Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling met

Titel: Implicit Surfaces
Richting: master in de informatica - multimedia Jaar: 2009

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.

Ik ga akkoord,

RYKX, Wim

Datum: 14.12.2009

Implicit Surfaces

Wim Rykx

promotor :
Prof. dr. Philippe BEKAERT

Eindverhandeling voorgedragen tot het bekomen van de
graad master in de informatica multimedia

Universiteit Maastricht

universiteit hasselt

Implicit surfaces

Wim Rykx

Promotor: Philippe Bekaert
Begeleiders: Bert De Decker, Erik Hubo

Eindverhandeling voorgedragen tot het behalen van de van
graad Master in de Informatica afstudeervariant Multimedia

transnationale Universiteit Limburg
Academiejaar 2007-2008

Samenvatting

Deze thesis handelt, zoals de titel reeds aangeeft, over implicit surfaces. Een implicit surface is een manier om een oppervlak voor te stellen. Het is een erg flexibele en schaleerbare methode en is onder andere daarom soms te verkiezen boven traditionele methoden om een object te beschrijven. De tekst is opgesplitst in drie delen: Oppervlak voorstellingen, Visualisatie en Implementatie. In het hoofdstuk oppervlak voorstellingen wordt besproken hoe men oppervlakken nog kan voorstellen. Hier wordt ook besproken wat een implicit surface is en welke soorten er bestaan. Vervolgens wordt elk van deze soorten in detail besproken. Er wordt tot slot voor elke methode aangegeven wat de voor- en nadelen hiervan zijn. In het hoofdstuk visualisatie worden drie verschillende manieren gegeven waarop een implicit surface gevisualiseerd kan worden. Deze zijn een polygonisatie methode en een raytracing methode. In het hoofdstuk implementatie tenslotte worden de resultaten van een aantal methoden om een implicit surface te construeren besproken.

Voorwoord

In dit "Woord vooraf" wil ik graag iedereen bedanken die mij geholpen heeft bij het tot stand brengen van deze master thesis. In de eerste plaats wil ik mijn begeleider, Bert De Decker, bedanken voor het overnemen van de begeleiding van Erik Hubo. Verder wil ik hem bedanken voor de prima begeleiding en vlotte samenwerking. Ook wil ik professor Bekaert bedanken voor de kennis die hij mij heeft bijgebracht. Tenslotte wil ik Erik Hubo bedanken voor de begeleiding bij de start van mijn thesis.

Tevens wil ik alle professoren en assisten bedanken die mij hebben begeleid doorheen de voorbije vier jaren.

Als laatste dank ik mijn ouders en familie voor hun steun tijdens mijn studie en tijdens deze scriptie in het bijzonder.

Wim Rykx

Inhoudsopgave

Hoofdstuk 1

Inleiding

Implicit surfaces (impliciete oppervlakken) vormt een onderdeel van de computergraphics. Computergraphics zijn op hun beurt een zijtak in de computerwetenschap. Deze onderzoekt de manipulatie van beeldmateriaal met behulp van digitale computertechnieken. We kunnen in de computergraphics vier grote gebieden onderscheiden [55], [41]. Allereerst is er computeranimatie. Hier bestudeert men de voorstelling en manipulatie van beweging. Verder is er de beeldverwerking. In dit gebied onderzoekt men hoe beelden samengesteld en bewerkt kunnen worden. Het voorlaatste onderdeel heet rendering. Hier worden algoritmes bestudeerd die lichttransport reproduceren. Tot slot is er de geometrie, waar ook het onderwerp impliciete oppervlakken zich bevindt. In de geometrie bestudeert men hoe oppervlakken voorgesteld en verwerkt kunnen worden.
Om deze oppervlakken en andere geometrische objecten voor te stellen worden in de computergraphics primitieven gebruikt. De primitieven waaruit een model of geometrische figuur bestaat zijn onder andere lijnen, vlakken, bollen en boxen [56]. Deze primitieven zijn dus de bouwstenen om het model te construeren. Deze bouwstenen worden ook gebruikt om een afbeelding uit een model te construeren, wat men renderen (zie verklarende woordenlijst B, [41]) noemt. Onder object of model verstaan we een verzameling punten of veelhoeken die een voorwerp definiëren. Als een verzameling veelhoeken het voorwerp definieert noemt men deze verzameling een polygonenmesh.

Een andere en meer flexibele manier om modellen voor te stellen is het gebruik van functies. Dit is ook de manier waarop implicit surfaces beschreven worden, deze bestaan namelijk uit één of meer functies. Een impliciet oppervlak is een contour (omschrijving van de grens van een oppervlak, in dit geval de plaats waar de functie 0 is), ook isosurface genoemd, door een scalair veld (zie verklarende woordenlijst B). Deze functie ziet er in 3D als volgt uit: $F(x, y, z) = 0$, waarbij een punt dat binnen het oppervlak ligt een negatieve functie waarde heeft en een punt dat buiten het oppervlak ligt een positieve waarde [6]. Op deze manier kan men eenvoudig bepalen of een punt binnen of buiten een object ligt. Deze eigenschap is handig om de impliciete functie te visualiseren. Hierdoor kan makkelijk het nulpunt van de functie en dus de plaats waar het oppervlak zich bevindt gevonden worden. Het construeren van een impliciet oppervlak is dus een techniek om een oppervlak van een

object of model te modelleren.
De toepassingsgebieden van implicit surfaces zijn erg uiteenlopend. Zo kan men ze terugvinden als hoogte lijnen op een topografische kaart en als lage- en hogedrukgebieden op weerkaarten. Verder zijn er in de wiskunde ook eenvoudige functies zoals een bol of torus te vinden die een voorwerp impliciet beschrijven (zie hoofdstuk 4.1).
Een ander interessante toepassing is het gebruik van implicit surfaces bij het maken van protheses. In dit geval wordt uit de MRI-scan een puntenwolk gesampeld. Deze puntenwolk is niet overal even dicht en er kunnen plaatsen bij zijn waar zich geen punten bevinden. Voor de constructie worden impliciete functies gebruikt die de eigenschap hebben dat ze gaten in het oppervlak kunnen herstellen. Dit komt omdat de gebruikte interpolatiefunctie (zie verklarende woordenlijst B) een invloed heeft op alle punten in het object. Door deze globale invloed en het feit dat ontbrekende delen hersteld kunnen worden is deze soort functies ideaal om protheses te modelleren.
Verder kan men met impliciete functies ook makkelijk overeenkomsten tussen verschillende objecten bepalen. Dit wordt in de literatuur shape matching genoemd. Een variant van matching, partial matching, heeft als doel bepaalde subdelen of gebieden te matchen. Dit gebeurt door lokale oppervlak descriptors te definiëren.
Een impliciet oppervlak kan ook voor verschillende detail niveaus geconstrueerd worden. Hierbij worden deze niveaus in een hiëarchy van weinig naar veel detail geplaatst [26]. Dit zorgt ervoor dat de berekening van het oppervlak sneller gaat. Verder kan men hierdoor makkelijk het niveau van detail te regelen. Hiernaar wordt in de literatuur verwezen als Level Of Detail (LOD).
Een andere interessante eigenschap van impliciete oppervlakken is dat deze gebruikt kunnen worden voor het animeren van objecten [34]. Hierbij wordt na elke positie wijziging van een of meer punten uit de puntenwolk van het object een nieuw impliciet oppervlak voor het object geconstrueerd.

In computergraphicstoepassingen vertrekt men meestal vanuit een puntenwolk of polygonenmesh, die een voorwerp beschrijft, om een impliciet oppervlak te construeren. Deze voorwerpen kunnen bijvoorbeeld een beeldhouwwerk van een persoon of dier zijn. Om een duidelijker beeld te krijgen van het resultaat in zo een geval kan je je een impliciet oppervlak voorstellen als een laag die over het object gelegd wordt. Als de surface een ruwe benadering is van het object lijkt het alsof het object in een ballon zit. Als het oppervlak meer exact geconstrueerd wordt kan je je dit het best als volgt voorstellen. Een vaas die gevallen is en waarvan de grootste stukken terug aan elkaar geplakt zijn lijkt opnieuw op een vaas. Als je wil dat deze vaas opnieuw waterdicht is kan je deze verpakken in dunne plastic folie. Wanneer deze folie goed is aangebracht zal het buitenste oppervlak exact hetzelfde zijn als bij de originele vaas. Dit plastic is precies wat een impliciete functie doet.

Dankzij het gebruik van functies kan men bij de visualisatie makkelijk de gewenste precisie bepalen [6]. Door deze functie voorstelling stelt een implicit surface een object eigenlijk met oneindige precisie voor. Dit is tevens een groot voordeel dat impliciete oppervlakken hebben ten opzichte van andere rendering primitieven. De methoden waartussen gekozen

kan worden om de impliciete oppervlakken te visualiseren zijn raytracing en polygonisatie. De laatste methode gaat meestal sneller, maar hierdoor krijgt men minder detail dan bij het raytracen.

Hoofdstuk 2

Oppervlak voorstellingen

2.1 Inleiding

Onder een oppervlak verstaan we de buitenste grens van een voorwerp die deze grens voorstelt. Er bestaan verschillende methoden om een oppervlak te benaderen: expliciet, parametrisch (zie figuur(2.2)) en impliciet. Al deze methode zijn een erg flexibele manier om een oppervlak voor te stellen. In de meeste gevallen is deze manier ook veel compacter dan traditionele methoden en dus een vorm van compressie. De traditionele methoden stellen een oppervlak bijvoorbeeld voor als een mesh, een puntenwolk en voxels. Het grote voordeel van impliciete oppervlakken is echter dat men de sampling dichtheid en hiermee dus ook de precisie makkelijk kan regelen tijdens de visualisatie van de functie.

2.1.1 Expliciet

Een expliciete functie is een functie waarvan de afhankelijke variabele expliciet geschreven kan worden in termen van de onafhankelijke variabele. Enkele voorbeelden van expliciete functies: $f(x) = x^2 - 3$ en $f(x) = \sqrt{x + 7}$ (zie figuur 2.1).

Een onafhankelijke variabele is dus een variabele waaraan om het even welke waarde toegekend kan worden zonder dat daardoor een andere variabele de waarde van deze onafhankelijke variabele beïnvloed. In de voorbeelden is x dus de onafhankelijke variabele. De variabele y is afhankelijk van de waarde die voor x wordt gekozen en is dus de afhankelijke variabele.

2.1.2 Parametrisch

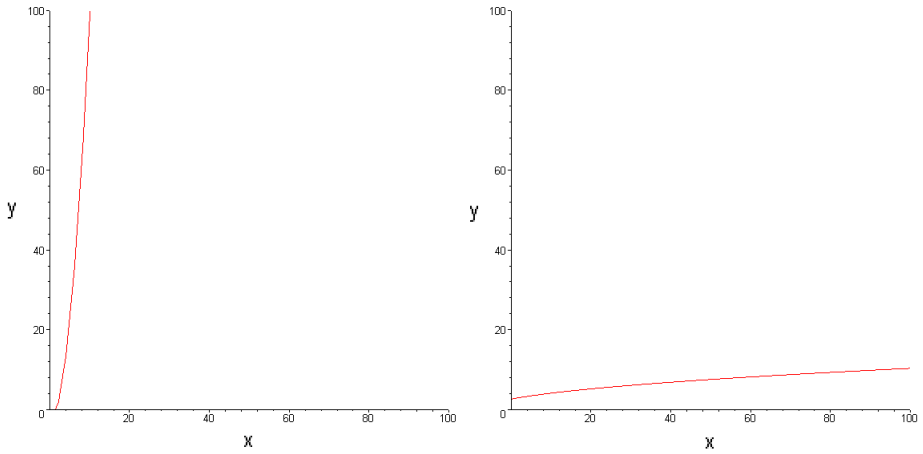
Parametrische vergelijkingen zijn een verzameling vergelijkingen die een figuur of object beschrijven. Voor elke dimensie van de ruimte waarin het object zich bevindt wordt een expliciete functie geconstrueerd. De parameter, in dit voorbeeld t (zie figuur 2.2), die aan deze functie meegegeven wordt noemt men het bereik. Voor een één-dimensionaal object (een curve) in een twee-dimensionale ruimte wil dit zeggen dat er een functie is voor de x-coördinaat en een voor de y-coördinaat, bv. $x = f_x(t)$ en $y = f_y(t)$, met

$f_x = (t * 2 - 1) + sin(t * 4 * \pi) * 4$ en $f_y = t * (-4) + 2$. Het resultaat van deze functie is te zien in figuur 2.2.

Voor een twee-dimensionaal object (een vlak) in drie-dimensionale ruimte geeft dit volgende functies: $x = f_x(s,t)$, $y = f_y(s,t)$ en $z = f_z(s,t)$.
Parametrische oppervlakken geven dus een afbeelding van het object naar de ruimte waarin het object zich bevindt. Voor een twee-dimensionaal object in een drie-dimensionale ruimte kan dus een drie-dimensionaal punt op het oppervlak omschreven worden door een paar (s,t).

2.1.3 Impliciet

Een impliciete functie is een functie waarbij één variabele niet expliciet in termen van een andere variabele is geschreven. $3x + y = 0$ en $x^2 + y^2 = 0$ zijn hier twee voorbeelden van. Elke variabele is in dit geval impliciet een functie van de andere. De eerste functie kan ook als een expliciete functie geschreven worden, namelijk $y = -3x$ of $x = y/3$. De tweede functie kan expliciet geschreven worden als twee vergelijkingen: $y = \sqrt{1 - x^2}$ en $y = -\sqrt{1 - x^2}$ of als $x = \sqrt{1 - y^2}$ en $x = -\sqrt{1 - y^2}$. Soms is het niet mogelijk is om de functie voor één van de variabelen in termen van de andere variabele te schrijven, zoals bij $y + e^y = x^3$. In dit geval kan men, gegeven een waarde voor één van de variabelen, een waarde voor de andere variabele vinden zodat deze aan de vergelijking voldoet. Als x bijvoorbeeld 2 is, is y ongeveer gelijk aan 3.35497961935092. Deze oplossing kan men vinden door bijvoorbeeld de regula falsi (zie verklarende woordenlijst B) methode toe te

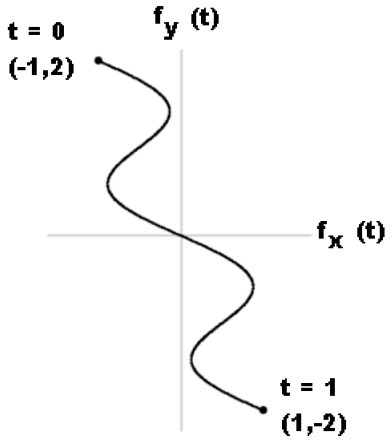


Figuur 2.1: De functies in bovenstaande afbeeldingen zijn expliciete functies. Van links naar rechts zijn dit respectievelijk $f(x) = x^2 - 3$ en $f(x) = \sqrt{x + 7}$.

passen. $y + e^y = x^3$ kan niet naar y opgelost worden. We kunnen $y + e^y = x^3$ wel naar x oplossen, dit geeft dan $x = \sqrt[3]{y + e^y}$.

Een impliciete functie in 3D wordt genoteerd als $F(x, y, z) = 0$, in 2D als $F(x, y) = 0$. Hiermee wordt bedoeld, zoals eerder uitgelegd dat deze functies expliciet in termen van x, y of z geschreven kunnen worden. Voor impliciete oppervlakken in 3D [5] geldt, $F(x, y, z) = c$, met c een punt in $\mathbb{R}$ en F een functie $\mathbb{R}^3 \rightarrow \mathbb{R}$, met c een scalaire constante. Als c nul is, definieert f impliciet een meetkundige plaats, impliciet oppervlak genoemd [5]. Als c negatief is weten we dat het punt (x, y, z) binnen het object ligt dat door F gedefinieerd wordt. Als c positief is weten we dat het punt (x, y, z) buiten het object ligt dat door F gedefinieerd wordt. De verzameling punten $p \in \mathbb{R}^3 : f(p) = 0$ is het impliciete oppervlak gedefinieerd door f. In de literatuur wordt een impliciete functie ook wel 'scalair veld', 'veldfunctie', 'mogelijke functie' genoemd. Tenslotte wordt het impliciete oppervlak soms de verzameling nullen (of nuloppervlak) van f genoemd en geschreven als $f^{(-1)}(0)$ of $Z(f)$ [5].
f wordt typisch gespecifieerd door wiskundige functies, dit kunnen bijvoorbeeld polynomen zijn die dan samen het oppervlak beschrijven.

Op een impliciet oppervlak kunnen ook normalen gedefinieerd zijn, indien niet kan men ze ook berekenen. Om deze normalen te kunnen berekenen moet de functie f continu en afleidbaar zijn [5]. Formeel wil dit zeggen dat de partiële afgeleiden [47] (zie verklarende



Figuur 2.2: De afbeelding geeft een parametrische functie weer. Meer bepaald een curve in een twee-dimensionele ruimte. Het bereik van de parametrische functies, $f_x = (t * 2 - 1) + sin(t * 4 * \pi) * 4$ en $f_y = t * (-4) + 2$, ligt tussen 0 en 1.

woordenlijst B) $\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}, \frac{\delta f}{\delta z}$ continu moeten zijn en niet allen nul, overal op het oppervlak [5]. Als normaal in een punt wordt daarom ook meestal de gradiënt genomen. Deze gradiënt wordt dan nog genormaliseerd, zodat de lengte ervan gelijk is aan 1 [46] (zie verklarende woordenlijst B).

Als de gradiënt in een punt p verschillend is van nul wordt gezegd dat p regulier(of enkelvoudig) is [5].

Als de gradiënt (of equivalent de tangens vector [48] (zie verklarende woordenlijst B) niet bestaat, is het punt singulier [5] (zie verklarende woordenlijst B) figuur(2.3) (ook wel kritiek of niet-regulier genoemd).

Figuur 2.3: Het punt van een kegel is een singulier punt

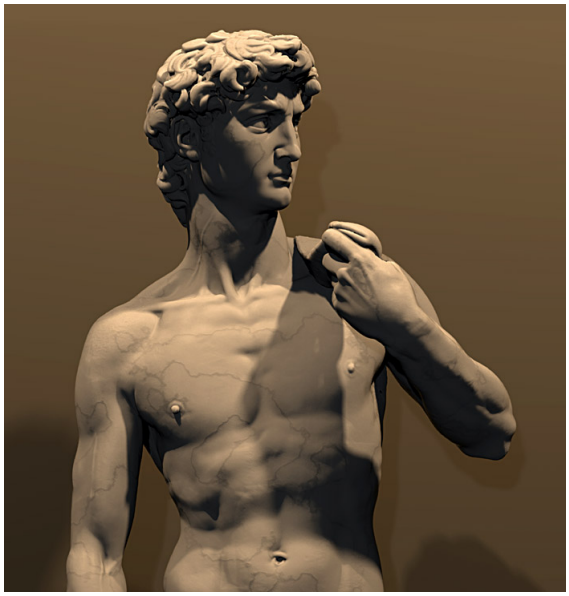
De normaal van een singulier punt wordt soms berekend door het gemiddelde van de omliggende normalen te nemen. Dit wordt gedaan omdat zoals eerder vermeldt een singulier punt geen gradiënt heeft en hiervoor dus ook geen normaal berekend kan worden. Door het gemiddelde van omliggende normalen te nemen bekomt men toch een goede schatting voor de normaal in een singulier punt.

Met parametrische functies kan men dus op een eenvoudige manier alle gebieden die door de functie beschreven worden afgaan. In figuur (2.2) geldt dit voor parameter t over het interval $[0, 1]$. Refererend naar vorig voorbeeld (zie hoofdstuk 2.1.2) kunnen we gegeven een coördinaat (s, t) op het 2D oppervlak in de 3D ruimte, de plaats in de 3D ruimte bepalen. Hoewel impliciete oppervlakken hier minder voordelig zijn [5], bieden zij meer informatie over de binnen- en buitenkant dan parametrische oppervlakken. Om het verschil tussen een parametrisch oppervlak en een impliciet oppervlak te verduidelijken wordt in [59] de vergelijking met een kustlijn gemaakt. Een parametrisch oppervlak kan men zich voorstellen als een kustlijn met elke kilometer een markering, terwijl een impliciet oppervlak een kustlijn is waarbij elke kilometer de hoogte en laagte van het land zowel onder als boven water wordt aangegeven.

2.1.3.1 Constructie

Zoals in de inleiding vermeld werd wordt in de computergraphics bij de constructie van een impliciet oppervlak meestal vertrokken vanuit een puntenwolk of een polygonenmodel. Deze puntenwolk kan onder andere bekomen worden door een voorwerp in te scannen met

een 3D scanner. De universiteit van Stanford heeft op deze manier heel wat historische beeldhouwwerken ingescand [17], [16], figuur(2.4), [33], figuur(2.5).



Figuur 2.4: De laserscan van de David van Michelangelo. Deze werd gerenderd uit een 3D model bestaande uit 8 miljoen polygonen [17] die elke 2 mm groot zijn. Het originele beeldhouwwerk is wit, de kleur werd artificieel aangebracht. Tijdens het scannen werd om de 0,25 mm een scan gemaakt waardoor de ruwe data uit zeer veel polygonen bestaat (2 miljard)

In het geval van een polygonenmesh gaat het om objecten waar geen puntenwolk voor bestaat (zie hoofdstuk 2.2).
In beide gevallen is er een minimalisatieprobleem dat opgelost moet worden. Bij dit probleem wordt gezocht naar een functie die zo goed mogelijk door een gegeven verzameling punten of polygonen fit. Zoals we verderop zullen zien bestaan er verschillende technieken om dit probleem op te lossen.

2.1.3.2 Renderen

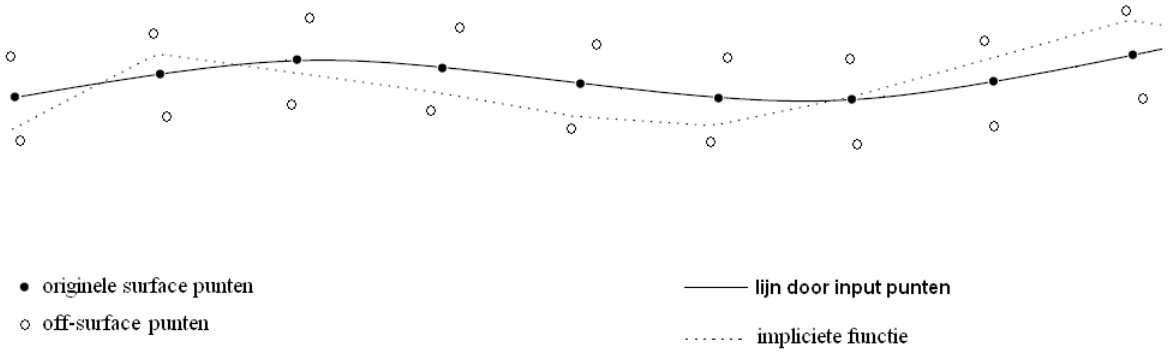
Het weergeven van een impliciete functie kan op twee manieren, ofwel via polygonisatie of via raytracing (zie hoofdstuk 3.1), (zie hoofdstuk 3.2). Bij polygonisatie wordt een impliciet oppervlak omgezet in polygonen, met behulp van het marching cubes algoritme [19], zie hoofdstuk(3.1.1). Dit algoritme loopt als het ware de impliciete functie af. Door gebruik te maken van polygonisatie wordt er echter aan kwaliteit ingeboet. Dit komt

Figuur 2.5: Een 3D weergave van het minerva beeld van Arezzo uit het archeologisch museum van Firenze. Achtereenvolgens is er een zijaanzicht van het hoofd en het volledige beeld te zien.

omdat de functievoorstelling een object oneindig scherp beschrijft, integenstelling tot een polygonenmesh. Een polygonenmesh is namelijk vast, deze kan niet op eenvoudige manier verfijnd worden zodat de mesh meer detail weergeeft. Bij een functievoorstelling kan men in de visualisatie stap het niveau van detail kiezen. Als het impliciete oppervlak geraytraced wordt is er al heel wat minder detailverlies, maar deze methode is complexer en duurder [12], [36]. We zouden er ook voor kunnen kiezen om zeer veel polygonen te construeren, zelfs meerdere per pixel. In dit geval is raytracing natuurlijk voordeliger.

2.1.3.3 Voorstelling

Er zijn heel wat verschillende methoden om een impliciet oppervlak voor te stellen: SLIM, polygonenmesh, MPU, RBF's en de hiërarchische methode. Sommige van deze methoden maken, naast de originele inputpunten, gebruik van extra punten, off-surface punten. Dit zijn punten die geconstrueerd worden aan de hand van de gegeven oppervlakte punten en naast het oppervlak liggen. Deze punten leggen constraints op aan de impliciete functie zodat deze tussen de off-surface punten gefit wordt (zie figuur 2.6).



Figuur 2.6: Voorbeeld van off-surface punten waartussen de impliciete functie gefit wordt. De lijn door de inputpunten stelt het impliciete oppervlak voor als dit exact door de input punten zou gaan.

Een aantal methoden gebruiken globale functies, de andere lokale. Als bij globale functies de waarde van een parameter verandert wordt, heeft dit invloed op de volledige voorstelling van het object. Bij lokale functies daarentegen blijft de invloed beperkt tot het gebied dat de functie voorstelt. Aan elk van deze strategiën zijn voor- en nadelen

verbonden. De lokale methode zorgt bijvoorbeeld voor een snelheidswinst in de berekening, maar hiermee is het moeilijker om ontbrekende data te herstellen [26]. Aan de andere kant is een globale methode wel goed in het herstellen van data, maar duurt de berekening hiervan langer [6].

2.2 Interpoleren van een Polygon mesh

Bij het construeren van impliciete oppervlakken voor een polygonenmesh wordt vertrokken van een verzameling polygonen. In de literatuur spreekt men van polygon soup. Deze term verwijst naar de ongeordende toestand van de polygonen. Dit wil zeggen dat de verzameling kan bestaan uit polygonen van verschillende grootte, op sommige plaatsen kunnen er polygonen ontbreken of intersecties zijn. Hierdoor zijn ze meestal enkel geschikt om gerenderd te worden [35]. Het is mogelijk om deze verzameling van „willekeurige polygonen" om te zetten in een meer bruikbare vorm, impliciete oppervlakken. Dit is een beschrijving van een oppervlak waarbij tijdens het renderen het detail gekozen kan worden. Bij real-time toepassingen kan er bijvoorbeeld voor gekozen worden om grote driehoeken te gebruiken. Verder kunnen deze impliciete functies bij CAD toepassingen gebruikt worden om de afgeleiden in bepaalde delen van een model te bepalen.
Het principe om van een polygonen mesh een impliciete functie te construeren begint lokaal. Allereerst wordt er voor elke polygoon een constraint bepaald in de vorm van een surface en off-surface punt, dit noemt men een puntconstraint. Het off-surface punt wordt op een bepaalde afstand van de polygoon geconstrueerd, in de richting van de normaal. Vervolgens kan de moving least-squares methode (zie **Waarde beperkingen voor punten** in hoofdstuk 2.2.1) gebruikt worden om een impliciete functie te construeren die aan deze constraints voldoet. Een andere manier om een impliciet oppervlak te construeren is door een benaderende functie te definiëren voor een polygoon of een groep van polygonen. In dit geval zijn de constraints geen eenvoudige puntconstraints, maar geïntegreerde puntconstraints over een polygoon.

Het stelsel van vergelijkingen dat we dan bekomen kan ook opgelost worden met behulp van de moving least-squares methode (zie **Waarde beperkingen geïntegreerd over polygonen** in hoofdstuk 2.2.1). Tot slot moet ervoor gezorgd worden dat het construeerde oppervlak alle input polygonen omsluit omdat er anders artefacten in het resultaat kunnen voorkomen.

De methode die Chen et al. [35] gebruiken om een impliciete functie voor een polygonen mesh te construeren bestaat uit vijf grote delen [35]:

- een interpolatieschema voor de data. Dit schema laat toe om constraints op te leggen aan de implicit surface die voor een groep van polygonen gelden
- een methode voor het opleggen van de constraints aan groepen polygonen zodat er geen grote afwijkingen optreden tussen aanliggende polygonen. Dit kan namelijk bulten en kuilen in het oppervlak tot gevolg hebben
- een aanpassingsprocedure die de impliciete functie aanpast zodat het impliciete oppervlak goed rond de polygonen past en er tevens voor zorgt dat de inputpolygonen volledig omsloten worden door het impliciete oppervlak
- een snel hiërarchisch evaluatieschema zodat de methode praktisch is voor grote datasets

- een optionele stap voordat de verwerking begint. In deze stap kunnen ongewenste vormen, zoals intersecties, bulten en kuilen, uit de polygonen mesh verwijderd worden en kan er consistentie opgelegd worden op de input normalen

Het belangrijkste element van deze methode is het interpolatieschema voor onregelmatig (niet uniform) gesampelde invoerdata, ook bekend als implicit moving least-squares IMLS (zie hoofdstuk 2.2.1). Een andere naam waaronder deze methode bekend is, is het kleinste kwadratenprobleem. Deze methode werkt echter alleen op constraints opgelegd op individuele punten en de invoer data zijn in dit geval polygonen. We willen namelijk constraints opleggen aan elke polygoon zodat hierover een implicit functie geconstrueerd kan worden. Ditzelfde effect kan verkregen worden door een puntenwolk op het oppervlak van de polygoon te definiëren en hier een functie door te fitten. Een nadeel van deze methode is dat, afgezien van het feit dat er veel punten nodig zijn, er ongewenste bulten en kuilen in het resultaat voorkomen [35]. Een manier om dit probleem te verhelpen is voor elke polygoon drie punten te construeren: een punt op de polygoon, een punt erboven en een punt eronder [35]. Het punt op de polygoon krijgt de waarde nul, het punt boven de polygoon krijgt een positieve waarde en het punt onder de polygoon krijgt een negatieve waarde.

2.2.1 Least-squares

2.2.1.1 Waarde beperkingen voor punten

Stel we hebben N punten op lokaties p_i, $i \in [1...N]$ en we willen een functie, $f(x) = b^T(x)\, c$, door deze punten fitten die de functiewaarden ϕ_i van $f(p_i)$ in deze punten benadert. Met $b(x)$ de vector van de basisfuncties die gebruikt worden om de functie te fitten en c de onbekende vectorcoëfficiënten. Bij een standaard kleinste kwadraten fit zouden we volgende matrix oplossen:

$$\begin{bmatrix} b^T(p_1) \\ \vdots \\ b^T(p_N) \end{bmatrix} \begin{bmatrix} c \\ \vdots \\ c \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}, \tag{2.1}$$

Aangezien er evenveel vergelijkingen als onbekenden zijn kan dit $M \times M$ lineair systeem op een efficiënte manier opgelost worden. M is het aantal basisfuncties (i.e. de lengtes van b en c). Als we bijvoorbeeld een vlak willen fitten kunnen we $b(x) = [1, x, y, z]$ kiezen, als we een constante willen fitten kiezen we $b(x) = [1]$. De resulterende functie is: $f(x) = c$. De complexiteit van de basisfuncties is dus afhankelijk van de complexiteit van het object dat we willen voor stellen.
De moving least-squares formulering laat toe dat de benaderende functie zich kan aanpassen afhankelijk van de positie waar het evaluatiepunt x zich bevindt, zodat c varieert met x. Dit wil dus zeggen dat de coëfficiënten c veranderen afhankelijk van de plaats waar de functie in het model gefit wordt. Dit gebeurt door elke rij van vergelijking (zie functie 2.1)

af te wegen tegen $w(||x - p_i||)$), de afstand van een te evalueren punt x tot een punt p_i, met $w(r)$ een bepaalde gewogen functie. Dit geeft dan:

$$\begin{bmatrix} w(x,p_1) & & \\ & \ddots & \\ & & w(x,p_N) \end{bmatrix} \begin{bmatrix} b^T(p_1) \\ \vdots \\ b^T(p_N) \end{bmatrix} \begin{bmatrix} c(x) \\ \vdots \\ c(x) \end{bmatrix} = \begin{bmatrix} w(x,p_1) & & \\ & \ddots & \\ & & w(x,p_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \tag{2.2}$$

met $w(x,p_i) = w(||x - p_i||)$. Nu wordt de functie f dus door volgende vergelijking gegeven:

$$f(x) = \sum_{i=1}^{N} w(x,p_i) b^T(x) c,$$

met N het aantal interpolatie punten.
We zoeken een gewogen functie, voor w, die bestaat uit de deling van het getal 1 door een variabele, zodat de functie ∞ is in de limiet naar nul. Hierdoor zullen punten die ver weg liggen minder meetellen (doorwegen) in het resultaat. Dit veroorzaakt interpolatie. Een goede keuze voor zo'n functie is:

$$w(r) = \frac{1}{(r^2 + \epsilon^2)}. \tag{2.3}$$

met ϵ de blending afstand tussen de interpolatiepunten die later meer in detail besproken wordt (zie hoofdstuk (2.2.3)).

2.2.1.2 Waarde beperkingen geïntegreerd over polygonen

In plaats van de polygonen te interpoleren, zoals in vorige methode, kunnen we de gewenste functie benaderen met punt constraints verspreid over het oppervlak van elke polygoon. Volgende functie is een expliciete sommatie over een verzameling van punt constraints:

$$\left(\sum_{i=1}^{N} w^2(x,p_i) b(p_i) b^T(p_i) \right) \; c(x) = \sum_{i=1}^{N} w^2(x,p_i) b^T(p_i) \phi_i \tag{2.4}$$

die bekomen kan worden door de matrices van vergelijking 2.2 in termen van x te schrijven:

$$W(x)Bc(x) = W(x)\phi$$

en vervolgens te normaliseren, door de deling van functie (2.3):

$$B^T(W(x))^2 Bc(x) = B^T(W(x))^2 \phi$$

Voor een inputverzameling van K polygonen, zij Ω_k, $k \in [1...K]$ de k-de invoer polygoon. Als de term tussen haakjes van vergelijking (2.4) en de term aan de rechterkant door integralen over de polygonen vervangen worden krijgen we:

$$\left(\sum_{k=1}^{K} A_k\right)\, c(x) = \sum_{k=1}^{K} a_k \tag{2.5}$$

waar A_k en a_k gedefinieerd zijn door:

$$\widetilde{A}_k = \int_{\Omega_k} w^2(x,p) b(p) b^T(p)\ dp$$

$$\widetilde{a}_k = \int_{\Omega_k} w^2(x,p) b^T(p) \phi_k\ dp$$

Op deze manier kunnen er constraints aan de polygonen opgelegd worden zonder dat er ongewenste effecten zoals kuilen of bulten in het resultaat voorkomen. Verder zorgt deze methode ervoor dat een gegeven oppervlak van polygonen geïnterpoleerd wordt.

2.2.2 Functie constraints

Het opleggen van de constraints door extra punten toe te voegen kan ruis veroorzaken naarmate deze verder van het oppervlak verwijderd zijn. Als de off-surface punten te dicht bij het oppervlak liggen bekomen we opnieuw de originele mesh die allerlei onvolmaaktheden bevat. Om dit probleem te verminderen hebben Chen en O'Brien een nieuwe methode [35] geïntroduceerd. Deze methode legt constraints op aan de polygonen door met behulp van de normaal van de polygoon een functie $S_k(x)$ (zie functie 2.6) te definiëren. De interpolatieconstraints worden hier dus gedefinieerd door een functie in plaats van door constanten zoals bij het gebruikt van off-surface punten. Bij off-surface punten wordt namelijk, zoals eerder vermeld, een positieve en negatieve waarde gebruikt om constraints op te leggen. De functie bevindt zich dicht bij de polygoon die de constraint bepaalt, net zoals de off-surface punten zich dicht bij de polygoon bevonden. Voor elke polygoon Ω_k wordt er een functie $S_k(x)$ geconstrueerd die deze polynoom benadert.

$$S_k(x) = \phi_k + (x - q_k)^T \hat{n}_k \tag{2.6}$$

$$= \psi_{0k} + \psi_{xk} x + \psi_{yk} y + \psi_{zk} z \tag{2.7}$$

met q_k een willekeurig punt op de polygoon Ω_k, $\hat{n}_k$ de normaal en Ω_k en $\psi_{0k}, \psi_{xk}x, \psi_{yk}y$ en $\psi_{zk}z$ de resulterende polynomiaal coëfficiënten [35]. De interpolatie tussen de constraintfuncties kan gebeuren door te interpoleren tussen de ψ coëfficiënten zoals normaal gebeurde tussen de constante waarden ϕ_k (zie hoofdstuk 2.2.1). De matrices van het kleinste kwadraten probleem zien er nu als volgt uit:

$$\begin{bmatrix} w(x,p_1) & & \\ \vdots & & \\ w(x,p_i) & & \end{bmatrix} \begin{bmatrix} b^T(p_1) \\ \vdots \\ b^T(p_N) \end{bmatrix} \begin{bmatrix} c(x) \\ \vdots \\ c(x) \end{bmatrix} = \begin{bmatrix} w(x,p_1) & & \\ & \ddots & \\ & & w(x,p_N) \end{bmatrix} \begin{bmatrix} S_1(x) \\ \vdots \\ S_N(x) \end{bmatrix} \tag{2.8}$$

met $p_1, ..., p_N$ de punten geconstrueerd voor elke polygoon en $w(x, p_i)$ een afstandsfunctie $||x - p_i||$.

Deze methode heeft echter ook een nadeel, ze vertoont namelijk een beetje ruisvorming in het resultaat. Dit komt omdat aan de hand van de waarde van ϵ niet kan worden afgeleid of alle originele punten binnen het impliciete oppervlak vallen. Hierdoor tellen niet alle punten voldoende mee in het resultaat. In volgende paragraaf wordt hiervoor een oplossing beschreven.

2.2.3 Constructie en evaluatie van de functie

Om een functie te vinden die zo goed mogelijk door de polygonen fit moet de parameter ϵ van de weighting- of afstandsfunctie $w(x, p_i)$ bepaald worden. Deze parameter ϵ geeft tevens ook de error of afwijkingsgraad ten opzichte van de originele figuur weer. Deze bepaald met andere woorden hoe dicht bij de polygonen de functie zal liggen. Als deze parameter op nul gezet wordt, zal de impliciete functie exact de constraint waarden interpoleren. Indien voor ϵ een grote waarde gekozen wordt krijgen we een bolvormige figuur waarvan de straal verschillende keren groter kan zijn dan de omtrek van het object. Aan de waarde van ϵ kan niet worden afgeleid of alle originele vertices binnen het impliciete oppervlak vallen.
Om te voorkomen dat het geconstrueerde impliciete oppervlak te ver afwijkt van de input data wordt de volgende strategie toegepast. Eerst wordt een moving least-squares functie geconstrueerd met een gewenste error maat ϵ_0 die handmatig gekozen wordt aan de hand van de polygonmesh. Dan wordt de gemiddelde waarde van ten opzichte van de input polygonen berekend. Vervolgens wordt ϵ gelijk gesteld aan deze gemiddelde waarde. Als deze gemiddelde waarde voor ϵ gebruikt zou worden, zou slechts de helft van de originele punten binnen het impliciete oppervlak vallen. Daarom worden de ϕ-waarden in de matrix van het kleinste kwadraten probleem op een iteratieve manier aangepast tot alle originele data binnen het impliciete oppervlak vallen [35]. Op het eerste zicht lijkt dit veel bijkomend werk te zijn, maar deze hoeveelheid is niet significant [35]. Dit komt omdat het meeste werk verricht wordt bij het construeren van het impliciete oppervlak en dit moet slechts één keer gebeuren.
Als het oppervlak van polygonen gaten of openingen bevat dan zal het impliciete oppervlak deze gaten of openiningen overspannen om een gesloten oppervlak te genereren. Er is echter geen garantie dat deze opvulling aan bepaalde criteria voldoet. Volgens de resultaten van Shen en O'Brien [35] gebeurt dit echter op een gelijkaardige manier als wanneer een mens deze gaten zou opvullen.
De kost voor het evalueren van de impliciete functie wordt voor het grootste deel bepaald door volgende som van integralen over alle polygonen:

$$\left(\sum_{k=1}^{K} A_k\right) c(x) = \sum_{k=1}^{K} a_k$$

(zie hoofdstuk 2.2.1)

Als de weightingfunctie niet afhankelijk was van x, zou de term constant zijn en zou de berekening maar een keer moeten gebeuren. Op plaatsen waar dicht bij het evaluatiepunt een lokaal minimum of maximum is, is de functiewaarde van de impliciete functie voor twee opeenvolgende punten erg verschillend. Op plaatsen waar de functie en dus ook de figuur vrij vlak is blijft de functiewaarde ongeveer constant. Deze laatste groep van termen kan vervangen worden door een benadering. Om de benadering te berekenen worden de termen gesommeerd en gedeeld door hun gemiddelde gewicht. Een gedetailleerde uitleg kan gevonden worden in hoofdstuk 2.2.1.

2.2.4 KD-boom

Om het impliciete oppervlak op een efficiënte manier te construeren wordt gebruik gemaakt van een KD-boom (zie verklarende woordenlijst B). Initiëel zitten alle polygonen in één node. Vervolgens wordt deze node opgesplitst tot er zich in elk blad van de boom één driehoek bevindt. Hierna worden de ongewogen integralen berekend:

$$\widetilde{A}_k = \int_{\Omega_k} b(p)b^T(p)\ dp$$

$$\widetilde{a}_k = \int_{\Omega_k} b(p)\phi_k\ dp$$

Voor elke driehoek in de boom worden deze integralen berekend. Het resultaat wordt samen met de boundingbox in de bladeren van de boom opgeslaan. Daarna worden in de andere knopen, de hogere niveaus van de boom, de ongewogen som van de integralen van hun kinderen opgeslaan. Als laatste wordt voor deze knopen de "center of mass" (zwaartepunt), dit is het gemiddelde punt van alle punten die zich in de boundingbox van deze knoop bevinden, berekend en opgeslaan.
Tijdens het evalueren van de impliciete functie (2.5) wordt voor elk punt waarvoor de functie geëvalueerd wordt tot elke node in de boom de afstand bepaald. Als het punt buiten een bepaalde afstand van de node, en dus buiten de onderliggende deelboom, ligt wordt de som van de integralen in die deelboom gebruikt samen met een gewicht. In dit geval is de afstand tot de deelboom groter dan α keer de diagonaal van de boundingbox. Uit de resultaten van Shen en O'Brien is gebleken dat de waarde voor α best tussen 0.01 en 0.1 gekozen wordt. Als ϵ klein is kan best voor een kleine α gekozen worden en als ϵ groot is voor een grote α-waarde. Het gewicht is de afstand tussen het zwaartepunt van de deelboom en het punt dat geëvalueerd wordt. Als het evaluatiepunt daarentegen dichterbij staat worden de kindknopen van de deelboom verder afgegaan. Dit proces gaat zo verder tot uiteindelijk de bladknopen bereikt worden. In een bladknoop aangekomen wordt de gewogen integraal voor die knoop berekend. De functiewaarde van het evaluatiepunt is dan het resultaat van alle bijgehouden sommen (tussen resultaten).
Door de impliciete functie op deze manier te evalueren kan deze methode ook gebruikt worden voor grote modellen die uit verschillende honderdduizenden driehoeken bestaan.

De kost van het evalueren van een enkel punt is namelijk slechts $O(logN)$, met N het aantal niveaus van de boom.

2.2.5 Constructie met behulp van een volume model

Yngve en Turk gebruiken een volumemodel om een implicit surface uit een polygonenmesh te construeren [59]. Dit volumemodel is een 3D grid die de hele mesh omvat. Deze omzettingsmethode naar een volumemodel wordt voxelization (voxelisatie) genoemd [59]. Het resultaat zijn voxels waardoor vervolgens een functie gefit kan worden. Deze voxels zijn met andere woorden de constraints waaraan de functie moet voldoen, dit wil zeggen dat de functie er zo goed mogelijk door moet fitten. Allereerst wordt er een grid rond het object geconstrueerd. Vervolgens worden er rays afgeschoten in de richting van de polygonenmesh. Deze rays worden op een parallelle manier afgeschoten en op een vaste afstand van elkaar [23]. De rays worden enkel door het grid dat rond het object geconstrueerd werd geschoten. Dit gebeurt vanuit drie zijden van de grid. Daarna worden alle intersectiepunten van de rays met de mesh berekend. Voor elk intersectiepunt wordt een voxel [53], [9], (zie verklarende woordenlijst B) geconstrueerd. Dit zijn dan de constraints. Om ook off-surface punten te construeren worden de rays over een regelmatige afstand gesampled. Door een pariteitsgetal bij te houden kan eenvoudig bepaald worden of een punt binnen of buiten het object ligt.
Het construeren van het impliciete oppervlak uit deze voxels gebeurt door gebruik te maken van Radial Basis Functies (RBF's) (zie hoofdstuk 2.5, 2.13). Het opstellen van de matrix gebeurt op dezelfde manier als in hoofdstuk 2.5. De radiale basisfunctie die gebruikt wordt voor de voxels is $\phi(x) = |x|^2 log(|x|)$. Daarna wordt de matrix opgelost met behulp van LU decompositie (zie hoofdstuk A.2).
Het aantal gebruikte constraints wordt bepaald aan de hand van een error metriek [59]. Deze metriek geeft aan hoe dicht de huidige berekening van het impliciete oppervlak de originele data benadert. Om te beginnen worden er x aantal constraints geselecteerd. Daarna wordt hiervoor een initiële impliciete functie geconstrueerd. Als deze te veel afwijkt van het originele model en de error dus groot is, worden er constraints toegevoegd. Aangezien het toevoegen van één constraint per keer een erg dure methode is, worden er meerdere per keer toegevoegd. Om te bepalen welke nieuwe constraints toegevoegd moeten worden, wordt de functie op alle plaatsen van de huidige constraints geëvalueerd. Als de error groot is, wordt er op deze plaats een constraint toegevoegd. Men moet echter voorzichtig zijn met toevoegen aangezien bepaalde constraints niet voor verbetering zorgen en anderen de error alleen maar vergroten. De error kan vergroten als een off-surface punt dat erg ver van het oppervlak ligt wordt toegevoegd. Het is mogelijk dat de functie op deze plaats dan een bult of kuil zal vertonen. Eerst wordt tussen de resterende constraints deze gezocht die de grootste error heeft. Als deze constraint ook een bepaalde minimumafstand heeft tot de constraints die al gebruikt worden om de impliciete functie te construeren, wordt deze toegevoegd [59]. Het toevoegen van extra constraints gaat verder tot:

- een bepaald niveau van accuraatheid bereikt wordt,

- het model niet gedetailleerder wordt gedurende x aantal iteraties
- er teveel iteraties geweest zijn.

2.2.6 Conclusie

De voxelisatiemethode van Yngve en Turk [59] is op het eerste zicht een tragere methode dan de methode van Chen en O' Brien [35]. Of dit werkelijk zo is, zou verder onderzocht moeten worden, maar dit valt buiten het bestek van deze thesis. We zullen hier kort motiveren waarom de voxelisatie methode waarschijnlijk trager is. Een mogelijke oorzaak hiervan is dat bij de voxelisatiemethode geen versnellingsstructuur zoals een KD-boom gebruikt wordt. Indien de voxelistiemethode gebruik zou maken van de versnellingsstructuren uit hoofdstuk 2.5, deze van Carr [6] of Wu [58] kan deze ook gebruikt worden voor grote modellen. Het andere grote verschil tussen beide methoden is het detail waarmee de modellen geconstrueerd worden. Bij de methode van Chen en O'Brien wordt elke polygoon in rekening gebracht, terwijl bij de voxelisatiemethode slechts een minimum aan punten gebruikt wordt. Bij Chen en O'Brien werden vooraf reeds de ruis en intersecties van polygonen weggefilterd. Dit zou bij de tweede methode ook kunnen gebeuren, maar door de creatie van de voxels komt er toch weer ruis in. Deze ruis ontstaat doordat er voor een polygoon meerdere voxels kunnen gemaakt worden en deze zijn niet allemaal nodig om de implcit surface te construeren. Hieruit concluderen we dat de methode van Chen en O'Brien waarschijnlijk de voorkeur geniet om een impliciet oppervlak te construeren uit een polygonenmesh.

2.3 SLIM

De Sparse-Low-degree IMplicits methode is een hiërarchische methode die een accurate voorstelling van een oppervlak geeft, vertrekkende van een verzameling inputpunten $P = p_1, ..., p_N$, samen met hun overeenkomstige normalen $n_1, ...n_N$. Het doel van deze methode is om het mogelijk te maken dat modellen met veel punten in real-time gevisualiseerd kunnen worden [24]. Als versnellingsstructuur wordt een octree gebruikt. Voor elke node van de octree wordt een surfel en de kromming (curvature) van het oppervlak in de buurt van deze node berekend. Een surfel is een oppervlak element dat bestaat uit 3 delen: een oorsprong c, een straal ρ en een lokale benaderingsfunctie $f(x) = 0$ voor het oppervlak. De lokale benaderingsfunctie geeft enkel een benadering van het oppervlak binnen de bol die gedefinieerd is door c en ρ. De functies die hiervoor gebruikt worden zijn polynomen van lage graad (kwadratisch en kubisch). De oppervlaktekrommingen worden geschat door de afgeleide te nemen van de lokale benaderingsfuncties. Een andere manier om de mate van kromming van een oppervlak te bepalen is door de hoek tussen normalen van naburige punten te berekenen (zie hoofstuk 2.4). Om een globale benadering te bekomen worden de lokale functies samengevoegd. De naburige oppervlaktefuncties worden hierbij geblend om een smooth resultaat te bekomen.
Eens de SLIM voorstelling van een gegeven oppervlak gemaakt is, kan men snel hogekwaliteitsrendering bereiken. Het grote verschil met de andere methodes is dat het oppervlakelement (zie verklarende woordenlijst B) bij de SLIM methode geen punt met een normaal is, maar een surfel.
Dankzij deze voorstellingsmethode kunnen grote modellen in real-time gevisualiseerd worden en is de renderingkwaliteit beter dan bij standaard splatting technieken (zie verklarende woordenlijst B). Omdat de vorm van een oppervlak bij deze methode erg snel en vrij nauwkeurig berekend kan worden, is deze techniek ook geschikt voor toepassingen waarbij geen fotorealistische kwaliteit vereist is [24]. Als men meer detail en dus nauwkeurigheid wenst, zal men een andere methode moeten kiezen. Men kan bijvoorbeeld voor Radiale Basis Functies (zie hoofdstuk 2.5) of de MPU methode (zie hoofdstuk 2.4) kiezen die verderop besproken worden.

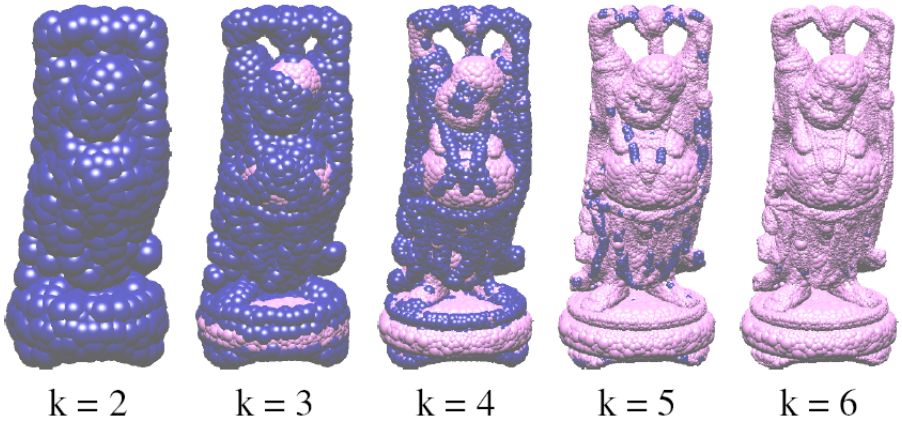
2.3.1 Methode

Om de octree te construeren beginnen we met een lege root node. Hieraan worden dan de surfels van het eerste level geconstrueerd. Het construeren van de surfels op het eerste level gebeurt door willekeurig een nieuw punt te kiezen tussen de punten P, die niet voldoende bedekt worden door surfels die reeds geconstrueerd zijn [24]. Als er nog geen surfels geconstrueerd zijn wordt er een willekeurig punt uit P gekozen. Daarna wordt naar een optimale straal ρ gezocht zodat het oppervlak, geconstrustrueerd met behulp van de punten binnen de straal, het oppervlak van de figuur in dit lokaal gebied zo goed mogelijk benadert. De straal wordt telkens met een vaste waarde vergroot en vervolgens wordt een nieuwe lokale functie gefit. Dit stopt zodra het verschil van de errorwaarde van twee opeenvolgende functies kleiner is dan een vooraf gedefinieerde drempelwaarde. De errorwaarde

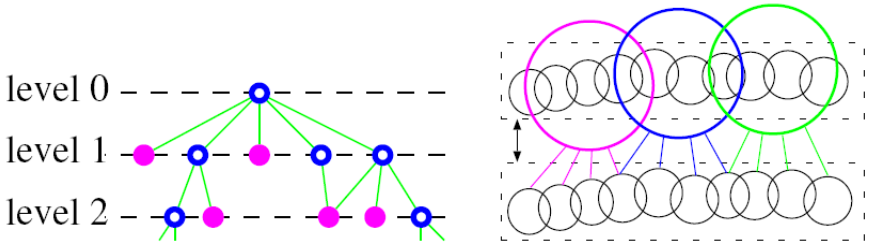
(we noemen dit de benaderingserror) wordt bekomen door het verschil te berekenen van de functiewaarde van de huidige lokale functie met de functiewaarde van de vorige lokale functie. In de praktijk blijkt het vinden van een correcte straal een moeilijke opgave als het te benaderen oppervlak op die plaats erg complex is. Er zijn dan erg veel punten nodig en de opeenvolgende berekeningen van lokale functies zorgen dan voor veel tijdsverlies. In plaats van vorige naïve methode wordt daarom een errorfunctie $E(\rho)$ gebruikt, waarbij een ρ gezocht wordt zodat $E(\rho)$ minimaal is. $E(\rho) = \epsilon(\rho)^2 + (T_{MDL}/\rho)^2\lambda$, waarbij ϵ een monotoon stijgende functie is, λ een constante is en T_{MDL} een parameter is die vooraf opgegeven wordt. λ is een gewicht afhankelijk van de punten in verzameling P en T_{MDL} geeft de afweging weer tussen de benaderingserror en het aantal gebruikte punten [28], [24].
Om de constante λ te berekenen wordt voor elk punt in P de kleinste eigenwaarde bepaald met behulp van zijn 10 dichtste buren. Dit gebeurt door voor deze 10 punten een co-variantie matrix op te stellen. De eigenwaarde in het punt wordt dan gelijkgesteld aan de kleinste eigenwaarde van de co-variantie matrix. λ wordt vervolgens gelijkgesteld aan het gemiddelde van alle eigenwaarden van de punten in P. Voor elk level wordt een vaste ρ bepaald, met $\rho_0 = L/10$ voor het eerste level, met L de diagonaal van de boundingbox van de verameling punten P. Straal ρ op level k wordt als volgt berekend: $(\rho_l^{(k)}, \rho_m^{(k)}, \rho_r^{(k)})$ $= \rho_0(g^{k+1}, g^k, g^{k-1})$, met $g = (\sqrt{5} - 1)/2$. Om de straal ρ te bepalen wordt parabolische interpolatie toegepast [49]. De resulterende straal, die de functiewaarde van E minimaliseerd, is het minimum van de parabool die door drie opeenvolgende punten (ρ_l, ρ_m, ρ_r) gaat, zodat $\rho_l < \rho_m < \rho_r$, in een interval van 0 tot 1/10 van de lengte van de boundingbox van P gefit kan worden [30]. Het construeren van nieuwe surfels in één level gaat verder tot alle punten in dit level voldoende bedekt zijn door surfels.
We hebben twee soorten surfels: interne- en bladsurfels. Een surfel is een bladsurfel als het gebied bedekt wordt door een bol $\mathcal{B}$. Een verzameling punten definieerd een bladsurfel als ze voldoen aan volgende vergelijking $o(\mathcal{B}, p_j) \geq 0.1$. Met p_j de punten binnen de surfel en $o(\mathcal{B}, p_j) = \sum_{b_j \in \mathcal{B}}(||p_j - c||)/(2/3 * \rho)$. De surfels die hier niet aan voldoen noemen we interne surfels. Deze interne surfels zullen gebruikt worden als interne knopen in de octree. Om de surfels voor het tweede level te construeren wordt vorige werkwijze toegepast op de verzameling punten van de interne surfels uit het eerste level, met een kleinder ρ. Deze werkwijze wordt recursief toegepast tot level k bereikt is, dit is het level dat alleen uit leafsurfels bestaat. We krijgen dus een verzameling van surfels van level 1 tot k.
Tenslotte wordt er een hiërarchische voorstelling geconstrueerd van de surfels (zie figuur 2.7), [18] om het object te kunnen renderen. De hiërarchische voorstelling (zie figuur 2.8) zorgt ervoor dat men op een eenvoudige manier het gewenste level van detail kan bepalen, door zelf een level te kiezen, voor het oppervlak gerenderd wordt. Zoals te zien is in figuur(2.7) voegt een lager level telkens wat meer detail aan het resultaat toe.

De constructie methode van de surfels zorgt ervoor dat verschillende surfels elkaar overlappen en dus ook lokale benaderingen van het oppervlak elkaar gedeeltelijk overlappen (zie figuur 2.9). Dit zorgt ervoor dat er in het resultaat geen gaten voorkomen [24]. Bij

het renderen van het oppervlak wordt een eenvoudige strategie gebruikt. Allereerst wordt een detail level gekozen. Vervolgens wordt gezocht naar intersecties tussen de rays die vanuit de kijkpositie geschoten werden en de surfels op dit level die zich het dichtst bij de kijkpositie bevinden. Dit gebeurt door de intersectiepunten met de rays te bepalen. Tot slot worden de resulterende punten op de straal dan geïnterpoleerd (zie figuur 2.9).



Figuur 2.7: In deze figuur is te zien hoe interne surfels (blauw) omgezet worden in leaf surfels (rood). Op het eerste level bestaat de hele figuur nog uit interne surfels. De interne surfels worden dan steeds verder opgesplitst zodat er in level 6 enkel leaf surfels overblijven.



Figuur 2.8: In de linkerfiguur zijn een aantal levels van de octree structuur te zien. De interne knopen zijn net als in de vorige figuur blauw en de leaf knopen zijn rood. In de rechterfiguur zijn de verbindingen tussen twee opeenvolgende levels in de hiërarchie van surfels te zien.

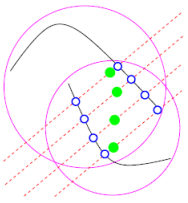
2.3.2 Level Of Detail

Als forward rendering gebruikt wordt om het resultaat te visualiseren worden de surfel gebieden eenvoudig op het scherm geprojecteerd. Hierdoor worden vele (kleine) surfels op een pixel geprojecteerd. Dit zal echter niet tot een beter resultaat leiden dan wanneer er maar een surfel per pixel gekozen wordt [24]. Daarom wordt de boomstructuur gebruikt om zicht-afhankelijke LOD verfijning toe te passen. De werkwijze die wordt toegepast werkt als volgt [24]:
De boom wordt, vanaf het hoogste level, tijdens het renderen naar beneden afgelopen als:

- de surfel regio (bol) overeenkomstig met de knoop binnen het view frustum (zie verklarende woordenlijst B) valt
- en de grootte van de surfel regio die op het scherm geprojecteerd wordt groter is dan een bepaald aantal pixels (vier pixels leidt tot een goed resultaat [24]),
- anders wordt er een pixel getekend.

2.3.3 Conclusie

Meestal is de rendering van samengestelde impliciete oppervlakken erg duur. De blending van eenvoudige oppervlak primitieven leidt tot een algebraïsch en geometrisch complex oppervlak. Dit zorgt ervoor dat het renderingprobleem, een accurate detectie van intersecties tussen een ray en een oppervlak, een hoge berekeningskost heeft [24], figuur(2.9).
De SLIM oppervlakvoorstelling die Ohtake en Belyaev [24] gebruiken voor renderingdoeleinden geeft echter geen accurate oppervlakbenadering. Strict genomen is de SLIM-gebaseerde visualisatieprocedure niet invariant wat betreft rigid transformaties (zie verklarende woordenlijst B) [24]. De methode heeft ook twee grote voordelen, namelijk de snelle rendering en berekening van de oppervlaktekromming. Ook de snelle berekening van de oppervlaktekromming heeft een nadeel: met behulp van surfels is het soms onmogelijk



Figuur 2.9: Als de intersecties gevonden zijn is het eenvoudig om deze te blenden.

om een benadering te vinden voor grote krommingsgebieden [24]. Bij globale methoden kan daarentegen altijd een correcte krommingsbenadering voor een groot gebied gevonden worden.

2.4 Multi-level Partition of Unity Implicits (MPU)

De multi-level partition of unity methode is een techniek die toelaat om een impliciet oppervlak te construeren voor grote modellen. Het hoofdidee van de partition of unity methode is dat de verzameling punten in verschillende delen wordt opgeplitst. Deze opsplitsing van de punten gebeurt met behulp van een octree. Voor elk van deze delen wordt dan een functie geconstrueerd. Daarna worden deze lokale functies geblend met behulp van lokale gewichten. De methode die hier beschreven wordt, werd ontwikkeld door Ohtake et al. [25] en is erg flexibel wat betreft de keuze van de lokale functies. Het is mogelijk om puntige kenmerken zoals randen en hoeken accuraat voor te stellen door de geschikte functie te kiezen. Het construeren van het impliciete oppervlak gebeurt in drie grote delen:

- een octree opstellen die als versnellingsstructuur gebruikt wordt,
- de lokale delen van de octree worden benaderd door verschillende soorten piecewise quadratic funties [45], (zie verklarende woordenlijst B) afhankelijk van de complexiteit van de lokale vorm wordt een ander soort functie gekozen en
- weighting functies berekenen (de partions of unity functies) die de lokale functies samen blenden.

2.4.1 Constructie van lokale functies

De methode vertrekt van een verzameling oppervlakte punten $\mathcal{P} = p_1, ..., p_N$. Er wordt aangenomen dat de inputverzameling punten $\mathcal{P}$ reeds normalen bevat $N = n_1, ..., n_N$. In de eerste stap wordt een boundingbox berekend rond alle punten $\mathcal{P}$. Aan de hand van de boundingbox wordt een octree geconstrueerd. De opsplitsing gebeurt aan de hand van een vooraf gekozen drempelwaarde ϵ_0, ook bekend als de Taubin afstand (zie functie 2.9) [37], en een minimaal aantal punten N_{min} per bladknoop [25]. De opsplitsing van de knopen gaat verder zolang het aantal punten in de knoop groter is dan de vooraf gespecifieerde drempelwaarde N_{min}. Vervolgens wordt met behulp van least-squares fitting een functie $Q'(x)$ in elke leafbox geconstrueerd voor de punten binnen een straal $R = \alpha d$, met $\alpha = 0.75$ en d de diagonaal van de boundingbox van de knoop.

$$\epsilon = \max_{|p_i - c| < R} |Q'(p_i)| / |\nabla Q'(p_i)| \tag{2.9}$$

met p_i een punt in de knoop, c het centrum van de boundingbox en ∇ de gradiënt. Daarna wordt in elke bladknoop de error van de punten berekend met behulp van de least-squares functie binnen een straal R voor de overeenkomstige $Q'(x)$. Als deze error groter is dan ϵ_0 wordt de knoop nog verder opgesplitst tot de error klein genoeg is. Vervolgens wordt voor elke knoop in de octree een piecewise quadratic functie (lokale vormfunctie) gefit door de punten in de knoop (zie figuur 2.10).

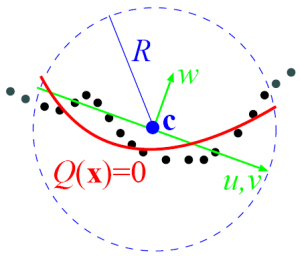
De lokale functies gedragen zich als een signed distance functie. Dit wil zeggen dat de functiewaarde nul is dicht bij het oppervlak, positief aan de binnenkant en negatief aan de buitenkant.
Voor elke knoop moet een lokale functie gekozen worden. Deze keuze hangt af van het aantal punten in de knoop en de spreiding van de normalen van deze punten. Het is namelijk niet altijd zo dat de normalen van de punten in een knoop allemaal ongeveer in dezelfde richting wijzen en daarom wordt een keuze gemaakt uit volgende drie soorten functies om de punten te fitten:

a) een algemene 3D quadric,

b) een bi-variante quadric polynomiaal (in lokale coördinaten),

c) een piecewise quadric oppervlak dat een rand of een hoek fit.

Om grote delen van het oppervlak te benaderen waar de vorm van de figuur ongeveer vlak is en bijgevolg de normalen allemaal in dezelfde richting wijzen, wordt voor algemene quadrics **a)** gekozen. Dit wil zeggen dat het maximale verschil van de gemiddelde normaal met de normalen van de punten in een lokale bol kleiner is dan $\pi/2$. In het andere geval wordt een gekeuze gemaakt tussen **b**) en **c**). Er wordt voor functies van het type **b)** gekozen als een meer vloeiende benadering noodzakelijk is en als het maximale verschil van de gemiddelde normaal van de punten in een lokale bol groter is dan $\pi/2$. Als er zich een rand of hoek in de lokale bol bevindt wordt deze gefit door een piecewise quadric **c)**. Om deze laatste functie te construeren wordt eerst nagegaan of de lokale bol daadwerkelijk een edge (richel, kam) of hoek bevat. Dit gebeurt door te berekenen of het oppervlak op deze plaats een puntige eigenschap heeft:



Figuur 2.10: De functie $Q(x)$ is een bivariate quadratic polynomiaal die door de surface punten gefit werd.

$$min_{i,j}\Big(n_i \cdot n_j\Big) < 0.9 \tag{2.10}$$

met n_i, n_j de normalen van de punten binnen een straal R van het centrum van het blad van de octree. Als dit het geval is, en het dotproduct dus kleiner is dan 0.9, bevat het oppervlak op die plaats geen puntige eigenschap en wordt een functie van het type **b)** door de punten gefit.
In het andere geval wordt gecontroleerd of er zich een hoek bevindt. Hiervoor wordt eerst een vlak geconstrueerd. De normaalvector van het vlak $n_3 = n_1 \times n_2$ wordt gedefinieerd door de normalen n_1 en n_2. Deze bepalen de maximale hoek tussen de punten in de lokale bol met straal $R = \alpha d$. Als de hoek tussen een normaal n_i, van een punt p_i uit de knoop, en het vlak voldoende groot is

$$(max_i|n_i \cdot n_3| > 0.7) \tag{2.11}$$

wordt de vorm in de lokale bol gedefinieerd als een hoek. Indien het dotproduct kleiner of gelijk is aan 0.7 is het een edge (richel, kam) (zie vergelijking 2.11) en worden de punten in twee clusters verdeeld aan de hand van hun normaal en wordt voor elke cluster opnieuw nagegaan welke type van functie gebruikt moet worden. Als aan beide functies (2.10), (2.11) voldaan is wordt de verzameling punten in drie clusters verdeeld [25], [32]. Als voorwaarde van functie (2.10) niet geldt voor de derde cluster wordt door de punten van de drie clusters een algemene 3D quadric gefit.

2.4.2 Constructie van de functies

2.4.2.1 geval a)

In dit geval wordt er een algemene quadric geconstrueerd die er als volgt uitziet: $Q(x) = x^T Ax + b^T x + c$. Met A een 3×3 matrix, b de overeenkomstige vector en c een constante. Hierbij wordt gebruik gemaakt van 9 extra punten. Dit zijn het centrum van de knoop en de 8 hoekpunten van de boundingbox van deze knoop (zie figuur 2.11).

Om de beste fit te bekomen wordt voor elk extra punt bepaald of dit een positieve bijdrage levert. Dit gebeurt door het scalair product met de dichtste zes buren te berekenen $n^i \cdot \Big(q - p^i\Big), i = 1, 2, ...6$. Indien ze niet allemaal hetzelfde teken hebben wordt het extra punt q verwijderd.

2.4.2.2 geval b)

De functie $Q(x)$ is een bivariate quadratic polynomiaal die door de surface punten gefit wordt (zie figuur 2.10). De bivariate quadric heeft volgende vorm: $Q(x) = w - (Au^2 + 2Buv + Cv^2 + Du + Ev + F)$, met (v, u, w) de coordinaten van x in het nieuwe coordinaten

systeem en A, B, C, D, E en F de coëfficiënten. Deze coëfficiënten worden berekend door volgende som te minimaliseren:

$$\sum_{p_i \in \mathcal{P}} w(p_i) Q(p_i)^2$$

Om deze polynoom te fitten zonder off-surface punten, wordt eerst een lokaal assenstel (u, v, w) geconstrueerd. Hierbij wordt gebruik gemaakt van een vlak. Dit vlak wordt geconstrueerd met c, het center van de knoop, als oorsprong. De constructie gebeurt op zo'n manier dat het vlak (u, v) orthogonaal is ten opzichte van n. Met n de normaal van c die samen valt met w.

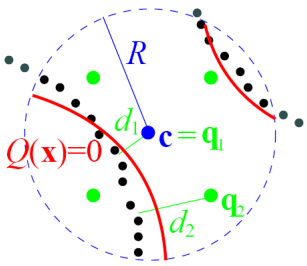
Voor een gedetailleerde beschrijving om de quadrics te berekenen verwijzen we naar de paper van Ohtake et al. [25].

2.4.3 Evaluatie van de functies

Om ervoor te zorgen dat het geconstrueerde oppervlak één geheel vormt, worden de functies geblend op plaatsen waar twee of meer knopen een gemeenschappelijke grens hebben. Dit gebeurt met behulp van de gewichten van de partition of unity functies. De som van al deze gewichten is gelijk aan 1. Het gewicht voor een functie wordt gevonden door volgende vergelijking op te lossen:

$$w_i(x) = b\Big(\frac{3|x - c_i|}{2R_i}\Big)$$

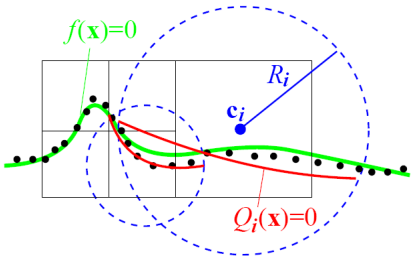
met c_i het centrum van de boundingbox van de node en R_i is 3/4 van de diagonaal van de boundingbox. $b(t)$ is de quadratische B-spline functie [42], (zie verklarende woordenlijst B) figuur(2.12).



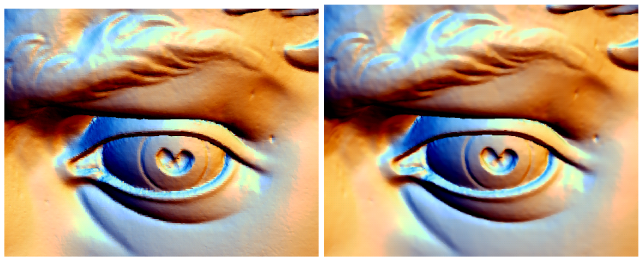
Figuur 2.11: Extra punten q die gebruikt worden om een algemene quadric te fitten.

2.4.4 Visualisatie

Om het impliciete oppervlak te visualiseren gebruikten Ohtake et al. [25] zowel de polygonisatiemethode van Bloomenthal [3] als de sphere tracing methode van Hart [13]. Uit de resultaten van Ohtake et al. (zie figuur 2.13) blijkt dat een benaderingserror van $\epsilon = 10^{-4}$ (dit is ongeveer 0.01% van de diagonaal van de grootste boundingbox) voldoende is om de kleinste kenmerken te reconstrueren.



Figuur 2.12: Een illustratie van de blendingsmethode. $Q(x)$ is een lokale benaderingsfunctie. c_i is de oorsprong van een boundingbox en R_i is 3/4 van de straal van die boundingbox. De functie $f(x)$ is het impliciete oppervlak van de volledige figuur. Verder geven de vierkanten de opsplitsing van de punten in 2D weer.



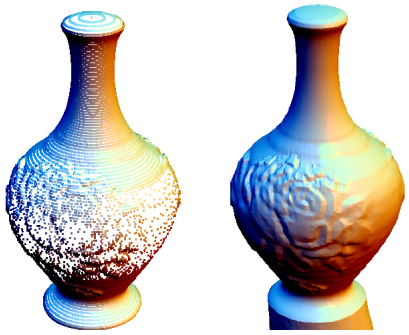
Figuur 2.13: Links: ingescand model van het oog van Michelangelo's David (gescand op een resolutie van 1mm). Rechts: reconstructie van dit model met behulp van MPU impliciete oppervlakken met nauwkeurigheid van $\epsilon_0 = 10^{-4}$. Het inscannen van dit beeld gebeurde op een uniforme manier. Dit wil zeggen dat de dichtheid van de punten overal even groot.

2.4.5 Conclusie

Het basisidee van partition of unity is de opsplitsing van het data domein in verschillende stukken. Verder heeft de MPU methode ook nog volgende eigenschappen:

- de mogelijkheid voor het maken van hoge kwaliteits impliciete oppervlakken uit grote verzamelingen data,
- de accurate reconstructie van puntige eigenschappen en
- de snelle en makkelijke toegang tot lokale vormen.

Deze eigenschappen maken de MPU methode robuust in verhouding tot de puntdichtheid. Dit wil zeggen dat ook als de puntdichtheid groot is, door de opsplitsingsvoorwaarden de methode toch snel en accuraat blijft. Deze robuustheid is het gevolg van het gebruik van een octree voor de opsplitsing van de punten. Een ander voordeel van de MPU methode is dat ze snel is omdat het een lokale methode is. De functies worden namelijk lokaal benaderd en ook lokaal geblend. Daardoor kan het impliciete oppervlak dus snel gecreëerd en geëvalueerd worden. De methode geeft bovendien goede resultaten voor onvolledige data (zie figuur 2.14).



Figuur 2.14: Reconstructie van een beeld met behulp van MPU implicits. Het beeld werd niet op een uniforme manier ingescand. Hierdoor is de dichtheid niet overal even groot en zitten er gaten in het ingescande beeld. Met behulp van de MPU methode, hierboven beschreven, worden de ontbrekende stukken gereconstrueerd.

Indien punten van range data (zie verklarende woordenlijst B) gebruikt worden moet hier speciale aandacht aan besteed worden. Het is zeer waarschijnlijk dat een deel van deze data elkaar overlapt [25]. In dit geval verbetert het resultaat, maar enkel als niet alle punten gelijk behandeld worden [25]. Dit wil dus zeggen dat er voor elk punt een

vertrouwenswaarde berekend dient te worden [25]. Er moet dus nagegaan worden voor hoeveel een overlappend punt meetelt.

2.5 Radiale basisfuncties

2.5.1 RBF methode van Carr et al.

De Radiale basisfuncties zijn basisfuncties die gebruikt kunnen worden om een globale impliciete functie te construeren. Deze functie wordt geconstrueerd voor een verzameling van surface en off-surface punten. In sommige gevallen kan deze verzameling bestaan uit alle invoerpunten. Meestal wordt de invoerdata echter in clusters opgedeeld en wordt voor elke cluster een basisfunctie berekend.
Deze Radiale basisfunctie is een functie die afhangt van de afstand tot de oorsprong, zodat $\phi(x, c) = \phi(||x - c||)$, met c een surface of off-surface punt en x het punt dat geëvalueerd wordt. Elke functie die hieraan voldoet is een radiale basis functie. Ze worden gebruikt om een oppervlak te benaderen, deze benaderingsfuncties (interpolanten) zijn van volgende vorm

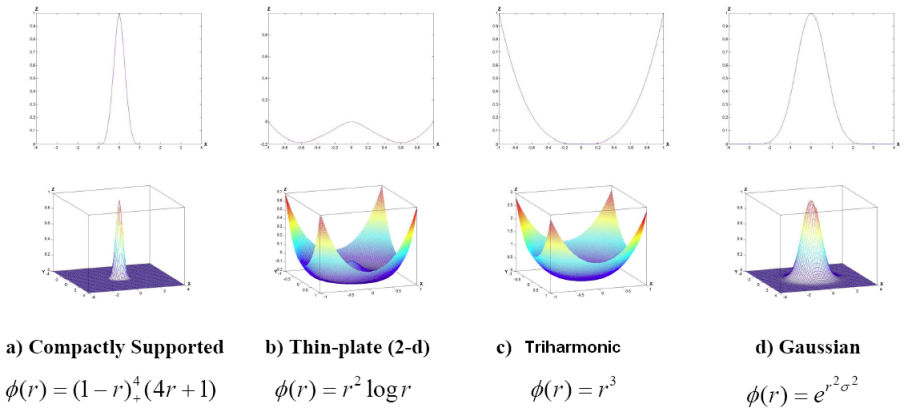
$$y(x) = \sum_{i=1}^{N} \lambda_i \phi(||(x - c_i)||)$$

met $y(x)$ de benaderingsfunctie, N het aantal radiale basis functies, c_i een surface punt en λ_i het overeenkomstige gewicht. Ze bezitten kenmerken om onvolledige objecten of objecten met gaten te herstellen. Om de functie snel en efficiënt te construeren wordt de data in verschillende clusters opgedeeld. Dit kan gebeuren met behulp van een KD-boom (zie hoofdstuk 2.5.2) of met de FMM methode die we later zullen bespreken [31], (zie hoofdstuk 2.5.1.4). In beide gevallen wordt een verzameling Radiale basisfuncties geconstrueerd die het oppervlak van een object voorstellen. Net zoals de functies die in het vorige hoofdstuk beschreven werden om impliciete oppervlakken te construeren bieden ook RBF's een compacte beschrijving van een verzameling oppervlakte elementen (zie verklarende woordenlijst B). Verder worden bij RBF's gradiënten en hogere-orde afgeleiden analytisch bepaald. Ze worden gebruikt om het dichtste nulpunt, en dus het oppervlak, te bepalen. Ze zijn continu en smooth en afhankelijk van de keuze van de basisfunctie. De basisfunctie is een functie die gebruikt wordt om een oppervlak in een bepaald gebied te benaderen. Deze basisfuncties kunnen verschillende vormen aannemen. Zo zijn er bijvoorbeeld de thin-plate spline $\phi(r) = r^2 log(r)$ (zie figuur2.15), de Gaussiaanse basisfunctie $\phi(r) = exp(-cr^2)$ (zie figuur 2.15), de multiquadric $\phi(r) = \sqrt{r^2 + c^2}$ (zie figuur 2.15), de biharmonic $\phi(r) = r$ (zie figuur 2.15) en de triharmonic $\phi(r) = r^3$ (zie figuur 2.15). De thin-plate spline wordt gebruikt om functies van twee variabelen te fitten. De Gaussiaanse basisfunctie wordt vooral gebruikt bij het construeren van neurale netwerken (zie verklarende woordenlijst B) en topologische data. De multiquadric wordt ook gebruikt bij toepassingen van topologische data. Om functies in drie dimensies te fitten wordt meestal gekozen voor biharmonic en triharmonic splines omdat deze het beste resultaat geven [6]. Om topologische data, zoals hoogtelijnen, te construeren zijn multiquadric functies dan weer het meest geschikt.

2.5.1.1 Het fitten van een enkele functie

In de paper van Carr en Beatson et al. [6] wordt een methode beschreven om een enkele functie te fitten door een verzameling van punten. Het gebruik van een enkele functie ten opzichte van meerdere heeft verschillende voordelen. De functie kan overal geëvalueerd worden om het impliciete oppervlak te construeren. Omdat het slechts één functie is moeten er, in tegenstelling tot het gebruik van meerdere functies bij de overgangen, geen blendingfuncties geconstrueerd worden. Het object wordt op een duidelijke, meer voor de handliggende manier beschreven.
De methode van Carr en Beatson heeft volgende kenmerken:

- heeft een snelheidsvoordeel ten opzichte van eerder ontwikkelde methodes
- beschikt over een center reduction techniek.
- maakt het mogelijk om een impliciet oppervlak te construeren voor grote objecten alsook voor complexe objecten
- de RBF's worden zowel voor oppervlakfitting als voor meshherstelling (hole-filling) gebruikt



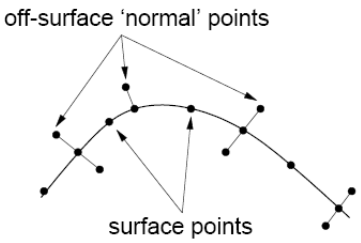
Figuur 2.15: Verschillende radiale basisfuncties die gebruikt kunnen worden als interpolatie functie. De bi-harmonic en tri-harmonic functie worden gebruikt om functies in drie-dimensies te fitten. Thin-plate splines worden gebruikt om vloeiende functies in twee dimensies te fitten. Gaussiaanse functies tenslotte worden meestal gebruikt in neurale netwerken (zie verklarende woordenlijst B) en om topologische data te construeren.

2.5.1.2 Methode

De methode van Carr en Beatson bestaat uit drie stappen:

- het construeren van een signed-distance functie.
- het fitten van een RBF door de gevonden afstandsfunctie.
- visualisatie van de gefitte RBF.

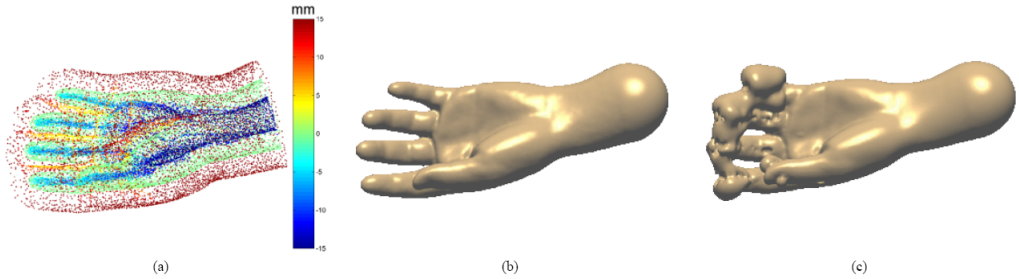
Om een triviale oplossing te vermijden en een meer vloeiend resultaat te bekomen worden punten die niet op het oppervlak liggen toegevoegd aan de invoerdata en krijgen deze niet-nulwaarden (zie figuur 2.16): $f(x_i, y_i, z_i) = 0 \quad i = 1, ..., n$ (oppervlakte punten)
$f(x_i, y_i, z_i) = d_i \neq 0 \quad i = n + 1, ..., N$ (punten die niet op het oppervlak liggen)
Een voor de hand liggende keuze voor f is een signed-distancefunctie. Hierbij wordt d_i gelijkgesteld aan de afstand tot het dichtstbijzijnde oppervlaktepunt. Punten die buiten het object liggen krijgen een positieve waarde toegekend, terwijl punten binnen het object een negatieve waarde toegekend krijgen. Experimenten hebben uitgewezen dat het het beste is om bij het toevoegen van een datapunt, aan beide zijden van het oppervlak een punt toe te voegen, in plaats van aan een enkele zijde [6]. Het is echter niet noodzakelijk om voor elk oppervlaktepunt één of meer off-surfacepunten te construeren, wat als gevolg heeft dat we niet in elk punt de normaal moeten kennen. Des te meer off-surface punten er echter gebruikt worden des te beter het resultaat zal zijn [6] (zie figuur 2.16).



Figuur 2.16: Voorbeeld van off-surfacepunten. Zeven off-surface punten geconstrueerd in de richting van de normaal, vier outside- in drie insidepunten. Zoals te zien zijn er niet voor elk surfacepunt twee off-surfacepunten geconstrueerd. Dit is zoals vermeld niet noodzakelijk voor de constructie van een impliciet oppervlak.

Als we een mesh hebben is het eenvoudig om punten toe te voegen die niet op het oppervlak liggen aangezien normalen impliciet aanwezig zijn bij de meshverbindingen van elke vertex. In het geval van een niet-gerangschikte puntenwolkdata worden normalen

geschat met behulp van een aantal lokale punten in de buurt. In het algemeen is het moeilijk om de normalen overal te bepalen, hoewel het niet kritiek is om overal normalen te bepalen, zoals reeds eerder vermeld. Als de richting van de normaal dubbelzinnig is wordt het off-surface punt op deze plaats verwijderd.
Bij de constructie van deze off-surface punten moet men verder opletten dat ze niet snijden met andere delen van het oppervlak aangezien hierdoor artefacten kunnen ontstaan (zie figuur 2.17).



Figuur 2.17: Reconstructie van een puntenwolk van een hand. Links: de puntenwolk zonder off-surface punten. Midden: het impliciete oppervlak geconstrueerd na aanpassing van de normalen en positie van de geprojecteerde punten. Rechts: het impliciete oppervlak met alle off-surface punten (ook diegene waarvan de richting dubbelzinnig is).

Het off-surfacepunt wordt daarom zo geconstrueerd dat het dichtsbijzijnde oppervlakte punt het oppervlaktepunt is dat het geprojecteerde punt creëerde.
We hebben nu een verzameling nul-waarde oppervlakte punten en niet nul-waarde oppervlakte punten. Hierin kan zich overbodige informatie bevinden. Dit wil zeggen informatie die niet nodig is om een gedetailleerde functie te fitten. Daarom zal eerst een center reduction algoritme op de verzameling toegepast worden voordat de impliciete functie geconstrueerd wordt. Hierdoor kan een impliciete surface met minder punten geconstrueerd worden, die toch erg gedetailleerd is.

2.5.1.3 RBF center reductie algoritme

Een RBF benadering gebruikt normaal alle invoerdatapunten als interpolatiepunten van de RBF. Deze invoerdata bevat meestal overbodige informatie. Hierdoor kunnen we een impliciete functie met de gewenste preciesie construeren en aanzienlijk minder punten gebruiken [6]. Om dit te bereiken wordt een greedy algoritme gebruikt. Dit kan er als volgt uitzien:

- kies een willekeurige deelverzameling van de interpolatie punten x_i en fit enkel een

RBF tussen deze. De grootte van de deelverzameling is afhankelijk van de figuur en de hoeveelheid punten waaruit deze bestaat.

- evalueer het resterende, $E_i = f_i - s(x_i)$, op alle knopen. Met f_i de te benaderen functie en $s(x_i)$ de functie die geconstrueerd wordt (zie functie 2.12).
- if $max|E_i| <$ fitting nauwkeurigheid then stop.
- else voeg nieuwe interpolatie knopen toe waar E_i groot is.
- re-fit RBF en ga naar 2.

Als ervoor gekozen wordt om in elk punt een andere nauwkeurigheid te kiezen, kan de voorwaarde in de derde stap vervangen worden door $|E_i| < \delta_i$.
De centerreductie zorgt voor minder geheugengebruik, compressie en snellere evaluatietijden, zonder dat aan nauwkeurigheid wordt ingeboet (zie figuur 2.18). Resultaten van Carr en Beatson et al. [6] hebben uitgewezen dat een nauwkeurigheid van $1.4 * 10^{-4}$ goede resultaten geeft als alle punten in de eenheidskubus tussen -1 en 1 liggen.

Figure	Number of surface points	Number of inter--polation nodes	Number of RBF centers	Peak RAM (MB)	Fitting time	Surfacing time	Relative accuracy
Face	14,806	29,074	3,564	29	68s	27s	7×10^{-4}
Hand	13,348	26,696	4,299	29	97s	32s	1×10^{-3}
Dragon	437,645	872,487	72,461	306	2:51:09	0:04:40	8×10^{-4}
Buddha	543,652	1,086,194	80,518	291	4:03:26	0:04:07	5×10^{-4}
Cherub statue	331,135	662,269	83,293	187	3:09:06	0:06:41	4×10^{-4}
Skeleton hand	327,323	654,645	85,468	188	3:08:44	0:04:04	3×10^{-4}
LIDAR statue	345,910	518,864	518,864	390	3:08:21	0:25:39	6×10^{-3}

Figuur 2.18: In bovenstaande tabel is duidelijk de grootte van de compressie te zien. Het aantal RBF centers die gebruikt worden om het impliciete oppervlak te fitten is duidelijk veel kleiner dan het aantal interpolatie punten. Het aantal interpolatie punten zijn het aantal punten na de constructie van de off-surfacepunten.

Interpolatie van de RBF functie Nu er een verzameling punten gecreëerd is kan er een interpolant $s(x)$ (zie hoofdstuk 2.12) geconstrueerd worden die de signed-distancefunctie $f(x)$ benaderd. Carr en Beatson et al. [6] definiëren dit probleem als volgt:

Gegeven een verzameling punten $X = \{x_i\}_{i=1}^N \subset \mathbb{R}^3$ en een verzameling functiewaarden $\{f_i\}_{i=1}^N \subset \mathbb{R}$, zoek een interpolant $s : \mathbb{R}^3 \to \mathbb{R}$ zodat

$$s(x_i) = f_i, \qquad i = 1, ..., N. \tag{2.12}$$

met N het aantal interpolatiepunten en $x = (x, y, z)$ voor punten $x \in \mathbb{R}^3$.

De algemene vorm van een interpolant van een radiale basisfunctie (RBF) heeft volgende vorm:

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \phi(|x - x_i|) \tag{2.13}$$

waarbij $p(x)$ een polynomiaal van lage graad is en volgende vorm heeft: $p(x) = c_1 + c_2x + c_3y + c_4z$. Deze functie doet dienst als compensatie- of correctiefunctie. Tenslotte moet de som van alle λ_i uit 2.13 gelijk zijn aan 0:

$$\sum_{i=1}^{N} \lambda_i = 0$$

Duchon [8] heeft aangetoond dat de smoothste (vloeiendste) interpolant, tevens een speciaal geval van de interpolant in functie (2.13), voldoet aan volgende vergelijking:

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i |x - x_i| \tag{2.14}$$

met polynoom p van de som RBF vergelijkingen 2.14 heeft volgende vorm: $p(x) = c_1 + c_2x + c_3y + c_4z$.
Om de interpolant $s(x)$ op te lossen hebben we een basis voor de polynomen nodig $\{p_1, ..., p_l\}$ en een basis voor de coëfficiënten $c = \{c_1, ..., c_l\}$ van polynoom p. De vergelijkingen kunnen vervolgens opgelost worden met behulp van volgende matrix:

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = B \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{2.15}$$

met

$$A_{i,j} = \phi(|x_i - x_j|), \quad i, j = 1, ..., N, P_{i,j} = p_j(x_i), \quad i = 1, ..., N, \quad j = 1, ..., l$$

In dit geval is de i-de rij van matrix P gelijk aan $(1, x_i, y_i, z_i)$, $\lambda = (\lambda_i, ..., \lambda_N)^T$ en $c = (c_1, c_2, c_3, c_4)^T$.

Om het stelsel 2.15 op te lossen kan de Gauss-Jordan methode (zie hoofdstuk A.1) of LU-decompositie (zie hoofdstuk A.2) toegepast worden. Deze methodes blijken echter niet geschikt voor grote verzamelingen inputdata [6]. Dit komt omdat bij deze methodes veel geheugen en tijd nodig is. De opslag van de linkerbenedenhoek van matrix B vereist $N(N+1)/2$ getallen. Het oplossen hiervan met een symmetric solver zoals LU-decompositie vereist $N^3/6 + O(N^2)$ flops. Voor 20.000 punten is dan ongeveer $1.5GB$ [7] nodig en 10^{13} flops. Dit is niet praktisch voor meer dan 2000 punten [6] en daarom wordt de FMM methode gebruikt om dit probleem op te lossen.

2.5.1.4 De Fast Multipole Methode (FMM)

De Fast Multipole Methode maakt gebruik van het eenvoudige feit dat als berekeningen worden uitgevoerd, oneindige precisie niet vereist en niet verwacht wordt [6].
Voor de evaluatie van een RBF wordt gebruik gemaakt van far- and near-field expansions. Dit wil zeggen dat voor de evaluatie van een punt de berekening in twee delen gebeurt. Voor de surface punten die dichtbij gelegen zijn wordt een accurate berekening gemaakt, voor de centers veraf wordt een benadering genomen. De centers worden hiervoor op voorhand hiërarchisch geclusterd in een octree [31]. Het gevolg van deze evaluatiemethode is dat deze toelaat om de RBF te berekenen tot een op voorhand bepaalde precisie en dat het een belangrijke daling is in de berekeningstijd in vergelijking met directe evaluatie.

De FMM die gebruikt werd door Carr en Beatson et al. [6] is gebaseerd op de methode ontwikkeld door Beatson en Greengard [31].
De FMM berekent een vergelijking voor de kracht in elk punt. De inputpunten (de surface punten) worden in deze methode beschouwd als particles en aan elk punt wordt een massa toegekend. Hiervoor worden niet enkel de massa en particles (zie verklarende woordenlijst B) van één box van de octree gebruikt, maar ook die van een vast aantal boxen rond de box van het punt waarvoor de kracht berekend wordt [21]. De kracht die uitgeoefend wordt op een particle in punt (x, y, z) door een particle in de oorsprong is gelijk aan $-(x, y, z)/r^3$, met $r = \sqrt{x^2 + y^2 + z^2}$ de afstand van het particle tot de oorsprong. In plaats van met een kracht te werken die een vector is, wordt de potential (vermogen of potentiaal) berekend $\phi(x, y, z) = -1/r$. Dit is een reëel getal en hiermee kan dus makkelijker gewerkt worden.

Voor een gedetailleerde beschrijving van deze methode, zie de paper van Beatson en Greengard [31].

Het Fast Multipole algoritme in de praktijk Het idee achter de FMM is zoals reeds eerder vermeld dat we het evaluatie proces van een functie willen versnellen. Het basisidee hiervan is dat we een accurate benadering van de functiewaarde berekenen voor punten die dicht bij het punt liggen dat geëvalueerd wordt en een benadering berekenen voor punten die veraf liggen. Zoals in vorige paragraaf beschreven werd, wordt in de FMM van Beatson en Greengard de kracht in een punt gebruikt om het impliciete oppervlak te construeren. Deze kracht of potentiaal bepaald de invloed van een punt of meerdere punten op andere punten.
Om dit mogelijk te maken wordt er een octree van de puntenwolk opgesteld. Om de potentiaal te berekenen die zich van een box n_i verwijderd wordt een outer expension, $Outer(n_i)$ berekend. Om de potentiaal in een box n_i te berekenen wordt een inner expansion $Inner(n_i)$ berekend. Op deze manier wordt dus in rekening gebracht hoeveel invloed naburige punten, ver en dichtbij, hebben op een punt waarvoor de potentiaal berekend wordt. Elke knoop in de octree heeft dus twee Taylor reeksen (of multipole reeksen).

Deze procedure ziet er als volgt uit:

- Bouw de octree die alle punten bevat

- Loop de octree van onder naar boven af, bereken $Outer(n_i)$ voor elke box in de boom
- Loop de octree van boven naar onder af, bereken $Inner(n_i)$ voor elke box in de boom
- Voeg voor elk blad de bijdragen van de dichtste buren en particles in het blad aan $Inner(n_i)$ toe
  - met $Inner(n_i) = (beta_{i,0}, beta_{i,1}, ..., beta_{i,p}, z_c)$
  - met $Outer(n_i) = (M, alpha_{i,1}, alpha_{i,2}, ..., alpha_{i,p}, z_c)$
  - met p het aantal punten in de box
  - met n het aantal knopen in de octree
  - met M de som van de gewichten van de punten in de box
  - met z_c het centrum in de box
  - met $z_i = (x_i, y_i, z_i)$ de punten in de box
  - met $beta_{k,i}$ de gewogen som van alle $alpha_{k-1,i}$ en kost $O(p^2)$
  - $alpha_j = \sum_{i=1,...,n}[m_i * z_i^j / j]$
  - $phi(z) = M * log(z) + sum_{j=1,...,p} alpha_j / z^j$

Voor elk box wordt dus bijgehouden hoeveel de punten erin bijdragen om de totale functiewaarde te berekenen. Om dan de functiewaarde van X te berekenen kan een benadering gebruikt worden voor punten in boxen die veraf liggen en een gedetailleerde waarde voor de boxen waar punt X in ligt.

Voor een gedetailleerde wiskundige uitleg verwijzen we naar de paper van Beatson en Greengard [31] en Greengard en Rokhlin [10].

2.5.1.5 Visualisatie

Carr en Beatson gebruiken een marching cube variant voor de visualisatie van het impliciete oppervlak. Er wordt gebruik gemaakt van een marching tetrahedra die geoptimaliseerd is voor het volgen van een oppervlak. Deze mesh optimalisatie methode [39] zorgt ervoor dat er minder driehoeken nodig zijn en dat lange driehoeken vermeden worden. Een typisch resultaat van deze methode is te zien in figuur (2.19).

2.5.2 RBF methode van Xiaojun Wu en Michael Yu Wang et al.

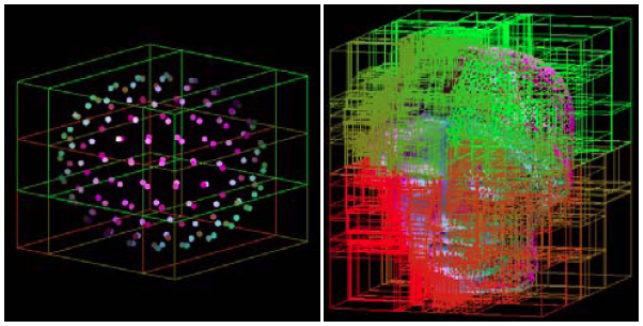
Een andere methode om radiale basisfuncties te construeren maakt gebruik van een KD-tree [58], die afwijkt van de standaard KD-boom. Bij het opstellen wordt ervoor gezorgd dat bij elke opsplitsing, beide delen gedeeltelijk overlappen. Het percentage dat deze delen overlappen wordt op voorhand opgegeven en wordt gekozen afhankelijk van de gebruikte figuur. Deze opsplitsings methode vertoond veel gelijkenissen met de partition of unity

(POU) methode, die gebruikt wordt om het globale domein op te splitsen in verschillende subdomeinen. Deze subdomeinen komen in de KD-boom overeen met een node. Bij de constructie van deze boom wordt rekening gehouden met een drietal parameters. Deze zijn Tmaximum, Tminimum en Tq. Een bladnode bevat punten en de boundingbox waarin deze punten zich bevinden. Tmaximum geeft aan hoeveel punten er zich maximaal in een subdomein mogen bevinden. Tmimimum bepaalt overeenkomstig het minimum aantal punten dat zich in het subdomein moet bevinden. Tq tenslotte geeft aan hoeveel percent overlapping er tussen de verschillende subdomeinen is (zie figuur 2.20).

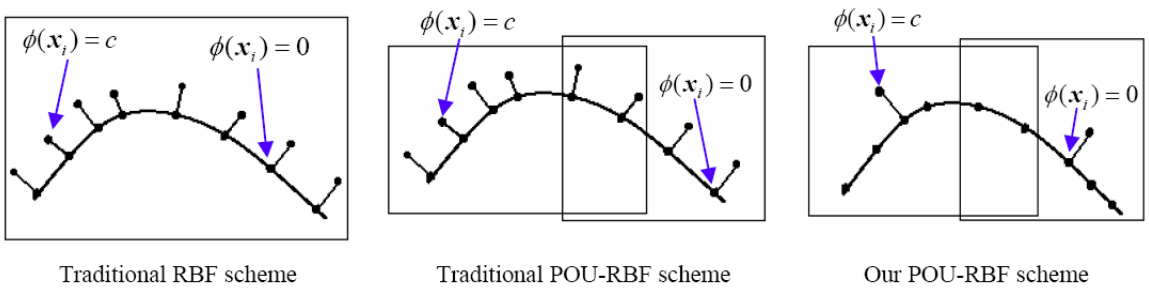
Ook bij deze methode wordt er net zoals bij de vorige methode gebruik gemaakt van off-surface punten. De constructie hiervan kan op verschillende manieren gebeuren. Als eerste kan gekozen worden om dezelfde methode te gebruiken als Carr et al. [6]. Hierbij worden voor elk surfacepunt twee off-surfacepunten geconstrueerd. Een andere methode is om slechts aan één zijde een off-surfacepunt te construeren (zie figuur 2.21).

Tenslotte kan gekozen worden voor de methode van Xiaojun Wu en Michael Yu Wang et al [58] (zie figuur 2.21). Deze methode gebruikt slechts één off-surfacepunt om een RBF voor een subdomein te berekenen. Stel box1 is de boundingbox van subdomein1. Eerst wordt het centrum P van de boundingbox berekend. Vervolgens wordt het dichtste surfacepunt hiervoor gezocht. Voor dit surface punt wordt dan een off-surface punt geconstrueerd in de richting van de normaal. De constructie van het off-surface punt nabij de rand van het subdomein kan leiden tot een onstabiele reconstructie van het oppervlak [58]. Ook de afstand van het off-surface punt tot het overeenkomstige surfacepunt heeft

Figuur 2.19: Reconstructie van het Boeddha model. De visualisatie van het impliciete oppervlak werd met behulp van het marching cubes algoritme van Carr en Beatson gedaan. Als marching cube werd een tetraheder gebruikt.

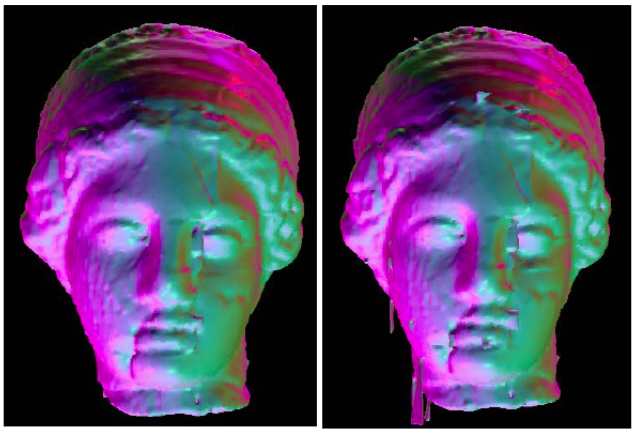


Figuur 2.20: Opdeling van de punten van een object (het Igea beeld) in overlappende subdomeinen.



Figuur 2.21: Verschillende mogelijkheden om een off-surfacepunt te construeren. Meest links is te zien hoe de off-surfacepunten op de traditionele manier aan één zijde geconstrueerd worden. In het midden wordt dezelfde constructie gebruikt voor de overlappende subdomeinen. Het meest rechtse deel laat zien hoe een off-surfacepunt geconstrueerd wordt, als er maar één per subdomein gebruikt wordt.

een grote invloed op het resultaat. Zo leidt een grotere off-surface afstand tot een minder stabiel resultaat (zie figuur 2.22). Uit onderzoek van Xiaojun Wu en Michael Yu Wang et al. blijkt dat als een off-surface punt dicht bij het center geconstrueerd wordt dit leidt tot een beter geconstrueerd impliciet oppervlak dan wanneer het off-surface punt ver van het center verwijderd is.



Figuur 2.22: Beide figuren zijn het resultaat van het gebruik van slechts één off-surfacepunt per subdomein. In figuur a) werd het off-surfacepunt op een afstand van 0.02 geconstrueerd en in figuur b) werd het off-surfacepunt op een afstand van 0.2 geconstrueerd. Zoals te zien in het resultaat is dit erg afhankelijk van de afstand van het off-surfacepunt.

De berekening van de RBF's gebeurt voor elk subdomein op dezelfde manier als in de eerste manier uit Carr et al. [6]. Dit wil zeggen dat de interpolatiefunctie er opnieuw als volgt uitziet:

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \phi(|x - x_i|)$$

met $p(x)$ de correctie polynoom, N het aantal interpolatie punten en ϕ de Radial basis functie. De matrix om de RBF te berekenen ziet er als volgt uit:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1N} & 1 & x_1 & y_1 & z_1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2N} & 1 & x_2 & y_2 & z_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \\ \phi_{N1} & \phi_{N2} & \dots & \phi_{NN} & 1 & x_N & y_N & z_N \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ x_1 & x_2 & \dots & x_N & 0 & 0 & 0 & 0 \\ y_1 & y_2 & \dots & y_N & 0 & 0 & 0 & 0 \\ z_1 & z_2 & \dots & z_N & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ \lambda_{N+1} \\ \lambda_{N+2} \\ \lambda_{N+3} \\ \lambda_{N+4} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

met $\phi_i j = \phi(|xi - xj|)$ (zie formule 2.15) en $f_1, ..., f_N$ de afstand van tot het overeenkomstige surface punt. Voor de sufacepunten zelf is deze waarde dus gelijk aan nul, voor de off-surfacepunten is dit de $L1norm$ of $-L1norm$, afhankelijk of het punt zich buiten of binne het oppervlak bevindt.

2.5.2.1 Methode

Om tot een globaal impliciet oppervlak te komen gaat men als volgt te werk. Het globale domein Ω wordt opgesplitst in M overlappende subdomeinen $\{\Omega_i\}_{i=1}^{M}$. Daarna worden er blendingfuncties $\{w_i\}_{i=1}^{M}$ gedefinieerd met een beperkte support en waarvan de som $\sum w_i$ gelijk is aan 1 over het gehele domein Ω. De blendingfuncties w_i worden berekend uit een verzameling vloeiende functies W_i door volgende normalisatie toe te passen op de functie W_i:

$$w_i(x) = \frac{W_i}{\sum_j W_j(x)}$$

De functie W_i moet continu zijn aan de grenzen van het subdomein Ω_i. Deze eigenschap kan bekomen worden door gebruik te maken van een afstandsfunctie $D_i : R^n \rightarrow [0, 1]$ en een vervalfunctie $V : [0, 1] \rightarrow [0, 1]$. De functie W_i is een samenstelling van deze afstands- en vervalfuncties $W_i(x) = V \circ D_i(x)$. De afstandsfunctie $D_i(x)$ moet aan de grenzen van het domein Ω_i gelijk zijn aan 1. Dit kan bereikt worden door de afstandsfunctie als volgt te definiëren:

$$D_i(x) = 1 - \frac{4(x - S) \cdot (T - x)}{(T - S)^2}$$

met $x \in \mathcal{R}^3$, S en T twee tegenoverliggende hoeken van de boundingbox het dotproduct tussen deze verschillende vectoren. De keuze van de vervalfunctie V hangt af van de continuïteit tussen de lokale reconstructie functies $s_i(x)$ in de globale reconstructie functie Φ. Hiervoor worden volgende functie aangeraden door Tobor et al. [38]:

$$
\begin{aligned}
C^0 : V^0(d) &= 1 - d \\
C^1 : V^1(d) &= 2d^3 - 3d^2 + 1 \\
C^2 : V^2(d) &= -6d^5 + 15d^4 - 10d^3 + 1
\end{aligned}
\tag{2.16}
$$

met C^n de graad van afleidbaarheid. Des te hoger de continuïteit C, des te precieser en dus beter de blendingfunctie is.
Voor elk van de subdomeinen wordt vervolgens een lokale reconstructie functie $s(x)$ geconstrueerd. De globale reconstructie functie Φ wordt dan gedefinieerd als: $\Phi = \sum_{i=1}^{M} s(x)w_i(x)$.

2.5.3 Conclusie

Een voordeel van beide bovenstaande methoden is dat ze goed schaleerbaar zijn. Ze maken het mogelijk om objecten met een willekeurige topologie voor te stellen door RBF's. Beiden zijn ook geschikt voor het herstellen van gaten of onvolledigheden in een model. De methode van Carr zorgt er dankzij het center reductie algoritme voor dat de reconstructietijd van de totale impliciete functie erg verminderd wordt. Langs de andere kant wordt de tijd die nodig is voor de constructie van de lokale impliciete functie van elke cluster er wel door verlengd. Voor elke cluster van punten moet er namelijk een exacte functie geconstrueerd worden, aangezien aan de hand hiervan moet berekend worden welke punten verwijderd mogen worden.
De tweede methode probeert dit te verhelpen door slechts één off-surface punt te construeren. Dit heeft zeker voordelen voor de snelheidswinst. Maar zoals reeds vermeld kan dit er in bepaalde gevallen ook voor zorgen dat het oppervlak slecht geconstrueerd wordt.
Door het globale kenmerk van de RBF voorstelling heeft het toevoegen of verwijderen van een surface punt invloed op het hele model. Om deze invloed te verminderen kan men ervoor kiezen om de data op te splitsen zoals in de tweede methode hierboven beschreven wordt.
De hiërarchische aanpak die in de volgende sectie besproken wordt biedt hier een efficiëntere oplossing voor. In dit geval kan het niveau van detail gekozen worden door een reconstructie level te kiezen in plaats van punten te verwijderen.

2.6 Hiërachische methode

De hiërarchische methode werkt op een lokale en hiërarchische manier. Dit wil zeggen dat het impliciete oppervlak bestaat uit een verzameling van lokale oppervlakbenaderingen. Verder wordt het niveau van detail bepaald door het niveau in de hiërarchie waarmee het oppervlak geconstrueerd wordt. Deze hiërarchie wordt geconstrueerd met behulp van een octree en bevat een grof naar fijn voorstelling van het object. Volgens de experimenten gedaan door Ohtake en Belyaev [26] is deze methode aanzienlijk sneller dan de benadering met globaal ondersteunde RBF's (zie hoofdstuk 2.5). Deze hiërarchische methode om range data te fitten werd voor het eerst gebruikt door Muraki [20].

2.6.1 Methode

De hiërarchische methode ontwikkeld door Ohtake en Belyaev et al. [25] is een combinatie van de voordelen van zowel lokale als globale RBF's. Er worden Compactly Supported RBF's (CSRBF) gebruikt om een gegeven verzameling van punten in een drie dimensionele ruimte te interpoleren. Het gebruik van deze lokale functies leidt tot een efficiënte berekeningsmethode en de grof naar fijn hiërarchie zorgt ervoor dat de methode ongevoelig is voor de dichtheid van de punten. Bovendien kunnen op deze manier ook grote delen ontbrekende data hersteld worden.

2.6.2 Single-level interpolatie

We beginnen met het uitleggen van de hiërachische methode voor één level, in volgende paragraaf wordt dan uitgelegd hoe de meerdere levels geconstrueerd worden.
De single-level interpolatiemethode is een methode om de punten van een enkel level te interpoleren. Beschouw hiervoor een verzameling van N punten $\mathcal{P} = p_i$ verspreid over een oppervlak. Voor elk van deze punten is er een eenheidsnormaal n_i die een oriëntatie definieert. Deze normalen worden meestal berekend aan de hand van naburige punten. Ze kunnen echter ook reeds in de data aanwezig zijn. We willen over deze data nu een impliciet oppervlak $f(x)$ construeren. De functie die hiervoor gebruikt wordt, ziet er als volgt uit:

$$f(x) = \sum_{p_i \in \mathcal{P}} \psi_i(x) = \sum_{p_i \in \mathcal{P}} [g_i(x) + \lambda_i] \phi_\sigma(||x - p_i||) \tag{2.17}$$

met $\phi_\sigma(r) = \phi(r/\sigma), \phi(r) = (1-r)_+{}^4(4r+1)$ Wendaland's compactly supported RBF, σ de support size en $g_i(x)$ en λ_i de onbekende functies en coëfficiënten.

$$(1-r)_+ = \begin{cases} r & \text{if } r \geq 0 \text{ en } r \leq 1; \\ 0 & \text{else.} \end{cases}$$

Een geschikte waarde voor σ wordt geschat uit de dichtheid van P. De functies $g_i(x)$ en de coëfficiënten λ_i worden gekozen aan de hand van volgende procedure:

- in elk punt p_i wordt een functie $g_i(x)$ gedefinieerd zodat zijn zero level-set de vorm van $\mathcal{P}$ in een kleine omgeving van p_i benadert.
- de coëfficiënten λ_i worden aan de hand van de interpolatie voorwaarden berekend zodat

$$f(p_i) = 0 = \sum_{p_i \in \mathcal{P}} [g_i(p_j) + \lambda_i] \phi(||p_j - p_i||). \tag{2.18}$$

Om de lokale benaderingsfunctie g_i te bepalen wordt voor elk punt $p_i \in \mathcal{P}$ een lokaal ortogonaal coördinaten systeem (U, V, W) bepaald. De oorsprong van het coördinaten systeem ligt in p_i. Daarna wordt een vlak geconstrueerd dat orthogonaal staat ten opzichte van de normaal n_i van p_i om de assen U en V te bepalen. De normaal W van het vlak valt samen met de richting van n_i. Daarna wordt een quadric (zie verklarende woordenlijst B), [44] geconstrueerd die P benadert in de buurt van het punt p_i.

$$w = h(u, v) \equiv Au^2 + 2Buv + Cv^2$$

Hiervan worden de coëfficiënten A,B en C bepaald via de least-squares minimalisatie

$$\sum_{(u_j, v_j, w_j) = p_j \in \mathcal{P}} \phi_\sigma(||p_j - p_i||)(w_j - h(u_j, v_j))^2 \rightarrow min$$

We stellen $g_i(x) = w - h(u, v)$ [26]. Hierdoor valt het oppervlak (of zero level-set) van $g_i(x)$ samen met de functie $w = h(u, v)$.

Zoals reeds vermeld, wordt de parameter σ, de supportgrootte van $\phi_\sigma(.)$, bepaald door de dichtheid van $\mathcal{P}$. Om deze dichtheid te berekenen wordt een octree geconstrueerd. De opsplitsing van de octree gebeurt aan de hand van de boundingbox van $\mathcal{P}$ en stopt als elke cel maximum 8 punten van $\mathcal{P}$ bevat. Daarna wordt de gemiddelde diagonaal van de blad cellen berekend. De parameter σ wordt tenslotte gelijk gesteld aan 3/4 van dit gemiddelde. Nu kan vergelijking (2.18) opgelost worden. Hiervoor gebruiken Ohtake en Belyaev et al. de preconditioned biconjate gradient method [27], met een initiële waarde voor $\lambda_i = 0$.
Een nadeel van de single-level methode is dat deze onvolledige data moeilijk kan herstellen (zie figuur 2.23).

Om dit gedeeltelijk te verhelpen moet σ vergroot worden, maar hierdoor wordt het reconstructie proces erg vertraagd.

2.6.3 Multi-level interpolatie

Om de problemen van de single-level methode, die hierboven beschreven werd, op te lossen kan een multi-scale hiërarchie gebruikt worden. Het idee hierachter is dat er meerdere $f^k(x)$ gemaakt worden, één voor elk level, en het geheel dan geblend wordt. Het eerste level bestaat uit erg weinig punten. Telkens er een volgend level geconstrueerd wordt, wordt er

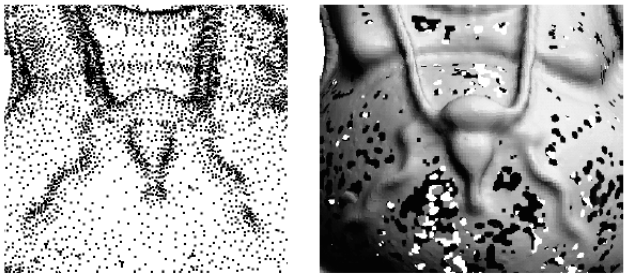
detail aan het resultaat toegevoegd.
De multi-scale hiërarchie bestaat uit een verzameling van punt verzamelingen $\mathcal{P}^1, \mathcal{P}^2, ..., \mathcal{P}^M = \mathcal{P}$, één voor elk level. Een punt verzameling $\mathcal{P}^{m+1}$ van de hiërarchie wordt geïnterpoleerd door de functie van het vorige level $\mathcal{P}^m$ te gebruiken om de functie van het volgende level te construeren. Figuur (2.24) toont de grote stappen van deze aanpak.

Om de multi-scale hiërarchie van punten verzamelingen $\mathcal{P}^1, \mathcal{P}^2, ..., \mathcal{P}^M = \mathcal{P}$ te construeren maakt men gebruik van een octree. Eerst worden de punten van $\mathcal{P}$ in een boundingbox geplaatst. Daarna wordt deze boundingbox in acht gelijke delen verdeeld, dit zijn de eerste 8 knopen van de octree. De punten verzameling $\mathcal{P}$ wordt dus geclusterd in verhouding tot de cellen van de gemaakte octree. Voor elke cel wordt ook een centrum berekend aan de hand van de punten van $\mathcal{P}$ die zich in de cel bevinden. De normaal voor dit centrum wordt verkregen door het gemiddelde te nemen van de normalen die toegekend zijn aan de punten van $\mathcal{P}$ die zich in de cel bevinden. Het gemiddelde van deze normalen wordt dan genormaliseerd. De initiële verzameling $\mathcal{P}^1$ komt overeen met het opdelen van het omvattende boundingbox van $\mathcal{P}$ in 8 gelijke delen. Om $\mathcal{P}^2$ te bekomen wordt de verzameling $\mathcal{P}$ geclusterd in verhouding tot de cellen van de gemaakte octree op het tweede level (64 delen). De berekening van de normalen voor elke cel gebeurt op dezelfde manier als de vorige verzameling $\mathcal{P}^1$.
Na de constructie van de hiërarchie wordt een grof naar fijn structuur gedefinieerd. De functie $f^0(x) = -1$ wordt als basisfunctie gebruikt. Daarna wordt de verzameling interpolatiefuncties recursief als volgt gedefinieerd:

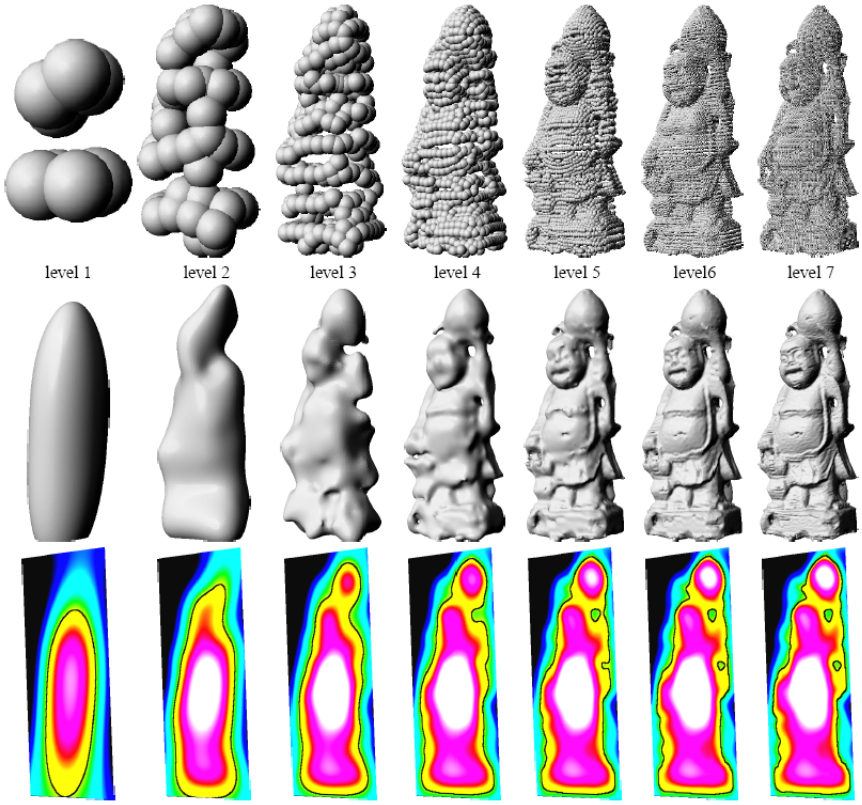
$$f^k(x) = f^{k-1}(x) + o^k(x)(k = 1, 2, ..., M).$$

waar $f^k(x) = 0$ $\mathcal{P}^k$ interpoleert. De offsetfunctie o^k ziet er als volgt uit:

$$o^k(x) = \sum p_j^k \in \mathcal{P}^k[g_i^k(x) + \lambda_i^k]\phi_\sigma^k(||x - p_i^k||)$$



Figuur 2.23: Het single-level niveau kan geen ontbrekende data herstellen. Links zijn de punten te zien waarmee het oppervlak rechts geconstrueerd werd.



Figuur 2.24: Als multiscale interpolatie voorbeeld werd het model van een monnik gebruikt. Dit model bestaat uit 60.000 punten. Bovenste rij: grootte van de supportstraal op elk niveau. De support straal die gebruikt wordt om de functie te construeren is vijf keer groter dan de bollen weergegeven in deze figuur. Middelste rij: reconstructie van het model met behulp van de multi-scale hiërarchie methode. Des te lager het level is in de octree, des te nauwkeuriger het oppervlak geconstrueerd wordt. Onderste rij: dwarsdoorsnede van het geconstrueerde oppervlak.

Deze heeft dezelfde vorm als functie 2.17 die in de vorige sectie gebruikt werd voor de single-level interpolatie. Om de lokale benaderingen $g_i^k(x)$ op te lossen wordt zoals eerder vermeld de least quares minimalisatie toegepast op $\mathcal{P}^k$. De shiftingcoëfficiënten λ_i^k worden gevonden door volgend stelsel van lineaire vergelijkingen op te lossen

$$f^{k-1}(p_i^k) + o^k(p_i^k) = 0.$$

Deze shiftingcoëfficiënten samen met de g_i functies hebben hetzelfde doel als de off-surface punten in de andere methodes. Evenals in het single-level interpolatiegeval wordt ook nu de preconditioned biconjugate gradientmethode [27] door Ohtake en Belyaev et al. [26] gebruikt om de λ_i^k's te bepalen. De supportgrootte σ^k wordt ten slotte gedefinieerd door

$$\sigma^{k+1} = \frac{\sigma^k}{2}, \sigma_1 = cL.$$

met L de lengte van een diagonaal van de omhullende boundingbox. De parameter c wordt gekozen zodat een octant van de boundingbox altijd omvat wordt door een bol met straal σ_1 gelegen in de octant. In de praktijk wordt $c = 0.75$ gebruikt door Ohtake en Belyaev et al. [25].
Het ideale reconstructieniveau M, is het level waarbij de snelheid en de accuraatheid in evenwicht zijn. Dit wil zeggen dat een volgend level enkel meer berekeningstijd vergt en geen significant detail meer toevoegd. Het ideale reconstructieniveau wordt bepaald door σ_1 en σ_0, met σ_0 de supportgrootte voor de single-level interpolatie. Ohtake en Belyaev et al. [26] behaalden goede resultaten als $M = \lceil -log_2(\frac{\sigma_0}{2\sigma_1}) \rceil$. Dit level is dus ook het diepste level van de octree dat geconstrueerd moet worden.

Compactheidsondersteunende basisfuncties geconstrueerd voor een single-level hebben essentiële beperkingen. Er is geen mogelijkheid om onvolledige data te herstellen, meer bepaald het interpoleren van onregelmatig gesampelde data en het vullen van gaten. Het vergroten van de supportgrootte σ, betekent dat de functie exacter door de puntenwolk gefit wordt. Als ervoor gekozen wordt om op deze manier de onvolledige data te herstellen wordt het reconstructieproces heel erg vertraagd. Het implementeren van een multi-scale interpolatiemethode elimineert deze problemen.
In figuur (2.25) is het resultaat van de interpolatie van onregelmatige punten te zien. Eerst werd het rechter deel van de Igea mesh voor 90% uitgedund, vervolgens werd alle verbindingsinformatie van het model verwijderd. Merk op dat het verminderen van de sampling dichtheid geen visueel zichtbare artefacts produceert in het impliciete oppervlak, gereconstrueerd uit de puntenwolk met behulp van de methode beschreven in deze sectie.

Om de impliciete oppervlakken te visualiseren worden ze gepolygoniseerd. Dit gebeurt met Boomenthal's methode [4]. Bepaalde modellen die gebruikt werden door Ohtake et al. [26] werden gepolygoniseerd met behulp van de dual-contouring methode [14]. Er zijn nog andere polygonisatie methoden mogelijk zoals de Marching Cubes [19] en extended Marching

Cubes [15]. Als postprocessing stap kan een methode voorgesteld in [27] gebruikt worden om de kwaliteit van de mesh te verbeteren.

2.6.4 Conclusie

De interpolatie methode ontwikkeld door Ohtake et al. [26] toont een goede prestatie aan bij het werken met onregelmatig gesampelde en/of onvolledige data. Het gebruik van Compactly Supported radial basisfuncties samen met de multi-scale hiërarchie maakt deze aanpak sneller dan een die gebruik maakt van globale basisfuncties. In de toekomst hoopt men om deze aanpak te verbeteren zodat ze ook grotere verzamelingen van puntoppervlakken (miljoenen punten) aankan [25]. De grote bottleneck is namelijk het oplossen van de $N \times N$ matrix met behulp van de preconditioned gradiënt methode.
De methode met de globale ondersteunende basisfuncties geeft betere resultaten voor onregelmatig gesampelde en/of ontbrekende data dan de hiërarchische methode. De hiërarchische methode heeft echter wel het voordeel dat het niveau van detail veel beter te regelen is. Elk level voegt namelijk meer detail toe. Als we dus een meer gedetailleerd resultaat willen moet met behulp van een dieper niveau het impliciete oppervlak gereconstrueerd worden. Om meer detail te krijgen bij de globale methode, die gebruikt maakt van radial basisfuncties 2.5, moet de boom opnieuw opgesteld worden en het aantal punten eventueel gepruned, met behulp van het center reductie algoritme, afhankelijk van de foutenmarge die gekozen werd.
We kunnen dus besluiten dat de hiërarchische methode de beste is als er weinig of geen onvolledigheden in de originele data zitten. Om ontbrekende data te herstellen is de globale methode nog steeds de beste.

Figuur 2.25: Links: puntenwolk waarvan een groot deel van de punten aan de rechterzijde van het beeld ontbreekt. Rechts: reconstructie van het model met behulp van de multi-scale hiërarchische methode.

2.7 Conclusie

We kunnen besluiten dat we met elke bovenstaande techniek goede resultaten kunnen behalen afhankelijk van de input data. Des te meer informatie de invoerdata over het object bevat, des te beter het resultaat zal zijn. Dit wil zeggen dat we met een polygonenmesh die uit een groot aantal uniforme driehoeken bestaat erg goede resultaten kunnen behalen. Als deze daarentegen bestaat uit driehoeken van verschillende grootte met eventueel ontbrekende delen zal het resultaat heel wat minder goed zijn. Deze redenering gaat ook op als we vertrekken van een puntenwolk waarvan de dichtheid niet overal even groot is. De beste methode om te gebruiken als men over een niet dense dataset beschikt is de globale methode van Carr [6] aangezien de ontbrekende data op een correcte manier kan herstellen. In het omgekeerde geval, als men over puntenwolk beschikt met een grote dichtheid is de hiërarchische methode van Ohtake [26] de meest geschikte manier om een impliciet oppervlak te construeren. Deze methode kan ontbrekende delen vrij goed herstellen maar is in de eerste plaats aanzienlijk sneller. De SLIM methode [24] (zie hoofdstuk 2.3) is eveneens een methode om op een snelle manier een impliciet oppervlak te construeren en weer te geven. Een nadeel is echter dat het resultaat minder accuraat is. Het is namelijk niet mogelijk om met deze methode alle krommingen uit de originele figuur te construeren. De methode van de MPU implicits tenslotte heeft heel wat overeenkomsten met de hiërarchische methode. Deze methode maakt ook gebruik van een octree, is erg snel en kan ook ontbrekende data herstellen. Het construeren van de lokale functies is echter complexer dan bij de hiërarchische methode. Aangezien de resultaten ongeveer even goed zijn kunnen we besluiten dat de hiërarchische methode de voorkeur geniet boven de methode van de MPU implicits.

Hoofdstuk 3

Visualisatie

3.1 Polygonisatie

3.1.1 Marching cubes

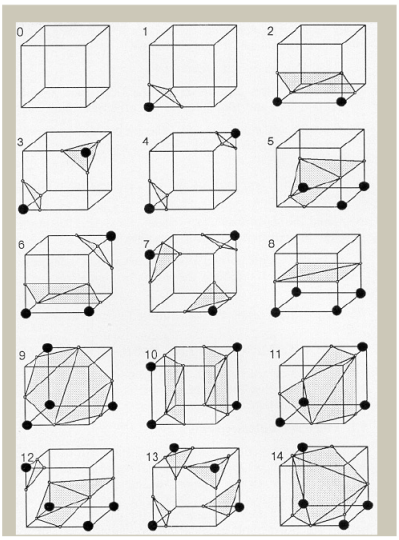
Marching cubes is een algoritme dat gebruikt wordt om een mesh van driehoeken te construeren. Deze polygonisatiemethode geeft dus een polygonale (i.e. parametrische) voorstelling van het impliciete oppervlak [4]. Het algoritme hiervoor werd voor het eerst beschreven door Lorensen en Cline [19] in 1987. De mesh wordt berekend aan de hand van het impliciete oppervlak, ook wel iso-surface (zie verklarende woordenlijsts B) genoemd. Vanaf een handmatig opgegeven startpositie wordt een plaats gezocht waar het oppervlak zich bevindt, hier wordt vervolgens een kubus geconstrueerd. Daarna wordt in elke richting van een zijde van deze kubus gecontroleerd of er nog een deel van het oppervlak ligt. Indien er nog een deel van het oppervlak ligt wordt op die plaats, vlak naast de vorige kubus dus, ook een kubus geconstrueerd die even groot is. Tenslotte wordt op de nieuw geconstrueerde kubussen recursief hetzelfde toegepast. Dit gaat verder tot er geen nieuwe kubussen gevonden worden waar een deel van het oppervlak in ligt. Doordat alle patches, gecreëerd door het marching cubes algoritme, in dit geval driehoeken, van alle kubussen van de iso-surfaces met elkaar verbonden worden krijgen we als resultaat een oppervlak. De marching cubes methode kan zowel in 2D als in 3D gebruikt worden [22].

3.1.1.1 Algoritme

Vanaf een startpositie wordt gezocht naar een plaats waar zich het oppervlak bevindt. Een voorwaarde is dat de startpositie zich in het te construeren object bevindt. Vervolgens wordt, door bij x,y en z cöordinaat een willekeurig klein getal op te tellen, het oppervlak gezocht. Als het oppervlak gevonden is wordt er op die plaats een kubus geconstrueerd, die gedeeltelijk in en gedeeltelijk buiten het oppervlak ligt. Daarna wordt in elke richting van een zijde van deze kubus gecontroleerd of er nog een deel van het oppervlak ligt. Indien er nog een oppervlak ligt wordt op die plaats, vlak naast de vorige kubus dus, ook een kubus geconstrueerd met de zelfde afmetingen als de eerste. Voor deze nieuwe kubussen

wordt ook gecontroleerd of er een deel van het oppervlak naast ligt. Dit stopt zodra er geen nieuwe kubussen meer gevonden worden waar een deel van het oppervlak in ligt.
Elke kubus is gedefinieerd door acht hoekpunten. Voor elk van deze acht punten wordt het impliciete oppervlak geëvalueerd. Als de kubus zich volledig binnen of buiten het oppervlak bevindt en dus nergens intersecteert, moet deze niet verwerkt worden. We weten dat een kubus het oppervlak intersecteert als de acht punten van de kubus niet allemaal hetzelfde teken hebben. Als met andere woorden de functiewaarden in deze punten niet allemaal negatief of allemaal positief zijn. Aangezien de waarde van een gridpunt positief of negatief is en er acht punten zijn, kan een oppervlak op $2^8 = 256$ mogelijke manieren een kubus snijden. Het marching cubes algoritme specifieert hoe in elk van deze 256 gevallen polygonen voor het oppervlak gegenereert kunnen worden. Doordat sommige gevallen dankzij symmetrie hetzelfde zijn, zijn er slechts 14 unieke gevallen.

Als er maar één hoekpunt een verschillend teken heeft, kan een enkele driehoek getekend worden. Deze driehoek intersecteert de randen die in deze hoek samenkomen. De normaal van deze driehoek wijst van de hoek weg. Aangezien er acht hoeken zijn kan dit geval dus acht keer voorkomen (nummer 1 in figuur(3.1))



Figuur 3.1: De verschillende mogelijkheden waarmee een kubus het impliciete oppervlak kan intersecteren.

. Als de normalen naar de hoek wijzen zijn er opnieuw acht configuraties mogelijk. Deze worden echter niet als unieke gevallen beschouwd [19].
Als er twee hoekpunten een verschillend teken hebben zijn er drie opstellingen mogelijk.

De hoeken kunnen tot dezelfde rand behoren, tot dezelfde rand van de kubus of diagonaal ten opzichte van elkaar gepositioneerd zijn (nummers 2,3 en 4 in figuur(3.1)).
Er zijn opnieuw drie verschillende opstellingen mogelijk als drie hoekpunten een verschillend teken hebben. Ze kunnen tot 0,1 of 2 dezelfde randen behoren (nummers 5,6 en 7 in figuur(3.1)). Indien ze tot 2 randen behoren vormen ze een 'L'.
Als vier hoekpunten een verschillend teken hebben zijn er zeven unieke mogelijkheden. Er zijn 0,2,3 or 4 randen die gemeenschappelijk zijn. Als er drie randen gemeenschappelijk zijn, zijn er drie varianten (nummers 8,9,10,11,12,13 en 14 in figuur(3.1)).
Tot slot is er nog het vijftiende geval, nummer 0. In dit geval is er geen intersectie met het oppervlak en moeten er dus ook geen polygonen geconstrueerd worden.

Het marching cubes algoritme dat gebruik maakt van de Bloomenthalmethode [4] gaat als volgt tewerk. Er wordt een startpunt en gewenste grootte voor de driehoeken meegegeven. Als het object waarvoor een oppervlak geconstrueerd moet worden zich rond of in de oorsprong bevindt kan hiervoor het punt $(0, 0, 0)$ gekozen worden. Voor dit punt wordt dan gezocht waar het oppervlak zich bevindt. Dit wil zeggen dat er een negatieve en een positieve functiewaarde gezocht wordt die het dichtst bij dit punt liggen. Als beide punten gevonden worden, wordt naar het oppervlak toe geconvergeerd. Op deze plaats (het startpunt) wordt nu een kubus geconstrueerd. De grootte van deze kubus is afhankelijk van de vooraf gekozen grootte van de driehoeken. Vervolgens worden de driehoek(en) voor deze kubus geconstrueerd.
Om te bepalen hoeveel en welke driehoeken geconstrueerd moeten worden, wordt eerst bepaald welke hoekpunten binnen en buiten het oppervlak vallen. Als dit bepaald is weten we tot welke van de vijftien gevallen deze kubus behoort. Hiermee kan bepaald worden welke zijden het oppervlak snijden. Er zijn drie snijpunten nodig om een driehoek te construeren. De positie op de randen waar deze snijpunten zich bevinden kan met berekenen met behulp van lineaire interpolatie [2], (zie verklarende woordenlijst B). Vervolgens worden voor deze drie punten de normalen berekend en wordt de driehoek getekend.
Tot slot worden de aanliggende kubussen geconstrueerd en als ze met het oppervlak intersecteren worden ook hiervoor driehoeken getekend. Het construeren van aanliggende kubussen gaat verder tot er geen nieuwe kubussen gevonden worden waarin een deel van het oppervlak zich bevindt. Het resultaat van dit proces is een aaneengesloten reeks kubussen waarin zich een vloeiend oppervlak bevindt dat het impliciete oppervlak (zie figuur 3.2) benadert.

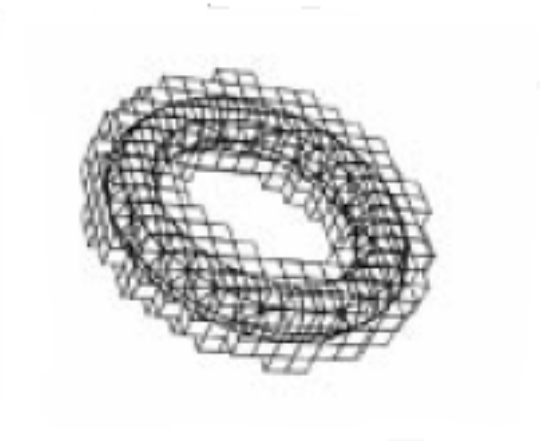
In plaats van gebruik te maken van een kubus als polygonisatie cel is het ook mogelijk om een tetraheder te gebruiken [4]. Deze manier zorgt voor een meer gedetailleerd resultaat. De constructie is echter verschillend. Zoals reeds vermeld wordt een kubus onmiddellijk gepolygoniseerd, een tetraheder daarentegen wordt eerst opgesplitst in kegels. Bij deze kegels kan men zestien gevallen onderscheiden, wat één meer is dan bij de kubussen.

3.2 Raytracing

Het raytracen van een object is complexer en meer tijdsintensief dan het polygoniseren. Het resultaat van raytracing zal er daarentegen beter uitzien. Deze methode is erg tijdsintensief doordat er vele rays nodig zijn voor het opbouwen van een beeld. Voor elke ray wordt de intersectie met het impliciete oppervlak moet vervolgens met behulp van een root finding algoritme gezocht worden [36]. Een root finding algoritme zoekt de nulpunten van een functie. In dit geval de nulpunten van de intersectie van de ray met het impliciete oppervlak. Stewart [36] stelt verschillende methoden voor om dit proces te versnellen.

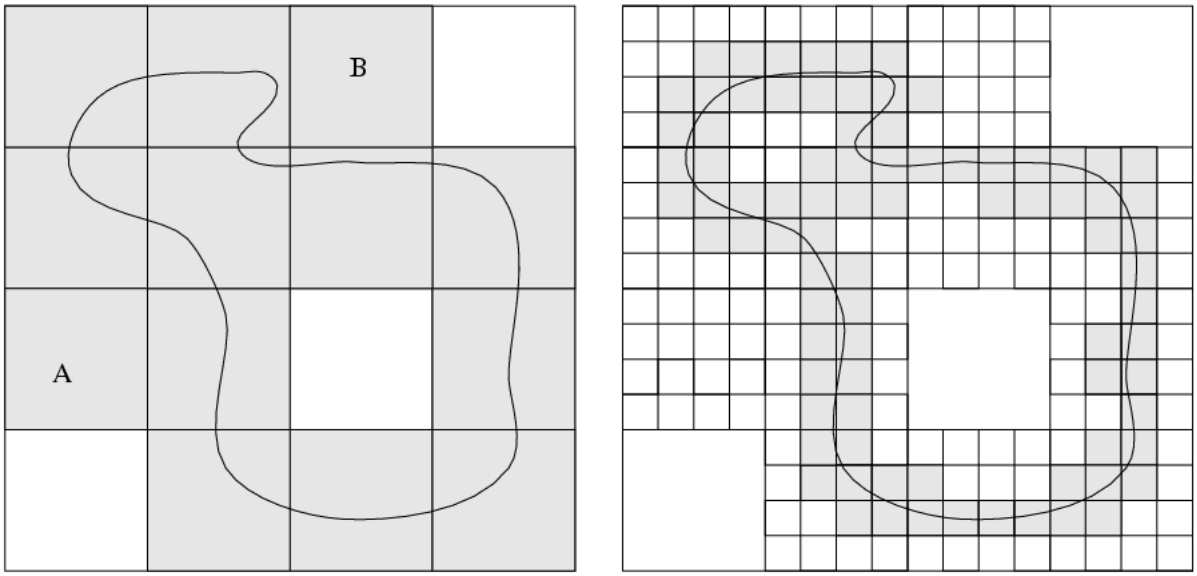
3.2.1 Methode

Een manier om het raytracingproces te versnellen is het opdelen van de ruimte in verschillende gebieden en voor elk van hen te bepalen of ze volledig binnen of buiten het object liggen. Dit kan op een eenvoudige manier omdat we weten dat voor een impliciete functie, $F(x, y, z) : \mathcal{R}^3 \rightarrow \mathcal{R}$, $F(x, y, z) > 0$ wil zeggen dat het punt binnen het oppervlak ligt en $F(x, y, z) < 0$ dat het punt buiten het oppervlak ligt. Elk gebied krijgt een waarde leeg, als ze zeker geen deel van de functie bevatten, of te onderzoeken, als ze een deel van de functie kunnen bevatten (zie figuur 3.3). Als voor een boundingbox de functiewaarde van alle hoekpunten kleiner is dan nul, ligt deze volledig buiten het oppervlak. In het omgekeerde geval, als de functiewaarde van alle hoekpunten groter is dan nul, ligt deze volledig binne het oppervlak. De opdeling blijft verder gaan tot de kleinste cellen kleiner of even groot



Figuur 3.2: De verzameling aaneengesloten kubussen die het oppervlak van het object bevatten. In dit geval een torus.

zijn als een vooraf bepaalde waarde.



Figuur 3.3: Links: initiële opdeling in lege en te onderzoeken cellen van een implicit surface in 2D. De witte vakjes geven een leeg gebied aan, de grijze zijn cellen waarvoor onderzocht moet worden of ze een deel van de functie bevatten. Rechts: Verdere opdeling van de cellen.

De methode die daarna gebruikt wordt noemt men ray marching. Elke cel van de opgedeelde ruimte wordt in een vaste volgorde afgelopen, om het aantal berekeningen te verminderen. De cellen worden gesorteerd van dichtste naar verste cel afhankelijk van het initiële intersectie punt van het impliciete oppervlak met de ray. Vervolgens wordt gezocht naar de tweede intersectie tussen de ray en het impliciete oppervlak, om te bepalen waar de ray het object verlaat. Als een lege cel gevonden wordt, gaat de ray door alle vorige cellen. Deze kunnen dan onderzocht worden op intersectie, te beginnen met de verste. Indien niet, dan wordt voor de volgende cel hetzelfde gedaan. Indien wel, dan kan men stoppen. Om het intersectiepunt in het interval van een cel te vinden worden analytische methoden zoals regula falsi (zie verklarende woordenlijst B) gebruikt. Er wordt een analytische methode gebruikt omdat het impliciete oppervlak een polynoom van hoge graad is. Hiervoor zijn er geen snellere methoden beschikbaar. Zou het een polynoom van de tweede, derde of vierde graad zijn dan bestaan er andere methoden om het nulpunt te vinden. Zoals reeds vermeld, zoeken we naar nulpunten van de samengestelde functie van de ray en de impliciete functie.

De nulpunten geven dan aan waar de ray het impliciete oppervlak snijdt [12].

3.2.1.1 Conclusie

Het marching cubes algoritme is vrij eenvoudig in vergelijking met raytracing. Een nadeel is echter dat men het aantal driehoeken moet bepalen dat een goed resultaat weergeeft. Dit kan enkel gebeuren door de grootte van de te construeren driehoeken te wijzigen. Als deze te klein gekozen wordt, worden er te veel driehoeken geproduceerd. Als deze te groot gekozen wordt, krijgen we maar weinig driehoeken en gaat er detail verloren.

Hoofdstuk 4

Implementatie

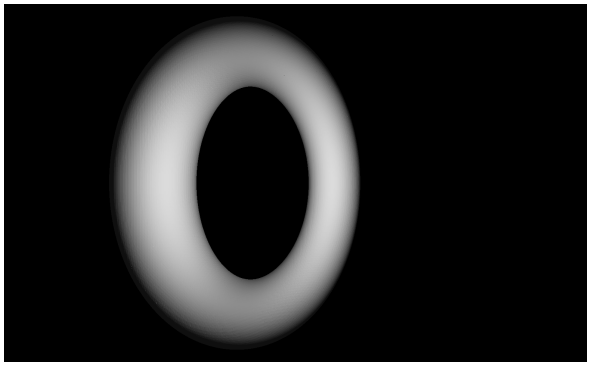
We hebben ervoor gekozen om drie technieken te implementeren, twee globale en een hiërarchische methode. Voor de globale methode werd de paper van Carr [6] en de paper van Wu [58] gebruikt. De hiërarchische methode is gebaseerd op de paper van Ohtake [28]. Als visualisatiemethode werd de Bloomenthal [4] polygonisatie methode gebruikt die eerder beschreven werd (zie hoofdstuk 3.1.1). De eerste stap bij deze methodes is het inlezen van een ply file. Dit gebeurt met behulp van de Trimesh library [40]. Vervolgens wordt de methode uit de overeenkomstige paper toegepast.
De gebruikte modellen waarvoor implicit surfaces geconstrueerd werden is de stanford Bunny en Buddha. We gebruiken verschillende resoluties van deze modellen om het effect hiervan op de verschillende methodes te onderzoeken. In tabel 4.1 wordt een overzicht gegeven van de resoluties van de Bunny en de Buddha.

Als implementatie worden eerst een aantal functies voorgesteld die handmatig werden opgesteld en gevisualiseerd werden met de Bloomenthal methode [4]. Vervolgens werden de paper van Carr [6], Wu [58] en Ohtake [28] gebruikt om impliciet oppervlak te construeren vertrekkende van een ply bestand. Het ply bestand bevat een verzameling surface punten die samen een figuur voorstellen.

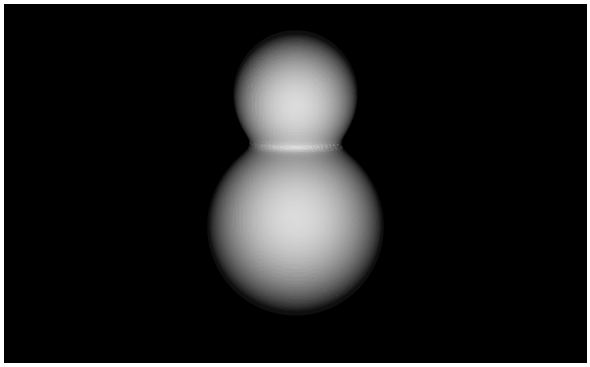
De complexiteit van een impliciete functie is zoals eerder vermeld onbeperkt. Deze functies kan men ook zelf samenstellen. In wat volgt worden een aantal van deze functies voorgesteld, beginnende met eenvoudig en eindigend met vrij complex. In figuur(4.1) is een torus te zien, de functie van deze figuur is vrij eenvoudig, net zoals deze van twee geblende bollen (zie figuur 4.2). In figuur(4.3) daarentegen is een 'wiffle cube' te zien. De functie die deze figuur beschrijft is al wat complexer. Ten slotte is in figuur(4.4) de afbeelding van een 'jack' te zien. Zoals te zien is de functie van deze figuur vrij complex. In tabel 4.2 is een overzicht van het aantal gebruikte driehoeken en de tijd die nodig was om deze functies te visualiseren met behulp van de Bloomenthal methode [4].

	Bunny	Buddha
resolutie	aantal surface punten	aantal surface punten
res4	453	7108
res3	1889	32328
res2	8171	144647
res	35947	543652

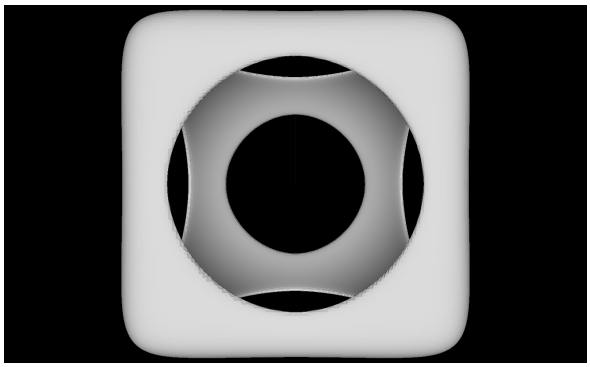
Tabel 4.1: Tabel met resoluties van de stanford Bunny en Buddha.



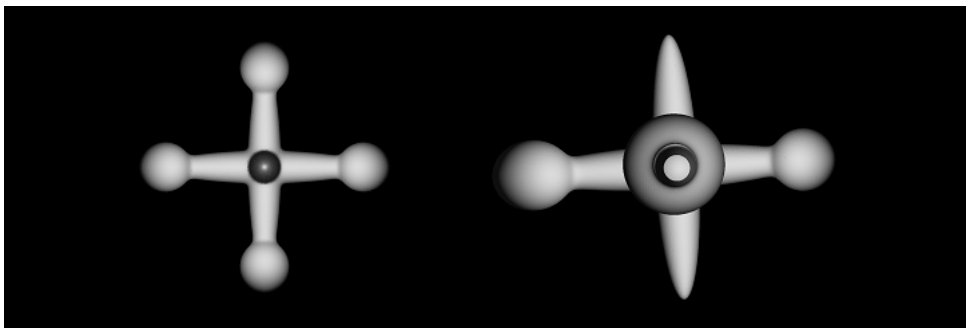
Figuur 4.1: De polygonisatie van een torus, met als vergelijking: $((x^2+y^2+z^2+R^2-r^2)*(x^2+y^2+z^2+R^2-r^2))-(4.0*R^2*(x^2+y^2)$, met r de binnenste straal en R de buitenste.



Figuur 4.2: De polygonisatie van twee geblende bollen, met als vergelijking: $(x^2+y^2+z^2-1)*(x^2+(y-1.5)^2+z^2-0.5)-0.015$.



Figuur 4.3: De polygonisatie van een wiffle cube, met als vergelijking: $1 - (a^2 * x^2 + a^2 * y^2 + a^2 * z^2)^{-6} - (b^8 * x^8 + b^8 * y^8 + b^8 * z^8)^6$, met $a = 1/2.3$ en $b = 1/2$.



Figuur 4.4: De polygonisatie van een jack, met als vergelijking: $(1/(x^2/9 + 4y^2 + 4z^2)^4 + 1/(y^2/9 + 4x^2 + 4z^2)^4 + 1/(z^2/9 + 4y^2 + 4x^2)^4 + 1/((4x/3 - 4)^2 + 16y^2/9 + 16z^2/9)^4 + 1/((4x/3+4)^2 + 16y^2/9 + 16z^2/9)^4 + 1/((4y/3-4)^2 + 16x^2/9 + 16z^2/9)^4 + 1/((4y/3+4)^2 + 16x^2/9 + 16z^2/9)^4) - 1/4 - 1$.

figuur	aantal driehoeken	tijd
torus	219752	21 sec.
wiffle cube	348036	55 sec.
jack	329068	26 sec.
twee geblende bollen	77624	5 sec.

Tabel 4.2: Tabel met resultaten van vier verschillende figuren.

4.1 Globale methode

4.1.1 Methode van Carr et al.

De eerste methode is gebaseerd op de paper van Carr [6], (zie hoofdstuk 2.5). In deze paper worden er voor elk surface punt eerst off-surface punten geconstrueerd. Als de richting van de normaal van een surface punt dubbelzinnig is wordt er geen off-surface punt voor geconstrueerd. Verder moet men opletten dat ze niet snijden met andere delen van het oppervlak aangezien hierdoor artefacten kunnen ontstaan (zie hoofstuk 2.5). Deze intersectie kan vermeden worden door ervoor te zorgen dat het te construeren off-surface punt het dichtst ligt bij het overeenkomstige surface punt. Een gevolg hiervan is dat de afstand tussen surface en off-surface punt niet overal gelijk is. Aangezien het om een klein percentage van de punten gaat heeft dit geen invloed op het resultaat.
Daarna verdelen we de punten met behulp van een KD-boom in verschillende clusters. Vervolgens wordt het center reductie algoritme uit de paper van Carr toegepast op elke cluster. Hierbij beginnen we met een kleine willekeurige deelverzameling van de cluster. Door deze punten wordt dan een RBF gefit. Vervolgens worden er surfacepunten uit de cluster die nog niet in deze deelverzameling aanwezig zijn toegevoegd. Daarna wordt opnieuw een RBF door de punten in de deelverzameling gefit. Dit gaat verder tot de error kleiner is dan een vooropgestelde drempelwaarde.
De punten in elke cluster worden daarna samengevoegd in één verzameling. Door de punten in deze verzameling wordt dan een RBF functie gefit. Om de lambdas voor elk punt te bepalen werd de LU decompositiemethode (zie hoofdstuk A.2) geïmplementeerd. Voordien hadden we de Gauss-Jordan methode (zie hoofdstuk A.1) geïmplementeerd, maar deze is aanzienlijk trager dan de LU decompositiemethode. Tenslotte wordt deze functie gevisualiseerd met behulp van de Bloomenthal methode.

De resultaten in tabel 4.3 werden verkregen door bij het center reductie algoritme telkens meerdere punten per iteratie toe te voegen. Dit aantal is geen vast aantal maar wordt bepaald door de grootte van error. Op elke plaats waar de error groter is dan een vooraf opgegeven drempelwaarde wordt het dichtst bijzijnde punt uit de cluster aan de deelverzameling toegevoegd.

De reconstructie gebeurde met behulp van een biharmonische functie. Voor de resultaten in tabel 4.4 gebeurde de reconstructie met behulp van een triharmonische functie.

De figuren (4.5) die gerecontrueerd zijn met behulp van de biharmonische functie geven duidelijk betere resultaten dan deze gereconstrueerd met behulp van de triharmonische functie (4.6). Dit is volgens de verwachting van de resultaten in de paper van Carr [6]. Hierin werd aangegeven dat de biharmonische functie de beste details construeerd. Dit komt omdat de biharmonische functie $\phi(r)$ overeenkomt met het speciale geval (functie 2.14) dat het smoothste resultaat geeft voor een 3D model.

De resultaten in tabel 4.5 en 4.6 werden verkregen door bij het center reductie algoritme telkens slechts één punt per iteratie toe te voegen.

De resultaten in de tabellen bestaan uit 7 kolommen. De eerste kolom geeft de precisie

res4				
precisie	tree	center reduction	construction	polygonisatie
all	110 ms.	n.a.	44 min. 35 sec.	9 min. 14 sec.
0.014	110 ms.	1 min. 31 sec.	14 sec. 500 ms.	1 min. 38 sec.
0.0014	120 ms.	12 min. 7 sec.	6 min. 55 sec.	5 min. 9 sec.
0.00014	80 ms.	13 min. 39 sec. 680 ms.	8 min. 17 sec. 950 ms.	5 min. 14 sec.

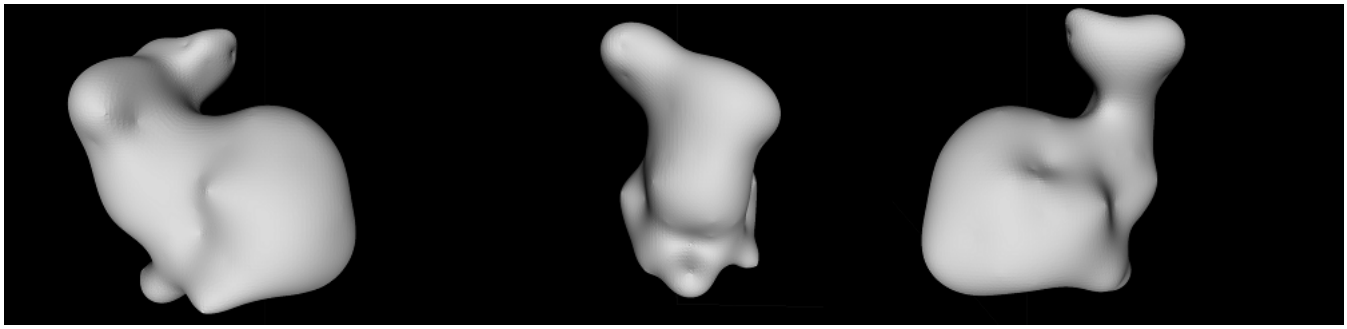
res4		
precisie	aantal driehoeken	aantal oppervlakte punten
all	122004	453
0.014	112936	79
0.0014	121708	247
0.00014	120408	309

Tabel 4.3: Tabel met resultaten van de stanford Bunny op resolutie Res4. De reconstructie gebeurde met behulp van een biharmonische functie. Bij het center reductie algoritme werd telkens meer dan één punt per iteratie toegevoegd.

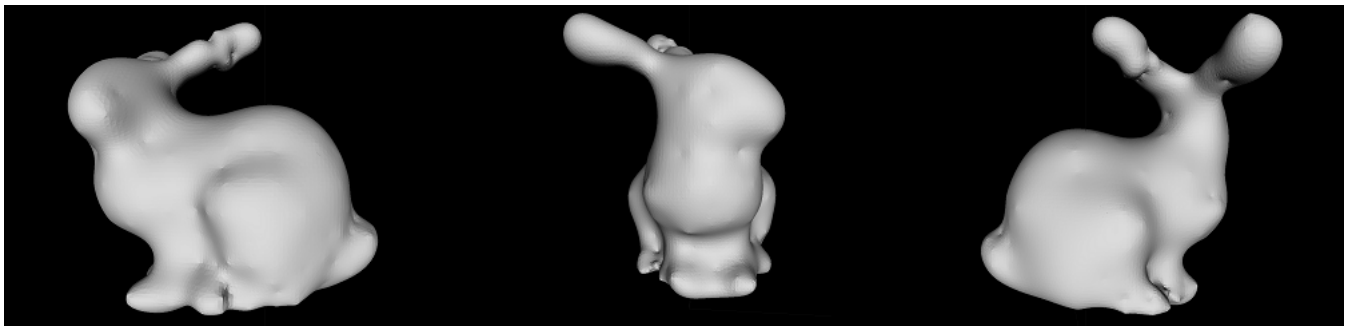
res4				
precisie	tree	center reduction	construction	polygonisatie
all	110 ms.	n.a.	43 min. 18 sec.	4 min. 26 sec.
0.014	110 ms.	1 min. 32 sec.	13 sec. 830 ms.	57 sec 610 ms.
0.0014	120 ms.	2 min. 2 sec.	51 sec. 150 ms.	45 sec. 630 ms.
0.00014	80 ms.	3 min.	1 min. 23 sec. 30 ms.	4 sec. 62 ms.

res4		
precisie	aantal driehoeken	aantal oppervlakte punten
all	44228	453
0.014	22120	77
0.0014	11164	121
0.00014	2764	148

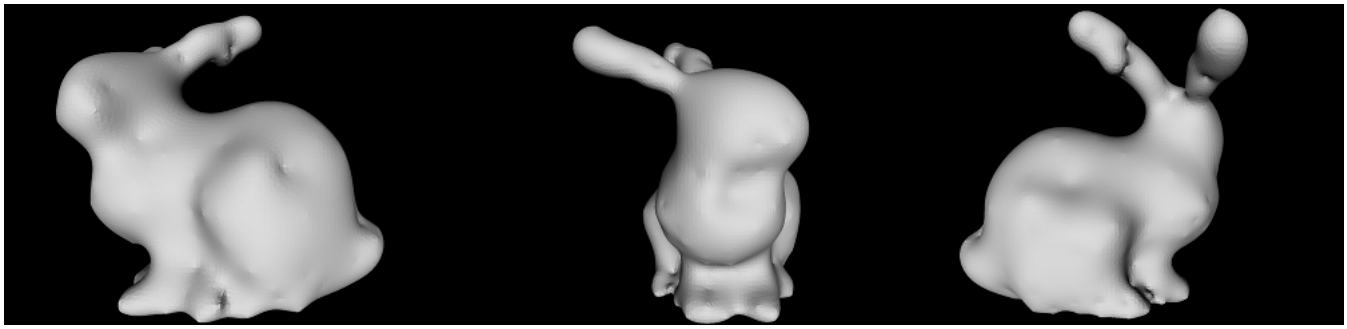
Tabel 4.4: Tabel met resultaten van de stanford Bunny op resolutie Res4. De reconstructie gebeurde met behulp van een triharmonische functie. Bij het center reductie algoritme werd telkens meer dan één punt per iteratie toegevoegd.



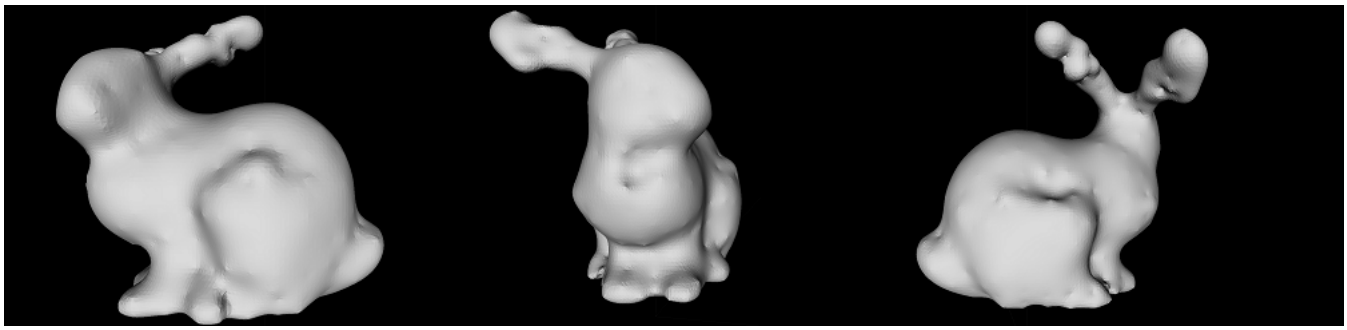
(a) Het center reductie algoritme toegepast met een precisie van 0,014.



(b) Het center reductie algoritme toegepast met een precisie van 0,0014.

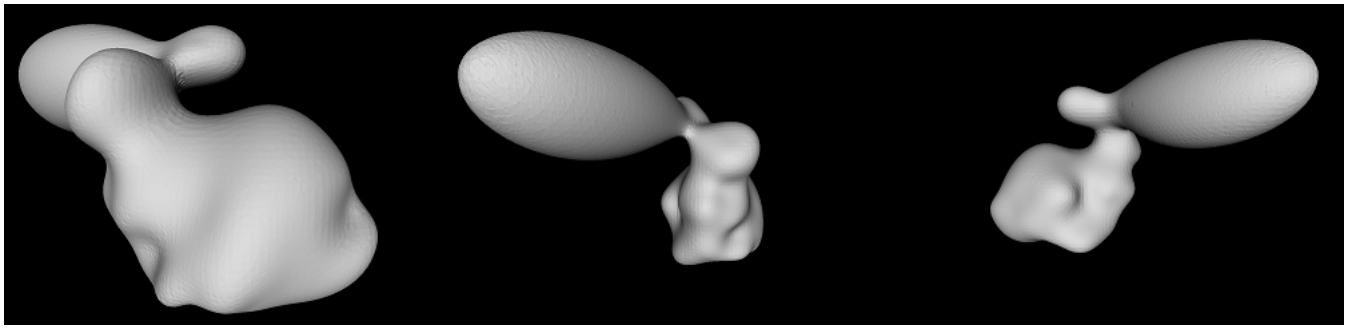


(c) Het center reductie algoritme toegepast met een precisie van 0,00014.

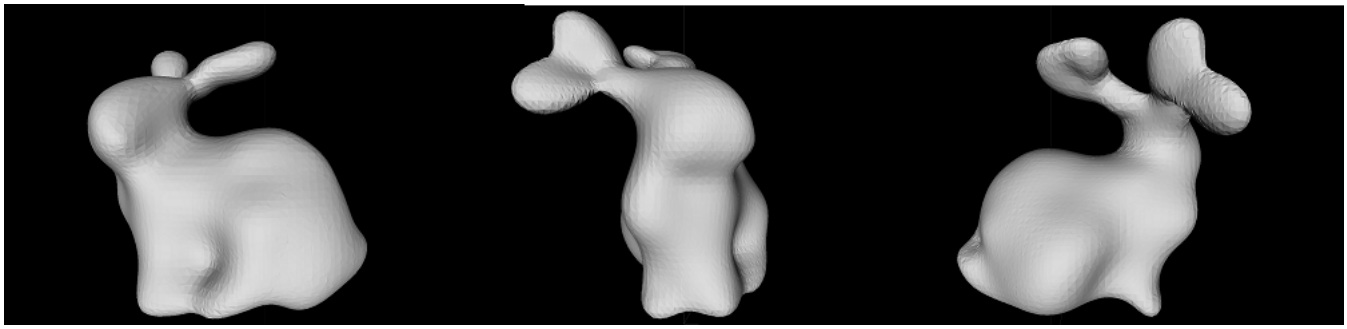


(d) In dit geval werden alle punten gebuikt.

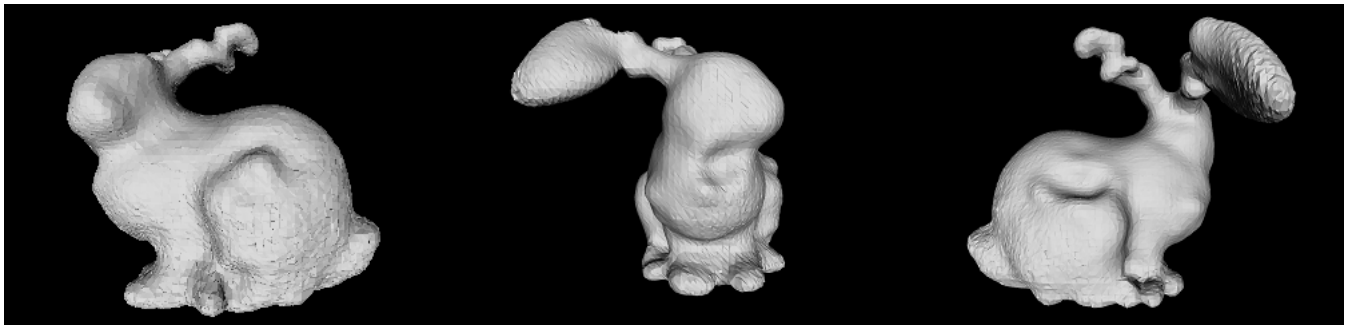
Figuur 4.5: Resultaat van de polygonisatie van het res4 model van de Stanford Bunny. De reconstructie werd gedaan met behulp van een biharmonische functie. Bij het center reductie algoritme werd telkens meer dan één punt per iteratie toegevoegd.



(a) Het center reductie algoritme toegepast met een precisie van 0,014.



(b) Het center reductie algoritme toegepast met een precisie van 0,0014.



(c) In dit geval werden alle punten gebuikt.

Figuur 4.6: Resultaat van de polygonisatie van het res4 model van de Stanford Bunny. De reconstructie werd gedaan met behulp van een triharmonische functie. Bij het center reductie algoritme werd telkens meer dan één punt per iteratie toegevoegd.

res4				
precisie	tree	center reduction	construction	polygonisatie
0.014	90 ms.	1 min. 30 sec.	14 sec. 710 ms.	1 min. 31 sec.
0.0014	80 ms.	19 min. 57 sec.	5 min. 6 sec.	4 min. 50 sec.
0.00014	100 ms.	31 min. 9 sec.	55 min. 32 sec.	5 min. 21 sec.

res4		
precisie	aantal driehoeken	aantal oppervlakte punten
0.014	106044	79
0.0014	123772	260
0.00014	122580	308

Tabel 4.5: Tabel met resultaten van de stanford Bunny op resolutie Res4. De reconstructie gebeurde met behulp van een biharmonische functie. Bij het center reductie algoritme werd telkens één punt per iteratie toegevoegd.

res4				
precisie	tree	center reduction	construction	polygonisatie
0.014	80 ms.	1 min. 32 sec.	13 sec. 730 ms.	53 sec 750 ms.
0.0014	80 ms.	2 min. 2 sec.	25 sec. 80 ms.	1 sec. 57 ms.
0.00014	90 ms.	6 min. 52 sec.	2 min. 5 sec. 140 ms.	560 ms.

res4		
precisie	aantal driehoeken	aantal oppervlakte punten
0.014	62068	80
0.0014	768	102
0.00014	208	170

Tabel 4.6: Tabel met resultaten van de stanford Bunny op resolutie Res4. De reconstructie gebeurde met behulp van een triharmonische functie. Bij het center reductie algoritme werd telkens één punt per iteratie toegevoegd.

weer. In de tweede wordt de tijd vermeld die nodig was om de boom te construeren, in de derde deze om het aantal centers te verminderen, in de vierde deze om de impliciete functie te construeren en in de vijde deze om de impliciete functie te polygoniseren. In de zesde kolom vinden we het aantal driehoeken terug die als resultaat van de polygonisatie en tenslotte in de zevende kolom het aantal surface punten dat gebruikt werd om de impliciete functie te construeren.
Uit de resultaten (zie figuur 4.7) blijkt dat het beter is om per iteratie meerdere punten toe te voegen, dit is echter niet sneller dan wanneer één punt per iteratie toegevoegd wordt. Dit is te verklaren omdat we te maken hebben met een globale functie. Als er op één bepaalde plaats een punt wordt toegevoegd heeft dit invloed op de hele functie. Als er dus op meerdere plaatsen waar de functie een grote errorwaarde heeft punten toegevoegd worden, wordt de globale functie op meerdere plaatsen tegelijk verbeterd. Als daarentegen slechts een punt per keer wordt toegevoegd heeft dit invloed op de errorwaarde op andere plaatsen. Deze kan dan verbeteren tot net onder de maximum errortolerantie en hierdoor is het eindresultaat niet zo goed is als bij het toevoegen van meerdere punten per keer.
Wat betreft de preciese is 0.00014 voldoende om een gedetailleerd resultaat te bekomen [6]. Bij de resultaten 4.7(c) is te zien dat deze ongeveer overeenkomen met de resultaten waarbij alle punten gebruikt werden 4.5(d). Het is helaas niet gelukt om de FFM methode te implementeren waardoor het ook niet mogelijk is om modellen te gebruiken met meer inputpunten [6].

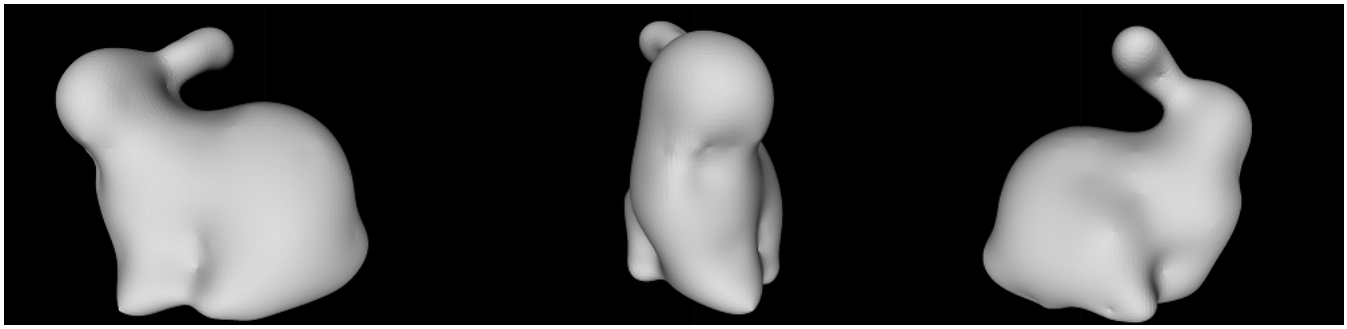
4.1.2 Methode van Wu et al.

Bij de deze implementatie wordt de methode van Wu et al. [58] gebruikt. Als acceleratiestructuur wordt hier een KD-boom gebruikt (zie hoofdstuk 2.6). Het aantal punten per blad en de overlapping van elke box worden op voorhand opgegeven. Volgens de paper van Wu [58] geven 60 punten en 3% overlapping goede resultaten. In de eerste stap zitten alle punten in één node. Daarna wordt dit recursief verder opgesplitst (zie hoofdstuk 2.5.2).
Voor elk blad wordt er een impliciete functie geconstrueerd. Het voordeel van deze methode is dat voor de evaluatie dan enkel de bladeren van de boom afgelopen moeten worden [58]. Om de functies samen te voegen worden blending functies $w_i(x)$ gebruikt (zie hoofdstuk 2.5.2). Voor de visualisatie tenslotte wordt ook de polygonisatie methode van Bloomenthal [4] gebruikt.

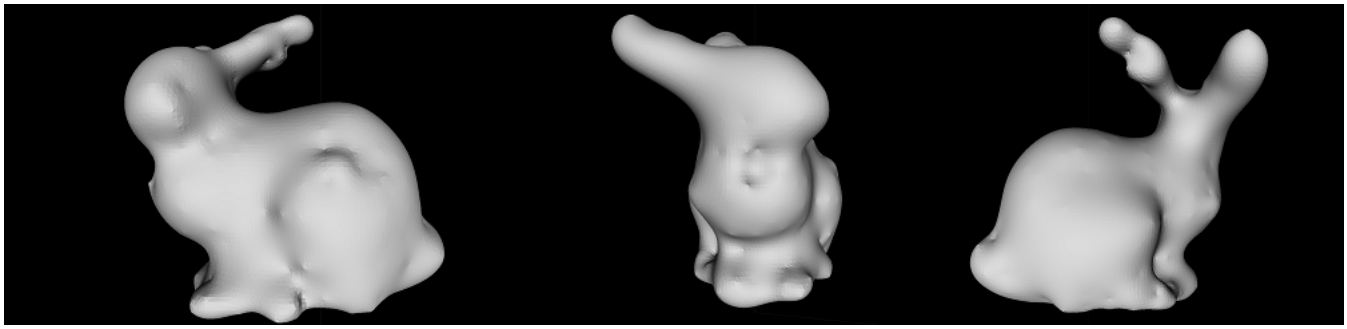
De resultaten in de tabellen bestaan uit 6 kolommen. De eerste kolom geeft het aantal punten per bladknoop en het overlappingspercentage weer. In de tweede wordt de tijd vermeld die nodig was om de boom te construeren, in de derde deze om de radial basisfuncties in de bladkonpen te construeren en in de vierde deze om de impliciete functie te polygoniseren. In de vijfde kolom vinden we het aantal driehoeken terug die als resultaat van de polygonisatie en tenslotte in de zesde kolom het aantal bladknopen in de boom.
De resultaten van deze methode zijn te zien in: tabel(4.7), tabel(4.8), tabel(4.9).

De resultaten van de visualisatie van de bunny zijn te zien in: figuur(4.8), figuur(4.9), figuur(4.10) en figuur(4.11).

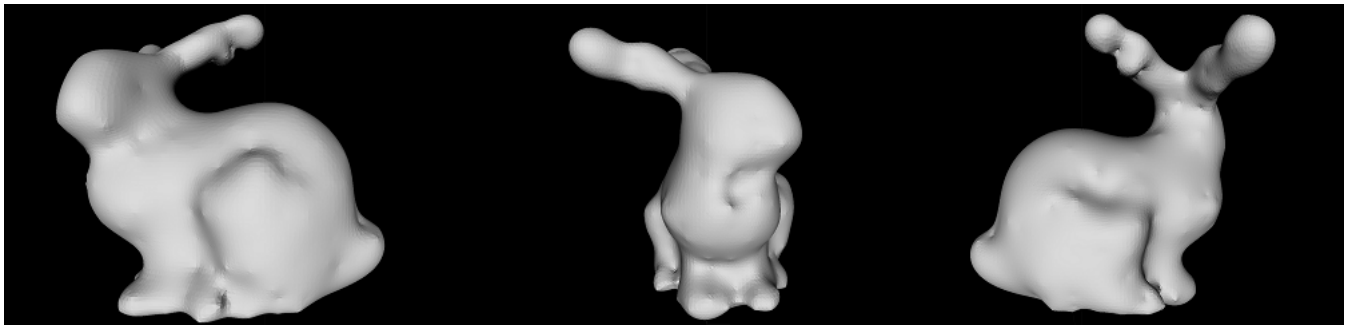
De resultaten van het buddha model zijn te zien in tabel 4.10, 4.11 en 4.12. Resultaten



(a) Het center reductie algoritme toegepast met een precisie van 0,014.



(b) Het center reductie algoritme toegepast met een precisie van 0,0014.



(c) Het center reductie algoritme toegepast met een precisie van 0,00014.

Figuur 4.7: Resultaat van de polygonisatie van het res4 model van de Stanford Bunny. De reconstructie werd gedaan met behulp van een biharmonische functie. Bij het center reductie algoritme werd telkens één punt per iteratie toegevoegd.

res4					
punten/perc.	tree	construction	polygonisatie	# driehoeken	# leafs
22 4%	90 ms.	40 ms.	1 min. 30 sec 400 ms.	147384	32
39 4%	80 ms.	70 ms.	1 min 23 sec 430 ms.	113816	16

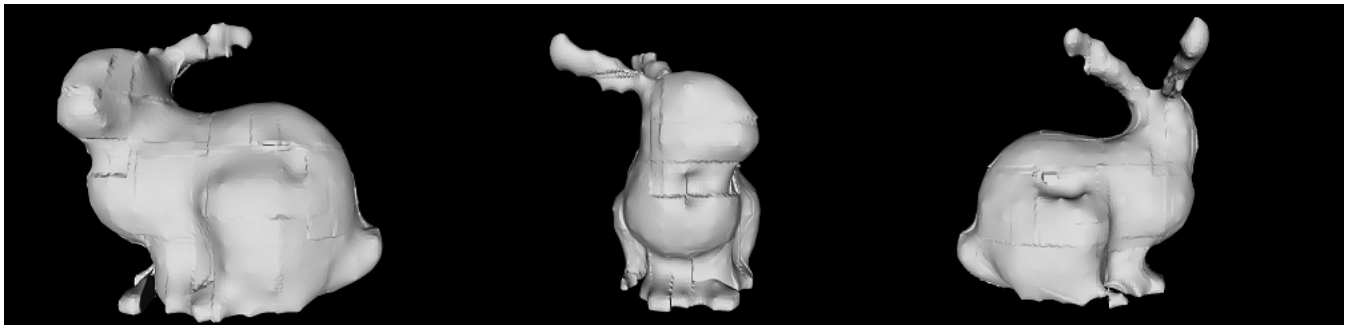
Tabel 4.7: Tabel met resultaten van de Stanford bunny voor de methode van Wu op resolutie res4.

res3					
punten/perc.	tree	construction	polygonisatie	# driehoeken	# leafs
15 4%	340 ms.	140 ms.	2 min. 46 sec 330 ms.	217256	256
17 12%	680 ms.	750 ms.	12 min. 2 sec 350 ms.	122516	1024
48 4%	300 ms.	470 ms.	2 min. 35 sec. 540 ms.	178260	64
67 12%	440 ms.	2 sec. 280 ms.	7 min 10 sec 960 ms.	121056	128
55 20%	1 sec. 490 ms.	10 sec. 830 ms.	1 uur 15 min	121056	1024

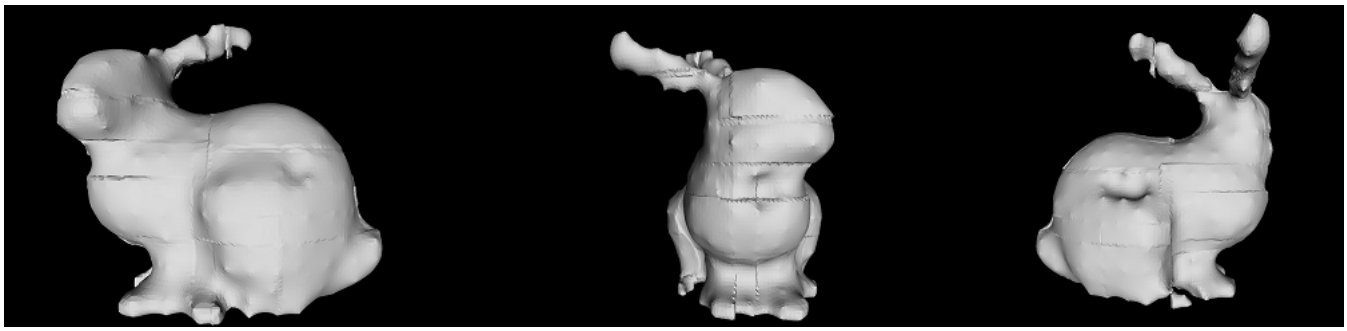
Tabel 4.8: Tabel met resultaten van de Stanford bunny voor de methode van Wu op resolutie res3.

res2					
punten/prec.	tree	construction	polygonisatie	# driehoeken	# leafs
18 12%	5 sec. 130 ms.	6 sec. 620 ms.	2 uur 20 min.	234760	8192
44 12%	3 sec. 460 ms.	12 sec. 290 ms.	55 min. 18 sec.	167360	2048
70 12%	3 sec. 160 ms.	20 sec. 670 ms.	20 min. 37 sec.	143736	1024
48 15%	5 sec. 910 ms.	30 sec. 560 ms.	1 uur 33 min.	130796	4096

Tabel 4.9: Tabel met resultaten van de Stanford bunny voor de methode van Wu op resolutie res2.

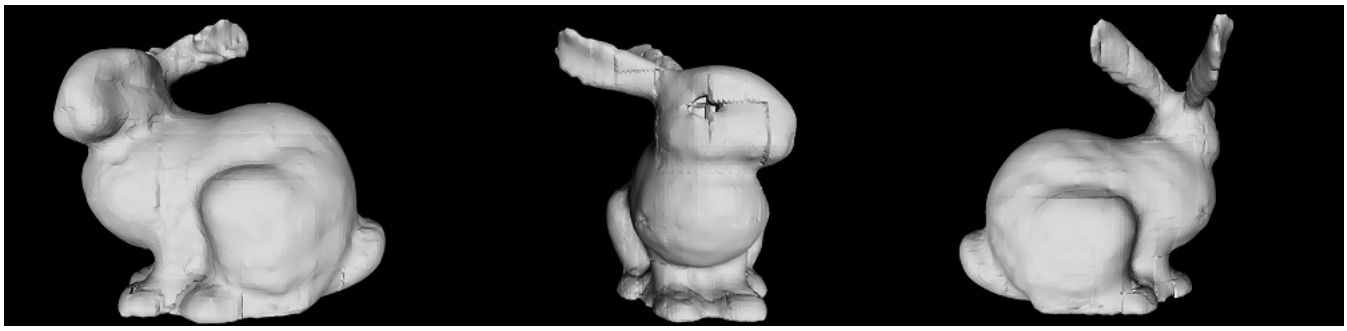


(a) Methode van Wu et al. met 20 surfacepunten per blad en 4% overlapping.

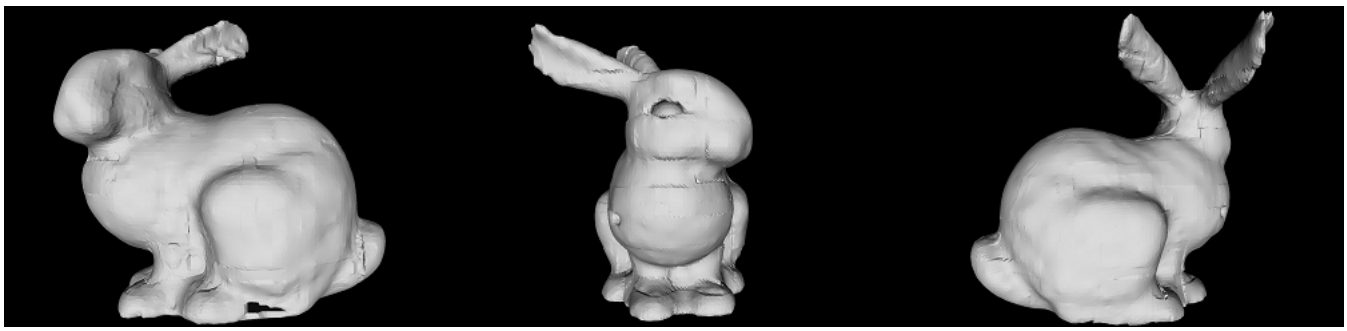


(b) Methode van Wu et al. met 37 surfacepunten per blad en 4% overlapping.

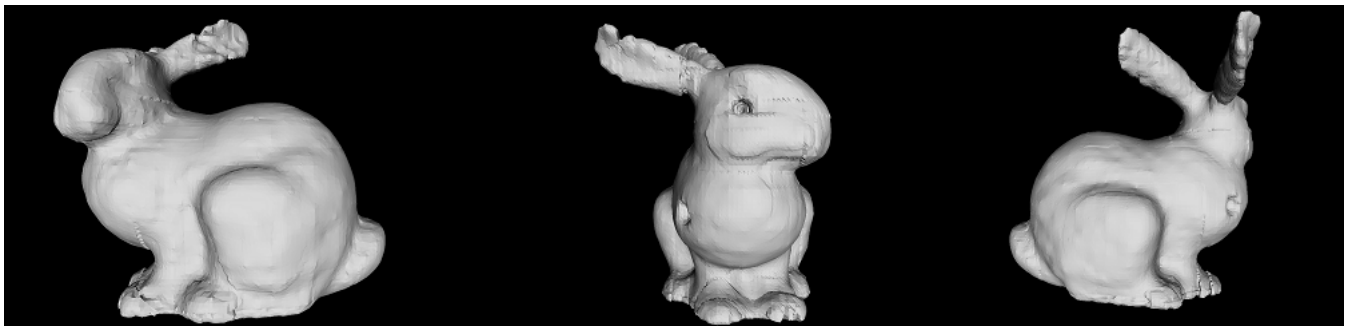
Figuur 4.8: Methode van Wu et al. met het bunny model op resolutie res4.



(a) Methode van Wu et al. met 65 surfacepunten per blad en 12% overlapping.

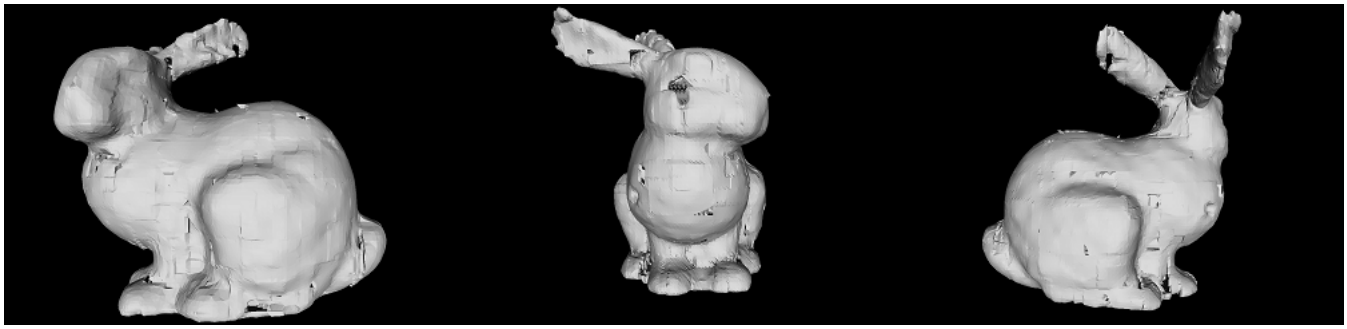


(b) Methode van Wu et al. met 15 surfacepunten per blad en 4% overlapping.

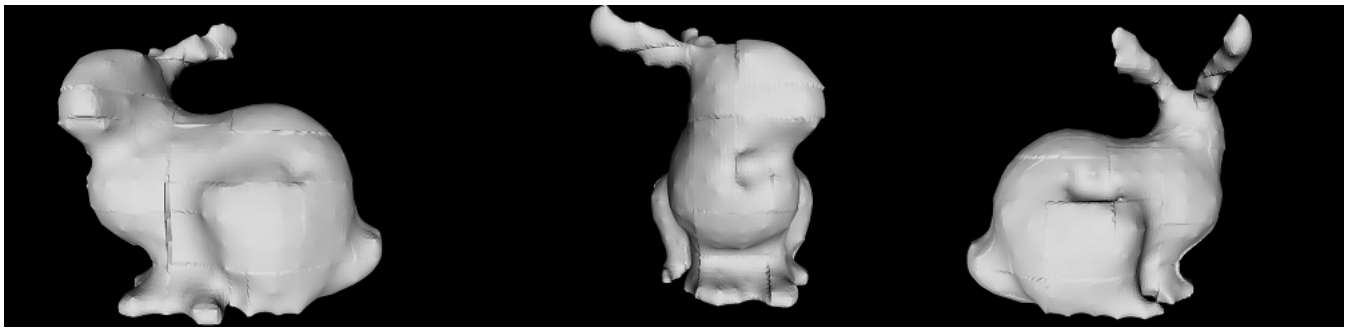


(c) Methode van Wu et al. met 53 surfacepunten per blad en 20% overlapping.

Figuur 4.9: Methode van Wu et al. met het bunny model op resolutie res3.

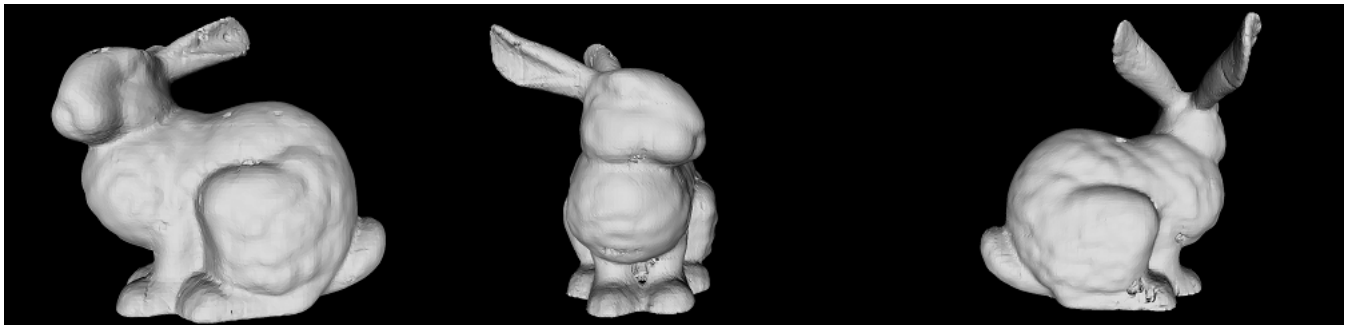


(a) Methode van Wu et al. met 13 surfacepunten per blad en 4% overlapping.

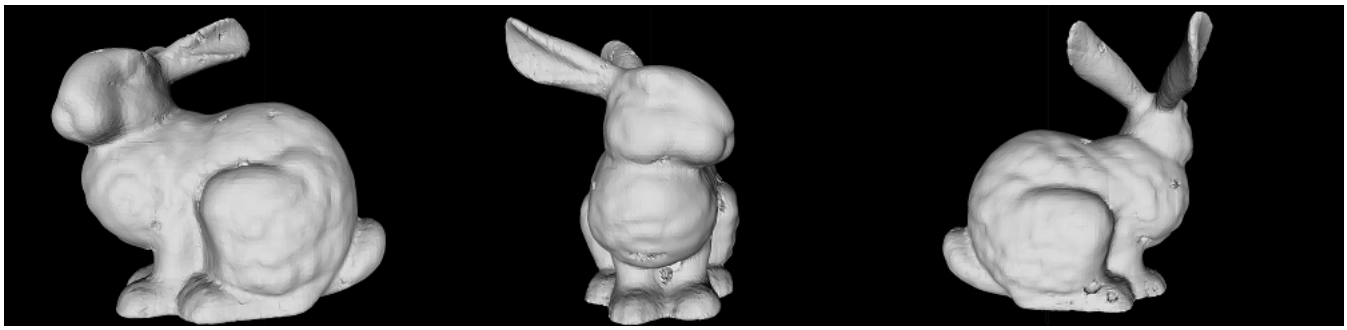


(b) Methode van Wu et al. met 46 surfacepunten per blad en 12% overlapping.

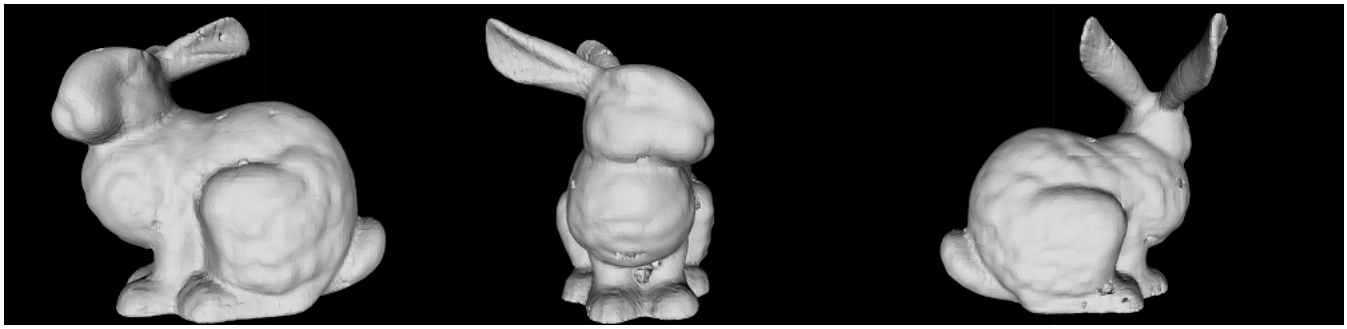
Figuur 4.10: Methode van Wu et al. met het bunny model op resolutie res3.



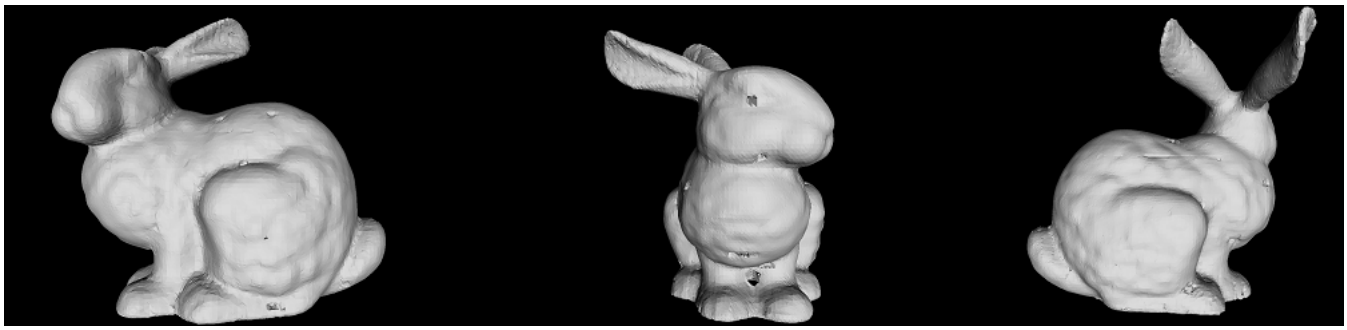
(a) Methode van Wu et al. met 46 surfacepunten per blad en 15% overlapping.



(b) Methode van Wu et al. met 68 surfacepunten per blad en 12% overlapping.



(c) Methode van Wu et al. met 42 surfacepunten per blad en 12% overlapping.



(d) Methode van Wu et al. met 16 surfacepunten per blad en 12% overlapping.

Figuur 4.11: Methode van Wu et al. met het bunny model op resolutie res2.

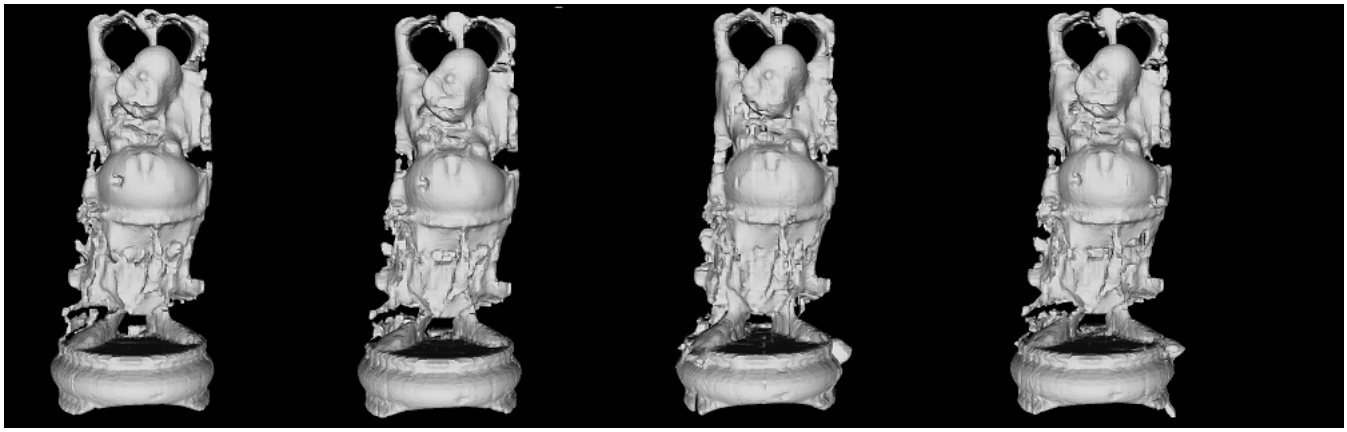
res4					
punten/perc.	tree	construction	polygonisatie	# driehoeken	# leafs
142 20%	12 sec. 920 ms.	6 min. 18 sec.	6 min. 35 sec. 110 ms.	2 uur 54 min.	2048
97 15%	6 sec. 700 ms.	56 sec. 790 ms.	1 min. 5 sec. 830 ms.	36 min.	1024
60 5%	1 sec. 510 ms.	3 sec. 380 ms.	3 min. 53 sec.	150280	256
64 15%	4 sec. 380 ms.	33 sec. 390 ms.	55 min. 36 sec.	123696	2048

Tabel 4.10: Tabel met resultaten voor de methode van Wu voor het buddha model op resolutie res4

res3					
punten/perc.	tree	construction	polygonisatie	# driehoeken	# leafs
58 3%	7 sec. 660 ms.	12 sec. 470 ms.	5 min. 46 sec.	196404	1024
68 7%	10 sec. 10 ms.	38 sec. 150 ms.	15 min. 54 sec.	190448	2048
46 5%	8 sec. 630 ms.	14 sec. 200 ms.	9 min. 41 sec.	193604	2048
66 12%	21 sec. 60 ms.	2 min. 20 sec.	3 uur 15 min. 18 sec.	173788	8192
121 15%	38 sec. 710 ms.	15 min. 7 sec.	5 uur 29 min. 46 sec.	156564	8192

Tabel 4.11: Tabel met resultaten voor de methode van Wu voor het buddha model op resolutie res3

van de visualisatie van de buddha zijn te zien in: figuur(4.12), figuur(4.13) en figuur(4.14).



(a) Methode van Wu et al. met van links naar rechts: 142 punten per blad en 20% overlapping, 97 punten per blad en 15% overlapping, 60 punten per blad en 5% overlapping en 64 punten per blad en 15% overlapping.

Figuur 4.12: Methode van Wu et al. met het buddha model van res4.

(a) Methode van Wu et al. met van links naar rechts: 58 punten per blad en 3% overlapping, 68 punten per blad en 7% overlapping, 46 punten per blad en 5% overlapping en 66 punten per blad en 12% overlapping.

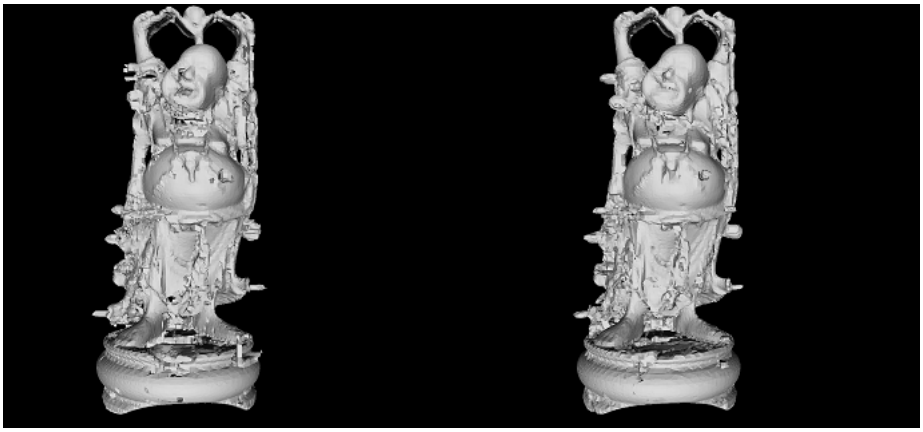
(b) Methode van Wu et al. met 121 punten per blad en 15% overlapping.

Figuur 4.13: Methode van Wu et al. met model res3.

res2					
punten/perc.	tree	construction	polygonisatie	# driehoeken	# leafs
115 10%	1 min. 48 sec.	23 min. 13 sec.	3 uur 28 min. 24 sec.	200020	16384
113 12%	2 min. 48 sec.	56 min. 31 sec.	4 uur 59 min. 32 sec.	200144	32768

Tabel 4.12: Tabel met resultaten voor de methode van Wu voor het buddha model op resolutie res2

Aan deze resultaten is te zien dat modellen met meer punten een smoother resultaat geven. Dit komt onder andere omdat de punt dichtheid groter is en er dus een gedetailleerdere functie gefit kan worden. Bij het gebruik van weinig punten, zitten er weinig punten in elke leafnode en in elk overlappend gedeelte. Als we het overlappend gedeelte heel erg vergroten komt dit eigenlijk op hetzelfde neer als alle punten gebruiken om één impliciete functie te construeren zoals in hoofdstuk 4.1.1 gedaan werd. Als er weinig punten voor handen zijn, kunnen er dus ook weinig punten in het overlappend gedeelte zitten en is het resultaat minder smooth en minder gedetaillerd.
Wanneer de puntdichtheid groter wordt, wordt ook het resultaat beter en gedetailleerder. De lokale delen kunnen door het hogere aantal punten veel beter geconstrueerd worden. En er zijn meer punten voor de overlappende delen beschikbaar. Een voorbeeld uit de paper van Wu [58] is het Stanford Bunny model met 28060 punten, 2048 bladeren en 50 punten per blad. Het totaal aantal surfacepunten is dan 102400, wat aanzienlijk meer is dan het aantal input surfacepunten. Als de overlapping groter wordt, wordt het totaal aantal groter. Als het aantal punten waaruit het model bestaat reeds erg groot is, zorgt een kleiner verhogen van het overlappingspercentage voor grote verhoging van het totaal aantal punten. Hierdoor wordt ook de berekeningstijd aanzienlijk groter. Indien het model uit veel punten bestaat is het niet nodig dat het overlappingspercentage groot is om een goed resultaat te bekomen.
Het gebruik van de methode van Wu werkt goed als de puntdichtheid in het model overal even groot is. Zijn er redelijke verschillen, dan vergt deze methode heel wat meer geheugen en rekentijd [58]. Dit komt omdat er hierdoor meer opsplitsingen per blad moeten gebeuren en hierdoor het totaal aantal gebruikte punten toeneemt. Deze opsplitsing bij plaatsen van grote dichtheid bevat eigenlijk overbodige punten. Deze liggen zo dicht bij elkaar dat ze bij het fitten van de functie niet zorgen dat de functie aanzienlijk beter wordt. Dit is te



(a) Methode van Wu et al. met van links naar rechts: 115 punten per blad en 10% overlapping en 113 punten per blad en 12% overlapping.

Figuur 4.14: Methode van Wu et al. met model res2.

res4				
precisie	tree	center reduction	construction	polygonisatie
all	80 ms.	n.a.v.	5 min 49 sec. 750 ms.	5 min 18 sec 870 ms.
0.014	90 ms.	14 sec. 850 ms.	2 sec. 400 ms.	2 sec 690 ms.
0.0014	90 ms.	1 min. 23 sec.	39 sec. 360 ms.	6 sec. 760 ms.
0.00014	80 ms.	4 min. 21 sec.	1 min. 36 sec. 700 ms.	1 min. 3 sec. 880 ms.

res4		
precisie	aantal driehoeken	aantal oppervlakte punten
all	117400	453
0.014	4132	84
0.0014	4516	202
0.00014	28648	340

Tabel 4.13: Tabel met resultaten van de stanford Bunny op resolutie res4. De reconstructie gebeurde met behulp van een triharmonische functie.

vergelijken met het center reductie algoritme uit de paper van Carr [6]. Een functie die door weinig punten gefit wordt kan hier toch nog een lage errorwaarde en goede preciese hebben zodat het visuele resultaat er tussen verwaarloosbaar is.
In onze implementatie construeren we voor elk surfacepunt één off-surfacepunt, in de paper van Wu wordt echter nog een andere methode uitgewerkt. Bij deze methode is er slechts één off-surface punt per blad nodig. In onze implementatie is dit echter niet volledig gelukt. Bij het oplossen van de matrix zijn alle lambdas gelijk aan nul. Hierdoor kunnen we de impliciete functie niet visualiseren. De reden waarom de lambdas allemaal nul zijn hebben we niet kunnen achterhalen, maar als de lambdas wel verschillend van nul zouden zijn is het visualiseren wel mogelijk.

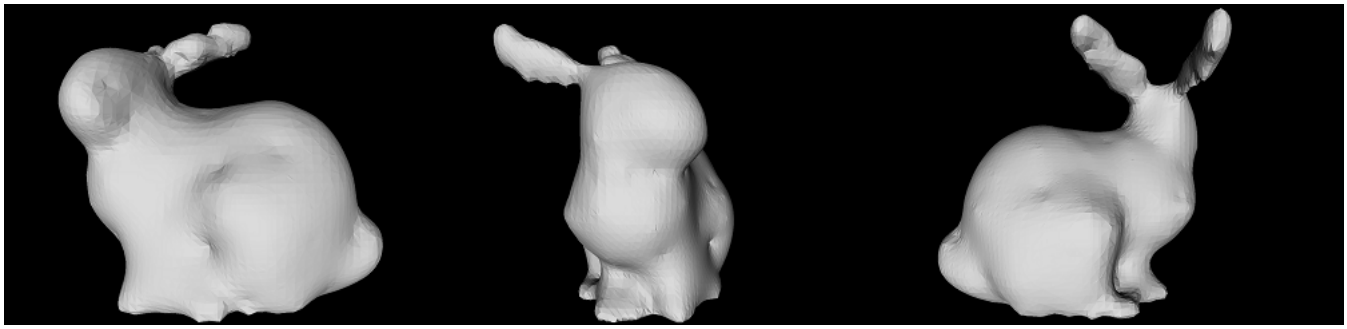
4.1.3 Derde manier

Deze manier is een variatie op de eerste manier. Bij deze methode worden off-surface punten die intersecteren verwijderd en dus niet alleen off-surface punten waarbij de normaal van het overeenkomstige surface punt dubbelzinnig is. Het is niet nodig dat elk surface punt een off-surface punt heeft [6]. Verder is het voldoende dat per node of cluster van punten één off-surface [58] geconstrueerd wordt. Het zou dus ook mogelijk moeten zijn om het aantal off-surface punten in geval van de methode van Carr [6] aanzienlijk te verminderen zonder kwaliteitsverlies.
Doordat er minder punten zijn neemt de berekeningstijd om de impliciete functie te construeren nog af (zie tabellen 4.13 en 4.14).

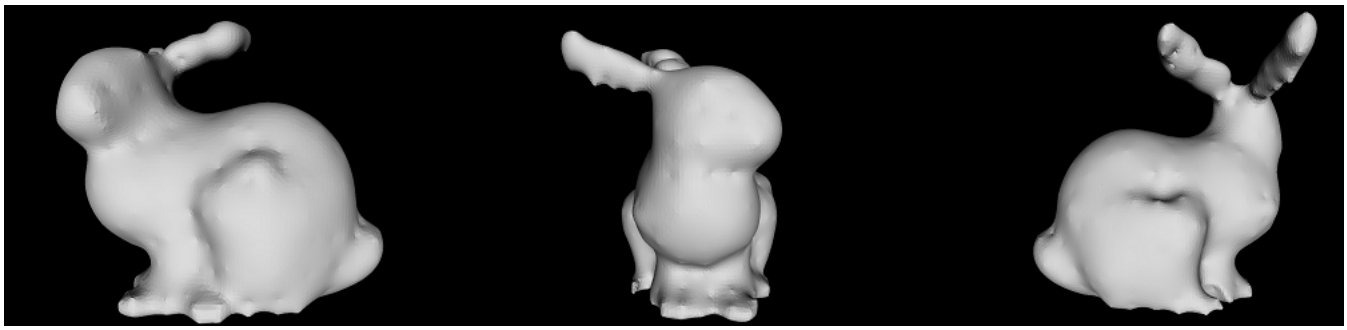
De resultaten van de biharmonische functies zijn ongeveer evengoed als deze van de eerste manier (vergelijk tabel 4.13 en figuur(4.15 met tabel 4.3 en figuur 4.5)).

(a) Derde globale methode toegepast met een precisie van 0,014.

(b) Derde globale methode toegepast met een precisie van 0,0014.



(c) Derde globale methode toegepast met een precisie van 0,00014.



(d) In dit geval worden alle punten gebruikt

Figuur 4.15: Afbeeldingen van de derde globale methode van de stanford Bunny op resolutie res4. De reconstructie gebeurde met behulp van een triharmonische functie.

res4				
precisie	tree	center reduction	construction	polygonisatie
all	80 ms.	n.a.v.	5 min. 47 sec. 20 ms.	5 min. 44 sec 840 ms.
0.014	70 ms.	14 sec. 780 ms.	1 sec. 650 ms.	2 sec 980 ms.
0.0014	80 ms.	20 sec. 140 ms.	9 sec. 790 ms.	1 sec. 57 ms.
0.00014	70 ms.	1 min. 36 sec. 440 ms.	32 sec. 820 ms.	39 sec. 480 ms.

res4		
precisie	aantal driehoeken	aantal oppervlakte punten
all	124284	453
0.014	5076	80
0.0014	6600	120
0.00014	28448	204

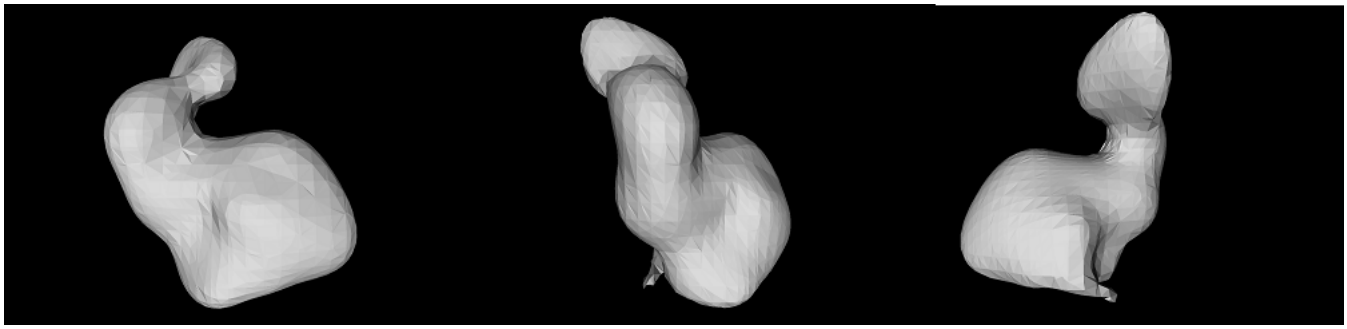
Tabel 4.14: Tabel met resultaten van de stanford Bunny op resolutie res4. De reconstructie gebeurde met behulp van een triharmonische functie.

De resultaten van de triharmonische functies zijn echter slechter, (vergelijk figuur(4.16).
Dit is te verklaren door het feit dat de triharmonische functie meer off-surface punten nodig heeft dan de biharmonische om goede resultaten te produceren.

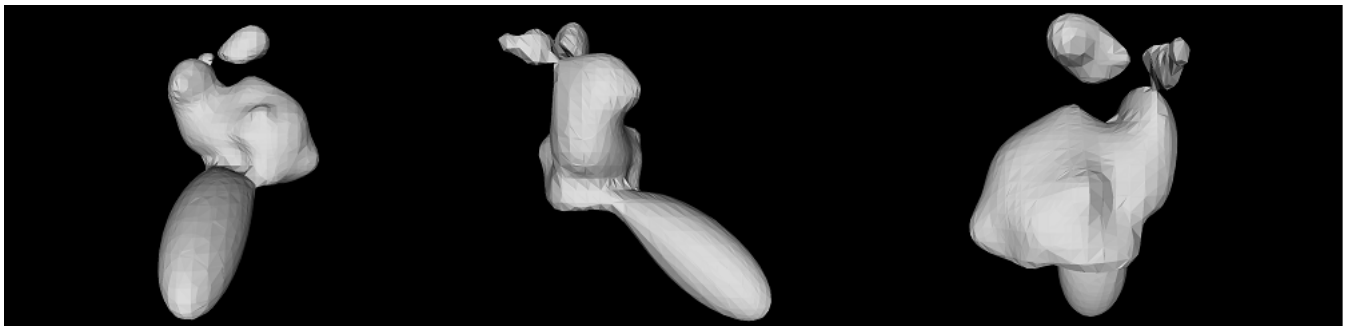
4.2 Hiërarchische methode

4.2.1 Methode van Ohtake et al.

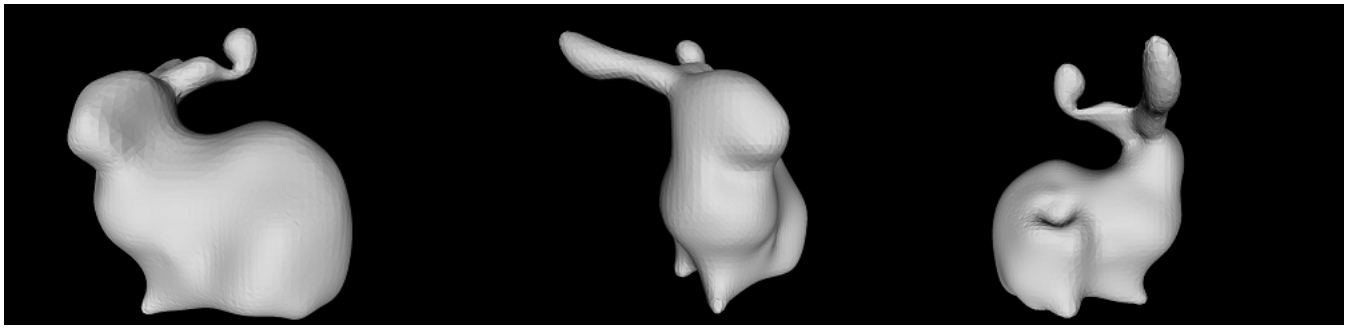
Net als bij de globale methode is ook hier de eerste stap het inlezen van een ply file.
De tweede stap is het opstellen van een octree om een single-level surface te construeren (zie hoofdstuk 2.6.2). Hiermee wordt het ideaal aantal levels $M = \lceil -log_2(\frac{\sigma_0}{2\sigma_1}) \rceil$, met $\sigma_1 = \frac{3}{4}$, berekent. σ_0 is de supportgrootte voor de single-level interpolatie. Deze wordt berekend door de som van de diagonalen van de bladeren te nemen en van dit resultaat $\frac{3}{4}$ te nemen. M is het niveau waarbij snelheid en accuraatheid in evenwicht zijn. Dit wil zeggen dat een hoger level meer tijd kost maar relatief weinig extra detail aan het resultaat toevoegd.
Nu kan de octree opgesteld worden die gebruikt wordt om de interpolatie functies te berekenen. Eerst worden de punten van $\mathcal{P}$ in een boundingbox geplaatst. Daarna wordt deze boundingbox in acht gelijke delen verdeeld, dit zijn de eerste 8 knopen van de octree. De punten verzameling $\mathcal{P}$ wordt dus geclusterd in verhouding tot de cellen van de gemaakte octree. Voor elke cel wordt ook een centrum berekend aan de hand van de punten van $\mathcal{P}$ die zich in de cel bevinden. De normaal voor dit centrum wordt verkregen door het gemiddelde te nemen van de normalen die toegekend zijn aan de punten van $\mathcal{P}$ die zich in de cel bevinden. Het gemiddelde van deze normalen wordt dan genormaliseerd. De initiële



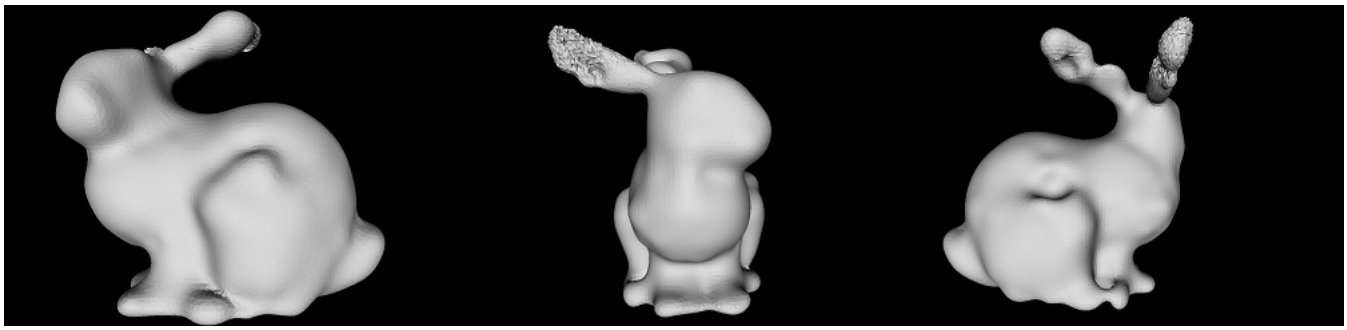
(a) Derde globale methode toegepast met een precisie van 0,014.



(b) Derde globale methode toegepast met een precisie van 0,0014.



(c) Derde globale methode toegepast met een precisie van 0,00014.



(d) In dit geval worden alle punten gebruikt

Figuur 4.16: Afbeeldingen van de derde globale methode van de stanford Bunny op resolutie res4. De reconstructie gebeurde met behulp van een triharmonische functie.

verzameling $\mathcal{P}^1$ komt overeen met het opdelen van het omvattende boundingbox van $\mathcal{P}$ in 8 gelijke delen. Om $\mathcal{P}^2$ te bekomen wordt de verzameling $\mathcal{P}$ geclusterd in verhouding tot de cellen van de gemaakte octree op het tweede level (64 delen). De berekening van de normalen voor elke cel gebeurt op dezelfde manier als in de vorige verzameling $\mathcal{P}^1$.
Deze octree heeft M niveaus. Hierbij wordt voor elk punt p_i op level M een functie $g_i(x)$ berekend met behulp van de punten rond punt p_i. Dit is een quadric die er als volgt uitziet: $g_i(x) = w - h(u, v)$, met $w = h(u, v) = Au^2 + 2Buv + Cv^2$. Deze functie wordt berekend in een lokaal assenstelsel (zie hoofdstuk 2.6). Als de punten binnen een straal σ omgezet zijn naar lokale coördinaten wordt volgende matrix ingevuld om de waarden voor A, B en C te bepalen.

$$\begin{pmatrix} \sum_{j=0}^{N} u_j^4\phi & \sum_{j=0}^{N} 2u_j^3v_j\phi & \sum_{j=0}^{N} u_j^2v_j^2 \\ \sum_{j=0}^{N} u_j^3v_j\phi & \sum_{j=0}^{N} 2u_j^2v_j^2\phi & \sum_{j=0}^{N} u_jv_j^3 \\ \sum_{j=0}^{N} u_j^2v_j^2\phi & \sum_{j=0}^{N} 2u_jv_j^3\phi & \sum_{j=0}^{N} v_j^4 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum_{j=0}^{N} u_j^2w_j\phi \\ \sum_{j=0}^{N} u_jv_jw_j\phi \\ \sum_{j=0}^{N} v_j^2w_j\phi \end{pmatrix} \tag{4.1}$$

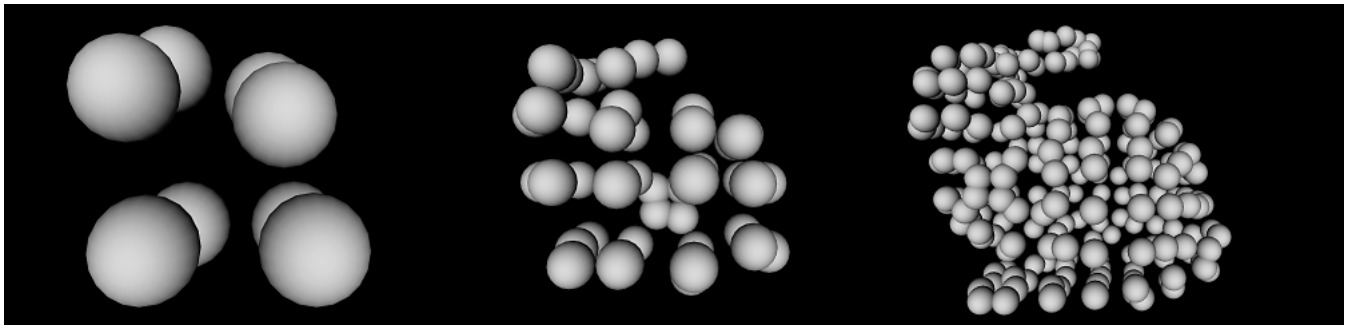
Het lokale coördinaten stelsel wordt als volgt bepaald: de lokale as w wordt gelijkgesteld aan de normaal n_i. Daarna worden willekeurige punten bepaald zodat deze loodrecht staan op w. Daarna wordt de Gram-Schmidt methode gebruikt om een ortonormaal assenstelsel te bekomen. De matrix (4.1) wordt opgelost met een eigen implementatie van de Gauss-Jordan eleminatie methode (zie hoofdstuk A.1).
Aangezien een groot deel van de elementen in de matrix nul zijn kunnen we geheugen besparen door enkel deze elementen op te slaan die een waarde verschillen van nul hebben. Het bespaarde geheugen hiervan is aanzienlijk, De matrix voor de bunny op level 4 zou normaal $931 \times 931 = 866761$ elementen bevatten. Er zijn echter slechts 28781 elementen verschillend van nul of 3.3%.
Als we alleen de elementen uit de matrix van boven of onder de diagonaal zouden bijhouden kunnen we het gebruikte geheugen van de matrix nog eens halveren. Dit is mogelijk omdat we te maken hebben met een symmetrische matrix. Bovendien hebben de elementen op de diagonaal allemaal dezelfde waarde. Een nadeel is echter dat bij een matrix vermenigvuldiging een deel van elke rij opnieuw geconstrueerd moet worden. Als we ervoor kiezen om de bovendiagonaal bij te houden moeten we voor elke rij i die we willen construeren alle rijen afgaan en voor elke rij het overeenkomstige element in kolom i zoeken. Dit kan het snelst gebeuren door voor elke rij binair zoeken toe te passen. Dit is de snelste methode omdat de elementen per rij opgeslagen worden als (index, reële waarde) en omdat de elementen die verschillend zijn van nul op willekeurige plaatsen in de matrix voorkomen. Aangezien de geheugen winst niet opweegt tegen de verloren tijd ($O(log(n))$), met n de lengte van de rij) voor het zoeken in een rij ten opzichte van $O(1)$ als het element in het geheugen staat hebben we ervoor gekozen om dit niet toe te passen.
Als laatste worden de λ_i berekend met behulp van de zelf geïmplementeerde preconditioned biconjugate gradient method (zie hoofstuk A.3). Deze methode lost een $N \times N$ matrix op, met N het aantal surface punten van het model. Dit leidt tot erg grote matrices en dus veel geheugen gebruik. Uit resultaten blijkt dat dit goed te doen is voor maximaal 25000

level	surface punten bunny	support σ bunny	surface punten buddha	support σ buddha
level 1	8	0.187685	8	0.171263
level 2	47	0.0938423	56	0.085809
level 3	220	0.0469211	291	0.0429045
level 4	931	0.0234606	1453	0.0214522
level 5	3685	0.0117303	6367	0.0107261
level 6	13156	0.00586514	25385	0.00536306
level 7	32208	0.00293257	78503	0.00268345
level 8	35948	0.00146628	133047	0.00134173

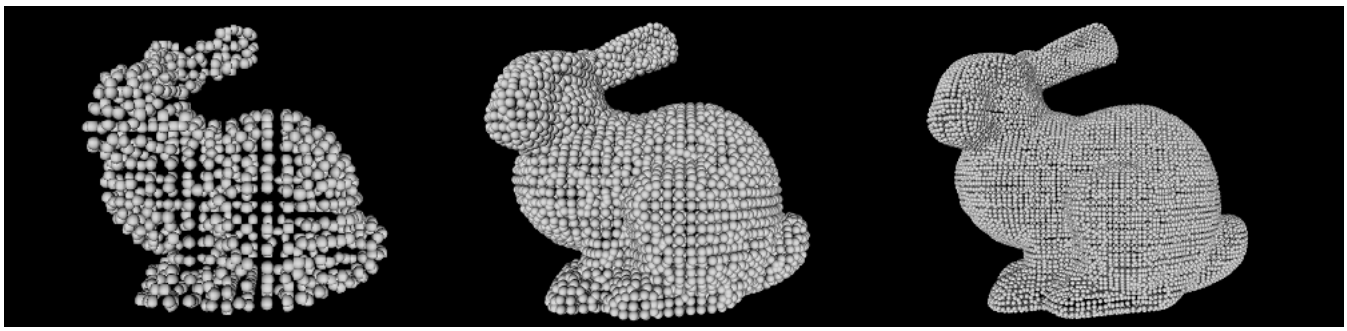
Tabel 4.15: Tabel met het aantal het aantal surface punten per level voor het bunny en buddha model.

surface punten. In dit geval is het geheugen gebruik ongeveer 1.5 Gb. Vandaar dat we dus enkel de waardes die verschillend zijn van nul in een matrix bijhouden. Omdat de matrix symmetrisch is, en de getallen op de diagonaal allemaal hetzelfde zijn moet er dus maar 48% van de matrix berekend worden. Als dit gedeelte bij elke iteratie opnieuw berekent wordt duurt de berekening ongeveer 2.5 keer langer dan als de matrix in het geheugen geladen wordt.
Tot slot is er de visualisatie. Ook hiervoor wordt net zoals in de vorige gevallen de Bloomenthal methode gebruikt [4].
Het aantal punten per level en de σ, zoals in deze in de paper berekend werd, is te zien in tabel 4.15. De grootte van de spheres per level is te zien in figuur 4.17 voor de Stanford bunny en in 4.18 voor de Buddha. De resoluties van de modellen die we gebruiken zijn res voor het bunny model en res2 voor het buddha model.
De resultaten in de tabellen bestaan uit 8 kolommen. De eerste kolom geeft het level dat geconstrueerd werd weer. In de tweede wordt de tijd vermeld die nodig was om de boom te construeren, in de derde deze om g(x) functies en de matrix te berekenen en in de vierde deze om de impliciete functie te polygoniseren. In de vijfde kolom wordt het maximale gebruikte ram, uitgedrukt in MB, bij deze berekeningen vermeld voor de polygonisatie. In de zesde kolom vinden we de gebruikte support en in de zevende kolom het aantal iteraties om de matrix op te lossen. In de achtste kolom tenslotte vinden we het aantal driehoeken terug die als resultaat van de polygonisatie verkregen werden.
De resultaten van de methode van de bunny zijn te zien in tabel 4.16 en figuren 4.19 en 4.20.
De resultaten van de methode van de buddha zijn te zien in tabel 4.17 en figuur 4.21.

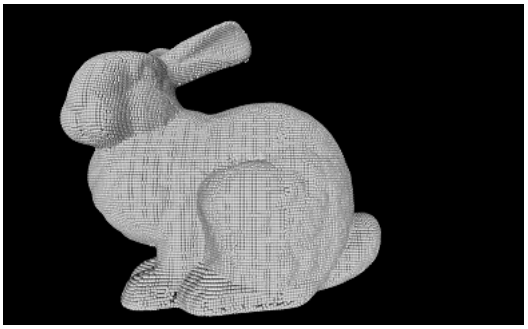
Aan de resultaten is duidelijk te zien dat naarmate de hiërarchie verder afgelopen wordt, en dus meer punten gebruikt worden, er meer detail toegevoegd wordt. Des te hoger de waarde σ des te beter het resultaat wordt, maar ook des te langer de methode duurt om een resultaat te bekomen. Om een goed resultaat te bekomen is het echter niet nodig om deze σ erg hoog te zetten. In de praktijk stellen we σ gelijk aan 3/4 van de lengte van



(a)

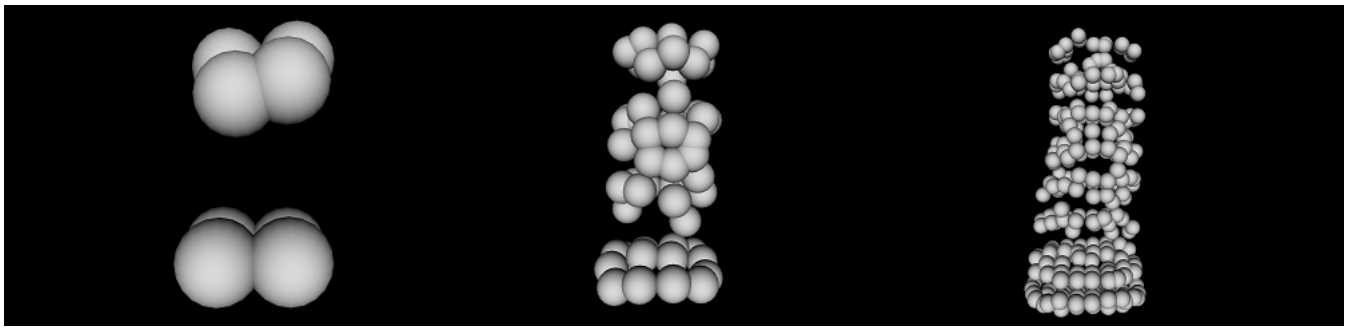


(b)

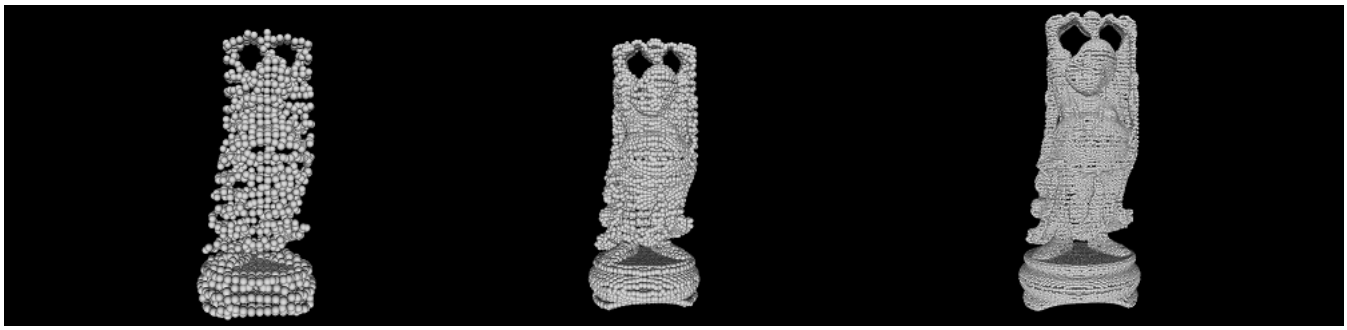


(c)

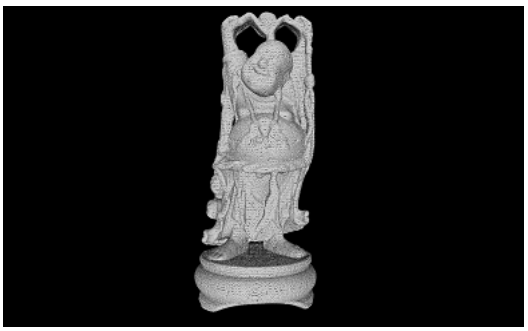
Figuur 4.17: Deze figuur geeft de bollen weer met straal σ voor elk level van de Stanford bunny. Voor deze visualisatie werd de straal, net als in de paper van Ohtake [26], een aantal keer kleiner dan deze gebruikt voor de visualisatie.



(a)



(b)



(c)

Figuur 4.18: Deze figuur geeft de bollen weer met straal σ voor de eerste 7 levels van de Stanford buddha. Voor deze visualisatie werd de straal, net als in de paper van Ohtake [26], een aantal keer kleiner dan deze gebruikt voor de visualisatie.

level	tree	g(x) en matrix	polygonisatie	ram (MB)
level 1	10 sec. 330 ms.	10 ms.	19 sec. 690 ms.	58.6
level 2	10 sec. 570 ms.	40 ms.	1 min. 20 sec.	58.7
level 3	10 sec. 600 ms.	260 ms.	2 min. 31 sec.	58.7
level 4	10 sec. 550 ms.	2 sec. 980 ms.	3 min. 53 sec.	59.0
level 5	10 sec. 560 ms.	23 sec. 870 ms.	7 min. 50 sec.	63.2
level 6	10 sec. 400 ms.	9 min. 41 sec. 340 ms.	16 min. 31 sec.	86.3
level 6	10 sec. 860 ms.	24 min. 4 sec. 650 ms.	37 min. 24 sec.	94.0
level 7	10 sec. 220 ms.	20 min. 29 sec. 260 ms.	35 min. 47 sec.	135.8

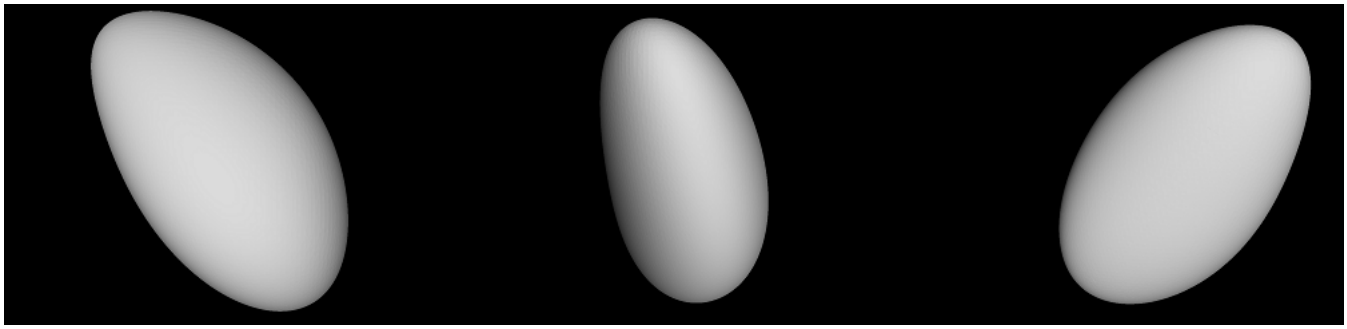
level	σ	# iteraties	# driehoeken
level 1	0.19	9	83976
level 2	0.1	23	99972
level 3	0.048	35	110860
level 4	0.024	54	119552
level 5	0.012	37	185448
level 6	0.008	83	204376
level 6	0.01	195	195880
level 7	0.006	14	219784

Tabel 4.16: Resultaten bunny hiërarchisch

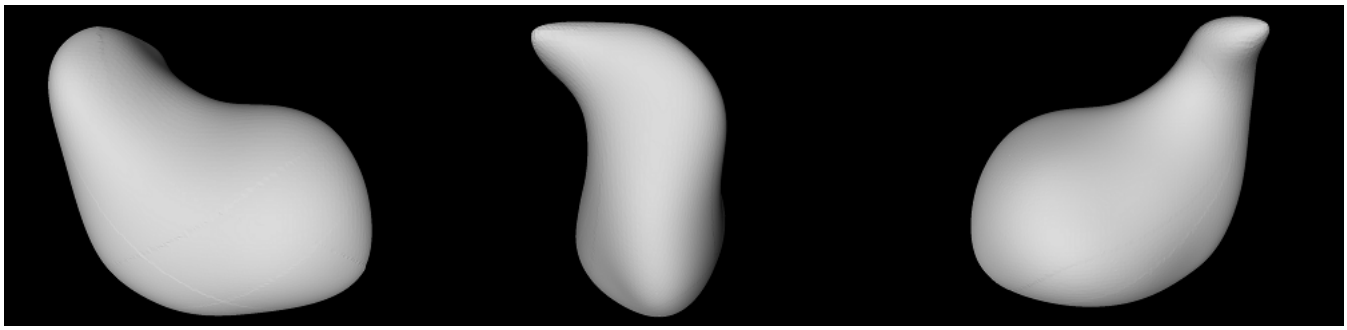
level	tree	g(x) en matrix	polygonisatie	ram (MB)
level 1	1 min. 7 sec. 770 ms.	10 ms.	13 sec. 180 ms.	282.0
level 2	1 min. 7 sec. 830 ms.	70 ms.	57 sec. 800 ms.	282.1
level 3	1 min. 7 sec. 470 ms.	750 ms.	2 min. 35 sec.	282.0
level 4	1 min. 7 sec. 170 ms.	12 sec. 120 ms.	5 min. 4 sec.	282.8
level 5	1 min. 7 sec. 890 ms.	2 min. 37 sec.	3 min. 54 sec.	292.4
level 6	1 min. 8 sec. 570 ms.	45 min. 18 sec.	2 uur 33 min. 54 sec.	331.5
level 7	1 min. 6 sec. 780 ms.	58 min. 43 sec.	51 min. 46 sec.	420.4

level	σ	# iteraties	# driehoeken
level 1	0.18	10	57200
level 2	0.09	46	67756
level 3	0.044	62	80968
level 4	0.022	96	96104
level 5	0.011	135	108372
level 6	0.0055	98	155640
level 7	0.0028	11	199992

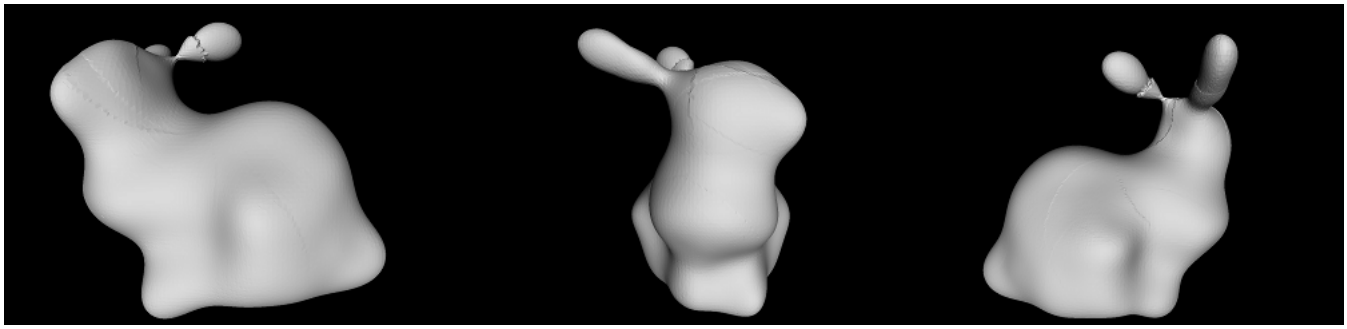
Tabel 4.17: Resultaten buddha hiërarchisch



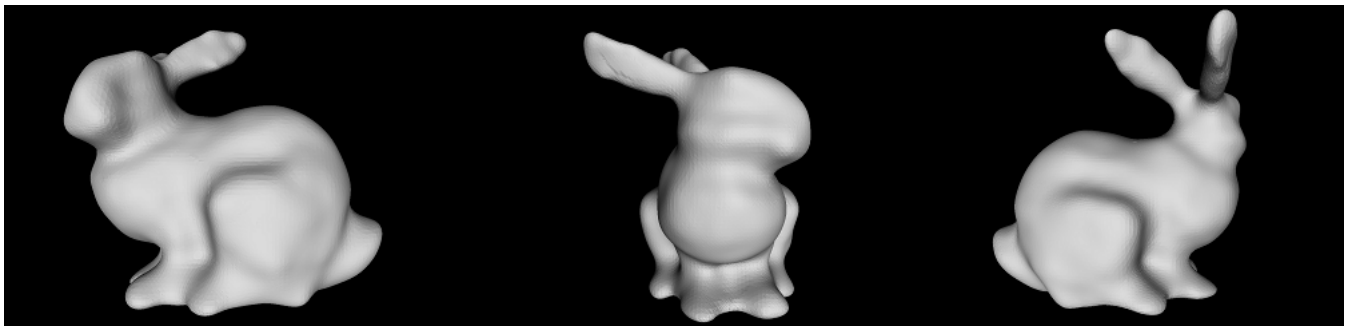
(a) Level 1 van de Ohtake methode



(b) Level 2 van de Ohtake methode

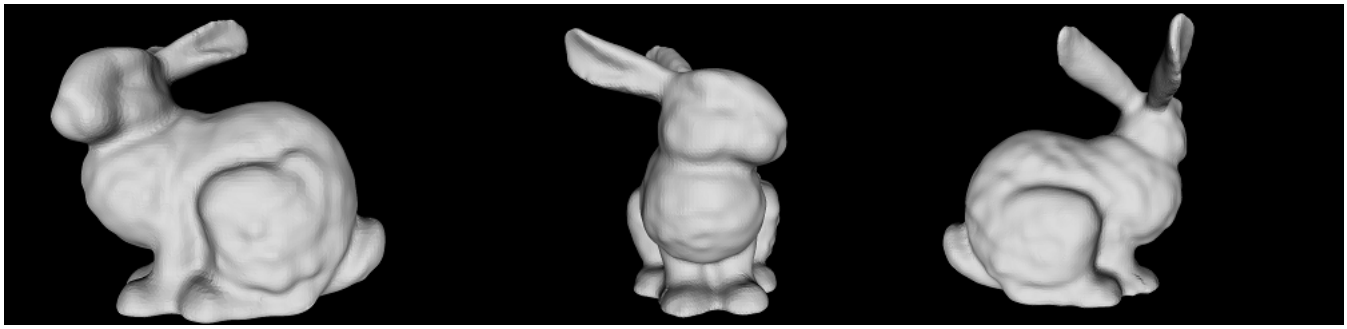


(c) Level 3 van de Ohtake methode

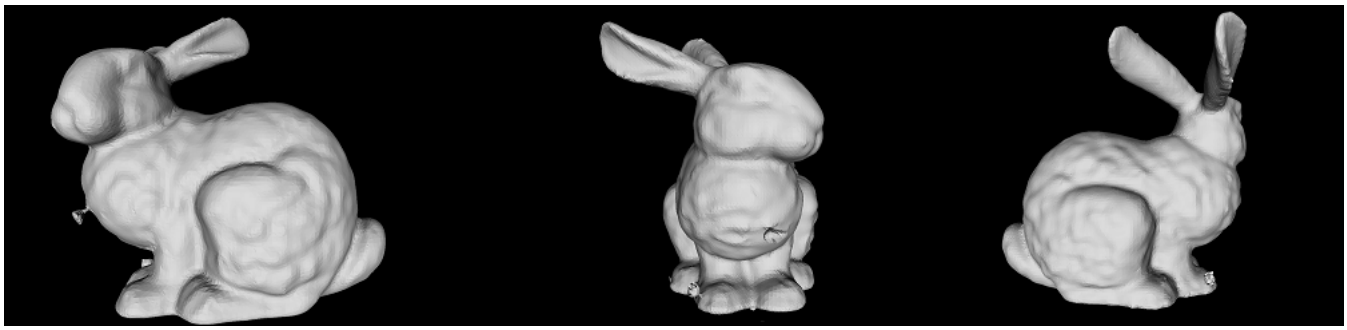


(d) Level 4 van de Ohtake methode

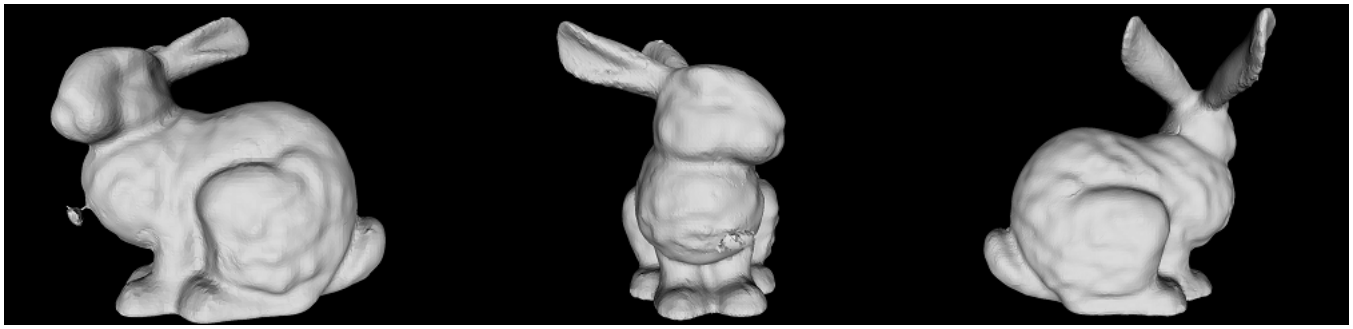
Figuur 4.19: Stanford Bunny van level 1 tot en met 4 van de Ohtake methode in drie aanzichten



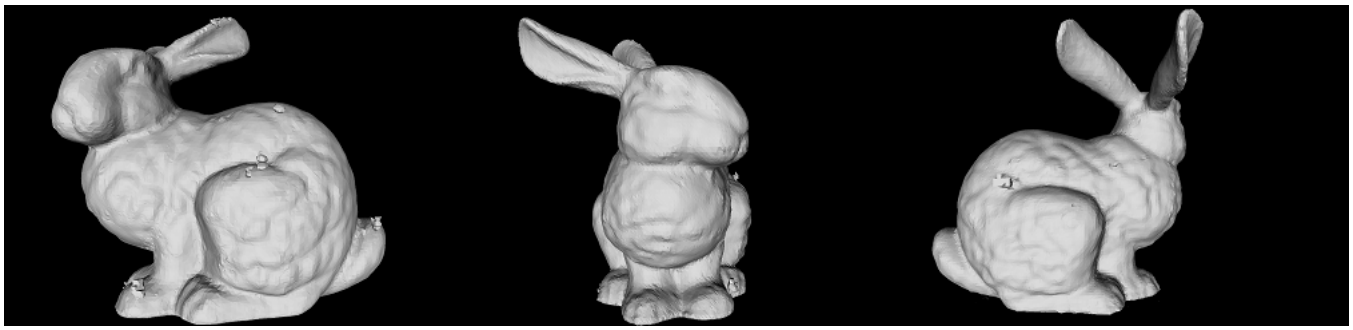
(a) Level 5 van de Ohtake methode



(b) Level 6 van de Ohtake methode, met $\sigma = 0.008$

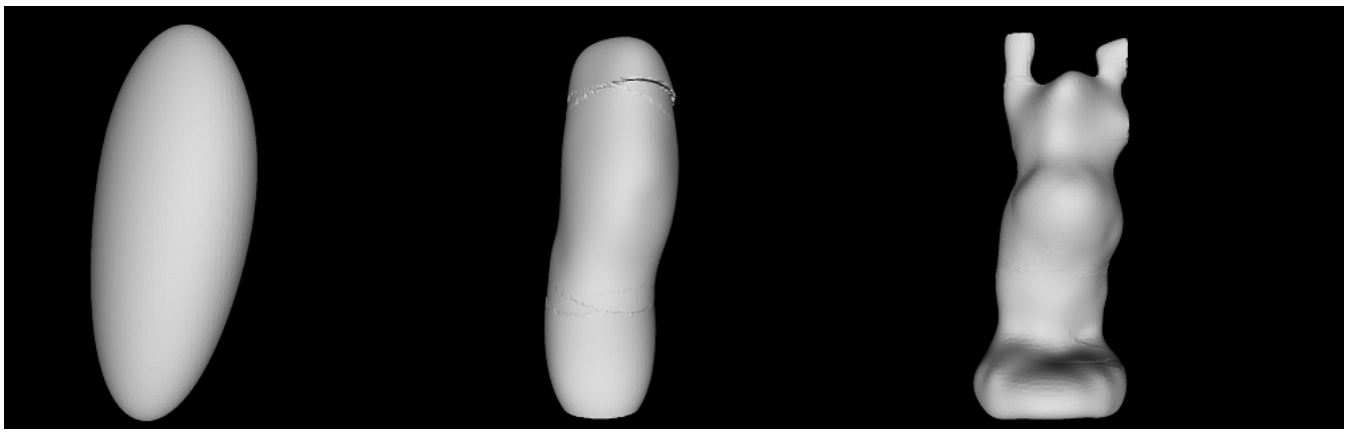


(c) Level 6 van de Ohtake methode, met $\sigma = 0.01$

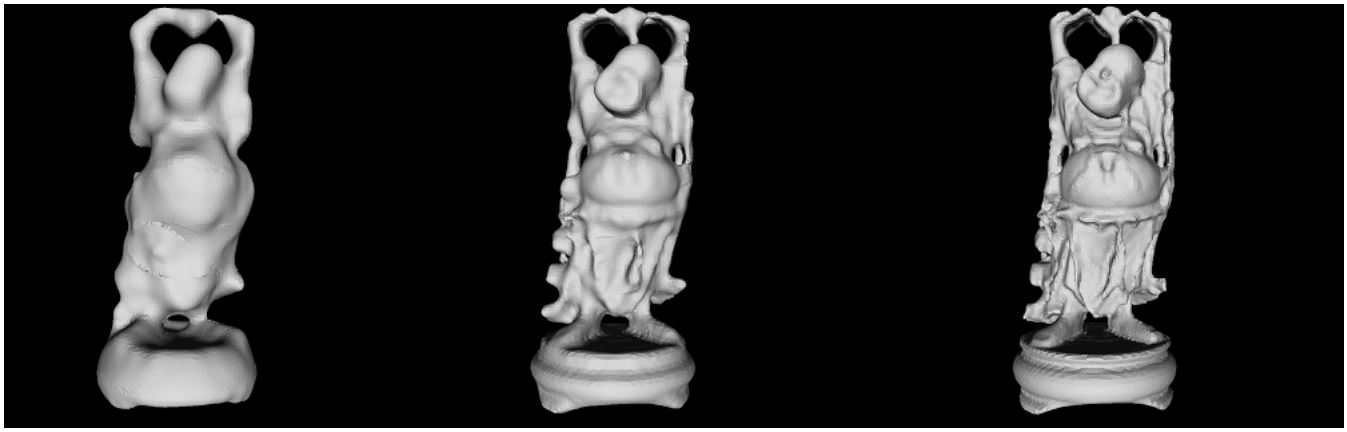


(d) Level 7 van de Ohtake methode

Figuur 4.20: Stanford Bunny van level 5 tot en met 7 van de Ohtake methode in drie aanzichten



(a) Van links naar rechts: level 1, 2 en 3



(b) Van links naar rechts: level 4, 5 en 6

(c) Level 7

Figuur 4.21: Resultaten van de hiërarchische methode van Ohtake van het buddha model

de diagonaal van de boundingbox op het level dat geconstrueerd wordt. In de resultaten tabel van de Stanford bunny (zie tabel 4.16) is te zien dat level 7 heel wat minder matrix iteraties nodig heeft om tot een oplossing te komen dan level 5 en 6. Dit is te verklaren door het feit dat de convergentie test minder strikt genomen werd. Er werd aangenomen dat de oplossing convergent is als 6 opeenvolgende iteraties een kleinere error geven in plaats van 15 in de andere gevallen. Verder werd er niet gecontroleerd op de maximum error, terwijl dit bij de andere levels telkens onder de 0.001 moest zijn. We kunnen hieruit concluderen dat we, om een goed resultaat te bekomen per level moeten testen wat het minimum aantal iteraties is om een goed resultaat te bekomen. Verder kunnen we zeggen dat de controle op de error niet noodzakelijk is om een goed resultaat te bekomen.

4.2.2 Tweede methode

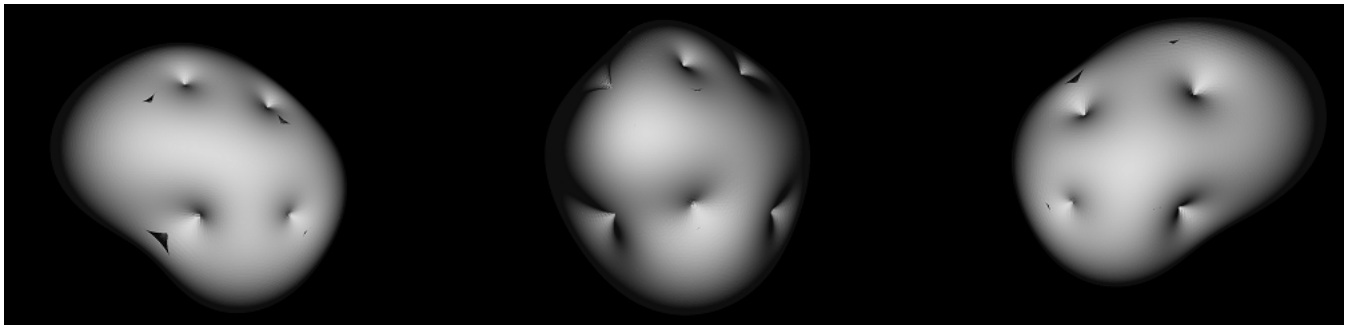
Hier zullen we proberen om de hiërarchy op een andere manier te berekenen. We beginnen ook bij deze methode met de constructie van een octree. De constructie hiervan gebeurt op dezelfde manier als bij de multi-scale hiërarchie van Ohtake [26], (zie hoofdstuk 2.6). We gebruiken nu echter rbf-functies om het impliciete oppervlak te benaderen. Dit gebeurt net als in de methode van Wu [58] met de constructie van één off-surface punt per surface punt.

Het resultaat is te zien in tabel (4.18) en figuren (4.22, (4.23 en (4.24))). Zoals bij de hiërarchische methode van Ohtake (zie hoofdstuk 4.2) vermeld werd is level 6 het ideale reconstructie level voor de Stanford bunny. Level 7 en 8 werden om deze reden dan ook niet meer geconstrueerd. Voor de eerste twee levels werden de lambdas omwille van het klein aantal punten enkel berekend met behulp van een enkele matrix. Voor level 2 en 3 werden de lambdas berekend met behulp van een enkele matrix en ook door per blad een biharmonische functie te construeren. Voor level 5 en 6 was het niet meer mogelijk om een matrix voor alle punten te gebruiken omwille van het grote aantal surface punten. Daarom werd hier gekozen om voor elk blad een biharmonische functie te construeren. De punten die gebruikt worden voor de functie zijn deze binnen een straal σ vanaf de oorsprong van het blad. Het resultaat wordt dan bekomen door al deze functies te blenden. Deze blend-functies zijn dezelfde als diegene die in de methode van Wu (zie hoofdstuk 4.1.2) gebruikt werden. Aan het resultaat is te zien dat deze echter niet geschikt zijn om in een octree te gebruiken. Een mogelijke verklaring hiervoor is dat er teveel gedeelten elkaar overlappen en de polygonisatie methode van Bloomenthal dan niet meer kan bepalen welke de juiste functiewaarde in een punt is.
Tenslotte kunnen we aan de resultaten zien dat het gebruik van een grotere waarde σ niet het gewenste effect heeft. Door meer punten te gebruiken om een functie te construeren zouden we intuïtief verwachten dat het resultaat beter zou worden. We kunnen in de resultaten echter zien dat dit afhankelijk is van het gebied. Zo is in level 6 het eerste zijaanzicht beter bij een grotere straal, maar zijn de oren dan weer slechter. Zoals eerder vermeld denken we dat de visualisatie methode een mogelijke verklaring is hiervoor.

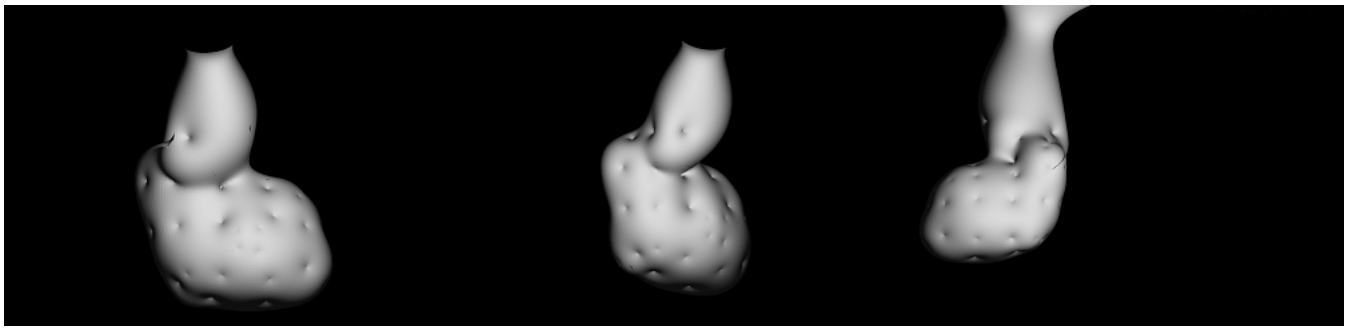
level	tree	construction	polygonisatie	
level 1	1.0	160 ms.	200 ms.	31 sec. 880 ms.
level 2	1.0	150 ms.	8 sec. 280 ms.	4 min. 13 sec. 160 ms.
level 3	0.05	15 sec. 720 ms.	21 sec. 130 ms.	1 min. 16 sec. 560 ms.
level 3	1.0	650 ms.	1 min. 18 sec. 550 ms.	3 min. 25 sec. 200 ms.
level 4	0.03	15 sec. 250 ms.	3 min. 38 sec, 250 ms	2 min. 17 sec. 600 ms.
level 4	1.0	2 sec. 950 ms.	1 uur 34 min 32 sec.	17 min. 5 sec.
level 5	0.01	30 sec. 230 ms.	28 sec. 110 ms.	1 min. 53 sec. 570 ms.
level 5	0.02	15 sec. 410 ms.	47 min. 2 sec.	3 min. 12 sec. 810 ms.
level 6	0.01	23 sec. 400 ms.	1 uur 32 min.	3 min. 10 sec. 500 ms.
level 6	0.012	15 sec. 480 ms.	3 uur 26 min.	3 min. 51 sec. 760 ms.

level	aantal driehoeken	aantal oppervlakte punten
level 1	153992	8
level 2	142938	47
level 3	131804	220
level 3	131692	220
level 4	161796	n.a.v. (931 verschillende)
level 4	201836	n.a.v. (931 verschillende)
level 5	215852	n.a.v. (3685 verschillende)
level 5	233224	n.a.v. (3685 verschillende)
level 6	215884	n.a.v. (13156 verschillende)
level 6	247916	n.a.v. (13156 verschillende)

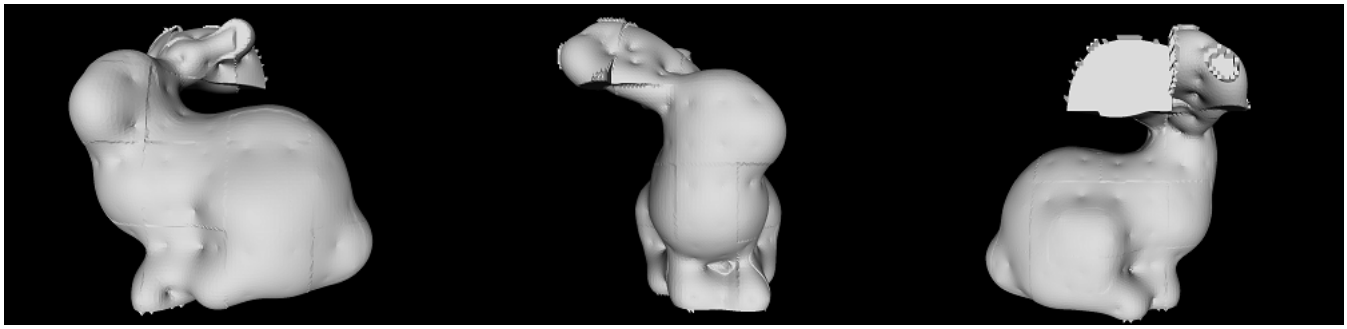
Tabel 4.18: Tabel met resultaten van de tweede hiërarchische methode. Dit gebeurt met het model res, dat het meeste detail bevat, telkens op een ander level.



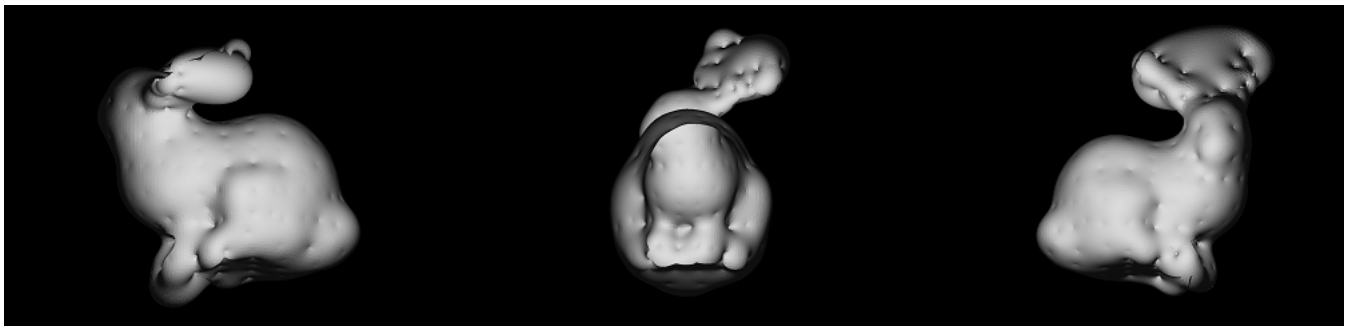
(a) Resultaat van level 1 met $\sigma = 1.0$



(b) Resultaat van level 2 met $\sigma = 1.0$

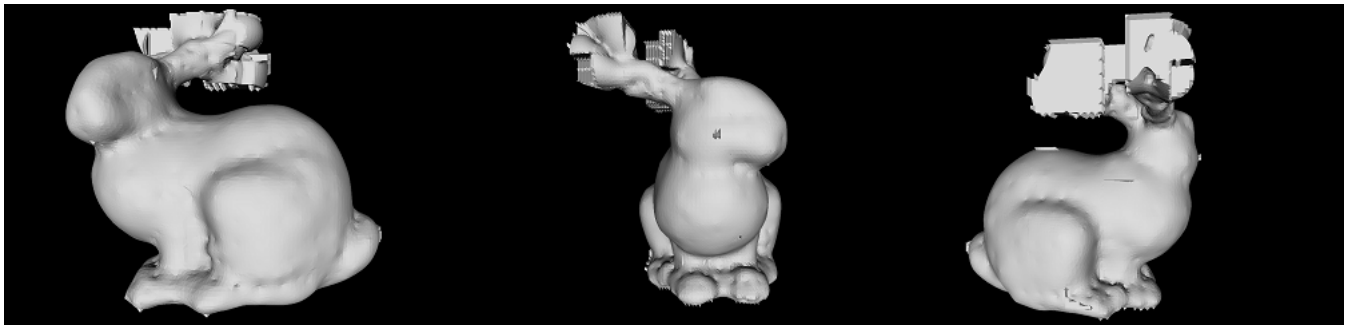


(c) Resultaat van level 3 met $\sigma = 0.05$



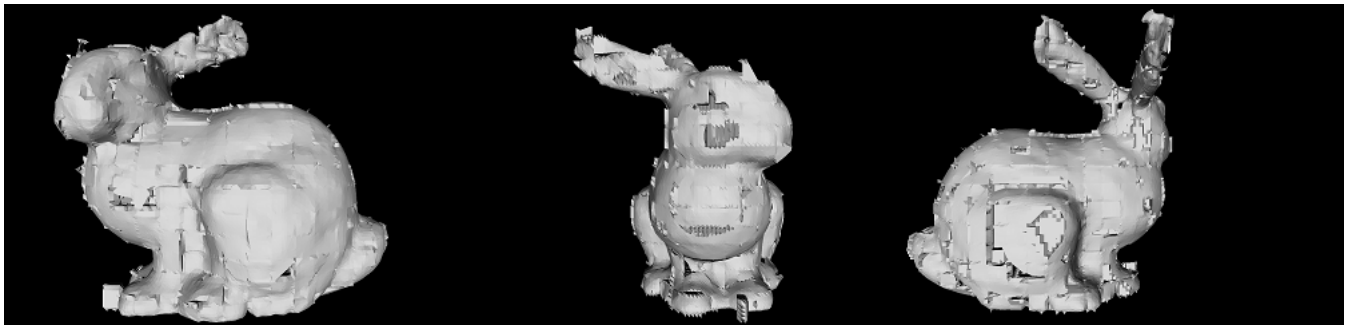
(d) Resultaat van level 3 met $\sigma = 1.0$

Figuur 4.22: Resultaten van de tweede hiërchische methode, level 1 tot en met 3.

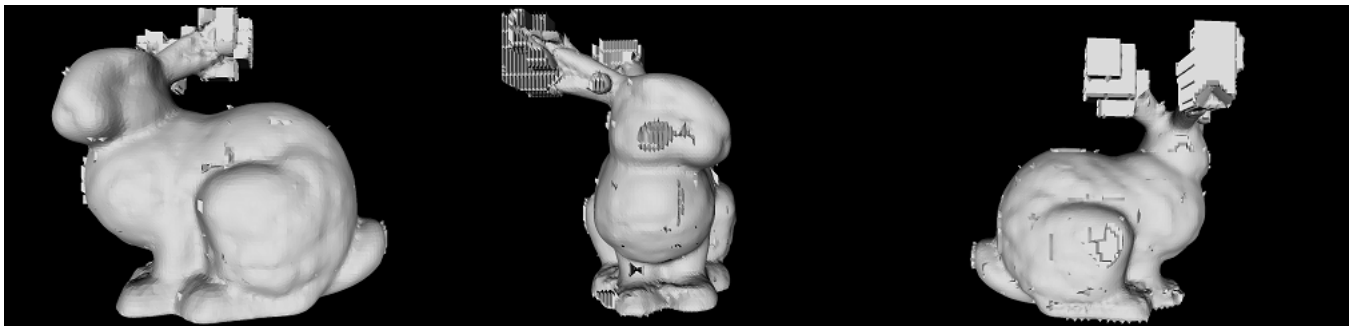


(a) Resultaat van level 4 met $\sigma = 0.03$

(b) Resultaat van level 4 met $\sigma = 1.0$

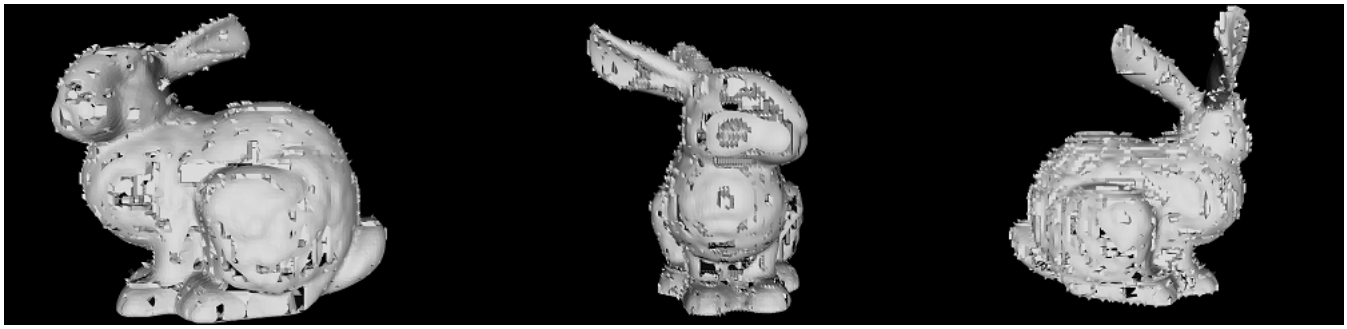


(c) Resultaat van level 5 met $\sigma = 0.01$

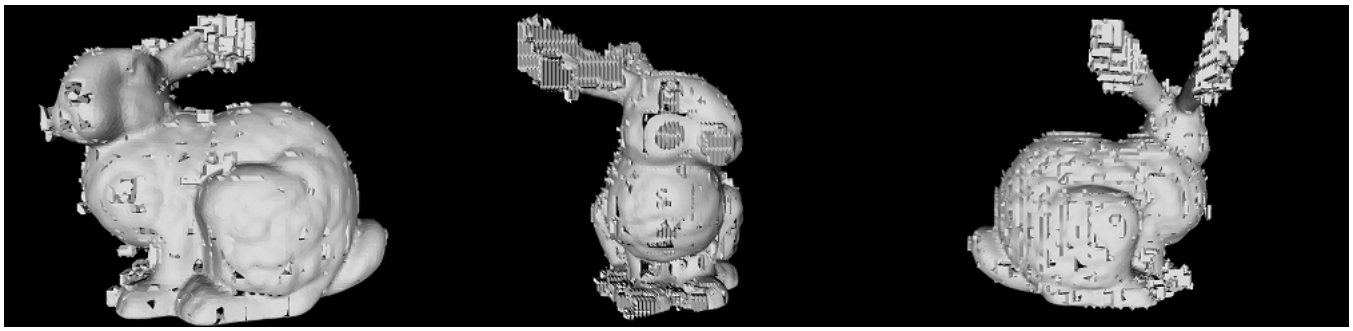


(d) Resultaat van level 5 met $\sigma = 0.02$

Figuur 4.23: Resultaten van de tweede hiërchische methode, level 4 en 5.



(a) Resultaat van level 6 met $\sigma = 0.01$



(b) Resultaat van level 6 met $\sigma = 0.012$

Figuur 4.24: Resultaten van de tweede hiërchische methode, level 6.

Hoofdstuk 5

Conclusie

We hebben gezien dat er verschillende methoden bestaan om het probleem om een impliciet oppervlak te construeren op te lossen. Sommige hiervan geven veel aandacht aan een gedetailleerd resultaat, zoals de RBF methode van Carr [6]. Andere methodes concentreren zich meer op snelheid, zoals bijvoorbeeld de Sparse-Low-degree IMplicits methode [24]. Tenslotte zijn er ook methodes die de twee voorgaande combineren, zoals de hiërarchische methode van Ohtake [26] en de MPU methode [25].
Welke methode het best gebruikt kan worden hangt zowel van het bronbestand als het doel van het resultaat af. Indien de bron een mesh van polygonen is moet er een keuze gemaakt worden uit de methode van Shen [35] of voxelisatie 2.2.5. De methode van Shen [35] is geoptimaliseerd voor meshes die bestaan uit veel driehoeken. De voxelisatie methode maakt gebruik van één enkele functie, dit is de naïve methode die beschreven werd in de paper van Carr [6]. Als in plaats hiervan de FMM methode [6] of de methode van Wu [58] gebruikt zou worden, zal dit vermoedelijk een aanzienlijke snelheidswinst geven. Het zou interessant zijn om de tijden van beide technieken te vergelijken. Voor grote verzamelingen puntenwolk data is de FMM methode te verkiezen boven deze van Wu. Dit omwille van de snelle groei van het aantal bladknopen bij de Wu methode als gevolg van het overlappingspercentage. Hierbij geldt dat een grotere overlapping een beter resultaat geeft.
Als er erg weinig punten zijn (< 2000) kan men best kiezen voor de naïve methode [6] en alle punten in één enkele matrix plaatsen. De beste keuze voor de RBF functie is biharmonische, omdat deze de meest exacte fit geeft. De hiërarchische methode is in dit geval niet zo een goede keuze. Dit omdat deze bij weinig punten slecht is in het herstellen van ontbrekende delen.
De MPU methode (zie hoofdstuk 2.4) en de hiërarchische methode zijn beide erg snelle methoden. De MPU methode is in tegenstelling tot de hiërarchische methode erg schaleerbaar. We kunnen zelfs concluderen dat de MPU methode de beste methode is om een implicit surface te construeren uit een puntenwolk. Er is echter één beperking, in de te construeren figuur mogen geen grote vlakke delen voorkomen aangezien de MPU methode slecht is in het construeren hiervan [25].
De hiërarchische methode heeft potentieel bij applicaties die LOD nodig hebben en waarbij snelheid belangrijk is. Ook de SLIM methode kan toegepast worden als vooral snelheid en

niet kwaliteit vereist is. Met geen kwaliteit bedoelen we dat de details in het resultaat niet erg belangrijk zijn. Dit kan bijvoorbeeld het geval zijn als een lage resolutie voldoende is. Een voorbeeld van een toepassing waarin beide methoden gebruikt zouden kunnen worden is in cartoon films. De karakters bestaan dan uit een puntenwolk. Om het oppervlak snel en accuraat te berekenen als het karakter naar een volgende positie inneemt kunnen implicit surfaces gebruikt worden [34]. Het is vooral interessant om gebieden zoals het gezicht meer natuurlijk te laten overkomen [34].

Bibliografie

[1] Answers.com. Aliasing. Sci-Tech Dictionary, McGraw-Hill Dictionary of Scientific and Technical Terms, http://www.answers.com/topic/aliasing.htm, 03 juni 2007.

[2] Answers.com. Linear interpolation. Sci-Tech Dictionary, McGraw-Hill Dictionary of Scientific and Technical Terms, http://www.answers.com/topic/linear-interpolation.htm, 29 oktober 2007.

[3] J. Bloomenthal. Polygonization of implicit surfaces. Comput. Aided Geom. Des., 5(4):341–355, 1988.

[4] Jules Bloomenthal. An implicit surface polygonizer. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[5] Marie-Paule Cani. Introduction to implicit surfaces, chapter Physically based animation. Jules Bloomenthal, 1997. Published under the name Marie-Paule Cani-Gascuel.

[6] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 67–76, New York, NY, USA, 2001. ACM.

[7] Jonathan C. Carr, W. Richard Fright, and Richard K. Beatson. Surface interpolation with radial basis functions for medical imaging. IEEE Transactions on Medical Imaging, 16:96–107, 1997.

[8] J. Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. Springer Berlin / Heidelberg, 1977.

[9] Encarta. Voxel. Voxel, http://encarta.msn.com/dictionary_701711465/voxel.html, 24 oktober 2008.

[10] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. J. Comput. Phys., 73(2):325–348, 1987.

[11] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, September 2000.

[12] John C. Hart. Ray tracing implicit surfaces. In SIGGRAPH 93 Modeling, Visualizing, and Animating Implicit Surfaces course notes, pages 13–1 to 13–15. 1993.

[13] John C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer, 12:527–545, 1996.

[14] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pages 339–346, New York, NY, USA, 2002. ACM.

[15] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 57–66, New York, NY, USA, 2001. ACM.

[16] Marc Levoy. The digital michelangelo project. The Digital Michelangelo Project, http://graphics.stanford.edu/projects/mich/, 3 juni 2008.

[17] Marc Levoy. The digital michelangelo project: 3d scanning of large statues. The Digital Michelangelo Project: 3D Scanning of Large Statues, http://graphics.stanford.edu/papers/dmich-sig00/, 3 juni 2008.

[18] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages pp 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[19] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph., 21(4):163–169, 1987.

[20] Shigeru Muraki. Volumetric shape description of range data using "blobby model". In SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques, pages 227–235, New York, NY, USA, 1991. ACM.

[21] N.N. Fast hierarchical methods for the n-body problem, part 2. Fast Hierarchical Methods for the N-body Problem, Part 2, http://www.cs.berkeley.edu/ ~ demmel/cs267/lecture27/lecture27.html, 19 september 2007.

[22] N.N. The marching cubes. http://www.polytech.unice.fr/Elingrand/MarchingCubes/accueil.html, 26 februari 2008.

[23] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics, 9(2):191–205, 2003.

[24] Yutaka Ohtake, Alexander Belyaev, and Marc Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In SGP '05: Proceedings of the third Eurographics symposium on Geometry processing, page 149, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[25] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. ACM Trans. Graph., 22(3):463–470, 2003.

[26] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In SMI '03: Proceedings of the Shape Modeling International 2003, page 153, Washington, DC, USA, 2003. IEEE Computer Society.

[27] Yutaka Ohtake and Alexander G. Belyaev. Dual/primal mesh optimization for polygonized implicit surfaces. In SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications, pages 171–178, New York, NY, USA, 2002. ACM.

[28] Seidel H.-P Ohtake Y., Belyaev A. G. 3d scattered data approximation with adaptive compactly supported radial basis functions. In SMI '04: Proceedings of the Shape Modeling International 2004 (SMI'04), pages 31–39, Washington, DC, USA, 2004. IEEE Computer Society.

[29] Rick Parent. Computer animation: algorithms and techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[30] Vetterling W. T. Flannery B. P. Press W. H., Teukolsky S. A. Numerical recipes in C: the art of scientific computing. Cambridge University Press, New York, NY, USA, 1993.

[31] L. Greengard R. Beatson. A short course on fast multipole methods. In Wavelets, Multilevel Methods and Elliptic PDE's. Oxford Science Publications, 1997.

[32] A. Ricci. A Constructive Geometry for Computer Graphics. The Computer Journal, 16(2):157–160, May 1973.

[33] C. Rocchini, Paolo Cignoni, Claudio Montani, P. Pingi, Roberto Scopigno, R. Fontana, M. Greco, E. Pampaloni, L. Pezzati, M. Cygielman, R. Giachetti, G. Gori, M. Miccio, and R. Pecchioli. 3d scanning the minerva of arezzo. In ICHIM (2), pages 266–272, 2001.

[34] Dietmar Saupe and Matthias Ruhl. Animation of algebraic surfaces. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[35] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, pages 896–904, New York, NY, USA, 2004. ACM.

[36] Ian Stewart. Ray-tracing of implicit surfaces. CS788: Topics in Graphics, June 19th, 1998.

[37] Gabriel Taubin. Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 13(11):1115–1138, 1991.

[38] I. TOBOR, P. REUTER, and C. SCHLICK. Multiresolution reconstruction of implicit surfaces with attributes from large unorganized point sets. In Shape Modeling International (SMI 2004), 2004.

[39] Graham M. Treece, Richard W. Prager, and Andrew H.Gee. Regularised marching tetrahedra: improved iso-surface extraction. Technical Report CUED/F-INFENG/TR 333, Speech, Vision and Robotics Group, Department of Engineering, Cambridge University, September 1998.

[40] Stanford University. Trimesh library. TriMesh library, http://graphics.cs.uiuc.edu/surface/doc/classTriMesh.html, 17 april 2007.

[41] Alan H. Watt. 3d Computer Graphics with Cdrom. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[42] Eric W. Weisstein. B-spline. B-Spline, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/B-Spline.html, 14 april 2007.

[43] Eric W. Weisstein. Bézier curve. Bézier curve, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/B-Spline.html, 14 april 2007.

[44] Eric W. Weisstein. Quadric. Quadric, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/Quadratic.html, 14 mei 2007.

[45] Eric W. Weisstein. Piecewise function. Piecewise Function, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/PiecewiseFunction.html, 15 april 2007.

[46] Eric W. Weisstein. Norm. Norm, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/Norm.html, 26 mei 2008.

[47] Eric W. Weisstein. Partial derivative. Partial Derivative, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/Norm.html, 26 mei 2008.

[48] Eric W. Weisstein. Tangent vector. Tangent Vector, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/TangentVector.html, 27 mei 2008.

[49] Eric W. Weisstein. Brent's method. Brent's Method, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/BrentsMethod.html, 27 oktober 2008.

[50] Eric W. Weisstein. Interpolation. Interpolation, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/Interpolation.html, 28 mei 2008.

[51] Eric W. Weisstein. Method of false position. Method of False Position, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/MethodofFalsePosition.html, 30 oktober 2007.

[52] Wikipedia. Aliasing. Aliasing, http://en.wikipedia.org/wiki/Aliasing.htm, 03 juni 2008.

[53] Wikipedia. Voxel. Voxel, http://en.wikipedia.org/wiki/Voxel, 08 mei 2007.

[54] Wikipedia. Neuraal netwerk. Neuraal netwerk, http://nl.wikipedia.org/wiki/Neuraal_netwerk.htm, 25 juli 2007.

[55] Wikipedia. Computer graphics. Computer graphics, http://en.wikipedia.org/wiki/Computer_graphics.html, 28 mei 2008.

[56] Wikipedia. Rendering_(computer_graphics). Rendering, http://en.wikipedia.org/wiki/Rendering_(computer_graphics).html, 28 mei 2008.

[57] Wikipedia. Regula falsi. Regula falsi, http://nl.wikipedia.org/wiki/Regula_falsi.html, 29 oktober 2007.

[58] Xiaojun Wu, Michael Yu, and Wang Qi Xia. Implicit fitting and smoothing using radial basis functions with partition of unity. In CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics, pages 139–148, Washington, DC, USA, 2005. IEEE Computer Society.

[59] G. Yngve and G. Turk. Creating smooth implicit surfaces from polygonal meshes. Technical report, Graphics, Visualization, and Usability Center. Georgia Institute of Technology, 1999.

Bijlage A

Wiskunde

A.1 Gauss-Jordan

Gauss-Jordan rij-eleminatie is een methode om een stelsel van lineaire vergelijkingen op te lossen. Dit stelsel ziet er als volgt uit:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1n}x_n & = & b_1 \\ a_{21}x_2 & + & a_{22}x_2 & + & ... & + & a_{2n}x_n & = & b_2 \\ \vdots & + & \vdots & + & \vdots & = & \vdots & & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & ... & + & a_{mn}x_n & = & b_m \end{array}$$

met $x_1, x_2, ...x_n$ de onbekenden en m het aantal vergelijkingen. Bij het oplossen zijn enkel volgende operaties toegelaten:

- het verwisselen van 2 rijen
- het vermenigvuldigen van alle elementen in een rij met een getal dat verschillend is van nul
- het veelvoud van een rij bij een andere rij optellen

Het oplossen van dit stelsel gebeurt in vijf stappen:

- Zoek in de linkerkolom een getal dat verschillend is van nul. Verwissel indien nodig een rij met de eerste rij om een niet-nulelement bovenaan deze kolom te bekomen.
- Als het element a bovenaan verschillend is van 1, vermenigvuldig dan elk element in de eerste rij met $1/a$
- Tel gepaste veelvouden van de eerste rij op bij de rijen eronder om nullen onder het hoofdelement (het element in de linkerbovenhoek) te bekomen.
- De eerste rij en eerste kolom zijn nu verwerkt. Herhaal de vorige drie stappen voor de overige rijen.

- Beschouw de laatste rij waarvan de elementen niet allemaal nul zijn. Tel gepaste veelvouden van deze rij op bij de rijen erboven om nullen boven het hoofdelement (het eerste element in deze rij dat verschillend is van nul) te bekomen. Herhaal deze operatie voor de rijen erboven.

Als we van een vierkante matrix vertrekken, waarvan alle vergelijkingen uniek zijn, zijn de oplossingen voor de onbekende $x_1, x_2, ..., x_n$ respectievelijk $b_1, b_2, ..., b_m$. Deze laatste zijn verschillend van de input waarden en werden bekomen door rij-operaties uit te voeren.

A.2 LU decompositie

Deze methode wordt net zoals de Gauss-Jordan methode gebruikt voor het oplossen van een stelsel lineaire vergelijkingen. De methode is echter aanzienlijk sneller. Het basisidee is om de initiële matrix A op te splitsen in een bovendiagonaalmatrix U en onderdiagonaal matrix L. Hierna kunnen de onbekende variabelen gevonden worden door matrixvermenigvuldiging, wat het sneller maakt dan de rij-operaties. Een nadeel is dat om deze methode toe te kunnen passen het om een vierkante matrix moet gaan.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{32} & A_{24} \\ A_{31} & A_{23} & A_{33} & A_{34} \\ A_{41} & A_{24} & A_{34} & A_{44} \end{bmatrix} = L \cdot U = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{23} & L_{33} & 0 \\ L_{41} & L_{24} & L_{34} & L_{44} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{32} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}$$

De L en U componenten worden per kolom als volgt berekend:

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} \cdot U_{kj} \quad voori = 1, ..., j$$

$$L_{ij} = \frac{1}{U_{jj}} \Big(A_{ij} - \sum_{k=1}^{j-1} L_{ik} \cdot U_{kj} \Big) \quad voori = j+1, ..., n$$

Voor de eerste kolom wordt dit:

$$\begin{aligned} U_{11} &= A_{11} \\ L_{21} \cdot U_{11} &= A_{21} \\ L_{31} \cdot U_{11} &= A_{31} \\ L_{41} \cdot U_{11} &= A_{41} \end{aligned}$$

Verder geldt:

$$
\begin{aligned}
A_{11} \cdot x_1 + A_{12} \cdot x_2 + A_{13} \cdot x_3 + A_{14} \cdot x_4 &= b_1 \\
A_{21} \cdot x_1 + A_{22} \cdot x_2 + A_{23} \cdot x_3 + A_{24} \cdot x_4 &= b_2 \\
A_{31} \cdot x_1 + A_{32} \cdot x_2 + A_{33} \cdot x_3 + A_{34} \cdot x_4 &= b_3 \\
A_{41} \cdot x_1 + A_{42} \cdot x_2 + A_{43} \cdot x_3 + A_{44} \cdot x_4 &= b_4
\end{aligned}
$$

$$
\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{32} & A_{24} \\ A_{31} & A_{23} & A_{33} & A_{34} \\ A_{41} & A_{24} & A_{34} & A_{44} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}
$$

$$
\begin{aligned}
A \cdot x &= b \\
(L \cdot U) \cdot x &= b \\
L \cdot (U \cdot x) &= b \\
U \cdot x &= y \\
L \cdot y &= b
\end{aligned}
$$

De waarden voor y worden op volgende manier gevonden:

$$
\begin{aligned}
L_{11} \cdot y_1 &= b_1 \\
y_1 &= b_1 / L_{11}
\end{aligned}
$$

Het stelsel $A \cdot x = b$ werd omgezet in volgend stelsel $L \cdot y = b$ dat op een efficiënte manier opgelost kan worden, namelijk door eenvoudig L met y te vermenigvuldigen. Dit is aanzienlijk sneller dan methoden die met rij-operaties werken, zoals onder andere Gauss-Jordan.

A.3 biconjugate gradient method

Deze methode is een iteratieve methode. De matrix A die gebruikt wordt is strikt positief definiet, dit wil zeggen dat het getal op de diagonaal het grootste is van de hele rij. b is de overeenkomstige kolommatrix van A. Verder wordt er ook nog een preconditioner M gebruikt om de initiële matrix voor de iteratie op te stellen. Een voorbeeld van een preconditioner is de Jacobi preconditioner:

$$
m_{i,j} = \begin{cases} a_{i,j} & \text{if } i = j; \\ 0 & \text{anders.} \end{cases}
$$

Bij de iteratie wordt in twee richtingen naar een oplossing gezocht: $p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}, \tilde{p}^{(i)} = \tilde{r}^{(i-1)} + \beta_{i-1} \tilde{p}^{(i-1)}$.. Bij elke iteratie wordt dan telkens een deel (residu)

toegevoegd $r^{(i)} = r^{(i-1)} - \alpha_i A p^{(i)}, \tilde{r}^{(i)} = \tilde{r}^{(i-1)} - \alpha_i A^T \tilde{p}^{(i)}$.
Het resultaat wordt ten slotte in een kolom matrix x geplaatst.

Bereken $r(0) = b - Ax(0)$ voor de initiële waarden van $x(0)$.
Kies $\tilde{r}(0)$(bijvoorbeeld, $\tilde{r}(0) = r(0)$).
for $i = 1, 2, ...$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  solve $M^T \tilde{z}^{(i-1)} = \tilde{r}^{(i-1)}$
  $\rho_{i-1} = {z^{(i-1)}}^T \tilde{r}^{(i-1)}$
  if $\rho_{i-1} = 0$, **method fails**
  if $i = 1$
    $p^{(i)} = z^{(i-1)}$
    $\tilde{p}^{(i)} = \tilde{z}^{(i-1)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p(i) = z(i-1) + \beta_{i-1} p^{(i-1)}$
    $\tilde{p}(i) = \tilde{z}(i-1) + \beta_{i-1} \tilde{p}^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\tilde{q}^{(i)} = A^T \tilde{p}^{(i)}$
  $\alpha_i = \alpha_{i-1} / {\tilde{p}^{(i)}}^T q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  $\tilde{r}^{(i)} = \tilde{r}^{(i-1)} - \alpha_i \tilde{q}^{(i)}$
check convergence; continue if necessary
end

Bijlage B

Verklarende woordenlijst

- aliasing:
  is het effect waarbij verschillende signalen in een beeld per sample tot hetzelfde resultaat kunnen leiden. Hierdoor kan het omgekeerde proces, waarbij een sample opgesplitst wordt in de signalen waaruit het was opgebouwd, niet meer gebeuren [52], [1].

- b-spline:
  een B-Spline is een veralgemening van de Bézier curve [43].

- geodesic snakes:
  Een interactieve tool voor feature detectie op een 3D driehoeken mesh.

- geometrische hashing:
  Is een methode om efficiënt twee-dimensionale objecten, voorgesteld door discrete punten die een affine transformatie ondergaan, te zoeken.

- harmonische functie:
  elke reële functie waarvan de tweede afgeleide continue is en aan de Laplace vergelijking voldoet wordt harmonische functie genoemd.

- hysteresis thresholding:
  Hysteresis is een fenomeen waarin twee of meer fysische hoeveelheden een relatie hebben die afhangt van eerdere waarden. Meer specifiek, de reactie van Y neemt andere waarden aan voor een verhoging van X dan voor een verlaging van X. Als men X waarden over een bepaald bereik afgaat geeft de grafische voorstelling van Y versus X een gesloten curve waarnaar verwezen wordt als de hysteresis loop. De reactie van Y lijkt achter te lopen op de input van X.

- interpolatie:
  Interpolatie in N-dimensies is een methode om een functie te vinden die door een aantal gekende punten gaat zodat ook voor een ander willekeurig punt dat op de functie ligt de functiewaarde berekend kan worden [50].

- iso-surface:
  een oppervlak dat impliciet omschreven wordt, wordt ook wel implicit surface of isosurface genoemd.

- KD-boom:
  is een multidimensionele zoekboom voor punten in een k dimensionele ruimte. Bij het opstellen van de boom wordt de node opgesplitst langs de langste as. De opsplitsing gaat verder tot er in elke blad maar 1 punt zit.

- Laplace vergelijking:
  de scalaire vorm van de Laplace vergelijking is de partiële afgeleide

  $$\nabla^2 \psi = 0$$

  waarbij ∇^2 de Laplaciaan genoemd wordt en de functie ψ de functie is waarvan de partiële afgeleide berekend wordt.

- Legendre polynoom: Legendre polynomen worden ook wel Legendre functies genoemd en voldoen aan volgende vorm:

  $$P_n(z) = \frac{1}{2\pi i} \oint (1 - 2tz + t^2)^{-1/2} t^{-n-1} dt.$$

  Enkele voorbeelden:

  $$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \frac{1}{2}(3x^2 - 1) \\ P_3(x) &= \frac{1}{2}(5x^3 - 3x) \\ P_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3) \end{aligned}$$

- neurale netwerken
  Neurale netwerkken worden gebruikt om complexe situaties te modelleren. Ze worden bijvoorbeeld gebruikt om de datastromen in computernetwerken of om de menselijke hersenen te simuleren. Het bestaat uit een verzameling knopen (neuronen) die verbonden zijn met elkaar. In een netwerk zijn er verschillende lagen. Functies, waaronder de gaussiaanse, zorgen voor de verwerking van de invoer [54], [11].

- norm:
  de norm geeft de lengte tussen twee punten weer en wordt als volgt berekend:

$$||x|| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$$

met n de dimensie van de punten.

- oppervlak element:
  element dat een deel van een oppervlak van een figuur voorstelt. Een verzameling punten kan bijvoorbeeld een oppervlak voorstellen. Een ander voorbeeld is een verzameling driehoeken die een mesh vormen en ook een oppervlak van een figuur voorstellen.

- particle:
  onderdeel van een groter geheel. Elk particle heeft verschillende eigenschappen zoals wrijving en viscositeit. Een verzameling van verschillende particles wordt gebruikt om objecten, waaronder vloeistoffen, te modelleren [29].

- partiële afgeleiden:
  zijn afgeleiden van een functie van meerdere variabelen waarbij de afleidbare variabele vastgehouden wordt tijdens de afleiding.

$$\frac{\delta f}{\delta x_m} = \lim_{h \to 0} \frac{f(x_1, ..., x_m + h, ..., x_n) - f(x_1, ..., x_m, ..., x_n)}{h}$$

  Voorbeeld:

$$\frac{\delta x^2 + y^2}{\delta x} = 2x + y^2$$

- piecewise functie:
  is een functie die gedefinieerd wordt over een sequentie van intervals. Een voorbeeld hiervan is de absolute waarde functie:

$$|x| = \begin{cases} x & \text{if } x > 0; \\ 0 & \text{if } x = 0; \\ -x & \text{if } x < 0. \end{cases}$$

- quadric:
  is een kwadratische functie van de vorm $Au^2 + 2Buv + Cv^2$ die gebruikt wordt om een oppervlak te construeren.

- range data:
  data van een digitaal ingescand 3D beeld of 3D voorwerp.

- regula falsi:
  is een numerieke methode om de nulpunten van een functie te bepalen. Het algoritme wordt meestal gebruikt om nulpunten van hogere graadsfuncties te berekenen. De methode werkt iteratief en gaat in intervallen zoeken naar een nulpunt [57], [51].

- renderen:
  is de constructie van een digitale afbeelding uit een drie-dimensionaal model met behulp van een computer. Het resultaat bevat informatie over de geometrie, de belichting, de schaduw en de eigenschappen van het model [41].

- rigid transformation:
  bij transformaties van een rigide object blijven de hoeken tussen de polygonen van de mesh bewaard. Ook de afstandsverhoudingen tussen de polygonen blijft bewaard.

- scalair veld of scalaire funtie:
  Een scalair veld is een functie $f : R^n -> R$ die elke x afbeeldt op een getal.

- scalaire functie:
  Is een functie $f(x_1, ..., x_n)$ van één of meer variabelen waarvan het bereik één-dimensionaal is. Dit wil zeggen dat we een functie-waarde in R terugkrijgen als we een punt$(x_1, ..., x_n)$ aan f meegeven.

- singulier punt:
  als de gradiënt (of, equivalent de tangens vector [48] (zie verklarende woordenlijst B)) niet bestaat, noemt men dit punt singulier.

- spin images:
  Een spin image geeft een beschrijving van een 3D object in 2D. Het object zelf bestaat uit een mesh van punten. In wezen stelt een spin image een straal en hoek voor ten opzichte van elke ander punt in de mesh. Het resultaat kan zorgt ervoor dat overeenkomsten van vertices tussen twee object makkelijk opgespoord kunnen worden.

- splatting:
  Splatting wordt ook wel ray-casting genoemd en is een techniek om een volume te renderen. Dit is het tegenovergestelde van ray-tracing. Voor elke voxel in de volume dataset wordt de bijdrage aan het beeld berekend. Het algoritme „gooit„ dus voxels in het beeld.

- tangens vector:
  voor een curve met positie vector $r(t)$ wordt de tangens vector $\hat{T}(t)$ gedefinieerd door:

$$\hat{T}(t) \equiv \frac{dr}{ds}$$

  waarbij t een parametrisatie variabele is en s de booglengte.

- view frustum:
  De kegel die gevormd wordt door lijnen vanuit de camera naar elke hoek van het kijkvlak te tekenen.

- voxel:
  Een enkele voxel is de kleinst onderscheidbare box in een drie-dimensionale ruimte. Dit wil zeggen dat als een regulier grid in een drie-dimensionale ruimte opgesplitst wordt in verschillende boxen, deze boxen allemaal voxels zijn. In twee dimensies wordt deze voorstelling een pixel genoemd.