ministerie van verkeer en waterstaat

rijkswaterstaat

dienst getijdewateren

VISUALISATIE VAN 3D SCALAIRE VELDEN

Taakverslag van Jack Versloot

Technische Universiteit Delft
Vakgroep Computer Graphics
maart 1991

VISUALISATIE VAN 3D SCALAIRE VELDEN

Jack Versloot
maart 1991

SAMENVATTING

Bij Rijkswaterstaat worden berekeningen en metingen uitgevoerd om het gedrag van allerlei natuurlijke processen te controleren, te modelleren en te voorspellen. Voor interpretatie van deze (hoofdzakelijk 2D) gegevens wordt gebruikt gemaakt van visualisatie.

Er worden echter steeds meer 3D modellen ontwikkeld. Deze modellen produceren een gigantische hoeveelheid data die in een zo eenvoudig mogelijke vorm gegoten worden voor de gebruiker. Ook hier wordt gedacht aan visualisatie voor interpretatie van de gegevens.

Mijn voorgangster Andrea Hin heeft, in het kader van haar afstudeerproject aan de Technische Universiteit Delft, al enig onderzoek hiernaar verricht, wat resulteerde in een programma. Dit programma maakt een afbeelding dat veel weg heeft van een aquarium; vandaar dat het ontwikkelde model de naam 'aquarium model' heeft meegekregen.

Het aquarium model bleek enige beperkingen te hebben qua snelheid en mogelijkheden. In dit verslag worden verschillende methoden besproken om deze beperkingen te minimaliseren of op te heffen. Hiervoor worden verschillende ideeën aangedragen om het programma te verbeteren, nieuwe mogelijkheden eraan toe te voegen of nieuwe methoden te gebruiken.

Aan het eind van dit verslag worden enige voorstellen gedaan om (in het kader van een afstudeerproject) nader te worden uitgewerkt.

VOORWOORD

De afsluiting van mijn studie informatica aan de Technische Universiteit in Delft vormt een afstudeerproject bij Rijkswaterstaat (Dienst Getijdewateren) in Den Haag. Dit taakverslag vormt een voorbereiding op het aansluitende afstudeerproject wat voort zal bouwen op een eerder onderzoek van Andrea Hin. Dit onderzoek heeft een programma opgeleverd voor visualisatie van 3D scalaire velden.

Hierbij wil ik Mustafa Elorche en Pieter Blokland (begeleiders Rijkswaterstaat), Frits Post en Edwin Boender (begeleiders TU-Delft) bedanken voor hun begeleiding en opbouwende opmerkingen.

J. Versloot
Den Haag, februari 1991

INHOUDSOPGAVE

1 INLEIDING

Binnen Rijkswaterstaat (RWS) worden diverse numerieke modellen gebruikt op het gebied van waterbeweging, waterkwaliteit en morfologie. Deze modellen zorgen voor een gigantische hoeveelheid berekende data die in een interpreteerbare vorm gegoten moet worden. Visualisatie van de data vormt hiervoor een oplossing; een beeld zegt immers al gauw meer dan een reeks getallen.

De huidige visualisatietechnieken bij RWS maken gebruik van een 2D kader: isolijnen (zie 2.2.1) worden afgebeeld op horizontale of verticale vlakken (figuur 1.1).

De behoefte aan visualisatie is de laatste jaren steeds sterker geworden. Bij RWS is de behoefte ontstaan om de visualisatie uit te voeren in een 3D kader. Andrea Hin heeft al eerder enig onderzoek hiernaar verricht en dit resulteerde in het aquarium model [Hin90].

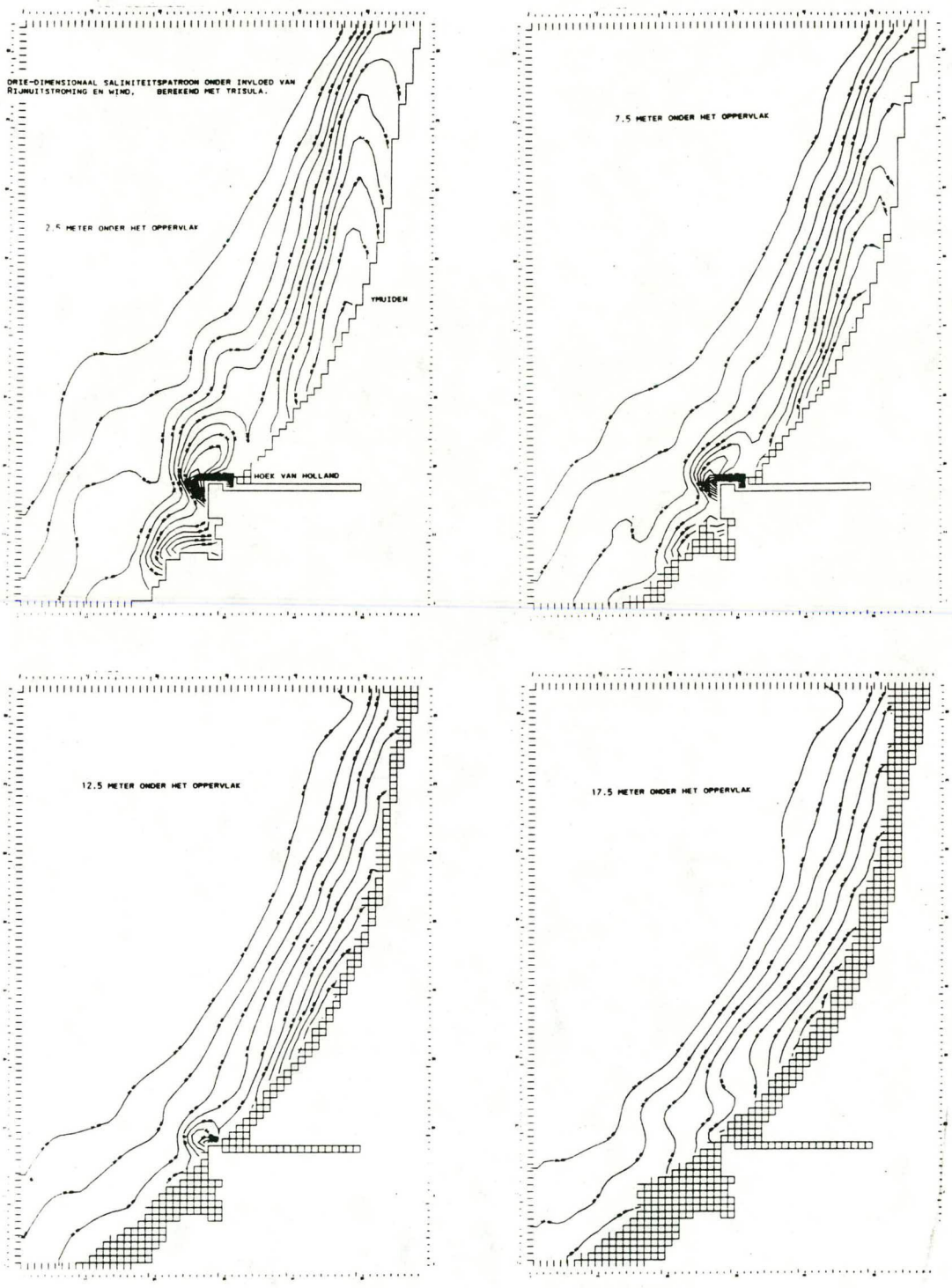
Het aquarium model is op zichzelf correct maar heeft enkele beperkingen, te weten:

- De snelheid waarmee een afbeelding wordt gegenereerd is te laag (± 1½ uur). Dit kan zeker sneller.
- Het model is gemaakt voor een bepaald type dataset.
- De afbeeldingsmogelijkheden van het model zijn te beperkt.

In dit verslag zullen verschillende mogelijkheden worden onderzocht om deze beperkingen te verminderen.

De indeling van dit taakverslag in hoofdstukken is als volgt.

In hoofdstuk 2 worden methoden besproken die te maken hebben met visualisatie van scalaire velden in het algemeen. Hoofdstuk 3 is speciaal gericht op het aquarium model en de gebruikte ray casting methode; hierin worden beschrijvingen gegeven van de gebruikte methode en van alle van belang zijnde facetten van het model. Hoofdstuk 4 stelt verschillende mogelijkheden voor om de beperkingen/tekortkomingen van het model te verbeteren of op te heffen. Tenslotte zullen in hoofdstuk 5 enige punten worden voorgesteld om nader te worden onderzocht tijdens het afstudeerproject.



figuur 1.1 Isolijnen van zoutconcentraties (salaniteit) op 2½, 7½, 12½ en 17½ meter onder het oppervlak.

2 VISUALISATIE VAN 3D SCALAIRE VELDEN

2.1 INLEIDING VOLUME VISUALISATIE

Volume visualisatie is het genereren van een afbeelding uit volume data. Onder volume data wordt verstaan: een 3D gebied waar op (meestal regelmatige) intervallen scalaire meet- of rekengegevens aanwezig zijn. Zodoende kan het gebied gezien worden als een verzameling volume-elementen (voxels) met in elk element één scalaire waarde.

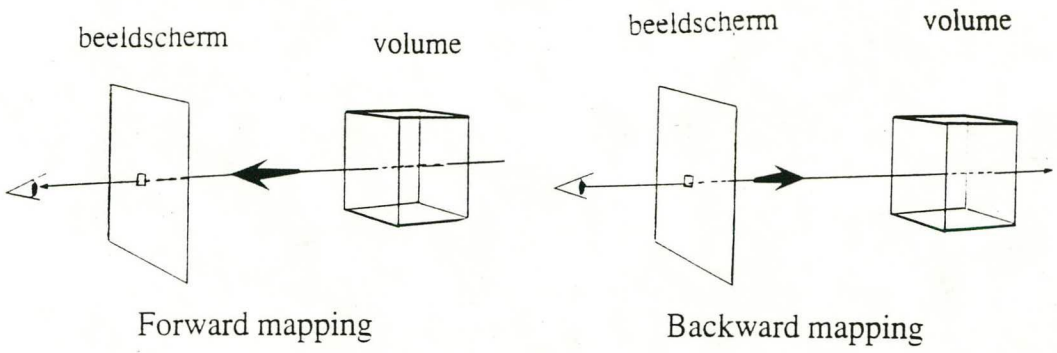
Traditionele methoden voor het afbeelden van polygons kunnen nu niet direct gebruikt worden voor het afbeelden van volume data. Hiervoor zijn andere methoden ontwikkeld.

Het primaire doel van volume visualisatie is te zorgen voor een grafische weergave van de aangeboden volume data en wel zodanig dat deze zo goed mogelijk interpreteerbaar is voor de gebruiker. Dit hoeft niet noodzakelijk te betekenen dat de kwaliteit van de representatie hoog moet zijn! Vaak is de kwaliteit toch van belang zoals in de medische wereld, en kunnen details een grote rol spelen.

Voor volume visualisatie zijn verschillende methoden bekend en deze zijn grofweg in drie categorieën in te delen:

- 'Contouring'
  Alleen die delen van de dataverzameling waarvan het scalaire veld een bepaalde constante waarde heeft worden weergegeven. Er ontstaan dan contouren. Dit zijn krommen in 2D (isolijnen) en oppervlakken in 3D (iso-oppervlakken).
- 'Slicing'
  Niet de hele dataverzameling wordt weergegeven maar slechts een of meerdere gekozen vlakken die het volume doorsnijden. Deze vlakken kunnen dan bekeken worden onder verschillende aanzichten.
- 'Volume rendering'
  De gehele dataverzameling wordt direct en zo volledig mogelijk weergegeven.
  Deze categorie valt weer te splitsen in (figuur 2.1):

  * 'forward mapping'
    Het uitgangspunt is hier de dataruimte.
  * 'backward mapping'
    Hier is het uitgangspunt de beeldruimte.



figuur 2.1 Forward mapping en Backward mapping.

Door de verscheidenheid van de aangeboden data, tezamen met de uiteenlopende wensen en eisen van de gebruiker, is de beoordeling van al deze methoden een moeilijke zaak. De volgende punten kunnen onder andere van belang zijn voor een beoordeling:

- De kwaliteit van de afbeelding (visuele effecten, mate van detaillering, ongewilde misvormingen enz).
- De snelheid waarmee de afbeelding gegenereerd wordt.
- De mate van interactie.
- Het geheugengebruik.

De nu volgende paragrafen zullen wat dieper ingaan op de verschillende methoden voor volume visualisatie.

2.2 CONTOURING

Contouring kan gebruikt worden voor zowel 2D als 3D scalaire velden. Niet de gehele dataverzameling wordt afgebeeld, maar slechts die delen waarvan het scalaire veld een bepaalde constante waarde heeft. Wanneer het scalaire veld beschreven wordt door de funktie f, dan is dit $f(x,y) = c$ voor 2D en $f(x,y,z) = c$ voor 3D. Op deze manier ontstaan dan contouren.

De gebruiker zal zelf de gewenste interpretatie hieraan moeten geven.

In de nu volgende paragrafen worden enige vormen van contouren besproken in 2D en in 3D.

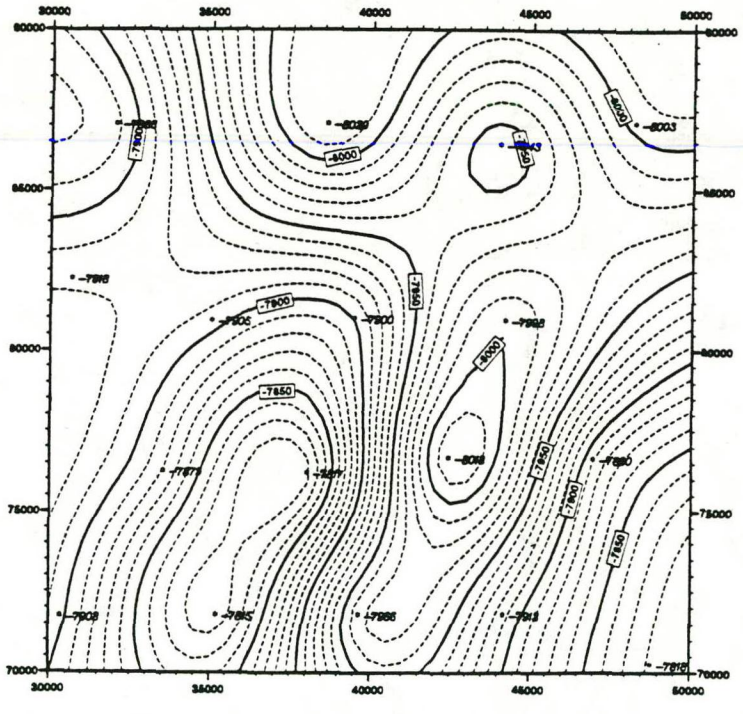
2.2.1 2D CONTOURING

Bij 2D contouring komen contourlijnen, contourbanden en kleurcontouren aan de orde.

CONTOURLIJNEN

Contourlijnen of isolijnen worden gegenereerd uit een 2D dataverzameling van een scalair veld [Lith90], [Hin89]. Voor zo'n contourlijn geldt dat $f(x,y) = c$ (zie paragraaf 2.2).

Meestal worden in een afbeelding meerdere lijnen tegelijkertijd weergegeven, zodat enig inzicht wordt verkregen in het verloop van het scalaire veld (figuur 2.2). Voor elke contourlijn i geldt dan dat $f(x,y) = c_i$. Om al deze lijnen van elkaar te kunnen onderscheiden wordt gewerkt met een kleurcodering of met kentallen.



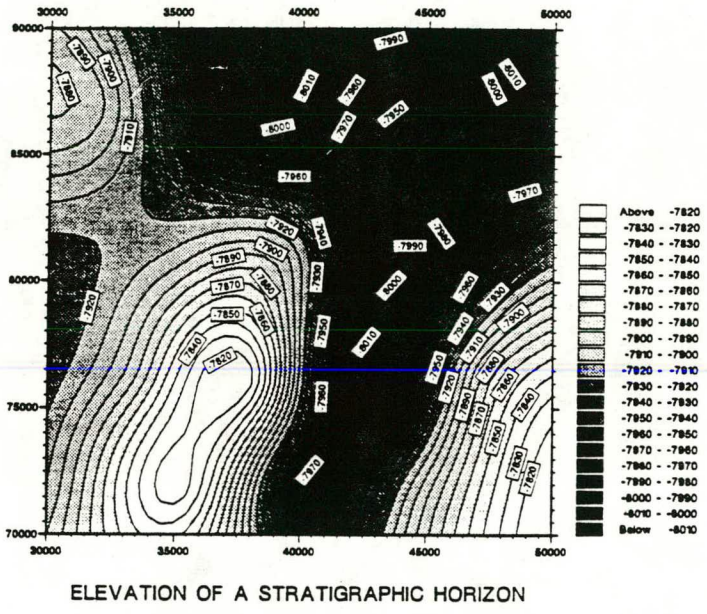
figuur 2.2 Isolijnen voorzien van kentallen.

Voor een uitgebreide beschrijving van de verschillende technieken voor het vinden van contourlijnen door interpolatie of door contourtracing zie [Hin89].

CONTOURBANDEN

Door verschillende contourniveaus c_i te gebruiken is het mogelijk om elk deel van de dataverzameling toe te kennen aan het dichtstbijzijnde contourniveau. Zo vindt er een opdeling plaats van het waardenbereik: $c_i \leq f(x,y) \leq c_{i+1}$. Als aan elk contourniveau een andere kleur wordt toegekend ontstaan nu geen lijnen meer maar banden [Lith90], [Stol90]. Figuur 2.3 geeft hiervan een voorbeeld.

Deze techniek levert dus een afbeelding met een discontinu kleurenverloop.

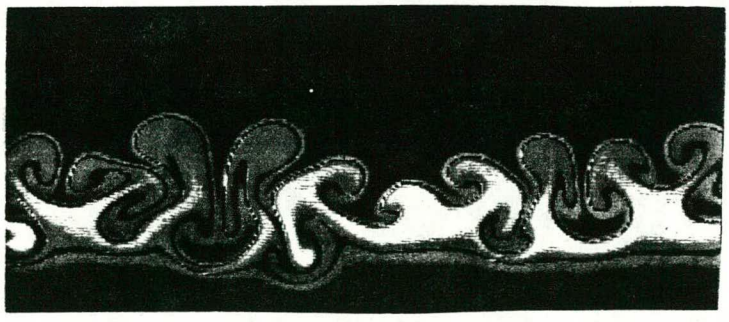


ELEVATION OF A STRATIGRAPHIC HORIZON

figuur 2.3 Contourbanden.

KLEURCONTOUREN

Elke scalaire funktiewaarde wordt nu omgezet naar een bepaalde kleurwaarde en geprojecteerd op het scherm [Lith90], [Stol90]. Door te interpoleren tussen deze waarden wordt nu een continu kleurenverloop verkregen (zie figuur 2.4).



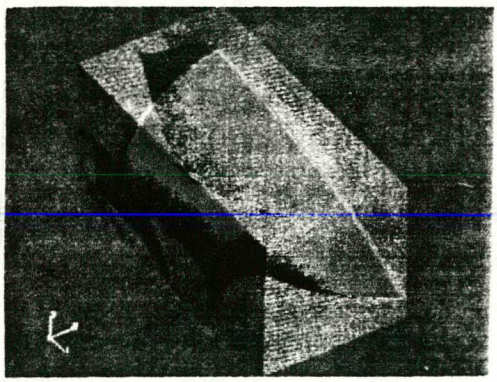
figuur 2.4 Kleurcontouren.

2.2.2 3D CONTOURING

Bij 3D contouring kunnen contouroppervlakken (iso-oppervlakken) weergegeven worden; zij zijn het 3D equivalent van contourlijnen.

CONTOUROPPERVLAKKEN

Het zoeken in de dataverzameling naar alle delen waar het scalaire veld een bepaalde constante waarde heeft ($f(x,y,z) = c$), levert in het algemeen meer dan één iso-oppervlak. Deze oppervlakken worden vaak eerst benaderd door polygons om daarna weergegeven te worden [Lith90], [Stol90]. Hiervoor maakt men vaak gebruik van transparantie om meerdere vlakken tegelijk af te beelden, en van shading om een goede indruk te geven van de 3D vorm (figuur 2.5).



figuur 2.5 Contouroppervlakken.

Door interactief de contourwaarde te variëren onstaat het idee van een animatie: het iso-oppervlak beweegt door de ruimte. De gebruiker kan nu ook een idee krijgen van het hele volume. Immers, waar de beweging van het contouroppervlak groot is zal het scalaire veld sterk in waarde veranderen en andersom.

Het meeste rekenwerk is het vinden van de iso-oppervlakken. Voor methoden voor het vinden van deze iso-oppervlakken zie [Hin89].

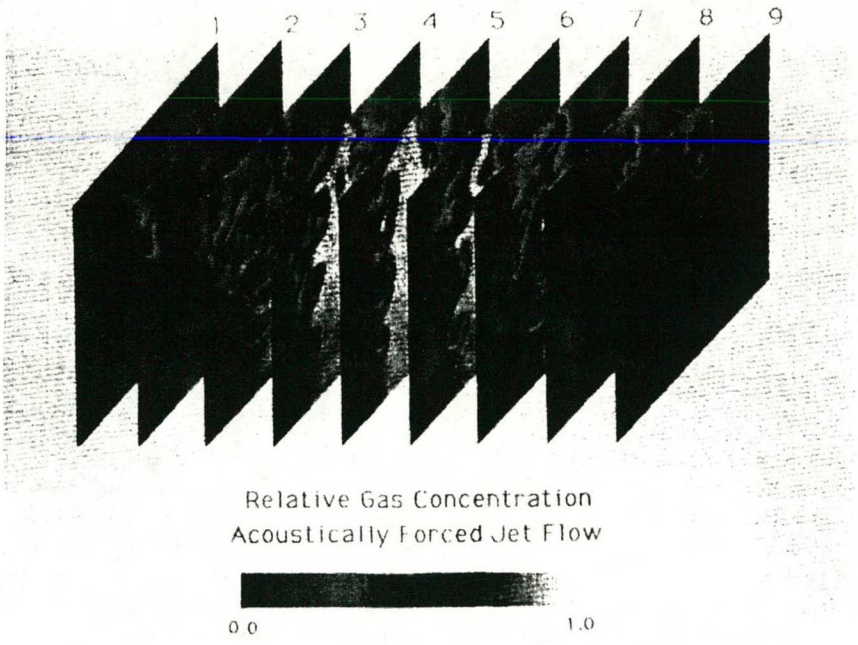
2.3 SLICING

Onder het 'slicen' van een volume wordt verstaan het doorklieven van het volume met een (snij)vlak en het vervolgens weergeven van de data op dit vlak. Deze methode levert geen echte 3D afbeelding op maar kan wel interactief worden gevisualiseerd. Het gebruik van meerdere parallelle vlakken kan wel enig inzicht verschaffen in het 3D verloop. In de nu volgende subparagrafen wordt achtereenvolgens behandeld: de opdeling van het volume met behulp van één of meerdere vlakken en de opdeling van het volume in subvolumes met behulp van een BSP-boom.

2.3.1 OPDELING MET BEHULP VAN VLAKKEN

Bij het opdelen van het volume in vlakken wordt door de gebruiker een snijvlak bepaald en vervolgens worden de 2D contourlijnen op dit vlak weergegeven. Dit is dus geen iso-oppervlak maar een willekeurig gekozen plat vlak.
De snelheid van dit visualisatieproces is groot. Met behulp van krachtige hulpmiddelen (bv parallelle processoren) kan het gekozen vlak eventueel real time en interactief door het volume bewogen worden.

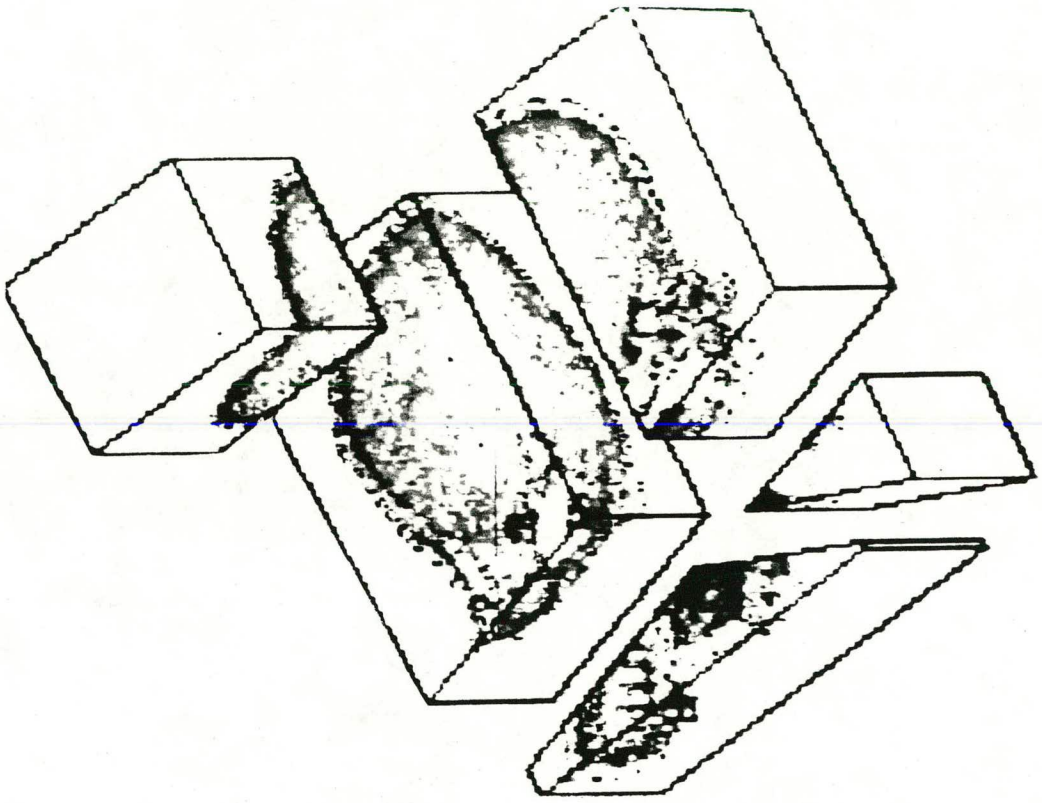
Er kunnen wel meerdere (niet noodzakelijk parallelle) vlakken tegelijkertijd worden weergegeven (figuur 2.6). Nu is echter wel 'hidden surface removal' [Watt89] nodig om overlappende stukken correct weer te geven.



figuur 2.6 Reeksen contourvlakken.

2.3.2 OPDELING MET BEHULP VAN EEN BSP-BOOM

De nu volgende methode maakt gebruik van BSP-bomen voor de onderverdeling van het volume [Stol90]. BSP staat voor Binary Space Partitioning. Zoals deze term al aangeeft wordt het volume telkens binair opgedeeld. Dit kan interactief en recursief gebeuren en geeft mogelijkheden voor interactieve dataselectie en voor het genereren van zogenaamde 'exploded views' (figuur 2.7).



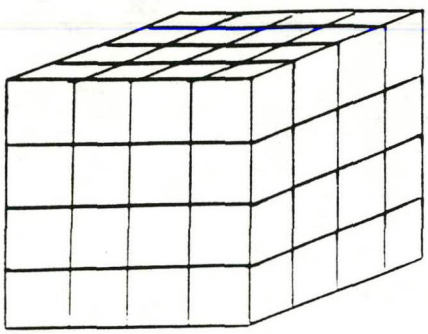
figuur 2.7 BSP-boom: exploded view.

Door nu de subvolumes in een back-to-front (BTF) volgorde af te beelden wordt er automatisch voor gezorgd dat alleen de voorste delen zichtbaar zijn.

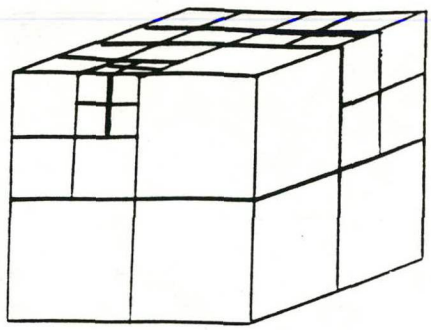
2.4 VOLUME RENDERING

Volume rendering is het rechtstreeks maken van een afbeelding van een 3D dataverzameling zonder daarbij eerst de conversie te maken naar oppervlakken. Het voordeel hiervan is duidelijk: er hoeft geen tijd verloren te gaan aan deze conversie die bovendien fouten introduceert door de benadering van de data door oppervlakken (meestal polygons).

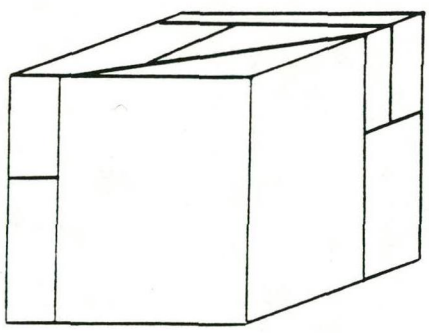
Meestal wordt aangenomen dat de meet- of rekenruimte regelmatig is opgedeeld in kubusvormige cellen (voxels). De datastructuur hiervan is een 3D array. Als de dataverzameling veel coherentie vertoont kan ook gekozen worden voor een octree benadering (zie [Garg86]). Figuur 2.8 laat een regelmatige opdeling, een opdeling door middel van octrees en een BSP opdeling zien. De eerste methode kan makkelijk gerepresenteerd worden in een array. De laatste methode is het meest flexibel, maar is wat moeilijker te representeren.



(a) regelmatige opdeling



(b) opdeling mbv octrees



(c) opdeling mbv BSP-bomen

figuur 2.8 Opdeling van een ruimte.

De methoden voor het genereren van zo'n afbeelding zijn te splitsen in (zie figuur 2.1):

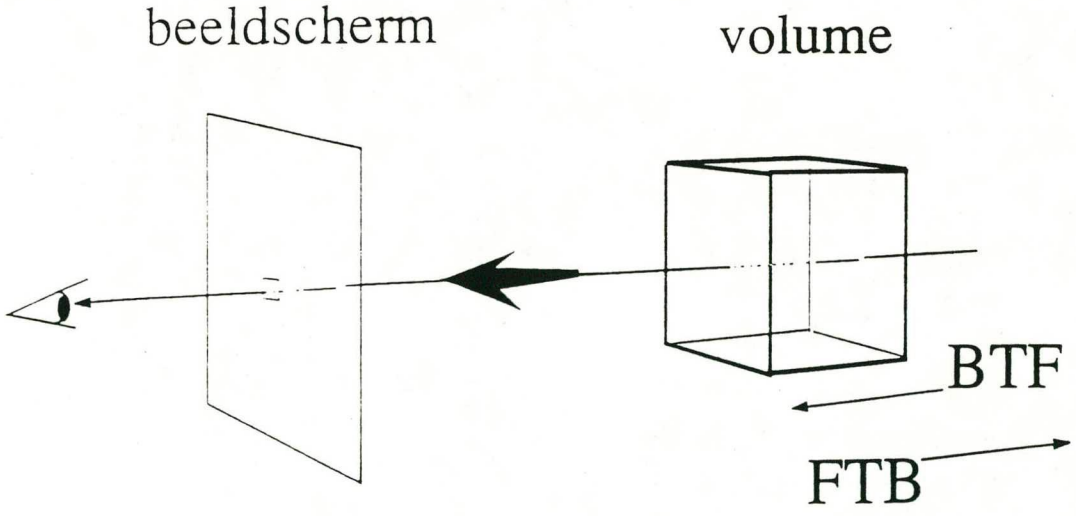
- forward mapping
  Er wordt uitgegaan van de dataruimte.
  Van elke voxel wordt bepaald welke pixels erbij horen.
- backward mapping
  Er wordt uitgegaan van de beeldruimte.
  Van elke pixel wordt bepaald welke voxels erbij horen.

De volgende paragraaf zal dieper ingaan op de forward mapping methode. Backward mapping zal worden besproken in hoofdstuk 3. Dit hoofdstuk behandelt het aquarium model, waarbij de gebruikte ray casting methode (backward mapping) ter sprake komt.

2.4.1 FORWARD MAPPING: CELL-BY-CELL

Bij de cell-by-cell methode worden alle voxels (cellen) op een speciale manier doorlopen en geprojecteerd op het beeldscherm.

De voxels kunnen op twee wezenlijk verschillende manieren worden doorlopen (figuur 2.9).



figuur 2.9 Back-to-front vs Front-to-back.

BACK-TO-FRONT (BTF)

Bij de Back-to-front methode worden de voxels op een zodanige manier doorlopen dat de voxels die het dichtst bij het oogpunt zijn gelegen later worden afgebeeld dan de voxels die verder van het oogpunt verwijderd zijn. Hierdoor kunnen eerder afgebeelde voxels overschreven worden.

Voordat het projecteren plaatsvindt kunnen de voxels geclassificeerd worden op basis van dichtheid. Er kunnen bijvoorbeeld alleen voxels worden geprojecteerd waarvan de dichtheid boven een bepaalde drempelwaarde uitkomt (binaire classificatie of thresholding). Door deze binaire classificatie kunnen afwijkingen ontstaan in de afbeelding (bijvoorbeeld gaten).

Voor iedere af te beelden voxel kan nu een grijswaarde bepaald worden door de normaalvector te schatten met behulp van een gradiënt.

Deze methode werkt het beste wanneer de geprojecteerde voxels voldoende klein zijn omdat de afbeelding anders te grof kan worden en allerlei vormen van aliasing gaat vertonen.

FRONT-TO-BACK (FTB): DE V-BUFFER METHODE

De V-buffer methode vertoont overeenkomsten met de Z-buffer methode die onder andere gebruikt wordt voor het afbeelden van polygons. De Z-buffer methode bewaart voor iedere pixel de minimale diepte (in de Z-buffer) van de geprojecteerde polygons (dit is de afstand tot het beeldscherm). Wanneer nu het volgende polygon geprojecteerd wordt, dan worden alleen die pixels afgebeeld waarvoor de bijbehorende diepte kleiner is dan de diepte in de Z-buffer.

In pseudocode is de werking als volgt:

```
VOOR elke pixel DOE
        Z-buffer[x, y] = max
        frame-buffer[x, y] = achtergrondkleur
EOD
VOOR elke polygon in een front-to-back volgorde DOE
        VOOR elke pixel van geprojecteerde polygon DOE
                Bereken diepte z
                ALS z < Z-buffer[x, y] DOE
                        Bereken kleurintensiteit i
                        Z-buffer[x, y] = z
                        frame-buffer[x, y] = i
                EOD
        EOD
EOD
```

Bij het afbeelden van de voxels op het scherm speelt niet alleen de diepte een rol maar ook de transparantie. De voxels worden nu gemengd met de al eerder afgebeelde voxels. De mate van het mengen gebeurt afhankelijk van de ondoorzichtigheid van het

volume. Dit is de reden waarom in de buffer niet de diepte maar de cumulatieve ondoorzichtigheid wordt opgeslagen (deze wordt dan de V-buffer genoemd). Bij elke menging wordt de ondoorzichtheid op de juiste plaats in de V-buffer erbij geteld. Er hoeft geen menging meer plaats te vinden als de cumulatieve ondoorzichtigheid een maximum waarde heeft bereikt. Meestal kan de cumulatieve ondoorzichtigheid een waarde hebben tussen de 0 (volledig doorzichtig) en de 1 (volledig ondoorzichtig).

Het algoritme in pseudocode voor de V-buffer methode wordt dan:

```
VOOR elke pixel DOE
        V-buffer[x, y] = 0
        frame-buffer[x, y] = achtergrondkleur
EOD
VOOR elke voxel in een front-to-back volgorde DOE
        VOOR elke pixel op de geprojecteerd voxel DOE
                ALS V-buffer[x, y] < 1 DOE
                        Bereken ondoorzichtheid o
                        Bereken kleurintensiteit i
                        Meng i met frame-buffer[x, y]
                        Bereken cum. ondoorz. in V-buffer[x, y]
                EOD
        EOD
EOD
```

Voor de V-buffer is net zoals bij de Z-buffer veel geheugenruimte nodig (afhankelijk van de resolutie van de afbeelding). Om geheugenruimte te besparen kan ook nu voor een scan-line V-buffer methode gekozen worden, net als bij de Z-buffer [Hear86], [Watt90].

Een voordeel van deze methode is dat er al vrij snel een resultaat zichtbaar is ten gevolge van het afbeelden van de voorste voxels. Naarmate er meer voxels afgebeeld worden onstaan er steeds meer verfijningen en details.

Nog een voordeel is dat deze methode een vrij simpele opzet heeft en dan ook gemakkelijk in hardware geïmplementeerd kan worden (net als BTF).

Als nadelen kunnen genoemd worden dat de FTB methode meer rekentijd en meer geheugen nodig heeft dan de BTF methode.

3 RAY CASTING

3.1 INLEIDING RAY CASTING

Ray casting is een 'volume rendering' techniek die per pixel bepaalt wat er te zien is. Vanuit het oogpunt wordt door elke pixel een straal het datavolume in gestuurd om, langs die straal enige optische grootheden te bepalen (backward mapping); vandaar de naam ray casting.

Deze methode is afgeleid van ray tracing: een methode voor het afbeelden van 3D objecten. Hierbij wordt rekening gehouden met licht, reflectie, schaduwen, transparantie enz. Het resultaat is een realistische afbeelding waarvan de kosten vrij hoog zijn.

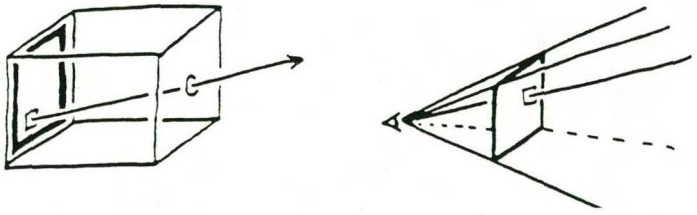
De eerst volgende paragraaf zal het principe van ray tracing uitleggen. De daarop volgende paragraaf zal gaan over het principe van ray casting, toegepast op volume data. Hier worden de verschillen met ray tracing uitgelegd en er wordt een beschrijving gegeven van een al bestaand ray casting programma.

3.2 PRINCIPE VAN RAY TRACING

Voor het maken van realistische afbeeldingen is het nodig om de wereld zo natuurgetrouw mogelijk weer te geven. Een bepaald model van de werkelijkheid wordt dan als uitgangspunt gekozen voor de te volgen methode. Je zou kunnen zeggen dat het uitgangspunt van ray tracing de 'pinhole camera' (camera obscura) [Dive89] is. In computer graphics wordt de camera als volgt gewijzigd:

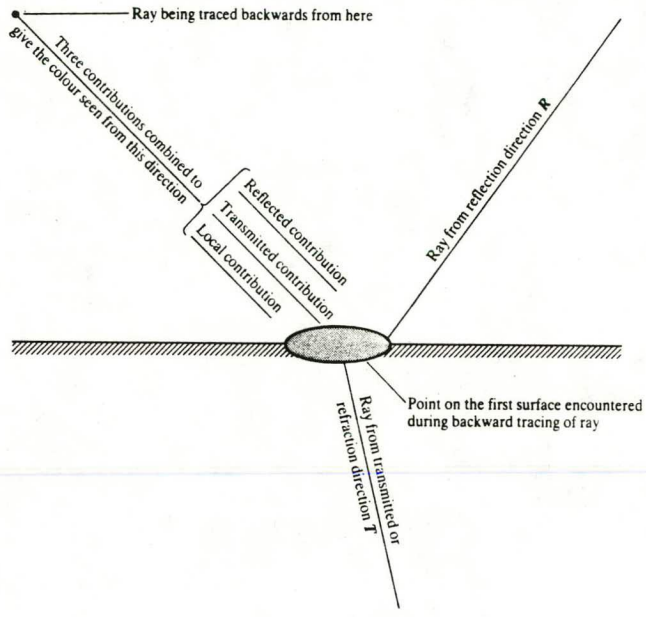
- Het gaatje in de doos wordt het oogpunt.
- De film wordt vervangen door een beeldscherm en wordt voor het oogpunt geplaatst.

Iedere pixel op het beeldscherm correspondeert nu met een klein gebiedje op de fotografische film (figuur 3.1).



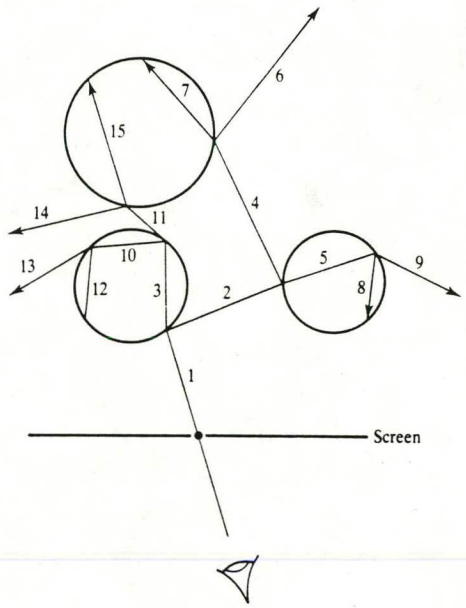
figuur 3.1 Ray tracing en de 'pinhole camera'.

Het bepalen van een kleur voor iedere pixel gebeurt als volgt (figuur 3.2).



figuur 3.2 Lichtbijdragen voor een bepaald punt.

Vanuit het oogpunt wordt door iedere pixel een straal geworpen. Elke straal wordt vervolgens gesneden met alle objecten en voor het voorste snijpunt wordt er een kleur berekend als gevolg van directe belichting (licht direct afkomstig van een lichtbron). Als het object reflecterend en/of transparant is, dan is de kleur ook afhankelijk van de stralen die gereflecteerd worden en/of de stralen die door het object heen gaan. Er zal dan een gewogen gemiddelde genomen moeten worden. Deze nieuwe secundaire stralen kunnen ook weer objecten raken en gesplitst worden enz. Hieruit volgt het recursieve karakter van ray tracing (figuur 3.3).



figuur 3.3 Het recursieve karakter van ray tracing.

Er kan tevens bepaald worden of het snijpunt zich in de schaduw van een ander object bevindt. Hiervoor worden zogenaamde schaduwstralen geworpen vanaf het snijpunt naar de lichtbron toe. Alleen wanneer de schaduwstraal geblokkeerd wordt door een ander object zal het snijpunt zich in de schaduw bevinden.

3.3 RAY CASTING: HET AQUARIUM MODEL

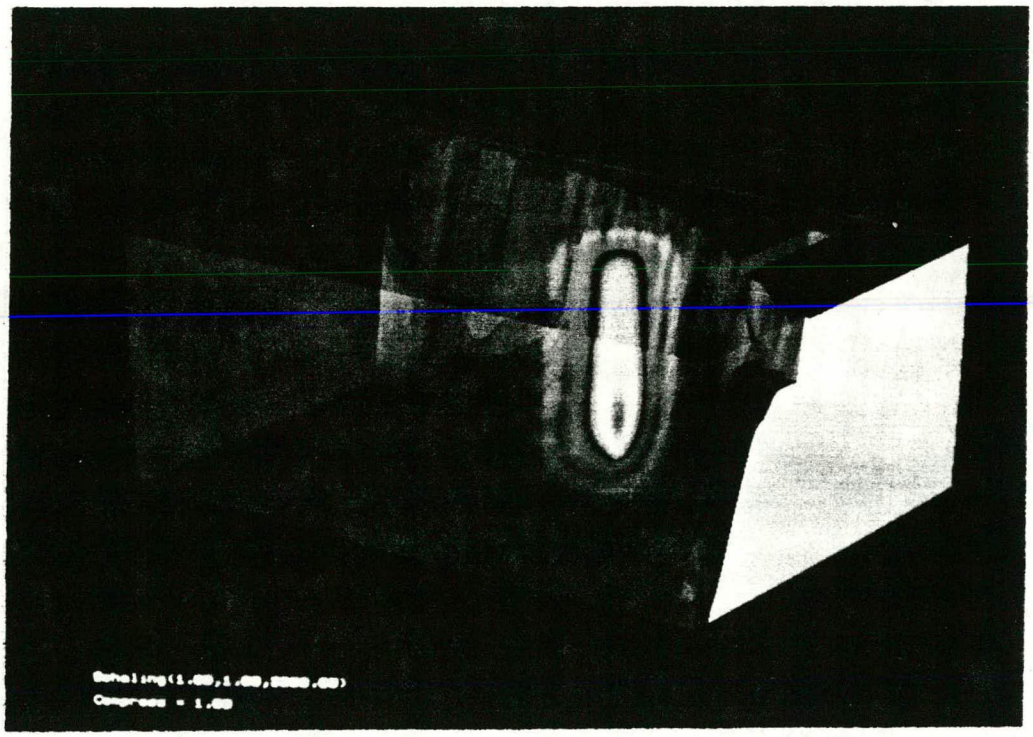
Ray casting is in principe gelijk aan ray tracing maar vertoont toch een significant verschil: er worden geen secundaire stralen verstuurd om reflectie, transparantie en schaduwen te modelleren.

Het programma van Andrea Hin [Hin90] is gebaseerd op deze methode. Er wordt alleen niet gewerkt met polygon data maar met volume data. Dit heeft de volgende veranderingen tot gevolg:

- Er worden geen snijpunten meer berekend met objecten maar met voxels. Zodoende wordt het pad bepaald van de straal door het datavolume om zo een waarde te verkrijgen van het scalaire veld langs die straal.

- Er moet rekening gehouden worden met transparantie om een zo goed mogelijk inzicht te krijgen in de ligging van het scalaire veld. De transparantie neemt af met een toename van de concentratie van de ondoorzichtige stof, en andersom.

Het programma gaat uit van een rechthoekige bak gevuld met water, een bodem en één scalair veld en produceert hiervan een afbeelding die lijkt op een aquarium; vandaar de naam aquarium model (figuur 3.4).



figuur 3.4 Een afbeelding van het aquarium model.

Om tot een goede interpretatie te komen zal de afbeelding van het model wellicht van verschillende kanten bekeken moeten worden. Een zekere mate van interactie is dus wel gewenst. Nu is de ray casting methode rekenintensief waardoor deze interactie waarschijnlijk (nog) niet gerealiseerd kan worden. Door het faseren van het programma kan toch enige interactie behouden blijven. Het is dan mogelijk om nog te experimenteren met enige grootheden zonder dat het ray casten opnieuw hoeft te gebeuren.

Het programma bestaat uit de volgende twee fasen:

- De preprocessing fase
  Hier vindt het ray casten plaats vanuit een vast aanzicht. De uitvoer van dit deel is een data file voor de volgende fase.
- De postprocessing of interactieve fase
  Hier worden de gegevens uit de data file weer ingelezen en omgezet naar kleurwaarden. Het is in deze fase mogelijk om nog te experimenteren met enige grootheden zoals kleur en ondoorzichtigheid.

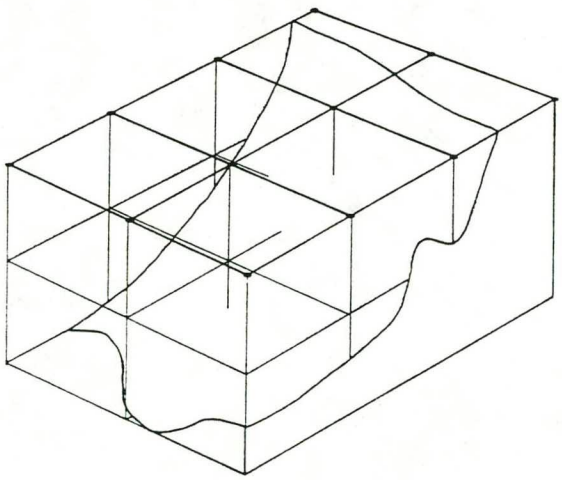
Wanneer een nieuw aanzicht gewenst is, dan zullen beide fasen opnieuw moeten worden doorlopen!

Deze opdeling levert wel de gewenste afbeelding maar is toch niet helemaal geslaagd in de opzet. De rekentijd van de preprocessing fase is zoals verwacht vrij lang, maar het afbeelden in de postprocessing fase duurt nog altijd zo'n 10-15 minuten! Dit is nauwelijks interactief te noemen.

De nu volgende paragrafen gaan wat dieper in op dit aquarium model. Achtereenvolgens wordt besproken: het gebruikte datamodel, de preprocessing fase en tenslotte de postprocessing fase.

3.3.1 HET DATAMODEL

Het aquarium model gaat uit van een regelmatig 3D rechthoekig rooster. Het hele volume wordt op deze manier opgedeeld in subvolumes: voxels (zie ook figuur 2.9a en figuur 3.5). In dit rooster moet op bepaalde punten de data van de bodem en het scalaire veld bekend zijn.



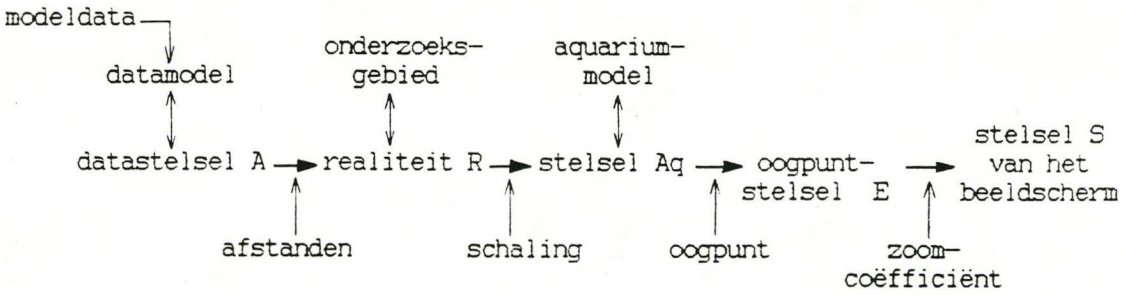
figuur 3.5 Opdeling van de ruimte in voxels.

Intern in het programma wordt het rooster opgeslagen in een statische datastructuur: de 2D array. De derde dimensie wordt dynamisch gealloceerd om geheugenruimte te sparen omdat er bij geringe diepte minder meetwaarden van het scalaire veld zijn. Onder de bodem zit immers geen scalair veld.

Om gemakkelijk te kunnen rekenen en om gemakkelijk de indices te kunnen bepalen binnen de array was het wenselijk dat alle voxels zijden zouden hebben met een eenheidslengte. Om geen beperkingen op te leggen aan de te gebruiken datasets kunnen de echte afmetingen van een voxel aan het begin van het programma worden ingevoerd.

Ook bestaat er een mogelijkheid om de afmetingen van het model te schalen in een bepaalde richting. Zodoende kunnen platte modellen opgeschaald worden om een wat betere afbeelding te verkrijgen. Dit geschaalde model wordt het aquarium model genoemd. In figuur 3.6 is dit verduidelijkt. De relatie van een punt P tussen het datamodel en het aquarium model is:

$$P_A = P_{AQ} * \text{Schaling} * \text{Afstand}.$$



figuur 3.6 Gebruikte datamodellen binnen het aquarium model.

Om het model van verschillende aanzichten te kunnen bekijken is het nodig dat er een oogpunt ingesteld kan worden.

Door het veranderen van een parameter is het mogelijk om de afstand tot het model te vergroten of te verkleinen. Dit resulteert in een kleinere respectievelijk grotere afbeelding van het model op het beeldscherm.
Voor de details over de transformatie van het beeldscherm S naar het datamodel A wordt verwezen naar [Hin90].

3.3.2 DE PREPROCESSING FASE: AQUA

In deze fase vindt het meeste rekenwerk plaats: het ray casten. Vanuit het oogpunt wordt door iedere pixel in het beeldscherm S een straal getrokken. Deze straal wordt getransformeerd naar het datamodel A en vervolgens gesneden met het datavolume. Wanneer de straal het datavolume doorsnijdt worden enige grootheden bepaald om vervolgens opgeslagen te worden in de datafile. Deze grootheden zijn onder andere:

- De hoek tussen de straal en de normaalvector van de verschillende wanden in het model (om de diffuse reflectie te modelleren; zie paragraaf 3.3.3).
- De lengte van de straal (om het diepte-effect te versterken; zie paragraaf 3.3.3).
- De cumulatieve scalaire veldwaarde langs de straal.

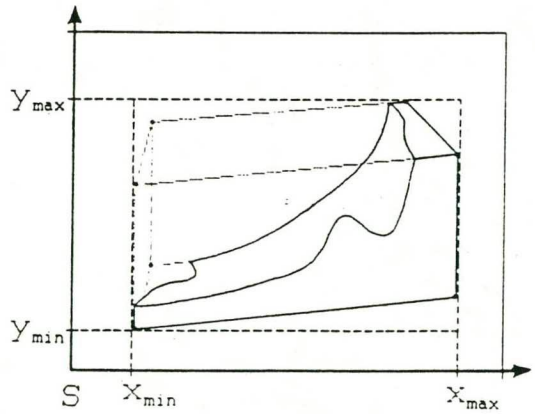
Door het toekennen van ondoorzichtigheden aan alle elementen in het model kunnen alle grootheden in de juiste volgorde en met het juiste gewicht worden gemengd. Zo onstaat transparantie.

Resultaat van deze fase is een datafile, waarin voor iedere pixel grootheden staan die van belang zijn voor de kleurbepaling (in de postprocessing fase).

Achtereenvolgens komen ter sprake: de bounding box, het doorkruisen van het model, het doorlopen van de voxels, het bepalen van de geïntegreerde waarde van het scalaire veld, en de inhoud van de datafile.

DE BOUNDING BOX

Afhankelijk van de ingestelde parameters zullen er veel of weinig stralen zijn die het datavolume in het geheel niet snijden. Om niet onnodig veel stralen te hoeven werpen is daarom gewerkt met een bounding box. Dit is de kleinst mogelijke rechthoek op het beeldscherm waarbinnen het model valt (zie figuur 3.7). Alleen voor die pixels die binnen deze bounding box vallen wordt een straal geworpen.



figuur 3.7 De bounding box.

HET DOORKRUISEN VAN HET MODEL

Wanneer de straal wordt gesneden met het datavolume is het resultaat altijd 0 of 2 snijpunten. Het snijpunt dat het dichtste bij het oogpunt ligt wordt het intreepunt genoemd. Het uittreepunt is dan het punt wat het verst van het oogpunt is verwijderd. Wanneer het intreepunt en het uittreepunt samenvallen (de straal raakt het datavolume) dan wordt het resultaat geforceerd naar 0 snijpunten.

Het vlak waarin het intreepunt zich bevindt heeft een speciale naam gekregen afhankelijk van de positie van de bodem:

- Bodemwand
  Het intreepunt ligt onder de bodem. De straal hoeft nu niet verder meer gevolgd te worden (als erachter nog een scalair veld aanwezig mocht zijn dan is deze toch onzichtbaar voor het oog).
- Waterwand
  Het intreepunt ligt boven de bodem in het water. De straal moet verder gevolgd worden totdat deze in de bodem verdwijnt of het uittreepunt bereikt.

HET VOXEL TRAVERSAL ALGORITME

Om de straal te volgen door het model is het noodzakelijk om te weten welke voxels die straal doorkruist. Het gebruikte algoritme is een incrementeel algoritme ontwikkeld door Amanatides en Woo [Aman87]. Het is een variant op het DDA algoritme [Watt89]] voor het afbeelden van lijnen op een 2D rechthoekig rooster (pixels).

DE BEPALING VAN HET SCALAIRE VELD

Wanneer de straal een scalair veld doorkruist dan wordt de totale "geïntegreerde" waarde van het scalaire veld langs die straal berekend. Dit kan op 2 manieren:

- Door constante benadering.
  Het scalaire veld binnen één voxel wordt als constant beschouwd. De bijdrage per voxel aan de totale waarde is de lengte van het straalinterval door de voxel vermenigvuldigd met de waarde van het scalaire veld van die voxel.
- Door lineaire interpolatie.
  Het verloop van het scalaire veld binnen een voxel wordt lineair verondersteld. Voor elk snijpunt wordt door lineaire interpolatie de waarde van het scalaire veld bepaald. De bijdrage per voxel is nu de lengte van het straalinterval door de voxel vermenigvuldigd met de gemiddelde waarde van het scalaire veld van de twee snijpunten van de straal met de voxel.

Wanneer de waarde van het scalaire veld bepaald wordt door lineaire interpolatie dan komt dit de snelheid niet ten goede.

De kwaliteitverbetering ten opzichte van de methode met een constante benadering is niet groot.

DE INHOUD VAN DE DATAFILE

In de datafile wordt informatie per pixel opgeslagen. Allereerst wordt opgeslagen welke elementen door de straal worden doorkruist. Daarna wordt van al deze elementen van belang zijnde informatie opgeslagen. Om bijvoorbeeld diffuse reflectie te modelleren is het nodig dat de hoek tussen de straal (lichtbron) en de normaalvector van een polygon bekend is. Deze hoek kan dan opgeslagen worden in de datafile. Zo is voor het creëren van een diepte-effect de straallengte nodig (afstand van het oogpunt tot het betreffende snijpunt).

3.3.3 DE POSTPROCESSING FASE: COLOR

In de postprocessing fase worden de gegevens uit de datafile ingelezen en via de door de gebruiker ingestelde parameters wordt de kleur aan de pixels gebonden. Hierbij wordt rekening gehouden met de transparantie van het scalaire veld en de diffuse reflectie als gevolg van lichtinval. Ook is het mogelijk om een extra diepte-effect te creëren door de elementen die verder naar achteren voorkomen in het model iets donkerder weer te geven (depth cueing).

Allereerst worden in het kort twee kleurmodellen besproken. Het aquarium model werkt met het HSV-kleurmodel, terwijl de monitor werkt met het RGB-kleurmodel. Er moet dus een conversie gemaakt worden van HSV naar RGB om de afbeelding correct op de monitor te laten verschijnen. (Deze conversie neemt overigens ruim 20% van de totale tijd van deze fase in beslag!)

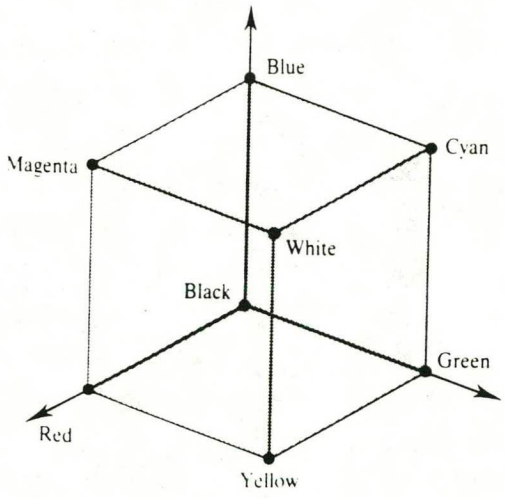
Hierna wordt besproken hoe de transparantie respectievelijk de diffuse reflectie, de ondoorzichtigheid van het scalaire veld en depth cueing is gerealiseerd.

HET RGB-KLEURMODEL

In het RGB-kleurmodel [Hin90], [Hear86], [Watt89] wordt uitgegaan van de drie primaire kleuren rood, groen en blauw. Het is mogelijk om voor elk van deze basiskleuren een bepaalde intensiteit in te stellen om zodoende door menging een kleur te verkrijgen (additieve kleurmenging). Wanneer de rgb-componenten alle op 0 zijn ingesteld onstaat zwart. Wit kan verkregen worden door alle componenten op het maximum in te stellen.

De kleurruimte die zo te verkrijgen is kan gezien worden als een kubus (figuur 3.8).

Het is met dit model mogelijk om kleuren te mengen, maar het is niet goed mogelijk om een gegeven kleur gemakkelijk iets donkerder of lichter in te stellen.

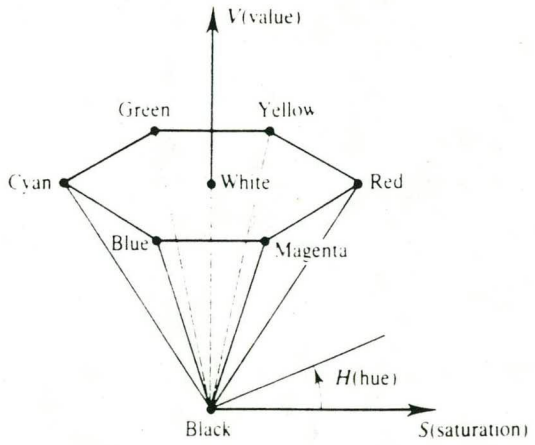


figuur 3.8 De RGB-kleurruimte.

HET HSV-KLEURMODEL

De betekenis van de afzonderlijke componenten van het HSV-kleurmodel [Hin90], [Hear86], [Watt89] is als volgt: de H staat voor Hue, de S voor Saturation en en de V voor Value. Met de H-component kan een bepaalde basiskleur gekozen worden door deze component een waarde te geven tussen 0 en 360 graden. De S-component kan een waarde hebben tussen de 0 en de 1 en geeft de mate van verzadiging met wit. Hoe hoger S, hoe minder wit en hoe zuiverder de kleur. Hoe lager S, hoe fletser de kleur (gaat naar grijs). Door de V-component te variëren tussen de 0 en de 1 kan een lichte respectievelijk donkere tint verkregen worden. V = 0 levert zwart en V = 1 geeft een heldere kleur. De nu verkregen kleurruimte kan worden weergegeven als een kegel (figuur 3.9).

Er is gekozen voor het HSV-kleurmodel omdat dit beter aansluit bij het intuïtief kleurbegrip van de mens. Zo kan de gebruiker op een eenvoudige wijze interactief de gewenste kleur invoeren.



figuur 3.9 De HSV-kleurruimte.

MODELLERING VAN TRANSPARANTIE

Om transparantie te modelleren is het mogelijk om aan ieder element (bodem, wanden, veld) een bepaalde ondoorzichtigheid toe te kennen. De bodem en de bodemwanden zijn volledig ondoorzichtig terwijl de waterwanden en het scalaire veld altijd een ondoorzichtigheid hebben die kleiner dan 1 is.

Het mengen vindt plaats op basis van de elementen die per straal worden doorkruist. Hoe groter de ondoorzichtigheid is van een element, hoe groter de invloed op de totale kleur.

Er wordt net zolang gemengd totdat de cumulatieve ondoorzichtigheid van de elementen gelijk is geworden aan 1.

De achtergrond wordt meegemengd wanneer de straal het model heeft verlaten met een cumulatieve ondoorzichtigheid kleiner dan 1.

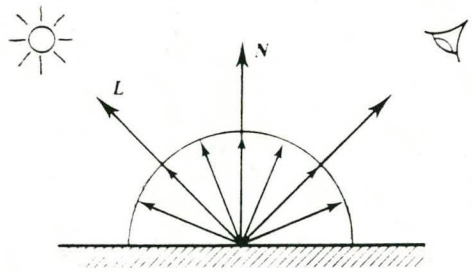
DIFFUSE REFLECTIE

Het licht dat gereflecteerd wordt door een dof oppervlak (wanneer deze belicht wordt door een lichtbron) wordt diffuse reflectie genoemd. De intensiteit van het gereflecteerde licht is vooral afhankelijk van de hoek die de lichtbron maakt met de normaalvector van het oppervlak. Voor meer details zie [Hear86], [Watt89].

In het aquarium model wordt diffuse reflectie gemodelleerd door middel van variatie in de V-component van het HSV-kleurmodel:

$$| N \cdot L | \; * \; \text{element[V]}$$

Met N de normaalvector van het vlak, L de richting van de lichtbron en element[V] de oorspronkelijke V-component van het vlak (zie ook figuur 3.10).



figuur 3.10 Diffuse reflectie.

Diffuse reflectie wordt gebruikt voor de waterwanden, de bodemwand en de bodem.

DE ONDOORZICHTIGHEID VAN HET SCALAIRE VELD

Om er zeker van te zijn dat het scalaire veld zichtbaar gemaakt wordt, is gebruik gemaakt van een genormaliseerde waarde langs de straal. Deze wordt bepaald door de geïntegreerde scalaire waarde langs de hele straal te delen door de maximaal tegengekomen scalaire waarde. De ondoorzichtigheid wordt nu gedefinieerd als een lineaire funktie (eventueel kan ook van een exponentiële funktie gebruik germaakt worden) van de genormaliseerde waarde en loopt ook van 0 tot 1.

Door de H-component van het scalaire veld te variëren, afhankelijk van de grootte van de ondoorzichtigheid, is het ook mogelijk de kleur van het scalaire veld een bepaalde reeks kleuren te laten doorlopen (regenboog effect). Dit effect kan natuurlijk door de gebruiker worden ingesteld.

DEPTH CUEING

Depth cueing is de techniek die verderweg gelegen delen van het model een donkerder kleur geeft dan de delen die dichterbij gelegen zijn. Dit wordt gedaan om het diepte-effect te versterken.

In het aquarium model wordt dit ook gerealiseerd door variatie in de V-component van het HSV-kleurmodel. Dit gebeurt als volgt:

[1 - C * (straallengte - RMIN)/(RMAX - RMIN)] * element[V]

Hierin is RMAX de grootste en RMIN de kleinste tegengekomen straallengte van oogpunt tot element, element[V] de oorspronkelijk V-component en C de door de gebruiker in te stellen depth cueing coëfficiënt waarmee het effect kan worden versterkt of verzwakt.

4 MOGELIJKE VERBETERINGEN

In dit hoofdstuk zullen verschillende verbeteringen en aanvullingen besproken worden die eventueel in het aquarium model toegepast kunnen worden. Deze mogelijkheden zijn gerangschikt in 3 paragrafen. Achtereenvolgens komen aan de orde: allerlei variaties om het aanwezige programma te versnellen zonder de functionaliteit ervan te veranderen, verschillende nieuwe methoden die een 3D scalair veld visualiseren en tenslotte volgen mogelijke toevoegingen aan het aquarium model.

4.1 ALGEMENE VERSNELLINGEN

De nu volgende ideeën zijn bedoeld om het aanwezige programma te versnellen zonder daarbij de functionaliteit te veranderen. De mogelijkheden van het programma blijven dus hetzelfde. Wel kan er een kwaliteitsverandering optreden.

4.1.1 ADAPTIEF RAY CASTEN

In het bestaande programma wordt voor elke pixel die binnen de bounding box valt een straal geworpen. In het algemeen geldt dat twee naast elkaar gelegen pixels enige coherentie vertonen: de kleurwaarden zullen niet veel verschillen. Dit komt doordat de stralen door deze pixels op ongeveer dezelfde plaats door het model heen gaan en dus ongeveer dezelfde elementen zullen doorkruisen. Er zal alleen een behoorlijke kleur-variatie kunnen plaatsvinden wanneer:

- Het scalaire veld sterk verandert.
- Verschillende wanden doorkruist worden.
- Verschillende bodempolygons geraakt worden (reflectie).

Van deze coherentie kan als volgt gebruik gemaakt worden. In eerste instantie worden er alleen stralen geworpen in een rooster van lagere resolutie dan het beeldscherm (bijvoorbeeld één straal voor een blok van 4x4 pixels). De op deze manier verkregen samples kunnen nu met elkaar vergeleken worden. Wanneer blijkt dat de verschillen minimaal zijn dan worden voor alle pixels in het desbetreffende blok een kleur berekend uit de omliggende samples.
Wanneer de verschillen echter een bepaalde drempelwaarde overschrijden, wordt het blok onderverdeeld in 4 kleinere blokken (van elk 2x2 pixels).

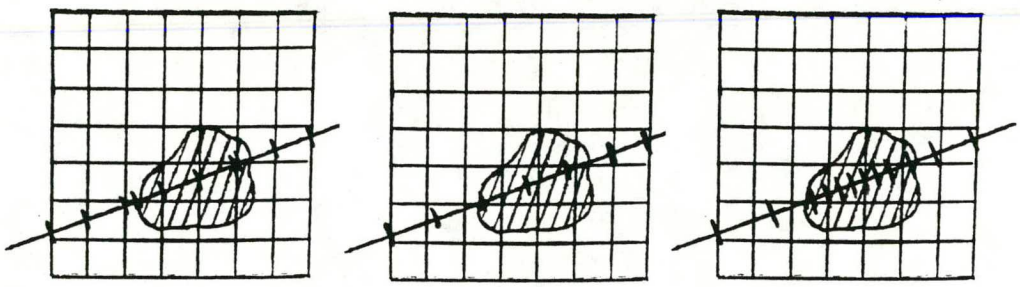
Voor ieder blok met pixels vindt nu de procedure opnieuw plaats enz, totdat er geen opdeling meer nodig is of totdat de maximaal toegestane resolutie bereikt is.

Dit is een methode die vaak toegepast wordt bij ray tracing en hier veel succes heeft geboekt. Bij het gebruik van een goed opdelingscriterium vindt nauwelijks kwaliteitsverandering plaats terwijl er een behoorlijke versnelling optreedt.

Wel is er meer geheugenruimte vereist om de samples te onthouden. Bovendien kunnen er gemakkelijk details worden gemist als gevolg van het samplen op een lagere resolutie.

4.1.2 SAMPLING STRATEGIEEN LANGS DE STRAAL

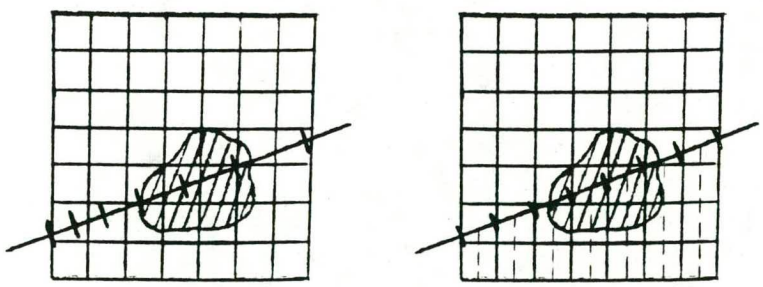
Bij het doorkruisen van de voxels worden samples genomen van het scalaire veld op alle doorsnijdingen met voxelgrenzen (figuur 4.1a). De bijdrage per voxel is de waarde van het scalaire veld vermenigvuldigd met de straallengte door de voxel. Het berekenen van deze doorsnijdingen kost relatief veel tijd. De samples hoeven echter niet op deze doorsnijdingen gekozen te worden; er zijn ook andere mogelijkheden [Wals91].



(a) samples op voxelgrenzen

(b) equidistante samples

(c) sample-afstanden gebaseerd op gradiënt



(d) sample-afstanden gebaseerd op diepte

(e) sample-afstanden gebaseerd op het Bresenham algoritme

figuur 4.1 Sample-strategieën.

GELIJKE SAMPLE-AFSTANDEN

De eerste sample vormt het intreepunt van het model, waarna er met constante intervallen door het model wordt gestapt totdat de bodem of de achterste waterwand wordt gepasseerd. De waarden van het scalaire veld van alle samples kunnen nu opgeteld worden om daarna pas vermenigvuldigd te worden met de totale interval lengte (sample-afstand * aantal samples). Alleen voor de laatste sample moet een correctie plaatsvinden omdat de laatste sample-afstand waarschijnlijk af zal wijken doordat de bodem geraakt wordt (figuur 4.1b).

Door de sample-afstanden gelijk te nemen kan het vóórkomen dat voxels gemist worden wanneer de straal slechts een hoekje van een voxel afsnijdt (zie ook figuur 4.1b). Het resultaat hiervan is een afbeelding van slechtere kwaliteit.

SAMPLE-AFSTANDEN GEBASEERD OP DE GRADIENT

De sample-afstand hangt nu af van de gradiënt van het scalaire veld. Wanneer de gradiënt groot is zal de sample-afstand klein zijn en andersom (figuur 4.1c).

De kans dat een voxel gemist wordt, is op deze manier gereduceerd. Het is echter nog steeds mogelijk om pieken in het scalaire veld te missen doordat de gradiënt geschat moet worden uit al genomen samples.

SAMPLE-AFSTANDEN AFHANKELIJK VAN DE DIEPTE

De sample-afstanden zijn voorin het model klein en worden steeds groter naarmate de straal dieper in het model verdwijnt (figuur 4.1d). Dit komt onder meer voort uit het feit dat voorwerpen onduidelijker worden naarmate ze meer naar achteren staan en omdat de stralen divergeren en dus ook in het beeldvlak steeds verder uit elkaar komen te liggen.

SAMPLE-AFSTANDEN GEBASEERD OP BRESENHAM ALGORITME

Het algoritme is gebaseerd op het 2D algoritme van Bresenham voor het tekenen van lijnen [Hear86]. Het samplen vindt plaats in de richting waar de voxels het snelst doorlopen worden. Elke straal wordt hiertoe verdeeld in gelijke intervallen die begrensd worden door de snijpunten van de straal met het midden van de voxels. In het midden van een interval wordt een sample genomen (figuur 4.1e).

Ook bij deze methode wordt dus equidistant gesampled, maar de afstanden tussen de samples is per straal verschillend.

Als gevolg van deze methode wordt er ongeveer één sample per laag genomen.

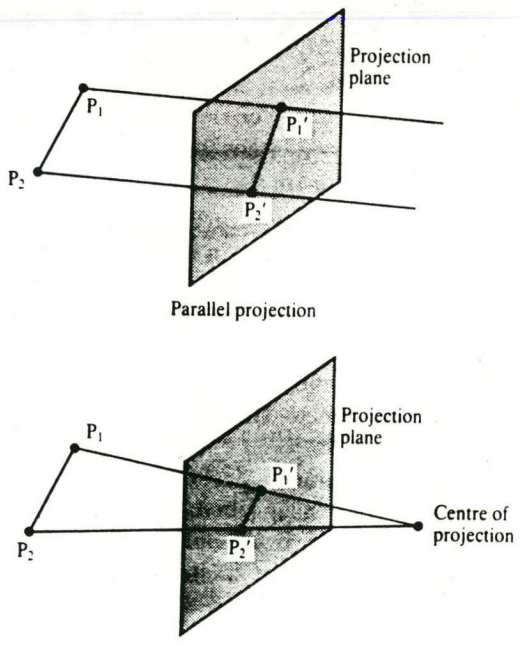
Deze methode levert het beste resultaat vergeleken met de andere genoemde methoden [Wals91].

4.1.3 PARALLEL PROJECTIE

Het huidige programma maakt gebruik van perspectief projectie: vanuit het oogpunt wordt door iedere pixel een straal geworpen. Elke straal heeft zodoende een iets andere richting.

Bij een parallel projectie staan alle stralen orthogonaal op het beeldvlak zodat alle stralen nu precies dezelfde richting hebben en alleen iets van elkaar verschoven zijn (figuur 4.2). Het voordeel hiervan is duidelijk: de vergelijking van elke straal hoeft niet expliciet uitgerekend te worden. Door gebruik te maken van de coherentie tussen de stralen is de vergelijking van een straal eenvoudig incrementeel te berekenen uit de vorige. Ook bij het doorkruisen van de voxels, het snijden van het model en het snijden van de bodem is gebruik te maken van coherentie doordat snijpunten met een constante factor verschoven zijn.

Helaas wordt met een parallel projectie wel een deel van het diepte-effect opgeofferd. Bij een perspectief projectie worden de wat meer naar achter gelegen delen kleiner afgebeeld. Dit is niet meer zo bij een parallel projectie (figuur 4.2).



figuur 4.2 Perspectief versus parallel projectie.

Met een parallel projectie kan dus wel een redelijke snelheidswinst geboekt worden, maar dit gaat ten koste van het diepte-effect. Dit bemoeilijkt het verkrijgen van inzicht in de ligging van het scalaire veld. In hoeverre dit bemoeilijkt wordt is niet precies na te gaan, maar het is de verwachting dat dit een significante rol zal spelen.

4.1.4 INTEGRATIE VAN DE TWEE FASEN

De preprocessing fase en de postprocessing fase kunnen worden samengevoegd. Hierbij ontstaat het volgende probleem. In de postprocessing fase wordt gewerkt met genormaliseerde waarden voor de straallengte en de geïntegreerde waarde van het scalaire veld. Dit is mogelijk doordat het gehele ray casting proces al in de preprocessing fase heeft plaatsgevonden zodat de minimale en maximale waarden al bekend zijn. Hier zal dus een oplossing voor gevonden moeten worden. Een mogelijkheid is om de minimale en maximale waarden te schatten door bijvoorbeeld eerst te ray casten met een (veel) lagere resolutie dan het beeldscherm.

Het voordeel van de integratie is dat er geen datafile meer hoeft te worden aangemaakt. Dit scheelt niet alleen tijd maar ook behoorlijk wat achtergrondgeheugen.

Een nadeel is dat de instellingen van visuele parameters zoals kleur, diepte-effect en ondoorzichtigheden nu vóór het ray casten moet gebeuren. Wanneer er een verandering van deze parameters gewenst is dan zal óók het ray casten opnieuw moeten gebeuren. Naarmate het ray casten sneller wordt zal dit nadeel echter minder zwaar gaan wegen.

4.1.5 VEREENVOUDIGING VAN DE DIFFUSE REFLECTIE

In het huidige programma wordt diffuse lichtreflectie gemodelleerd voor de waterwanden en de bodempolygons. De grootte van deze diffuse reflectie is omgekeerd evenredig met de grootte van de hoek die de invallende lichtstraal maakt met de normaalvector van de waterwand. Hoe groter deze hoek, hoe kleiner de diffuse reflectie (zie 3.3.3).

Doordat in het huidige programma de lichtbron in het oogpunt wordt verondersteld kan de diffuse reflectie verschillen voor punten op dezelfde waterwand. Immers, voor elk punt zal de invalshoek van het licht iets verschillen. Het op deze manier modelleren van diffuse reflectie is hierdoor erg rekenintensief omdat de diffuse reflectie voor elk punt expliciet berekend moet worden.

Het laten samenvallen van de lichtbron en het oogpunt is niet verkeerd, maar wordt gewoonlijk niet gedaan.

Om de berekening van de diffuse reflectie minder rekenintensief te maken komen deze (deel)oplossingen in aanmerking:

1 - de lichtbron verschuiven naar het oneindige.
2 - de waterwanden vervangen door een omlijning.

De eerste oplossing zorgt voor een constante diffuse reflectie voor elk punt op dezelfde polygon. Deze reflectie hoeft nu maar één keer uitgerekend te worden voor iedere polygon waardoor een significante snelheidswinst verkregen kan worden.

Deze oplossing is niet moeilijk te realiseren.

De tweede oplossing is vrij drastisch. Er wordt nu geen diffuse reflectie gemodelleerd voor de waterwanden. In plaats daarvan worden de waterwanden vervangen door de omlijning van de wanden van de bak. Om toch de indruk te krijgen dat er water in de bak bevindt kan de achtergrond op blauw ingesteld worden.

Er onstaat nu een snelheidswinst doordat er helemaal geen diffuse reflectie van de waterwanden meer is waardoor er ook minder gemengd hoeft te worden.

Ook deze oplossing is niet moeilijk te realiseren.

De tweede oplossing is weliswaar de snelste maar produceert een afbeelding zonder diffuse reflectie van de waterwanden. De voorkeur lijkt daarom uit te gaan naar de eerste oplossing omdat deze een redelijke versnelling teweeg zal brengen met een minimaal kwaliteitsoffer.

4.1.6 EFFECTEN REALISEREN IN HET RGB-KLEURMODEL

Binnen het aquarium model wordt gewerkt met twee kleurmodellen; ieder voor een bepaald doel:

- Het RGB-kleurmodel voor het mengen van kleuren.
- Het HSV-kleurmodel voor het realiseren van diffuse reflectie en depth cueing, en het laten invoeren van parameters door de gebruiker.

Doordat er gewerkt wordt met het HSV-kleurmodel is elke keer een conversie nodig van HSV naar RGB. Deze conversie neemt veel tijd in beslag (20-25 procent).

Om deze conversies zoveel mogelijk te elimeren kan diffuse reflectie en depth cueing direct in het RGB-kleurmodel gerealiseerd worden. Voor diffuse reflectie kan dit als volgt gebeuren.

De mate van diffuse reflectie hangt af van de intensiteit van het licht en komt overeen met de hoeveelheid licht wat op een bepaald oppervlak valt. Wanneer een oppervlak schuiner naar de lichtbron toe komt te staan dan zal dit een vermindering van de intensiteit tot gevolg hebben omdat minder lichtstralen het oppervlak raken (zie ook 4.1.5). Dit kan dan gemodelleerd worden door een afname van de drie RGB-componenten volgens:

$$r_{nieuw} = |N \cdot L| * r_{oud}$$
$$g_{nieuw} = |N \cdot L| * g_{oud}$$
$$b_{nieuw} = |N \cdot L| * b_{oud}$$

Hierin is N de normaal van het oppervlak en L de richting van de lichtbron.

Elke component neemt dus procentueel evenveel af. Dit is hetzelfde als een afname van de V-component in het HSV-model zoals dit in het huidige programma gebeurt (zie 3.3.3).

Door het direct in het RGB-kleurmodel realiseren van diffuse reflectie en depth cueing wordt voor elke component dezelfde berekening uitgevoerd. Dit zijn 2 extra berekeningen tov het HSV-kleurmodel. Toch zal er een versnelling optreden omdat het aantal conversies van HSV naar RGB drastisch wordt verminderd.

Volgens [Hall88] is voor elke RGB-component de reflectiecoëfficiënt anders bij een gelijke invalshoek van het licht. Ook is de diffuse reflectie afhankelijk van het reflecterende materiaal. Het implementeren hiervan zal een beter resultaat geven tegen hogere kosten.

De gebruiker zal nog steeds kunnen werken met het HSV-kleurmodel om bepaalde parameters in te stellen. Alleen hiervoor is dan een conversie nodig van HSV naar RGB.

4.2 NIEUWE METHODEN

In deze paragraaf zullen methoden besproken worden die in ingrijpende verandering in het al bestaande programma teweeg zullen brengen en de functionaliteit hiervan zullen beïnvloeden.

4.2.1 PROJECTED TETRAHEDRA

Een methode om direct de scalaire data af te beelden is het projecteren van een volume cel op het scherm (backward mapping). Het afbeelden van zo'n volume cel is echter wel wat moeilijker dan het afbeelden van een polygon.

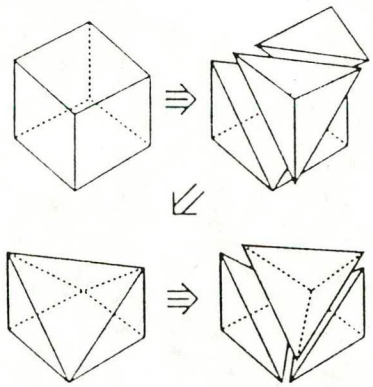
Nu wordt een methode besproken die volume cellen in de vorm van tetrahedra (volume element begrensd door 4 driehoeken) benaderd door transparantie driehoeken [Shir90]. Deze driehoeken kunnen dan vervolgens snel worden afgebeeld.

Deze methode kan in zes stappen worden onderverdeeld: In stap 1 worden eventueel de cellen omgezet naar tetrahedra om ze geschikt te maken voor het algoritme. Stap 2 classificeert de tetrahedra in vier afzonderlijke klassen, om na het projecteren (stap 3) de decompositie van de tetrahedra in driehoeken te vergemakkelijken (stap 4). In stap 5 worden vervolgens de ondoorzichtigheden en de kleuren uitgerekend waarna de driehoeken tenslotte in stap 6 worden afgebeeld.

Elke stap zal nu vervolgens apart worden besproken.

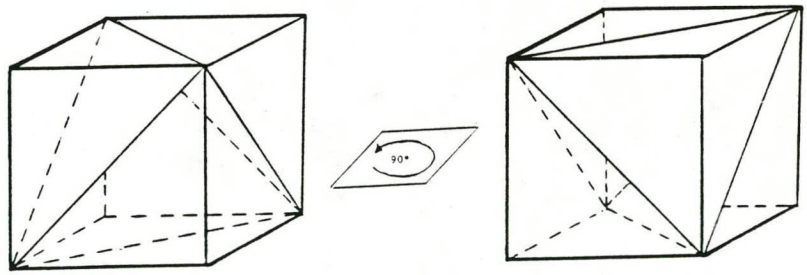
1 DECOMPOSITIE IN TETRAHEDRA

Wanneer de dataset niet uit tetrahedra bestaat dan zal een conversie moeten plaatsvinden. Een rechthoekige element (hier voor het gemak een kubus) kan opgedeeld worden in minimaal 5 tetrahedra (figuur 4.3).



figuur 4.3 Opdeling van een kubus in 5 tetrahedra.

Wanneer alle elementen op dezelfde wijze worden onderverdeeld ontstaat een probleem: één punt kan in twee tetrahedra voorkomen. Zo onstaan oneffenheden in de afbeelding doordat er punten zijn die twee keer afgebeeld worden. Om dit probleem te omzeilen worden alle direct naast elkaar gelegen elementen 90° ten opzichte van elkaar geroteerd (figuur 4.4). Zo onstaat een soort 3D schaakbord patroon met om en om onderverdelingen die ten opzichte van elkaar 90 graden geroteerd zijn.

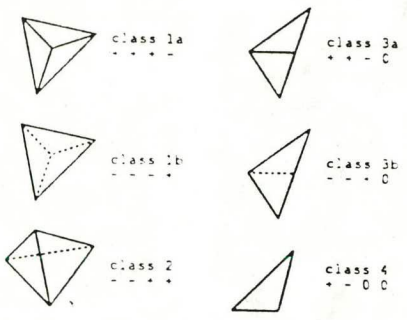


figuur 4.4 Geroteerde opdelingen.

2 CLASSIFICATIE VAN TETRAHEDRA

De verkregen tetrahedra worden geclassificeerd op basis van de geprojecteerde vorm om de latere decompositie in driehoeken te vergemakkelijken.

Door de normaalvectoren van de vier vlakken van het geprojecteerde tetrahedron te vergelijken met de kijkrichting en het oogpunt is het mogelijk om de tetrahedra in vier klassen in te delen op basis van de geprojecteerde vorm (figuur 4.5). De klassen 1 en 3 zijn onderverdeeld in de subklassen a en b omdat de projecties qua vorm hetzelfde zijn, terwijl de tetrahedra verschillende posities in de ruimte innemen.



figuur 4.5 Classificatie van tetrahedra.

De vier vlakken worden geclassificeerd met een -, + of 0. Dit betekent dat de normaalvectoren van zo'n vlak naar achteren wijzen, naar voren wijzen of loodrecht op de kijkrichting staan (dit is dus het teken van het inproduct tussen kijkrichting en normaalvector).

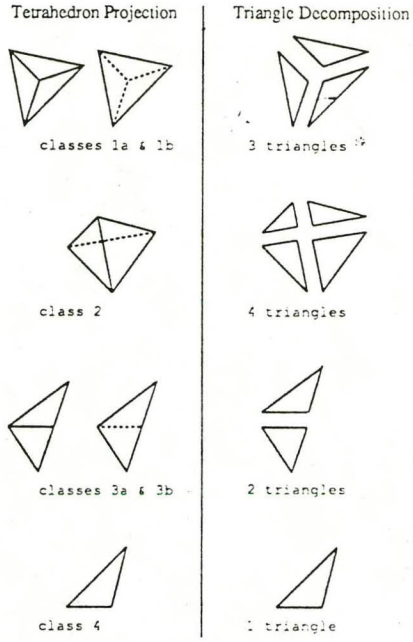
Door nu de aantallen -, + en 0 te tellen en te gebruiken als een index is het mogelijk om een tetrahedron snel in één van de vier klassen in te delen.

3 PROJECTIE

Elk hoekpunt van de tetrahedron wordt nu geprojecteerd op het beeldscherm met behulp van transformatie matrices.

4 DECOMPOSITIE IN DRIEHOEKEN

De tetrahedra kunnen gemakkelijk in driehoeken verdeeld worden op basis van bovengenoemde classificatie, zoals in figuur 4.6 is aangegeven.



figuur 4.6 Decompositie in driehoeken.

5 HET BEPALEN VAN KLEUR EN ONDOORZICHTIGHEID

Voor elke driehoek geldt dat er twee hoekpunten zijn waarop de ondoorzichtigheid α nul is. Als reden is hiervoor aan te geven dat de dikte van de tetrahedron in dit geprojecteerde punt nul is. Onder dikte wordt hier verstaan de lengte van de straal door de tetrahedron.

De ondoorzichtigheid van het derde hoekpunt wordt afgeleid van de dikte.

De afleiding van de kleur in dit derde hoekpunt gebeurt als volgt.

Er wordt gebruik gemaakt van deeltjes met een doorsnede van A_p. Verder wordt er verondersteld dat voor elk]e scalaire waarde σ een dichtheid $N_p(\sigma)$ en kleur $C_p(\sigma)$ bestaat.

Om te bepalen hoe de deeltjes van kleur veranderen langs de straal wordt de kleur opgevat als een functie C(t), met t de afstand langs de straal en oplopend naar het oogpunt toe. De verandering in kleur als de gevolg van een kleine toename Δt is dan de volgende:

$$C(t+\Delta t) = (1-N_p(t)A_p\Delta t)C(t) + N_p(t)A_p\Delta tC_p(t)$$

Dit resulteert in de volgende differentiaal vergelijking wanneer Δt geschreven wordt als d/dt:

$$dC(t)/dt + N_p(t)A_p[C(t) - C_p(t)] = 0$$

Deze vergelijking kan niet analytisch worden opgelost. Echter, als verondersteld wordt dat:

- $t_0 = t$ en $t_1 = t + \Delta t$
- de kleur van het deeltje constant is: C_0
- er al een waarde is voor $C(t_0)$
- N_p lineair variërt tussen N_0 (op t_0) en N_1 (op t_1)

dan kan de ondoorzichtigheid α en kleur $C(t_1)$ geschreven worden als:

$$\alpha = A_p(t_1 - t_0) * (N_0 + N_1)/2$$
$$C(t_1) = \alpha * (C_p(t_0) + C_p(t_1))/2 + (1 - \alpha)C(t_0)$$

Door gebruik te maken van de vergelijkingen wordt voor het dikste punt (het derde hoekpunt) de ondoorzichtheid en de kleur berekend.

6 HET AFBEELDEN VAN DE DRIEHOEKEN

Nu de ondoorzichtigheid en de kleur bekend is voor elk hoekpunt van elke driehoek kunnen de driehoeken afgebeeld worden in een back-to-front volgorde waarin de kleurwaarde C en ondoorzichtigheid α lineair worden geïnterpoleerd tussen de hoekpunten. Elke pixel verandert nu als volgt:

$$C_{new} = \alpha*C + (1-\alpha)*C_{old}$$

Volgens [Shir90] zijn de stappen 1 tot en met 5 'heel snel' gerealiseerd. De laatste stap (stap 6) is niet gemeten omdat deze uitgevoerd werd door de hardware.

Wanneer het niet in hardware uitgevoerd kan worden is het nog maar de vraag of het algoritme sneller is dan ray casting. Er worden immers vele driehoeken gegenereerd! Het zou alleen lonend kunnen zijn voor een dataset met lage resolutie.

Er kunnen eventueel geometrische primitieven aan het model worden toegevoegd, door deze primitieven mee te nemen in het sorteren van de driehoeken in een back-to-front volgorde. Als gevolg hiervan ontstaan fouten in de afbeelding wanneer een primitief een volume-element doorsnijdt. Dit zou pleiten voor een dataset met hoge resolutie. De effectiviteit wordt hierdoor wel verminderd. Het algoritme levert pas een behoorlijke snelheidswinst op wanneer de transparante driehoeken meerdere pixels gaan beslaan. Dit kan alleen wanneer de resolutie van het datamodel kleiner is dan de resolutie van de afbeelding.

Ondanks de lage resolutie van het datamodel wordt de toepasbaarheid op het aquarium model sterk verminderd doordat er vele bodempolygons aanwezig zijn.

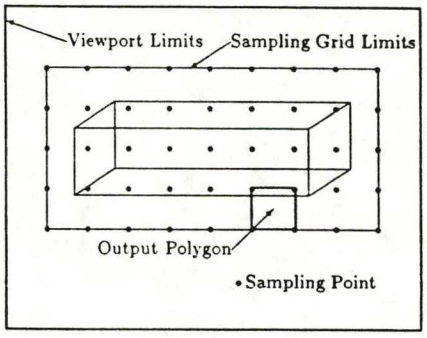
4.2.2 RAY CASTING MET POLYGONALE UITVOER

Bij de volgende methode wordt ray casting gecombineerd met het afbeelden van polygons. Hiertoe wordt ray casting toegepast op bepaalde pixels van het beeldscherm. De op deze manier verkregen kleursamples worden gegroepeerd tot vierkante polygons en worden hierna met behulp van Gouraud shading [Watt89] afgebeeld [Meye90].

Er zullen vervolgens verschillende methoden worden beproken om deze kleursamples te nemen.

SAMPLEN MET EEN VASTE RESOLUTIE

Er wordt een sample genomen voor elke n^{de} pixel (figuur 4.7).



figuur 4.7 Samplen met een vaste resolutie.

Zodoende worden er maar $1/n^2$ keer het aantal samples genomen in vergelijking met het ray casten voor de volle resolutie (1 sample per pixel). De keuze van n (de sample frequentie) is hierbij erg belangrijk. Hoe hoger de sample frequentie hoe beter het model wordt benaderd. Wanneer de sample frequentie echter te laag wordt gekozen ontstaat een afbeelding met een vloeiend verloop waarop weinig detail te zien is.

De instelling van de sample frequentie zal ook per model verschillend zijn. Zo zal de sample frequentie van een model waarin het scalaire veld sterke gradiënten vertoont hoger moeten zijn dan bij een model met kleine gradiënten.

Deze sample methode is niet geschikt voor het maken van afbeelding met veel details, maar kan zeker wel gebruikt worden. De gebruiker kan op deze manier snel een afbeelding verkrijgen voor het eventueel bijstellen van enige parameters. Later kan dan gekozen worden voor een kwalitatief betere methode.

INCREMENTEEL SAMPLEN

Nu wordt er ook weer gesampled met een vaste resolutie. Allereerst wordt er een afbeelding gegenereerd door te samplen met een lage resolutie. De afbeelding wordt vervolgens verfijnd door de sample-frequentie kleiner te maken. Dit gaat net zo lang door tot de hoogste sample frequentie bereikt is of totdat de gebruiker een parameter verandert. Zodoende onstaat er dus een afbeelding waarop langzamerhand steeds meer detail te zien is.

Deze methode is natuurlijk langzamer dan de vorige wanneer de hoogste sample frequentie gelijk wordt gekozen aan de sample frequentie van de vorige methode.

Het voordeel van deze methode is dat de gebruiker direct al een resultaat te zien krijgt als gevolg van de lage sample frequentie.

SAMPLEN GEBASEERD OP DE GRADIENT

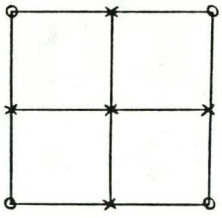
Zoals al gezegd is, kan de sample frequentie zo goed mogelijk gekozen worden door deze aan te passen aan de gradiënt van het scalaire veld. Het is duidelijk dat er geen hoge sample frequentie nodig is in die delen van het scalaire veld waar de gradiënt klein is. De sample frequentie hoeft dus alleen hoog te zijn bij een grote gradiënt.

Wanneer begonnen wordt bij een initiële sample frequentie dan kan deze plaatselijk bijgesteld worden afhankelijk van de gradiënt van het scalaire veld. Dit gebeurt op de volgende manier.

Van elke sample is de kleurwaarde in het RGB-kleurmodel bekend. De gradiënt wordt bepaald door de kleurwaarde van vier samples (van één vierkante polygon) met elkaar te vergelijken. Dit gebeurt door het absolute verschil te nemen tussen de rgb-waarden van de samples:

$$\text{gradiënt} = |r_2 - r_1| + |g_2 - g_1| + |b_2 - b_1|$$

Wanneer deze gradiënt een bepaalde tolerantie overschrijdt, vindt er plaatselijk een verfijning plaats. De polygon wordt dan in vieren onderverdeeld door vijf nieuwe samples te nemen; vier in het midden van de zijden en één in het midden van de polygon (figuur 4.8). Dit gaat zo door totdat er geen verfijningen meer plaats vinden of totdat de maximale sample frequentie bereikt is.



figuur 4.8 Opdeling van een polygon.

Er treedt volgens [Meye90] een versnelling op in de orde van 10^2 zonder veel kwaliteitsverlies bij een goede instelling van de tolerantie. Dit komt hoofdzakelijk doordat er in het scalaire veld vaak maar één wolk voorkomt waar de gradiënten hoog zijn.

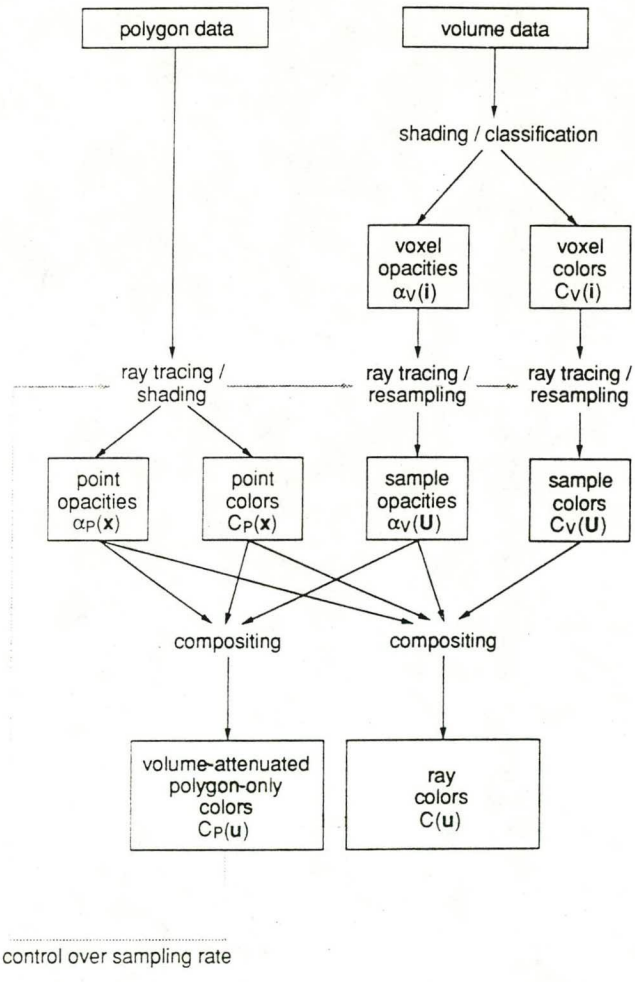
De laatste methode lijkt veel op het adaptief ray casten (4.1.1). Doordat de polygons door de hardware afgebeeld worden met behulp van Gouraud shading is het echter mogelijk om een lagere sample frequentie te nemen voor een afbeelding van dezelfde kwaliteit.

4.2.3 HYBRIDE RAY CASTING

Bij hybride ray casting wordt ray casting apart toegepast voor het scalaire veld en voor de polygons [Levo89]. Het ray casten door het scalaire veld wordt zodanig aangepast dat adaptief samplen goed mogelijk is.

Het is ook mogelijk om de polygons en het scalaire veld apart af te beelden om ze daarna in de goede diepte volgorde samen te voegen. Een nadeel hiervan is dat er geen meerdere transparante vlakken achter elkaar mogen voorkomen wanneer er een Z-buffer algoritme gebruikt wordt. Bij de nu volgende methode kan dit wel.

Het vereenvoudigde algoritme werkt als volgt (figuur 4.9).



figuur 4.9 Overzicht hybride ray tracing.

Voor elke voxel worden de ondoorzichtheid en de kleur bepaald, resulterend in $\alpha_V(\mathbf{i})$ en $C_V(\mathbf{i})$ respectievelijk. Hierin is **i** een vector die de voxel indiceert.

Nu worden er (parallelle) stralen door de voxels gestuurd voor iedere pixel **u** = (u,v). Langs iedere straal worden samples **U** = (u,v,w) genomen op gelijke afstanden. Dit levert de ondoorzichtheid $\alpha_V(\mathbf{U})$ en de kleur $C_V(\mathbf{U})$ voor iedere sample.
Tevens worden de stralen doorsneden met alle polygons wat resulteert in een ondoorzichtigheid $\alpha_P(\mathbf{x})$ en kleur $C_P(\mathbf{x})$ voor ieder snijpunt **x** = (x,y,z).

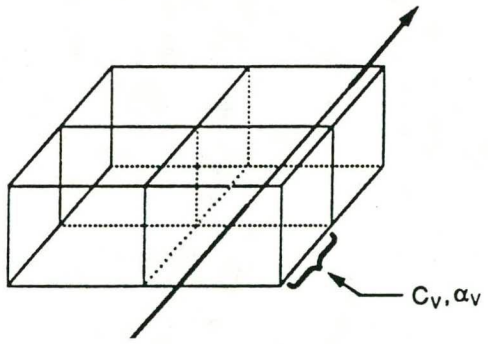
Tot slot worden de ondoorzichtigheden en de kleuren van de samples en de polygon intersecties naar diepte samengevoegd, wat resulteert in een pixelkleur C(**u**) voor iedere pixel **u**.

ADAPTIEF SAMPLEN

Om adaptief te samplen worden de kleuren van omliggende pixels met elkaar vergeleken. Om te voorkomen dat er onnodige verfijningen optreden bij lage frequentie veranderingen in het scalaire veld wordt er in de laatste fase ook de kleur $C_P(\mathbf{u})$ berekend. Dit is de kleurbijdrage van de polygons aan de totale kleur C(**u**). Wanneer C(**u**) verandert en $C_P(\mathbf{u})$ niet dan duidt dit op een lage frequentie verandering in het scalaire veld en hoeft er niet verfijnd te worden.

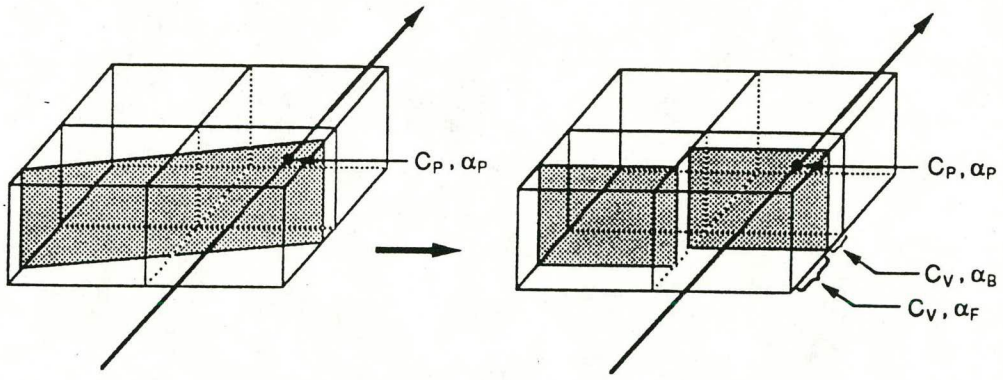
POLYGON-VOLUME INTERSECTIES

Doordat de parallelle stralen equidistant worden gesampled (door middel van trilineaire interpolatie) wordt het volume opgedeeld in 'slabs' met een lengte die overeenkomt met de sample-afstand en een hoogte en een breedte die overeenkomt met de ruimte tussen twee stralen (figuur 4.10).



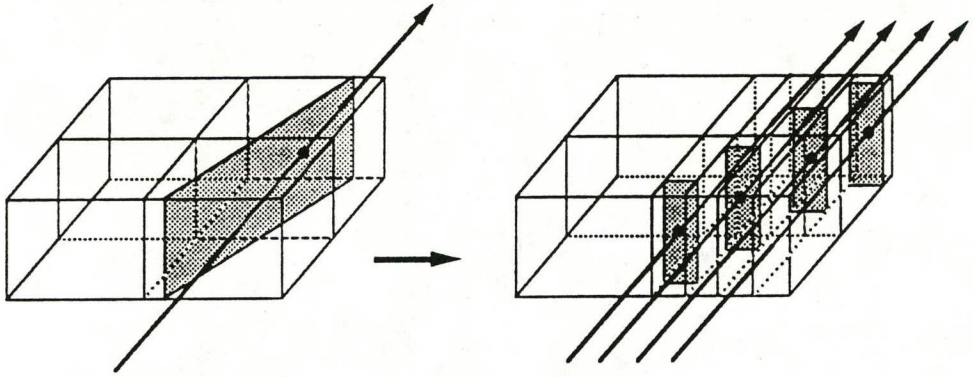
figuur 4.10 Slabs.

Wanneer een polygon deze slabs doorsnijdt dan kunnen de volume bijdragen voor de polygon en achter de polygon worden bepaald. Dit is gemakkelijk voor de polygons die alleen de zijkanten raken (figuur 4.11).



figuur 4.11 Doorsnijding polygon met zijkanten.

Wanneer een polygon de voor- of achterkant raakt dan snijdt de straal deze polygon maar een keer, terwijl er contributies zijn voor meerdere slabs. Door ook hier te verfijnen kan dit probleem worden gereduceerd totdat de stukken polygon ook alleen maar de zijkanten raken (figuur 4.12).



figuur 4.12 Doorsnijding polygon met voor- en/of achterkant.

Volgens Levoy levert dit een versnelling op in de orde van 10^2. De voorbeelden zijn echter geproduceerd met weinig polygons. Het aquarium model werkt met veel bodempolygons zodat de intersecties met willekeurige polygons veel tijd gaat kosten.

Een voordeel van deze methode is dat nu wel gewerkt kan worden met een willekeurig aantal (transparante) polygons. Dit geeft een behoorlijke verbetering in de functionaliteit. De transparante waterwanden kunnen dan ook op deze manier worden meegenomen.

Het is goed mogelijk om behalve polygons ook andere elementen zoals parametrische patches in het model op te nemen.

4.3 TOEVOEGINGEN AAN HET MODEL

In deze paragraaf worden enige ideeën besproken die eventueel aan het aquarium model toegevoegd kunnen worden.

4.3.1 HET INSTELLEN VAN EEN LICHTBRON

In het aanwezige programma wordt de lichtbron in het oogpunt verondersteld. Er kan wat meer flexibiliteit verkregen worden door de lichtbron te laten instellen door de gebruiker. Op deze manier kan er een beter diepte-inzicht verkregen worden door een juiste plaatsing van de lichtbron.

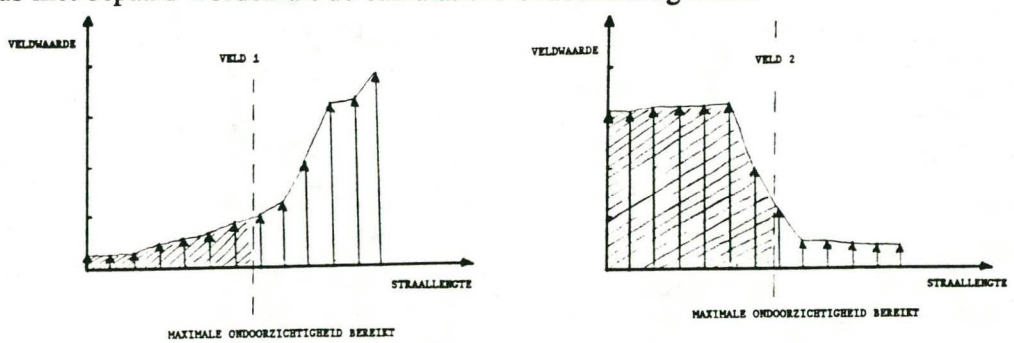
Het instellen van een lichtbron zal waarschijnlijk alleen bestaan uit het invoeren van een richting zodat de lichtbron in het oneindige wordt geplaatst (zie 4.1.5).

4.3.2 VISUALISEREN VAN MEERDERE SCALAIRE VELDEN

Met het aanwezige programma is het slechts mogelijk om één scalair veld weer te geven. Langs de hele straal wordt dan de cumulatieve veldwaarde bepaald om hieruit de ondoorzichtigheid te berekenen. Hierna wordt de kleur van het scalaire veld gemengd met de totale kleur voor zover de cumulatieve ondoorzichtigheid dit toelaat (mag niet groter worden dan 1.0). Zie ook paragraaf 3.3.3.

Het moet mogelijk zijn om te kiezen voor het weergeven van meerdere scalaire velden tegelijkertijd. Elke voxel bevat dan meerdere grootheden.

Nu kan echter niet simpelweg de cumulatieve veldwaarden van de scalaire velden worden bepaald om die te gebruiken bij het mengen. Er is immers niet bekend in hoeverre de verschillende velden meegemengd moeten worden wanneer de cumulatieve ondoorzichtigheid het maximum heeft overschreden. In figuur 4.13 zijn de veldwaarden van twee scalaire velden aangegeven. De cumulatieve ondoorzichtigheden voor beide velden langs de gehele straal kunnen gelijk zijn terwijl de invloed van het tweede veld vele malen groter is dan het eerste veld. Dit had net zo goed andersom kunnen zijn! De invloed van de scalaire velden op de totale kleur kunnen dus niet bepaald worden uit de cumulatieve ondoorzichtigheden.



figuur 4.13 Invloed van twee scalaire velden langs de straal.

Een oplossing hiervoor kan zijn om te stoppen met het volgen van de straal zodra een voxel wordt bereikt waarbij de som van de ondoorzichtigheden het maximum overschrijdt. Deze laatste voxel kan dan verdeeld worden.
Hierna is verder samplen overbodig.

Door het visualiseren van meerdere scalaire velden wordt de functionaliteit van het programma verhoogd. Dit gaat helaas wel ten koste van de snelheid en de benodigde veranderingen in de code zijn groot (mengen van de velden, andere datastructuur enz.).

4.3.3 MEERDERE AANZICHTEN PER SCHERM

Voor een goede interpretatie van het scalaire veld zal het model waarschijnlijk bekeken moeten worden onder verschillende aanzichten. Wanneer deze aanzichten achter elkaar worden berekend en afgebeeld zal het voor de gebruiker moeilijk zijn om de aanzichten in gedachten te combineren. Door de verschillende aanzichten op één scherm weer te geven is dit probleem grotendeels verholpen.

Hiervoor is een window-viewport transformatie nodig om het juiste gebied op het scherm te selecteren.

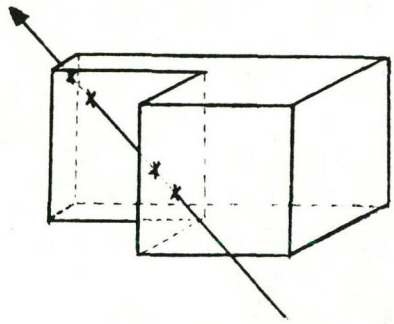
4.3.4 DATASELECTIE

Door bepaalde voxels uit de dataset op non-actief te zetten kan er als het ware binnen in het model gekeken worden. De gebruiker kan zo een beter inzicht krijgen in de ligging van het scalaire veld.

De veranderingen in het programma als gevolg hiervan zijn groot:

- In de datastructuur moet een extra bit worden opgenomen om aan te geven of een voxel wel of niet meedoet in de berekening van het scalaire veld.
- Het bepalen van het intree- en uittreepunt is complexer.
- Er kunnen nu niet maximaal twee waterwanden doorsneden worden maar in principe veel meer (figuur 4.14).
- Het gebruik van verschillende sample-strategieën kan worden bemoeilijkt doordat het binnenkomen en het verlaten van het model steeds weer voor kan komen.

De snelheid van het programma zal minder worden. Toch is dit een aantrekkelijke optie om een beter inzicht te verkrijgen, vooral als er een goede interface naar de gebruiker bestaat om de juiste voxels te selecteren.



figuur 4.14 Dataselectie: doorsnijden van het model.

Er kan ook gekozen worden voor een iets eenvoudigere dataselectie door alleen bepaalde lagen te selecteren. Op deze manier blijft het af te beelden deel van het model altijd een convexe rechthoek, zodat deze optie vrij eenvoudig te implementeren is.

4.3.5 ANIMATIE

Om de inzicht in de ligging van het scalaire veld te verbeteren kan er een voorwerp door het scalaire veld bewogen worden. Wanneer de baan van het voorwerp bekend is dan kan de gebruiker een beter idee krijgen van het verloop van het scalaire veld. Het voorwerp zal immers steeds meer verduisterd worden door het scalaire veld als de baan van het voorwerp naar achteren is gericht.

Wanneer gekozen wordt voor een klein voorwerp dan hoeft niet de hele afbeelding opnieuw te worden gegenereerd. Alleen een klein deel van de afbeelding waarin het voorwerp zich bevind moet opnieuw uitgerekend worden voor elke frame van de animatie. Als dit niet interactief kan gebeuren dan kunnen de benodigde delen van alle afbeeldingen van de animatie opgeslagen worden op achtergrondgeheugen, om daarna achter elkaar te worden afgebeeld.

Wanneer de snelheid en het geheugen van het grafische werkstation het toelaat kan ook een animatie gemaakt worden van het model terwijl het een verandering van oogpunt ondergaat (oogpunts animatie). Nu moet echter wel de hele afbeelding opnieuw worden uitgerekend voor elke frame van de animatie. Dit lijkt voorlopig ook niet haalbaar.

4.3.6 HET MODELLEREN VAN WATER

In de huidige situatie wordt het water volledig doorzichtig verondersteld. Door voor het water een kleine constante ondoorzichtigheid in te stellen wordt het effect versterkt dat er water in het model aanwezig is.

Het implementeren hiervan wordt bemoeilijkt doordat per straal de cumulatieve scalaire veldwaarde wordt bepaald. Het mengen van het water kan dus niet achteraf plaatsvinden omdat er de kans bestaat dat dan de cumulatieve ondoorzichtigheid de maximale waarde al heeft bereikt zodat mengen geen zin meer heeft. Er zal dan per sample gemengd moeten worden waardoor de snelheid af zal nemen.

Dit probleem is nauw verwant aan dat van het visualiseren van meerdere scalaire velden. Het water kan immers ook opgevat worden als een scalair veld (met een constante dichtheid en dus een lineair verloop langs de straal).

Het is zeer de vraag of de voordelen van het modelleren van water in het model opwegen tegen de nadelen. Wanneer echter gekozen wordt voor het visualiseren van meerdere scalaire velden dan is het modelleren van water eenvoudig te realiseren.

4.3.7 HET GENEREREN VAN ISO-OPPERVLAKKEN

Een andere mogelijkheid is het genereren van iso-oppervlakken uit het scalaire veld. Hiervoor kan ook de ray casting methode benut worden door het scalaire veld te drempelen op de constante waarde van het iso-oppervlak.

Er kan ook gekozen worden voor de marching cubes [Stol90] methode. Deze methode heeft echter niets met ray casting te maken.

De implementatie van deze optie door middel van ray casting is vrij eenvoudig te realiseren.

4.3.8 SLICING

Dit is het doorsnijden van het model met een door de gebruiker gekozen plat vlak (zie paragraaf 2.3). Op dit vlak kunnen nu enkele karakteristieken van het scalaire veld worden weergegeven, zoals bijvoorbeeld isolijnen.

Voor het implementeren hiervan is een geheel andere methode nodig. Dit is niet zo moeilijk te implementeren maar het heeft niets te maken met ray casting en is alleen een uitbreiding van de mogelijkheden.

5 CONCLUSIES EN PUNTEN VOOR VERDER ONDERZOEK

Tijdens het schrijven van dit taakverslag zijn al enige ideeën verwerkt in nieuwe versies van het aquarium model [Wals91]. In al deze versies is adaptief ray casting succesvol geïmplementeerd en leverde een behoorlijke versnelling op.

Een hybride versie scan converteert de bodempolygons wat resulteert in een dieptebuffer met voor iedere pixel informatie over de dichtst bij gelegen bodempolygon. Op deze manier is voor elke pixel (en straal) nu direct bekend welke bodempolygon wordt doorsneden. Zodoende is er een kleine versnelling gerealiseerd en is de functionaliteit verhoogd.

Tevens werden er verschillende sample-strategieën uitgeprobeerd. Het beste resultaat vormde de methode gebaseerd op het 3D Bresenham algoritme (zie 4.1.2). Er trad een versnelling op zonder daarbij veel aan kwaliteit te moeten inleveren.

Alle versnellingen zijn gerealiseerd in de preprocessing fase. Nu blijkt dat de postprocessing fase inmiddels langer duurt dan de preprocessing fase. De fasering van het programma heeft dus geen nut meer. Er moet daarom gestreefd worden naar een geïntegreerd programma waarin zeker nog enige optimalisaties kunnen plaatsvinden:

- pure code optimalisaties.
- het zoveel mogelijk elimineren van de kleurconversie.

Het geïntegreerde programma laat tevens de weg open voor onder andere het genereren van iso-oppervlakken en het toepassen van slicing.

Alle methoden worden voornamelijk beoordeeld op snelheid en kwaliteit. Het streven naar een methode die de snelste is en een afbeelding oplevert met een perfecte kwaliteit, is tot mislukken gedoemd. Wanneer een methode een grote snelheidswinst oplevert dan zal dit meestal resulteren in een afbeelding van mindere kwaliteit. Dit komt doordat het werkelijke model dan onnauwkeuriger wordt benaderd.

Toch kan er wel gekozen worden voor een combinatie van snelheid en kwaliteit. Het programma zal dan twee methoden bevatten: één om zo snel mogelijk een afbeelding te verkrijgen en één om een afbeelding te genereren met een zo hoog mogelijke kwaliteit. Zodoende kan de gebruiker de snelle methode gebruiken voor het instellen van parameters en kwalitatieve methode voor het genereren van de eindafbeelding.

De snelle methode kan dan gebaseerd zijn op:

- parallel projectie.
- eliminatie van waterwanden.
- een snelle sample strategie.
- constante diffuse reflectie.

Voor de meest interessante toevoegingen aan het model zou ik zeker willen kiezen voor:

- Het instellen van een lichtbron.
  Hierdoor worden de mogelijkheden van het programma vergroot en heeft de gebruiker meer vrijheid.
- Het visualiseren van meerdere scalaire velden tegelijkertijd.
  Uitbreiding van het aantal mogelijkheden.
- Het genereren van meerdere aanzichten op één scherm.
  Hierdoor is het voor de gebruiker makkelijker om inzicht te krijgen in de ligging van het scalaire veld.
- Het toepassen van dataselectie.
  Verbetering van het inzicht.
- Het maken van een animatie.
  Verbetering van het inzicht.

Om al deze verbeteringen en toevoegingen te implementeren zal het nodig zijn om een geheel nieuwe opzet te maken van het programma, waarin van het begin af aan rekening gehouden wordt met de (eventuele) vernieuwingen.

VERKLARENDE WOORDENLIJST

algoritme	methode van oplossen.
aliasing	ongewilde misvormingen in de afbeelding ten gevolge van de discretisering van een continu signaal.
equidistant	van gelijke lengte.
gradiënt	grootte en richting van de toename/afname van een grootheid.
isolijn	lijn waarop de waarde van een 2D scalair veld constant is.
iso-oppervlak	oppervlak waarop de waarde van een 3D scalair veld constant is.
pixel	kleinste deel van het beeldscherm wat apart een kleur valt te geven (beeldpunt).
polygon	vlakke veelhoek.
pseudocode	een formulering van een algoritme: half in een programmeertaal en half in gewone taal.
scalair veld	een gebied met waarden die alleen een grootte hebben en geen richting (dit itt tot een vector veld).
tetrahedron	volume-element begrensd door 4 driehoeken.
vector veld	een gebied met grootheden die zowel een grootte als een richting hebben.
voxel	kleinste deel van een opgedeeld volume (meestal een kubus).

BRONVERMELDING VAN OVER-GENOMEN FIGUREN

figuur 1.1
Kok J. de, Rijkswaterstaat, dienst getijdewateren.

figuur 2.1, 2.7, 2.8, 2.10
Stolk J, 'Afbeeldingstechnieken voor wetenschappelijke visualisatie', Taakverslag september 1990, Technische Universiteit Delft.

figuur 2.2, 2.3
UNIRAS: 'User guide and reference manual AGL/Contours'

figuur 2.4, 2.5, 2.6
Lith M.S.P., 'Inventarisatie van afbeeldingstechnieken voor visualisatie van stromingen', Taakverslag februari 1990, Rijksuniversiteit Leiden, Waterloopkundig Laboratorium.

figuur 3.1
Diverse auteurs, *Bundel Computer Graphics - a213 deel I*, Samenstelling van de Vakgroep Technische Informatica, Leerstoel Computer Graphics, Technische Universiteit Delft, mei 1989.

figuur 3.2, 3.3, 3.8, 3.9, 3.10, 4.2
Watt A., *Fundamentalsof Three-Dimensional Computer Graphics*, Addison-Wesley 1989.

figuur 3.4, 3.5, 3.6, 3.7
Hin A.J.S., 'Een aquarium model voor de visualisatie van 3D scalaire velden door ray-casting', Afstudeerverslag januari 1990, Technische Universiteit Delft.

figuur 4.3, 4.5, 4.6
Shirley P., A. Tuchman, 'A Polygon Approximation to Direct Scalar Volume Rendering', Computer Graphics 24 (1990), 5, p. 63-69.

figuur 4.7
Meyers R.J., M.B. Stephenson, 'Ray traced Scalar Fields with Shaded Polygonal Output', IEEE Computer 23 (1990), 10, p. 263-273.

figuur 4.9, 4.10, 4.11, 4.12
Levoy M., 'Volume Rendering: A Hybrid Ray Tracer for Rendering Polygonal and Volume Data', IEEE Computer Graphics & Applications 10 (1989), 2, p. 33-40.

LIJST VAN GEBRUIKTE AFKORTINGEN

2D	2-dimensionaal
3D	3-dimensionaal
BSP	Binary Space Partitioning
BTF	Back-To-Front
DGW	Dienst getijdewateren
FTB	Front-To-Back
RGB	Red, Green and Blue
RWS	Rijkswaterstaat
HSV	Hue, Saturation and Value

LITERATUURLIJST

[Aman87] Amanatides, J., A. Woo, 'A Fast Voxel Traversal Algorithm for Ray Tracing', Eurographics 1987, p. 3-10.

[Dive89] Diverse auteurs, *Bundel Computer Graphics - a213 deel I*, Samenstelling van de Vakgroep Technische Informatica, Leerstoel Computer Graphics, Technische Universiteit Delft, mei 1989.

[Garg86] Gargantine I., T.R. Walsh, O.L. Wu, 'Viewing Transformations of Voxel-Based Objects via Linear Octrees', IEEE Computer Graphics & Applications 6 (1986), 10, p. 12-21.

[Hall88] Hall R., *Illumination and color in Computer Generated Imagery*, Springer-Verlag 1988.

[Hear86] Hearn D., M.P. Baker, *Computer Graphics*, Prentice Hall 1986.

[Hin89] Hin A.J.S., 'Inventarisatie van visualisatie technieken voor de eindige elementen methode', Taakverslag april 1989, Technische Universiteit Delft.

[Hin90] Hin A.J.S., 'Een aquarium model voor de visualisatie van 3D scalaire velden door ray-casting', Afstudeerverslag januari 1990, Technische Universiteit Delft.

[Levo88] Levoy M., 'Volume Rendering: Display of Surfaces from Volume Data', IEEE Computer Graphics & Applications 8 (1988), 3, p. 29-37.

[Levo89] Levoy M., 'Volume Rendering: A Hybrid Ray Tracer for Rendering Polygonal and Volume Data', IEEE Computer Graphics & Applications 10 (1989), 2, p. 33-40.

[Lith90] Lith M.S.P., 'Inventarisatie van afbeeldingstechnieken voor visualisatie van stromingen', Taakverslag februari 1990, Rijksuniversiteit Leiden, Waterloopkundig Laboratorium.

[Meye90] Meyers R.J., M.B. Stephenson, 'Ray Traced Scalar Fields With Shaded Polygonal Output', IEEE Computer 23 (1990), 10, p. 263-273.

[Pate90] Pater G. de, 'Een inventarisatie van visualisatie technieken voor 3-dimensionale-, stationaire stromingen', Taakverslag januari 1990, Technische Universiteit Delft.

[Shir90] Shirley P., A. Tuchman, 'A Polygon Approximation to Direct Scalar Volume Rendering', Computer Graphics 24 (1990), 5, p. 63-69.

[Stol90] Stolk J., 'Afbeeldingstechnieken voor wetenschappelijke visualisatie', Taakverslag september 1990, Technische Universiteit Delft.

[Wals91] Walsum T. van, A.J.S. Hin, Versloot J., F.H. Post, 'Improving Speed and Functionality in Volume Ray Casting', Artikel gepresenteerd tijdens de 'Second Eurographics Workshop on Visualization in Scientific Computing', Delft, 22-24 april 1991.

[Watt89] Watt A., *Fundamentals of Three-Dimensional Computer Graphics*, Addison-Wesley 1989.