een korte tijd. Na de analyse volgt er een implementatie van de pipeline waarbij reeds gekende concepten gebruikt worden, samen met nieuwe concepten, om zo enkele deelproblemen aan te pakken.

Bij de implementatie van het concept zal er zoveel mogelijk gebruik worden gemaakt van off-the-shelf beschikbare producten. Gezien de recente ontwikkelingen op het gebied van AR zijn er veel deelproblemen reeds uitvoerig onderzocht. De laatste jaren zijn Augmented Reality en Virtual Reality twee vormen van games geworden die zeer veel aandacht krijgen. Zeker VR heeft hierbij enorme stappen vooruit gemaakt. Er kan bij hardware worden gedacht aan VR brillen zoals de Oculus Rift of Gear VR. Ook in de AR zijn er op gebied van hardware enkele nieuwe technieken op de markt gekomen zoals Google Glass waarbij een bredere markt wordt gezocht. Maar ook in specifieke toepassingen worden er ontwikkelingen maakt. Zo zijn er veel AR brillen beschikbaar voor skiërs die beelden kunnen opnemen, snelheden kunnen weergeven en andere statistieken kunnen bijhouden. Ondanks de grote stappen voorwaarts op het gebied van hardware zijn er enkele elementen van het conceptvoorstel op het moment

van schrijven nog niet mogelijk. Hierbij wordt vooral gedacht aan het real-time werken van het algoritme.

Naast de vooruitgang op het gebied van hardware voor AR is het ook opmerkelijk hoe groot de vooruitgang is op het gebied van het scannen van objecten naar 3D modellen. Het onderzoek domein voor AR bestaat reeds een lange tijd, maar toch wordt er hedendaags nog veel onderzoek gedaan naar het verbeteren van de gekende algoritmen of het vinden van nieuwe algoritmen. Vooral voor high-end toepassingen vormt dit nog steeds een probleem. Om hoge kwaliteit 3D modellen te maken door middel van scanners is er een hoge kost voor het materiaal met daarnaast een grote verwerkingstijd om onzuiverheden op te lossen.

De implementatie zelf zal zicht richten op een optimalisatie techniek voor het snel te kunnen aanmaken van volledige 3D modellen. Er wordt vooral gekeken naar instanties waar objecten worden gedigitaliseerd en over netwerken moeten worden gedeeld, of waar de 3D modellen eenvoudig moeten worden geanimeerd. Hiervoor wordt een techniek voorgesteld die de naam Skeleton Mapping For Kinect mee krijgt. De bedoeling is om verkregen input via 3D scanning hardware (de Kinect van Microsoft) direct te plaatsen op een 3D karakter dat eenvoudig te animeren is. De focus binnen de implementatie zal liggen op het proces waarbij het skelet volledig wordt aangemaakt zonder het te animeren. De moeilijkheid ligt hier in het herkennen van lichaamsdelen en het correct omzetten van coördinaatstelsels. Daarnaast ligt ook een moeilijkheid bij het wegfilteren van fouten die worden gemaakt bij de captatie stap.

1.2 Type masterproef en onderzoeksvragen

De beginvraag die werd gesteld voor de masterproef is: “Hoe kan captatie helpen bij het genereren van content voor AR-games?”. Deze vraag wordt aangepakt in de eerste hoofdstukken (2,3 en 4) van de thesis. Hierbij worden mogelijke technieken besproken en vergeleken. Dit geeft een algemeen overzicht van de mogelijke aanpakken om door middel van captatie content te creëren voor AR-games. Een tweede onderzoeksvraag gaat in dieper in op de vraag “Zijn er deelproblemen voor AR-toepassing?”. Deze vraag wordt beantwoord doorheen de thesis waar verschillende voor- en nadelen van bepaalde technieken worden aangehaald. Deze eerste twee onderzoeksvragen bevinden zich onder eenzelfde type masterproef namelijk een Domain Study, waar het volledige domein in verband met het creëren van 3D modellen aan de hand van captatie wordt besproken.

De tweede onderzoeksvraag geeft echter enkele problemen met de huidige technologieën. Aan de hand hiervan wordt Skeleton Mapping voorgesteld als een concept dat de kwaliteit van de modellen wilt verhogen en daarnaast zorgen voor een verlaging van de bandbreedte bij het streamen van deze data over een netwerk. Tot slot focust het ook op de eenvoudige animatie en creatie van deze modellen. De onderzoeksvraag die hieruit voorkomt is: “Kan Skeleton Mapping zorgen voor een verhoging van de kwaliteit van een object zodat deze eenvoudig in AR toepassingen kan worden gebruikt?”. Deze vraag wordt beantwoord aan de hand van een concept-implementatie in het laatste hoofdstuk van deze thesis. Dit behoort echter niet meer tot de Domain Study, zoals de vorige vragen, maar behoort tot een Feasiblity Study.

1.3 Structuur van de tekst

De tekst is opgedeeld in twee grote delen. Het eerste gedeelte beschrijft de volledige Domain Study zoals verklaard in de vorige sectie. Dit gedeelte beslaat hoofdstuk 2 tot en met hoofdstuk 4. Daarnaast is er de Feasiblity Study die staat uitgeschreven in hoofdstukken 5 en 6. Daarnaast is er ook dit inleidende hoofdstuk met een algemene introductie tot de tekst. Het laatste hoofdstuk van de thesis bevat de conclusie die bestaat uit een reflectie over de thesis en de uiteindelijke implementatie.

Hoofdstuk 2 beslaat een algemeen overzicht van het capteren en visualiseren van objecten. Hierbij wordt eerst de algemene pipeline besproken. Deze bestaat uit twee verschillende delen. Ieder deel werkt volledig onafhankelijk van het andere, al dient de output van het eerste deel wel als input voor het tweede deel. Het is belangrijk op te merken dat de pipeline een voorstel is, dat wordt gebaseerd op meerdere papers. Zoals bij de meeste technologieën, is er geen vaste pipeline die gevolgd moet worden om tot een oplossing te komen, maar zijn er meerdere mogelijkheden om het probleem aan te pakken. Er is echter wel een algemeen aanvaarde opvolging van sub-processen, die wordt gebruikt in de meeste toepassingen, die captatie en visualisatie van objecten doen. Er wordt dus ook vanuit deze pipeline gewerkt voor het bespreken van alle sub-processen. De bedoeling in hoofdstuk twee is om enkel dit algemeen beeld over te brengen en dus niet te praten over de mogelijke invullingen van de sub-processen.

Na de algemene bespreking van de pipeline wordt elk deel van deze pipeline besproken. *Hoofdstuk 3* beslaat het hele gebied in verband met Data Generatie. Hier worden de technieken besproken; die gebruikt worden in het proces voor het genereren van 3D modellen. Hiertoe behoort het binnenhalen van 3D data door middel van 3D Acquisition (sectie 3.1). Hierbinnen worden verschillende technieken besproken die worden gebruikt voor het digitaliseren van werelddata, alsook de verschillende hardware die hiervoor nodig is. Daarnaast worden ook enkele stappen besproken die nodig zijn om de input van deze hardware te verwerken. Daarna volgen de optimalisatie technieken voor het verkregen model te verbeteren. Eerst wordt surface reconstruction (sectie 3.2) besproken. Deze techniek zorgt voor het omzetten van puntenwolken in 3D meshes of voor het beteren van de kwaliteit van de puntenwolken door het toevoegen van extra punten. Daarna worden enkele stappen besproken in verband met processing (sectie 3.3). Deze algoritmen worden gebruikt voor het verwijderen van fouten in de uiteindelijke oplossing. Dit kan gaan van ruis (punten die afwijken van hun werkelijke positie) tot het opvullen van gaten in het object (die zijn ontstaan door occlusies). Daarnaast behoort ook het samenvoegen van diverse puntenwolken uit verschillende opnames tot processing. Het laatste deel van hoofdstuk 3 bespreekt de verschillende opslagtechnieken voor deze data (sectie 3.4). Het gaat specifiek over de keuze tussen voxel representaties of 3D mesh representaties. Er wordt hierbij ook al even gekeken naar Skeleton Mapping, een eigen voorstel waarbij karakter data wordt opgeslagen aan de hand van skeletten. De focus ligt dus op de verschillen tussen de verschillende technieken en de voordelen die deze bieden in verdere stadia van de pipeline.

Het tweede gedeelte van de algemene pipeline wordt in *hoofdstuk 4* besproken. Dit gedeelte van de pipeline bevat alle elementen die nodig zijn voor het visualiseren van de data. Eerst

worden alle elementen besproken die behoren tot de rendering (sectie 4.1). Er wordt een verschil gemaakt tussen het renderen van polygonen of het renderen van voxels. Net zoals bij sectie 3.4 waar de verschillen tussen de opslagtypes worden besproken, worden hier de verschillen tussen de gebruikte technieken besproken. Daarna wordt het gedeelte netwerken besproken (sectie 4.2). Hierbij gaat het vooral over de redenen waarom het streamen van polygonen en voxels niet efficiënt is tijdens het verdelen van 3D modellen. Dit is ook een grote reden waarom Skeleton Mapping wordt uitgewerkt als concept omdat hiervoor een lagere bandbreedte nodig is voor het synchroniseren van data. Tot slot wordt er gepraat over het animeren van verkregen objecten (sectie 4.3). Er worden eerst enkele representatie technieken besproken van karakters, gevolgd door het bespreken van twee verschillenden aanpakken van skelet animatie (kinematics).

In *hoofdstuk 5* wordt het eerste deel van een eigen implementatie besproken. Het gaat hier over de implementatie van het algoritme en idee dat wordt besproken in sectie 1.4. In dit hoofdstuk wordt dieper ingegaan op de captatie van data en de analyse van deze data. Er wordt gewerkt aan de hand van de Kinect, die ontwikkeld werd voor de Xbox consoles van Microsoft. Tijdens de bespreking worden zowel slechte als goede oplossingen onder de loep genomen. De nadruk ligt steeds op het verklaren van de genomen keuzes alsook de werking van de uiteindelijke implementatie. Daarnaast wordt er ook gekeken naar het opnemen van data door middel van meerdere scantoestellen. De alignatie van de verschillende datastromen en het encoderen van deze stromen is hier prioriteit. Ook het versturen van de commando's over een netwerk wordt kort toegelicht.

In het *6de hoofdstuk* worden alle elementen met betrekking tot het herkennen, mappen en visualiseren van de data besproken. Eerst volgt een algemene inleiding in verband met het concept van mapping en animatie. Daarna worden enkele herkenningsalgoritmen voor het detecteren van de correcte lichaamsdelen bij 3D punten besproken. Hoe bepaalde keuzes voor te gebruiken algoritmen zijn gemaakt, wordt ook verklaard aan de hand van voorbeelden en fouten in voorgaande technieken. Na de bespreking van de skelet herkenning wordt er dieper ingegaan op het mappen van de punten. Hierbij wordt kort aangetoond dat de techniek steunt op wiskundige transformaties. Daarnaast wordt ook gekeken naar de eventuele mogelijkheden van het algoritmen. Tot slot wordt er gekeken naar het visualiseren en animeren van de data. Dit wordt bekeken voor zowel puntenwolken als polygonen. Omdat polygonen reeds door vele toepassingen worden geanimeerd, wordt er niet gekeken naar dergelijke animatie technieken tijdens de implementatie. Meer informatie daarover is terug te vinden in de domeinstudie.

Het laatste hoofdstuk van deze thesis bevat de conclusie. Hierbij wordt er kritisch gekeken naar alle hoofdstukken alsook de uiteindelijke conclusie die kan worden getrokken voor Skeleton Mapping. Daarbij wordt er ook nagegaan wat er in de toekomst mogelijk zou zijn met dit concept.

1.4 Skeleton Mapping For Kinect

Zoals in de vorige secties wordt vermeld, bestaat deze thesis uit twee onderdelen. Een eerste onderdeel zorgt voor de bespreking van de huidige technologie voor het scannen en

visualiseren van 3D-data. Het tweede deel bespreekt een implementatie die is gemaakt om enkele technieken uit te testen alsook te kijken of het mogelijk is om op een eenvoudige manier een 3D personage aan te maken. Deze techniek wordt eerst toegelicht zodat het duidelijk is voor de lezer waarom alle elementen binnen de domein study worden aangehaald. Skeleton Mapping is een combinatie van verschillende algoritmen. Deze combinatie maakt het mogelijk een bewegende 3D puntenwolk om te vormen naar een 3D karakter. Het hoofddoel van het algoritmen is het wegwerken van artefacten, die in huidige toepassingen worden gevonden, met betrekking tot het 3D filmen van karakters voor een 3D wereld.

Skeleton Mapping For Kinect (voortaan afgekort als SMK) is een manier om content eenvoudig te digitaliseren en om te zetten naar een animeerbaar 3D karakter. Er zijn reeds technieken die toelaten eenvoudig animaties te creëren voor games (zie sectie 4.3) en voor het digitaliseren van objecten (zie hoofdstuk 1). Deze technieken zijn echter niet compatibel of zeer omslachtig in gebruik. SMK biedt een combinatie van deze twee technieken aan om zo zeer snel en eenvoudig een geanimeerd personage in 3D te creëren. De techniek maakt gebruik van één of meerdere puntenwolken van eenzelfde persoon die worden afgeleverd door de Kinect. Hoe meer puntenwolken hoe meer detail kan worden gegenereerd. Deze verschillende puntenwolken van een bepaald persoon worden dan geanalyseerd om tot slot te worden geprojecteerd op een digitaal skelet. Hierbij is het noodzakelijk dat er meerdere stappen worden ondernomen om een zo hoog mogelijke kwaliteit te kunnen afleveren. De eerste stap in de analyse is het herkennen van de punten die behoren tot een bepaald persoon. Alle informatie over de achtergrond is namelijk onnodig voor het construeren van een nieuw model voor de persoon. De volgende belangrijkste stap bestaat uit het herkennen van ledematen en welke punten tot welke ledematen behoren. Dit is nodig zodat de volgende stappen, waaronder het mappen van de punten op het skelet en het animeren van de lichaamsdelen, goed kunnen verlopen en zodat er na afloop eenvoudig aanpassingen aan het skelet kunnen worden gemaakt. Op deze manier kan het skelet nadien worden geanimeerd. Eenmaal een punt herkend is kan dit worden opgeslagen in een puntenwolk. Er wordt één puntenwolk bijgehouden voor iedere ledemaat, dat kan worden herkend. In het kort samengevat bevat het Skeleton Mapping concept de volgende elementen: het opnemen van 3D puntenwolken en deze opsplitsen in de verschillende geregistreerde lichaamsdelen. Deze verschillende puntenwolken worden dan omgevormd zodat ze passen op een extern skelet. Tot slot laat Skeleton Mapping for Kinect ook direct remapping toe op andere skeletten. Zo is het mogelijk een opgenomen karakter te animeren door acties van andere personen.

2.High-Level Analyse

In het vorige hoofdstuk werd enkele malen gesproken over een pipeline die ervoor zorgt dat objecten kunnen worden ingelezen naar 3D modellen. Dit hoofdstuk biedt een mogelijke interpretatie van de pipeline en zijn componenten. Hiervoor wordt er eerst een overzicht gegeven van wat net de scope is. In de hierop volgende hoofdstukken worden alle elementen uitgewerkt zodat deelproblemen zichtbaar worden.

2.1 Scope Analyse

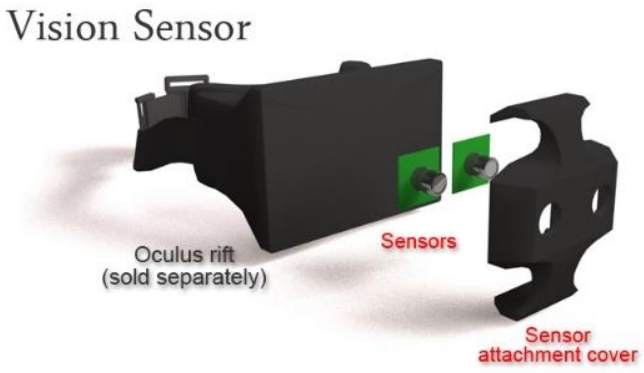
Zoals vermeld in sectie 1.1 is het doel van deze masterproef een overzicht te bieden voor het aanmaken van 3D modellen voor game-development. Daarvoor wordt er eerst gekeken naar de huidige technieken die worden gebruikt voor het;

- scannen van objecten,
- verbeteren van het verkregen 3D model,
- comprimeren van het 3D model,
- versturen van de uiteindelijke data over een netwerk.

Figuur 1: Google Glass (www.google.com/glass)

Hierbij is het noodzakelijk op te merken dat de hardware waarop de uiteindelijke beelden worden weergegeven see-through schermen zijn zoals Google Glass (zie Figuur 1). Deze schermen hebben de mogelijkheid om digitale contexten weer te geven, maar laat de gebruiker nog steeds in staat om door deze schermen heen te kijken alsof het glas is.

Voor game-development met deze hardware zijn er echter nog steeds enkele beperkingen. Deze schermen hebben geen Graphics Processing Unit (GPU) die kan zorgen voor het

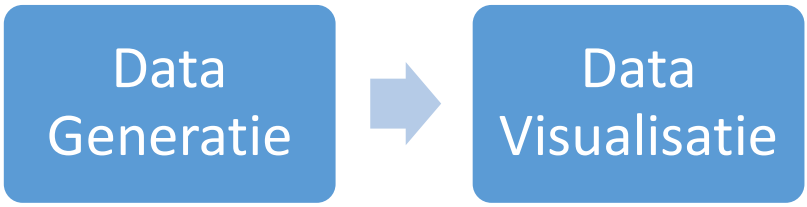


Figuur 2: Augmented Reality via Virtual reality headsets. (http://ovrvision.com/)

berekenen van 3D weergaven van een hoge kwaliteit. Daarnaast bevinden de huidige schermen zich maar in een klein deel van het gezichtsveld waarbij games het echter nodig is om het volledige gezichtsveld te beslaan. Er bestaan enkele methoden om dit te simuleren zoals het gebruik van een Oculus Rift (Virtual Reality Headset, zie Figuur 2) waarbij een camera wordt gebruikt om het zicht van de persoon weer te geven op de schermen. Omdat huidige technologie nog niet voldoende krachtig is om deze applicaties te draaien, zal er gebruik worden gemaakt van krachtigere toestellen om deze beelden te berekenen en simuleren (desktop waarbij camera beelden worden gezien als de achtergrond van het beeld).

Om een goed beeld te krijgen van welke elementen allemaal zitten in deze masterproef, wordt er een pipeline opgesteld om te kunnen beginnen aan de analyse. Hierbij wordt er gekeken naar reeds bekende oplossingsmethoden. De pipeline in deze masterproef is een mogelijke manier van aanpak waarbij een real-time weergave geen vereiste is. Hierdoor kan er meer aandacht worden besteed aan het verbeteren van het 3D model tijdens processtappen.

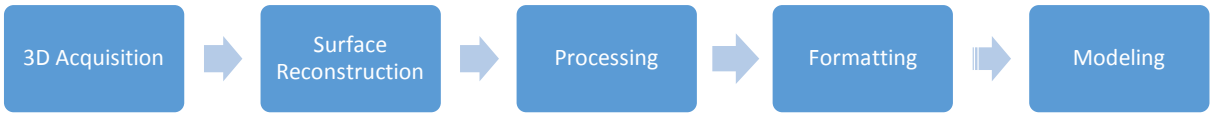
2.2 Pipeline Structuur



Figuur 3: De relatie tussen Data Generatie en Data Visualisatie.

Het volledig proces bestaat uit twee grote secties respectievelijk “Data Generatie” en “Data Visualisatie”. Beide elementen kunnen worden gezien als verschillende processen die hun eigen verloop hebben. Dit komt omdat beide processen in tijd onafhankelijk van elkaar zijn. “Data Generatie” kan gebeuren voor alle 3D modellen waarbij er nog geen directe visualisatie wordt gemaakt. Op een later moment kan een visualisatie worden gedaan van de eerder verzamelde data. Evenwel is er een belangrijke samenhang tussen de twee processen. Dit komt aan bod bij het laatste punt van “Data Generatie”.

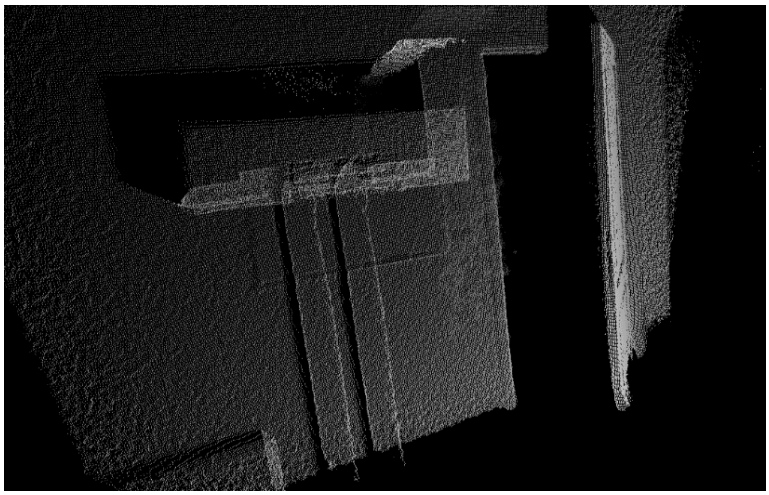
2.2.1 Data Generatie



Figuur 4: Data Generatie pipeline; modelling is een optionele stap in dit proces.

De bovenstaande pipeline is een voorstelling van het Data Generatie proces dat objecten kan omvormen naar 3D modellen in een virtuele omgeving. Om dit te kunnen realiseren zijn er enkele belangrijke stappen nodig. Het uiteindelijk verkregen 3D model van het volledige proces moet een zo hoog mogelijke kwaliteit zijn. Om dit te realiseren, wordt bij elke stap hier aandacht aan besteedt. De volgende alinea’s verklaren alle componenten en wat ze bijbrengen aan het volledige proces.

3D Acquisition zorgt voor het inscannen van een object en het genereren van een zogenaamde puntenwolk. Puntenwolken zijn namelijk het digitale resultaat van een 3D scanner. Een puntenwolk bestaat uit punten (al dan niet georiënteerd) die, indien samen weergegeven, een model voorstellen. Het verschil tussen georiënteerde en niet georiënteerde puntenwolken kan worden terug gevonden in sectie 3.1.1. Er zijn echter meerdere stappen nodig om ervoor te zorgen dat de punten van het 3D model op correcte wijze worden verwerkt. Deze stappen worden in het vervolg van de pipeline toegepast.



Figuur 5: Voorbeeld van een puntenwolk, aangemaakt met een Microsoft Kinect V2

Surface Reconstruction is een benaming voor een groep algoritmen die proberen een model te reconstrueren vanuit een puntenwolk. Deze algoritmen proberen aan de hand van de 3D puntenwolk die wordt gescand, een digitaal model te creëren dat zo dicht mogelijk aansluit bij de contouren van het object. Deze algoritmen proberen dit te doen door extra punten te genereren volgens tendensen in het gescande oppervlak. Door deze technieken toe te passen is er zo meer data beschikbaar om het model te verbeteren. Bij het aanmaken van extra punten moet er echter ook worden gekeken dat ruis geen grote invloed heeft op de nieuwe data. Ruis is een benaming voor de afwijkende elementen binnen de data, deze kunnen ontstaan door een inaccurate scanner of door extreme omstandigheden zoals mist. Ook ontbrekende data van het object mag niet over het hoofd worden gezien. Het negeren van deze problemen zorgt namelijk voor een 3D model dat met zeer lage kwaliteit is gedigitaliseerd met een reeks artefacten. Artefacten zijn fouten in het resultaat, die ontstaan door het slecht verwerken van de verkregen data.

Processing is een stap uit de pipeline waarbij de verkregen 3D worden gecorrigeerd. Eerst en vooral worden de artefacten, die er nog inzitten, eruit gehaald. Deze artefacten zijn meestal uitschieters die verkeerd zijn opgenomen, of zelfs ontbrekende stukken data die niet zijn geregistreerd. Ook al proberen de Surface Reconstruction algoritmen dit probleem ook aan te pakken, toch zijn deze algoritmen vaak niet geoptimaliseerd om alle uitschieters op te merken, daarvoor worden specifieke algoritmen gebruikt. Daarnaast gebruikt deze stap ook enkele geavanceerde technieken om de kwaliteit van een puntenwolk te verhogen. Hieronder vallen bijvoorbeeld technieken die meerdere puntenwolken samenvoegen aan de hand van een relatieve verhouding tussen de twee (of meerdere) opnameposities.

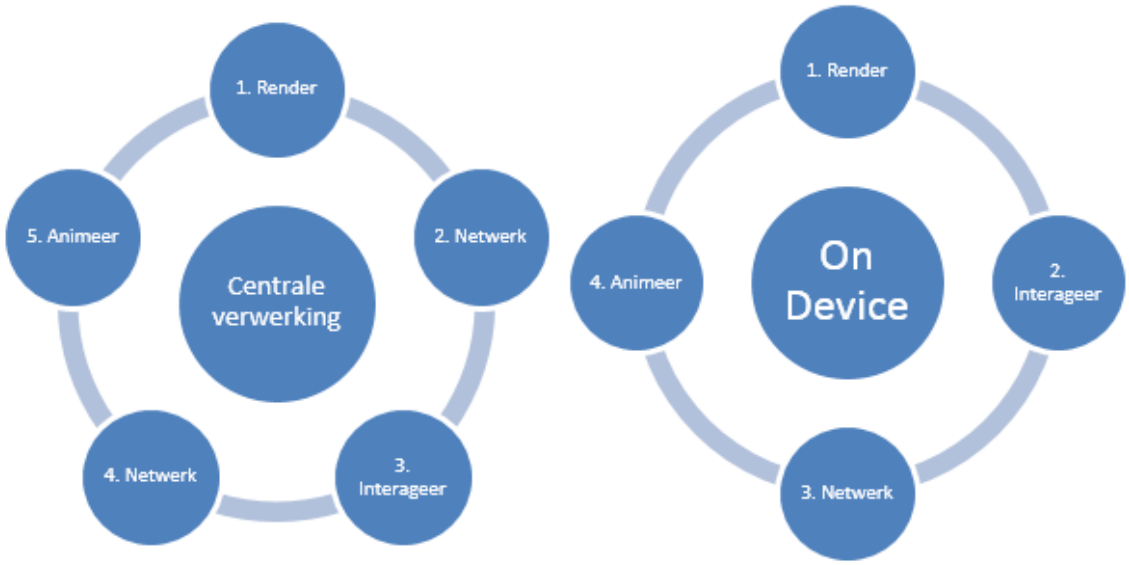
Formatting omvat het omzetten van het 3D model naar een representatie die zich leent tot eenvoudige implementaties van de volgende stappen in de pipeline. Zo wordt er gekeken naar de voordelen van “voxel data” of “polygon data” en hoe bepaalde structuren geoptimaliseerd kunnen worden. Hier is het wel belangrijk dat deze stap zowel voor als na “modelling” kan worden gedaan, afhankelijk van de modelleringtool. Indien er een eigen tool wordt ontwikkeld is het aangewezen eerst “formatting” te doen. In andere gevallen kan “formatting” best na “modelling” gebeuren.

Modelling laat gebruikers toe om de verkregen data te controleren en eventueel handmatig bij te werken. Dit zorgt ervoor dat gemiste elementen bij 3D registratie, of slechte oplossingen bij Surface Reconstruction of processing, kunnen worden weggewerkt. Het is belangrijk dat de modelling tool met de data overweg kan. Zoals reeds bij formatting wordt vermeld, kan het dus handig zijn om deze stappen om te wisselen, indien de compatibiliteit anders niet kan worden gegarandeerd.

2.2.2 Data Visualisatie

Om ervoor te zorgen dat de verkregen 3D modellen of 3D sequenties kunnen worden weergegeven en gebruikt via “Augmented Reality” is het belangrijk ervoor te zorgen dat er een proces wordt opgesteld dat de 3D modellen kan animeren, distribueren (in het geval van een spel voor meerdere personen) en weergeven. Figuur 6 geeft weer hoe dit visualisatieproces werkt. Belangrijk is op te merken dat dit proces repetitief is. Dit komt voort uit de continue rendering, die nodig is om ervoor te zorgen dat het weergegeven object zich op een constante plaats bevindt in de 3D omgeving.

In het voorbeeld wordt er een verschil gemaakt tussen twee soorten van verwerking. Er kan worden gekozen om een centrale server (of gewone desktop) de verwerking van alle data te laten doen en de resultaten te streamen naar de AR toestellen. Hierbij is het nodig om vanuit het toestel twee maal informatie over een netwerk te sturen. De eigen beweging- en interactiedata moet worden verstuurd naar deze centrale server. Eenmaal deze de data heeft



Figuur 6: Visualisatie pipeline met (links) een centrale verwerkingsserver en (rechts) on device verwerking

verwerkt en de rendering heeft gemaakt, moet de gegenereerde rendering worden verstuurd naar het AR device om te kunnen worden weergegeven.

Een andere methode, waarbij er meer processing kan worden gedaan op het AR toestel, zorgt ervoor dat er maar één maal data moet worden verstuurd over het netwerk per data-cyclus. Er moeten namelijk maar twee acties worden ondersteund. Enerzijds welke interactie er is gebeurd en anderzijds moet het mogelijk zijn om te vragen naar specifieke data afhankelijk van de gemaakte interactie. Elk toestel, dat deze data ontvangt, kan dan zelf zijn data verwerking toepassen. Zo is het mogelijk om subgroepen aan te maken die deze data ontvangt en dan zelf als een centrale server dient voor andere gebruikers. De verschillende stadia wordt uitgelegd in het geval van On Device verwerking, al zijn er maar kleine verschillen toch is het belangrijk op te merken dat vooral op het gebied van netwerking er verschillende noden zijn bij beide benaderingen.

Rendering beslaat het grootste gebied van dit proces. Dit gaat vooral over het positioneren van het AR-toestel in een 3D omgeving en over de juiste rendering maken voor het huidige standpunt van de gebruiker. Hierbij is de precisie van de positie zeer belangrijk. Indien de kleine veranderingen niet duidelijk worden opgenomen door het toestel, zal de weergave een onnatuurlijke representatie zijn. Daarnaast moet ook de kwaliteit van het object op een voldoende hoog niveau worden weergegeven, zodat het een realistisch model voorstelt.

Interactie zorgt voor de mogelijkheid voor gebruikers om te interageren met de getoonde objecten. Dit kan mogelijk worden gemaakt door gebruik te maken van 3D scanning bij de gebruiker zodat armen of andere objecten kunnen worden geplaatst in dezelfde 3D wereld als het object dat wordt gerenderd. Op die manier is de gebruiker in staat krachten uit te voeren op de getekende objecten. Hierbij kan er ook worden gekeken naar omgeving interactie. Zo is het nodig om 3D objecten bovenop de oppervlakte weer te geven. Anders is het mogelijk dat een deel van het object zich in de grond bevindt.

Netwerking is een topic dat meerdere gebieden bestrijkt. Het gaat niet enkel over de data transfers zelf maar ook over de compressie van de data die moet worden verstuurd. Hier moet dus samen met formatting worden gekeken naar efficiënte datastructuren die zich laten comprimeren of op een slimme manier over een netwerk zijn te sturen.

Animatie slaat op het aanpassen van modellen op runtime. Bij enkele interacties zal het model zich namelijk moeten aanpassen. Deze animatie kan gaan van het draaien van het object tot specifieke elementen van een object aan te passen. Deformatie of kinematics kunnen een voorbeeld hiervan zijn.

De volledige pipeline, die is besproken, geeft een algemeen overzicht voor het verwerken van een object tot een 3D model. In de volgende hoofdstukken worden deze processen besproken. Het eerste volgende hoofdstuk beslaat Data Generatie samen met al zijn deelprocessen. Hierbij ligt de focus voornamelijk op het genereren van verwerkbare data met een hoge kwaliteit.

3.Data Generatie Analyse

Het vorige hoofdstuk gaf een algemeen overzicht van alle componenten van het proces. Dit hoofdstuk zal dieper ingaan op alle componenten van het "Data Generatie" proces (zie Figuur 4). Zo zal voor elk onderdeel worden gekeken naar de huidige oplossingen en hoe deze zich met elkaar verhouden. In het volgende hoofdstuk zullen we dan de rest van de voorgestelde pipeline bespreken. In beide hoofdstukken zal er worden gekeken naar elementen die problemen vormen voor de nodige game-toepassingen.

3.1 3D Acquisition

Volgens (Bernardini & Rushmeier, 2002; Gross & Pfister, 2007) bevat een 3D Acquisition Pipeline op zijn minst een aantal van de volgende elementen:

- 3D Scanning
- Registration
- Merging
- View Planning
- Postprocessing

Volgens eerder vermelde bronnen zijn deze deelprocessen niet allemaal noodzakelijk om een oplossing te verkrijgen. Maar het ontbreken van sommige stappen zoals postprocessing kan mogelijk zorgen voor het verlies van kwaliteit. Vooraleer er dieper wordt ingegaan op deze verschillende topics is het belangrijk te weten wat puntenwolken zijn.

3.1.1 Puntenwolken

Een puntenwolk is een verzameling gepositioneerde punten. Dit kan zowel in een 2D als een 3D omgeving. Een puntenwolk wordt steeds gepositioneerd ten opzichte van een bepaald coördinatensysteem. Een punt in een 3D puntenwolk bevat zowel een positie als een normaal. De positie van een 3D punt bevat drie waarden voor de digitale posities in een cartesiaans assenstelsel. De normaal van een punt duidt de zogenaamde richting aan van een punt. De richting is een aanduiding voor het vlak waarin het punt en zijn buren liggen. Deze normaal kan, afhankelijk van het opnametoestel, zowel georiënteerd als niet-georiënteerd zijn. Voor een puntenwolk wordt er zo gesproken over respectievelijk een georiënteerde puntenwolk en een niet-georiënteerde puntenwolk.

Het verschil tussen beide wordt verklaard door (Berger et al., 2014). Berger stelt dat een georiënteerde puntenwolk bestaat uit consistente normalen die ofwel allen naar de binnenkant van het object wijzen ofwel naar de buitenkant van het object wijzen. Bij niet-georiënteerde puntenwolken is er geen consistentie bij de normalen en kunnen ze zowel naar binnen als naar buiten wijzen. Er zijn eenvoudige algoritmen die puntenwolken kunnen omvormen naar een georiënteerde puntenwolk. Een voorbeeld hiervan is het kijken naar de buren van een punt en aan de hand van de normalen van deze een nieuwe normaal bepalen.

Er zijn enkele problemen die worden voorkomen door het bepalen van de oriëntatie van punten. Dit kan gebeuren door middel van een algoritme, zonder dit direct op hardware niveau te doen. Er is namelijk geen zekerheid dat een bepaald punt een correcte normaal heeft en dus kan het zijn dat de beginstelling fout is. Indien er gebruik wordt gemaakt van complexere algoritmen die contouren gaan berekenen en hierop normalen definiëren kunnen deze slechte contouren bepalen dankzij ontbrekende data. Meer hierover in sectie 3.2.

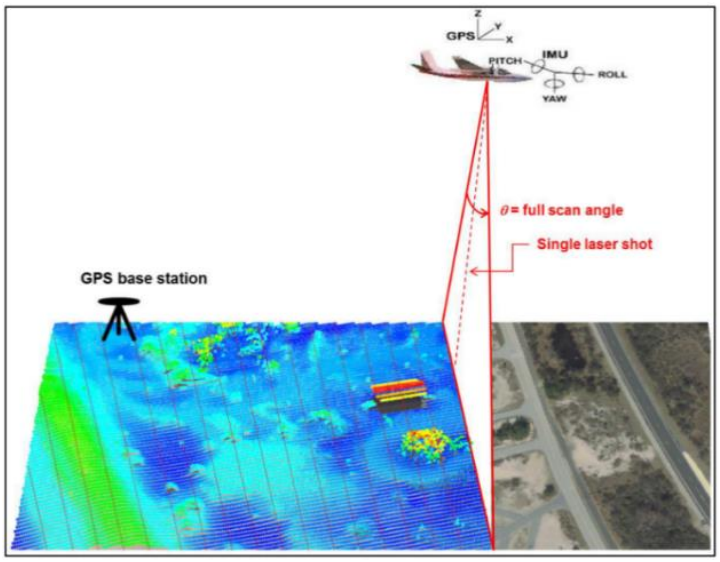
3.1.2 Scanning

Om het probleem omtrent 3D scanning aan te pakken zijn er meerdere mogelijke oplossingen. Een deel hiervan wordt besproken door (Pfeifle & Group, n.d.);

- Laserscanning
  - Time Of Flight (pulse based) (Kinect V2)
  - Time Of Flight (Airborne)
  - Phase-Base
  - Handheld/Close-range
- Stereoscopie
  - Single Target - Multi Pictures
  - Multi Target - Multi Pictures (Cloud Data)
- Structured Light Scanners
  - Static
  - Handheld (Kinect V1)
- Sonar
  - Single Beam
  - Multi-Beam
  - Robust Multi-Beam

Hieronder zal elke techniek worden uitgewerkt. Aan de hand van de technieken wordt een tabel opgesteld die een overzicht moet bieden. Dit overzicht geeft dan aan welke technieken gebruikt kunnen worden om het 3D scannen van objecten binnen deze pipeline.

Laserscanning (ook wel LIDAR genoemd; afkomstig van Light Detection And Ranging) gebruikt lasers om de posities van punten in een 3D omgeving te bepalen. Hierbij wordt er gebruik gemaakt van laserstralen die worden afgeschoten vanuit een bepaald punt. De reflecties van deze laser en de tijd dat deze laser nodig heeft om terug te geraken tot bij de opnameapparatuur, wordt gebruikt om de positie van het punt te bepalen. Laserscanners hebben geen licht nodig om 3D informatie te verzamelen en kunnen dus onder elke belichting worden gebruikt. Het LIDAR-systeem is net als het RADAR-systeem bedoeld om grote landschappen of gebieden in kaart te brengen. De werkingsmethode is dan ook vergelijkbaar. Waar er bij het RADAR-systeem gebruik wordt gemaakt van radiogolven wordt er bij een LIDAR-systeem gebruik gemaakt van lichtgolven. Deze ondervinden minder ruis bij weerkaatsingen, wat gewenst is om afstanden op te meten. De grote nadelen van deze techniek (bij landschap opname) zijn de blocking factoren zoals bijvoorbeeld wolken, regen of een dichte mist. Zeker omdat deze techniek wordt gebruikt bij het meten op lange afstanden kunnen deze factoren een grote rol spelen bij het opnemen van data. (Vallet & Skaloud, 2004).

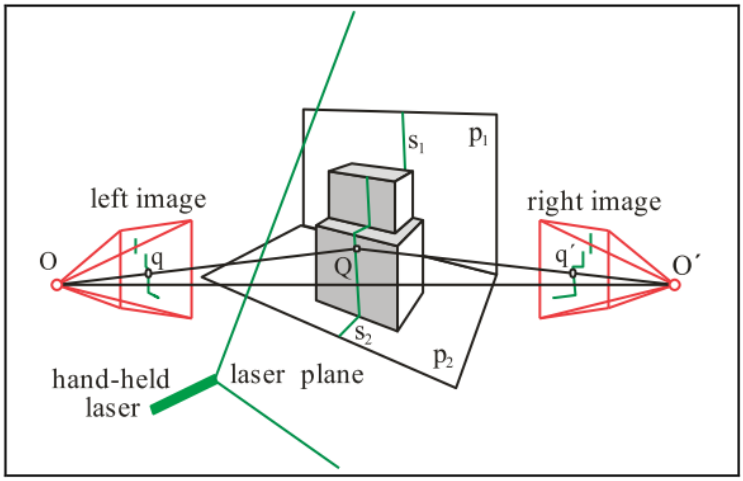


Figuur 7: Time Of Flight Systemen (Oceanic, 2012)

Time of Flight (ToF) systemen maken, zoals hun naam doet vermoeden, gebruik van de 'vliegduur' van een laserstraal om te bepalen wat de afstand is tot een object. Deze systemen zijn zeer handig om precieze metingen over een grote afstand te maken. Bij deze techniek worden er afzonderlijke laserstralen afgeschoten waarbij de turnaround time wordt bijgehouden van de verschillende stralen. De turnaround time is de tijdsduur die nodig is voor de lasterstralen om zich te verplaatsen vanaf de laser tot aan het object en terug tot aan het meetpunt. Binnen ToF-systemen zijn er 2 onderscheidingen, pulse-based en airborne. De pulse-based techniek gebruikt afzonderlijke puntstralen waarbij specifieke metingen kunnen worden gemaakt. De frequentie waarmee er stralen kunnen worden uitgestuurd ligt boven de 150 kHz (150,000 samples). Hierdoor kan snel een groot oppervlak in kaart worden gebracht, wat ervoor zorgt dat dit systeem zeer efficiënt is indien deze vanuit de lucht worden gebruikt. De airborne techniek werkt dus vergelijkbaar maar stuurt lijnen licht uit waarop er metingen kunnen worden gedaan. Deze toestellen worden dan onder een vliegtuig gehangen om zo snel een groot oppervlak te kunnen bestrijken.

Een andere laserscanning techniek is phase-based scanning. Hierbij worden er fase-gebaseerde laserstralen uitgestuurd. Op deze manier kan er een continue straal worden uitgestuurd. Door middel van het analyseren van de fase in de laserstraal kan dan worden bepaald naar welk punt de laser was afgeschoten. Deze techniek is veel sneller dan de vorige technieken. Hier is er namelijk geen pulserend mechanisme meer. Maar omdat er zoveel data wordt gegenereerd is het soms moeilijk deze data bij te houden en efficiënt op te slaan. De rest van het principe is vergelijkbaar. Omdat deze techniek echter wel meer fouten genereert over grote afstanden (meer als 1000m) wordt deze gebruikt voor relatief korte afstanden (ten opzicht van de vorige vermelde systemen). Er wordt gesproken over afstanden tussen de 50 en de 1000 meter, al zijn deze afstanden sterk afhankelijk van de gebruikte hardware.

De laatste methode maakt gebruik van "handheld" toestellen. Deze toestellen zijn ideaal voor de grootte van objecten die in deze masterproef worden besproken. "Handheld" toestellen meten namelijk op zeer korte afstand. Dankzij de eerder vermelde methode, namelijk ToF-gebaseerde scanning, is deze zeer accuraat omdat het licht amper afwijkingen vertoont op

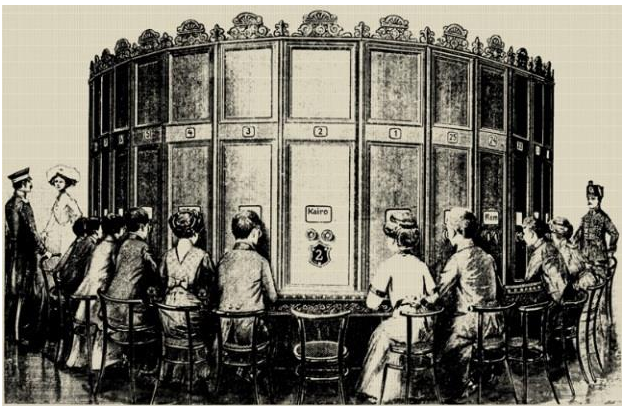


Figuur 8: Stereoscopie op basis van laserstralen. De lasterstralen worden gebruikt als referentie tussen de verschillende afbeeldingen. (Prokos et al., 2010)

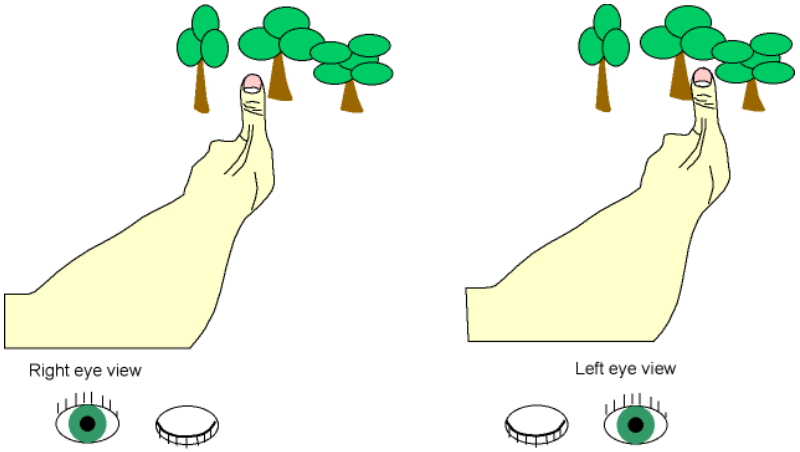
deze korte afstanden. Het nadeel is echter, en dit geldt trouwens voor alle technieken binnen laserscanning, dat frame per frame moet worden gescand omdat er geen volledig beeld wordt verwerkt per captatie. LIDAR-systemen nemen telkens maar een straal of punt op van een bepaalde scene. Dit is een groot nadeel ten opzichte van real-time scanners.

Voor het opnemen van hoge kwaliteit bewegende frames zijn laserscanners eigenlijk niet de beste optie. Het grootste nadeel aan deze techniek is dat het voor het opnemen van volledige frames er een lager aantal punten kan worden opgenomen met een lagere kwaliteit. Omdat LIDAR-apparatuur nooit een volledig beeld gebruikt voor het opmaken van een diepte scan kunnen bewegend objecten dus moeilijk in te scannen zijn. Het toepassingsdomein voor dit onderzoek beslaat het inscannen van bewegende 3D objecten/beelden. Daarnaast is de kostprijs voor deze toestellen zeer hoog (voor niet handheld apparatuur is er al snel sprake van 70.000$ of meer (Hardware, n.d.)) wat een gigantische investering vraagt. Tot slot is er nog het probleem van de afstanden waarop deze toestellen effectief zijn. Buiten de handheld toestellen dienen deze voornamelijk voor super lange afstandsmetingen.

Er is echter een grote stap vooruit gezet met de intrede van de Kinect 2 van Microsoft. Dit toestel gebruikt een ToF techniek om de diepte van punten te bepalen. Ook al bevat de Kinect 2 geen super accurate sensor ten opzichte van de eerder besproken ToF-gebaseerde



Figuur 9: Kaiserpanorama (1880) gebruikt stereo-beelden voor het weergeven van beelden. (http://en.wikipedia.org/wiki/Stereoscopy)



Parallax effect illustrated with your thumb. Notice how your thumb held at arm's length appears to shift with respect to background objects when you look at it with one eye and then the other eye.

Figuur 10: Het parallax effect bij twee verschillende standpunten. (http://www.astronomynotes.com/)

hardware, toch geeft dit enkele grote voordelen. De Kinect 2 is namelijk in staat om een volledige frame in te scannen op een bepaald moment.

Stereoscopie is een oud concept waarbij er gebruik wordt gemaakt van twee afbeeldingen om diepte perceptie te creëren (Prokos, Karras, & Petsa, 2010). De eerste grote toepassingen van deze techniek worden terug gevonden in de 19de eeuw, in het bijzonder Kaiserpanorama (zie Figuur 9). Stereoscopie is afgeleid van de manier waarop onze ogen werken. Elk beeld heeft een eigen kijkhoek op een bepaald object. Door de twee beelden te vergelijken kan men komen tot een 3D representatie van een wereld. Deze techniek kan dus ook worden gebruikt om 3D punten af te leiden van afbeeldingen. Hierbij wordt er enkel gebruik gemaakt van meerdere afbeeldingen om de dieptebepalingen te doen er komen dus geen andere technieken (zoals lichtpulsen) bij kijken om een 3D punt af te leiden uit een beeld.

Stereoscopie is daarom dus sterk afhankelijk van het algoritme dat wordt gebruikt om deze berekening te doen. Al zijn er veel verschillende algoritmen, de basisprincipes zijn voor al deze algoritmen hetzelfde. Het eerste wat er wordt gedaan, zijn referentie punten zoeken tussen de afbeeldingen. Dit is het moeilijkste element van de pipeline aangezien er zeer veel problemen kunnen opduiken. Zo kunnen sommige gebieden vanuit één kijkpunt wel zichtbaar zijn en vanuit het andere niet. Reflecties en kleuren kunnen veranderen vanuit verschillende perspectieven. Een object kan er heel anders uitzien vanuit een andere hoek. Het feit dat punten moeilijk te herkennen zijn tussen twee verschillenden standpunten zorgt ervoor dat het moeilijk is deze techniek te gebruiken, daarom opteren de meeste algoritmen ervoor om eerst referentiegebieden te zoeken binnen de afbeeldingen. Voor het bepalen van de diepte van een punt wordt er gebruik gemaakt van het effect van de parallax. De parallax is een verschijnsel waarbij voor een waarnemer objecten minder fel bewegen naarmate ze verder weg liggen van een bepaald standpunt. Bij dieptebepalingen wordt dan gekeken naar het verschil tussen de afstanden tussen de twee beelden ten opzichte van het referentie punt. Een voorbeeld van het effect van de parallax wordt gegeven in Figuur 10.

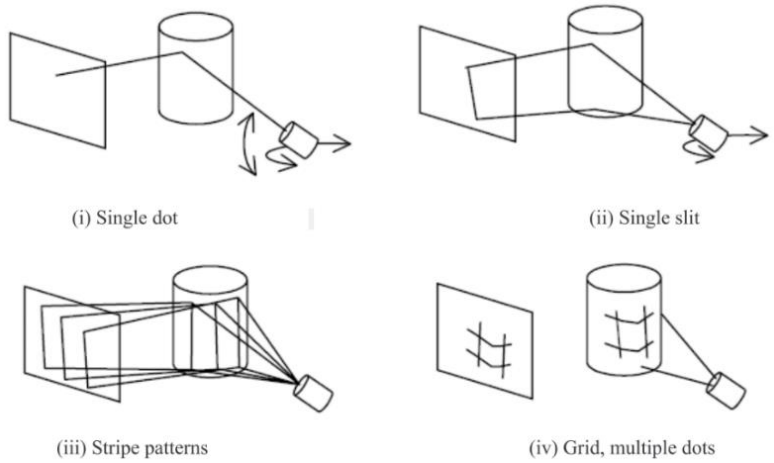
Het gebruik van het parallax effect bij diepte bepaling volgt enkele kleine stappen. Eenmaal er referentie punten zijn bepaald voor een bepaalde scene, wordt de relatie tussen deze punten

bekeken vanuit de verschillende kijkpunten. Dit kan bijvoorbeeld worden gedaan door een lijn te trekken door verschillende referentiepunten op één afbeelding en deze lijn proberen te reconstrueren op de andere afbeelding. De relatie tussen deze lijnen kan dan dienen om door middel van triangulatie de diepte van punten te bepalen. Hoewel deze techniek theoretisch zeer krachtig is, blijft het voor algoritmen zeer moeilijk om referentiepunten te vinden.

Wat precisie betreft is deze techniek volledig afhankelijk van de afbeeldingen die worden gegeven aan het algoritme. Hoe groter de resolutie van de afbeeldingen, hoe preciezer het algoritme berekeningen kan maken. Het is namelijk zeer moeilijk om van meerdere afbeeldingen een gedetailleerdere 3D puntenwolk op te bouwen. Dit is een gevolg van de referentiepunten die worden gebruikt bij het berekenen van de diepte.

Deze techniek heeft enkele interessante voordelen voor de voorgestelde situatie zoals het feit dat het enkel twee beelden nodig heeft en het feit dat er geen dure materialen nodig zijn. Het grootste voordeel is het eerst genoemde, namelijk dat er slechts twee beelden nodig zijn voor het vinden van dieptepunten. Hiervoor is het natuurlijk wel nodig dat deze beelden op hetzelfde moment worden genomen, dit kan door middel van specifieke hardware zoals stereoscopische camera's. Het ander voordeel dat werd gegeven, is dat deze zeer goedkoop kan zijn, indien er wordt geopteerd voor 2 individuele camera's daalt de prijs maar is het wel belangrijk de beelden goed op te lijnen. Er is echter ook een nadeel aan deze techniek, de CPU kracht die nodig is om het algoritme te voltooien is groot. Zeker aangezien er voor een goede precisie gebruik moet worden gemaakt van hoge resolutie camera's. Hoe hoger de resolutie van de beelden, hoe langer de verwerking duurt. Er zijn echter wel sprongen vooruit gemaakt waarbij stereoscopie haalbaar wordt met een korte verwerkingstijd (Mattoccia, Arces, Viti, & Ries, n.d.).

Structural Light Scanners werken op een gelijkaardig manier als de stereoscopische aanpak. Bij deze aanpak wordt echter één van de camera's vervangen door een lichtbron (Fofi, Liwa, & Voisin, n.d.; Pfeifle & Group, n.d.). Omdat er licht interferentie kan zijn, afhankelijk van de situatie waarin wordt gefilmd, is het belangrijk om een lichtbron te kiezen die zo weinig mogelijk (of geen) interferentie heeft met andere lichtbronnen. De werking van de techniek verandert echter niet of minimaal indien er verschillende lichtbronnen worden gebruikt. Om



Figuur 11: Basis werking van structurele licht scanners. Referentie punten worden geprojecteerd op een object zodat een camera deze kan herkennen. (Fofi et al., n.d.)

ervoor te zorgen dat de objecten niet verkleuren tijdens opname kan er worden geopteerd voor lichtbronnen die lichtstralen uitsturen buiten het visuele spectrum van de mens.

De techniek werkt door het projecteren van een patroon op het object. Een andere camera kan dan de reflectie van deze lichtstralen waarnemen en aan de hand daarvan 3D punten bepalen. Hierbij is het belangrijk op te merken dat deze techniek stereoscopie toepast waarbij er geen referentiepunten moeten worden bepaald aangezien deze zelf worden uitgestuurd door de hardware-setup. De patronen die worden gebruikt, kunnen verschillende eigenschappen hebben. Er kan gebruik worden gemaakt van punt of lijn projectie waarbij er een dimensie moet worden gescand. Daarnaast is het ook mogelijk om een volledig patroon te projecteren waarbij een volledige scan kan worden genomen van het beeld. Hierbij is het wel belangrijk dat elk punt dat wordt geprojecteerd, kan worden onderscheiden van alle andere punten in het vlak. Dit kan door een uniek patroon te maken waarbij geen enkel gebied hetzelfde is.

Zoals eerder vermeld zijn er meerdere mogelijkheden voor de lichtprojectie. Er worden 3 mogelijke lichtprojecties vermeld namelijk: infrared (IRSL), imperceptible(ISL) en filtered(FSL). Eerder werd vermeld dat de techniek tussen deze 3 methodes niet veel verschilt. Er zijn echter wel verschillen tussen de methodes. Een verschillende lichtprojectie kan wel meerdere opties met zich meebrengen. Zo kunnen ISL en FSL gecodeerde patronen gebruiken, dit zorgt voor een eenvoudigere manier om het patroon te herkennen.

(Fofi et al., n.d.) toont aan de hand van de Tabel 1 aan wat de verschillende voor en nadelen zijn van de technieken. Hierbij is het belangrijk op te merken dat voor onze doeleinden de IRSL hardware de beste methode zal zijn, aangezien deze een hoge precisie heeft en de hoogste resolutie mogelijkheden heeft. Hierbij is het belangrijk te vermelden dat de Kinect van Microsoft een IRSL toestel is. Er zijn echter ook optimalisatietechnieken mogelijk op het basis concept van structureel licht, waardoor er verbeteringen kunnen worden gemaakt(Nayar &

SENSOR		
IRSL	• Highest resolution • High accuracy • Easy implementation	• Mechanical scanning • No codification
ISL	• Codification capabilities • Modularity and adaptivity • Colour image + SL image • One-shot reconstruction	• Occupancy • Synchronization between projection and capture
FSL	• High resolution • Coded pattern can be designed • Multi-band pass sensor	• A filter is required • Not as good as IRSL when composed by laser source, not as good as ISL when composed by video-projector

Tabel 1: Vergelijking van verschillende lichtprojecties.(Fofi et al., n.d.)

Gupta, 2012). Op die manier wordt er hardware matig een beter model gevormd waarmee kan worden gewerkt.

SONAR (afkomstig van Sound Navigation And Ranging) waarbij geluidsgolven worden gebruikt (op een gelijkaardige manier als laserscanning). De laserstralen worden in deze methode echter niet gebruikt, in de plaats hiervan wordt er gebruik gemaakt van geluidsgolven. Deze golven ondervinden echter zeer veel interferentie indien ze worden gebruikt in gesloten gebieden/ruimten door middel van het echo-effect.

Dat is ook direct de reden dat er wordt geopteerd voor LIDAR bij bovengrondse 3D imaging en Sonar bij het in kaart brengen van gebieden onder water. Onder water heeft de omgeving namelijk een veel hogere massadichtheid dan boven water waardoor de geluidsgolven zich preciezer kunnen voortplanten. Hierdoor wordt de kans op interferenties dus ook verkleind. Voor bovengrondse toepassingen kunnen sonarsystemen ook worden gebruikt, al blijft een LIDAR-oplossing hier veel preciezer en betrouwbaarder.

Voor de toepassing, die wordt beschreven in deze thesis, is er echter geen mogelijkheid om van de sonartechnologie gebruik te maken. Daarom zal deze niet worden opgenomen in de vergelijkende lijst verder in dit document.

Het overzicht dat volgt vergelijkt de hierboven vermelde scanning apparatuur. Alle details in de volgende tabel zijn veranderlijk vanwege de verandering en de verbetering van de technologie. Bij het bestuderen van deze tabel is het belangrijk op te merken dat de gebruikte waarden bestaan uit benaderingen en gemiddelden van meerdere producten. Dit komt doordat elk product andere doelen heeft en andere specificaties. De prijzen zijn gebaseerd op gevonden producten waarvan de prijs publiekelijk beschikbaar was. Voor de meeste laserscanners zijn deze prijzen enkel aan te vragen.

Voor een toepassing in de game-industrie zijn extreem dichte puntenwolken niet belangrijk. En de afstand waarop objecten worden gescand zal eerder klein zijn, dus moet er worden gekeken naar structureel light of stereoscopie oplossingen. Hierbij is het vooral kijken naar de huidige technologie en de prijs die er tegenover staat. Daarbij komt de Kinect van Microsoft direct naar voren. Deze goedkope hardware is in staat zeer goede 3D scans te maken, in real time.

Scantype	Methode	Prijs (€)	Afstand (m)	Nauwkeurigheid	Speed (punten/s)
Laser	Time Of Flight Pulse	> 2.000	50 – 1000	> 1 cm	< 150.000
	Time Of Flight Airborne	> 5.000	100 – 5000	> 3 cm	< 150.000
	Phase-Based	> 2.000	50 – 1000	> 1cm	> 150.000
	Handheld	> 300	0,3 – 100	> 0,1 cm	< 80.000
Stereoscopy	-	> 100	0,5 – 100	Pixelgrootte	Framerate
Structured Light	-	> 100	0,5 – 100	Pixelgrootte	Framerate

*Tabel 2: De verschillende scantechnieken met elkaar vergeleken. Speed is hierbij het aantal punten dat kan worden gegenereerd per seconde. Pixelgrootte duidt erop dat de nauwkeurigheid afhankelijk is van werkelijke grootte die wordt vertegenwoordigt door een bepaalde pixel (dit is dus niet consistent over het volledige frame). Framerate duidt aan dat het afhankelijk is van de behaalde framerate en de resolutie die wordt gebruikt. Met andere woorden het aantal pixels * het aantal frames per seconde.*

3.1.3 Post Scan Handling

De vorige sectie ging dieper in op de captatie methodes die er bestaan. Deze werden gerelateerd aan hardware, en wat de invloed van deze hardware is op scansnelheden en accuraatheid. De rest van het proces voor het inlezen van data via een device is algemeen beschreven omdat de meeste elementen zich ook in hardware (black-box) afspeelt. Daarom zullen deze stappen enkel kort worden overlopen om het overzicht toch volledig te houden.

De eerste stap wordt *Registration* of registratie genoemd. Registratie is de eerste stap die wordt uitgevoerd na het verkrijgen van de 3D-punten. Tijdens deze stap wordt er gezorgd dat meerdere scans in 3D worden gealigneerd. Hierbij komen echter enkele belangrijke elementen bij kijken. Het belangrijkste aspect bij het samenvoegen van meerdere beelden is het aligneren van de beelden in tijd of ruimte. Dit wil zeggen dat enkel beelden die op hetzelfde moment zijn genomen, of beelden die dezelfde scene beslaan, met elkaar worden samengevoegd. Indien er meerdere apparaten worden gebruikt of een bepaald apparaat dat beweegt over tijd, is het belangrijk om te zorgen dat de verschillen in opname posities worden berekend en verwerkt. Bij hardware waar meerdere sensoren in zitten, kan dit op voorhand worden bepaald door de ontwikkelaar, waardoor een gebruiker geen alignatie moet doen. Bij het combineren van meerdere toestellen moet er hier echter wel aandacht aan worden geschonken. Een eerste techniek is het volgen van de positie en rotatie van de meetapparaten door een bepaalde vorm van tracking. Deze posities en rotaties kunnen dan worden gebruikt om meerdere beelden te synchroniseren. In sectie 3.3.1 wordt er dieper ingegaan op gebruikte technieken om verschillende beelden te aligneren.

Indien er opnames worden gemaakt doorheen de tijd, of indien een bepaald scanapparaat gebruik maakt met meerdere sensoren, is het mogelijk dat er veel punten meerdere keren voorkomen. Het samenvoegen van deze punten, of volledige puntenwolken, noemt men *merging*. Dit kan door slechts van 1 scan de data bij te houden indien dit over hetzelfde object gaat. Ook kan er gebruik worden gemaakt van meerdere datasets om zo artefacten van de scanner weg te werken (volgens Bernardini wordt dit opgesplitst in line-of-sight error compensation, gevolgd voor het mergen naar een single mesh). De moeilijkheid bij deze techniek bestaat er uit om zo weinig mogelijk data te verliezen tijdens het samenvoegen van verschillenden punten of puntenwolken. Zo kunnen punten, die zeer dicht op elkaar liggen, vaak worden samengevoegd zonder optisch een verschil te geven in weergave. Voor andere toepassingen zijn deze kleine verschillen echter wel cruciaal. Daarom moet er bij merging altijd worden gekeken naar de manier waarop punten worden samengevoegd. Een gebruikelijke techniek gebruikt een vooraf bepaalde maximale densiteit om er zo voor te zorgen dat er geen detail verloren gaat bij het weglaten van punten.

View Planning bestaat uit het bepalen van de volgende positie van de scanner. Indien we er vanuit gaan dat een gebruiker met 3D scanner of dergelijk toestel, rond een object beweegt, is dit een niet geautomatiseerd proces. Hierbij wordt Registration gebruikt om deze “View Planning” alsnog te kunnen bepalen. Indien er een automatische route wordt gegenereerd hoeft registratie dus geen plaatsbepaling meer te doen (al kan dit wel ter controle worden gebruikt). View Planning zorgt dus voor het aanpassen van de coördinatensystemen aan de hand van gemaakte of ingestelde bewegingen. Dit zorgt ervoor dat in de volgende iteratie de

registration en merging stap eenvoudiger zijn. De meeste berekeningen zijn namelijk gemaakt tijdens het positioneren in de View Planning stap.

De laatste, optionele, stap zorgt voor een verwerking van de verkregen data, dit wordt ook *post processing* genoemd. Hierbij kunnen extra controles of situationele algoritmen worden gebruikt. Zoals het vervolledigen van modellen indien de scanner enkele gaten creëerde of het verhogen/verlagen van het aantal punten in de puntenwolk door middel van cube-division. Post-Processing bevat daarnaast alle technieken, die worden toegepast nadat de data is verkregen. In de rest van dit hoofdstuk worden enkele van deze technieken toegelicht.

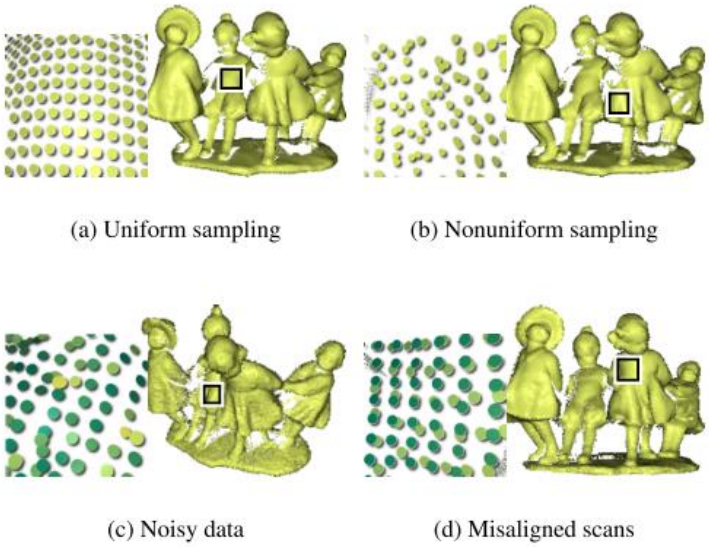
3.1.4 Discussie

De meeste elementen uit deze pipeline zijn opgeloste problemen, waarvoor technieken staan beschreven. Deze zijn onafhankelijk van de captatie methode, die wordt gebruikt bij het verkrijgen van de data. Daarnaast is het duidelijk dat er, afhankelijk van de toepassing, een optimale setup is voor het capteren van data. Dit wil zeggen dat één bepaald type van hardware beter is voor het capteren van een bepaalde scene/omgeving, terwijl andere hardware of technieken een optimale oplossing bieden voor een andere scene. In het geval van deze thesistekst, waarin wordt gekeken naar captatie-technieken voor gaming, wordt er best gekozen voor een techniek die een volledige frame kan opnemen per registratie event. Dit is nodig aangezien de scènes, die worden gecapteerd dynamisch van aard zullen zijn. Daarom moet er worden gekeken naar oplossingen binnen Structured Light scanners of moet er gebruik worden gemaakt van stereoscopie. Al is het belangrijk de ontwikkeling rond de Kinect V2 niet uit te sluiten, aangezien deze ook een volledig frame per registratie event kan afleveren. In de volgende hoofdstukken zullen we zien dat de Kinect V2 de beste oplossing is, aangezien deze ook andere technieken ondersteund, die vitaal zijn voor de captatie van karakters.

3.2 Surface Reconstruction

Surface Reconstruction (SR) is het gedeelte van de pipeline dat zorgt voor het oplossen van het probleem in verband met de discrete punten, die worden gebruikt bij het inlezen van objecten. Bij het gebruik van punten-wolken is het belangrijk op te merken dat niet alle punten worden ingelezen, of dat er 'noise' is op de puntenwolk. Om deze elementen weg te werken, wordt er gebruik gemaakt van enkele algoritmen. Elk algoritme gaat daarbij anders om met de verkregen puntenwolken. Een belangrijk aspect van de algoritmen is het uiteindelijke formaat van de data. Bepaalde algoritmen dienen uitsluitend voor het aanmaken van polygonen aan de hand van de eerder verworven datasets. Andere algoritmen dienen eerder voor het oplossen van noise en missing data. Het is hierbij belangrijk op te merken dat de datasets die worden gegenereerd door middel van captatie zeer veel noise hebben. Daarnaast is ook missing data een groot probleem bij frontale opnames. Het verwerken van deze verschillende problemen brengt echter een hogere complexiteit met zich mee. Daarom wordt er ook gekeken naar de performantie van de bekeken algoritmen.

Er zijn in de introductie tot SR drie belangrijke parameters vermeld namelijk: noise, missing data en performantie. Het probleem bij de vergelijkingen van algoritmen is dat de performantie daalt naarmate er meer rekening moet worden gehouden met missing data en noise. Om toch een vergelijking te kunnen maken, is het belangrijk op te merken dat performantie niet het belangrijkste aspect is voor deze pipeline. Indien er wordt gesproken over missing data, dan gaat het over puntenwolken die onvolledig zijn en dus 'gaten' vertonen



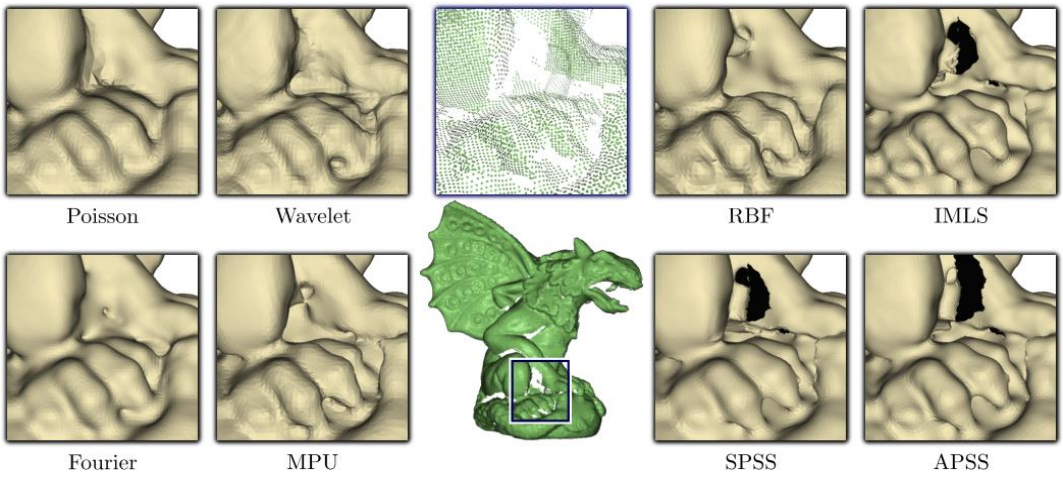
Figuur 12: Voorbeelden van fouten die Surface Reconstruction probeert weg te werken. (Berger et al., 2013)

in een oppervlakte. Daarnaast beschrijft noise het aspect van punten die afwijken van de werkelijke positie. Hoe verder de 3D punten gemiddeld van hun werkelijk punt liggen, hoe hoger de noise factor is. Beide worden gegenereerd door de captatie stap. Missing data valt niet weg te denken, aangezien niet alle punten van het object altijd zichtbaar zijn. Een frontaal aanzicht toont namelijk maar een bepaalde zijde van een object, alle andere zijden zijn dan niet zichtbaar voor de scanner. Daarnaast is een scanner niet 100% accuraat. Dit zorgt inherent voor de noise, die terug is te vinden op data. Omdat gebruikte klokken en sensoren altijd een kleine afwijking vertonen, die niet eenvoudig is te achterhalen, zullen er altijd fouten zitten in de opgenomen dataset.

In (Berger, Levine, Nonato, Taubin, & Silva, 2013) wordt er een vergelijking gemaakt tussen verschillende algoritmen. Bij het bespreken van de algoritmen wordt er vanuit gegaan dat er een georiënteerde puntenwolk wordt gebruikt. Er zijn echter ook 3D scanning methodes die enkel een niet georiënteerde puntenwolk kan registreren. Hierbij moet dan gebruik worden gemaakt van normaalalgoritmen, die zoeken naar normalen op de puntenwolk en op die manier toch georiënteerde puntenwolken kunnen genereren. Dit geeft wel een impact op de performantie van het algoritme. Naast de reeds besproken algoritmen in de thesis, geven we ook aandacht aan de vergelijking die is gemaakt op het gebied van verschillende algoritmeklassen en hoe deze ten opzichte van elkaar presteren (Berger et al., 2014). Tot slot is het ook belangrijk te kijken hoe real-time algoritmen zich kunnen meten met deze algoritmen. Specifiek zal er worden gekeken naar KinectFusion (Izadi et al., n.d.), een door Microsoft ontwikkelde oplossing voor real-time SR.

Om het beste algoritme te vinden, zullen we eerst enkele algoritmen in het kort verklaren zodat de werkwijze van de algoritme gekend is. Na de korte bespreking bekijken we de werking onder enkele omstandigheden zoals veel noise en missing data. Tot slot volgt er een tabel om de algoritmen met elkaar te kunnen vergelijken. Er wordt vooral gekeken naar het onderzoek van Berger (Berger et al., 2013)(Berger et al., 2014) om een totaal overzicht te bekomen.

De meeste van de voorgestelde algoritmen behoorden tot de klasse van “Indicator Function”’s (IF). Hierbij wordt er gezocht naar een functie die voor elk punt x weet of deze ligt op het object. De algoritmen in deze klasse proberen dus deze functie op te lossen op verschillende manieren.



Figuur 13: Verschillende resultaten van SR-algoritmen met centraal de input data. (Berger et al., 2013)

3.2.1 Algoritmen

Poisson is één van de functies die behoort tot de IF-klasse. Er wordt eerste een IF berekend van de verkregen data. Hierbij wordt gebruik gemaakt van georiënteerde normalen. Hoe deze berekening precies te werk gaat, kan terug worden gevonden in (Kazhdan, Bolitho, & Hoppe, 2006). Deze IF geeft als resultaat een integraal, die kan worden berekend als een Poisson probleem. Dit probleem wordt enkel opgelost nabij de verkregen punten. Om dit te kunnen doen, wordt er gebruik gemaakt van een octree (zie sectie 3.4.2 voor meer informatie). Deze methode zorgt echter wel voor een lagere resolutie van de octree, indien er data ontbreekt, al blijft de techniek toch zeer precies (zoals blijkt uit de benchmarking van Berger). Ondanks de lagere resolutie, vindt Poisson echter wel een oplossing om lege data-vlekken op te vullen. Zoals ook zichtbaar in de bovenstaande figuur.

Fourier-gebaseerde technieken komen zeer goed overeen met Poisson SR. Omdat Fourier Transformatie zelf een zeer trage methode is, wordt er vaak gebruik gemaakt van geoptimaliseerde methoden zoals Fast Fourier Transform of Adaptive Fourier Transform (Schall, Belyaev, & Seidel, 2006). De laatste van de twee gebruikt een techniek waarbij de puntenwolk wordt opgedeeld. Deze techniek zorgt ervoor dat details beter bewaard blijven en de geheugenbelasting drastisch wordt verlaagd. Details blijven beter bewaard omdat er geen algemene functie wordt bepaald. Algemene functies zijn handig voor het oplossen van gaten in de data. Maar omdat een functie het hele object beslaat, worden details vervaagd of afgeplat.

Omdat Fourier een som nodig heeft over alle samples binnen de data, is deze techniek redelijk gelimiteerd. Er is namelijk maar een eindig aantal geheugen aanwezig waardoor het verwerken van miljoenen punten soms tot problemen kan leiden. De bovenvermelde methode (Adaptive Fourier Transform) biedt hier wel een oplossing aan maar is nog niet optimaal. Een aanpak door middel van *wavelets* probeert hier nog een verbetering op aan te brengen (Manson, Petrova, & Schaefer, 2008). Wavelets werken in principe net zoals een Fourier (deze kan worden gezien als een speciaal geval van wavelets). Het voordeel bij wavelets is dat er op voorhand enkele functies worden gedefinieerd waarmee het object kan worden voorgesteld. Dit zorgt ervoor dat er kan worden bepaald of men voor performantie kiest of voor kwaliteit. Het is namelijk zo dat een "smooth"-wavelet zorgt voor een hogere computationele kost, maar een beter resultaat. Indien er echter nood is aan een eerder real-time oplossing, kunnen de basisfuncties worden aangepast, zodat er een lagere kost nodig is om deze te overlopen. Net zoals Fourier en Poisson is wavelet een IF-gebaseerd algoritme, maar geeft dit algoritme ten opzichte van Fourier en Poisson een minder accuraat resultaat.

Dit algoritme is in tegenstelling tot de vorige algoritmen vertrokken van een niet globale aanpak. Er bestaan echter wel lokale varianten van enkele van de vorige algoritmen (zoals bij wavelets is besproken). (Ohtake, Belyaev, Alexa, Turk, & Seidel, 2003) legt de werking uit van *Multi-level Partition of Unity*(MPU). Het belangrijkste element van het algoritme is de Partition Of Unity, een methode waarbij lokale oplossingen worden gebruikt om een globale oplossing te vinden. Op deze manier wordt er bespaard op geheugengebruik en kan er meer in detail worden gewerkt met de data. Om deze verdeling te doen, wordt er (net zoals bij wavelets) gebruik gemaakt van octrees. Deze opdeling laat namelijk toe een 3D ruimte eenvoudig op te delen in verschillende delen. Per deel kan er dan een schatting worden gemaakt naar het model dat zich hierin bevindt. Dit komt overeen met de IF van de vorige methoden. Daarna wordt ervoor gezorgd dat alles wordt aaneengeschakeld. Dit wordt ook wel "smoothening" van het oppervlak genoemd. Tot slot wordt er nog een precieze functie toegepast, die zorgt voor de scherpere elementen binnen het domein. Daarna worden alle domeinen samen gevoegd. Op deze manier wordt een globale reconstructie gevormd.

Radial Basis Functions (RBF) hergebruikt het grootste deel van de techniek die wordt gebruikt bij MPU. Zo wordt er ook gewerkt door middel van Partition of Unity. Maar de functies voor het bepalen van de modellen worden veranderd door Radiale Basisfuncties. Indien er weinig noise op de data zit, kunnen deze functies veel sneller en accurater een oplossing bieden als het eerder verklaarde MPU. Maar indien er noise aanwezig is, heeft RBF het moeilijk om hier omheen te werken. Dit komt voort uit het feit dat elk punt meetelt en dus noise het model gaat verstoren. De noise blijft zo dus achter in het uiteindelijke resultaat van de SR.

Moving least Squares (Levin, 1998) is een aanpassing op Least Squares om benaderingen op te lossen. Bij deze aanpassing wordt een punt bewogen over alle punten en is er een sterker gewicht gebonden aan punten, die dicht bij het gekozen punten liggen. Op deze manier worden scherpe randen en andere bruuske veranderingen in het oppervlak deels bewaard. In andere gevallen worden deze punten bijna volledig afgevlakt waardoor het object onherkenbaar kan worden.

Least Squares (Nealen, 2003) zelf is een benaderingsmodel, waarbij het de bedoeling is dat de foutenmarge van de oplossingen worden beperkt. Meer in detail moet de som van de kwadraten van de foutenmarges zo klein mogelijk worden gehouden. Op die manier wordt ervoor gezorgd dat de verkregen oplossing zo dicht mogelijk bij het model aanleunt.

Een volledig ander oplossingsmodel is *Kinect Fusion*. KF is een SDK ontwikkeld door Microsoft ter ondersteuning van hun 3D scanning hardware (Kinect). In tegenstelling tot alle voorgaande algoritmen is KF ontwikkeld om een real-time oplossing te bieden voor Surface Reconstruction. Het is hierbij belangrijk dat de software specifiek ontwikkeld is voor het gebruik van de Kinect en zijn real-time toepassing. Er is reeds veel gekend over Kinect Fusion (Izadi et al., n.d.) en de methode van werken. De huidige SDK is erin geslaagd om een Surface Reconstruction rate te hebben van 2Hz (2 beelden per seconde) wat een groot verschil is met de meerdere seconden die nodig zijn om één model te maken in de eerder vermelden algoritmen.

Het grootste nadeel aan dit reconstructiemodel is dat er geen tijd is om ontbrekende data op te vangen en te verwerken. Hierdoor krijgen bewegende objecten een slechte kwaliteit na de verwerking van de data; er wordt namelijk eenmaal per 500MS een data sample genomen dat kan worden gebruikt. Een oplossing hiervoor zou kunnen zijn dat alle samples van de vorige 500MS worden samengevoegd, al wordt het data-volume dan te groot voor deze real-time verwerking. Om samen te vatten kan er worden gezegd dat de Kinect is ontwikkeld om real-time toepassingen te ondersteunen en hiervoor aan kwaliteit moet inleveren.

3.2.2 Discussie

In de onderstaande tabel proberen we de algoritmen te vergelijken met elkaar. Het is natuurlijk moeilijk deze te vergelijken aangezien volgens (Berger et al., 2013) de prestaties van de algoritmen verschillen van object tot object. Objecten met meer textuur kunnen voor bepaalde algoritmen beter zijn dan objecten met vlakkere oppervlakken en omgekeerd. We gebruiken als vergelijkende data dus de analyse die door Berger werd gemaakt in zijn beide algoritmen. Het is belangrijk te weten dat Implicit in Tabel 3 slaat op implicit surfaces. Dit zijn oppervlakten die worden bepaald aan de hand van een wiskundige vergelijking. Deze zijn eenvoudig om te vormen naar zowel puntenwolken of naar vertex data.

Algoritme	Noise	Missing Data	Perf.*	Point-cloud	Output
Poisson	Robuust	Zeer Robuust	36.83	Georiënteerd	Implicit
Fourier	Robuust	Zeer Robuust	28.70	Georiënteerd	Implicit
Wavelet	Robuust	Robuust	2.13	Georiënteerd	Implicit
MPU	Robuust	Robuust	12.83	Georiënteerd	Implicit
RBF	Robuust	Niet Robuust	34.78	Georiënteerd	Implicit
MLS	Niet Robuust	Robuust	34.11	Niet georiënteerd	Point data
KinectFusion	Niet Robuust	Niet Robuust	0.5	Georiënteerd	Vertex

Tabel 3: Verschil tussen de verschillende SR-algoritmen. (*) performantie wordt aangetoond op basis van de gegevens in Tabel 2 op pagina 19 van (Berger et al., 2013), lager is beter.

De elementen die worden vergeleken in de tabel zijn allen van belang voor het gebruik in een AR toepassing. (1) Het eerste wat we bekijken is noise, dit toont aan hoe goed het algoritme

overweg kan met ruis op de input puntenwolk. Hoe robuuster het algoritme hoe beter, er kan dan namelijk gebruik worden gemaakt van meet apparatuur zoals de Kinect, waarbij de scans meer noise op leveren, maar wel goedkoper zijn om te maken. (2) Een tweede punt is missing data. Bij real-time scanning, zoals dat gebeurt bij bewegende beelden, is het grote probleem dat de data nooit volledig is. De densiteit van de puntenwolk is namelijk niet hoog. Daarom is dit ook de belangrijkste eigenschap, die we moeten in acht nemen bij de vergelijking van de data. (3) Een andere eigenschap, die we vergelijken, is performantie. Ondanks dat er zelf geen onderzoek is gebeurd naar de performantie, gebruiken we beschikbare gegevens uit (Berger et al., 2014) om de algoritmen met elkaar te vergelijken. (4) Tot slot handelt de laatste kolom over de puntenwolk. Hiermee willen we aantonen of het algoritme een georiënteerde puntenwolk verwacht of niet.

Uit dit overzicht kunnen eenvoudig enkele conclusies worden getrokken. Zo zien we direct dat KinectFusion voor zowel noise als missing data geen robuustheid vertoont. Daarom is dit algoritme ook geen correcte oplossing voor het voorgestelde probleem. Daarnaast zien we direct dat Fourier en Poisson reconstructie de elementen zijn die de hoogste factoren hebben op het gebied van noise en missing data. Al is op het gebied van performantie er toch een zeer grote achteruitgang ten opzichte van wavelets, die ook sterk presteren voor missing data en noise. Hier kan dus best een afweging worden gemaakt tussen winst in performantie ten opzichte van een robuuster algoritme. Ondanks het gebruik van een surface reconstruction algoritme, is er toch nog steeds ruis aanwezig in het model. Daarom moet er nog gebruik worden gemaakt van andere technieken om deze ruis te reduceren. In het vervolg van het hoofdstuk zullen deze problemen met hun oplossingen worden besproken.

3.3 Processing

Zoals eerder verklaard, is processing een extra stap in de pipeline, waarbij er nadruk wordt gelegd op het verwerken en verbeteren van de verkregen data. Hierbij wordt speciaal aandacht gegeven aan: outlier removal, interpolatie, positionering, etc. . Volgens (Gross & Pfister, 2007) dient deze stap voor het oplossen van voornamelijk noise en outliers. Hierbij stellen zij voor om meerdere verschillende algoritmen, met parameters, aan te bieden waartussen de gebruiker dan kan kiezen. Deze mogelijkheid laat de gebruiker toe om de best mogelijke oplossing te vinden voor het huidige model. Zowel bij Surface Reconstruction als bij Processing is het uiteindelijk doel een beter resultaat te bekomen. Omdat generieke oplossingen voor de voorgestelde problemen echter niet mogelijk zijn (dankzij onder andere gewenste densiteit, noise niveau van de inputdata en andere aspecten van de dataset) zijn er veel verschillende algoritmen, die ieder een specifiek type van fouten kan rechtzetten.

In dit hoofdstuk zullen drie verschillende problemen worden aangehaald en besproken. Voor ieder van de problemen wordt er ook een mogelijke oplossing gegeven. Het eerste deel geeft meer uitleg over het samenvoegen van meerdere datasets. Dit kunnen al dan niet puntenwolken zijn. Ook hiervoor wordt een onderscheidt gemaakt. Daarna wordt er dieper ingegaan op outlier removal en welke technieken dit probleem kunnen oplossen. Tot slot wordt er ook dieper ingegaan op het opvullen van ontbrekende data in de uiteindelijke dataset. Al de processen die worden besproken in dit hoofdstuk kunnen zowel voor als na

surface reconstruction plaatsvinden. Sommige technieken zelfs in beide gevallen, dit kan de resultaten drastisch verhogen, indien er gebruik wordt gemaakt van surface reconstruction technieken, die zelf punten verplaatsen of bijvoegen.

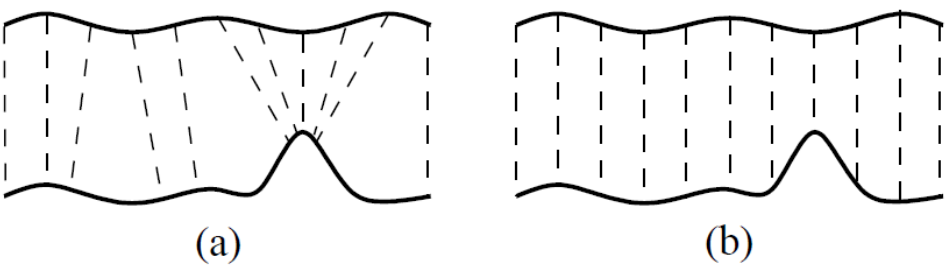
3.3.1 Multi-Source Stitching

Indien een scene of object wordt ingescanned door middel van dieptescanners, is het belangrijk dat alle delen van het object worden bekeken. Bij statische scenes of objecten kan dit worden gedaan door middel van registration en merging van verschillenden opnames. Voor meer uitleg over deze technieken wordt er verwezen naar sectie 3.1.3. In het algemeen zorgt het samenvoegen van meerdere frames, standpunten of resultaten voor een verhoging in detail voor het object. Hiervoor moet echter wel worden gezorgd dat de verschillende captaties correct op elkaar zijn afgesteld. Om deze technieken te bespreken moet er een onderscheid worden gemaakt tussen verschillende technieken. Een eerste techniek is het gebruiken van meerdere opeenvolgende frames doorheen de tijd. Hierbij wordt er meestal gebruik gemaakt van een bepaald toestel, dat rond het object beweegt. Een tweede techniek maakt gebruik van meerdere opname toestellen, die vanuit verschillende standpunten een dieptescan maken van het object. Door deze te aligneren kan er dan gebruik worden gemaakt van de verschillenden scanners voor eenzelfde resultaat te genereren. Het is ook mogelijk een combinatie van beide technieken te maken.

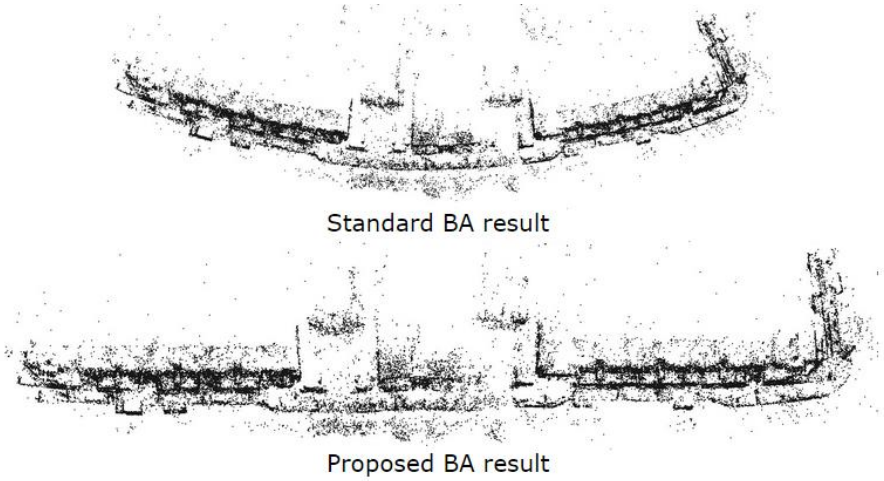
Bij het stitchen van verschillende scenes is het ook belangrijk om het verschil tussen dynamische en statische scenes aan te halen. Bij een statische scene is het onbelangrijk op welk tijdstip een bepaalde opname is gemaakt. Zolang dat het over dezelfde scene gaat is het mogelijke een alignatie van de data te maken. Bij dynamische scenes moet er echter rekening worden gehouden met het opname moment. Enkel beelden die hetzelfde moment zijn opgenomen kunnen worden gealigneerd. Zoals eerder vermeld is het mogelijk om ook doorheen de tijd bepaalde frames samen te voegen. Hierbij wordt er gebruik gemaakt van warping of mapping om een nieuwe frame te hervormen zodat deze past op een voorgaande frame. Op deze manier is het wel mogelijk informatie uit verschillende frames te combineren en zo het uiteindelijke resultaat te verbeteren.

Eerst worden er verschillenden *automatische technieken* besproken. Deze technieken zorgen voor het stichten van meerdere beelden, frames of datasets tot een volledig resultaat. Hierbij is er geen interactie nodig van de gebruiker. In sommige gevallen is het echter niet mogelijk om deze technieken te gebruiken. Zo kunnen sommige beelden te weinig informatie bevatten om een referentie punt te vinden, maar omdat menselijke gebruikers de scene kennen, of zelf kunnen waarnemen, is een manuele alignatie ook een mogelijkheid. Deze technieken bevatten meestal een tool waarmee de verschillende datasets over elkaar worden geschoven tot ze volledig overlappen. De automatische technieken zijn echter eenvoudiger in gebruik indien er aan de voorwaarden van de algoritmen wordt voldaan. Meestal gaat het hier over overeen lappende delen in de verschillende datasets. Omdat manuele oplossingen vaak toepassing gebonden zijn, wordt er in deze bespreking enkel focus gelegd op de automatische algoritmen.

Statische mapping is een belangrijke techniek, die sinds lange tijd wordt gebruikt om 3D data aan te maken. De Kaiser panorama, vermeld in sectie 3.1.2, is een voorbeeld van een techniek waarbij duidelijk twee verschillende frames worden gebruikt. Hierbij heeft elke individuele afbeelding echter zelf geen diepte informatie. Indien iedere frame (in het vorig geval de afbeelding) een eigen diepteframe heeft, kunnen we deze ook samenvoegen. In dit geval spreekt men van *Registration* (zie sectie 3.1.3). Hierbij worden de twee datasets aan enkaar gemapped door middel van de kennis van de camera standpunten van het toestel. Het is belangrijk op te merken dat deze stap enkel mogelijk is indien er kennis is over de exacte beweging van de dieptescanner. Zonder deze informatie is het nodig om te zoeken naar referentiepunten of referentiestructuren om de verschillende frames te aligneren. De meest gebruikte techniek voor dit te realiseren in Iterative Closest Point (Besl & McKay, 1992). Deze techniek berekent de rotatie en translatie die nodig is om een bepaalde dataset gelijk te stellen met een tweede dataset. Hiervoor is er wel een vereiste dat outlier removal eerst wordt toegepast, zie sectie 3.3.2. ICP biedt een oplossing voor het aligneren van point clouds zonder vooraf bepaalde referentiepunten, alsook voor het matchen van free-form curves. Het algoritme gaat er vanuit dat er gestart wordt met een gok naar de juiste rotatie en translatie. Eenmaal deze is gemaakt, zal het algoritme ervoor zorgen dat de afstanden tussen de twee datasets zo klein mogelijk worden. Er wordt gebruik gemaakt van de kwadratische afstanden tussen gelijk vallende punten. Voor meer informatie rond deze techniek wordt verwezen naar (Fitzgibbon, 2003; Rusinkiewicz & Levoy, n.d.). Zonder in detail te moeten treden in verband met ICP en de varianten, zijn er toch enkele belangrijke eigenschappen van de puntenwolken, die nodig zijn alvorens het algoritme kan werken. De belangrijkste eigenschap is aangaande overeenkomende punten. Voor een bepaald aantal beelden met elkaar te kunnen aligneren door middel van ICP, is het belangrijk dat een groot deel van de data overeenkomstig is. Indien er dus gebruik wordt gemaakt van een voor- en achteraanzicht van een object zal het ICP algoritme geen correcte oplossingen kunnen vinden. Er zijn namelijk geen punten die kunnen worden vergeleken aan de hand van hun onderlinge afstand. Voor enkele toepassingen is het dus onmogelijk om gebruik te maken van dit algoritme. Indien er echter sprake is van een voor- en achteraanzicht van een bepaald model, wordt dit vaak gedaan door middel van meerdere dieptescanners. In het geval van een enkele dieptescanner zouden namelijk alle tussenliggende beelden ook worden geregistreerd waardoor ICP wel werkt. Tussen elke twee opeenvolgende frames zitten meer dan voldoende referentiepunten (punten die overeenkomen in beide frames).

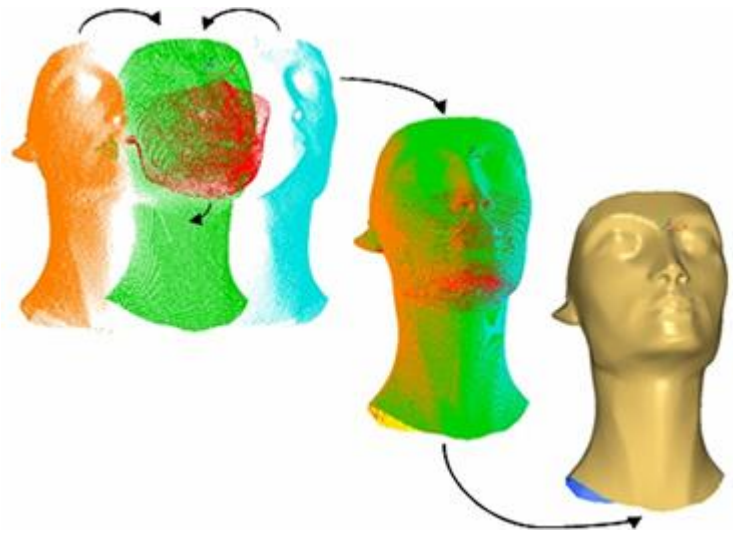


Figuur 14: ICP Detectie bij ruis. (Rusinkiewicz & Levoy, n.d.)



Figuur 15: Het gebruik van SfM na digitaliseren (Cohen, Zach, Sinha, & Pollefeys, 2012)

Er is ook een andere manier om door meerdere opeenvolgende frames 3D-data op te bouwen. Omdat er bewegende beelden worden geregistreerd, is het mogelijk om door middel van *Structure from Motion* (Ozden, n.d.) meer detail toe te voegen aan een 3D model. Dit is mogelijk omdat er van elk object meerdere standpunten zichtbaar worden doorheen de tijd. Zoals eerder vermeld, zorgen meerdere standpunten voor een hoger detail in de gecreëerde puntenwolk. Men kan dus gebruik maken van de bewegingen om meer informatie te krijgen over de objecten die zijn in gescand. Deze methode verschilt van registration. Registration gaat er namelijk van uit dat er enkel statische objecten in beeld worden gebracht. Bij Structure from Motion is het echter de beweging die voor extra informatie zorgt. Dit kan zowel de camera zijn (statische scene) of het object (dynamische scene). Structure from Motion is echter ontwikkeld om een statische scene in kaart te brengen en dus standaard niet opgebouwd om dynamische modellen samen te stellen. Al biedt (Ozden, n.d.) hier wel een oplossing voor. Deze oplossing laat toe om meerdere dynamische objecten op te bouwen doorheen de tijd, iets wat dus zeker extra punten kan opleveren voor kort opeenvolgende frames op te bouwen. Er moet hier wel worden vermeld dat SfM een intensief algoritme is, zeker indien er eerst een digitale scene moet worden opgesteld, waarin het SfM algoritmen wordt gedigitaliseerd en nagebootst op de verkregen data.



Figuur 16: Aligneren van meerdere puntenwolken. (www.mathnathan.com)

Naast de eerder vermelde technieken, die opeenvolgende frames gebruiken voor de kwaliteit te verhogen, is het ook mogelijk om twee volledige datasets van verschillende toestellen te gebruiken. Zoals vermeld werd bij registration, wordt er vanuit gegaan dat er slechts gebruik wordt gemaakt van een enkel scanapparaat dat alle data genereert en wordt geregistreerd. Voor meerdere toestellen moet er dus een andere techniek worden gebruikt. Bij het gebruik van meerdere toestellen, en dus meerdere puntenwolken, wordt er gesproken van *Multi-cloud data*. Dit is een belangrijke methode om de kwaliteit van puntenwolken te verbeteren. 3D scanning hardware is namelijk niet in staat om exacte 3D modellen op te bouwen. Dit is een probleem, dat voortkomt uit de discrete waarden die worden opgehaald door 3D scanning hardware. Omdat er dus wordt gewerkt met individuele punten, puntenwolken, is het de bedoeling deze met een zo hoog mogelijke accuraatheid te ontwerpen. Door meerdere datasets te combineren, kan men echter ervoor zorgen dat er meer detail komt in de puntenwolk. Bij een statische scene, met statische camera's, kan dit eenvoudig worden gerealiseerd door puntenwolken te roteren en positioneren volgens de verschillen in positie tussen de twee camera's. Hiervoor is ook geen alignatie in tijd nodig. Indien er echter wordt gewerkt in een dynamische scene met meerdere camera's, kan zowel de scene als de camera-positie veranderlijk zijn en is het noodzakelijk ook een alignatie in tijd te voorzien. Deze manier van alignatie wordt ook wel *dynamische mapping* genoemd. Strikt gesproken wilt dit zeggen dat tijdens de captatie er een tijdsynchronisatie is tussen de verschillende scanapparaten. In het optimaal geval zijn ook de intervallen tussen verschillende frames vastgelegd. Bij het aligneren van twee dynamische scenes zijn er echter enkele problemen. Zo is het zeer moeilijk referentie punten te vinden aangezien er bewegende elementen zijn in de scene. Een oplossing hiervoor is door elke scene te aligneren met de corresponderende scene uit de andere datastroom. Dit vergt echter veel tijd aangezien dit voor elke frame opnieuw moet worden gezocht en er geen referentiepunten kunnen worden behouden. Een andere techniek is gebruik te maken van extra kennis in de scene. Dit kan door een bepaald referentie object te plaatsen en dit te volgen gedurende de volledige opname. Een handige techniek is het gebruiken van een skelet indien dit aanwezig is in de scene. Deze methode wordt uitgelegd door (Zheng et al., 2009) en gebruikt door (Yeung, Kwok, & Wang, 2013). Er wordt een skelet voorzien waarnaar wordt gekeken in elke frame. Voor de translaties en rotaties terug te vinden, wordt er gekeken naar de verschillen in de skeletten. Deze kunnen namelijk voor elke frame worden bepaald. Het gelijkstellen van skeletten kan eenvoudig door de ruggengraat gelijk te stellen. Hiervoor moeten er geen referentiepunten meer gezocht worden. Alsook is het niet mogelijk dat referentiepunten niet zichtbaar zijn op andere dieptescanners. Dit lost ook direct het probleem op indien er enkel een voor- en achteraanzicht beschikbaar is. Indien het skelet kan worden geproduceerd kan alignatie altijd gebeuren.

3.3.2 Outlier Removal

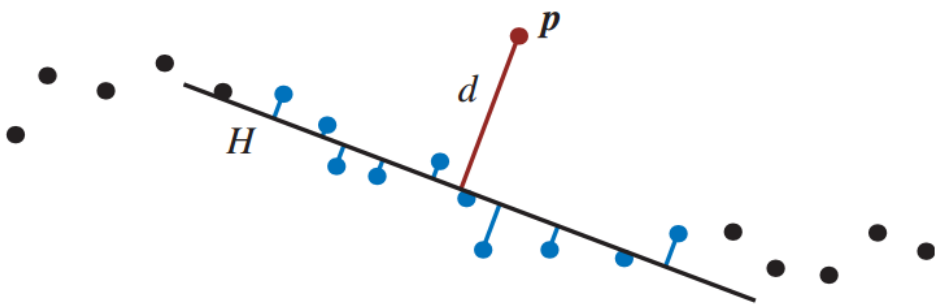
Bij surface reconstruction hebben we enkele elementen besproken die invloed hebben op het gebruikte model. Eén van deze elementen was noise, of ruis. Dit is een factor die zegt hoeveel van de opgemeten punten niet correct zijn geplaatst in de puntenwolk. Aangezien deze punten dus zorgen voor oneffenheden op het oppervlak van het model is het nodig deze te verwijderen uit de data. Het is wel geen triviaal probleem om op te lossen. Elk punt dat wordt gescand kan in principe ruis bevatten en dus afwijken van de werkelijke positie. Om erachter

te komen welke punten net afwijken van deze positie is er nood aan een heuristiek. Het concept waarbij er wordt gezocht naar de punten die afwijken van het oppervlak van een object, noemt men Outlier Removal (Xie, McDonnell, & Qin, 2004).

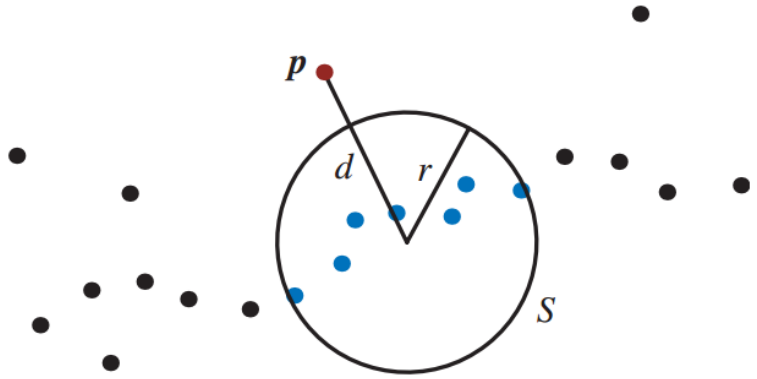
In een optimale omgeving weet het algoritme welke punten exact zijn gescand en welke zich niet op het model bevinden. Met andere woorden het algoritme heeft kennis over het volledige model dat wordt gemodelleerd. In realiteit is dit natuurlijk nooit het geval aangezien het net de bedoeling is deze informatie te achterhalen. Het is dus ook onmogelijk om met zekerheid een punt te schrappen van de puntenwolk. Er zijn echter wel enkele benaderingen die zich in het algemeen zeer robuust gedragen. Hierbij is het vooral belangrijk om te zorgen dat er niet teveel correcte punten worden geschrapt. Volgens (Xie et al., 2004) is het dus ook belangrijk om aan te nemen dat Outliers niet vaak voorkomen. In (Gross & Pfister, 2007) worden drie criteria voorgesteld die een schatting maken of bepaalde punten al dan niet Outliers zijn ten opzichte van het oppervlak. Hiervoor wordt er bij elk algoritme gebruik gemaakt van k-nearest neighbours voor elk punt P dat wordt gecontroleerd. Door deze methode toe te passen wordt elk punt individueel behandeld (en heerst er dus geen bias).

Het eenvoudigste algoritme voor het bepalen van outliers is het *Plane Fit Criterion.* Zoals alle benadering die hier worden besproken, wordt er hierbij gekeken naar de buren van het gekozen punt. Deze benadering heeft de simpelste oplossing voor het probleem, een visualisatie van deze oplossing kan worden teruggevonden in Figuur 17. Er wordt gekeken naar de k-dichtste punten van 'P'. Door deze punten wordt één rechte getrokken, zodat het kwadraat van de afstand van de k-punten tot deze rechte zo klein mogelijk is. Daarna wordt er gebruik gemaakt van een bepaalde factor om te kiezen of een punt al dan niet te ver van het vlak ligt. Deze factor wordt genomen als $\frac{d}{d+a}$. Hierbij is 'd' de afstand tussen het punt 'P' en de rechte waarvoor de boven vermelde eigenschap geldt, en 'a' de gemiddelde afstand van alle punten 'k' tot dezelfde rechten. De ontwikkelaar kan zelf kiezen hoe groot deze factor mag worden afhankelijk van de gekozen dataset en gewenste resultaten.

Het *Miniball Criteria* (zie Figuur 18) heeft een andere aanpak ten opzichte van de dichtste buren. Hierbij wordt ervan uit gegaan dat punten zich vormen in clusters. Deze clusters kunnen worden omvat door een cirkel. Deze benadering begint met het vinden van de kleinst mogelijke cirkel die alle k-dichtste punten omvat. Voor deze cirkel 'S' geldt dan dat 'r', de straal van de cirkel, de afstand is tot de twee (of meer) verste punten van alle buren. De factor die



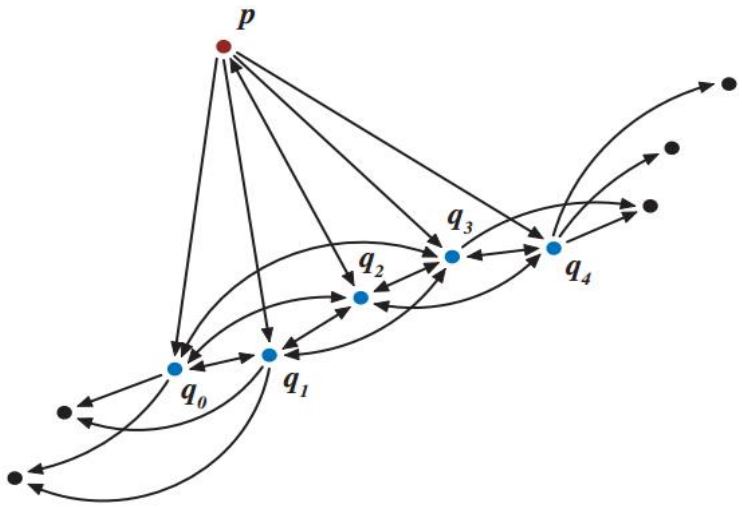
Figuur 17: Het Plane Fit Criteria; waarbij p het geselecteerde punt en H de rechte waarvoor de kwadratische afstand van de k-dichtste punten het kleinste is.(Gross & Pfister, 2007)



Figuur 18: Het Miniball Criteria; waarbij S de kleinst mogelijke cirkel is die alle buren omvat en r de straal van deze cirkel. (Gross & Pfister, 2007)

bepaalt of een punt al dan niet een outlier is, is zeer gelijkend op de factor-vergelijking die wordt voorgesteld bij de Plane Fit benadering. Hier verandert echter de factor 'a' naar '$2r\sqrt{k}$' waarbij 'k' het aantal punten buren is dat wordt genomen voor de cirkel te bepalen. Hierdoor is de formule dus $\frac{d}{d+2r\sqrt{k}}$ waarbij d de afstand is van het punt 'P' tot het middelpunt van de cirkel 'S'.

Een andere benadering is te gaan kijken naar de verzameling van k-dichtste punten van elk punt, dit noemt het *nearest-neighbour reciprocity criterion*. We kunnen namelijk stellen dat, indien een punt 'P' ligt in een sector van punten 'Q_1' tot 'Q_k' waarbij deze elementen zijn van de k-dichtste punten tot 'P', voor de meeste elementen 'Q_1' tot 'Q_k', P een element zal zijn van hun k-dichtste punten, zie Figuur 19. Met andere woorden, indien 'Q_1' een buur is van 'P' dan is de kans dat 'P' een buur is van 'Q_1' groot indien 'P' een element is van dezelfde sector. Indien 'P' echter een outlier is, dan merken we op dat de meeste punten van 'Q_1' tot 'Q_k' punt 'P' niet als buur hebben. Ook hier wordt een factor bekeken om te beslissen of er sprake is van een outlier of niet. Deze is $\frac{\|Unidirectionele\ buren\|}{k}$ waarbij de uni directionele buren, buren zijn waarvoor geldt dat indien 'Q' een element is van de buren van 'P', 'P' geen element is van de buren van 'Q'. In deze vergelijking is 'k' het aantal buren dat wordt gekozen.



Figuur 19: Graaf van dichtste punten; p samen met de 5 dichtste punten zijn afgebeeld. Merk op dat p enkel een element is van q2 zijn buren. (Gross & Pfister, 2007)

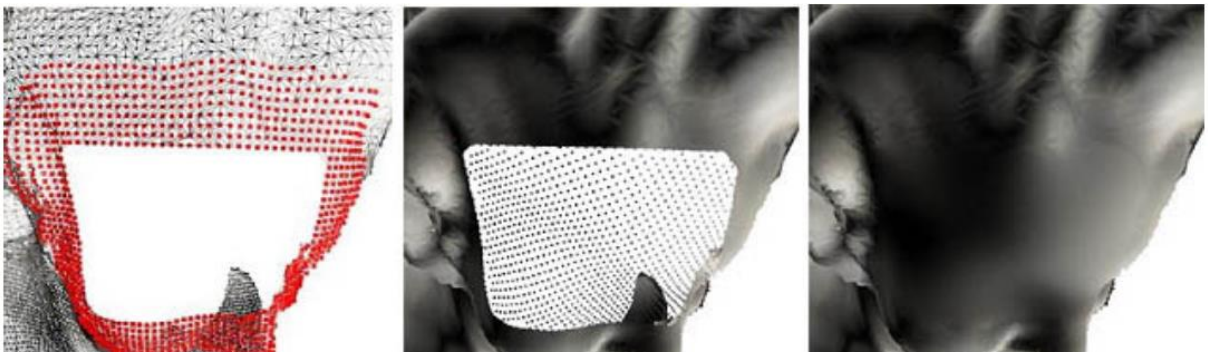
3.3.3 Hole Filling

De meeste objecten die worden gescand, door middel van eerder vermelde technieken binnen deze thesis, hebben problemen met ontbrekende of onbekende data (Brook & Alegre, n.d.; Brunton, Wuhrer, Shu, Bose, & Demaine, 2014). Hierdoor krijgen objecten lege ruimten of ontbrekende vlakken. Dit kan op meerdere manieren ontstaan. Veel sensoren zijn beperkt in hun accuraatheid en creëren daardoor gebieden waarbij er geen, of te weinig data aanwezig is. Een ander probleem is self-occlusion, waarbij een object een deel van zichzelf onzichtbaar maakt voor de sensor. Beide situaties leiden echter tot exact dezelfde waarneming in de puntenwolk; namelijk een lege ruimte in het object.

Om deze data toch aan te maken wordt er gebruik gemaakt van zogenaamde hole filling technieken. Deze technieken gebruiken de reeds vastgelegde data om nieuwe data aan te maken. Daarbij baseren de technieken zich op curves, die worden ontwikkeld door het gekende oppervlak. Door deze trends in de curves verder te zetten is het mogelijk data te genereren die passend is voor de data. Een andere mogelijkheid is de omtrek of grens (boundary) van het gat te interpoleren en zo het gat te dichten. In dit deel worden verschillende technieken besproken die mogelijk zijn om hole filling toe te passen. Er zijn enorm veel technieken omtrent hole filling en daarom proberen we enkel te focussen op recent uitgebrachte algoritmen. Belangrijk op te merken is dat niet elk algoritme gegarandeerd een oplossing vindt voor het vullen van de gaten.

In deze sectie leggen we enkele algoritmen in het kort uit om aan te tonen hoe de algemene aanpak voor dit probleem is. Er zijn echter veel meer mogelijkheden dan hieronder staan uitgelegd. Voor meer informatie over deze technieken kan er worden gekeken naar eerder onderzoek omtrent dit onderwerp; (Salamanca, Merch, Ad, Cerrada, & Inform, 2008)(Davis, Marschner, & Levo, n.d.)(Kumar, Shih, Ito, Ross, & Soni, n.d.)(Sciences, n.d.)(Wilczkowiak, Brostow, Tordoff, & Cipolla, 2005). Deze technieken gebruiken zowel 2D als 3D oplossingen om het probleem van gaten in data op te lossen.

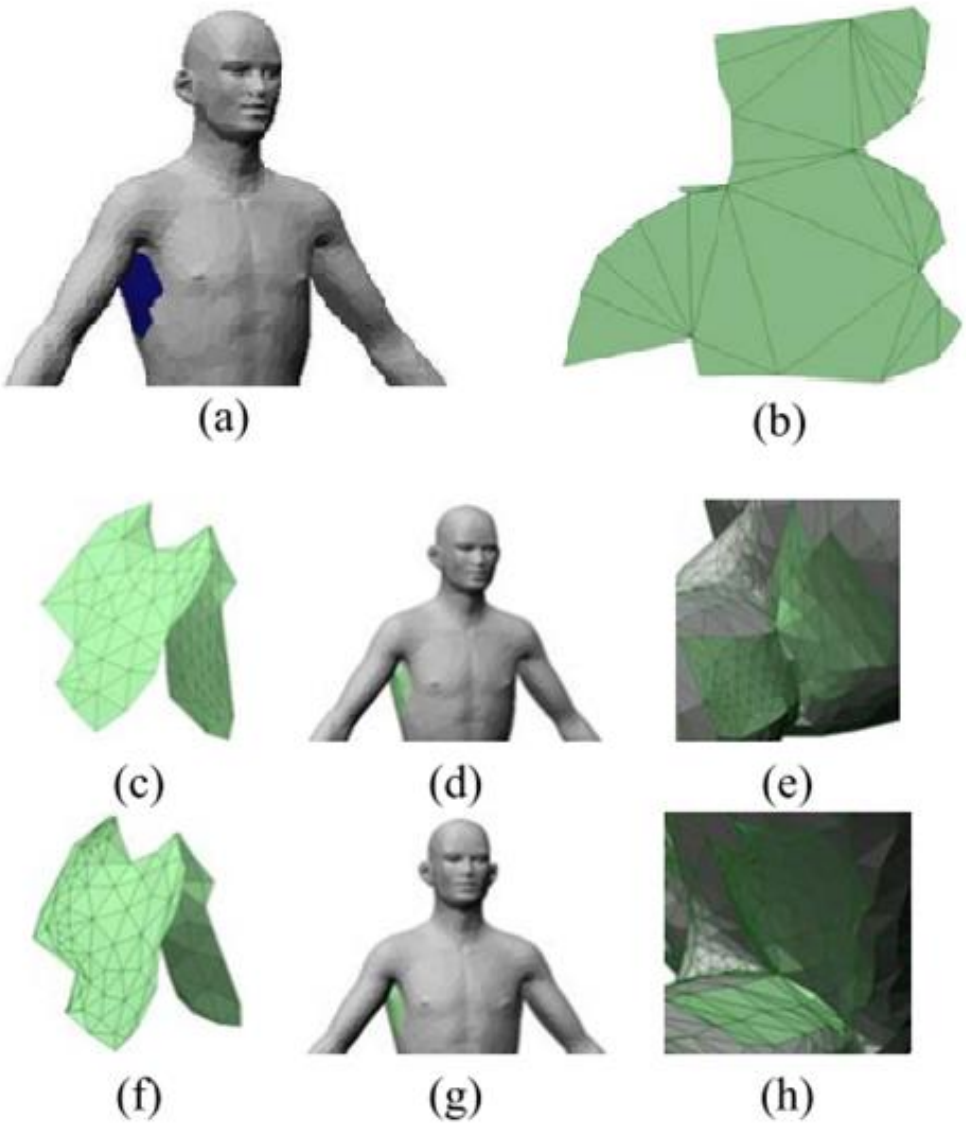
Eerst gaan we dieper in op het algoritme voorgesteld door (Brunton et al., 2014), genaamd *Curve Unfolding*. Dit algoritme vertrekt van een zogenaamde manifold surface (Grimm, 2002). Bij een manifold surface is het eenvoudig te achterhalen wat een rand is van het object, namelijk een zijkant dat slechts één polygon raakt. De rest van het algoritme vertrekt van een 2-dimensionale aanpak voor het probleem. In een twee dimensionale omgeving worden er 2D vlakken opgevuld. Dit zijn gebieden die we slechts over twee dimensies verschillen in waarden kennen. Indien we in een 3-dimensionale omgeving werken worden dit 3D surfaces genoemd.



Figuur 20:Hole Filling; Ontbrekende data wordt ingevuld door polygonen bij te voegen. (Brook & Alegre, n.d.)

Hierbij kunnen de waarden in de drie verschillende dimensies veranderen. Omdat het eenvoudiger is een 2D vlak op te vullen dan een 3D surface op te vullen, wordt de rand van het de ontbrekende data van het object 'geplooid' tot een 2D oppervlak. Dit zorgt ervoor dat het 3-dimensionale probleem wordt herleidt tot de 2 dimensionale oplossing van het probleem. Dit wordt aangetoond in Figuur 21.

Het verkregen oppervlak kan na het plooien dus worden gezien als een polygoon. Het is belangrijk dat bij het plooien van het oppervlakte er geen intersecties optreden, aangezien het de bedoeling is deze polygoon terug te plooien naar de oorspronkelijke positie van de randen. Indien er dus toch intersecties waren bij het plooien van de polygoon, dan zouden deze intersecties zich voordoen als fouten in de oppervlakte waarbij polygonen elkaar gaan kruisen. Dit oppervlak kan dan door middel van het Delaunay algoritme worden opgedeeld in verschillende polygonen. Deze afgewerkte polygonen worden dan terug geplaatst op de boundary waarvan men vertrokken is. Op deze manier vormen de polygonen een extensie van het reeds gekende oppervlak.



Figuur 21: Curve Unfolding. (a) het ontbrekende gebied. (b) De uitgeplooide polygoon. (c-e) Oplossing door minimaal oppervlak. (e-h) Oplossing door minimaal energie. (Brunton et al., 2014)

In sectie 3.2.1 bespreken we de toepassing van het MLS algoritme op het gebied van Surface Reconstruction. Daar blijkt dat het algoritme enkele interessante toepassingen heeft. Eén van de handige elementen van het algoritme is dat alle punten in acht worden genomen. Dit wil zeggen dat indien we MLS gaan gebruiken om gaten op te vullen, dat de eerder verkregen data niet wordt veranderd (Brook & Alegre, n.d.). Indien MLS enkel tendenslijnen zou gebruiken, die worden verkregen bij het bekijken van de volledige oppervlakte, is het mogelijk dat eerder verkregen data wordt aangepast. Dit komt omdat de trendlijnen die worden gebruikt niet volledig overeenstemmen met de eerder gevonden data. Dit is een ongewenste gebeurtenis aangezien we weten dat de oorspronkelijk data correct is. Deze artefacten zouden zich vooral toonbaar maken aan de randen van de ontbrekende data.

Alhoewel de meeste technieken berusten op hetzelfde schema wordt er gebruik gemaakt van verschillende benaderingen naar dit oplossingsmodel. Zo ook met MLS, de grote lijnen van het algoritme zijn gelijk aan die van andere oplossingen. Eerst wordt er gezocht naar een rand van het zogenaamd gat. Dan wordt dit gat geprojecteerd op een vlak. Wanneer dit is gedaan wordt het vlak opgevuld met nieuwe punten zodat het model afgesloten kan worden. En tot slot wordt het nieuwe vlak terug in het reeds bestaande oppervlak geprojecteerd. Bij de MLS methode is de grote lijn hieraan gelijk. Er zijn echter enkele kleine veranderingen. Zoals te verwachten aan de naam van de techniek wordt de MLS-methode gebruikt om afwijkingen te detecteren. Aangezien MLS het kwadraat van de afwijkingen zo klein mogelijk wilt houden, kunnen projecties en sample punten dus ook beter worden gekozen.

Sample punten zijn zo een andere benadering. Bij Curve Unfolding spraken we van een mesh gebaseerde aanpak waarbij er gebruik wordt gemaakt van polygonen. De MLS-benadering maakt echter gebruik van punten om opnieuw te samplen. Een resampling algoritme definieert enkele punten op het projectie vlak. Deze punten worden dan samen met het projectie vlak terug geplaatst op het reeds bestaande oppervlak. Eenmaal deze terugstaan, wordt de MLS methode gebruikt om nieuwe punten te definiëren op het oppervlak. Deze techniek is gelijk aan de techniek die eerder werd beschreven int sectie 3.2. Op deze manier wordt dus op een hoge kwaliteit nieuwe data aangemaakt die zorgt voor een mooi golvend oppervlak.

3.3.4 Discussie

De vorige secties stelden we enkele algoritmen voor die zorgen voor een hogere kwaliteit voor de puntenwolken of 3D modellen. Het toepassen van deze technieken is optioneel maar sterk aan te raden voor het verbeteren van een model. Zo wordt de ruis drastisch verlaagd, maar het is echter moeilijk om te kiezen wanneer deze technieken worden toegepast. Een procedure zoals Outlier Removal kan best worden gebruikt voordat er gebruik wordt gemaakt van een reconstructie algoritme. Enkele van deze algoritmen (zoals besproken in sectie 3.2) maken namelijk gebruik van functies die trendlijnen gebruiken voor het aanmaken van extra punten. Indien hier veel outliers tussen zitten, zullen de trendlijnen niet meer correct zijn en dus een slecht resultaat als gevolg hebben. Deze processing stappen kunnen ook gelijk met andere processen gebeuren (zoals bijvoorbeeld hole filling samen met reconstructie algoritmen. Dit kan ervoor zorgen dat het algoritme rekening houdt met de originele data alsook de data met minder ruis en fouten.

Voor 3D captatie wordt er in ieder geval geopteerd om altijd gebruik te maken van Outlier Removal technieken. Daarnaast is er bij reconstructie bijna altijd spraken van Hole Filling technieken. Dit is het gevolg van de captatie stap, waarin zijden van het object die niet direct zichtbaar zijn voor de camera niet worden gecapteerd en dus als gaten worden gedigitaliseerd. Indien echter deze algoritmen worden gebruikt, wordt het resultaat van een hogere kwaliteit.

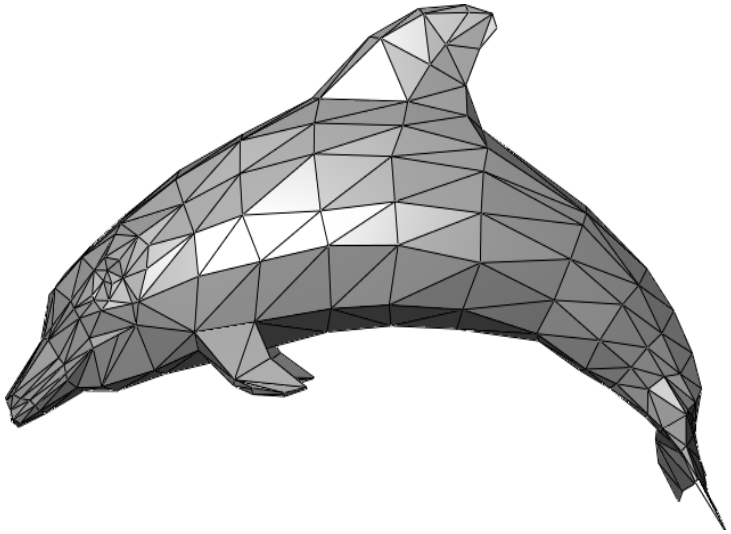
3.4 Formatting

Data formatting, in deze thesis formatting genoemd, beslaat de keuze van een representatie model voor de verkregen data. Er zijn in principe twee verschillende representatie technieken beschikbaar namelijk: Voxel data en Polygon data. Deze sectie bespreekt de voor en nadelen van deze gekende technieken alsook de alternatieve mogelijkheden die deze technieken aanbieden. Aangezien elke methode zijn voor- en nadelen heeft, is het dus sterk gebonden aan de situatie van de implementatie. Zo kan het zijn dat bepaalde bedrijven opteren om een bepaald model te gebruiken omdat hun in-house tools reeds overweg kunnen met de gekozen data (zie sectie 3.5).

Het belangrijkste aspect om een representatiemodel te kiezen is het vervolg van de pipeline. Afhankelijk van hoe de visualisatie wordt geïmplementeerd, kunnen er voordelen worden gehaald uit deze modellen. Hiermee wordt er vooral verwezen naar het gebied van rendering en netwerken, waarbij er voor beide modellen een volledig andere aanpak geldt. Zo is het concept van polygon rendering bijna volledig aangepakt en opgelost, terwijl er nog steeds veel onderzoek gebeurt in de rendering van voxels. Op het gebied van netwerken zijn vooral de compressie technieken bij beide modellen zeer belangrijk om zo de totale netwerkbelasting te kunnen verlagen.

3.4.1 Polygon Representatie

Het meest bekende model, en ook meest gebruikte model, om 3D modellen weer te geven in computer games of andere 3D computer graphics applicaties is het polygon model. Het polygon model baseert zich op het gebruik van polygonen. Hierbij worden een object dus opgesteld uit verschillende punten die met elkaar worden verbonden. Meestal worden hiervoor driehoeken gebruikt aangezien grafische hardware hiervoor geoptimaliseerd is.



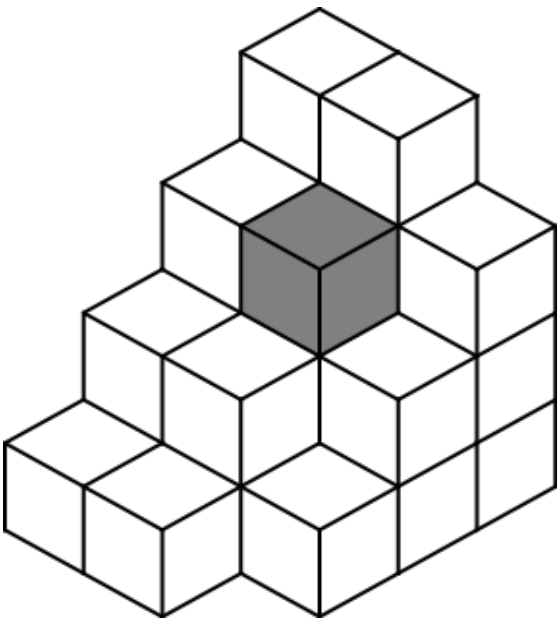
Figuur 22: Een Polygon mesh gerenderd zonder kleuren. (Wikipedia, 2014)

Objecten die worden opgebouwd uit verschillende polygonen worden vaak vermeld door “Polygon Meshes”. Dit zijn aaneenschakelingen van verschillende polygonen. Polygonen hebben echter maar één kant. Iedere polygoon (die dus standaard wordt opgedeeld in meerdere driehoeken) heeft een normaalvector, die aanduidt aan welke kant er een kleur/textuur/belichting moet worden weergegeven. Dit zorgt ervoor dat indien 3D objecten niet gesloten zijn en er langs een bepaalde kant doorheen kan worden gekeken.

Op het gebied van opslag en data formaat zijn polygonen zeer beperkt in schijfgrootte. Dit volgt uit het gebruik van driehoeken voor de representatie van modellen. Voor elke driehoek is het namelijk voldoende enkel de hoekpunten op te slaan. Door middel van deze hoekpunten kan er dan een 3D vlak worden gemaakt met een bepaalde kleur of textuur. Het nadeel hieraan is echter dat bij het verplaatsen van 1 hoekpunt er (in het geval van een gesloten 3D object) altijd meerdere polygonen worden beïnvloed door de verplaatsing. Hierbij moet rekening worden gehouden indien dit niet het gewenste resultaat is. Indien we echter objecten animeren, is dit een groot voordeel aangezien het herrekenen van de hoekpunten volstaat om een object te vervormen. Om animatie te doen zijn er echter ook andere aanpakken mogelijk zoals het gebruik van een skelet waaraan de polygonen worden gelinkt. Dit wordt later in deze sectie nog uitgewerkt. Een ander voordeel van polygonen is belichting. Omdat polygonen een normaalvector hebben is het eenvoudig de lichtsterkte te bepalen op elk punt van de polygoon. Al wordt er algemeen (Phong of dergelijke) gebruikt gemaakt van enkele gemeten lichtpunten op de polygoon waartussen wordt geïnterpoleerd. In het algemeen wordt hiervoor gebruik gemaakt van de hoekpunten van de polygoon.

3.4.2 Voxel Representatie

De data die wordt ingelezen door middel van 3D Acquisition vormt een puntenwolk. Deze puntenwolken zijn een samenhorende groep punten in een 3D omgeving. Om alle punten op te slaan die er zijn (dus ook lege punten) kost het gigantisch veel data. Daarom wordt de ruimte ingedeeld in verschillende kleine vakken. Deze vakken worden voxels genoemd. Zoals



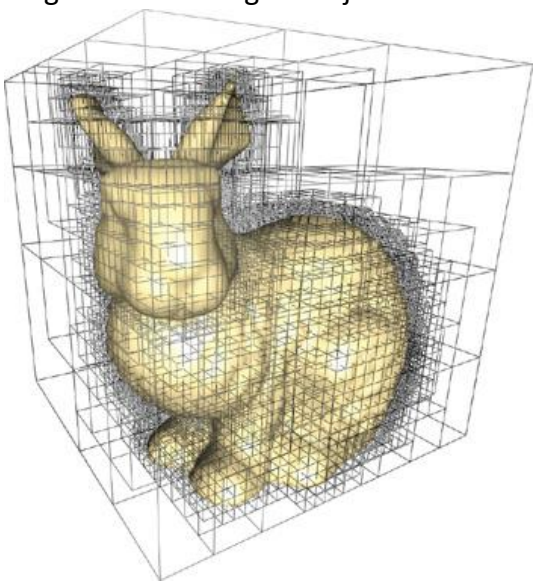
Figuur 23: Een Voxel grid. (Wikipedia, 2014b)

in Figuur 23 wordt getoond, neemt elke voxel een gebied in. Op deze manier kunnen grotere gehelen worden opgeslagen.

Het eerste probleem blijft echter gelden. Om data op te slaan is er enorm veel schijfruimte nodig. Om even kort een beeld hiervan te schetsen: een gebied van 1 kubieke meter, waarbij we per mm² een voxel genereren heeft 1 miljard voxels, indien we per voxel 64 bytes opslaan (kleur, positie,…), is er in totaal 59 Gigabyte aan schijfruimte nodig om deze scene op te slaan. Indien we dus naar grotere scenes gaan, neemt dit getal alleen nog maar toe. Het is hierbij wel belangrijk op te merken dat de precisie van de voxel-grid zeer belangrijk is, de opslaggrootte neemt namelijk toe met een factor x^3. Omdat deze ruimtes vooral uit leegte bestaan, is het niet nodig om al deze posities op te slaan. Daarom zijn er enkele methoden ontwikkeld die met deze kennis rekening houden en op die manier proberen efficiënter 3D data op te slaan. De volgende subsecties geven een korte introductie tot enkele van de meest gebruikte technieken om 3D punten data op te slaan. Deze zijn echter maar enkele technieken uit de vele die er bestaan om dit probleem aan te pakken. In het algemeen is het belangrijk te kijken naar de gewenste toepassing. Bepaalde methoden focussen zich op het snel toevoegen en veranderen van data in de puntenwolk terwijl andere technieken zich eerder focussen op het snel aanmaken en ondervragen van de puntenwolk.

Zoals eerder vermeld bestaat 3D data van voxels voornamelijk uit grote gebieden die leeg zijn. Omdat het niet nodig is deze gebieden op te slaan kunnen we dit als lege data zien en dus met nullen opvullen. Om deze efficiënt op te slaan kan er gebruik worden gemaakt van een hiërarchische (Elseberg, Borrmann, & Nuchter, 2011; Gross & Pfister, 2007) boom, deze wordt een *Octree* genoemd. De naam is afgeleid van het aantal onderverdelingen dat word gemaakt per niveau. Zo kan elke onderverdeling die volledig leeg is worden weergegeven door een “null” waarde. Andere gebieden waar wel nog gevulde voxels in zitten worden dan opnieuw onderverdeeld tot op het niveau van een individuele voxel.

De opsplitsing van de voxels kan gebeuren op twee verschillende manieren. Elk gebied kan worden opgedeeld in acht gelijke gebieden (region octree). Deze techniek zorgt voor een uniforme octree waarbij alle gebieden even groot zijn. Het is ook mogelijk om een gebied te



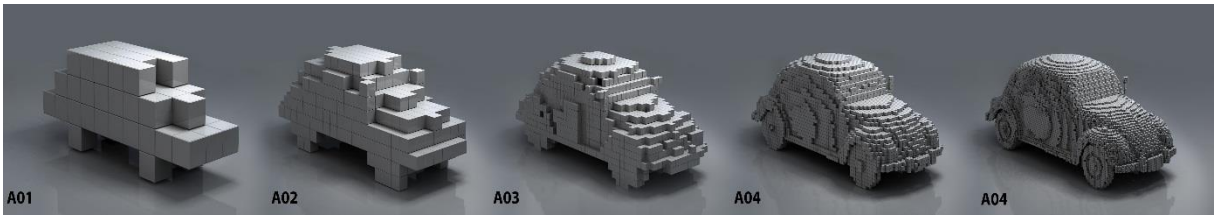
Figuur 24:Region octree datastructuur voor een 3D object.

splitsen over een bepaalde as op een bepaald punt binnen het gebied (point octree). Deze tweede methode zorgt voor een meer geoptimaliseerde boom aangezien de punten waarop wordt gedeeld specifiek kunnen worden gekozen. Een nadeel van deze techniek is dat er meer informatie moet worden bijgehouden voor iedere opdeling. Zo is het nodig om de as en de plaats van de opsplitsing op te slaan, iets wat niet nodig is bij een binaire opdeling over beide assen.

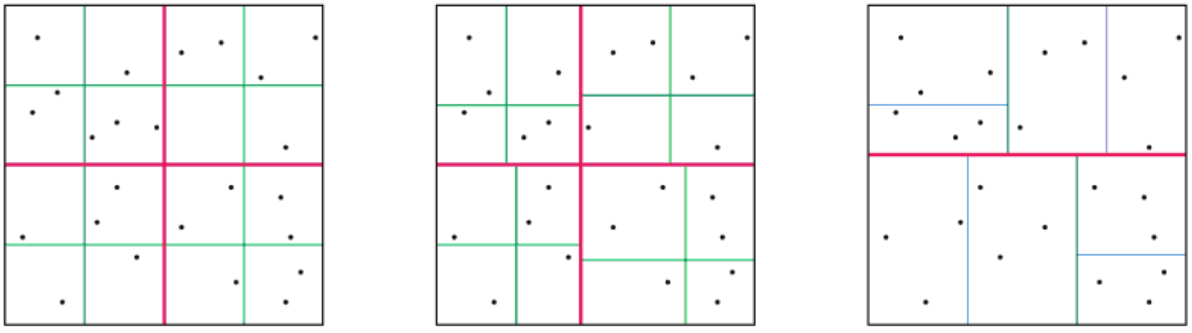
Het voordeel aan deze boomstructuren zit voornamelijk in het overlopen van de data. Omdat elk gebied onafhankelijk staat van elkaar is het eenvoudig om op een GPU elke tak van de boom individueel te overlopen om op deze manier sneller een weergave te krijgen. Wel is belangrijk op te merken dat beide types van octrees een ander resultaat kunnen hebben op het gebied van de verdeling. Aangezien een point octree een slimmere opsplitsing maakt dan de region octree zal deze een betere verdeling hebben op het gebied van de data in de boom. Beide bomen blijven echter met hetzelfde probleem zitten op het gebied van dataverdeling. Dit komt voort uit de binaire opdeling van de cellen. Een oplossing voor dit probleem is een K-D tree. Later deze sectie wordt deze datastructuur in detail besproken.

Een ander groot voordeel van een octree zien we terug bij LOD (Level-Of-Detail). LOD zorgt ervoor dat indien we ver van een object zijn we dit met minder punten/objecten gaan tekenen. LOD zal niet uitvoerig worden besproken binnen de thesis maar wordt aangehaald als voordeel van bepaalde structuren zoals de octree. Indien we een octree gaan tekenen, kunnen we kiezen met hoeveel detail we dit willen doen. Hoe dieper in de boom we gaan voor een weergave te genereren, hoe meer detail we terugvinden op het object. Figuur 25 toont aan hoe een object meer detail krijgt naarmate we dieper in de boom gaan. Vooral op het gebied van processing kracht zien we zo dat LOD geen berekeningen meer vereist maar enkel de huidige diepte in de boom moet weten. Ook voor designers is het op deze manier eenvoudiger om meerdere LOD's aan te maken. Deze zitten namelijk standaard in een object.

Zoals reeds werd vermeld, heeft een puntenwolk in de meeste gevallen geen evenredige verdeling. Omdat een octree geen rekening houdt met de verdeling van de elementen werd reeds voorgesteld om region octrees te gebruiken. In Figuur 26 zien we beide verdelingen voor een 2D opstelling. Hierbij zien we dat een normale quadtree (a) gewoon de gebieden onderverdeelt. Bij een meer adaptieve aanpak zoals region octrees, in dit geval een quadtree variant (b), kan een betere oplossing worden gevonden. De laatste oplossing (c) toont een andere methode om gebieden op te delen. Het is namelijk een variant van een zogenaamde K-D boom.



Figuur 25: LOD op een voxel gebaseerd object. (Figuur verkregen via http://www.bilderzucht.de/blog/3d-pixel-voxel/)



Figuur 26: Vergelijking tussen een (a) een quadtree, (b) een adaptieve quadtree en (c) en 2D split tree. (Gross & Pfister, 2007)

K-D tree is de korte aanduiding voor K-dimensionale boom. De K-waarde duidt in dit geval op de dimensie van elke node binnen de boom. Een K-D boom gaat elk gebied opdelen in 2 gebieden met evenveel punten. Op deze manier is de boom volledig evenredig opgebouwd aangezien in elke deelboom evenveel punten zitten. Dit zorgt ervoor dat het eenvoudig is bomen te overlopen aangezien elke deelboom ongeveer evenveel tijd kost om te overlopen. Bij een octree is hier echter geen zekerheid van aangezien vele gebieden leeg kunnen zijn en daardoor dus geen data bevatten. In het gegeven voorbeeld is de K-waarde gelijk aan 2 aangezien er over 2 verschillende dimensies wordt opgesplitst en hierdoor dus ook 2 nieuwe cellen aanmaakt.

Op het gebied van LOD zien we hier echter niet hetzelfde effect als bij een octree. Omdat elke deelboom gevuld is met evenveel punten zijn er geen lege deelbomen en moet dus de hele boom worden doorlopen alvorens we een volledig object kunnen tekenen. Waar we bij een octree standaard een LOD implementatie hebben, is dit niet het geval bij K-D bomen.

3.4.3 Skeleton Mapping

De twee verschillende representaties die hierboven werden beschreven hebben ieder hun eigen voor- en nadelen. Maar voor beide geldt hetzelfde probleem. Indien we deze over netwerken moeten streamen, krijgen we problemen met de nodige bandbreedte. Om dit probleem aan te pakken, is het belangrijk dat deze data efficiënt opgeslagen kan worden. Al wordt er nog teruggekomen op dit probleem in sectie 4.2, toch is het belangrijk op te merken welke techniek er gebruikt wordt om de gegeven data eenvoudiger op te slaan en door te sturen.

Aangezien de toepassing voor deze techniek voornamelijk is bedoeld voor het opnemen van bewegend personen, kan er worden gekeken naar de datastructuren, die worden gebruikt om zulke objecten te animeren of op te slaan. Aangezien we personen proberen weer te geven is het logisch dat we gaan kijken naar skeletten om door middel van enkele joints een volwaardige persoon op te bouwen op het gebied van animatie. Joints zijn de weergaven voor de schakelpunten in het menselijke lichaam. Daarnaast moet de 3D puntenwolk of the polygon mesh ook worden meegestuurd met deze data. Omdat er vanuit wordt gegaan dat we het hebben over onveranderlijke objecten, namelijk geen verlies van lichaamsdelen, is het niet nodig om elke keer de volledige puntenwolk mee te sturen. Daarom is het mogelijk deze puntenwolk te mappen op een skelet. Op deze manier moet enkel in het begin de puntenwolk

worden doorgestuurd en hoe deze op het skelet wordt geplaatst. In het vervolg is het dan voldoende om enkel het skelet of de veranderingen binnen het skelet door te sturen.

In sectie 5.1 is de Kinect van Microsoft besproken. Een interessante functie van de Kinect is dat deze skeletten opneemt tijdens de 3D Acquisition. Dit zorgt ervoor dat het eenvoudig is om een 3D puntenwolk te mappen op het door de Kinect SDK meegekregen skelet (niet te verwarren met KinectFusion). Deze techniek zorgt er ook voor dat eventuele animatie veel sneller kan gaan. Dankzij een Denavit-Hartenberg Skelet (zie sectie 4.3.1) is het eenvoudig om animaties uit te voeren door middel van forward of inverse kinematics. Al blijft het ook mogelijk om het object gewoon te zien als een statisch object binnen de scene.

3.4.4 Discussie

Tijdens de bespreking van formatting zijn er enkele elementen naar voor gebracht om de data in op te slaan. Zo zijn polygonen een efficiënte manier van data opslag. Puntenwolken daarentegen kunnen worden opgeslagen als input data zoals ook de Kinect ze levert. Hierdoor is het eenvoudiger om bepaalde algoritmen uit te testen aangezien de captatie stap kan worden geëmuleerd. Dit is vooral handig bij het verwerken van meerdere algoritmen en bij het aanpassen van dergelijke technieken. Daarnaast zijn polygonen een traditionele manier voor het voorstellen van 3D karakters. Hierdoor is er veel informatie over dergelijke datastructuren en hoe ze zo efficiënt mogelijk worden gebruikt beschikbaar. Tot slot werd er kort een introductie gegeven over een manier van opslaan waarbij er een combinatie wordt gemaakt tussen de traditionele opslag van een karakter, waar zowel een skelet als polygon data wordt opgeslagen, en het opslaan van puntenwolken.

Aangezien de uiteindelijke implementatie, die wordt gemaakt in deze thesis zich focust op het verwerken van puntenwolken, lijkt het ook logisch dat er gebruik wordt gemaakt van dergelijke representatie technieken. Daarnaast ondersteund een combinatie van puntenwolken en skelet-data ook alle acties die mogelijk nodig zijn tijdens de implementatie. Aangezien de puntenwolken de originele data bevatten, sluit het ook uit dat er eventuele fouten worden gemaakt alvorens het algoritme van Skeleton Mapping wordt gebruikt. De samenloop van al de voordelen die worden geboden door gebruik te maken van de combinatie techniek leidt er toe dat er wordt geopteerd deze te gebruiken tijdens de implementatie.

3.5 Modeling

Alvorens 3D Acquisition werd gebruikt voor het verkrijgen van 3D objecten op digitale devices, werden deze objecten gemodelleerd. Een designer werkte door middel van een toolset een object uit in een 3D omgeving. Zoals we reeds weten zijn de modellen die worden aangemaakt door 3D Acquisition niet altijd correct of kwalitatief goed genoeg voor de toepassing. In dit geval is het mogelijk gebruik te maken van modelling tools om eventuele fouten handmatig weg te werken. Het is belangrijk in te zien dat voor bijvoorbeeld game-development dit wordt gedaan in de engine. Ook al kan er gebruik worden gemaakt van 3D pakketten zoals 3Ds Max, Maya, Ogre3D, toch wordt er meestal geopteerd voor een eigen set van tools om objecten mee te modellen of aan te passen. Om die reden wordt er ook niet dieper ingegaan op modelling binnen deze thesis.

Dit volledige hoofdstuk bespreekt alle elementen om tot een goed model te komen dat kan worden gebruikt binnen games of andere 3D toepassingen. Er zijn echter nog enkele stappen die moeten worden doorlopen om tot een volwaardige visualisatie te komen. De stappen die hier voor nodig zijn, al dan niet optioneel in het proces, worden in het volgende hoofdstuk besproken. Het is hierbij belangrijk op te merken dat het 3D model dat wordt aangemaakt door de reeds besproken technieken als input dient voor de algoritmen, die worden besproken in het volgende hoofdstuk.

4.Visualisatie Analyse

Hoofdstuk 3 behandelde de volledige pipeline over het aanmaken en vervolledigen van 3D modellen vanuit een object. Eenmaal dit object volledig is aangemaakt, moet het echter nog worden weergegeven. Hiervoor zijn er diverse methodes binnen het renderen. Maar de visualisatie pipeline bespreekt niet enkel het renderen maar ook het versturen van de data over een netwerk en het animeren van deze 3D modellen. Al deze elementen zullen worden behandeld in de volgende secties. Omdat alle stappen in dit hoofdstuk sterk afhankelijk zijn van de applicatie die er gebruikt van maakt is het echter niet mogelijk om alle specifieke voorbeelden te overlopen. Daarom zal het algemeen concept worden duidelijk gemaakt met de grootste onderverdelingen. Ook zal bij het netwerk gedeelte worden gekeken naar de pipeline zonder een centrale server. Het netwerk gedeelte wordt hier gesitueerd omdat het niet bijdraagt tot het aanmaken van een 3D mesh of digitaal object. Het netwerk gedeelte bespreekt de manier waarop verkregen data kan worden verdeelt tussen meerdere computers.

4.1 Rendering

Indien heel de Data Generation pipeline is doorlopen, hebben we een verzameling van 3D objecten die samen horen in de tijd. Het belangrijkste aan de Visualisatie pipeline is het weergeven van deze 3D objecten (polygonen of puntenwolken) doorheen de tijd. Hierbij is het belangrijk op te merken dat dit wordt gedaan op augmented reality brillen. Het standpunt van de kijker kan dus veranderen doorheen de tijd. Ook is deze thesis geschreven voor toepassingen in de gaming, en wordt er dus real-time rendering verwacht. Al is het mogelijk dat de hardware daarvoor op dit moment nog niet aanwezig is. Methodes die pas correcte weergaves genereren doorheen de tijd zijn dus niet van toepassing hier. Er kan vooral worden gedacht aan raytracers die op een stochastische manier worden opgebouwd. Deze nemen namelijk elke cyclus enkele punten om het beeld te verbeteren, wat ervoor zorgt dat het basisbeeld slecht is maar beter wordt doorheen de tijd.

Het blijft wel belangrijk op te merken dat er twee verschillende mogelijke data elementen worden gebruikt. De eerder besproken representatie modellen (Polygon meshes en Voxels) hebben beide een verschillende rendering methode. Zoals eerder vermeld hangt de keuze van de representatie af van de inhouse toolkits en de verwachten performantie in rendering. In het verloop van dit hoofdstuk zullen we deze tweede invloed bespreken. Daarnaast zullen ook andere voor- en nadelen van beide technieken worden besproken. Toch blijft deze keuze afhankelijk van de uiteindelijke implementatie. Dit hoofdstuk biedt slechts een overzicht van enkele gebruikte technieken die op dit moment gebruikt worden, samen met de algemene voor- en nadelen van deze technieken.

4.1.1 Polygon based rendering

Polygon rendering is een veelbesproken onderwerp. Vooral in de gaming is er veel vooruitgang geboekt in de laatste decennia. Dankzij de vele onderzoeken die zijn gebeurd naar dit onderwerp zijn de best-practices al vastgelegd over de jaren. Er zijn enkele mogelijkheden om

polygonen of andere te renderen door middel van bibliotheken. Zo zijn DirectX (Luna, 2008) en OpenGL (Hill & Kelley, 2007) zeer bekende bibliotheken om grafische representaties van modellen te maken.

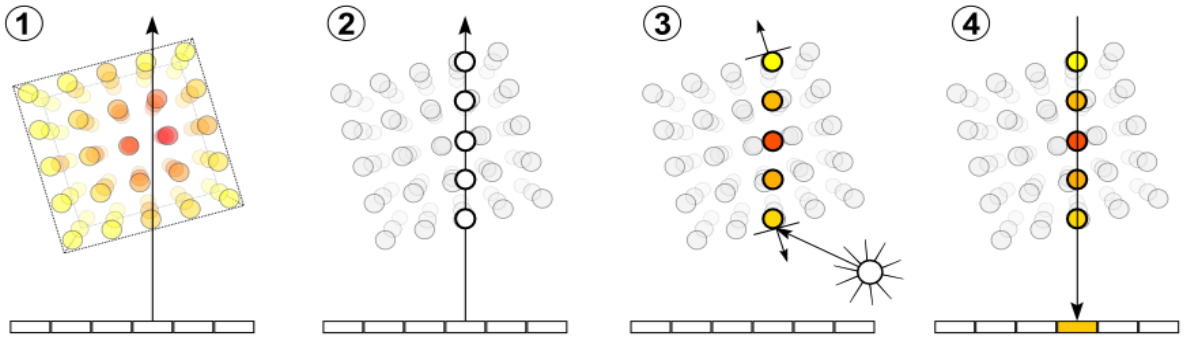
Deze bibliotheken maken gebruik van de grafische kaart om polygonen om te zetten naar pixels op het scherm. Hiermee wordt een volledige 3D scene opgebouwd waarin de camerapositie kan veranderen over tijd. Deze techniek is enorm bekend geraakt bij computer games omdat de nodige rekenkracht voor deze representatie laag is. Hierdoor kan in real-time een representatie worden gemaakt van een bepaalde scene, ervan uitgaande dat de scene niet teveel polygonen bevat. Iets wat wel direct duidelijk wordt bij deze weergave is dat de snelheid van het renderen zeer sterk afhankelijk is van de gebruikte hardware. Dit is iets wat natuurlijk geldig is voor alle berekeningen op een computer. Maar aangezien grafische toepassingen, zoals computer games, meestal gebruik maken van zoveel mogelijk polygonen is het moeilijk om oudere hardware te gebruiken om state-of-the-art representaties te maken. Indien we kijken naar methoden die niet real-time zijn, zoals raytracing, valt dit probleem weg omdat het algoritme inherent zichzelf verbetert over tijd. Het blijft wel een onderzoek om deze techniek in real-time te krijgen (Purcell, Buck, Mark, & Hanrahan, 2002). Dankzij de grote vooruitgang in computer-hardware is het tegenwoordig wel mogelijk om enkele soorten raytracing in real-time te kunnen uitvoeren.

4.1.2 Voxel based rendering

In tegenstelling tot Polygon Based Rendering is het onderzoek nog maar pas begonnen. Hiermee wordt bedoeld dat er nog geen echte best-practices zijn. Dit is een resultaat van de marktdominantie van de grafische kaarten. De huidige GPU's zijn geoptimaliseerd voor polygon rendering waardoor het voor software fabrikanten interessanter is deze techniek te blijven gebruiken. Er wordt daarom ook veel teruggegrepen naar polygonen om puntenwolken te renderen. Puntenwolken worden omgezet naar een polygon mesh waardoor er nog steeds bekende algoritmen kunnen worden gebruikt om dit weer te geven.

De laatste jaren wordt er meer en meer aandacht geschonken aan het renderen van de puntenwolken zonder omzetting naar polygonen. Dankzij dit onderzoek zijn er vele technieken ontdekt die zorgen voor een snelle en betrouwbare weergave van de data. In deze sectie worden enkele methode besproken. Hieronder valt (1) Volume Ray Casting, een adaptatie op het ray cast algoritme dat wordt gebruikt bij polygon rendering. Daarna wordt ook de (2) Splatting methode besproken. Splatting is de meest gebruikte methode voor het renderen van voxel data. De (4) KinectFusion maakt ook gebruikt van deze methode. Tot slot worden ook (5) Shear Warp en Texture Based rendering besproken. Het algemeen beeld van deze technieken werd gevormd door (Corp & Systems, 1995; Gross & Pfister, 2007).

Volumetric ray casting is een zeer bekende techniek om digitale data weer te geven. Het is ook gekend als een trage methode. Maar door middel van nieuwe implementatie technieken zoals GPU implementaties zorgen ze voor een veel snellere rendering door middel van ray casting (Marsalek, Hauber, & Slusallek, 2008). Dankzij deze vooruitgang wordt het interessant om voxels te gebruiken.

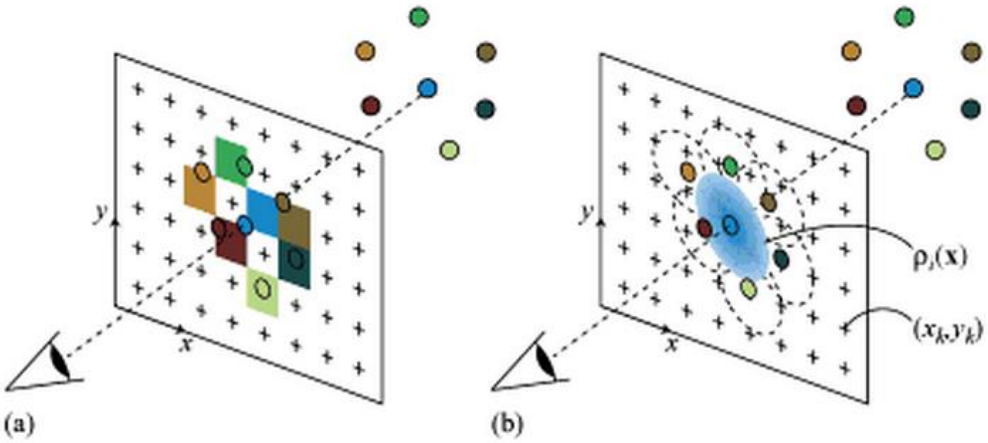


Figuur 27: Volumetric ray casting; Het zoeken van de juiste kleur. Afbeelding via http://blog.micfort.org/2012/07/ray-marching.html

De werking van het algoritme is gelijk aan de werking bij polygonen (Gross & Pfister, 2007; Pfister, Hardenbergh, Knittel, Lauer, & Seiler, 1999). Ray casting is gebaseerd op schermpixels in tegenstelling tot een 3D omgeving. Per pixel op het scherm wordt een straal afgeschoten doorheen de volume data (Figuur 27.1). Op deze straal worden dan punten gedefinieerd die de puntenwolk gaan voorstellen (Figuur 27.2). Omdat de kans echter klein is om een bepaald punt te raken wordt hiervoor gekeken naar dichtbij zijnde punten van de straal. Aan de hand van deze punten wordt dan de kleur bepaald voor de eerder bepaalde punten (Figuur 27.3). Tot slot wordt er gekeken welk punt zich het dichtste bij de camera bevindt en dus de kleur bepaalt. Omdat hiervoor veel punten moeten worden overlopen, is deze techniek niet optimaal. Er zijn echter veel optimalisaties voor ray casting in volumes. Voor meer informatie over enkele van deze optimalisatie technieken wordt er verwezen naar (Knoll et al., 2009; Kr, n.d.; Pfister et al., 1999).

Indien deze data wordt opgeslagen in bijvoorbeeld een boom structuur blijft door deze techniek het LOD effect behouden. Omdat de octree meer detail geeft naar mate er dieper in de structuur wordt gegaan, kan er worden gekozen op welke diepte de keuze moet worden gemaakt om de kleur van een bepaalde voxel te bepalen. Deze methode zorgt er dus voor dat ray casting veel sneller kan werken. Daarnaast heeft een boomstructuur nog enkele voordelen voor ray casting, aangezien de volgende cel, die wordt geraakt, altijd een kind is van dezelfde ouder tenzij er naar een hoger niveau wordt gegaan. Dit zorgt voor veel mogelijkheden voor real-time rendering van deze data. Zo kwam Euclideon in 2012 met een real-time rendering voor 3D omgevingen genaamd “Unlimited Detail Engine”. Door middel van Sparse Voxel octree’s en een krachtig zoekalgoritme zorgen zij voor een real-time rendering van grote 3D omgevingen.

Een groter probleem van deze techniek is echter dat een straal niet steeds recht door een punt gaat maar eerder tussen meerdere punten door gaat. Bij polygonen raakt een straal altijd een polygoon of hij mist deze. In beide gevallen is het duidelijk welke kleur het geraakt object heeft. Bij een puntenwolk is het echter niet altijd duidelijk welk punt wordt geraakt door een straal. Om dit probleem op te lossen worden punten geïnterpoleerd waardoor het algoritme de juiste kleur kan tonen op het scherm. Figuur 28 toont hoe dit algoritme in zijn werk gaat, zodat punten toch worden weergegeven als een vlak.



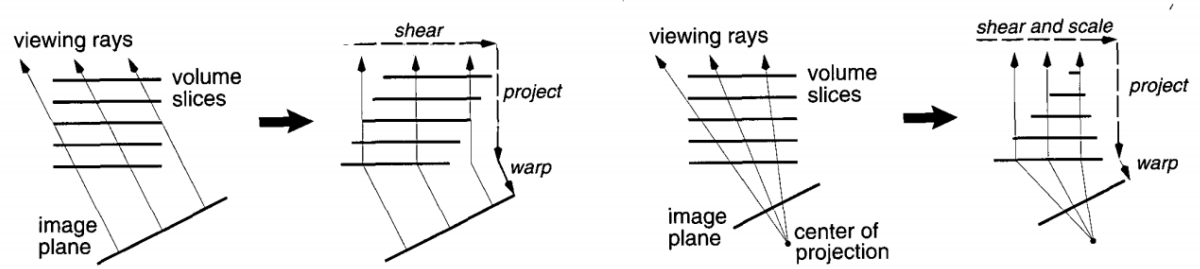
Figuur 28: (a) Naïeve forward projection techniek, (b) splatting voor het renderen van puntenwolken. (Gross & Pfister, 2007)

Bij ray casting wordt er een straal afgeschoten doorheen elk pixel van het beeld en vervolgens door de data. Hierbij wordt dan gekeken naar het dichtstbijzijnde punt voor het bepalen van de kleur van de pixel. Indien er geen pixel dichtbij ligt, is het mogelijk dat er gaten komen tijdens de rendering van het object. *Splatting* (Zwicker, Pfister, van Baar, & Gross, 2001) lost dit probleem op door punten een emissie van kleur te geven. Omdat er hierdoor geen directe diepte kan worden weergegeven op de ray, wordt er gebruik gemaakt van een z-buffer om te bepalen welke object het dichtst bij de camera ligt.

Splatting werkt door punten te omgeven met een kleuren emissie. De vergelijking kan worden gemaakt met paintballen te schieten tegen een muur. Hierdoor heeft elk punt een invloed op het nabije gebied. Dit zorgt ervoor dat er geen gaten zijn indien een ray door een bepaalde pixel geen punt zou raken. Daarnaast wordt ook het probleem opgelost wanneer meerdere punten terecht zouden komen op één pixel. Om de emissie van deze punten te bepalen wordt er gebruik gemaakt van het zogenaamde “elliptical weighted average”, ook wel EWA genoemd. Deze functie zorgt voor de kracht van de kleuren emissie. Omdat alle kleuren worden bepaald door splatting in de volgorde van de z-buffer is er dus ook zekerheid dat de juiste kleur wordt getoond op het scherm. (Zwicker et al., 2001) geeft de wiskundige achtergrond voor het gebruik van splatting. Door middel van het gebruik van deze kleuren is er ook geen nood meer aan interpolatie tussen de verschillende kleuren. Deze realistische benadering van de kleurenverdeling op een object zorgt ervoor dat objecten met minder punten in het model nog steeds een correcte weergave kennen op het scherm.

Dankzij de efficiëntie van het algoritme en de grote kwalitatieve voorsprong op ray casting, is splatting het algoritme bij uitstek voor het renderen van puntenwolken. Zo gebruikt ook Kinect Fusion deze techniek om in real-time beelden te kunnen weergeven (zie sectie 3.2.1).

De vorige algoritmen zoals ray casting en splatting zijn puur opgebouwd om straal gebaseerd door een puntenwolk te gaan en hierdoor intersecties te vinden met punten of voxels. Het *Shear Warp* algoritme probeert de datastructuur van een voxelgrid te gebruiken. In Figuur 29 zien we op de linkse afbeelding hoe stralen doorheen een 2D verzameling zouden gaan bij een orthogonale representatie. Shearing probeert er nu voor te zorgen dat de stralen die worden getrokken zich recht doorheen de data verplaatsen. Op die manier kan voor elke rij punten of voxels worden gekeken of er gevulde pixels zijn op plaatsen waar eerder nog geen punt werd



Figuur 29: Shear warping van (l) orthogonale projectie, (r) perspectieve projectie. (Lacroute & Levoy, 1994)

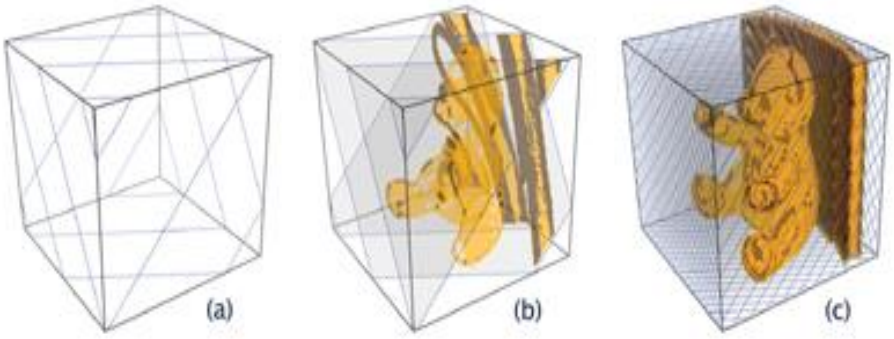
gevonden. Indien dit het geval is kan dit punt worden getekend. Anders zit er hopelijk in een volgende rij een gekleurde voxel. Om dit te doen moeten alle rijen worden verschoven zodat de stralen in plaats van schuin op de dataset, hier recht doorheen bewegen. Dit effect zien we op de 2de afbeelding terugkomen.

Deze techniek werkt zeer goed voor orthogonale camera's. Maar indien we een perspectief projectie gebruiken zien we dat stralen zich verder van elkaar bevinden hoe verder te straal wordt getrokken. Om dit probleem op te lossen moeten de rijen van voxels of punten worden geschaald, zodat de stralen vanuit het oogpunt nog steeds recht doorheen de dataset kan bewegen. Dit concept wordt weergegeven in de 4de afbeelding.

Deze manier van werken is zeer snel. Indien we de kijkhoek van de camera ten opzichte van de data kennen, moeten we enkel de rijen verschuiven en schalen. Eenmaal dit gedaan is, kan er zeer eenvoudig door de rijen worden gelopen om punten of voxels te vinden. Helaas heeft deze techniek, net zoals ray casting, een verlies op het gebied van kwaliteit. Dit komt doordat er weer met een pixelgrid wordt gewerkt waarop alles wordt getekend. Bij splatting werd hiervoor gebruik gemaakt van emissiestralen die kleuren kunnen overbrengen naar andere cellen in de grid. Ondanks deze opoffering van kwaliteit blijft het een goede keuze om shearing te gebruiken om het weergeven van puntenwolken. Het is nog steeds de snelste methode om 3D puntenwolken weer te geven (Sweeney & Mueller, n.d.). Al moet wel alle data worden ingeladen in het geheugen waardoor het moeilijk wordt om zeer grote datasets weer te geven met behulp van dit algoritme. Hiervoor zijn wel streaming-algoritmen voor gevonden, al blijft het nog steeds een beperking op te werksnelheid ten gevolge van de overhead bij het streamen.

Al de vorige vermelde algoritmen zijn in principe niet ondersteund op een GPU. Al kan een GPU (of GPGPU) wel worden gebruikt om deze technieken uit te voeren. *Texture based volume rendering* probeert gebruik te maken van de ingebouwde ondersteuning voor textures die zich bevindt in grafische kaarten (Kr, n.d.; Lamar & Joy, n.d.). Deze methode steunt op het feit dat grafische kaarten snel zijn in het mappen van textures op een bepaald 2D of 3D object.

De verkregen puntenwolk wordt opgedeeld in meerdere evenwijdige slices. Elk van deze slices stelt een texture voor die door een GPU kan worden geprojecteerd op polygonen. Deze polygonen hebben de afmeting van de slices die worden gegenereerd door het eerste algoritme. Indien al deze texturen achter elkaar worden geplaatst en er dan tussen wordt



Figuur 30: Texture based volume rendering. (http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html)

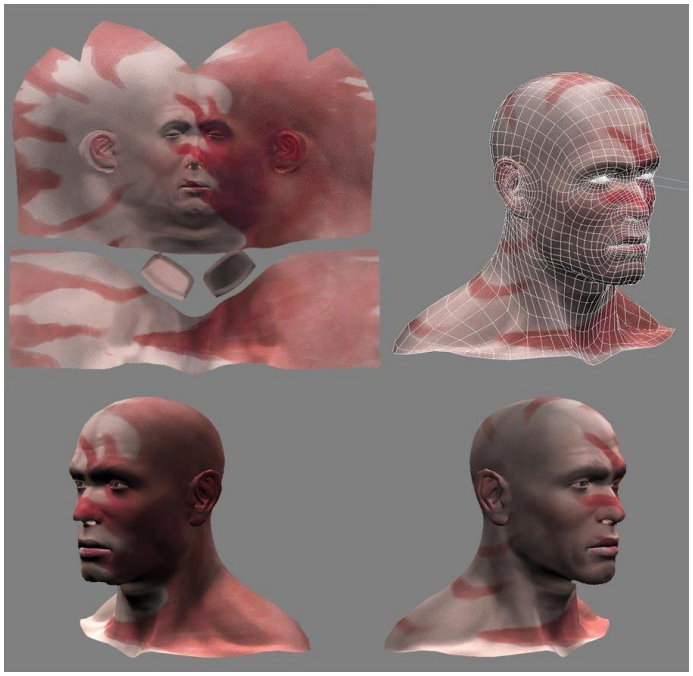
geïnterpoleerd, wordt er een 2D weergave gecreëerd van de 3D puntenwolk. In de bovenstaande afbeelding wordt dit principe duidelijk gemaakt.

Het grote nadeel van deze techniek is de nood om veel geheugen aangezien de hele puntenwolk moet worden ingelezen. Daarnaast is het belangrijk op te merken dat er veel slices nodig zijn om een goede weergave van het object te krijgen. Dit is duidelijk tussen (a) en (b) van bovenstaande figuur. Door het gebruik van weinig slices is het onmogelijk een volledig object te reconstrueren. Iets wat ook direct opvalt, is dat de achterste slices worden gemaakt ondanks deze niet altijd nodig zijn. Zo kan het zijn dat de voorste slices deze al volledig bedekken. Ook al zijn er hier voor enkele optimalisatietechnieken, toch blijft het een probleem voor zeer grote datasets, omdat het berekenen van de slices dan toch zwaarder begint te worden voor de computer om in real-time te berekenen.

4.1.3 Texturing point cloud based data

Indien een model wordt gerenderd, is het nodig dat deze ook volledig wordt ingekleurd of voorzien van enige vorm van visuele informatie. Deze stap wordt standaard gezien als texturing. Texturing bestaat uit het proces dat een bepaalde afbeelding plaatst op een polygoon. Indien dit voor alle verschillende polygonen van een object wordt gedaan verkrijgt met een volwaardig model. In het kort kan texturing worden gezien als het transleren en roteren van een bepaalde afbeelding zodat deze op een polygoon past. In de realiteit wordt er vaak gebruik gemaakt van een bepaalde afbeelding per object. Aan de hand van deze 2D afbeelding wordt een 3D oppervlakte ingekleurd. Hierdoor is het mogelijk dat de 2D afbeeldingen er raar uitzien. Dit is het gevolg van het schalen van de afbeelding zodat deze op de betreffende polygoon past. Hiervoor wordt iedere polygoon loodrecht geprojecteerd op de afbeelding. Dergelijke afbeelding en texturing staat afgebeeld in Figuur 31.

In de korte bespreking van texturing wordt echter snel duidelijk dat er steeds een afbeelding nodig is die loodrecht geprojecteerd wordt vanuit een polygoon. De data waarmee wordt gewerkt doorheen de tekst is echter een puntenwolk. Hierdoor zitten er geen polygonen in het oorspronkelijk model en dus ook geen textures. In feite moet er een omgekeerde beweging worden gemaakt. Aan de hand van de puntenwolken moeten textures worden aangemaakt zodat deze over een eventuele polygon representatie worden geplaatst.



Figuur 31: Links bovenaan bevindt zich een texture map voor het rechtse karakter. Aan de hand van de zichtbare polygonen wordt de texture op de mesh geplaatst. Onderaan het resultaat. (Afbeelding via www.deviantart.net)

MehLab biedt een eenvoudige methode voor het generen van kleuren data voor een mesh. Hierbij wordt vanuit een puntenwolk eerst een mesh aangemaakt. Er wordt gebruik gemaakt van Surface Reconstruction algoritmen zoals uitgelegd in sectie 3.2. Het model dat wordt aangemaakt door middel van het Surface Reconstructie algoritmen wordt hierna ingekleurd aan de hand van vertex coloring. Iedere vertex wordt individueel ingekleurd waardoor er toch kleurwaarden verschijnen in het uiteindelijk model. Er gaat echter veel detail verloren tijdens deze stap. Zo zijn sommige punten niet bevat door een polygoon omdat het reconstructie algoritme een minder dense puntenwolk genereert op de bepaalde plaatsen. Het verlies van dergelijke data kan ervoor zorgen dat bepaalde elementen van een karakter niet meer zichtbaar zijn. Dit probleem valt niet triviaal op te lossen. Het is namelijk nodig dat voor elke frame die is gecapteerd de RGB afbeelding ook wordt bijgehouden. Die moet dan worden gemapped op de polygonen die zichtbaar zijn. Omdat die echter niet het volledig model zou kunnen omvatten moet doorheen de tijd de texture worden aangepast. Daarnaast is het ook nodig om bij elk frame opnieuw een texture te generen aangezien de puntenwolken andere bewegingen maken dan polygonen. Dit zorgt ervoor dat bepaalde texturen feller uiteengerukt worden dan in andere gevallen. Omdat er nog steeds geen correcte oplossing bestaat voor het verwerken van dit probleem, zal er niet dieper worden op ingegaan. Er zijn reeds oplossingen die voor een statische scene werken. Hiervoor kan er worden gekeken naar een diepere bespreking van KinectFusion. Voor dynamische scenes is er echter nog geen consistente oplossing die een degelijk resultaat aflevert.

4.2 Networking

Binnen het concept waarbij 3D modellen worden weergegeven op AR-brillen zijn er meerdere vereisten voor het versturen van data over een netwerk. Eerst en vooral moet de data voor de scene worden ingeladen indien deze niet op het toestel zelf kan staan. Vervolgens moet de

gebruiker in staat zijn deze data te kunnen delen met naburige mensen om zo multiplayer games toe te laten op dit concept. Ondanks het feit dat deze communicatie relatief gelijk blijft voor zowel een set-up met een centrale server als een lokale verwerking op het AR-toestel, wordt er in dit hoofdstuk vanuit gegaan dat de verwerking gebeurt op het AR-toestel zelf.

Indien er data over een netwerk wordt gestuurd is het belangrijk dat dit zo efficiënt mogelijk wordt gedaan. Binnen deze thesis is er namelijk sprake van 3D modellen die worden gegenereerd door middel van puntenwolken. Het is zeer belangrijk op te merken dat de puntenwolken onafhankelijk van elkaar zijn en er bij een octree-weergave (zie sectie 3.4.2) nood is aan een volledige puntenwolk per frame. Ook bij polygon-weergave is het nodig het volledige model te krijgen na een beweging van het object omdat er geen referentiepunten liggen in een puntenwolk. Dit probleem kan worden opgelost door gebruik te maken van Skeleton Mapping, de techniek die is voorgesteld in sectie 3.4.3.

Indien er gebruik wordt gemaakt van een representatietechniek die geen referentiepunten heeft voor het animeren van de data, is het nodig om van elke frame de volledige dataset te verkrijgen, of alle aanpassingen door te sturen. Dit vergt enorm veel bandbreedte voor de applicatie. Indien we echter een referentieskelet kunnen opbouwen volstaat het om de referentiepunten van het skelet door te sturen eenmaal we de data rond het skelet hebben. Op deze manier moet enkel in de eerste frame van een bepaald object het volledige object worden meegestuurd. Op alle volgende frames volstaat het om enkel de nieuwe plaatsen van een referentiepunt mee te delen. Ieder AR-toestel dat deze data ontvangt kan dan zelf de nieuwe staat van het skelet berekenen en hierop alle punten (of polygonen) plaatsen.

Voor het aanmaken van deze skeletten is het belangrijk te weten welke objecten worden ondersteund. Deze thesis kijkt in het specifiek naar het animeren van personages. Daardoor moeten we focussen op het aanmaken van skeletten voor menselijke karakters. De techniek voor het achterhalen van een skelet uit een beeld noemt met Skeleton Tracking (Gall et al., 2009). Door het skelet te gebruiken voor animaties zoals in deze paper wordt voorgesteld, is het eenvoudig om opeenvolgende frames aan elkaar te linken.

4.3 Animation

Het laatste punt van de pipeline dat we behandelen is animatie. Animatie is het bewegen van objecten in een digitale omgeving. Omdat dit een zeer groot onderzoeksdomein is met enorm veel beschikbare informatie, beperkt deze bespreking zich tot de animatietechnieken die nuttig zijn voor de toepassing in computergames in samenwerking met de 3D modellen. Hierbij geven we aandacht aan de skelet-structuren en hoe deze gemanipuleerd kunnen worden. Het eerste wat we hiervoor moeten bekijken is hoe een skelet kan worden weergeven. (Rick, 2012) dient als basis voor de besprekingen in dit hoofdstuk. Eerst wordt een representatie methode van het skelet besproken in sectie 4.3.1. Daarna worden twee methodes besproken voor het animeren van dergelijk skelet. Hierbij kijken we naar forward kinematics en inverse kinematics, terug te vinden in sectie 4.3.2.

4.3.1 Skelet representative

Om een skelet voor te stellen zijn er twee noodzakelijk elementen, de beenderen en de gewrichten. Vanaf hier zullen we deze links en joints noemen. Links zijn een voorstelling van een bot en hebben een bepaalde lengte. Joints hebben een bepaalde draaihoek. Het is hierbij belangrijk om op te merken dat bepaalde representatie technieken vanzelf een joint plaatsen op het einde van een link. Dit zorgt ervoor dat er enkel een opeenvolging van links bestaan die het skelet voorstellen. Dit hoofdstuk zal twee representatie technieken laten zien: Denavit-Hartenberg en Kinect Skeleton Structure. De eerste techniek is een algemeen aanvaarde standaard voor het eenvoudig weergeven van een skelet door middel van joints en links. De tweede techniek gebruikt niet rechtstreeks joints en links voor het bepalen van het skelet maar punten die een bepaalde joint voorstellen. Deze kunnen worden verbonden om zo over te stappen naar een Denavit-Hartenberg representatie. Het is belangrijk op te merken dat beide technieken gebruik maken van een hiërarchische boom om het skelet op te slaan. Dit wil zeggen dat er een bepaalde root-node is waarvan het skelet vertrekt.

Het eerste representatie model is het *Denavit-Hartenberg* representatie model. Dit model maakt gebruik van zowel joints als links om een skelet op te bouwen. Het is hierbij belangrijk om op te merken dat joints slechts kunnen bewegen in één dimensie. Indien er gewrichten in het lichaam zijn die over meerdere dimensies kunnen bewegen, dan worden deze voorgesteld door gebruik te maken van meerdere joints waartussen een link van lengte 0 wordt geplaatst. Bij de boomvorm die wordt gebruikt bij Denavit-Hartenberg kunnen meerdere links vertrekken vanuit één joint, maar iedere link heeft exact één joint. Er zijn 4 waarden die de relatie tussen twee joints bepalen. De eerste waarde is de link offset, dit is de waarde voor de verschuiving van de positie over de z-as van het assenstelsel. Een tweede, gerelateerde waarde, is de link lengte. Deze geeft aan hoever de positie is verschoven over de x-as. Hier moet worden opgemerkt dat er geen verschuiving mogelijk is over de y as aangezien iedere joint slechts in één dimensie mag roteren. Deze rotatie hoek wordt aangeduid door twee

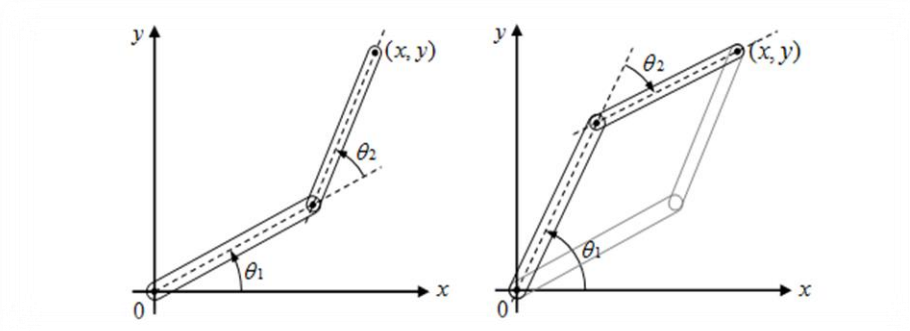
Figuur 32: Voorbeeld van een skelet in computer graphics. (Zheng et al., 2009)

overgebleven waarden, deze stellen elk een hoek voor om de vector te bepalen. De eerste hoek is de joint angle wat de hoek over de x-as aanduidt. De tweede hoek, die over de z-as van het assenstelsel gaat, noemt men de link twist. Deze vier waarden zorgen ervoor dat iedere joint eenvoudig kan worden bepaald ten opzichte van de vorige joint in de skelet-tree (de boomstructuur die het skelet bepaald). Bij inverse kinematics zal duidelijk worden waarom dit een handige representatievorm is om een digitaal skelet op te slaan.

Een ander representatie model is *Kinect Skeleton Structure*, dit model wordt door de Kinect gebruikt om skelet-data op te slaan. De structuur van de Kinect heeft andere eigenschappen als de voorgaand vermelde Denavit-Hartenberg representatie. De belangrijkste eigenschap is dat bij de Kinect er geen relatie word opgeslagen tussen verschillende joints in termen van rotaties. De skeletten worden namelijk voorgesteld aan de hand van joint-posities en joint-rotaties. De posities stellen één exact punt voor en dus geen volledige link. Een link kan worden gereconstrueerd door de joint te verbinden met de child- of parent-nodes van een bepaalde joint. De rotaties van het skelet worden op twee verschillende manieren gebruikt bij de Kinect. Er wordt een lijst bijgehouden van alle rotaties en deze lijst bevat één rotatie per joint. De voorstelling hiervan gebeurt aan de hand van globale quaternions, rotaties die onafhankelijk zijn van de parent-node. De andere rotatie is één die kan worden berekend aan de hand van de gekende joint-posities. Het is belangrijk op te merken dat beide rotaties echter verschillend zijn. De quaternion representatie voor de rotaties houdt niet enkel rekening met de hoek ten opzichte van de Y-as in het assenstelsel, maar ook met de rotatie van de arm. Ook op het gebied van hoeken zijn deze verschillend omdat ze op een verschillende manier worden berekend. De quaternion representatie wordt berekend onafhankelijk van de joint-posities. Hierdoor kunnen er afwijkingen ontstaan tussen het visuele skelet en de berekende quaternion.

4.3.2 Kinematics

Een eerste mogelijkheid om bewegingen te animeren door middel van een computer is het verplaatsen van individuele joints en hun parameters. Deze benadering wordt *forward kinematics* genoemd. Indien we bijvoorbeeld twee joints hebben met 2 links en we proberen deze zo te plaatsen dat een bepaald object wordt geraakt door de laatste link, zullen we voor deze iedere joint vanaf het vaste punt bewegen om een oplossing te krijgen. Het wordt echter snel duidelijk dat er twee mogelijkheden kunnen zijn om dezelfde oplossing te geven (zie Figuur 33). Ook is het moeilijk een bepaalde positie te bereiken omdat de positie van de eerste joint al bepaalt of er al dan niet een oplossing mogelijk is. Het is daarom ook niet handig om



Figuur 33: Forward kinematics twee mogelijke oplossingen. (www.robotis.com)

deze methode te gebruiken om real-time een oplossing te voorzien. Hierbij zou er brute-force moeten worden gezocht naar een oplossing voor de gekozen scene. Deze manier van werken is echter wel handig om vooraf bepaalde animaties precies te bepalen. Het is aan te raden gebruik te maken van forward kinematics indien er in een game een bepaalde sequentie moet zijn voor het karakter dat altijd identiek moet zijn en perfect moet zijn.

In tegenstelling tot wat forwards kinematics doet, waarbij de veranderingen in de joints worden aangebracht om een doeltoestand te bereiken, wordt er bij *inverse kinematics* een eindtoestand meegegeven. Deze eindtoestand zal worden berekend zonder individuele joints te moeten aanpassen. Dit zorgt ervoor dat 3D artiesten of 3D animatoren geen waarden in de joints zelf moeten aanpassen maar enkel de punten waar joints moeten staan, hoeven aan te duiden. De algemene werking van inverse kinematics werkt als volgt; Er wordt een bepaald punt gekozen al dan niet binnen het bereikbare domein. Indien het gekozen punten buiten het bereikbare domein ligt zal het dichtst mogelijke punt bij het bereikbare punt van de joints worden gekozen. Het bereikbare gebied beslaat in deze context alle mogelijke punten waar de laatste joint, de end-effector, kan plaatsnemen zonder de beperkingen van de joints te verbreken. De volgende stap is het berekenen van de eindposities van de joints in het skelet. Hierbij moet er voornamelijk aandacht worden besteedt aan het vergelijken van meerdere oplossingen. Traditioneel is de beste optie om de oplossing te kiezen waarvoor de verandering in de hoek per joint het kleinste is. Tot slot is er interpolatie mogelijk tussen de huidige configuratie van het skelet en de configuratie die is berekend in stap twee.

De hierboven beschreven aanpak geeft echter geen directe oplossing voor het berekenen van de posities, of hoe de interpolatie precies het beste verloopt. In enkele oplossingen worden deze twee stappen namelijk samen gedaan. Deze methoden worden iteratieve methoden genoemd. Bij iteratieve methoden wordt op elke stap hetzelfde algoritme toegepast wat zo een goede oplossing geeft. Een mogelijk algoritme dat op deze manier het probleem kan oplossen is een algoritme dat bij iedere stap de end-effector herpositioneert zodat deze naar het doelpunt beweegt. De volgende joint wordt dan gepositioneerd naar het eindpunt van zijn link. Dit algoritme kan worden afgelopen tot aan de root of een bepaalde diepte. Na enkele iteraties zal het algoritme zijn uiteindelijk doel bereiken.

Een andere manier is het gebruiken van een mathematische aanpak. Hiervoor kan de “jacobiaan” worden gebruikt. De “jacobiaan” is een matrix van afgeleiden. Deze matrix kan worden vermenigvuldigd met de posities en krachtvectoren van alle joints die moeten worden aangepast. Het eindresultaat van deze vermenigvuldiging geeft een matrix met nieuwe hoeken voor de joints. Voor een verdere uitwerking van dit concept wordt verwezen naar het boek (Rick, 2012).

4.3.3 Motion capture

Bij de twee eerder vermelde technieken is het soms omslachtig om een volwaardig animatie te creëren. De designer moet voor het gebruik van inverse kinematics namelijk enkele punten plaatsen waar het skelet langs moet bewegen om zo een volledige animatie te bekomen. Voor forward kinematics is het zelfs nog moeilijker aangezien elke joint zelf moet worden geroteerd tot de juiste rotatie. Om dit op te lossen wordt deze data direct gecapteerd vanuit de werkelijkheid (Rick, 2012). Deze techniek wordt Motion Capture (MOCAP) of Performance

Capture genoemd en is sinds de jaren ’90 niet meer uit het ontwikkelen van games weg te denken. (Moeslund & Granum, 2001) biedt een overzicht aan van de volledige pipeline die wordt gebruikt bij MOCAP. Hierbij wordt vooral aandacht besteed aan de verschillende elementen van het proces, namelijk de zogenaamde initialisatie stap, het capteren van de data, het zoeken van de werkelijke pose en uiteindelijk de volledige herkenning van het skelet.

De initialisatie fase wordt gebruikt voor het afstellen van het systeem. Bij MOCAP wordt er gebruik gemaakt van speciale detectie systemen voor posities te kunnen bepalen. Dit is duidelijk zichtbaar in Figuur 34 waar de drie acteurs een speciaal pak dragen. Hierbij is ook direct zichtbaar dat een gewoon persoon niet wordt geregistreerd aangezien de man zonder pak niet is terug te vinden in de resultaten. Het detectie systeem gebruikt meerdere camera’s rondom een bepaalde zone. Tijdens de initialisatie stap worden deze camera’s afgesteld op elkaar zodat punten correct kunnen worden geplaatst in een 3D scene. Dankzij deze initialisatie stap kunnen de verschillen toestellen zeer precies worden afgesteld waardoor er een zeer hoge precisie kan worden gehaald (voor bepaalde opstellingen tot kleiner als een tiende van een millimeter).

Figuur 34: Motion Capture shoot waarbij bovenaan het werkelijke scenario wordt gespeeld en onderaan het uiteindelijke resultaat wordt getoond. (Afbeelding van de ontwikkeling van Star Citizen - www.robertsspaceindustries.com)

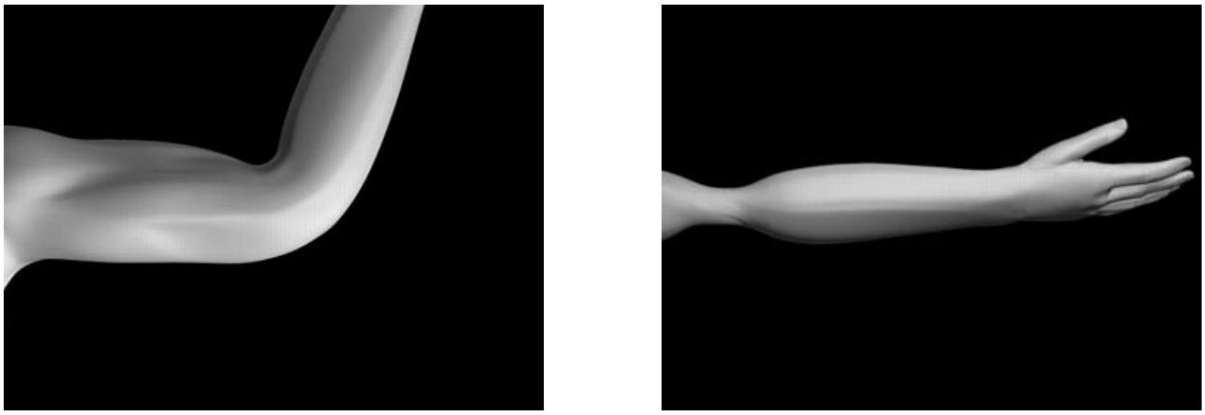
De tweede stapt voor het capteren van de data werkt vergelijkbaar met andere captatie technieken. Enkel wordt hier over een actieve captatie techniek gesproken. Bij actieve captatie, vermeldt in sectie 3.1.2, maakt de camera zelf signalen/lichtstralen of andere aan waarmee posities worden bepaald. Bij actieve captatie worden er bepaalde sensoren or emitters geplaatst op het te volgen model. Bij MOCAP wordt er gebruik gemaakt van een speciaal pak dat wordt uitgerust met kleine infrarood-emitters. Het pak dat de acteurs aanhebben stuurt namelijk zelf lichtsignalen om er zo voor te zorgen dat het enorm accuraat kan worden geplaatst in een 3D omgeving. De verbetering ten opzichte van passieve captatie zit hem vooral in het feit dat vanuit verschillende posities exact dezelfde positie wordt gecapteerd, namelijk die van het actieve element. Bij passieve technieken worden vaak licht verschillende posities bekeken waardoor het moeilijk is om een plaats een hoog accurate positie te geven. Deze techniek is echter zéér duur voor het vastleggen van bewegingen. Zo moet er een volledige zaal worden gebruikt die is uitgerust met speciale camera's voor het (infrarood)licht op te vangen. Naast de camera uitrusting zijn er ook nog de dure actieve pakken die moeten worden gedragen door de acteurs. Deze pakken hebben echter ook nog een nadeel omdat niet alle bewegingen mogelijk zijn dankzij de actieve elementen op het pak. Dit kan ervoor zorgen dat bewegingen soms artificieel overkomen door de speciale houding van de acteurs.

De twee volgende stappen behoren volledig tot skelet herkenning. Tijdens deze stappen wordt een digitale reconstructie gemaakt van het skelet aan de hand van de gemeten posities van de nodes. Deze nodes kregen namelijk in de eerste stap een referentie skelet mee zodat dit eenvoudig mee kan bewegen. Door gebruik te maken van camera's die in 360° rondt het karakter hangen, is het eenvoudig een zeer accurate plaatsbepaling te maken van de nodes alsook hier een zeer goede kwaliteit van skelet uit te halen. Het uiteindelijke resultaat van een MOCAP shoot is een gedigitaliseerde animatie van de opgenomen data. Dit maakt het zeer eenvoudig voor gameontwikkelaars om snel een complexe animatie op te bouwen.

4.3.4 Data animation

Het animeren van een bepaald object is een belangrijk iets voor het genereren van een realistisch model. In de eerder vermelde subsecties worden enkele manieren besproken om het skelet van een karakter te animeren. Dit is de basis van het volledige verhaal rond karakteranimatie. Het skelet wordt bepaald aan de hand van bepaalde input data (zoals bij MOCAP shoots). De posities van het weergegeven karakter worden hieraan gelijkgesteld en een geanimeerd karakter is zichtbaar. Hoewel dit eenvoudig klinkt, wordt er een grote stap overgeslagen. Zo is het nodig om de data die rondom het skelet zit ook te animeren. Deze data is namelijk de visuele representatie van het skelet in 3D. Zoals doorheen de volledige tekst wordt vermeld, zijn er twee verschillende methoden voor het weergeven van 3D data. Namelijk door middel van punten of door middel van polygonen. Beide mogelijkheden zullen kort worden aangehaald in deze sectie.

Point cloud animation is een recente ontwikkeling in het gebruik van puntenwolken. Tot voor kort was het gebruik van puntenwolken voornamelijk voor het generen van een hoge kwaliteit representatie van een statisch model. Hierbij wordt er gerefereerd naar de Da Vinci sculpturen die dienen als de voorbeelden voor 3D point graphics. Recent is de computer echter krachtiger



Figuur 35: Joint deformation zonder blend-weights. (Lewis et al., 2000)

geworden. Door de vooruitgang in performantie is het realistisch geworden om puntenwolken te animeren. Het animeren van dergelijke datasets is echter niet triviaal. Er is namelijk geen samenhorigheid tussen bepaalde punten in een scene. Daarom worden er eenvoudige aanpakken gebruikt om alsnog een model te kunnen animeren. Deze aanpak rust op de datastructuren waarin puntenwolken worden opgeslagen. Dit zijn namelijk boomstructuren. Indien een puntenwolk geanimeerd wordt, dan verandert de volledige structuur van dergelijke bomen. Daarom worden tijdens de animatie van puntenwolken meestal volledige deelbomen verplaatst. Hierdoor is het minder rekenintensief voor een computer om alle punten te verplaatsen. Het nadeel aan deze technieken is echter dat er geen directe controle is over bepaalde punten in de dataset. Het is echter ook mogelijk om bepaalde groepen van punten aan elkaar te binden door lokale groepen te definiëren. Ook tijdens de implementatie wordt een dergelijke techniek gebruikt om een puntenwolk te animeren. Voor meer informatie rond dergelijke technieken wordt er verwezen naar de beschrijving van de implementatie in sectie 6.5.

Mesh animation is in tegenstelling tot het animeren van puntenwolken een reeds opgelost probleem. Vrijwel iedere 3D toepassing die gebruik maakt van een karakter dat kan bewegen gebruikt mesh animation. Het algemeen concept hierbij is dat meshes worden hervormt door vertexen te verplaatsen. De polygonen veranderen hierdoor ook van positie. Omdat niet iedere polygoon individueel wordt verplaatst, maar de deelelementen van een polygoon worden verplaatst, is het mogelijk om een gesloten object te behouden tijdens het uitrekken van bepaalde onderdelen.

Indien er sprake is van skelet-gebaseerde animatie wordt het iets complexer dan enkele vertexen te verplaatsen. Zo moet er rekening worden gehouden met de verplaatsen aan de hand van een bepaald lichaamsdeel van het skelet. Hierdoor worden de joints tussen de verschillende delen uitgetrokken of samengeduwd. De huid van de mens strekt zich uit en krimpt in rond joints. Dit is duidelijk zichtbaar aan bijvoorbeeld het ellebooggewricht. Indien een 3D karakter gewoon zou worden geanimeerd door joints te herpositioneren en te roteren zou de huid niet worden mee verplaatst. Om dit probleem op te lossen kan er gebruik worden gemaakt van zo genaamde blending technieken. De bespreking hiervan wordt gebaseerd op (Kavan, 2002; Lewis, Cordner, & Fong, 2000; Wang & Phillips, 2002)

Blending technieken zijn in staat om een polygoon, tijdens de verplaatsen, niet enkel te verplaatsen volgens het lichaamsdeel waartoe het behoort, maar ook rekening te houden met omliggende lichaamsdelen. Aan de hand van zogenaamde blending-weights kan men aanduiden hoeveel invloed een bepaalde joint uitoefent op een polygoon (of verzameling van polygonen). Indien er geen gebruik wordt gemaakt van dergelijke technieken, worden joints op een niet natuurlijke manier weergegeven. Dit kan worden gezien in Figuur 35. De linkse afbeelding toont hier al dan dat de joint lichtjes naar binnen plooit. In de rechtse afbeelding is het duidelijk dat dankzij de animatie de joint zeer fel is vervormd. Deze problemen zijn het gevolg van het mappen van vertexen. Indien elke vertex wordt mee geroteerd, en hierbij de verhoudingen tussen de vertexen behouden blijft, zal er niet voldoende oppervlakte zijn voor de correcte weergave. Hierdoor klappen bepaalde joints toe. Om dit op te lossen moeten blend-weights worden toegevoegd tijdens het animeren. Dit zorgt ervoor dat het resultaat veel realistischer is zonder gaten te creëren. Daarnaast worden ook eventuele ongewenste vervormingen vermeden. Het resultaat van het gebruik van weight blending kan worden teruggevonden in Figuur 36. Het is hier duidelijk dat het lichaam ervoor zorgt dat de punten die dicht bij het lichaam liggen niet fel mee bewegen indien de arm beweegt.

Figuur 36: Links, geen (of beperkte) skin animation. Rechts gebruik van blend weights voor het bepalen van skin animation. (Kavan, 2002)

5.Kinect Captatie

In de vorige hoofdstukken werd een algemene analyse van de huidige methodes voor 3D captatie en visualisatie gegeven. Vanaf dit hoofdstuk wordt er een implementatie gemaakt van het voorgestelde algoritme in sectie 1.4. Hierbij wordt een opsplitsing gemaakt in de captatie stap en de verdere verwerking van de gecapteerde data. Het capteren van (her)bruikbare data is een zeer belangrijk gegeven voor het algoritme. Niet enkel omdat data noodzakelijk is om puntenwolken te generen en om te kunnen vormen naar een mesh, maar ook omdat door het splitsen van captatie en verwerking er gebruik kan worden gemaakt van dezelfde data door verschillende algoritmen. Hierdoor kunnen verschillende technieken eenvoudig worden vergeleken om zo de beste optie te gebruiken bij het uitwerken van het algoritme. Zoals eerder vermeld in sectie 3.1.2 zijn er veel verschillende mogelijkheden om data te scannen. In dit hoofdstuk wordt de keuze die gemaakt werd voor de implementatie toegelicht. Dit gaat samen met een diepere uitleg op hardware niveau van enkele mogelijke opties.

5.1 Captatie hardware

Het capteren van data is een probleem dat men reeds een lange tijd probeert op de lossen. In het recent verleden zijn er verschillende toestellen ontwikkeld die de mens in staat stelt om niet enkel een 2D beeld te maken van de realiteit, maar ook een 3D beeld te capteren. Zoals aangegeven in sectie 3.1.2 zijn er enorm veel verschillende technieken beschikbaar om deze beelden te maken. Voor de implementatie van het algoritme waren er enkele limieten en vereisten waaraan moest worden voldaan. Zo moest het toestel in staat zijn om hoge kwaliteit 3D punten beelden te genereren. Hierbij was het ook belangrijk dat een volledige beeld in één keer werd geregistreerd. Een ander element dat wordt gebruikt in het Skeleton Mapping algoritme is de 3D skelet representatie. Aangezien het niet binnen de scope van de thesis valt om zowel een mapping algoritme als een skelet herkenning algoritme te schrijven, moest ook dit skelet worden geleverd door een systeem. Een directe implementatie in de hardware is hierbij de beste oplossing. Tot slot is het ook belangrijk dat meerdere scanningapparaten samen scenes kunnen registreren. Hierdoor vallen enkele methoden volledig weg omdat deze voor interferentie zorgen. Een belangrijke restrictie in de selectie van het toestel was ook prijs gebonden. Voor het concept aan te tonen is er geen nood aan het micrometer precieze dieptescanner. Een scanner die een goede accuraatheid heeft (om en bij de centimeter), is voldoende voor de gemaakte implementatie.

Er werd reeds snel geopteerd om te gaan voor de Microsoft Kinect. De Kinect is een low budget 3D dieptescanner ontwikkeld voor de console systemen van Microsoft. Sinds de eerste editie heeft de Kinect reeds een update gekregen. Deze nieuwe versie pakt enkele elementen volledig anders aan. Voor deze reden wordt er dan ook eerst een vergelijking gemaakt tussen beide systemen om zo de uiteindelijke keuze te beargumenteren. Uiteindelijk is er gekozen voor de Kinect V2 omdat deze een hogere kwaliteit aan dieptescans biedt. Daarnaast biedt de Kinect V2 net als de eerste Kinect ook een 3D skelet aan. Bij de tweede iteratie is deze echter

van een veel hoger niveau waardoor de rotaties en posities van alle skelet onderdelen betrouwbaarder zijn.

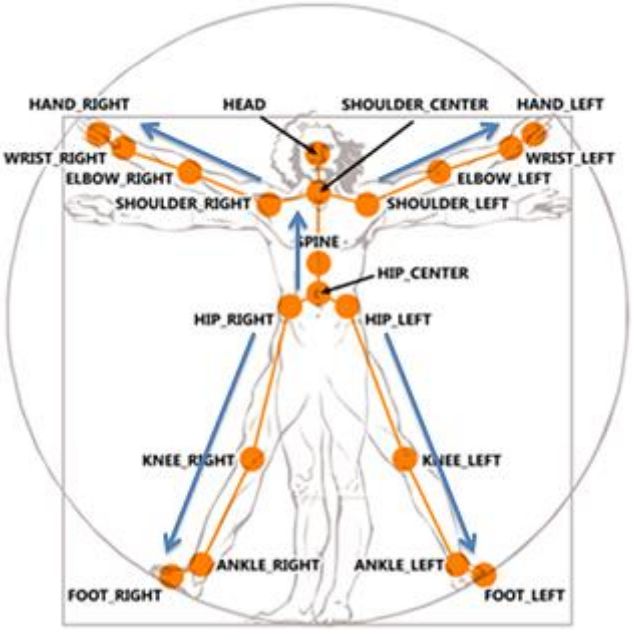
5.1.1 Kinect V1

De Kinect V1 is ontwikkeld voor het gebruik met de Xbox-360 van Microsoft. Deze camera laat de gebruikers toe op een nieuwe creatieve manier gebruik te maken van hun console. Om dit mogelijk te maken heeft Microsoft een goedkope, maar krachtige, dieptescanner op de markt gebracht die door middel van machine learning skeletten kan genereren. In deze sectie worden enkele algemene technieken van de Kinect aangehaald om zo de voor- en nadelen van het toestel duidelijk te maken. Voor een diepgaande bespreking van de Kinect V1 wordt verwezen naar (Ivan Tashev, 2013; Zhang, 2012).

Figuur 37: Kinect V1 voor Xbox-360 (www.xbox.com)

De Kinect V1 is het eerste low-cost scantoestel dat op de markt is gekomen. Hierdoor zijn er enkele problemen die niet volledig zijn aangepakt of op een verkeerde manier zijn opgelost. Ondanks deze problemen blijft het toestel zeer handig en nuttig voor het maken van diepte gerelateerde toepassingen. De Kinect V1 maakt gebruik van Structured Light voor dieptebepalingen te maken. Hierbij wordt, zoals in sectie 3.1.2 uitgelegd, gebruik gemaakt van patroon herkenning op, in infrarood verstuurde, patronen. Het voordeel van deze techniek is de efficiëntie en de accuraatheid van een sensor. Het grootste deel van de dieptebepaling wordt namelijk op softwareniveau afgehandeld. Het nadeel van deze toepassing is interferentie. Indien er meerdere toestellen worden gebruikt zullen patronen elkaar overlappen en zorgen voor fouten in het resultaat. Een oplossing die hiervoor is ontwikkeld, maakt gebruik van hoge frequentie trillingen waarbij iedere Kinect op een andere frequentie werkt. Door deze setup te gebruiken kan worden herkend bij welk patroon welke punten liggen. Ook al lost dit niet alles 100% op, toch zorgt dit ervoor dat meerdere Kinects samen gebruikt kunnen worden.

Naast de techniek die gebruikt wordt bij de Kinect is ook de kwaliteit van het resultaat belangrijk. De Kinect V1 beschikt over een kleinere resolutie in verband met dieptescans. De resolutie voor het scannen van dieptebeelden is voor de Kinect 320x240 pixels. De kleuren camera daarentegen maakt beelden van 640x480 pixels. In de volgende sectie wordt snel duidelijk dat de Kinect V2 hier veel hogere specificaties kan voorleggen en daarom ook een interessantere keuze is voor de gemaakte implementatie. Zoals vermeld in de inleiding over captatie hardware, is de Kinect V1 in staat om skeletten te genereren uit de puntenwolken die worden verkregen door de dieptescan, zie Figuur 38. Hierbij wordt een skelet gegenereerd met 20 joints die ieder hun eigen 3D posities en rotatie hebben. Dit is een zeer handig element van het toestel. Maar omdat de puntenwolk over niet al te veel punten beschikt, gezien de



Figuur 38: Skelet met 20 joints dat gevormd door de Kinect V1. (msdn.microsoft.com)

scan-resolutie, is de data van het skelet niet altijd even accuraat. Er zit dan ook redelijk wat ruis op de data die verkregen wordt door middel van de Kinect.

Omdat de Kinect V2 (zie volgende sectie) betere specificaties heeft dan de Kinect V1, wordt er niet te diep in gegaan op de exacte werking van het toestel. Een belangrijk aspect dat nog niet is vermeld, betreft de opslag van de data van de Kinect V1. Het is mogelijk om meerdere Kinect V1's aan te sluiten op een PC. Dit kan omdat de data die wordt verstuurd over de USB-aansluiting past binnen een USB2-bus en omdat iedere Kinect individueel aanspreekbaar is via de API die door Microsoft wordt geleverd. Iedere Kinect heeft namelijk een eigen voorbehouden gedeelte RAM waarin de recentste frames worden opgeslagen.

5.1.2 Kinect V2

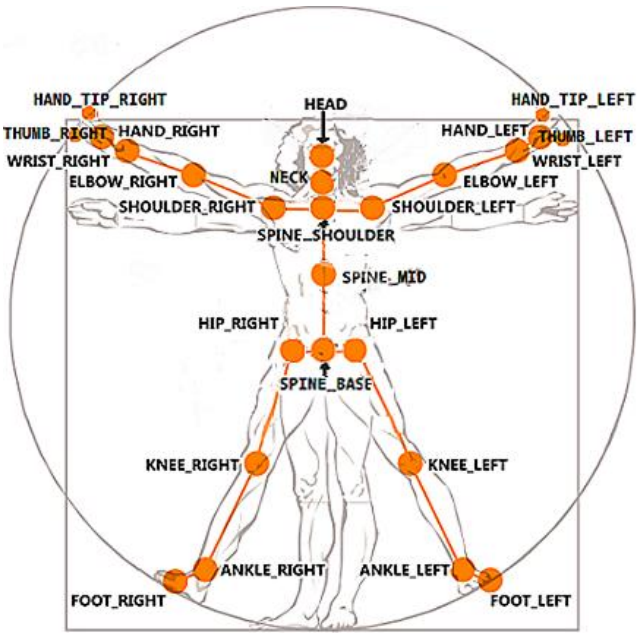
De Kinect V2 is een product, vergelijkbaar met de Kinect V1. Al zijn er wel enkele grote verschillen tussen het toestel dat is ontwikkeld voor de Xbox 360 en het toestel dat is ontwikkeld voor de Xbox One. In deze sectie wordt net zoals bij de Kinect V1 ingegaan op de verschillende elementen van de Kinect. Net zoals de Kinect V1 is het toestel ontwikkeld met games in het achterhoofd. Om die reden ligt de focus van de Kinect niet op de perfecte accuraatheid maar op het bieden van een eenvoudig systeem voor skeletherkenning en dieptemapping. Ondanks de focus zijn er toch enkele grote sprongen vooruit gemaakt tussen de twee verschillende versies.

De Kinect V2 werkt, in tegenstelling tot zijn voorganger, aan de hand van de Time of Flight methode die werd uitgelegd in sectie 3.1.2. Hierbij wordt er geen gebruik gemaakt van infraroodpatronen, maar worden afstandsmetingen gemaakt aan de hand van lasers. Dit zorgt voor een hogere accuraatheid aangezien de verwerking van de diepte niet meer in software gebeurt, maar in hardware. Daarnaast is het ook eenvoudiger om meerdere Kinects samen te gebruiken. Er is namelijk weinig tot geen interferentie tussen de verschillende toestellen. Dit

Figuur 39: De Kinect V2, ontwikkeld voor de Xbox One (www.microsoft.com)

blijkt ook uit de resultaten die zijn behaald door gebruik te maken van meerdere Kinects tijdens captatie, zie sectie 5.3.

De Kinect V2 beschikt over veel betere specificaties dan de Kinect V1. Zo is het mogelijk kleurenbeelden op te nemen in Full HD(1920x1080 pixels), waarbij een dieptecamera zit die een resolutie van 512x424 pixels heeft. Deze opnames worden op 30 FPS gemaakt bij een felle belichting. Bij een minder belichte scene zal de Kinect zich automatisch aanpassen de beelden opnemen op 15 FPS. Dit zorgt ervoor dat de Kinect nog steeds correcte diepte informatie kan weergeven en dat de infrarood beelden steeds correct blijven. Ook op het gebied van skelet-generatie zet de Kinect V2 een grote stap voorwaarts. De skeletten die worden aangemaakt bevatten 26 verschillende joints (zie Figuur 40) waarbij deze niet enkel voor 1 persoon worden opgenomen maar zelfs tot 6 verschillende mensen kunnen worden gevolgd. Ondanks dat de laatste eigenschap van weinig nut is voor de implementatie die wordt gemaakt, is de verbetering van de kwaliteit van het skelet wel een gigantische vooruitgang. Dankzij deze verbetering is de Kinect V2 ook direct een pak interessanter om te gebruiken voor het mapping algoritmen. Hoe meer joints er worden aangeboden aan het algoritmen, hoe beter het resultaat kan zijn (zie sectie 6.2).

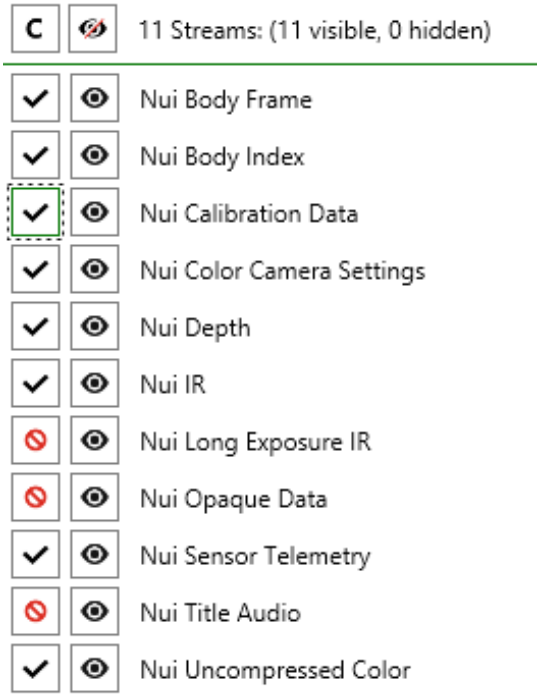


Figuur 40: Skelet van 26 joints dat wordt gevormd door de Kinect V2. (msdn.microsoft.com)

Er zijn echter ook enkele nadelen aan de Kinect V2. Aangezien deze maar recent beschikbaar is voor ontwikkelaars, is er veel minder documentatie te vinden dan bij de vorige API van de Kinect. Daarnaast is ook de API compleet veranderd. Dit laatste is het resultaat van de nieuwe aanpak. Bij de Kinect V1 konden er meerdere toestellen worden aangesloten op een bepaalde computer. Omdat de Kinect V2 echter de volledige snelheid van een USB3-bus gebruikt en daardoor ook de verbinding naar het moederbord bijna volledig in gebruikt neemt, is het onmogelijk meerdere Kinects te verbinden. Het is daarnaast nu wel mogelijk om vanuit meerdere toepassingen informatie op te vragen aan de Kinect. Er is nu namelijk een gedeeld geheugen voorzien waarin de Kinect de data van het toestel plaatst. De API laat dan toe om informatie op te vragen uit dit geheugen. Voor de gemaakte implementatie is er echter steeds één proces actief dat de Kinect gebruikt. Hierdoor wordt er geen gebruik gemaakt van het gedeeld geheugen.

5.2 Datastromen

De Kinect V2 (vanaf nu Kinect) gebruikt meerdere datastromen om informatie beschikbaar te maken via de API (zie Figuur 41). Ondanks het feit dat alle datastromen wel nuttig kunnen zijn, is het niet nodig al deze data op te slaan. Zo wordt de IR data door de Kinect gebruikt om het skelet te vinden (hierdoor is er geen interferentie door licht). Het is belangrijk om enkel de nuttige datastromen op te slaan voor de dataverwerking. Dit komt door de grote hoeveelheid data die doorheen de USB3-bus wordt gestuurd. Om te weten welke datastromen belangrijk zijn, geven we een korte bespreking over elke datastroom die wordt opgeslagen. Daarnaast is er een bespreking waarom deze stromen nodig zijn voor de verdere verwerking van de data. Een eerste belangrijke datastroom bevat de *camerapunten* (Nui Uncompressed Color). Deze worden gebruikt voor het geven van textuur en kleur aan het gecreëerde object. Daarnaast



Figuur 41: De verschillende datastromen van Kinect V2. De eerste kolom geeft aan of een datastroom wordt gebruikt door het algoritme. (Kinect Studio)

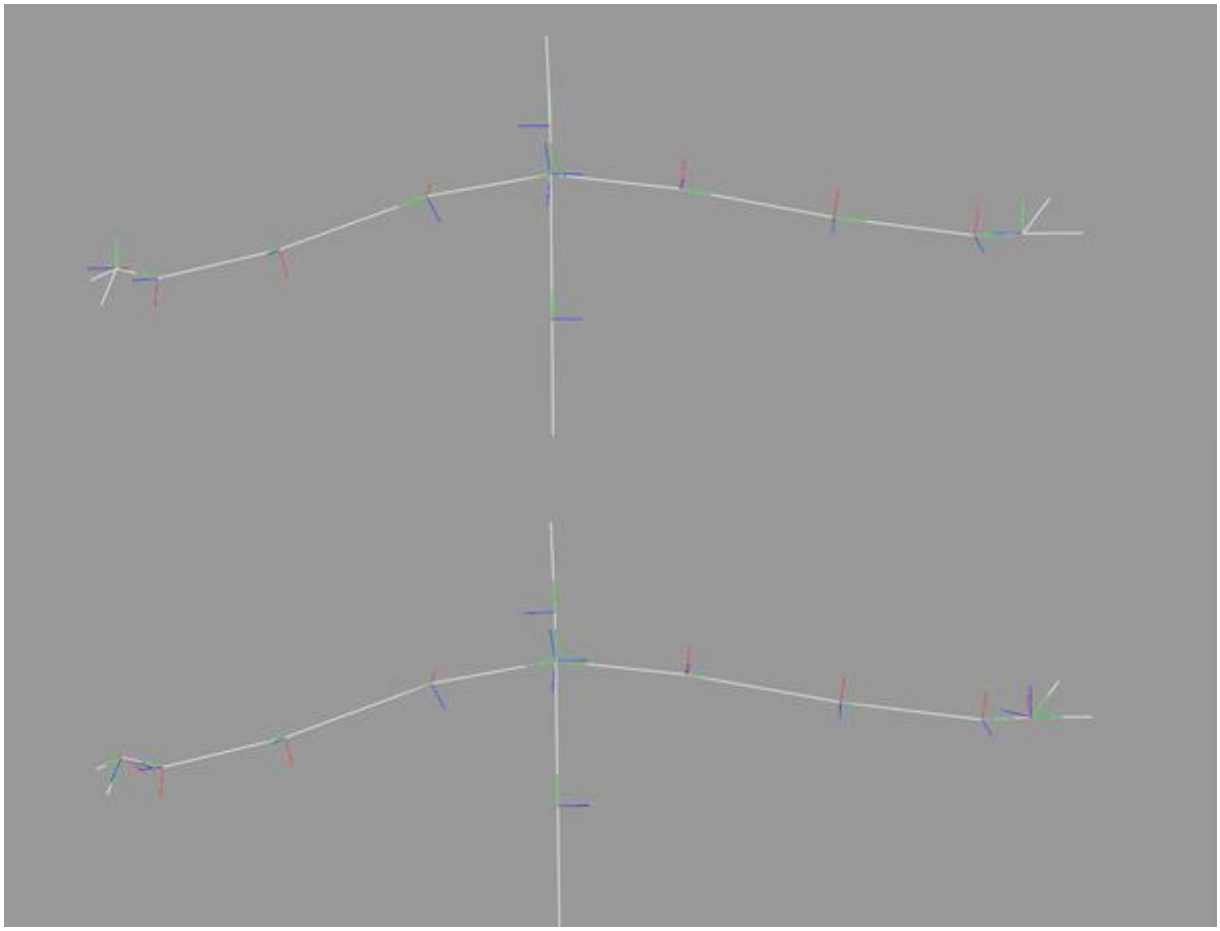
worden ook de *dieptepunten* bij gehouden. De dieptepunten geven informatie over de dieptemapping van het punt ten opzichte van de camera. Dit zorgt ervoor dat men eenvoudig kan controleren hoe diep een bepaald punt zich bevindt. Er wordt gebruik gemaakt van kalibratie data (Nui Calibration Data) en specifieke instellingen (Nui Color Camera Settings) om de verkregen dieptepunten samen met de camerapunten om te zetten naar een 3D puntenwolk. Voor skeletten te herkennen wordt er per frame een volledige berekening gedaan (Nui Body Frame). Hierin zit alle *skeletdata*. Daarnaast geeft de *bodyindex* data aan op welke plaatsen zich een skelet bevindt. De Kinect is instaat om 6 skeletten te tracken op elk gegeven moment. Deze bodyindex geeft aan tot welk van deze 6 skeletten een bepaald punt zich bevindt.

Bij het capteren van de data zijn er echter ook enkele problemen die moeten worden verholpen. Het eerste probleem is de grootte van de datastroom die wordt geleverd door de Kinect. Deze bedraagt ongeveer 45 Megabyte per seconden aan 15 frames per seconden. Het probleem hierbij is de schrijfsnelheid van het programma naar een harde schijf. Om deze problemen te voorkomen worden opnames gemaakt op een SSD. Hierdoor kan er voldoende data worden opgeslagen zonder frames over te moeten slaan. Een ander probleem heeft betrekking tot het tracken van de skeletten. Hierbij is het namelijk belangrijk dat alle mogelijke standen van een persoon kunnen worden herkend. De Kinect van Microsoft is echter gericht op gametoepassingen, waarbij enkel frontale herkenning belangrijk is. Hierdoor is de herkenning van de rug onmogelijk bij het gebruikmaken van de Kinect. Het concept blijft echter wel gelijk om enkel de voorkant te herkennen, en dus is het mogelijk om het concept verder uit te werken met de Kinect. Een ander probleem bij het gebruik van de Kinect is dat deze niet is ontwikkeld voor het gebruik bij hoge kwaliteit captatie. De Kinect is ontwikkeld voor game-gerelateerde toepassingen waarbij snelheid en efficiëntie belangrijker is dan de kwaliteit van het resultaat. Hierdoor zijn de marges op de datacaptatie groter dan bij specifiek ontwikkelde hoge kwaliteit hardware.

Eén van de belangrijkste datastromen die worden beïnvloed door de afwijkingen van de hardware, is de datastroom voor skeletrotaties. De rotaties bestaan uit 3 componenten: X, Y en Z. De Y-as geeft aan in welke richting het been van de joint gericht is. Deze ligt dus samen met de verbindingen tussen de child- en parent-joint. De X en Z-assen geven aan hoe het bot tussen de twee joints gedraaid is, de zogenaamde twist van de Denavit-Hartenberg representatie (zie hoofdstuk 4.3.1). Het is echter mogelijk dat de Y-as niet overeenkomt met het werkelijke bot van het skelet. Om deze incorrecte data te verwerken wordt er gebruik gemaakt van een correctie-algoritme. Dit algoritme verandert de rotatie over de kortste hoek naar het skelet. Het resultaat hiervan wordt afgebeeld op Figuur 42. Voor foute X- en Z-waarden is er echter geen direct correctie mogelijk aangezien er geen vergelijkingspunten zijn, die wel aanwezig voor de andere oriëntatie (namelijk child- en parent-nodes). Omdat deze rotaties echter van groot belang zijn voor het mappen van punten op de correcte plaats, is dit ook direct de grootste bron van ruis voor de latere oplossing van het concept.

Om deze accuraatheidsproblemen op te lossen kan er gebruik worden gemaakt van meerdere Kinects. Deze methode wordt besproken in (Yeung et al., 2013). Hierbij worden twee Kinects gebruikt om de posities van de skeletten te verbeteren. Volgens hun kan er gebruik worden

gemaakt van de verschillen tussen twee invalshoeken om zo occlusies te verhelpen. Er wordt ook gelet op het verwijderen van ongewenste trillingen in het object. De voorgestelde aanpak gebruikt beenderlengte als vaste constraint op het object. Zo wordt ervoor gezorgd dat er geen extreme afwijkingen kunnen plaatsvinden tussen verschillende frames. Dit gecombineerd met de correcties in skeletposities zorgt voor een grote verbetering in de kwaliteit van het skelet. Wel moet hierbij worden opgelet dat de puntenwolkdata nog steeds relevant is voor het gecreëerde skelet. Indien joint posities worden veranderd, is het belangrijk dit te controleren. In de volgende sectie wordt er meer uitleg gegeven om meerdere Kinects te gebruiken voor een scene te digitaliseren.



Figuur 42: (boven) Skelet representatie zonder correctie, (onder) gecorrigeerde rotaties. De correctie is voornamelijk zichtbaar in de handen en lichtelijk in de nek. De groene lijn is de representatie van de Y-as na de rotatie door de quaternion.

5.3 Multi-Kinect setup

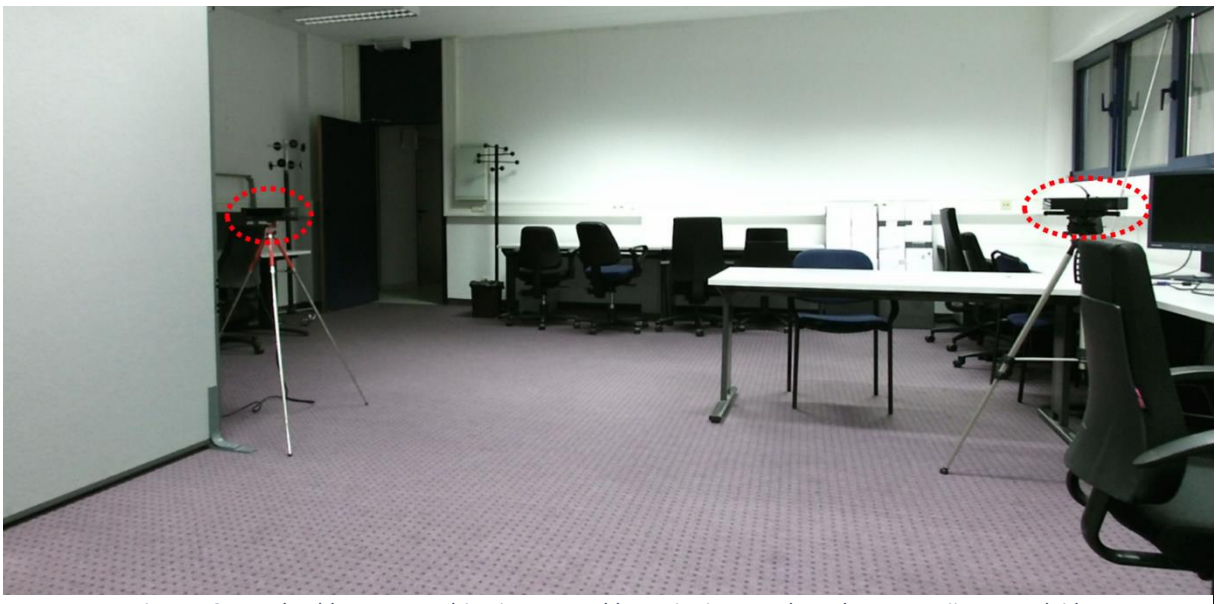
Zoals reeds eerder vermeld, zijn er enkele nadelen aan het gebruik van de Kinect of andere 3D scanning hardwareOm enkele van deze problemen op te lossen, wordt er voorgesteld om meerdere Kinects te combineren. Het gebruik van meerdere Kinects kent echter niet altijd een triviale oplossing. Zoals in sectie 3.3.1 wordt uitgelegd, is het stitchen (aan elkaar verbinden) van puntenwolken soms een complex gebeuren. Verder in deze sectie wordt de gebruikte techniek voor het aligneren en stitchen van de data verder uitgelegd. De bedoeling van deze stappen is een uiteindelijke verhoging in kwaliteit te bewerkstelligen. Tijdens de eerste

iteraties van het programma werd er gebruik gemaakt van slechts een enkele Kinect. Ondanks de degelijke resultaten was het onmogelijk een volledig resultaat te creëren. Er is namelijk geen informatie over de achterkant van een bepaald object. Door meerdere Kinects te gebruiken (die gesynchroniseerd zijn) is het mogelijk om ook de achterkant van het skelet te capteren en tevens een hoger aantal punten per frame te genereren. Daarnaast zijn er ook technieken, die een verbetering van het skelet toelaten door middel van meerdere Kinects. Deze technieken zijn echter niet van toepassing voor deze implementatie (zie sectie 5.3.1). Voor de volledigheid van thesis wordt er toch verwezen naar het onderzoek dat is verricht omtrent deze technieken. Het verbeteren van Kinect data, door middel van duplex Kinect opstellingen, wordt uitgelegd (Yeung et al., 2013).

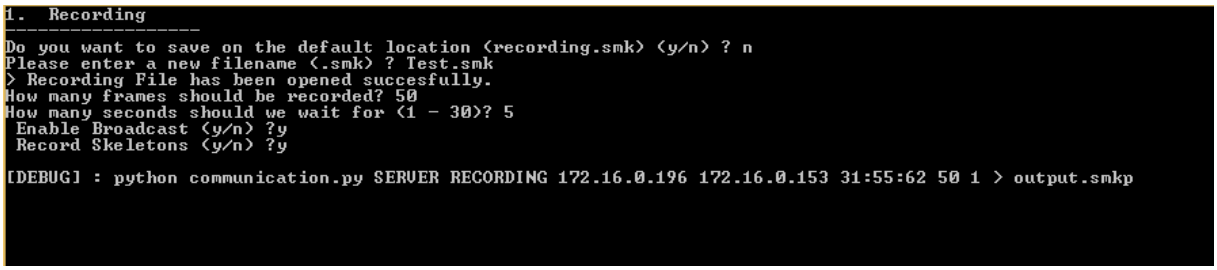
5.3.1 Gedistribueerde opnames

Zoals vermeld in de bespreking over de Kinect, is het onmogelijk meerdere Kinects met een computer te verbinden. Hierdoor moet er worden gezorgd voor een aanpak waarbij meerdere systemen synchroon data opnemen en verwerken. Hierbij moet de nadruk liggen op de verschillen in tijd tussen de verschillende systemen alsook de eisen die voor elk systeem moeten worden gesteld. Omdat het noodzakelijk is meerdere systemen in tijd te synchroniseren alvorens de captatie kan werken (zie sectie 3.3.1) moet er hiervoor een oplossing worden aangeboden. Een handige techniek is het gebruik van Network Time Protocol (NTP). Dit protocol stelt verbonden computers in staat hun klokken te synchroniseren tot op tienden van een seconden of preciezer. Om dit te realiseren werd er gebruik gemaakt van een externe applicatie genaamd Network Time Synchronisation. Deze applicatie laat eenvoudig toe meerdere computers met elkaar te synchroniseren. Eenmaal meerdere computers op dezelfde klok staan, is het nodig om gelijktijdig een opname te starten.

Om een gelijktijdige start van opnames te garanderen wordt er gebruik gemaakt van een, in python geschreven, netwerk script. Deze verstuurt een commando vanaf de zogenaamde server (de centrale computer in het systeem) naar de workers (systemen die enkel instaan voor captatie). In het verstuurde commando wordt een bepaald tijdstip doorgeven. Dit tijdstip



Figuur 43: Voorbeeld setup met drie Kinects. Beeld vanuit Kinect 1, de andere twee zijn aangeduid.



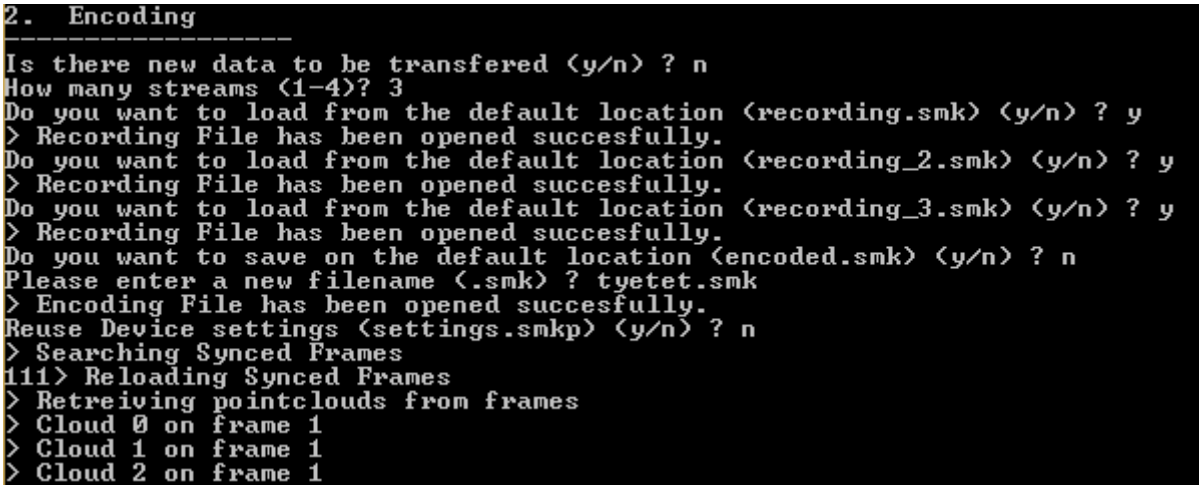
Figuur 44: Instellingen voor de captatie stap van de implementatie.

wordt bepaald door de gebruiker. Tijdens het programma worden er verschillende stappen in stelling doorlopen om ervoor te zorgen dat de captatie stap degelijk is ingesteld. Voor een voorbeeld wordt er verwezen naar Figuur 44. De eerste instelling van de captatie is het bepalen op welke locatie de data wordt opgeslagen. Eenmaal het bestand succesvol is geopend, worden er enkele opname gerelateerde parameters opgevraagd. De eerste parameter bepaalt hoeveel frames er worden opgenomen. Aan de hand van het aantal frames wordt er verzekerd dat meerdere Kinects op hetzelfde moment stoppen. Een tweede parameter duidt aan hoelang er moet worden gewacht met het starten van de opnames. Het moment dat de instellingen zijn afgelopen wordt een timer ingeschakeld. Na het aflopen van de voorop ingestelde tijd beginnen dan alle opnames te lopen. Tot slot wordt er gevraagd of het commando enkel voor de server geldt of voor alle workers in het netwerk. Indien het gaat om een gedistribueerde opname zal het commando naar iedere worker worden gestuurd. Omdat niet alle Kinects een goed zicht hebben op het karakter is het mogelijk dat de verkregen skelet data van de Kinects onbruikbaar is. Zoals vermeld in sectie 5.1, kan een Kinect enkel op het vooraanzicht een skelet plaatsen. Bij andere zichten is het dus onmogelijk een skelet te achterhalen. In deze gevallen is het mogelijk het skelet niet op te nemen. Dit is nodig omdat de implementatie enkel een frame opneemt indien het skelet overeenkomt met de dieptedata. Indien dit echter uitgeschakeld is, wordt enkel de puntenwolk opgeslagen.

Uiteindelijk wordt er een commando gestuurd zoals afgebeeld staat op de eerder vermelde figuur. Deze geeft aan dat het om een ‘SERVER’ instantie gaat in de stap ‘RECORDING’. De volgende stap geeft de IP-adressen mee van de andere computers die met een Kinect verbonden zijn. Tot slot wordt er aangegeven wat het tijdstip is om te beginnen (Minuut: Seconden: Milliseconden) gevolgd door het aantal frames en een boolean in verband met de skeletopname. Eenmaal de opnames compleet zijn, kan de gebruiker deze verzamelen op de server-computer. Op dit moment is het dan mogelijk om meerdere puntenwolken met elkaar te verbinden. Meer informatie over de gebruikte techniek wordt gegeven in de volgende sectie.

5.3.2 Point cloud stitching

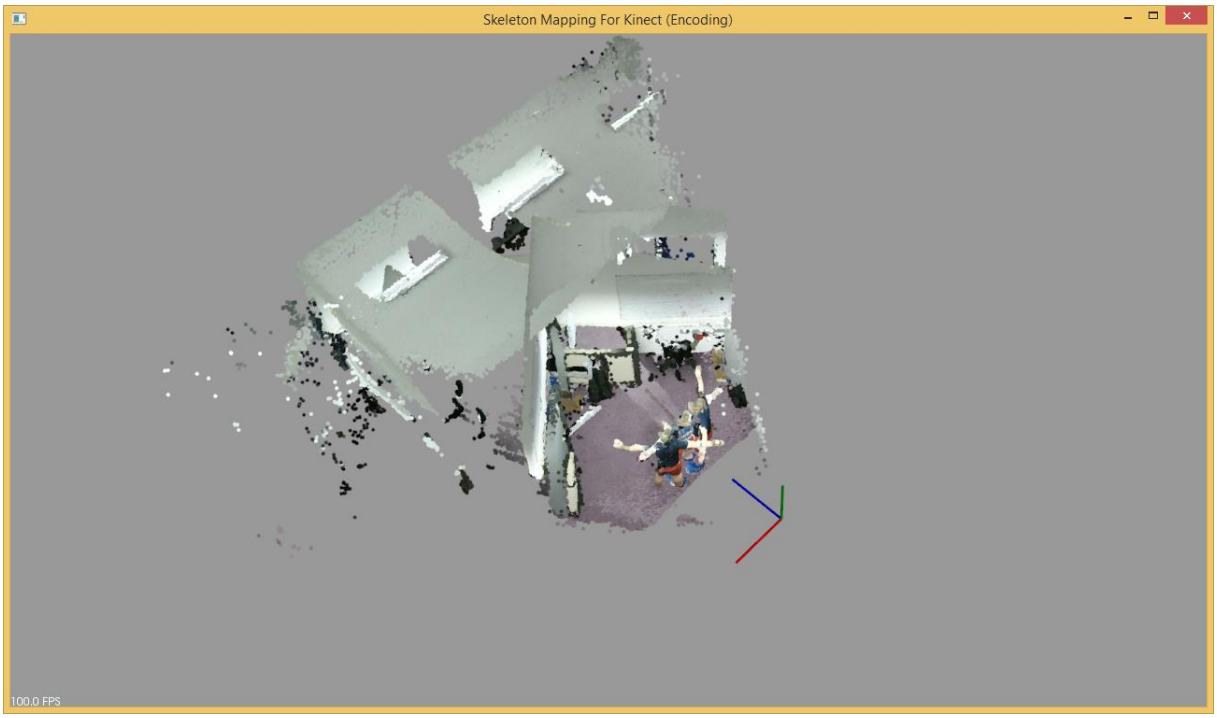
Zoals vermeld in sectie 3.3.1, zijn er meerdere technieken en situaties voor aan stitching te doen. De situatie van de implementatie voldoet echter niet aan een statische scene. De bedoeling is dat de gebruiker kan rond bewegen. Tijdens de bewegingen moet de alignatie intact blijven. Bij de dynamische stitching technieken zitten enkele mogelijkheden die ook niet van toepassing zijn. Zo is het niet mogelijk gebruik te maken van skelet data voor het aligneren van verschillende datasets. Het is namelijk mogelijk dat bepaalde Kinects geen skeletten gaan



Figuur 45: Het gebruik van de stitching stappen in de implementatie. Eerst worden er files gevraagd om tot slot gelijktijdig opgenomen frames te zoeken.

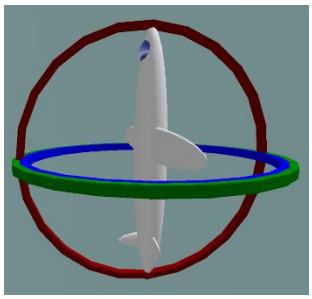
herkennen of het fout gaan herkennen vanwege de houding van de persoon. Hierdoor is enkel het gebruik van een eventueel referentie object mogelijk. Echter geldt bij het gebruik van deze objecten dat er een deel van het object overeenkomt tussen elke dataset. Indien echter de camera's op 120° van elkaar zijn gepositioneerd, is ook dit geen mogelijkheid.

Er is gekozen voor manuele synchronisatie aangezien automatische synchronisatie niet direct mogelijk was. De gebruiker wordt hierbij wel niet aan zijn lot overgelaten. Alvorens er een visuele alignatie wordt gevraagd, zorgt het programma ervoor dat de datasets doorheen de tijd gesynchroniseerd worden. Zoals verteld bij de gedistribueerde opnames, wordt er gezorgd dat start en eindtijd redelijk gelijklopend zijn. Het is echter nog steeds mogelijk dat een bepaalde computer er langer over doet een frame te controleren en weg te schrijven naar de



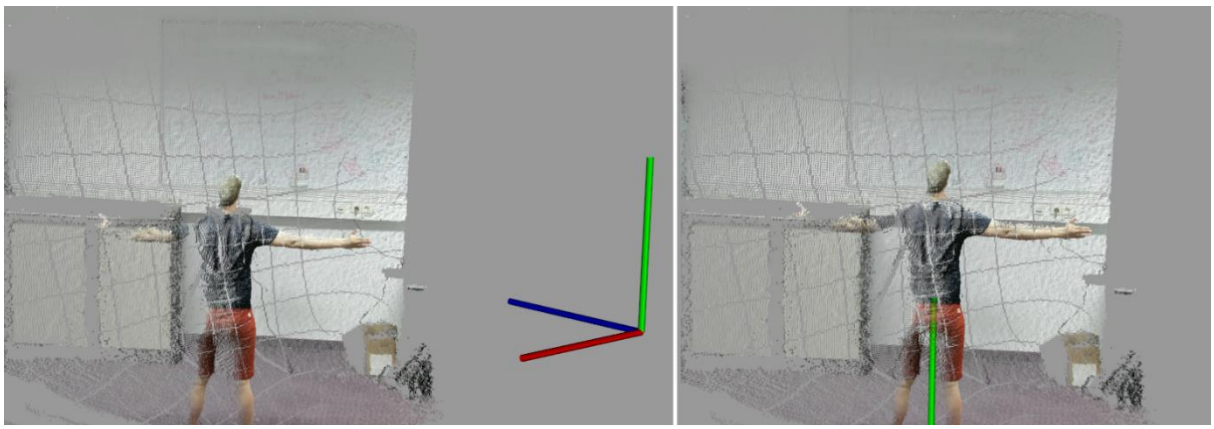
Figuur 46: Meerdere puntenwolken die gealigneerd worden. Het rotatie punt is hier de oorsprong voor alle puntenwolken.

harde schijf. In dit geval zullen frame 20 van beide datasets nog steeds niet overeenkomen. Verondersteld dat computer twee over iedere frame 5 milliseconden langer doet, dan is de afwijking doorheen de tijd van 20 frames reeds opgelopen tot 100 milliseconden, wetende dat de Kinect opneemt aan een snelheid van 15 frames per seconden (66 milliseconden per frame) bij slechte belichting en 30 frames per seconden (33 milliseconden per frame) bij goede belichting. Het is nodig dat er een controle wordt uitgevoerd dat enkel de correcte frames worden vergeleken. Hiervoor wordt het tijdstip van opname, dat per frame wordt opgeslagen tijdens de captatie, gebruikt. De opname van de server dient als vaste opname waar bij iedere frame wordt gezocht naar een beeld dat binnen een bepaalde afstand ligt. Bij de implementatie wordt er hier een waarde van 25 milliseconden voor gebruikt. Dit wil zeggen dat een beeld 25 milliseconden voor of na het serverbeeld moet zijn opgenomen om gelijk gesteld te worden. Na de gelijkstelling begint de visuele alignatie.

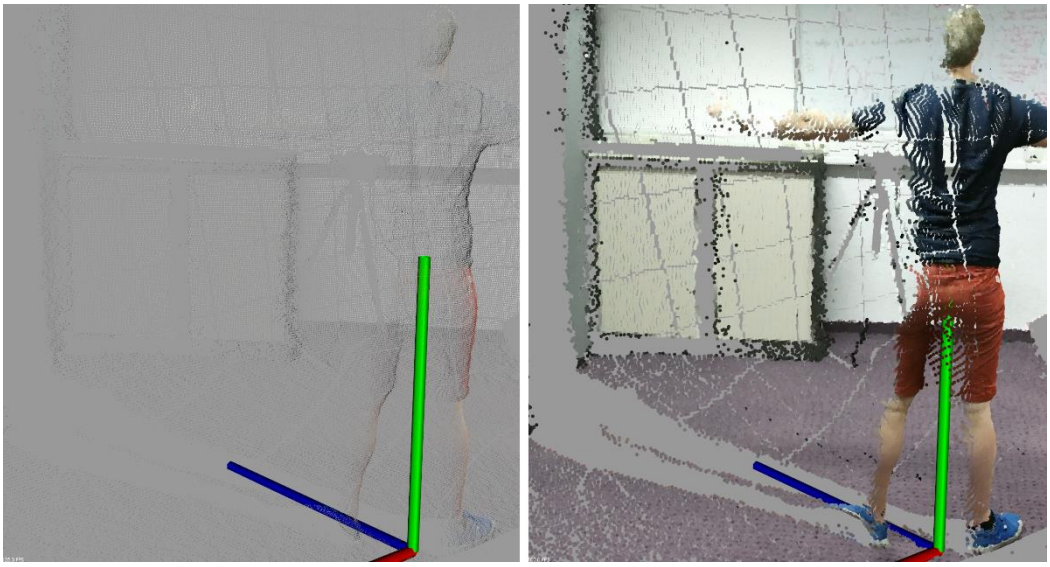


Figuur 47: Gimbal lock: De blauwe en de groene rotatie as vallen samen. Het roteren over de blauwe en rode as geeft hetzelfde resultaat. (Afbeelding via en.wikipedia.org)

Bij de *eerste iteratie* werd er rechtstreeks gestart met een alignatie aan de hand van de verkregen datastromen. Hierbij werden de verschillende datastromen weergegeven en was het nodig een rotatie en translatie voor iedere puntenwolk te zoeken zodat deze overeenkwam met de frame van de server. Omdat het rotatiepunt echter in de oorsprong van de Kinect ligt (zie Figuur 46), en niet op het personage dat gealigneerd moet worden, is het haast onmogelijk een degelijke alignatie te vinden. Eenmaal er translaties zijn gebeurd, is het zelfs nog complexer een resultaat te creëren. Hierbij is het ook de mogelijkheid dat er zich een Gimbal lock voordoet. Dit wil zeggen dat er een dimensie van rotaties verloren gaat. De oorzaak hiervoor is dat twee rotatie-assen op elkaar komen te liggen. Een visualisatie hiervan is zichtbaar in Figuur 47 Dankzij onder andere een Gimbal lock zijn rotaties na een tijd onmogelijk of niet intuïtief. Hierdoor kostte het aligneren van twee frames 10-20 minuten. Om deze verwerkingstijd te verlagen, is er een nieuwe methode toegevoegd. Bij de *tweede iteratie* worden er enkele technieken verplaatst. In plaats van gebruik te maken van XYZ rotaties en posities, wordt de gebruiker eerst gevraagd de puntenwolk te herpositioneren zodat het oriëntatiepunt niet meer in de oorsprong van de Kinect staat, maar op een gewenste plaats (de voeten van het karakter geven een ideale referentie). Het resultaat is een

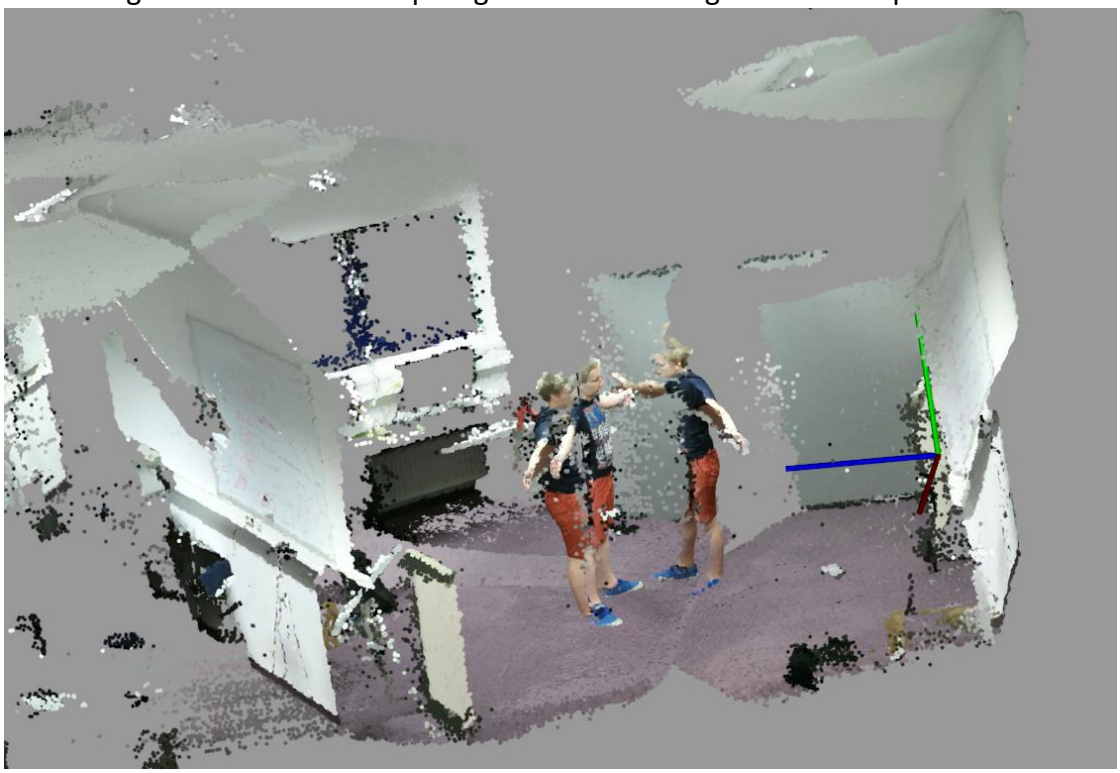


Figuur 48: Verplaatsen van de oorsprong van een puntenwolk. (Links) Het oorspronkelijke rotatiepunt. (Rechts) Rotatiepunt na de herpositionering.



Figuur 49: Het gebruik van dikkere punten. Hierdoor blijft de scene duidelijker. Deze optie is mogelijk dankzij PCL.

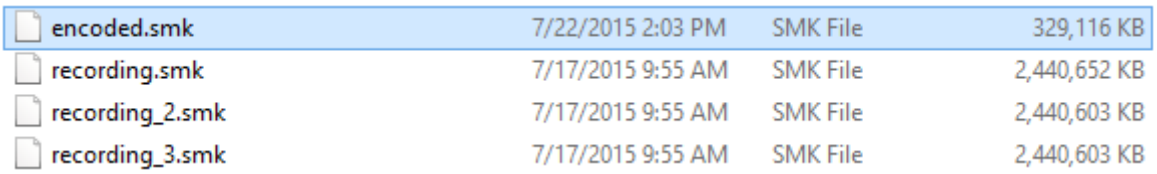
puntenwolk die volledig verplaatst is naar een nieuwe oorsprong, zie Figuur 48. Daarna worden er nog steeds XYZ posities gebruikt voor de puntenwolken te plaatsen. Maar de rotaties, die opgeslagen worden als quaternionen, worden aangepast door middel van een vector en een rotatie. De gebruiker bepaalt een vector vanuit de gekozen oorsprong. Deze bepaalt de vroegere X en Z rotaties van het object. Tot slot kan de gebruiker een hoek opgeven rond deze vector. Deze zorgt voor de vroegere Y-rotatie. Daarnaast is het ook mogelijk de grootte van de punten aan te passen tijdens de alignatie (zie Figuur 49). Op deze manier is het eenvoudig elementen te aligneren indien er dicht wordt ingezoomd op een bepaald aspect van de puntenwolk. Ook wordt het afstellen van de puntenwolken individueel gedaan, zodat het eenvoudiger is een nieuwe oorsprong te kiezen. Het aligneren van de puntenwolken wordt



Figuur 50: Alignatie van 3 verschillende datastromen.

wel gedaan met alle puntenwolken zichtbaar. Indien dit niet wordt gedaan is de alignatie moeilijker te corrigeren. Door deze techniek te gebruiken is het veel eenvoudiger voor een gebruiker om een object correct te positioneren. Eenmaal de voeten correct staan moet er namelijk geen positie meer worden veranderd. Alle rotaties die worden gedaan, zijn ten opzichte van de voeten. Het aligneren van twee datasets duurt zo slechts enkele minuten in de plaats van enkele tiental minuten.

Eenmaal deze alignatie is gebeurd, is het nodig om deze te *encoderen*. De gekozen instellingen op gebied van rotatie en translatie worden opgeslagen, zodat een bepaalde setup maar éénmaal moet worden gealigneerd. Daarna worden de verschillende datasets herleidt naar slechts één dataset. Hierbij worden alle datastromen omgevormd aangezien deze allemaal resolutie-afhankelijk zijn. In de plaats hiervan worden alle 3D punten opgeslagen van al de verschillende datasets. Indien de datastroom ook een skelet opname bevat, wordt ook deze opgeslagen. Deze punten worden nog steeds per stream opgeslagen zodat er eventueel verschillende kleuren aan toegekend kunnen worden. Dit 'feature' was in eerste instantie voor het debuggen van de uitkomst en is aanwezig gebleven aangezien er geen extra data voor moet worden opgeslagen. Het resultaat is een beduidend kleiner bestand (zie Figuur 51) dat nog steeds alle kleur informatie bevat. Dit komt vooral door het weglaten van de data die niet tot het karakter behoort en het samenvoegen van de andere data. De data van de omgeving wordt dus wel weggelaten éénmaal de datastromen zijn samengevoegd.

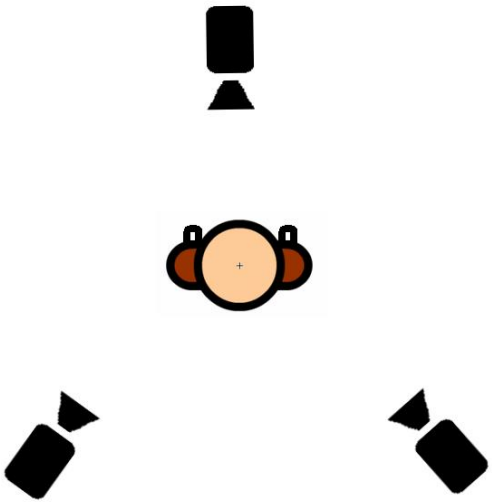


Figuur 51: Gecodeerde file van 50 frames aan de hand van 3 inputstromen. 330 MB ten opzichte van 7,2 GB.

5.3.3 Gemaakte opnames

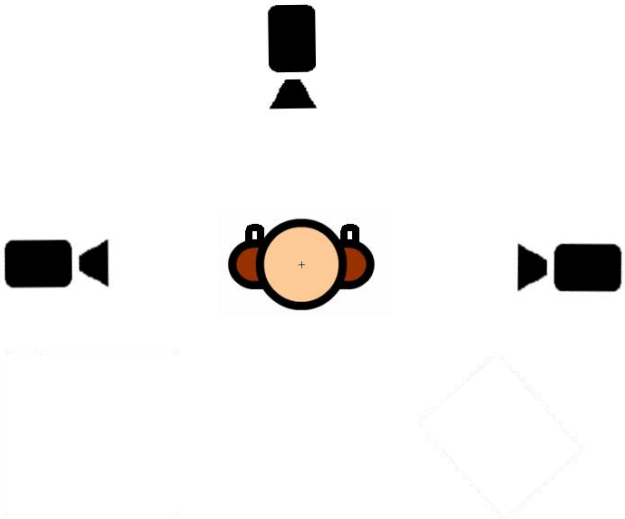
Tijdens de implementatie van de verschillende algoritmen, die worden gebruikt bij Skeleton Mapping For Kinect, was het noodzakelijk alles te kunnen testen met een bepaalde dataset. Voor dit mogelijk te kunnen maken zijn er meerdere opnames gemaakt. Zo zijn er vier verschillende opstellingen voor de Kinects opgenomen en per opstelling vier verschillende poses/acties. Elke pose zal besproken worden met de setup van de camera's en waarom de gegeven opstelling belangrijk is voor het testen van de code.

Eerst worden de verschillende poses kort toegelicht. Omdat het algoritme een verhoging van het model met zich meebrengt in de tijd, is het belangrijk dat alle delen van het lichaam zichtbaar zijn geweest doorheen de frames. Om dit na te gaan is de eerste opname een statische T-pose. Deze pose zorgt ervoor dat de bovenkant van het hoofd en de armen nooit in beeld komt. Hierdoor kan er worden gekeken of het programma nog steeds een correct model afwerkt. Een tweede pose, die wordt opgenomen, is er eentje met lichte beweging. Hierbij wordt er gestart met een T-pose waarna de armen trage bewegingen maken. Zo worden alle delen van het lichaam zichtbaar. Zo kan er worden gecontroleerd of er wel degelijk een verbetering is indien er meerdere frames worden gecombineerd (en dus meer van het model zichtbaar is). De derde opname is vergelijkend met de voorgaande enkel



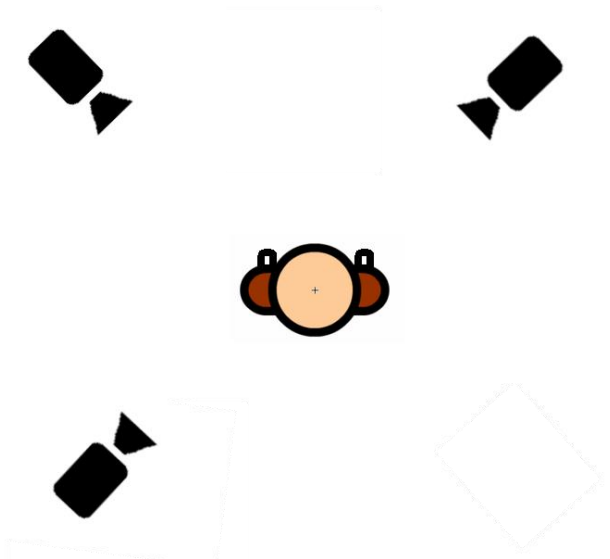
Figuur 52: Opstelling 1. De Kinects staan op 120° van elkaar met één camera recht voor de persoon.

worden hier meer abrupte bewegingen gebruikt. Samen met de beweging van de benen test dit hoe het algoritme omgaat met fouten. De Kinect is namelijk niet volledig in staat deze bewegingen volledig correct af te handelen. De foutmarge is namelijk zeer groot bij de rotaties van de benen. Bij deze opname wordt er wel nog steeds rekening gehouden met de restricties van de Kinect. Dit wil zeggen dat het karakter constant met het gezicht naar de server-Kinect toe staat (dit is de Kinect die aangesloten is op het systeem dat commando's verstuurd). Tot slot is er een laatste opname die puur wordt gebruikt om extremen, en foute input, te testen. Hierbij worden er enorm veel bewegingen gemaakt, waarvan een groot deel weggericht van de server-Kinect. Hierdoor is de input van het skelet vrijwel altijd fout. Aan de hand van de resultaten van deze opname kan er worden gekeken hoe ernstig fouten zijn voor het reconstructie-algoritmen en het mapping proces. Al de voorgaande vermelde sequenties worden opgenomen bij elke opstelling, die wordt getest. Hierdoor kunnen eenvoudig de verschillende opstellingen worden vergeleken, alsook welke invloed deze opstelling heeft op de verschillende sequenties.

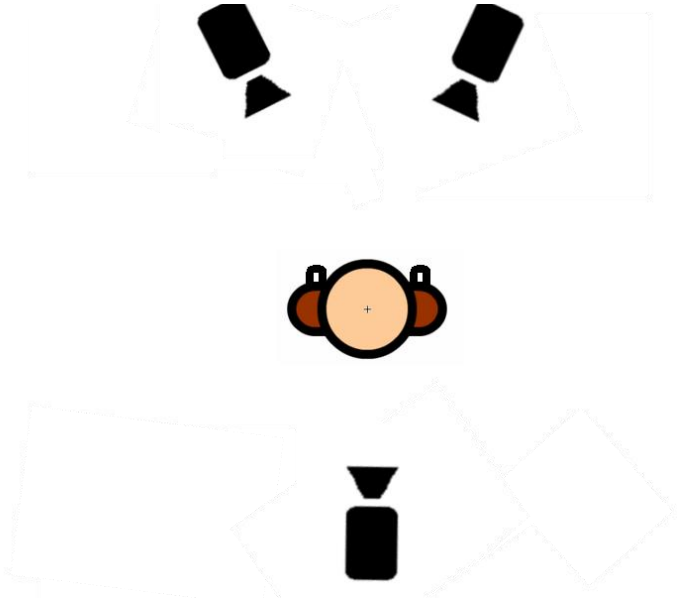


Figuur 53: Opstelling 2. De Kinects staan op 90° van elkaar met één camera frontaal op het karakter.

De eerste opstelling is terug te vinden op Figuur 52. Hierbij worden drie kinects op ieder 120° geplaatst rondom het karakter. Hierbij neemt enkel de frontale camera het skelet op. Zoals in sectie 5.3.1 wordt aangegeven, is het niet altijd mogelijk een skelet te herkennen onder een bepaalde hoek. Gedurende testen die in eerste instante zijn gebeurd, werd het reeds snel duidelijk dat de Kinects, die niet frontaal gericht stonden, geen data registreerden. Deze wachtten namelijk op een volledig zichtbaar skelet. Dankzij de hoek van 60° die wordt gemaakt tussen het frontaal zicht en de cameras, is het onmogelijk een skelet te detecteren. Hierdoor worden de workers gevraagd deze niet te registeren. *De tweede opstelling* focust meer op de fouten van de dieptescanner. De opstelling is terug te vinden in Figuur 53. Omdat de handen van het karakter op deze manier dicht bij de Kinect komt en daardoor grote diepte verschillen maakt over een kleine afstand in de 2D array van diepte punten, is het enorm moeilijk voor het systeem om correcte punten te genereren. In deze opstelling wordt er geen skelet data verzameld bij de niet frontale cameras. De reden is hier dezelfde als bij de eerste opstelling. *De derde opstelling,* die is gebruikt, staat afgebeeld op Figuur 54. Hierbij staat elke camera onder een hoek van 45° ten opzichte van een rechte kijkhoek. Hierdoor is het mogelijk om skelet data op te nemen bij iedere Kinect. De opnames onder deze vereisten blijken echter moeilijk te verlopen. Zo registreert niet elke camera constant een skelet waardoor opnames soms stilvallen. Daarnaast moeten de cameras een pak verder staan dan bij de eerder gebruikte opstelling zodat de kans groter is dat de Kinect het skelet kan herkennen. Waar de vorige opstellingen van de eerste keer direct lukte, was het nodig om deze opstelling tot tien keer toe te proberen. Dankzij de vergaarde kennis in deze opstelling is er *een vierde opstelling* gemaakt die het eenvoudiger moet maken skelet data te verzamelen voor alle Kinects. Deze opstelling is terug te vinden op Figuur 55. Bij deze opstelling zijn de twee frontale cameras dichter bij elkaar gebracht. Dit moet ervoor zorgen dat beide onder een goede hoek staan voor het detecteren van een skelet. Daarnaast moet dit ook helpen het probleem van frontale herkenning op te lossen. Zoals eerder vermeld, kan de Kinect enkel een frontaal zicht herkennen. Dit is ook het probleem bij de vorige opstelling, waar de Kinect ook aan de achterkant wordt geplaatst. Ondanks het feit dat de Kinect deze zal tekenen in het



Figuur 54: Opstelling 3. De Kinects staan op 90° van elkaar. In deze opstelling staat geen enkele camera frontaal op het karakter.



Figuur 55: Opstelling 4. Twee frontale Kinects en een kinect achter het karakter zorgen voor drie skeletbeelden.

vooraanzicht, blijft de data die verkregen wordt toch nuttig. Zo blijft de ruggengraat op vrijwel exact dezelfde positie, waardoor het mogelijk blijft deze informatie te gebruiken voor een eventuele alignatie of het verbeteren van de skelet data. Ook wordt de Kinect, die in opstellling twee zich aan de achterkant bevindt, naar een centrale plaats achter het karakter verzet. Het nadeel van al deze posities en setups, die zijn besproken, is dat het karakter vrij weinig bewegingsruimte heeft. Dit is het resultaat van het herkenningsalgoritme van de Kinect. Aangezien dit voor pure gametoepassingen is ontwikkeld, waar de gebruiker met het gezicht naar het scherm is gericht, is het niet nodig een algoritme te hebben dat een skelet langs achter kan herkennen. Bij een eventuele verbetering van dergelijke algoritmen is het mogelijk een volledig 360° beeld te hebben van het skelet waardoor herkenning eenvoudig wordt bij dynamische scenes.

In het volgende hoofdstuk wordt er dieper ingegaan op de analyse van de verkregen data. Hier worden ook de eventuele resultaten getoond van de opnames. Doorheen de thesis is het grootste deel gebaseerd op de meest ideale omgeving. Hierbij moet worden aangetoond dat het basisconcept werkt indien de input zo correct mogelijk is. Ter uitbreiding wordt een vergelijking gemaakt met andere input en andere opstellingen. Het is belangrijk op te merken dat het uiteindelijk product van de captatie en encoding stappen een enkele datastroom is waar enkel punten en eventuele skeletpunten in worden opgeslagen.

6.Skeleton Mapping With Kinect Data

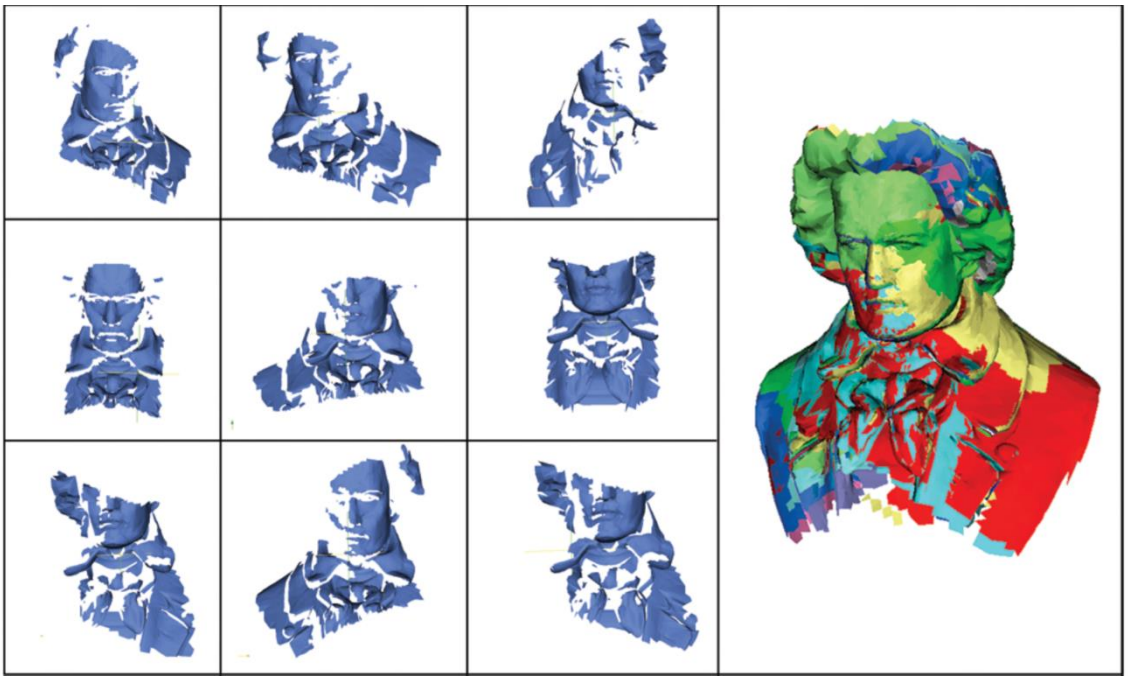
Dit hoofdstuk geeft een diepgaande bespreking van Skeleton Mapping For Kinect (SMK), een eigen voorgesteld algoritme dat enkele problemen (die zijn gevonden in de vorige hoofdstukken) probeert aan te pakken. Er wordt hier verder gewerkt op de inleiding voor het algoritme dat kan worden teruggevonden in sectie 1.4. Het is belangrijk op te merken dat het voorgestelde concept en implementatie verder werken op de reeds verwerkte data uit hoofdstuk 5. Daarnaast wordt de domeinstudie in het eerste deel van de thesis gebruikt als blauwdruk voor het ontwerpen van de algemene pipeline van het algoritme. Eerst wordt het algemene concept verder toegelicht door kort over elke tussenstap wat meer informatie te geven. Deze informatie is nodig om de keuzes te onderbouwen die doorheen de volledige implementatie zijn gemaakt. Hieronder valt de keuze van bepaalde reconstructie algoritmen, alsook de keuze van het gebruiken van bepaalde tussenstappen voor het verbeteren van het resultaat.

6.1 Concept

Zoals in de inleiding van de thesis wordt aangehaald, is het de bedoeling een algoritme/proces te voorzien dat game-ontwikkelaars eenvoudig in staat stelt om content te maken voor Augmented Reality games. Hierbij wordt er vooral gekeken naar het ontwerpen en animeren van karakters. Doorheen de thesis zijn verschillende methoden, algoritmen en processen besproken die deelproblemen van de pipeline (zie sectie 2.2) oplossen. Er zijn echter enkele problemen met technieken, die op dit moment gebruikt worden. Eerst zal het concept van Skeleton Mapping For Kinect worden uitgelegd, vervolgens wordt er ingegaan op de problemen, die door dit algoritme worden aangepakt.

De data captatie stap is reeds uitgelegd in het vorige hoofdstuk. Na het verkrijgen van de ruwe data uit de Kinect is het nodig deze te analyseren. De eerste stap bestaat uit het zoeken van de juiste ledenmaten van het skelet. Dit is noodzakelijk voor het kunnen plaatsen van de punten op het skelet. Tijdens de analyse wordt er gekeken welke punten bij welke ledenmaten horen. Omdat iedere representatie een andere hoeveelheid aan joints kan bevatten, is het belangrijk ervoor te zorgen dat bij iedere stap hetzelfde skelet wordt gebruikt. Het is daarom noodzakelijk om de herkenning te doen aan de hand van het skelet dat de Kinect opneemt en niet aan de hand van pose-herkenning. Eenmaal is herkend bij welke ledenmaat een bepaald punt hoort, kan deze worden geplaatst bij het betreffende lichaamsdeel.

Voor het plaatsen van een bepaald punt op een bepaald ledemaat moet er echter een nieuwe positie worden berekend. Hierbij moet er rekening worden gehouden met enkele verschillenden elementen. Het eerste, en voornaamste, is het feit dat de twee skeletten verschillende afmetingen kunnen hebben. De twee skeletten, die hier worden gebruikt, zijn enerzijds het skelet dat wordt opgenomen met de 3D camera en anderzijds het referentie-skelet, waarop alle puntenwolken worden geplaatst. De verschillende afmetingen kunnen ook een andere reden hebben, namelijk omdat meetapparatuur niet exact is of omdat er een volledig ander basis skelet wordt gebruikt voor de reconstructie. Indien de afmetingen van het



Figuur 56 Het samenvoegen van meerdere puntenwolken tot één puntenwolk met meer informatie. Uit deze puntenwolk wordt een mesh gegenereerd.

skelet anders zijn, zal er dus een conversie moeten worden gedaan tussen de twee punten. Een ander element, dat aandacht nodig heeft, is de algemene conversie tussen punten. De rotatie en positie van het punt uit de puntenwolk zal waarschijnlijk niet overeenkomen met een vergelijkbaar punt op het basis-skelet. Hierdoor moet er een berekening worden gemaakt om ervoor te zorgen dat het punt op de correcte plaats in de nieuwe puntenwolk wordt gezet. Tot slot moet er ook rekening worden gehouden met afwijkingen in de meetapparatuur op het gebied van rotaties. Indien rotaties niet kloppen zal er een grotere afwijking zijn op het eindresultaat.

De bedoeling van SMK is het eenvoudig creëren van 3D karakters waarbij animatie eenvoudig is. Om deze toepassing mogelijk te maken, moeten er eerst echter enkele problemen worden opgelost die bij huidige oplossingen bestaan. Op het oplossen van problemen na, zijn er ook enkele voordelen, die zich automatisch aanbieden dankzij het gebruik van SMK. Deze komen allemaal voort uit het feit dat er een volwaardig 3D model wordt gecreëerd, dat verbonden zit aan een skelet, dat eenvoudig is te animeren.

Eén van de problemen van 3D video is dat deze slechts zichtbaar is vanuit één bepaalde kijkhoek. Door gebruik te maken van SMK wordt het volwaardige object echter gevuld met punten doorheen de tijd. Op het einde van de sequentie is het zo mogelijk alle zijden van het object te hebben bekeken. Hierdoor is het mogelijk een volledig 3D model te reconstrueren. Als de opgenomen data dan opnieuw wordt afgespeeld met dit skelet, kan de opgenomen sequentie vanuit elke willekeurig gekozen hoek worden bekeken. Zolang deze zijde op een bepaalde frame is vastgelegd, zal er data aanwezig zijn. Het grote voordeel hiervan is dat er geen synchronisatie tussen meerdere punten wolken moet worden gedaan, waardoor er per frame nog steeds artefacten zijn, zoals zichtbaar bij huidige 3D video technieken (zie Q3D in sectie 6.6.1).

Omdat doorheen de tijd meerdere puntenwolken worden gecombineerd, zal ook de kwaliteit van de puntenwolk worden verhoogd. Door meer punten te gebruiken bij reconstructies van de 3D mesh, is er meer detail aanwezig in het object. Hierdoor geeft deze techniek niet enkel de mogelijkheid om in 360° te worden bekeken maar ook de mogelijkheid om dit in hogere kwaliteit te kunnen realiseren. Daarbij kan de data eenvoudig worden gereduceerd om de puntenwolk te verkleinen. Er kan namelijk een preciezere keuze worden gemaakt voor welke punten moeten overblijven. Dit is mogelijk omdat er meer punten aanwezig zijn voor trendlijnen te bepalen alsook voor, zoals net vermeld, meer detail te hebben op plaatsen met veel reliëf. Omdat na een bepaalde tijd het object volledig is omgezet naar een 3D model moet er bij het streamen van deze data ook geen puntenwolken meer worden doorgestuurd. Indien dus een livestream wordt opgezet waarbij een personage in 3D wordt gefilmd, is het mogelijk dat na een bepaalde tijd enkel nog skelet-posities en -rotaties moeten worden doorgestuurd en textuur-data en puntenwolk-data kan worden weggelaten. Dit verlaagt de nodige bandbreedte van een 3D-stream drastisch. Al is er in het begin van elke stream wel een hoge bandbreedte nodig.

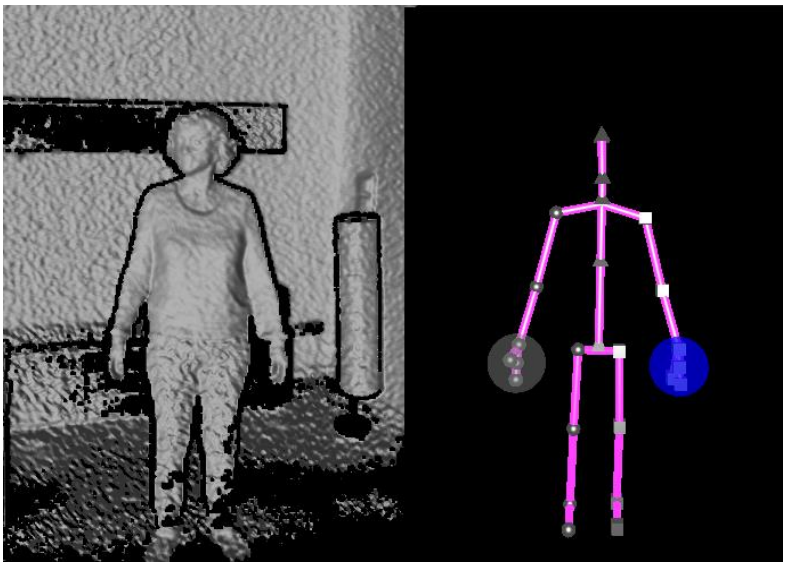
Door het gelijktijdig opnemen van zowel visuele data, in de vorm van puntenwolken, als het registreren van de skeletten, is het zeer eenvoudig een bepaalde animatie op te nemen. Waar vroeger gebruik moest worden gemaakt van zowel dure Motion Capture (zie sectie 4.3.3) als het laten ontwikkelen van een 3D karakter, is het nu voldoende om enkel gebruik te maken van een 3D film van het personage met alle acties die worden ondernomen. Dit zorgt er voor dat er minder tijd nodig is voor het ontwerpen van een karakter. Daarbij moet een designer nu enkel het karakter tekenen met alle benodigde elementen. Hierbij valt het ontwikkelen van een 3D model (bijna) volledig weg. Het is mogelijk dat het geregistreerde model naderhand moet worden bijgewerkt om enkele artefacten weg te werken of van enkele betere kleuren te voorzien voor de immersie van het spel te verhogen.

In de volgende secties worden telkens bepaalde delen van het algoritme besproken en uitgewerkt. Hierbij worden problemen van bepaalde technieken besproken alsook de aanpassingen die nodig zijn om bepaalde technieken te laten werken. Er wordt vanuit gegaan dat er wordt gewerkt op de data die verkregen is door gebruik te maken van de captatie stap zoals beschreven in hoofdstuk 5. Eerst zal er een bespreking zijn van de skelet herkenning en de verschillende methoden die hiervoor zijn getest. Dit wordt gevolgd door een bespreking van de mapping stap, waarbij punten worden geplaatst op een nieuwe puntenwolk. Tot slot wordt er gekeken naar de animatie van dergelijk punten-wolken alsook de visualisatie van het uiteindelijke resultaat. Bij de visualisatie wordt ook besproken hoe de conversie van een puntenwolk naar een mesh wordt gerealiseerd. Er wordt hier echter geen geanimeerde mesh voorgesteld maar een statische 3D reconstructie van het resultaat dat is verkregen uit de andere stappen. Ten opzichte van het oorspronkelijke concept zijn er wel enkele aanpassingen gemaakt. Zo gaat het concept ervan uit dat het algoritme in real-time kan worden gebruikt om data te streamen over een netwerk. In deze implementatie wordt er echter gebruik gemaakt van een programma dat in twee delen werkt. Het eerste deel valt onder captatie, waarin alle data wordt uitgelezen naar de harde schijf of het werkgeheugen. Dit wordt gevolgd door het tweede gedeelte wat het mapping algoritme en het animatie gedeelte betreft. Enkel het animatie algoritme kan een sequentie op ware snelheid verwerken. De andere stappen

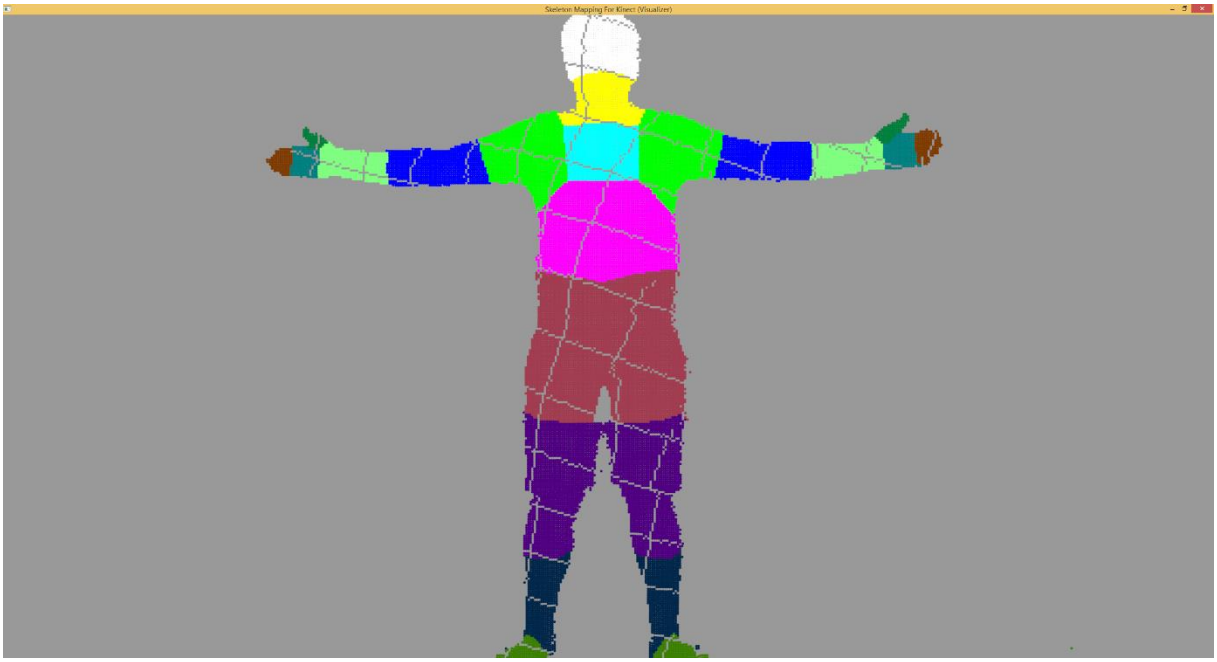
gebeuren allemaal op voorhand voor de volledige sequentie aangezien de verwerkingstijd per frame te hoog is voor een real-time analyse en representatie.

6.2 Skelet Herkenning

Zoals eerder vermeld is het herkennen van het skelet belangrijk. Hierbij wordt echter niet enkel het detecteren van het skelet geteld maar ook het detecteren van de verschillende lichaamsdelen en welke punten bij welk lichaamsdeel hoort. Het eerste deelprobleem, het herkennen van het skelet met al zijn joints en rotaties, wordt verwerkt door de Kinect SDK zelf. Hierbij levert de SDK zowel de rotaties als de posities van alle 26 joints die aanwezig zijn in het skelet (zie Figuur 57). Deze stap wordt uitgevoerd tijdens de captatie (zie hoofdstuk 5) stap van het programma. Het vinden van de correcte ledematen daarentegen wordt gedaan in de processing stap in het tweede gedeelte van het programma. SMK probeert puntenwolken van het gecapteerde skelet te plaatsen op een tweede skelet dat als representatie dient voor het karakter. Het zogenaamde referentie skelet wordt voornamelijk gebruikt voor het bepalen van de lengte van de verschillende ledenmaten van het skelet. Verschillen in lengte tussen twee verschillende lichaamsdelen wordt op het moment van mapping aangemaakt, dit is volledig gebaseerd op de verhouding tussen het referentie skelet en het gecapteerde skelet, dat ieder frame verandert. Daarnaast dient het referentie skelet ook als hulpmiddel bij het visualiseren van data alvorens een tweede frame wordt ingelezen. Zo wordt tijdens het aligneren van verschillende datastromen (zie sectie 3.3.1 en sectie 5.3.2) reeds gebruikt gemaakt van een referentie skelet. Dit referentie skelet is het skelet van de eerste frame van de datastroom. Het is namelijk niet mogelijk een vooraf gedefinieerd skelet te gebruiken. Het is echter wel mogelijk op het einde van de mapping stap (zie sectie 6.3) het volledige skelet en al de puntenwolken te projecteren op een vooraf bepaald skelet. Wel is het vereist dat dit skelet uit evenveel joints bestaat en dat de structuur van beide skeletten overeenkomen. De structuur van de skeletten die worden geleverd door de Kinect zijn terug te vinden in de bespreking van de hardware in sectie 5.1.



Figuur 57:Links een geregistreerde dieptemap van de kinect. Rechts het hieruit resulterende skelet. (Captatie gemaakt door Kinect Evolution)



Figuur 58: Resultaat van het Closest Joint algoritme. Iedere kleur stelt een andere joint voor.

Het direct beginnen herkennen van bepaalde punten binnen de puntenwolk is echter niet aan te raden. Zoals al enkele keren is aangehaald is de Kinect geen perfect toestel. Er zijn vele fouten in de data waarmee rekening moet worden gehouden. Eén van de belangrijkste problemen is de accuraatheid van de rotaties die worden gegeven door de Kinect. Omdat deze aan de basis liggen van bijna alle verdere technieken en algoritmen is het belangrijk toch kort wat aandacht te schenken aan het aanpakken van dit probleem. Zoals eerder vermeld bij captatie (zie sectie 5.2), worden rotaties opnieuw georiënteerd door middel van een eenvoudige techniek. Dit zorgt er echter wel voor dat de rotaties niet altijd volledig overeenkomen aangezien de correctie enkel de hoeken veranderd. Het resultaat hiervan is dat er inherent al fouten zitten door gebruik te maken van deze rotaties. Het is echter onmogelijk deze rotaties niet te gebruiken in het verdere verloop van de implementatie.

Voor Skeleton mapping is het essentieel te weten bij welk lichaamsdeel een bepaald punt hoort. Het doel van de mapping stap is namelijk het plaatsen van een bepaald punt in een puntenwolk die wordt gebonden aan een bepaalde joint. Dit hoofdstuk zal meer uitleg geven over het vinden van de correcte lichaamsdelen van de punten uit de puntenwolk. Om te bepalen bij welke joint een bepaald punt ligt, kunnen veel methodes worden gebruikt. In de implementatie is er gebruik gemaakt van drie verschillende implementaties. Deze worden allen verklaard in de volgende sub-secties. Ieder algoritme is een opvolger van de voorganger. Deze opvolgers zijn bepaald aan de hand van fouten in de voorgaande oplossing. Aan de hand van de verkregen kennis door het testen van bepaalde algoritmen is er uiteindelijk vooruitgang gemaakt in de aanpak van dergelijke problemen.

6.2.1 Closest Joint

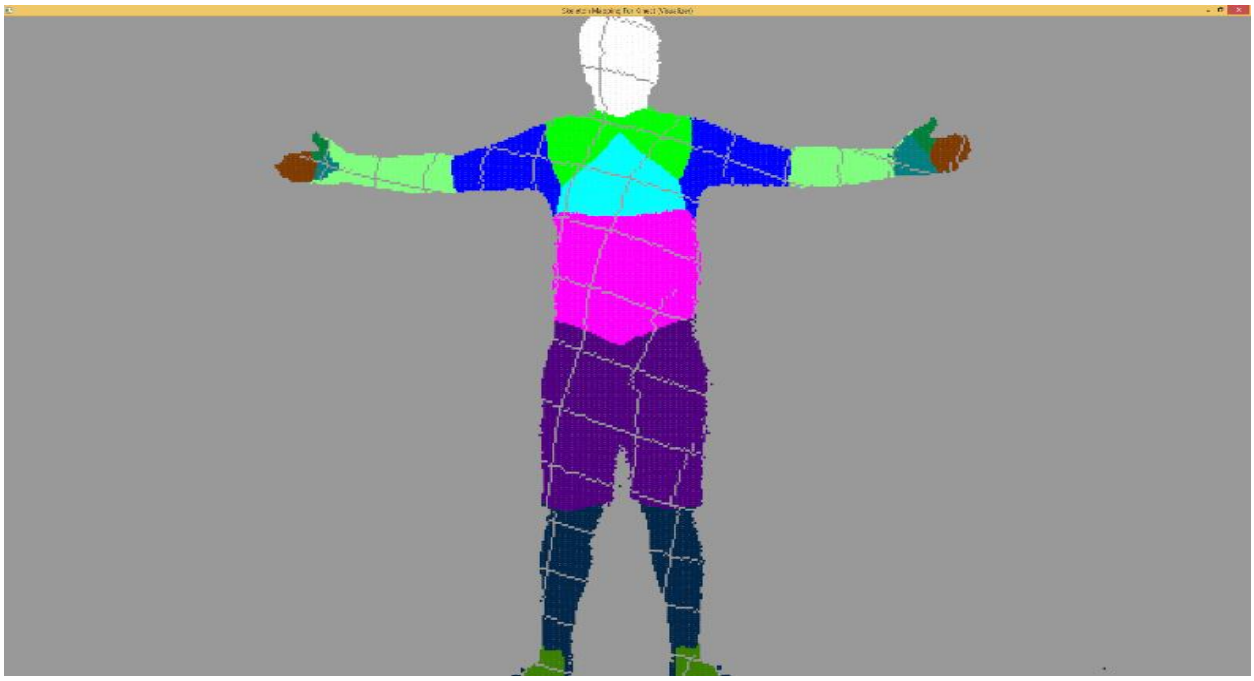
De eerste aanpak die is geïmplementeerd, is ook de meest eenvoudige manier om een selectie te maken. Bij deze aanpak wordt voor elk punt alle mogelijke joints overlopen. Voor iedere joint wordt de afstand tussen het geselecteerde punt en de joint berekend. Indien deze korter is dan een voorheen berekende afstand, dan wordt deze joint opgeslagen als de bijhorende

joint. Eenmaal alle mogelijke joints zijn overlopen wordt de joint met de kortste afstand teruggegeven. Het voordeel aan deze techniek is dat er amper berekeningen moeten worden gemaakt. Er zijn echter ook veel problemen met de gebruikte techniek. Een eerste probleem is dat, indien er sprake is van bijvoorbeeld de elleboog, zowel punten vóór de joint als achter de joint worden gezien als horende bij de elleboog. Indien echter de elleboog wordt geplooid, bewegen enkel de punten achter de joint zich in de werkelijkheid. Het is daar dus duidelijk dat de twee helften van de puntenwolken niet dezelfde joint bespreken. Dit probleem geldt voor iedere joint uit het lichaam. Ook in het visuele resultaat zien we dit duidelijk. In Figuur 58 is duidelijk zichtbaar dat een deel van het gezicht wordt gezien als de nek. Ook bij de schouders is het duidelijk merkbaar dat het resultaat grote fouten geeft.

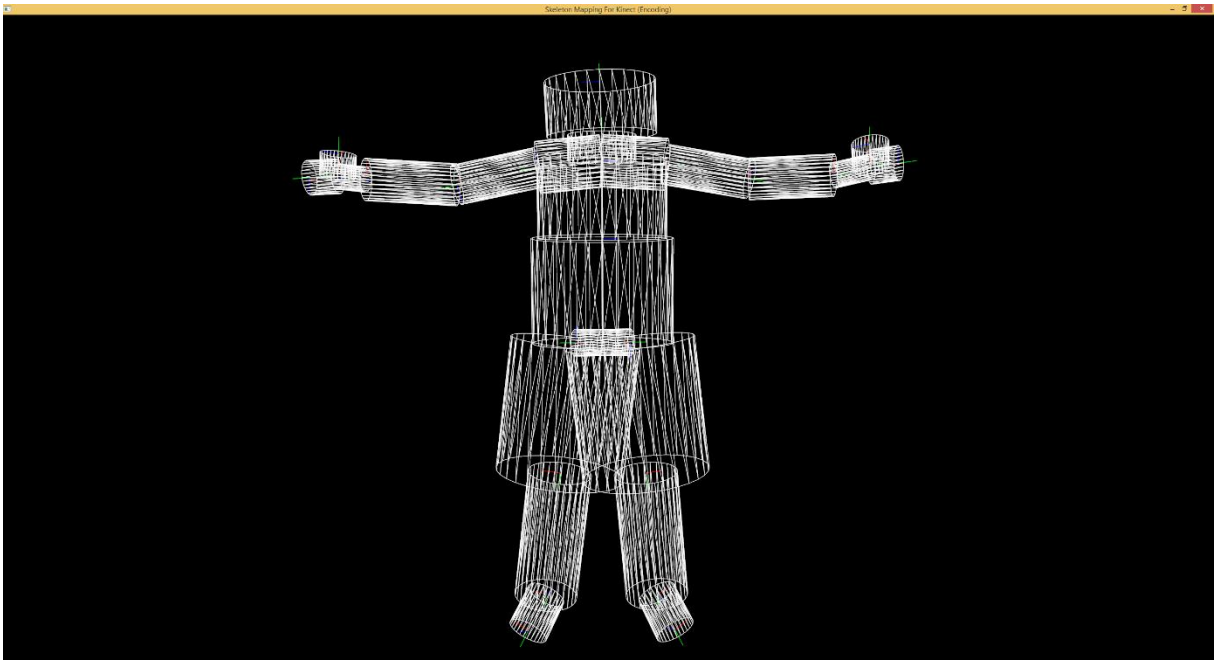
6.2.2 Triangular Distance

In het vorige model zijn er enkele elementen, die niet mee worden geteld voor het bepalen van het correcte ledemaat. Zo wordt er enkel naar de child-joint gekeken en niet naar de parent-joint. De tweede aanpak houdt hier wel rekening mee. In plaats van enkel de afstand tussen het gevonden punt P en de child-joint in rekening te nemen, wordt ook de afstand tot de parent node in rekening gebracht. Gewoonweg de som van beide afstanden is echter geen correcte bepaling van de afstand van de joint. Een goed tegenvoorbeeld is een punt op de schouder te nemen. Deze zal voor beide joints van de nek een lage waarden hebben maar de afstand naar het uiteinde van de schouder zal groter zijn. Hierdoor zou er worden gekozen voor de nek, terwijl eigenlijk de schouder het gewenste resultaat zou zijn. Daarom brengt het algoritme ook dit in rekening. Er wordt namelijk gerekend met de verhouding tussen de lengte van de ledematen en de afstanden tot de respectievelijke joints van de ledematen. De formule is dan als volgt;

(Afstand tot child + Afstand tot parent) / Afstand tussen child en parent



Figuur 59:Herkenning aan de hand van Triangular Distance

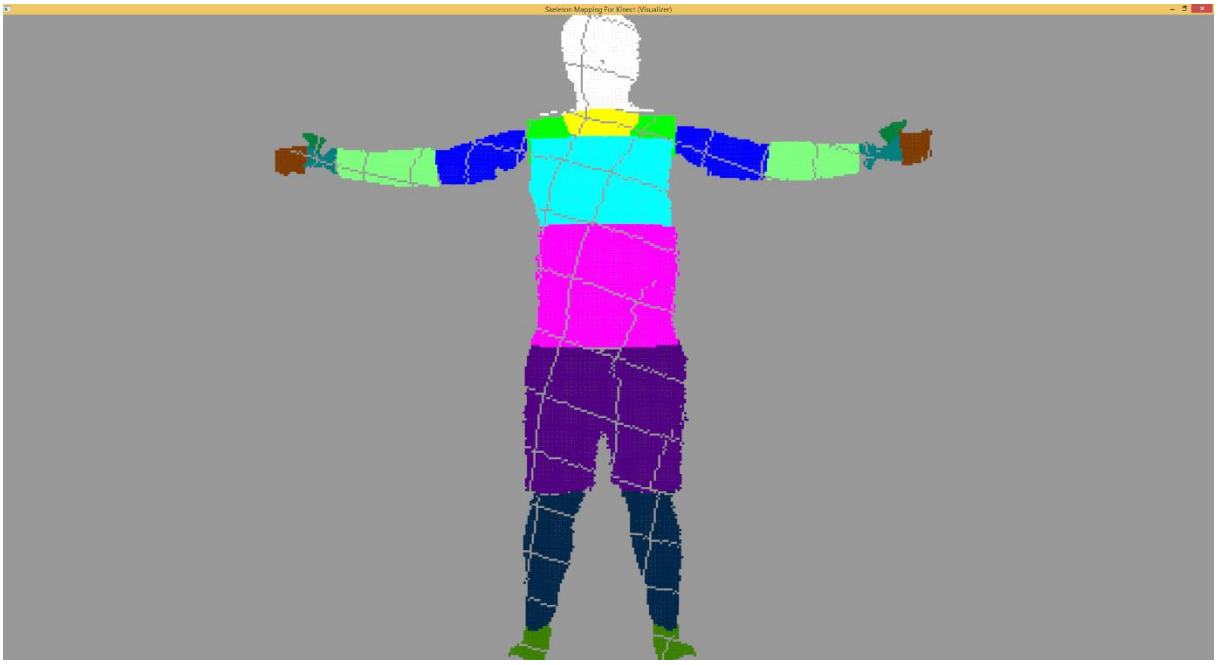


Figuur 60: Het instellen van het Radius Based algoritmen door middel van een visuele representatie

Hierbij wordt er een driehoek gevormd tussen het punt, de child joint en de parent joint. Aangezien er rekening wordt gehouden met verhoudingen werkt het algoritme reeds een pak correcter. In Figuur 59 wordt weergegeven wat de uitkomst is van deze techniek toegepast op dezelfde data als de voorgaande methode. We merken hier duidelijk dat de ledematen dichter aanleunen ten opzichte van de werkelijkheid. Deze formule zorgt ook al voor een ander element verder in de berekeningen. Zo wordt de verhouding tussen het punt en de joints al vastgelegd. Zoals vermeld in het vorig hoofdstuk over Data Captatie, is het mogelijk dat ledematen verschillende lengtes krijgen doorheen meerdere frames. Door rekening te houden met de verhouding kan het punt verplaatst worden zodat de verhouding constant blijft over alle frames. Hierdoor worden punten beter geplaatst op de lichaamsdelen. Dit proces wordt verder uitgelegd in de volgende sectie over Skeleton Mapping.

6.2.3 Radius Based

Omdat ook de vorige techniek niet correct bleek te zijn na animatie is een volledige andere aanpak gebruikt. Tijdens de animatie werd het namelijk duidelijk dat de schouderbladen van het karakter volledig naar voor komen. Dit zorgt voor een zeer slechte ervaring voor gebruikers. Om dit probleem op te lossen is er een techniek gebruikt die niet 100% automatisch werkt, maar aan de hand van vooraf ingegeven parameters. De gebruiker dient namelijk zelf te definiëren hoe groot een bepaald ledenmaat is. Dit moet echter niet op een complexe manier. Het enige wat de gebruiker moet aangeven is de straal van elk ledenmaat. Dit wordt visueel getoond aan de gebruiker zodat het eenvoudiger is de instellingen te maken. In Figuur 60 wordt getoond hoe dit visueel aan de gebruiker wordt getoond. Er moet rekening worden gehouden dat een andere lichaamsbouw eventueel enkele kleine aanpassingen vereist aan het model om ervoor te zorgen dat deze correct is. Dit is een nadeel aan het gebruik van een niet volledig automatische techniek. Eenmaal de gebruiker de grootte van de ledenmaten heeft bepaald wordt de puntenwolk overlopen zoals in de vorige algoritmen. Nu wordt er echter niet gekeken naar afstanden of andere elementen, maar er wordt enkel nagegaan of een bepaald punt in een bepaalde cilinder ligt. Indien het punt in een bepaalde



Figuur 61: Joint herkenning gebaseerd op de straal van bepaalde ledenmaten.

cilinder ligt wordt hij hieraan toegekend. Het is echter ook mogelijk dat een punt in twee cilinders ligt. Indien dit het geval is, wordt er gebruik gemaakt van een afstandstechniek, namelijk Closest Joint (zie sectie 6.2.1). Dit zorgt ervoor dat bijvoorbeeld de benen en schouders correct worden herkend. Deze techniek heeft echter nog een voordeel bij het gebruik. De voorgaande technieken geven altijd de dichtstbijzijnde joint in 'kost' terug. Het kan dus zijn dat een punt dat fout is herkend door de hardware, en hierdoor dus 50 cm van het lichaam af ligt, toch werd toegekend aan een bepaalde joint. Bij de nieuwe techniek is dit echter niet meer mogelijk. Het punt moet liggen in een bepaalde joint alvorens het toegekend wordt. Indien een punt dus te ver zou liggen wordt het niet meer verwerkt. Het resultaat van de techniek is terug te vinden in Figuur 61. Er is hier duidelijk zichtbaar dat de schouders volledig los staan van de armen in tegenstelling tot eerdere resultaten. Ook wordt de hand nu beter opgesplitst en worden de schouderbladen individueel aangeduid. Enkel in de streek van de nek blijkt alles als het hoofd te worden herkent. Dit is grotendeels te wijten aan de foute

Figuur 62: Incorrecte instellingen bij de stralen kunnen fouten opleveren voor de herkenning.

positioneren van het skelet binnen het karakter. Al is dit in principe geen probleem aangezien de nek mee beweegt met het hoofd bij eventuele rotaties tijdens een animatie. Het is enorm belangrijk dat de afstellingen van het skelet en de stralen correct zijn ten opzichte van het gebruikte model. Ter vergelijking stelt Figuur 62 het resultaat voor indien er voor de romp een te grote straal wordt gekozen. Dit geeft duidelijk weer dat een groot deel van de schouders wordt geplaatst bij de romp. Hierdoor wordt er tijdens de animatie stap geen enkel punt in de betreffende zone verplaatst

6.3 Skeleton Mapping

Alle stappen die tot hiertoe zijn overlopen, dienen als voorbereiding voor deze stap, namelijk Skeleton Mapping. Ondanks alle voorgaande stappen, is het belangrijk op te merken dat er geen perfecte data wordt gegenereerd. Er zit, zoals eerder vermeld, veel ruis op de input die door de hardware wordt gegenereerd en helaas kan die niet volledig worden weggewerkt. In deze stap wordt er echter geen rekening gehouden met de fouten in de hardware. Eventuele fouten die hierdoor blijven bestaan, kunnen nadien worden weggewerkt. Dit is mogelijk omdat Skeleton Mapping enkel een herpositionering doet van alle punten uit de puntenwolk waarbij lokale puntengroepen (puntengroepen die behoren tot dezelfde lichaamsdelen) op dezelfde manier worden verplaatst. Hierdoor ondervindt iedere lokale groep geen verandering (ten opzichte van naburige punten), er komen enkel meer punten bij aangezien meerdere frames worden samengevoegd.

Skeleton Mapping probeert punten vanuit een puntenwolk te plaatsen op een Skeleton. Hierbij zijn er enkele belangrijke stadia die moeten worden doorlopen en daarnaast enkele belangrijke aspecten die in rekening moeten worden gebracht. De belangrijke stadia zijn in de vorige secties reeds besproken. Voor de volledigheid worden deze kort nog eens overlopen. Eerst en vooral moet er data worden ingelezen via een 3D scanner. Deze 3D data moet dan worden gecorrigeerd, zoals skelet rotaties, om daarna door een analyse proces te gaan. Dit analyse-proces bepaalt welke punten bij welke lichaamsdelen horen. Eenmaal dit gebeurd is, kan er een mapping worden gedaan naar een basis skelet. Dit basis skelet dient als representatie model, waarop alle verschillende frames worden geplaatst. Vanaf hier zal dan ook het basis-skelet worden gebruikt als aanduiding voor het uiteindelijke model, waarbij rotaties, posities en eventuele verhoudingen anders zijn dan de oorspronkelijke skeletten en puntenwolken. Het is belangrijk op te merken dat alle eigenschappen van de twee skeletten volledig verschillend kunnen zijn. Enkel het aantal joints en de structuur van het skelet moeten overeenkomen in het huidige concept. Deze verschillen moeten in acht worden genomen bij het overzetten van puntenwolken van het basis-skelet naar het uiteindelijke model.

In de volgende alinea's worden de problemen individueel besproken om tot slot een volledige formule te voorzien die de punten correct kan corrigeren bij het mappen. Het eerste probleem is de positie van de skeletten en hun joints. Zoals eerder vermeld heeft het Kinect Skeleton Structure (zie sectie 4.3.1) individuele coördinaten voor alle joints. Dit geldt zowel voor begin- als eindpunten van elke joint. Het is dus nodig om elk lichaamsdeel individueel te bekijken en de punten te verplaatsen aan de hand van de posities van de joints. Hierbij is het dus niet nodig om een algemene translatie te vinden tussen de verschillende skeletten. Eenmaal een

Figuur 63: Input stream die niet gemapped wordt. Gevormd door het optellen van 10 frames (Opstelling 1 - Lichte beweging als dataset)

punt wordt geplaatst bij een lichaamsdeel is het eenvoudig de translatie te corrigeren. Dit kan eenvoudig worden gedaan door het verschil van de parent-joint locaties te nemen voor zowel het basis-skelet als het uiteindelijke model. Dit verschil moet dan enkel worden bijgeteld bij het punt van het basis skelet.

Naast de translatie, of de positie van het skelet, moet ook de rotatie worden gecorrigeerd. In de eerdere analyse hebben we reeds enkele elementen van de rotaties veranderd om ervoor te zorgen dat deze minder onderhevig zijn aan ruis. Tijdens de implementatie wordt er vanuit gegaan dat de rotaties, die zich op het moment van Skeleton Mapping in de datastructuur bevindt, de correcte rotaties zijn. Om de rotatie te corrigeren kan er geen rotatie worden uitgevoerd op de quaternion, die zich in het basis-skelet bevindt. Aangezien het skelet los staat van de puntenwolk moet elk punt individueel worden geroteerd naar de correcte plaats. Voor de simpliciteit wordt er even vanuit gegaan dat de joint-positie zich in de oorsprong bevindt, waardoor er een gewone rotatie kan worden uitgevoerd. Indien het punt zich niet in de oorsprong bevindt moet er eerst een translatie naar de oorsprong gebeuren. De rotatie is eenvoudig terug te vinden door een quaternion aan te maken die het verschil tussen de twee quaternionen van beide skeletten voorstelt. Indien het punt dan wordt geroteerd door middel van deze quaternion, bevindt deze zich op de correcte plaats.

Er rest nu nog een probleem bij de Skeleton Mapping. Zoals eerder verklaard kunnen de skeletten verschillen tussen verschillende frames. Dit betreft ook de lengte van bepaalde lichaamsdelen. Om ervoor te zorgen dat deze punten zich toch op een correcte plaatsen bevinden moet elk punt volgens een juiste factor worden verplaatst. Ook dit probleem valt wiskundig eenvoudig op te lossen. Elk punt, dat wordt gemapped op een nieuw skelet, kan worden gezien als een driehoek, namelijk het punt dat wordt gemapped samen met de

Figuur 64: Resultaat van 10 frames die worden gemapped door middel van Skeleton Mapping. De gebruikte dataset is gemaakt met Opstelling 1 - lichte bewegingen.

parent- en child-joint. Deze driehoek moet gelijkvormig zijn voor en na het mappen van een bepaald punt. Indien de zijde tussen de twee joints dus kleiner wordt, moet het gemapped punt ook worden verplaatst. Dit kan worden gedaan door het lijnstuk tussen de parent-joint en het punt te vermenigvuldigen met de factor die wordt verkregen door de lengtes van de twee lichaamsdelen met elkaar te delen. Indien ervoor wordt gezorgd dat het beginpunt van het lijnstuk (de parent-joint) op dezelfde plaats blijft, blijft de gelijkvormigheid behouden.

De net vermelde problemen moeten in principe op hetzelfde moment worden afgewerkt. Het volgende stuk pseudo-code geeft aan hoe dit in de implementatie gebeurt:

```
# Pointcloud: de puntenwolk van de huidige frame
# Base_parentPos: positie van de parent van het basis-skelet
# Dest_parentPos: doelpositie van de parent
# Base_Rotation: rotatie van het basis-skelet voor dit punt
# Dest_Rotation: rotatie van het doel-skelet voor dit punt
# Base_Lenght: lengte van het lichaamsdeel op het basis-skelet
# Dest_Length: lengte van het lichaamsdeel op het doel-skelet

for Points in Pointcloud:
     NewRotation = Difference(Base_Rotation,Dest_Rotation)
     Factor = Dest_Length / Base_Length

     Point -= Base_parentPos
     Point *= Factor              #Correctie op lichaamsdelen
     Point.Rotate(NewRotation)    #Correctie van rotatie
     Point += Dest_parentPis      #Correctie van translatie
```

Naast het plaatsen van de punten op het nieuw skelet is het ook belangrijk de kleurinformatie mee over te brengen naar het nieuw model. Zoals vermeld in sectie 5.1 geeft de Kinect een

datastroom dat die kleurinformatie bevat. Deze wordt door middel van de Kinect geplaatst op overeenkomstige punten van de dieptecamera. Deze kleureninformatie bestaat uit RGB waarden. Er zicht echter een lichte afwijking op de kleurencamera en de diepte camera. Dit komt deels door occlusies. Soms kan de dieptecamera een dieptepunt meten maar heeft de kleurencamera een ander beeld. hierdoor worden de foute kleuren geplaatst op een bepaald dieptepunt. Dit kan worden gezien in Figuur 64. Aan de zijkanten van het lichaam zijn kleuren van de achtergrond terug te vinden. Door de kleine verschillen tussen opeenvolgende frames en de ingebouwde accuraatheid van de Kinect-hardware worden bepaalde punten niet 100% correct geplaatst op het skelet. Deze afwijking zorgt er bij het gebruik van kleuren ook voor dat bepaalde punten door elkaar worden gehaald en dus bepaalde gebieden foute kleuren krijgen.

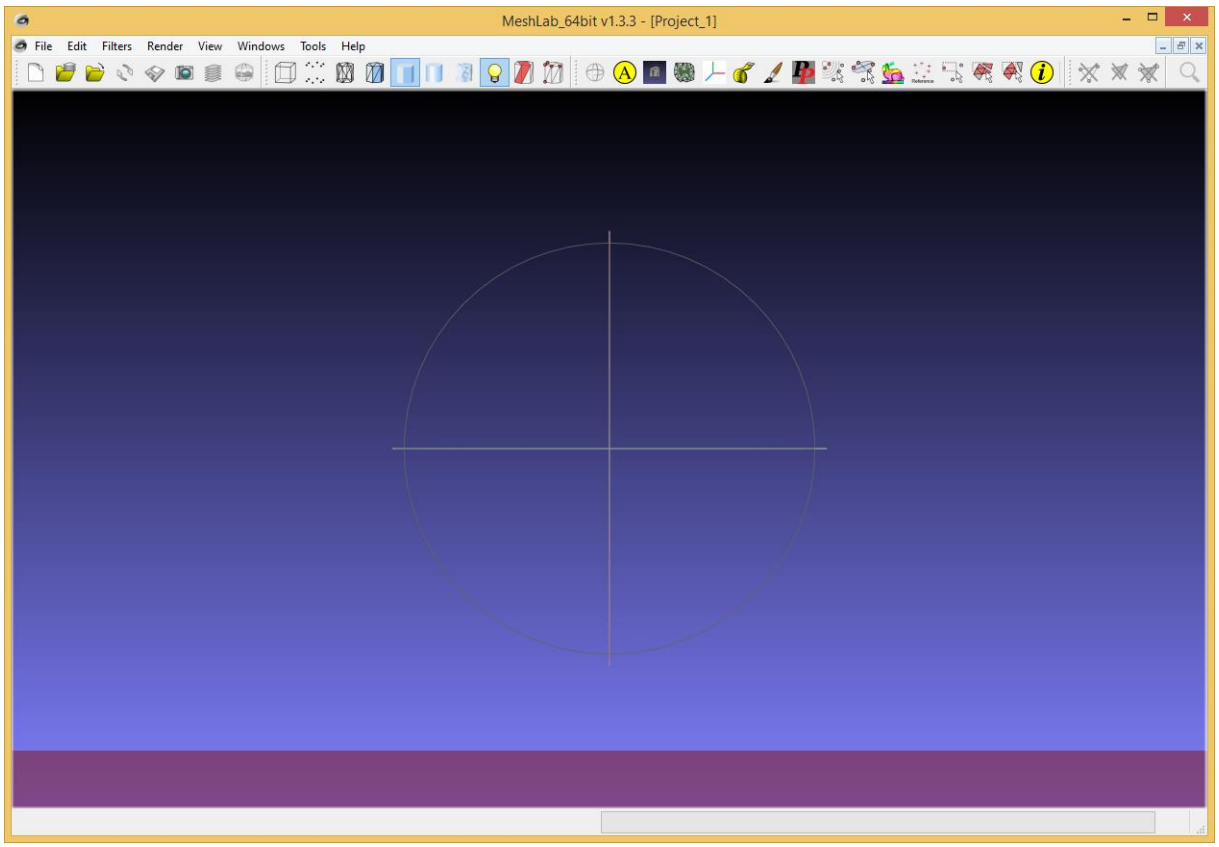
6.4 Data Visualisatie

Na het verwerken van alle data, geeft het algoritme een skelet terug met aan iedere joint een rotatie, translatie en een puntenwolk. Zoals eerder vermeld bevat de puntenwolk enkel elementen die horen bij de betreffende joint. De volgende stap voor deze data is het visualiseren van het resultaat. Zoals aangetoond in sectie 4.1, kan er zowel een polygon representatie als een puntenwolk representatie worden gebruikt. Voor dit gedeelte van het programma is er gebruik gemaakt van de PCL Library. Deze heeft de functionaliteit om eenvoudig punten toe te voegen aan een 3D omgeving. Alvorens een volledige integratie in het programma te maken is er ook gebruik gemaakt van MeshLab. Dit opensource programma is gebaseerd op de VCG-library en biedt enkele belangrijke mogelijkheden. Deze sectie gaat in eerste instantie een korte beschrijving geven over hoe de visualisatie door middel van de puntenwolken verloopt. Aangezien dit redelijk eenvoudig is, omdat reeds alle puntenwolken zijn aangemaakt, zal er nadien uitleg worden gegeven over het maken van een mesh. Tijdens de implementatie is er eerst gebruik gemaakt van het eerder vermelde Meshlab. De resultaten hieruit waren echter niet overtuigend en daarom is er nadien overgeschakeld naar een eigen implementatie. Op deze manier was het eenvoudiger de fouten en/of optimalisaties te vinden.

De *representatie aan de hand van de puntenwolken* zit volledig doorheen het programma. Zo geeft de hardware alle directe informatie om puntenwolken te genereren. Hierdoor is het vrij eenvoudig om als uiteindelijk resultaat een puntenwolk te verkrijgen. Ook doorheen de rest van alle stappen wordt er gewerkt aan de hand van de puntenwolken. Zo wordt voor iedere joint een puntenwolk opgesteld waar gedurende de mapping fase punten aan worden toegekend. Tijdens deze toekenning worden er enkele rotaties en translaties uitgevoerd (zie sectie 6.3). Dit zorgt ervoor dat iedere puntenwolk exact de punten van de betreffende joint bevat. Voor de dataset te visualiseren, worden alle puntenwolken samengevoegd. Dit wordt gedaan door de algemene rotatie en translatie die opgeslagen zit bij de joint uit te voeren op de puntenwolk. Het resultaat van deze berekeningen wordt dan samengevoegd tot een grote puntenwolk. Deze kan eenvoudig worden toegekend aan een visualisatiescherm door middel van de PCL Library. De resultaten van zulke visualisatie stappen zijn terug te vinden in dit hoofdstuk. Om een puntenwolk te visualiseren, moet er hierbuiten niets worden gedaan. Enkel indien er aan animatie wordt gedaan (zie sectie 6.5), moeten er eventuele veranderingen gebeuren. Het voordeel van het gebruik van puntenwolken bij de representatie

is het feit dat de datastructuur opgesplitst is per joint. Dit zorgt ervoor dat het eenvoudig is om een bepaalde joint niet te visualiseren. Hiervoor moet namelijk enkel de puntenwolk van de bepaalde joint niet worden opgevraagd. Dit maakt het een eenvoudig manier om resultaten te vergelijken tijdens de implementatie stappen. Doorheen de tekst wordt er echter enkel gekeken naar volwaardige modellen, aangezien de afwijking in punten minimaal is per puntenwolk en enkel het volwaardige beeld een goed overzicht geeft.

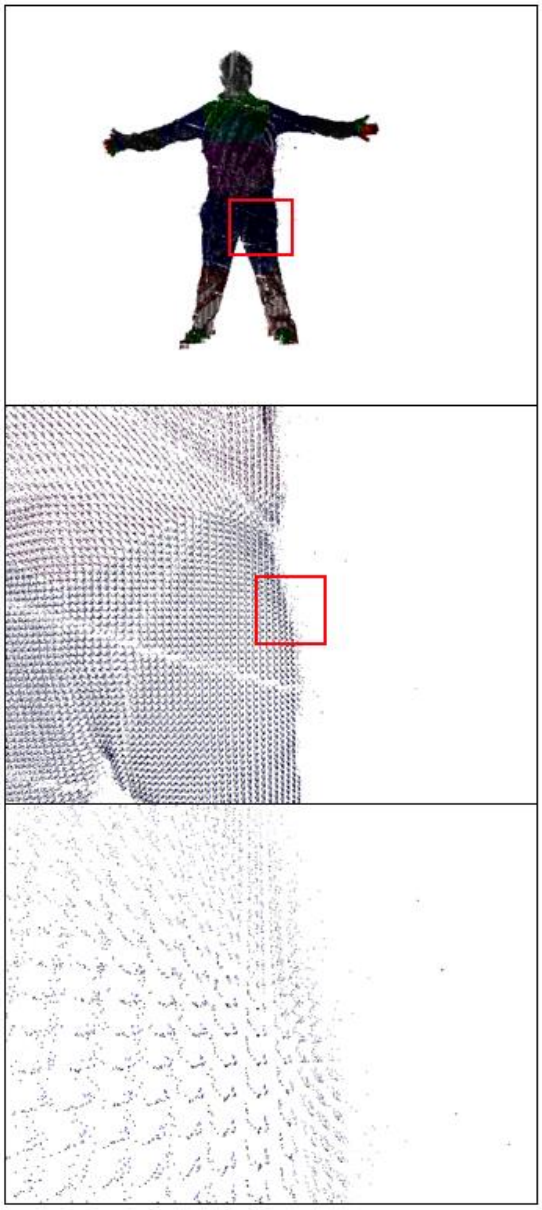
Het volgende gedeelte van deze sectie wordt toegewijd aan het zoeken naar een *representatie door middel van polygonen*. Hierbij moet er worden gekeken naar de beste mogelijke techniek voor een geloofwaardig en accuraat eindresultaat. De technieken die hier nodig zijn, zijn reeds besproken in sectie 3.2 in verband met Surface Reconstruction en in sectie 3.3 in verband met het vinden van fouten in het model. Voordat de puntenwolk kan worden omgezet moet er worden gekeken welk algoritme hier optimaal voor is. De besprekingen over reconstructie algoritmen lieten duidelijk blijken dat Poisson en Fourrier de twee beste algoritmen zijn. Uit de verdere vergelijking blijkt ook dat van beide algoritmen Poisson het beste overweg kan met fouten, maar hiervoor wel aan performantie moet inboeten (zie sectie 3.2.2). Aangezien het algoritmen niet real-time moet werken is er gekozen voor het Poisson algoritmen. Dit zorgt dankzij de functiebenadering voor een geloofwaardig resultaat. De verdere onderdelen van deze sectie bespreken de verschillende aanpakken, die tijdens de implementatie zijn gebruikt. Er wordt hierbij ook dieper ingegaan op de gebruikte software en waarom bepaalde keuzes niet goed zijn voor het visualiseren van het resultaat.



Figuur 65: MeshLab Interface.

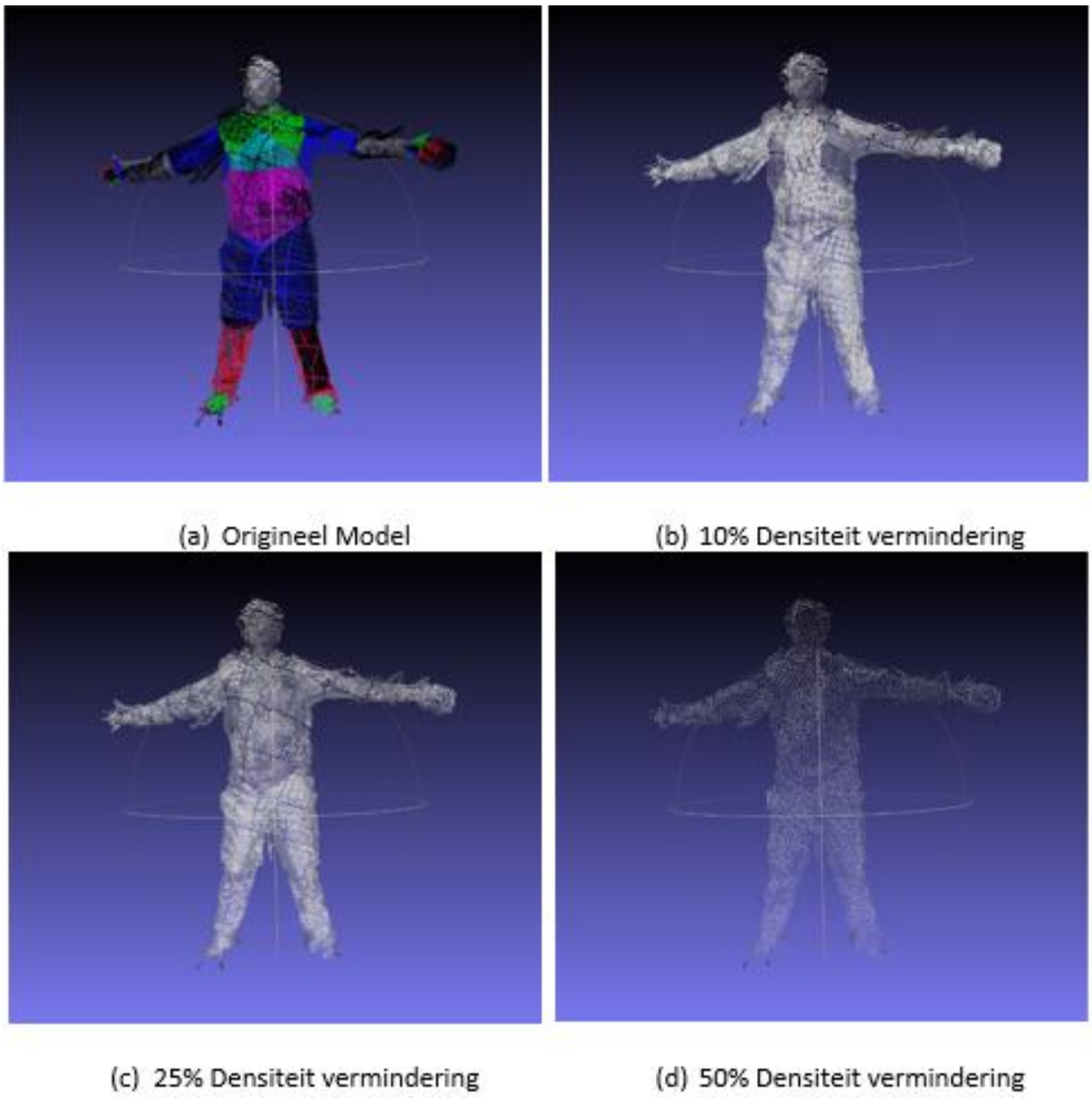
6.4.1 Meshlab

Ondanks het feit dat het aanmaken van polygonen aan de hand van puntenwolken een probleem is dat reeds door velen onderzocht en opgelost is, blijft het een complex iets. Het is namelijk zeer moeilijk om dezelfde kwaliteit van representatie te krijgen als het oorspronkelijke model. Een grote reden hiervan is dat de densiteit van polygonen afhankelijk is van de densiteit van het aantal punten in de puntenwolk. Tijdens het gebruik van Meshlab werd reeds snel duidelijk dat het aantal polygonen veel te hoog was in de buikstreek en daarom veel artefacten tevoorschijn kwamen. In het gezicht waar het aantal polygonen hoog moet zijn, was dit echter minder een probleem. MeshLab is een open source systeem voor het aanpassen van 3D meshes. Daarnaast laat het ook toe om puntenwolken te analyseren en enkel functies op toe te passen. Een lijst van alle functionaliteiten, die worden aangeboden, kan worden gevonden op de webpagina van MeshLab (meshlab.sourceforge.net). De interessante functies voor Skeleton Mapping zijn echter beperkt.

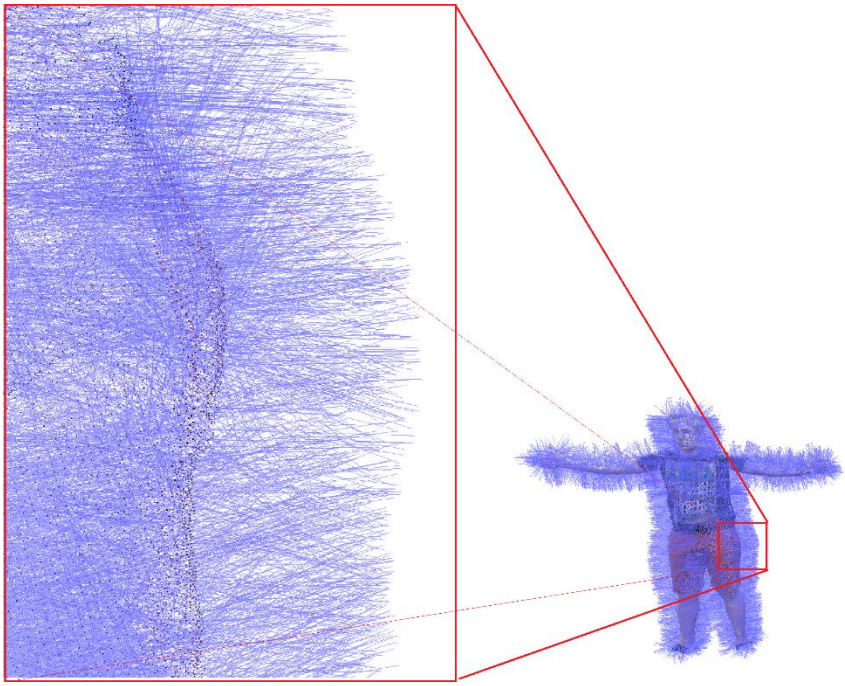


Figuur 66: Densiteit van een puntenwolk gegeneerd na 25 frames van Skeleton Mapping. Op de onderste afbeelding is ook duidelijk zichtbaar hoe klein de variaties zijn van een bepaald punt. (Afbeeldingen gegenereert door MeshLab)

Om dit eerste probleem aan te pakken, is het de bedoeling dat er wordt gekeken naar een verlaging van het aantal punten. Dit zorgt ervoor dat er minder polygonen moeten worden bekeken en dat een fout punt minder buren heeft. Dit laatste zorgt ervoor dat Poisson minder snel een fout punt als correct gaat aanzien. Het grootste probleem met de densiteit van de puntenwolken is duidelijk in Figuur 66. Op het onderste beeld is het duidelijk dat er enorm veel punten bij elkaar liggen die eigenlijk eenzelfde punt aanduiden. Er zitten echter wel enkele kleine afwijkingen op in de diepte. Het Poisson algoritme, zoals geïmplementeerd door MeshLab, is sterk onderhevig aan deze kleine verschillen. Tijdens de implementatie door middel van PCL bleek dit minder het geval te zijn (zie volgende sectie). Gezien de densiteit en de verschillen krijgen we een gigantisch groot aantal polygonen die geen effen oppervlak gaan genereren. Daarom is het nodig deze data eerst te analyseren om een betere visualisatie te kunnen maken.



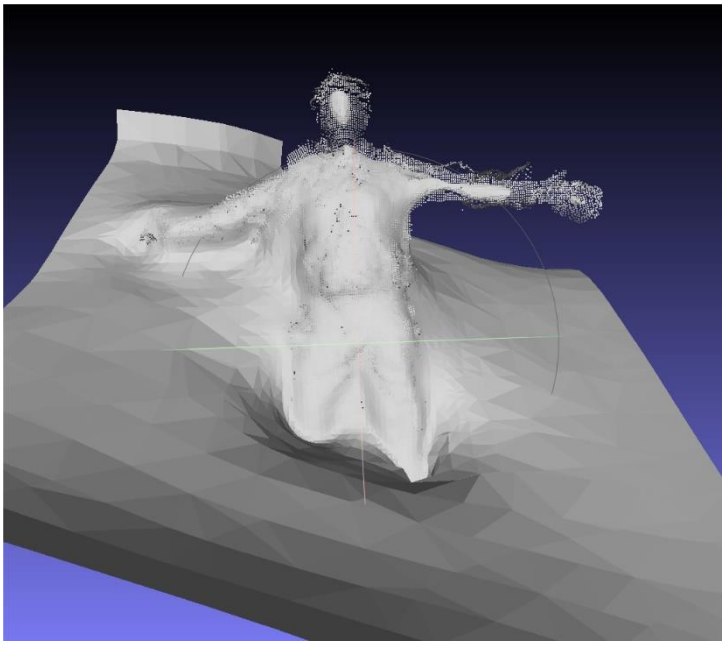
Figuur 67: Vergelijking tussen het origineel en 3 verschillende densiteit instellingen voor het samenvoegen van punten die dicht bij elkaar liggen.



Figuur 68: Genereren van normalen op een puntenwolk (Afbeelding gegenereert door Meshlab)

Een eerste, en belangrijk functionaliteit van Meshlab is het verwijderen van gedupliceerde punten. Zoals eerder vermeld is de densiteit veel te hoog om direct polygonen te construeren of zelfs surface reconstruction te doen. Indien er eenmaal gebruik is gemaakt van een van de zogenaamde “Mesh Cleaning Filters” is het model eenvoudig te analyseren en ook eenvoudiger te reconstrueren. In Figuur 67 wordt dit duidelijk weergegeven doormiddel van het verlagen van de densiteit. Deze vermindering in densiteit wordt gedaan aan de hand van een filter in Meshlab. De meeste functies ,die worden aangeboden in MeshLab, kunnen door middel van parameters worden aangepast. Om een optimaal resultaat te vinden worden meerdere opties bekeken en onderling vergeleken. Op deze manier wordt de best mogelijke instelling genomen voor elke functie. De eerste functie die wordt gebruikt, is: “Merge Close Vertices”, wat punten die dicht bij elkaar liggen, zal samennemen. Dit is noodzakelijk om fouten uit de dieptescanning te halen (de kleine afwijkingen) en om ervoor te zorgen dat er geen polygonen worden aangemaakt, die niet zichtbaar zijn omdat ze te klein zijn. De parameter voor deze functie bepaalt hoe zwaar de puntenwolk wordt uitgedund. Hoe hoger de factor, hoe meer punten worden verwijderd. Er worden 3 verschillende parameters weergegeven voor de gebruikte dataset. Het is belangrijk te weten dat voor elke dataset, die wordt gebruikt ,er een andere parameter nodig is voor het uitdunnen van de puntenwolk. Dit is noodzakelijk aangezien voor bepaalde datasets er meer punten worden geregistreerd.

Na het uitdunnen van het model wordt er een minder dichte puntenwolk teruggevonden. In het gebruikte voorbeeld is het duidelijk zichtbaar dat er bij een vermindering van 50% er nog steeds voldoende punten zijn voor het aanmaken van een mesh. Zoals is aangeduid in het onderzoek voor Surface Reconstruction, is het nodig dat een puntenwolk georiënteerde punten bevat. Hiervoor moet er een normale berekening worden gemaakt voor alle punten in de dataset. Er wordt gebruik gemaakt van een andere filter binnen MeshLab. Deze laat toe



Figuur 69: Uiteindelijk resultaat na 25 frames te combineren door middel van Marching Cubes (Afbeelding gegenereert door Meshlab)

om de volledig set te voorzien van normalen. Er kan enkele keren over het model worden gegaan met een smoothing filter op deze normalen. Zo wijzen ze voor een bepaald vlak steeds in de correcte richting. Het resultaat van deze stap is terug te vinden in Figuur 68.

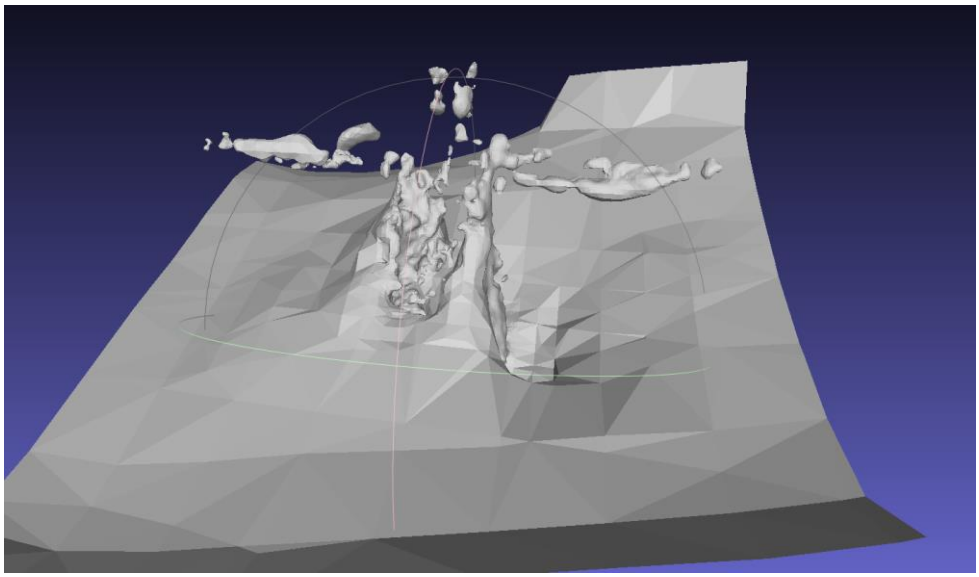
Eenmaal de normalen zijn bepaald, kan er een mesh worden aangemaakt. Er zijn echter enkele problemen met het genereren van deze mesh. De meeste krachtige algoritmen, zoals Poisson, verwachten een gesloten dataset. Dit wil niet zeggen dat de data geen gaten mag hebben, maar wel dat er vanuit wordt gegaan dat er zowel een voorkant als een achterkant aan het model is. Dit komt voort uit de wiskundige aard van de reconstructie, die namelijk gebeurt aan

Figuur 70: Poisson reconstructie op een niet-gesloten dataset. (Afbeelding gegeneert door MeshLab)

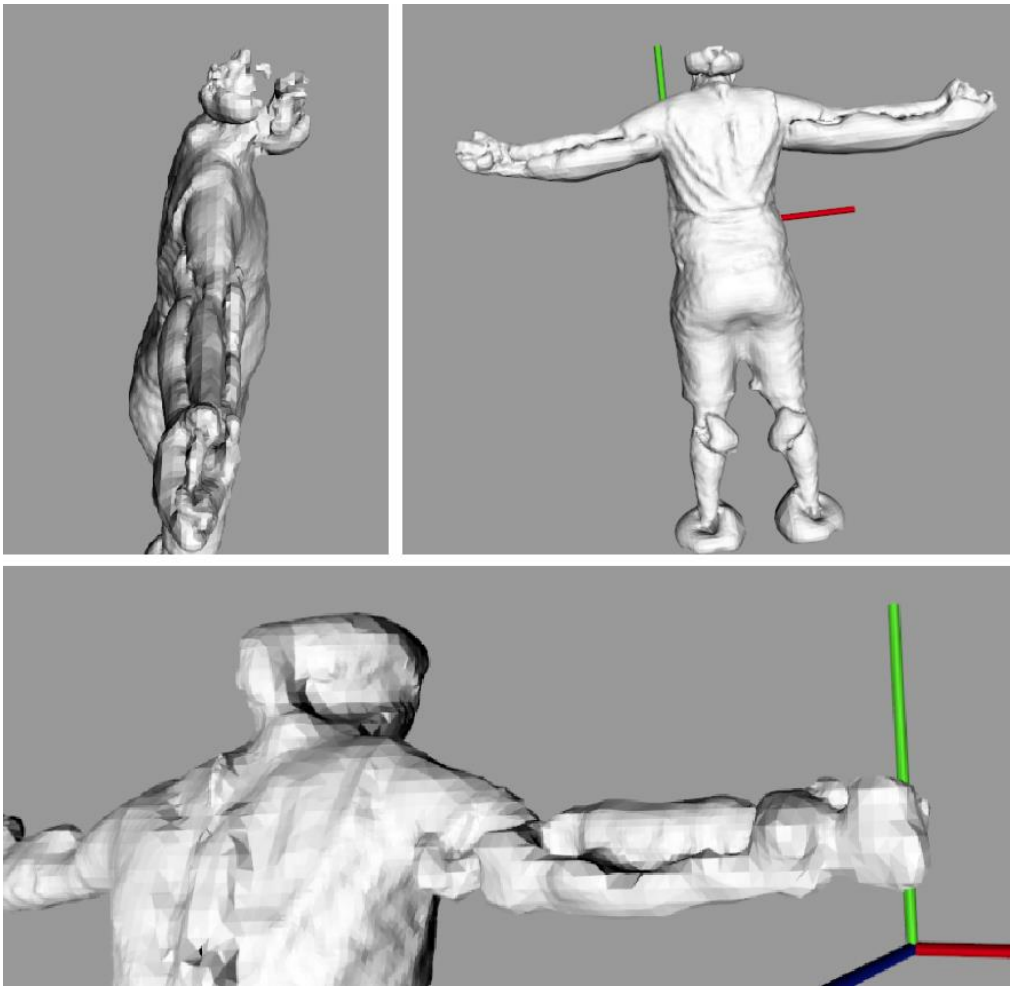
de hand van formules en trendlijnen. Indien er echter zeer grote stukken data missen, zoals de achterkant van het object zorgt dit voor zeer ernstige trendlijnfouten. Een voorbeeld hiervan kan worden terug gevonden in Figuur 70. In deze figuur wijkt de reconstructie door middel van Poisson enorm af van de puntenwolk die werd gebruikt voor de representatie te maken. Dit is te wijten aan het ontbreken van data in de achtergrond van het object. Indien er een mesh moet worden aangemaakt voor een niet gesloten dataset moet er dus gebruik worden gemaakt van een ander algoritmen. Een mogelijkheid is gebruik te maken van een MLS (zie sectie 3.2.1) gebaseerde aanpak genaamd Marching Cubes. Deze techniek zit in Meshlab en is daarom interessant om te bekijken. Het resultaat hiervan wordt weergegeven in Figuur 69. Het is hier duidelijk dat ieder punt in de dataset met elkaar wordt verbonden. Dit zorgt ervoor dat ruis zeer sterk aanwezig is in het uiteindelijke model. Zo zien ze diepteverschillen over het volledige lichaam en zijn er delen aan de heup en het hoofd die volledig verkeerd worden aangemaakt. Iets wat te verwachten is, aangezien de gebruikte techniek niet voorzien is op een dataset met veel ruis. Het grote probleem met Meshlab is het feit dat de werking van het Poisson algoritmen niet kan worden aangepast aan de hand van de grootte van het object. Dit zorgt ervoor dat voor de dense puntenwolk, die wordt aangeboden, geen correct resultaat wordt gegenereerd. Wel toont het aan dat enkele problemen binnen Meshlab persistent zijn. Zo lost het vervolledigen van het model niets op, in tegenstelling, het zorgt ervoor dat het model nog slechter wordt aangemaakt. Het tegenvallend resultaat door het gebruik van Meshlab kan worden teruggevonden in Figuur 71.

6.4.2 PCL Library

Omdat Meshlab niet het gewenste resultaat levert, is er geopteerd voor een eigen implementatie te maken aan de hand van het door PCL geleverde algoritme. Dit kan enkele problemen oplossen. Zo is de dataset, die wordt gebruikt door Meshlab ook aangemaakt door PCL, waardoor enkele waarden verschillende geschaald kunnen zijn. Zo wordt bij PCL rechtstreeks gebruik gemaakt van de VTK library in plaats van de VCG library. De verschillen tussen beide implementaties kan zorgen voor eventuele fouten in het resultaat. Bij de



Figuur 71: Generatie door meshlab op een volledige puntenwolk. (Afbeelding gegenereert door Meshlab)

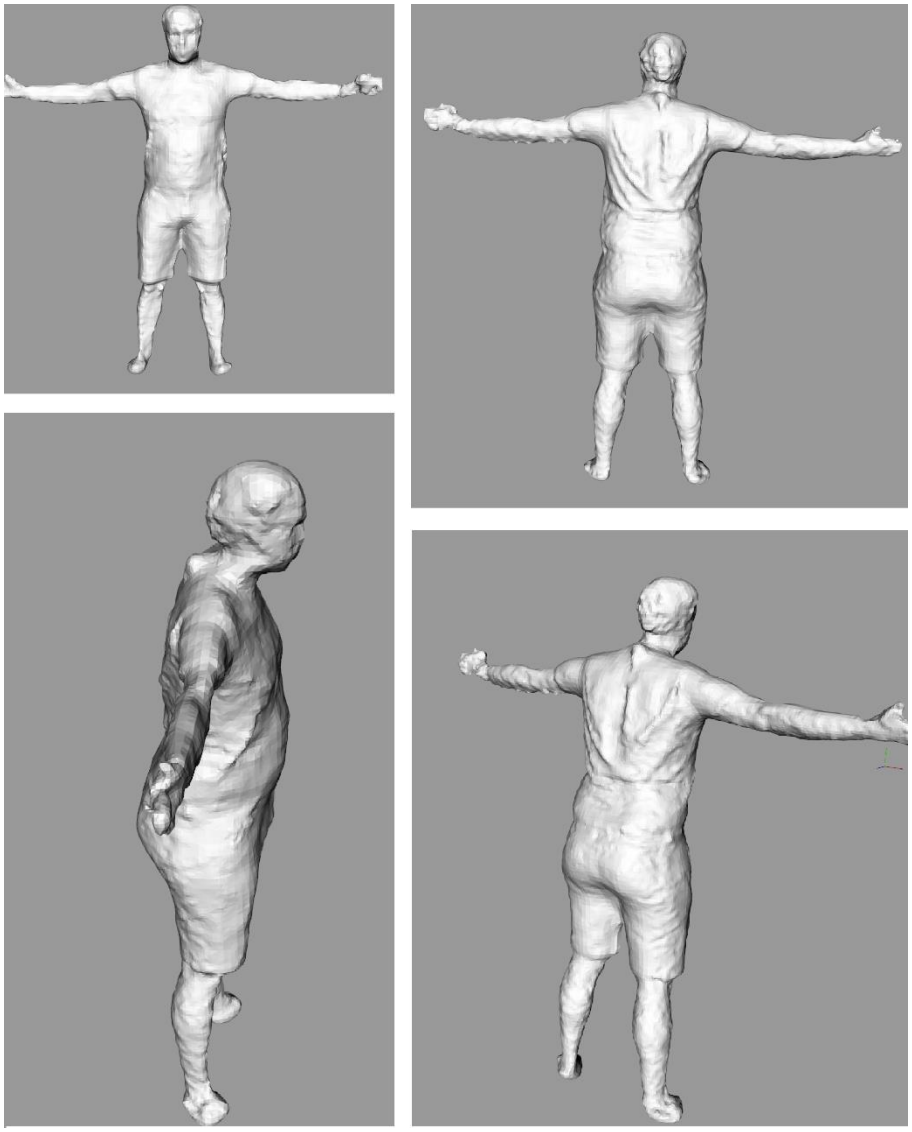


Figuur 72: Fouten in de visualisatie indien er globaal normalen worden berekend

implementatie door Meshlab moeten er enkele elementen zelf worden verwerkt. De PCL library biedt echter een oplossing voor alle delen die gebruikt zijn tijdens de verwerking van de puntenwolken door Meshlab.

Net zoals bij Meshlab wordt er gebruik gemaakt van normalen voor het Poisson algoritme. In tegenstelling tot bij Meshlab is het mogelijk deze stap af te werken per individuele puntenwolk. De eerste aanpak, die is geïmplementeerd, gebruikt, net zoals Meshlab, de volledige puntenwolk voor het vinden van de normalen. Het bleek echter dat het resultaat dat Poisson leverde aan de hand van die normalen fout was. Zo werden trendlijnen voortgezet vanuit het bovenlichaam en zorgde een kleine fout in de hand voor het maken van een gigantische fout doorheen de arm. Deze fouten zijn te zien in Figuur 72. Het is hier duidelijk dat het hoofd slecht wordt gereconstrueerd. Vooral bij de armen onderaan in de figuur zijn de fouten duidelijk. Ook deze zijn te wijten aan het gebruik van de normalen op het volledige model.

Voor de fouten aan de hand van de normalen op te lossen, wordt er gebruik gemaakt van een individuele verwerking van de puntenwolken. Het algoritme overloopt voor iedere puntenwolk de normalen. Hierdoor worden er geen fouten doorgetrokken tussen verschillende puntenwolken. Ook de houding van een bepaalde joint heeft geen invloed meer op de normalen. Bij de vorige berekening werden de andere punten ook in de vergelijking



Figuur 73: Representatie door middel van individuele normaal berekening.

gebruikt. Indien er gebruik wordt gemaakt van deze techniek is het wel nog steeds noodzakelijk om de uiteindelijke mesh te tekenen aan de hand van de volledige puntenwolk. Dit is een noodzaak aangezien anders iedere joint als gesloten lichaam wordt gezien door Poisson. Het gevolg hiervan is dat iedere joint afgerond wordt aan de uiteinden. Indien er met deze elementen rekening wordt gehouden, wordt er een redelijk degelijk resultaat afgeleverd. Dit resultaat is te zien in Figuur 73. Er zijn enkele duidelijke verschillen tussen de twee representaties. Ook het gezicht wordt nu volledig afgewerkt. Door deze verandering is ook de nek duidelijk te onderscheiden. Daarnaast zijn ook de armen volledig afgesloten. Dit is een gevolg van de lokale berekening voor de normalen. Tot slot zijn ook de voeten minder omgeven door een bol van polygonen en geeft de knie geen artefacten meer. De verschillen tussen beide technieken is ook te danken aan het afstellen van het Poisson algoritme. Door met meer buren te werken (aangezien de normalen correct zijn), wordt er een beter resultaat gevormd. Het gebruik van Poisson in PCL geeft duidelijk betere resultaten dan MeshLab in de vorige sectie. Een grote reden hiervoor is dat de puntenwolk wordt aangemaakt en verwerkt door dezelfde library (VTK) in plaats van twee verschillende.

Figuur 74: Afgewerkte polygon mesh. Ingekleurd door elke vertex een kleur te geven.

In tegenstelling tot de puntenwolk is er echter een probleem met het kleuren van een mesh. Zoals vermeld in sectie 4.1.3 zijn er enkele technieken om dit te doen. Deze algoritmen zijn echter zeer complex voor het vinden van de textures uit een puntenwolk. Omdat de nadruk binnen de thesis ook wordt gelegd op de puntenwolk en het nagaan of het mapping algoritme een mogelijke techniek is, wordt er niet verder ingegaan op het hertexturen. Voor de implementatie wordt er gebruik gemaakt van een eenvoudig algoritmen om ervoor te zorgen dat de mesh toch van kleur wordt voorzien. Voor het inkleuren van de volledige mesh wordt er verder gebouwd op de hoge densiteit van polygonen in het model. Omdat het model zeer veel polygonen heeft ,is het een goede aanpak om elke polygon in te kleuren met de kleuren informatie van de dichtstbijzijnde punten. Hiervoor wordt er voor elke vertex gezocht naar het punt in de oorspronkelijke puntenwolk waarvoor de afstand het kleinste is. De kleur van het gevonden punt wordt dan overgebracht op de vertex. Door deze techniek te gebruiken is het mogelijk een vergelijking te maken tussen de puntenwolk weergave en de mesh representatie. Ook is de invloed van het Poisson algoritme duidelijker zichtbaar door de kleuren toe te kennen. Zo geeft het inkleuren van het vorige getoond model een degelijke representatie. Deze is te vinden in Figuur 74.

6.5 Animatie

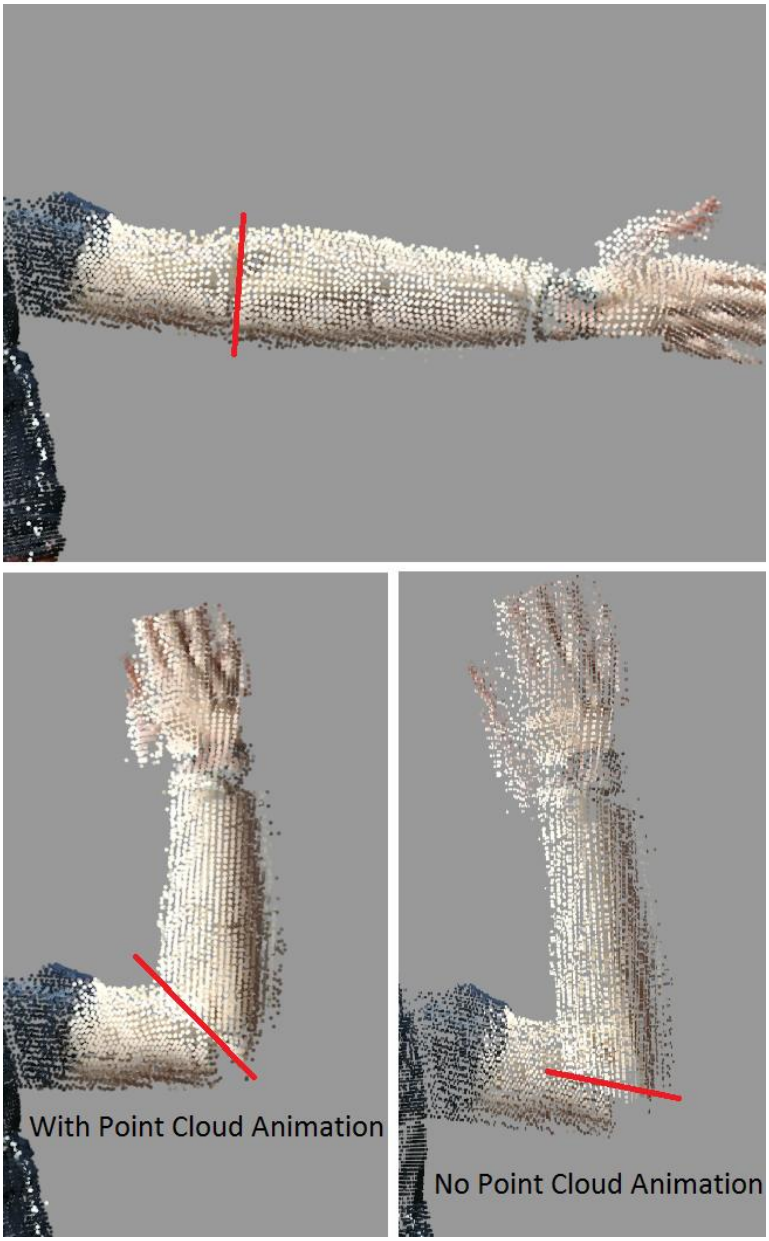
Eenmaal een volwaardige representatie is aangemaakt is het mogelijk deze te animeren. Hierbij zijn er echter enkele elementen die moeten worden bekeken. Zoals eerder vermeld in sectie 4.3 wordt er meestal gebruik gemaakt van MOCAP voor het aanmaken van animaties voor computer games. Indien er echter gebruik wordt gemaakt van Skeleton Mapping for Kinect is dit niet meer nodig. De gewenste animaties zijn namelijk opgenomen doorheen te

Figuur 75: Een remapping van een opname. De opname aan de onderkant wordt gemapped op de persoon zoals die zichtbaar is voor de Kinect (onder).

tijd. Hierdoor is het mogelijk zowel een model als de animaties gelijktijdig op te nemen. Indien dit echter niet gewenst is, is het ook mogelijk om nadien animaties op te nemen. Het programma dat is ontwikkeld, biedt meer dan één mogelijkheid voor animatie. De verschillende methodes zullen hieronder worden uitgelegd. Een belangrijke opmerking is wel dat er enkel wordt gewerkt aan de hand van puntenwolken. De focus binnen de tekst ligt op het gebied van animatie, namelijk op het animeren van de puntenwolken en niet op het animeren van meshes. De reden hiervoor is onder andere terug te vinden in sectie 4.3.4. Het animeren van meshes is een opgelost probleem, waarvoor reeds diverse technieken bestaan.

Het animeren van een karakter kan op meerdere manieren. Zo laat het programma toe opgenomen sequenties af te spelen. Daarnaast is het ook mogelijk om real-time een animatie te doen van een karakter. Beide technieken gebruiken dezelfde techniek, namelijk skelet animatie. Voor elke frame, die moet worden weergegeven, wordt een skelet opgevraagd. Dit kan worden opgevraagd binnen een bestand, van vooraf opgenomen sequenties. Of dit kan worden gevraagd aan de Kinect, waardoor real-time het karakter wordt hervormd. Eenmaal een nieuw skelet beschikbaar is, worden alle joints verplaatst en geheroriënteerd. Dit is de stap die ervoor zorgt dat de puntenwolken correct staan. Daarnaast worden de ledematen ook geschaald. Dit is nodig om ervoor te zorgen dat er een constant aansluiting is tussen de verschillende ledematen. Het is deze laatste stap, die het ook mogelijk maakt om een karakter te projecteren op een andere persoon. Aangezien alles wordt geschaald op basis van de lengte van de beenderen van de persoon, is het vrij eenvoudig om ervoor te zorgen dat het algoritme een geloofwaardig model presenteert. Dit geeft een resultaat zoals afgebeeld in Figuur 75. Hierbij is er geen zichtbaar gat tussen verschillende joints en worden de puntenwolken correct georiënteerd.

Een ander probleem tijdens de animatie is het verbinden van de puntenwolken per joint. Eerder is reeds aangetoond dat het schalen van de puntenwolken ervoor zorgt dat het beginpunt van een joint altijd samenvalt met het eindpunt van de vorige joint. Dit garandeert echter niet dat de puntenwolk altijd even ‘gesloten’ is. Hiermee wordt er gerefereerd naar het



Figuur 76: Point Cloud Animation op de elleboog.

verschijnsel, dat plaatsvindt bij het plooien van de armen. Door de armen te plooien wordt het ellebooggewricht uit elkaar getrokken. Dit zorgt ervoor dat er nu een opening wordt gevormd waar er voordien punten aanwezig waren. Om dit op te lossen moet er een techniek worden gebruikt die vergelijkbaar is met blend weights voor mesh animatie (zie sectie 4.3.4). Met deze techniek zijn er echter twee problemen verbonden met de huidige implementatie. De skeletten die traditioneel worden gebruikt bij game ontwikkeling en karakter animatie kunnen voorzien zijn van veel meer joints dan het aantal dat de Kinect levert. Ook het omgekeerde is een mogelijkheid. Dit zorgt ervoor dat er weinig connecties zijn tussen verschillende joints. Indien er geen connectie is tussen joints is het dus ook moeilijk om gewichten toe te kennen aan bewegingen. Zo is het schouderblad bij de Kinect volledig onafhankelijk van de romp. Een ander probleem is het feit dat er gigantisch veel punten aanwezig zijn in dergelijke puntenwolken. Indien alle punten moeten worden geëvalueerd, kan dit niet meer in real-time.

De oplossing hiervoor is tijdens het mapping algoritme een extra controle toe te voegen. Tijdens deze controle worden punten die op een bepaalde afstand van de joint liggen toegevoegd aan een extra verzameling. Deze verzameling geeft per lichaamsdeel aan welke punten onderhevig zijn aan de beweging van de parent- en child-joint. Hierdoor moeten niet alle punten worden overlopen in het geval van een nieuw frame. Eenmaal de mapping voorbij is, wordt er per frame een update gedaan ter animatie van de verschillende joint. Voor de punten die omvat zitten in de jointlijst wordt er echter een andere rotatie en translatie gebruikt als de joint zelf. Deze punten worden uitgestrekt tussen hun normaal punt binnen de puntenwolk en de punten van de parent. Door deze techniek toe te passen kunnen er enkele verbeteringen worden gemaakt in de elleboog en andere joints. Zoals zichtbaar in Figuur 76, is deze manier van werken echter niet perfect. Zo blijven er nog steeds gaten over. Dit is voornamelijk te wijten aan het verplaatsen van de punten. Indien er extra punten worden toegevoegd zou het mogelijk zijn een betere verbinding te maken. Hiervoor moet op elke frame een controle gebeuren die checkt welke punten terug weg moeten. Dit kan door alle punten opnieuw te mappen. Indien een punt niet meer binnen een joint valt, is dit na het bewegen van de joint toegevoegd en mag dit worden verwijderd. Het mappen van de punten kan echter niet in real-time en dus is deze techniek niet bruikbaar als vervanger voor het eerder voorgesteld algoritme. Wat wel duidelijk is op de afbeelding is dat de referentielijn van de puntenwolk duidelijk wordt schuin getrokken indien de animatie wordt gebruikt.

Naast de mogelijkheid om een skelet te animeren, zijn er nog enkele voordelen aan de gebruikte techniek. Zoals eerder vermeld bij het concept van het algoritme, zorgt het streamen van puntenwolken voor een enorme datastroom over het netwerk. Door echter gebruik te maken van SMK wordt deze datastroom gevoelig kleiner. Hierbij is het mogelijk om een threshold in te stellen voor de kwaliteit van het object. Deze threshold zal gaan over de densiteit van de puntenwolk. Indien in bepaalde gebieden reeds voldoende informatie doorgestuurd is, kan in de volgende frames deze data worden genegeerd bij het doorsturen. Enkel relevante data, in gebieden die nog niet aan de gewenste densiteit zitten, moet worden doorgestuurd. Hierbij kan er eventueel gebruik worden gemaakt van updates van bepaalde ledematen. De updates worden dan gestuurd op bepaalde intervallen. Dit is mogelijk nodig omdat de kwaliteit van de puntenwolk blijft verbeteren waardoor er meer en meer detail terugkomt in het object. De intervallen worden naarmate de stream langer loopt ook groter, aangezien de verschillen tussen frame 1 en 10 groter zijn als die tussen 1000 en 1010. Eenmaal deze data is doorgestuurd, moet enkel nog gebruik worden gemaakt van de skelet data. Deze kan worden doorgestuurd door middel van eenvoudige updates op een vast interval. Er kan hier zelfs met zo genaamde I en P frames worden gewerkt, zoals ook het geval is bij videocompressie. Voor meer informatie omtrent deze techniek voor het encoderen van frames, wordt er verwezen naar (Schwarz, Marpe, & Wiegand, 2007).

6.6 Vergelijkingen

Om te kunnen valideren hoe geslaagd de voorgestelde techniek en de daarbij horende implementatie zijn, moeten beide elementen worden vergeleken met andere reeds gekende technieken of implementaties. Omdat technieken reeds eerder werden vergeleken in de domeinstudie zal dit hoofdstuk zich toespitsen op verschillende implementaties. Er zijn enkele

eigenschappen die worden vergeleken tussen de verschillende toepassingen. Een eerste is *performantie*. Hiervoor wordt er een onderscheidt gemaakt tussen real-time algoritmen die alle verwerkingen kunnen doen van alle aangeboden frames of dataprocessing algoritmen die pas na het volledig afronden van de captatie beginnen aan het verwerken van frames (het is mogelijk op hetzelfde moment te starten waarbij beide processen los van elkaar werken en de captatie dus niet moet wachten op het verwerken van de data). Een tweede vergelijking die wordt gemaakt, is *ruis* in het uiteindelijke model. Dit wordt niet enkel beïnvloed door de processen maar ook door de captatie. Aangezien sommige implementaties niet eenvoudig aan te passen zijn naar andere hardware is ook deze vergelijking nuttig. Het is hierbij belangrijk om vooral op te letten hoe ruis het oppervlakte van de mesh verandert. Een volgende vergelijking die moet worden gemaakt is de *snelheid van convergentie*. Hierbij wordt gekeken naar hoeveel frames/tijd een implementatie nodig heeft om gaten te dichten in het model. Om deze vergelijking te maken wordt er gekeken naar de progressie van het oppervlak in de tijd. Daarnaast moet er ook worden gekeken naar *artefacten* gegenereerd door het algoritme. Bepaalde algoritmen maken gebruik van technieken, die zelf voor veranderingen in data zorgen en die op hun beurt artefacten genereren in het uiteindelijk model. Tot slot is het ook nodig de *kwaliteit* van de verschillende algoritmen te vergelijken. Bepaalde algoritmen geven een grafisch kwalitatief model af, terwijl andere zo snel mogelijk een degelijke oplossing proberen te maken.

Voordat het mogelijk is om implementaties te vergelijken, is het nodig deze te bespreken zodat de algemene werking van deze implementaties duidelijk zijn alsook het toepassingsdomein van de techniek. Er zijn veel verschillende algoritmen die modellen proberen te creëren aan de hand van gecapteerde data. Daarom is het doel van de uiteindelijke data niet altijd hetzelfde. Voor sommige aanpakken ligt de focus op het genereren van hoge kwaliteit meshes waar andere algoritmen focussen op het animeren van de verkregen 3D data. Naast deze voorbeelden zijn er nog talloze andere toepassingen voor deze modellen. Omdat elke implementatie een ander doel heeft, en dus een andere werking heeft, zal elke implementatie worden besproken om zo meer duidelijkheid te geven tussen de verschillen en gelijkenissen tussen de software. Het is belangrijk op te merken dat alle technieken op het moment van schrijven nog steeds in ontwikkeling zijn. Hierdoor zijn er geen programma's beschikbaar om een vergelijking mee te maken. Daarom wordt er vergeleken op basis van concepten en verkregen output. Er zal dus geen zelfde dataset worden gebruikt voor alle technieken.

6.6.1 Quaternion Software – Q3D

Q3D is een software pakket ontwikkeld in België door Quaternion Software. Het pakket stelt gebruikers in staat om eenvoudig Kinect-puntenwolken te synchroniseren en samen te voegen tot één datastroom. Hierbij moet een gebruiker zelf de synchronisatie doen.

De algemene workflow werkt als volgt. Eerst worden er meerdere Kinect datastromen gecreëerd. Daarna worden de verschillende datastromen manueel gesynchroniseerd en gealigneerd. Tot slot worden de datastromen gecodeerd zodat er één datastroom overblijft die eenvoudig kan gebruikt worden. Deze datastromen kunnen dan eenvoudig worden geïmporteerd in editors zoals Unity om eenvoudig een scene te maken.

Q3D is bedoeld om eenvoudige 3D opnames te kunnen tonen in een 3D scene. Er moet namelijk maar één opname worden gemaakt met enkele camera's en het bewegend 3D model is klaar. Er zijn echter enkele nadelen aan het samenvoegen van deze scenes. Het aligneren wordt manueel gedaan waardoor er ook hier kleine fouten kunnen voorkomen. Al kunnen deze worden weggewerkt indien er gewoonweg een tweede maal wordt gealigneerd. Een groter probleem zijn de artefacten die duidelijk zichtbaar zijn op de plaats waar twee puntenwolk elkaar ontmoeten. Zoals te zien in Figuur 77 is dit vooral het gevolg van de afwijkingen op de randen van de puntenwolk (vanuit camera standpunt bekeken). Het voordeel aan deze techniek is dat deze zeer snel is en weinig computationele kracht vereist om een model te maken. Ook is de kwaliteit zeer consistent (op de artefacten na) waardoor een zeer mooi model kan worden afgeleverd. Ook ruis kan eenvoudig worden weggewerkt omdat dit frame per frame kan worden geanalyseerd.

Om te weten of Skeleton Mapping een goede aanpak is, kunnen we enkele punten vergelijken met Quaternion Software 's Q3D. Het eerste element dat wordt besproken is performantie. Alhoewel dat beide algoritmen geen output leveren tijdens captatie (zie real-time) is er toch een duidelijk verschil in de performantie tussen beiden algoritmen. Q3D is een beduidend snellere oplossing voor het genereren van 3D modellen. Dit komt omdat bij Q3D enkel een synchronisatie stap nodig is om een volwaardig model te maken, terwijl er bij Skeleton Mapping veel meer informatie wordt geabstraheerd van de captatie-stromen geleverd door de hardware. Dit zorgt ervoor dat Skeleton Mapping een veel grotere verwerkingstijd heeft en dus na captatie een lange tijd moet verwerken voor een resultaat. Q3D heeft direct een resultaat van het moment dat de synchronisatie tussen beide stromen is gebeurd.

Het tweede element dat wordt vergeleken is ruis. Zoals eerder aangetoond in Figuur 77, is er toch redelijk wat ruis te vinden op de snijlijn van de puntenwolken. Indien we dit vergelijken met de ruis die wordt gegenereerd bij Skeleton Mapping zien we duidelijk hetzelfde probleem terugkomen. Punten op de randen van het object (zoals de zijkant van het lichaam in de boven vermelde afbeelding) hebben vaak foute diepte-informatie zodat er fouten worden gemaakt in de reconstructie. Op het gebied van artefacten is Q3D dan wel veel consistenter. Omdat er geen volledige Surface Reconstruction van het oppervlakte wordt gedaan, heeft de ruis geen extra invloed in het maken van meer fouten in het object. Bij Skeleton Mapping wordt echter gebruik gemaakt van alle punten om een nieuw oppervlakte te creëren. Hierdoor zorgt de ruis voor extra artefacten doorheen het proces van Skeleton Mapping.

Een ander zeer belangrijk aspect is de snelheid van convergentie. Hoe lang heeft het object nodig om tot een volwaardig 3D model te komen? Bij Q3D is er geen convergentie tijd aangezien er geen gebruik wordt gemaakt van opeenvolgende scenes om de kwaliteit van de datastroom te verhogen. Bij Skeleton Mapping zien we dit echter wel gebeuren. De convergentie tijd is echter wel sterk afhankelijk van de scene en de bewegingen van het model. Hoe meer het model beweegt hoe korter de convergentie tijd gaat zijn om alle punten te verzamelen. Meestal duurt dit enkele frames om één bepaalde kijkrichting correct te mappen. Dit zorgt ervoor dat om een volledig object te komen, toch enkele minuten nodig kunnen zijn.

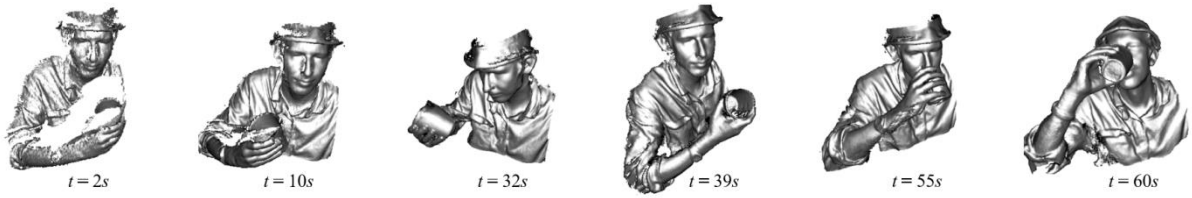
Figuur 77: Puntenwolk gegenereerd door Q3D. (Afbeelding van Quaternion Software 's website)

Tot slot moet ook de kwaliteit van het model worden bekeken. Met de huidige implementatie (en de problemen die deze implementatie met zich meebrengt) geeft Skeleton Mapping nog geen kwalitatief goed resultaat. Verder onderzoek naar de reconstructie van het algoritmen en het oplossen van de verschillen in diepte kan dit echter verbeteren. Het concept (samen met de wiskundige achtergrond van de rotaties etc.). geeft echter een beter resultaat dan Q3D. Dit komt omdat Q3D geen verhoging van kwaliteit creëert doorheen de verschillende frames. Hierdoor blijft de oorspronkelijke captatie kwaliteit geldig doorheen alle frames

6.6.2 Dynamic Fusion

Tijdens het schrijven van deze thesistekst is er een grote doorbraak gekomen in het mapping van dynamische scenes. Zo is door de Universiteit van Washington (in samenwerking met Google en Intel Science and Technology Center for Pervasive Computing) het eerst real-time SLAM-algoritme ontwikkeld voor dynamische scenes. DynamicFusion heeft alle voordelen die zijn besproken in verband met Skeleton Mapping, met daarnaast nog het voordeel dat het onafhankelijk is van een skelet. DynamicFusion is in staat om alle dynamische scenes te mappen door middel van warping en canonieke velden. Voor meer informatie wordt er verwezen naar (Newcombe, Fox, & Seitz, n.d.).De enige nadelen die DynamicFusion heeft ten opzichte van het voorgestelde Skeleton Mapping algoritme bevinde zich in het gebied van

animatie. Door middel van de mapping van de 3D mesh op een skelet is het eenvoudig deze te animeren. Men hoeft slechts het skelet te animeren en het model volgt vanzelf. Bij DynamicFusion is dit wat omslachtiger aangezien dit werkt aan de hand van warping.



Figuur 78: Convergentie van het model doorheen de tijd, gebruik makende van Dynamic Fusion. (Newcombe et al., n.d.)

Het is eenvoudig een vergelijking te maken tussen DynamicFusion en Skeleton Mapping aangezien ze beide focussen op hetzelfde, namelijk het verbeteren van een 3D model over tijd om zo tot een hoge kwaliteit 3D mesh te komen. Net zoals bij de andere oplossing wordt er eerst gekeken naar performantie. Zoals aangehaald in de korte introductie van DynamicFusion is dit het eerste real-time algoritme voor dynamische-SLAM. Hierdoor biedt het heel wat voordelen ten opzichte van Skeleton Mapping. Dit komt onder anderen omdat er geen gebruik wordt gemaakt van Skeleton voor mapping maar er gebruik wordt gemaakt van warping om bepaalde weergaven te verkrijgen.

In betrekking tot ruis doet DynamicFusion een perfecte zaak. In zowel de paper als de demo is er geen ruis te detecteren na 'afloop' van het algoritme. Hiermee wordt bedoelt wanneer convergentie is bereikt. Daarnaast slaagt het erin een bijna perfecte reconstructie te maken. De ruis, die in het begin wel zichtbaar is, wordt volledig weg gewerkt. Dit gaat gelijk op met de convergentie van het model. Zoals in Figuur 78 zichtbaar is, duurt het ongeveer tot aan het beeld van 39 seconden om een 3D model te genereren zonder echte ruis. Het model heeft een zeer hoge kwaliteit en convergeert zeer snel. Zelfs na 10 seconden zien we duidelijk een verhoging van de kwaliteit, al zijn er nog niet voldoende zijden van het object (in dit geval een persoon) bekeken door de camera. Hierbij is de convergentie vergelijkbaar met die van

Figuur 79: Hoge kwaliteit reconstructie door middel van DynamicFusion.(Newcombe et al., n.d.)

Skeleton Mapping. Dit is logisch aangezien elke frame evenveel data verleent aan DynamicFusion als aan Skeleton Mapping.

Tot slot is er nog de kwaliteit van het model dat kan worden bekeken. Hier wordt snel duidelijk dat DynamicFusion een zéér krachtige methode is voor het maken van deze 3D modellen. De kwaliteit van object is enorm hoog. In Figuur 79 zien we ook dat elk detail wordt opgenomen (zoals de haren) en dus niet het gevolg is van een goed reconstructie algoritme maar van het goede aanmaken van het 3D model.

6.6.3 Discussie

Indien alle resultaten worden vergeleken met elkaar is het zéér duidelijk om een conclusie te trekken. Skeleton Mapping for Kinect geeft een nieuwe aanpak naar het mappen van dynamische scenes waarbij de aandacht wordt gelegd op het aanmaken van een beweegbaar karakter. Op het gebied van kwaliteit is het echter niet vergelijkbaar met het pas ontwikkelde DynamicFusion algoritme. Al ontbreekt ook aan dit algoritme enkele functionaliteiten die wel beschikbaar zijn bij Skeleton Mapping. Voor game-development toepassingen, waar karakters moeten worden geanimeerd en kwaliteit niet van het aller hoogste niveau moet zijn (omdat dit grafisch niet gerenderd kan worden), heeft Skeleton Mapping een goede oplossing. Voor het ontwikkelen van een 3D model vanuit een dynamische scene is DynamicFusion een ideaal algoritme. Het is duidelijk dat het algoritme van Skeleton Mapping nog niet optimaal is en nog enkele fouten bevat. Deze fouten, vooral ruis gebonden, moeten er in de toekomst nog worden uitgehaald om ervoor te zorgen dat de kwaliteit van het object wordt verbeterd.

7.Conclusie

Deze thesis bespreekt de concepten die worden gebruikt voor het digitaliseren van objecten. Het eerste wat werd besproken ging over de algemene pipeline die wordt gebruikt bij het aanpakken van dit probleem (hoofdstuk 2). Hierin worden al twee verschillende voorstellen gedaan voor het aanpakken van het streamen van deze 3D modellen tijdens captatie. Hier is duidelijk dat er meerdere mogelijke aanpakken zijn voor een bepaald probleem op te lossen, iets wat doorheen de hele thesis duidelijk naar voorkomt. Naderhand gezien is het gebruik van deze pipeline compleet afhankelijk van de toepassingen waarin wordt gewerkt. Bepaalde toepassingen waar een low-budget toestel wordt gebruikt voor het weergeven van bepaalde 3D modellen opteren voor het renderen van deze beelden op een centrale server, waarbij enkel de output moet worden gedeeld. Andere toepassingen, die een zo kort mogelijke delay mogen hebben, zullen dan eerder op het device zelf renderen. Er is sinds het schrijven van deze thesis echter ook meer informatie bekend geraakt over systemen die hybride systemen willen gebruiken voor rendering te doen. Een bekend voorbeeld hiervan op de commerciële markt is de Xbox One van Microsoft. Deze maakt gebruik van Kahawai (Cuervo et al., n.d.), een techniek ontwikkelt door Microsoft Research in samenwerking met Duke University en Washington University.

In het verdere verloop van de domain study worden algemeen bekende algoritmen besproken voor een totaalbeeld van het onderzoeksgebied te creëren. De meeste van deze technieken bestaan reeds enorm lang, waardoor er veel verschillende optimalisaties zijn gebeurd alsook verschillende varianten zijn ontwikkeld. Dit maakt het moeilijk om alle mogelijke oplossingen te bespreken. Daarom zijn enkel de algemene algoritmen besproken, die als meest generieke oplossingen worden gebruikt voor bepaalde problemen. Dit heeft het echter moeilijk gemaakt om tijdens de implementatie het correcte algoritme te kiezen. Omdat de kleine varianten, die zijn ontwikkeld voor specifieke toepassingen, niet zijn vergeleken, moest er een misschien minder performante of minder geschikte oplossing worden gekozen omdat er niet voldoende kennis over de andere aanwezig was. Dit probleem komt vooral terug bij surface reconstruction. In het laatste gedeelte aangaande Data Visualisatie (hoofdstuk 4) wordt er minder aandacht besteed aan bepaalde algoritmen. Deze keuze werd gemaakt omdat de meeste algoritmen voor het renderen van objecten zo fel ingeburgerd zijn dat deze triviaal zijn geworden. Hierbij wordt vooral gedacht aan het gebruik van DirectX of openGL, bibliotheken die door heel de wereld worden gebruikt voor vrijwel alle games die beschikbaar zijn op de markt. Ook in verband met netwerken wordt er enkel kort besproken welke invloeden er kunnen zijn. Het is namelijk altijd nodig een specifieke aanpak te ontwikkelen per toepassing. Indien er data wordt verstuurd over het internet, onafhankelijk van de connectie die wordt gebruikt, moet er altijd optimalisatie gebeuren voor het specifieke object dat wordt verstuurd. Het is daarom dus onmogelijk een algemene aanpak neer te schrijven in termen van deze thesis.

In het laatste deel van de thesis wordt er dan een eigen voorstel gemaakt om enkele vastgestelde problemen op te lossen. De problemen, die werden vastgesteld betreffen de bandbreedte die nodig is bij het versturen van puntenwolken of gecapteerde 3D meshes.

Daarnaast was het ook te omslachtig om een nieuw karakter te creëren voor een bepaald spel. Hiervoor moest alles opnieuw worden getekend en ontworpen door een 3D artiest. De bedoeling van Skeleton mapping was dit op te lossen door middel van skelet gebaseerde captatie. Het concept van het algoritme belooft inderdaad een oplossing aan al deze problemen, al blijkt de implementatie een pak moeilijker te realiseren als eerst verwacht. Zo zijn er vooral problemen met de verwerkingssnelheid van de verkregen data. Al zou het mogelijk zijn nog enkele optimalisatie technieken door te voeren (zoals het gebruik maken van GPU's voor de verwerkin van de data), toch blijft de implementatie wat achter op de verwachten resultaten. Ook de kwaliteit van de verkregen 3D meshes schiet te kort ten opzichte van andere technieken (zie DynamicFusion). Bij de vergelijking wordt snel duidelijk dat er nog enkele elementen missen aan het concept waardoor het op dit moment nog niet klaar is om direct te worden gebruikt in toepassingen. Er zullen nog enkele elementen moeten veranderen aan zowel het concept, de implementatie als de mogelijke hardware. Deze aanpassingen worden besproken in de volgende sectie.

In het verloop van de thesis zijn meerdere aanpakken besproken en is er uiteindelijk een implementatie gemaakt als concept voor Skeleton Mapping. Zoals vermeld in de vergelijkingen zijn er recent enkele werkende producten uitgekomen die dit probleem oplossen. Daarom zal het werk dat in de toekomst nog wordt verwacht zich enkel toespitsen op de gegeven implementatie in plaats van het volledige onderzoeksgebied. Er zullen enkele punten worden besproken die deze technologie toegankelijker moeten maken voor consumenten of voor grootschalig gebruik binnen de commerciële wereld (als toepassingen voor de gebruikers).

Een eerste probleem dat moet worden aangepakt is preciezere hardware tegen een lagere prijs. Op dit moment zijn er zeer nauwkeurige toestellen die de gebruiker in staat stellen om zeer hoge kwaliteit data te verkrijgen. Hierbij is er echter wel nog steeds veel ruis. Zeker op goedkopere hardware is ruis een zeer groot probleem voor deze soort toepassingen (zie toepassingen die zich richten op grafische nauwkeurigheid). Daarnaast moet er voornamelijk worden gekeken naar betere algoritmen om ledematen te herkennen. Iets wat in de huidige implementatie niet krachtig of consistent genoeg is, zijn de rotaties van ledematen. Hierdoor is het moeilijk om een algoritme te bouwen bovenop deze informatie. Voor een verdere ontwikkeling moet er worden gekeken naar het maken van consistente algoritmen voor het vinden van deze rotaties.

Het mapping algoritme zelf is wiskundig correct, maar er moet worden gekeken naar de ledemaat-herkenning. Het is moeilijk te weten bij welke ledematen bepaalde punten horen omdat kledij hier een verhulling voor kan zijn. Hierdoor is het nodig om dit algoritme verder uit te breiden zodat er accurater kan worden gevonden bij welk lichaamsdeel iets hoort als ook het gebruik van meerdere joints kan worden ondersteunt.

8.Referenties

Berger, M., Alliez, P., Tagliasacchi, A., Seversky, L. M., Silva, C. T., Levine, J. A., & Sharf, A. (2014). State of the Art in Surface Reconstruction from Point Clouds.

Berger, M., Levine, J. a., Nonato, L. G., Taubin, G., & Silva, C. T. (2013). A Benchmark for Surface Reconstruction. *ACM Transactions on Graphics*, *32*(2), 20:1–20:17. doi:10.1145/2451236.2451246

Bernardini, F., & Rushmeier, H. (2002). The 3D Model Acquisition Pipeline, *21*(2), 149–172.

Besl, P. J., & McKay, H. D. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *14*(2), 239–256. doi:10.1109/34.121791

Brook, S., & Alegre, P. (n.d.). A Hole-Filling Strategy for Reconstruction of Smooth Surfaces in Range Images.

Brunton, A., Wuhrer, S., Shu, C., Bose, P., & Demaine, E. D. (2014). Filling holes in triangular meshes by curve unfolding.

Cohen, a., Zach, C., Sinha, S. N., & Pollefeys, M. (2012). Discovering and exploiting 3D symmetries in structure from motion. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1514–1521. doi:10.1109/CVPR.2012.6247841

Corp, S. P., & Systems, I. (1995). FAST VOLUME RENDERING USING A SHEAR-WARP FACTORIZATION OF THE VIEWING TRANSFORMATION Philippe G . Lacroute Technical Report : CSL-TR-95-678 September 1995 OF THE VIEWING TRANSFORMATION, (September).

Cuervo, E., Wolman, A., Cox, L. P., Lebeck, K., Razeen, A., Saroiu, S., & Musuvathi, M. (n.d.). Kahawai : High-Quality Mobile Gaming Using GPU Offload Categories and Subject Descriptors.

Davis, J., Marschner, S. R., & Levo, M. (n.d.). Filling Holes in Complex Surfaces using Volumetric Diffusion.

Elseberg, J., Borrmann, D., & Nuchter, A. (2011). Efficient processing of large 3D point clouds. *2011 XXIII International Symposium on Information, Communication and Automation Technologies*, 1–7. doi:10.1109/ICAT.2011.6102102

Fitzgibbon, A. W. (2003). Robust registration of 2D and 3D point sets. *Image and Vision Computing*, *21*(13-14), 1145–1153. doi:10.1016/j.imavis.2003.09.004

Fofi, D., Liwa, T., & Voisin, Y. (n.d.). A Comparative survery on invisible structured light.

Gall, J., Stoll, C., de Aguiar, E., Theobalt, C., Rosenhahn, B., & Seidel, H.-P. (2009). Motion capture using joint skeleton tracking and surface estimation. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 1746–1753. doi:10.1109/CVPR.2009.5206755

Grimm, C. M. (2002). Fitting Manifold Surfaces To 3D Point Clouds, 1–13.

Gross, M., & Pfister, H. (2007). *Point-based Graphics*. *Computers & Graphics* (Vol. 28, pp. 799–800). doi:10.1016/j.cag.2004.08.008

Hardware, L. (n.d.). Lidar Hardware, 3–8.

Hill, F., & Kelley, S. (2007). *Computer Graphics Using OpenGL, 3/E*. Pearson.

Ivan Tashev. (2013). Kinect Development Kit: A Toolkit for Gesture- and Speech-Based Human-Machine Interaction. *Signal Processing Magazine*.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., … Fitzgibbon, A. (n.d.). KinectFusion : Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera *.

Kavan, L. (2002). Spherical Blend Skinning : A Real-time Deformation of Articulated Models.

Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson Surface Reconstruction.

Knoll, A., Hijazi, Y., Westerteiger, R., Schott, M., Hansen, C., Member, S., & Hagen, H. (2009). Volume Ray Casting with Peak Finding and Differential Sampling, *15*(6), 1571–1578.

Kr, J. (n.d.). Acceleration Techniques for GPU-based Volume Rendering.

Kumar, A., Shih, A., Ito, Y., Ross, D., & Soni, B. (n.d.). A Hole-filling Algorithm Using Non-uniform Rational B-splines.

Lacroute, P., & Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation. *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '94*, 451–458. doi:10.1145/192161.192283

Lamar, E., & Joy, K. I. (n.d.). Multiresolution Techniques for Interactive Texture-Based Volume Visualization, *Vi*.

Levin, D. (1998). The approximation power of moving least-squares. *Mathematics of Computation*, *67*(224), 1517–1532. doi:10.1090/S0025-5718-98-00974-0

Lewis, J. P., Cordner, M., & Fong, N. (2000). Pose space deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00* (pp. 165–172). New York, New York, USA: ACM Press. doi:10.1145/344779.344862

Luna, F. D. (2008). *Introduction to 3D game programming with DirectX 10*. Jones & Bartlett Publishers.

Manson, J., Petrova, G., & Schaefer, S. (2008). Streaming Surface Reconstruction Using Wavelets. *Computer Graphics Forum*, *27*(5), 1411–1420. doi:10.1111/j.1467-8659.2008.01281.x

Marsalek, L., Hauber, A., & Slusallek, P. (2008). High-speed volume ray casting with CUDA. *2008 IEEE Symposium on Interactive Ray Tracing*, 185–185. doi:10.1109/RT.2008.4634648

Mattoccia, S., Arces, D., Viti, M., & Ries, F. (n.d.). Near real-time Fast Bilateral Stereo on the GPU.

Moeslund, T. B., & Granum, E. (2001). A Survey of Computer Vision-Based Human Motion Capture. *Computer Vision and Image Understanding*, *81*(3), 231–268. doi:10.1006/cviu.2000.0897

Nayar, S. K., & Gupta, M. (2012). Diffuse structured light. *2012 IEEE International Conference on Computational Photography (ICCP)*, 1–11. doi:10.1109/ICCPhot.2012.6215216

Nealen, A. (2003). An As-Short-As-Possible Introduction to the Least Squares , Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation, (1), 0–2.

Newcombe, R. A., Fox, D., & Seitz, S. M. (n.d.). DynamicFusion : Reconstruction and Tracking of Non-rigid Scenes in Real-Time.

Oceanic, N. (2012). Lidar 101 : An Introduction to Lidar Technology , Data , and Applications, (November).

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., & Seidel, H.-P. (2003). Multi-level partition of unity implicits. *ACM Transactions on Graphics*, *22*(3), 463. doi:10.1145/882262.882293

Ozden, K. E. (n.d.). Multibody Structure-from-Motion in Practice, (vi), 1–8.

Pfeifle, S., & Group, S. P. (n.d.). WHAT IS 3D DATA CAPTURE ?

Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., & Seiler, L. (1999). The VolumePro real-time ray-casting system. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '99*, 251–260. doi:10.1145/311535.311563

Prokos, A., Karras, G., & Petsa, E. (2010). AUTOMATIC 3D SURFACE RECONSTRUCTION BY COMBINING STEREOVISION WITH THE SLIT-SCANNER APPROACH, *XXXVIII*, 1–5.

Purcell, T. J., Buck, I., Mark, W. R., & Hanrahan, P. (2002). Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, *21*(3), 703–712. doi:10.1145/566654.566640

Rick, P. (2012). *Computer Animation*. *Computer Animation*. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84882055486&partnerID=40&md5=ca0fe20ae1be1067b68d5870d924b995

Rusinkiewicz, S., & Levoy, M. (n.d.). Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 145–152. doi:10.1109/IM.2001.924423

Salamanca, S., Merch, P., Ad, A., Cerrada, C., & Inform, E. T. S. D. I. (2008). Filling Holes in 3D Meshes using Image Restoration Algorithms, 1–8.

Schall, O., Belyaev, A., & Seidel, H. (2006). Adaptive Fourier-Based Surface Reconstruction, 34–44.

Schwarz, H., Marpe, D., & Wiegand, T. (2007). Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, *17*(9), 1103–1120. doi:10.1109/TCSVT.2007.905532

Sciences, C. (n.d.). Hole Filling in Images Siddharth Jain Video and Image Processing Laboratory.

Sweeney, J., & Mueller, K. (n.d.). Shear-Warp Deluxe : The Shear-Warp Algorithm Revisited.

Vallet, J., & Skaloud, J. (2004). DEVELOPMENT AND EXPERIENCES WITH A FULLY-DIGITAL HANDHELD MAPPING SYSTEM OPERATED FROM A HELICOPTER, 3–8.

Wang, X. C., & Phillips, C. (2002). Multi-weight enveloping. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '02* (p. 129). New York, New York, USA: ACM Press. doi:10.1145/545261.545283

Wikipedia. (2014). Polygon Mesh. Retrieved November 11, 2014, from http://en.wikipedia.org/wiki/Polygon_mesh

Wilczkowiak, M. M., Brostow, G. J., Tordoff, B., & Cipolla, R. (2005). Hole Filling Through Photomontage. *Procedings of the British Machine Vision Conference 2005*, 52.1–52.10. doi:10.5244/C.19.52

Xie, H., McDonnell, K. T., & Qin, H. (2004). Surface reconstruction of noisy and defective data sets. *IEEE Visualization 2004*, 259–266. doi:10.1109/VISUAL.2004.101

Yeung, K.-Y., Kwok, T.-H., & Wang, C. C. L. (2013). Improved Skeleton Tracking by Duplex Kinects: A Practical Approach for Real-Time Applications. *Journal of Computing and Information Science in Engineering*, *13*(4), 041007. doi:10.1115/1.4025404

Zhang, Z. (2012). Microsoft Kinect Sensor and Its Effect. *IEEE Multimedia*, *19*(2), 4–10. doi:10.1109/MMUL.2012.24

Zheng, Q., Sharf, A., Tagliasacchi, A., Chen, B., Zhang, H., & Sheffer, A. (2009). Consensus Skeleton for Non-rigid Space-time Registration, *0*(0).

Zwicker, M., Pfister, H., van Baar, J., & Gross, M. (2001). Surface splatting. *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '01*, 371–378. doi:10.1145/383259.383300

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Dynamic character reconstruction using Kinect

Richting: **master in de informatica-multimedia**
Jaar: **2015**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Schreurs, Gunter

Datum: **8/09/2015**