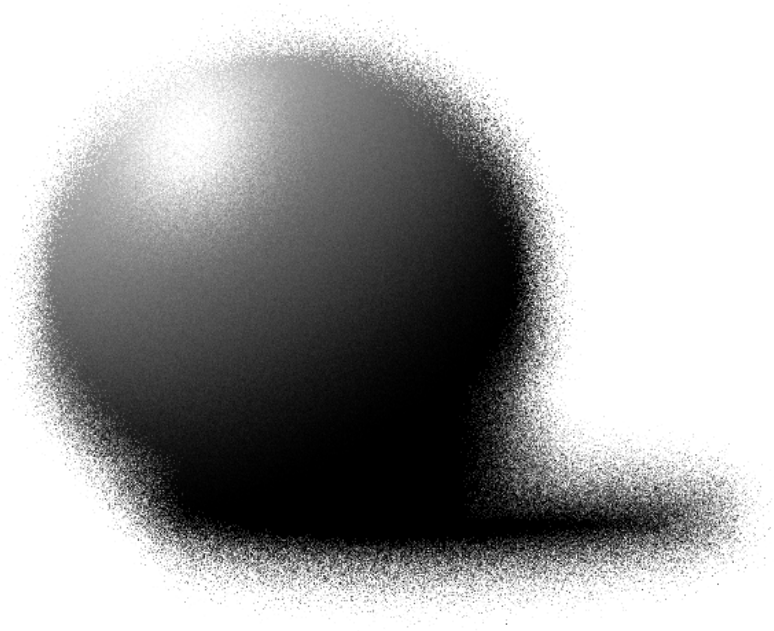
Ray Tracing



Gabe Dijkstra Robert Hensing

Profielwerkstuk 6 VWO
Mentor: dhr. J. M. Debets

Christiaan Huygens College

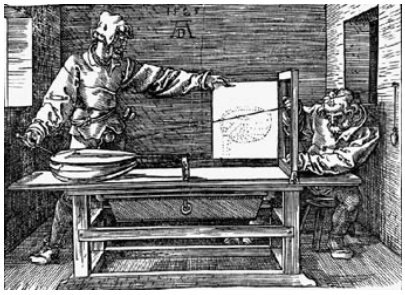
5 april 2008

Inhoudsopgave

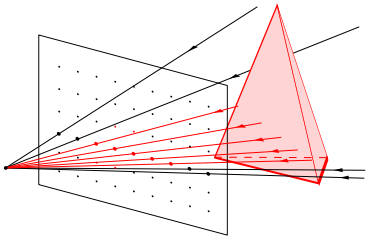
Verklaring van de voorpagina

De rendering op de voorpagina is van een bol met zowel de diffuse reflectiviteitscoëfficiënt als de speculatieve op 0,5. Onder en achter de bol bevinden zich twee witte overbelichte vlakken. De glans is gegeven door $(\mathbf{r} \cdot \mathbf{v})^{15}$. De bol weerspiegelt niet. De scène is negenvoudig stochastisch bemonsterd met een afwijking van 70 beeldpunten in zowel de horizontale als verticale richting.
Wat dit alles betekent zal duidelijk worden in het werkstuk.

Inleiding



Figuur 1 Perspectiefonderzoek tijdens de Renaissance.



Figuur 2 Kleurbepaling van beeldpunten volgens de ray tracing techniek.

Een belangrijk onderdeel van computer graphics is rendering: de overkoepelende term voor bewerkingen die van een beschrijving van driedimensionale vormen een tweedimensionale afbeelding maken. *Ray tracing* is de oudste renderingmethode. De ideeën die hierbij gebruikt worden zijn gedeeltelijk al in de Renaissance bedacht, toen men onderzoek deed naar perspectief (zie figuur 1).

Ray tracing doet precies wat haar naam aangeeft: het volgt lichtstralen. Een ray tracing programma bekijkt hoe de lichtstralen zich in een scène gedragen. Als een straal de camera (een gegeven punt) bereikt, krijgt het corresponderende beeldpunt de kleur van die straal (zie figuur 2).

Het is tevens de meest gebruikte methode voor niet-realtime[1] toepassingen, zoals in films van PIXAR[2] Het is namelijk een natuurkundig verantwoorde techniek en kan daarom natuurgetrouwe beelden leveren. Het probleem is wel dat het een berekeningsintensieve methode is. Je zult daarom nog geen interactieve, realtime toepassingen, zoals videospellen, tegenkomen die ray tracing gebruiken: de computers zijn op dit moment hiervoor te langzaam. Er zijn echter al wel initiatieven die realtime ray tracing proberen te bereiken met behulp van gespecialiseerde chips op videokaarten. Onder andere INTEL, dat een belangrijke rol speelt in de processorenmarkt, is bezig met een dergelijk project.

In het werkstuk hebben we veel mogelijk onbekende termen verduidelijkt met voetnoten. Tevens willen we u erop wijzen dat er achterin een korte lijst van begrippen en een index te vinden zijn.

[1] niet-realtime: waarbij het niet nodig is dat de gebruiker binnen enkele milliseconden het resultaat ziet van zijn interactie met de computer

[2] PIXAR is een bedrijf dat films maakt, uitsluitend met computers.

Notatie

Wiskundige notatie

a, b, c	reële getallen (cursieve kleine letters)
P, Q	punten (cursieve hoofdletters)
$\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$	vectoren (vetgedrukte kleine letters)
$\mathbf{M}$	matrix (vetgedrukte hoofdletter)
θ, ϕ	hoeken (Griekse kleine letters)
$\lvert\mathbf{M}\rvert$	determinant van matrix $\mathbf{M}$
$\lvert\mathbf{v}\rvert$	lengte van vector $\mathbf{v}$
$\triangle ABC$	driehoek gevormd door de punten A, B, C
$\max(a, b)$	a als $a > b$ en b als $b > a$
$+$, $-$, $/$, $*$	som, verschil, deling, product
$\cdot$	inproduct ($\mathbf{a} \cdot \mathbf{b}$ moet vaak gelezen worden als $\max(\mathbf{a} \cdot \mathbf{b}, 0)$)
$\times$	uitproduct
$\circ$	functie compositie operator ($(f \circ g)(x) = g(f(x))$)

Pseudocode

Om de algoritmen[3] in dit werkstuk te beschijven, gebruiken wij pseudocode. Dit is een notatie voor algoritmen die wat weg heeft van een heuse programmeertaal, maar gewone taal gebruikt om de stappen te beschrijven. Hierdoor is pseudocode onmogelijk direct uit te voeren door een computer, maar het is uiterst geschikt om de werking van een algoritme uit te leggen.

Pseudocode wordt regel voor regel doorlopen. De volgende sleutelwoorden geven aan waar dat niet gebeurt. Het sleutelwoord geldt dan voor alleen de regel erna.

als *waarheidsuitdrukking*
: Pseudocode die alleen wordt uitgevoerd als de waarheidsuitdrukking (bijvoorbeeld een ongelijkheid) 'waar' als resultaat heeft.

voor elke *naam* **in** *uitdrukking* **doe**
: Pseudocode die steeds opnieuw wordt uitgevoerd voor alle onderdelen van *uitdrukking*. Voor elk onderdeel wordt de pseudocode uitgevoerd met *naam* om dat onderdeel aan te duiden.

Met de sleutelwoorden **begin** en **eind** kunnen constructies zoals **als** toegepast worden op meerdere regels. Om te verduidelijken wat van toepassing is op welke regels, zijn ondergeschikte regels verder ingesprongen dan normaal.

Een 'programma' dat het vuil buiten zet zou er zo uit kunnen zien:

```
als niet regen
  voor elke prullenbak in huis
    begin
      haal afvalzak uit prullenbak
      deponeer afvalzak in vuilnisbak
    eind
```

Zo kunnen op een overzichtelijke manier ook ingewikkeldere programma's beschreven worden. Verder gebruiken wij de notatie $A := B$ wanneer A de waarde van B aan moet nemen.

[3] Een algoritme is een eindige reeks instructies om te beschrijven hoe uit een begintoestand een bepaald doel bereikt wordt.

Hoofdstuk 1

Probleemstelling

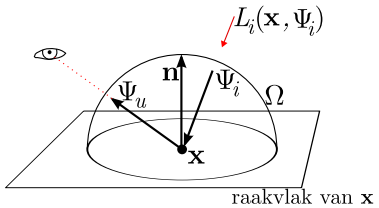
Het algemene probleem dat men met rendering oplost is het maken van een tweedimensionale afbeelding, aan de hand van een (numerieke) bijschrijving, ofwel de scène. Wij gaan dit probleem oplossen door lichttransport in de gegeven scène met behulp van ray tracing te modelleren.

1.1 Rendervergelijking

De gebruikelijke wijze om dit te modelleren is met behulp van de rendervergelijking. Deze vergelijking beschrijft hoe licht een punt op een oppervlak verlaat aan de hand van de eigenschappen van het oppervlak en de hoeveelheden licht die daar binnenkomen. Omdat deze vergelijking een prominente rol speelt in de deelonderwerpen, hebben wij deze opgenomen in de probleemstelling. De hierboven genoemde hoofdstelling is dus breder dan alleen het oplossen van de vergelijking die we in deze paragraaf behandelen.

Aan de linkerkant van de vergelijking staat de hoeveelheid straling die vanuit eén punt, in eén richting het oppervlak verlaat. Deze hoeveelheid straling is de som van de straling die het oppervlak op dat punt zelf uitzendt en de straling die wordt gereflecteerd. De rendervergelijking wordt als volgt weergegeven:

$$
\begin{aligned}
L_u(\mathbf{x}, \Psi_u) &= L_e(\mathbf{x}, \Psi_u) + L_r(\mathbf{x}, \Psi_u) \\
L_r(\mathbf{x}, \Psi_u) &= \int_\Omega f_r(\mathbf{x}, \Psi_i, \Psi_u) L_i(\mathbf{x}, \Psi_i)(\Psi_i \cdot \mathbf{n}) d\omega_i
\end{aligned}
$$



Figuur 1.1 Variabelen van de rendervergelijking ruimtelijk weergegeven

L_u: *u*itgaande straling van punt $\mathbf{x}$ in de richting van de straal Ψ_u

L_e: *e*missie, de straling die het oppervlak zelf uitzendt

L_r: de gereflecteerde straling

Ω: alle mogelijke binnenkomende richtingen, als het ware een bol (voor elk punt op de bol een richting naar het middenpunt)

f_r: de twee-richtings reflectie functie, ofwel BDRF (*bi-directional reflectance function*)

L_i: de b*i*nnenkomende straling

n: de normaal van het oppervlak op punt **x**

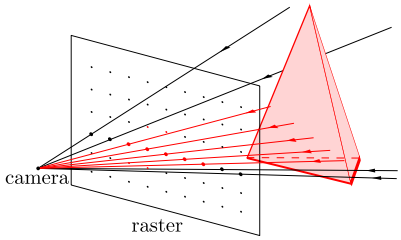
$\Psi_i \cdot \mathbf{n}$: 'verdunning' van binnenkomend licht als gevolg van de invalshoek. Dit leggen we uit in hoofdstuk 3 (Belichting).

De twee-richtings reflectie functie heet 'twee-richtings' omdat deze functie iets zegt over de mate waarin reflectie plaatsvindt tussen twee richtingen, namelijk de invalsrichting en uitvalsrichting. Voor een glimmend materiaal zullen de uitkomsten van deze functie er dus anders uitzien dan voor een mat materiaal.

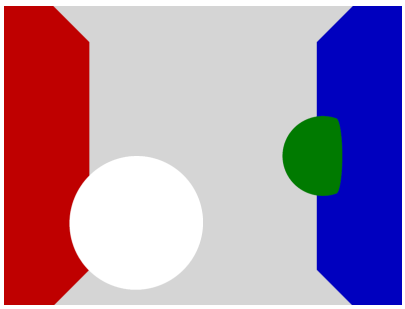
In de hoofdstukken 3 en 4 (Belichting en Recursieve belichting) zullen we de rendervergelijking benaderen.

Hoofdstuk 2

Simpele ray tracing



Figuur 2.1 Het idee van ray tracing



Figuur 2.2 Resultaat van een simpele ray trace

We zullen nu niet meteen ingaan op de rendervergelijking, maar we zullen laten zien dat het ray tracing algoritme werkt. We zullen aannemen dat de kleur van het object direct verantwoordelijk is voor de kleur van het beeldpunt.

De raytracer 'schiet' voor elk punt van het raster van beeldpunten een *straal* vanuit de 'camera' door het punt. Vervolgens kijkt de raytracer of deze straal een object uit de scène snijdt. Als dat het geval is, krijgt het beeldpunt de kleur van het gesneden object.

2.1 Het ray tracing algoritme

In pseudocode ziet het ray tracing algoritme er als volgt uit:

Programma 2.1 Het ray tracing algoritme

```
voor elk beeldpunt in raster doe
  begin
    bepaal straal vanuit camera richting raster
    bepaal welk object een snijpunt dichtstbij camera heeft

    als object gesneden
      beeldpunt := kleur van gesneden object
    anders
      beeldpunt := achtergrondkleur van scène
  einde
```

2.2 Stralen

Een straal is een lijn die aan eén kant eindig is. In het geval van een simpele raytracer, zijn alle stralen aan de kant van de camera eindig, aangezien we alleen de stralen gebruiken direct die nodig zijn voor het beeld.

In de volgende vergelijking is de wiskundige definitie van de straal gegeven: $\mathbf{o}$ is de oorsprong van straal R, dus de eindige kant. $\mathbf{d}$ is de richting en t is een variabele.

$$R(t) = \mathbf{o} + \mathbf{d}t \qquad (t \geq 0) \tag{2.1}$$

2.2.1 Eenheidsvectoren

De vector $\mathbf{d}$ is een eenheidsvector. Dat wil zeggen dat de vector een lengte heeft van 1. Van elke vector met een lengte > 0 kan een eenheidsvector gemaakt worden, door alle componenten door de lengte van de vector te delen. Deze operatie wordt ook 'normaliseren' genoemd.

Het berekenen van de lengte van een driedimensionale vector kost een computer veel tijd, omdat er naast drie kwadraten en twee optellingen een wortel getrokken moet worden, wat erg lang duurt.

Voor veel berekeningen is alleen de richting van een vector relevant. Vaak is het dan ook efficiënter om van te voren vectoren te normaliseren, zodat in de daarop volgende berekeningen aangenomen kan worden dat de lengte van de vector 1 is. Zo wordt niet herhaaldelijk dezelfde langzame berekening uitgevoerd.

2.3 Snijpuntsberekeningen

Snijpuntsberekeningen zijn essentieel voor een ray tracing programma. Er moet immers gekeken worden of de straal een object snijdt. Een object is een oppervlak, dat meestal impliciet gedefinieerd is. Dat wil zeggen dat de verzameling van punten van het oppervlak gevormd wordt door de verzameling van oplossingen voor de vergelijking $f(\mathbf{x}) = 0$

De voorwaarde voor een bol is dan dus:

$$f(\mathbf{x}) = (\mathbf{x} - \mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}) - r^2 = 0 \tag{2.2}$$

$\mathbf{c}$: middenpunt

r: radius[1]

Om te zien of er op straal R een punt $\mathbf{x}$ ligt dat voldoet aan de vergelijking moeten we kijken naar de definitie van een straal (vergelijking 2.1). Deze functie vullen we in in $f(\mathbf{x})$ en de vergelijking wordt dan:

$$F(t) = (R \circ f)(t) = 0 \tag{2.3}$$

Voor de oplossing(en) van 2.3 moet uiteraard gelden $t \geq 0$. Het is ook mogelijk dat er twee oplossingen zijn. Alleen de kleinste t, de t het dichtst bij $\mathbf{o}$, is interessant: die is zichtbaar.

[1] In dit werkstuk zullen we altijd *radius* gebruiken in plaats van *straal*, omdat *straal* uiteraard al gebruikt wordt voor een ander begrip.

Hoofdstuk 3

Belichting

In dit hoofdstuk zullen we de rendervergelijking te benaderen.

$$L_u(\mathbf{x}, \Psi_u) = L_e(\mathbf{x}, \Psi_u) + \int_\Omega f_r(\mathbf{x}, \Psi_i, \Psi_u) L_i(\mathbf{x}, \Psi_i)(\Psi_i \cdot \mathbf{n}) d\omega_i$$

De integraal van de rendervergelijking (L_r) wordt versimpeld door deze op te delen in een *direct* en een *indirect* deel en ze apart te benaderen.

Direct licht is licht dat nadat het de bron heeft verlaten maar eén oppervlak raakt, voordat het wordt waargenomen. Direct licht wordt dus maar eenmaal weerkaatst. Indirect licht daarentegen is licht dat meer dan eén keer is gereflecteerd. In dit hoofdstuk zullen wij ons beperken tot direct licht.

3.1 Belichtingsmodellen

We gaan het directe deel benaderen. In ray tracing worden L (luminantie) en I (intensiteit) door elkaar gebruikt. We houden geen rekening met eenheden, dus maakt het uiteindelijk weinig uit. De intensiteit die wordt beschreven is die op de camera, dus de kleur van het beeldpunt. Deze beschrijven we met een belichtingsvergelijking.

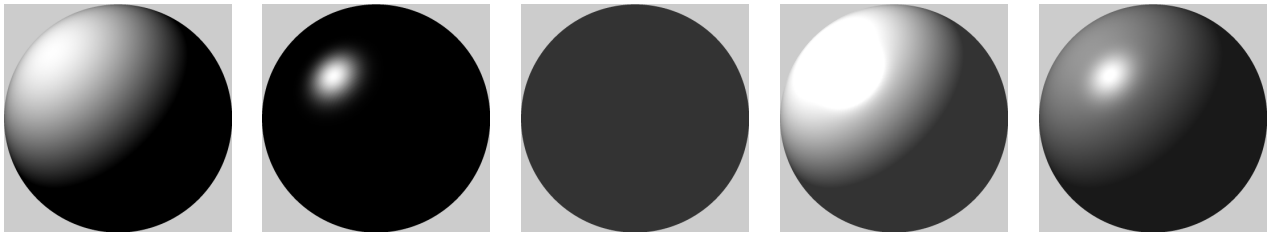
Het belichtingsmodel dat we in programma 2.1 hebben gebruikt, houdt alleen rekening met omgevingslicht: aangenomen wordt dat elk punt in de scène evenveel licht ontvangt. Het is een benadering van het indirecte licht in de scène. Met directe lichtbronnen wordt dus geen rekening gehouden:

$$I = k_a i_a \tag{3.1}$$

k_a: de omgevingslichtwaarde van het object: de kleur van het object zelf.

i_a: de kleur van de lichtbron: de hoeveelheid omgevingslicht.[1]

[1] Het subscript 'a' komt van het Engelse woord 'ambient', wat 'omgevings-' betekent.



(a) Diffuse weerkaatsing

(b) Speculatieve weerkaatsing

(c) Omgevingslicht

(d) Som van (a), (b) en (c), overbelicht

(e) Zelfde som, na correctie met factor 0,5

Figuur 3.1 De onderdelen van het Phong belichtingsmodel

3.2 Belichtingsmodel van Phong

In 1973 heeft Bui Tuong Phong een belichtingsmodel ontwikkeld dat schaduwen op de voorwerpen zelf (een puntlicht belicht een object niet van alle kanten even fel) en glimlichten ('weerspiegeling' van het puntlicht op het object) nabootst: diffuse weerkaatsing en speculatieve weerkaatsing.[2]

3.2.1 Diffuse weerkaatsing

Matte oppervlakken, zoals krijt, vertonen diffuse weerkaatsing, ook wel Lambertiaanse weerkaatsing genoemd. Ongeacht van welke hoek je naar een punt op een dergelijk materiaal kijkt, komt er evenveel licht naar het oog. In alle richtingen wordt het licht dus met dezelfde intensiteit weerkaatst. De gereflecteerde lichthoeveelheid is dus evenredig met de hoeveelheid licht die binnenvalt op dat punt.

Hoeveel licht er binnenvalt is echter afhankelijk van de hoek tussen de normaal en de lichtbron. Voor schuine vlakken geldt namelijk dat de verhouding van hoeveelheid licht per oppervlakte kleiner wordt.

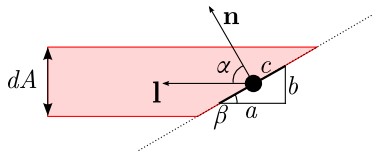
Hoe licht een oppervlak op een bepaalde plaats is, hangt af van de hoek tussen de normaal en de richting naar de lichtbron:

$$I_d = k_d i_d \cos\alpha \tag{3.2}$$

i_d: *d*iffuse intensiteit van de lichtbron

k_d: *d*iffuse intensiteit van het object zelf

[2]Wij gebruiken het woord 'weerkaatsing' voor reflecties van lichtstralen vanuit bijvoorbeeld de lichtbron via het oppervlak naar de camera. 'Weerspiegeling' gebruiken wij voor reflecties waarbij de hoek van inval precies gelijk is aan de hoek van uitval ten opzichte van de normaal en waarbij er dus een beeld zichtbaar is wanneer er naar het oppervlak wordt gekeken.



Figuur 3.2 Verdunning van licht $a \parallel \mathbf{l}$, $b \perp \mathbf{l}$, $c \perp \mathbf{n}$

α: hoek tussen normaal en vector van snijpunt richting lichtbron

Om beter te begrijpen hoe $\cos\alpha$ tot stand komt nemen we aan dat de stralen en punten waar we aan rekenen een klein oppervlak hebben, lim $dA \to 0$.

Wanneer de lichtstraal tegen een 'punt' komt dat zich onder een hoek bevindt, wordt dit punt niet door de hele bundel geraakt, maar door een deel ervan. De intensiteit van het licht dat gereflecteerd kan worden is dus de intensiteit van de lichtbron vermenigvuldigd met een factor kleiner dan 1.

In het volgende bewijs wordt aangetoond dat deze factor (k) gelijk is aan de verhouding $b : dA$ en aan $\cos\alpha$:

$$\begin{aligned} c &= dA \\ \sin\beta &= \frac{b}{c} \\ b &= dA\sin\beta \\ k &= \frac{b}{dA} = \sin\beta \\ &= \sin(\tfrac{1}{2}\pi - \alpha) \qquad \text{(rechte hoek en z-figuur)} \\ &= \cos\alpha \end{aligned}$$

α hoeven we niet te berekenen, omdat we meteen $\cos\alpha$ kunnen berekenen. Immers: $\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\alpha$. Als $\mathbf{a}$ en $\mathbf{b}$ genormaliseerd zijn, is $|\mathbf{a}||\mathbf{b}|$ gelijk aan 1, en heb je meteen $\cos\alpha$.

De hoeveelheid diffuse weerkaatsing van een oppervlak is dus:

$$I_d = k_d i_d(\mathbf{l}\cdot\mathbf{n}) \tag{3.3}$$

3.2.2 Speculatieve weerkaatsing

Een glanzend oppervlak vertoont 'speculatieve' weerkaatsing. Deze weerkaatsing zorgt bijvoorbeeld voor de glimlichten op auto's die in de zon staan, de felle reflecties van zonlicht op het natte asfalt als de zon al wat lager staat, etc. Voor dit soort weerkaatsing is de richting tot de waarnemer dus wel relevant. Nat asfalt is bijvoorbeeld alleen hinderlijk als men tegen de zon in rijdt.

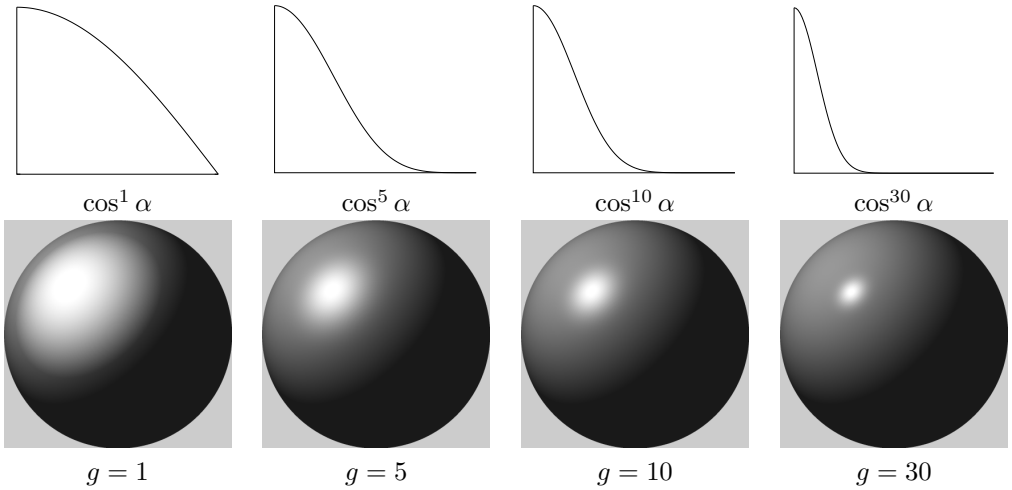
Speculatieve reflecties zijn reflecties waarbij de (bekende) regel van invals- en uitvalshoeken geldt. Voor de meeste materialen geldt echter dat de normaal erg varieert per punt, omdat het materiaal toch wat ruw is. Er is dus ook speculatieve reflectie als de hoek van inval en uitval niet precies gelijk zijn.

Wanneer de reflectiehoek (hoek tussen normaal en $\mathbf{r}$) en de waarnemingshoek (hoek tussen normaal en $\mathbf{v}$) gelijk zijn, is de speculatieve reflectie volledig. Als de afwijking groter wordt, moet de reflectie factor snel afnemen. Het Phong model benadert deze eigenschap door een macht g (> 1) te nemen van $\mathbf{r}\cdot\mathbf{v}$. Als deze

macht hoger is, daalt de cosinus eerder: we krijgen dan een kleiner glimlichtje met een duidelijkere overgang, waardoor het glimmender lijkt.

De vergelijking is dus:

$$I_s = k_s i_s (\mathbf{r} \cdot \mathbf{v})^g \tag{3.4}$$



Figuur 3.3 Hoe hoger g, des te glimmender.

3.2.3 Alles bijelkaar

Als we deze berekend hebben, moeten we ze bij elkaar optellen. Het sommatieteken geeft aan dat we de diffuse en speculatieve intensiteit ten opzichte van elke lichtbron moeten berekenen en al deze resultaten moeten optellen:

$$I = I_a + \sum (I_d + I_s) \tag{3.5}$$

Zoals aan figuur 3.1(d) te zien is, is het resultaat overbelicht aangezien we een bereik van $[0, 1]$ verwachten. Voor dit voorbeeld hebben we dit opgelost door het totaal met een factor 0,5 te vermenigvuldigen.

Niet elk materiaal heeft echter dezelfde verhoudingen speculatieve en diffuse reflectie. Daarvoor zorgen de object-specifieke constanten k_a, k_d en k_s. Om een scène in kleur te belichten, dienen al deze berekeningen driemaal uitgevoerd te worden, voor rood, groen en blauw. Voor elke kleur kunnen er dan andere waarden voor i_a, i_d, i_s, k_a, k_d en k_s gekozen worden. De k constanten zijn dan gelijkwaardig aan de kleur van het object, i gelijkwaardig aan de kleuren van het licht.

Hoofdstuk 4

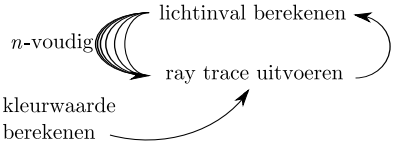
Recursieve belichting

Tot nu toe hebben we alleen maar gekeken naar de lichtinvalsrichtingen (Ψ_i) van een lichtbron naar het oppervlak. Volgens de rendervergelijking moeten we elke Ψ_i bekijken:

$$L_u(\mathbf{x}, \Psi_u) = L_e(\mathbf{x}, \Psi_u) + \int_\Omega f_r(\mathbf{x}, \Psi_i, \Psi_u) L_i(\mathbf{x}, \Psi_i)(\Psi_i \cdot \mathbf{n}) d\omega_i$$

4.1 Lichtinval opnieuw bekeken

Elke Ψ_i bekijken is onmogelijk: het zijn er oneindig veel. Daar komt bij dat voor elke inkomende straal vaak weer een andere straal berekend moet worden. De hoeveelheid inkomend licht op een punt $\mathbf{x}_A$ wordt bijvoorbeeld deels bepaald door de hoeveelheid licht die vanuit $\mathbf{x}_B$ wordt gereflecteerd in de richting van $\mathbf{x}_A$, oftewel: $L_i(\mathbf{x}_A, \Psi_i) = L_u(\mathbf{x}_B, \Psi_u)$. Ook die hoeveelheid licht kan weer worden beïnvloed door een ander punt, net zo lang tot we bij een object met $L_e > 0$ komen: de rendervergelijking is recursief.



Figuur 4.1 Voor de ene berekening is de andere nodig en vice versa.

Deze recursie kan hoog oplopen, dus moeten we deze begrenzen. Na een aantal m recursies nemen we aan dat de lichtinval gelijk is aan de achtergrondkleur van de scène. Als we dan bijvoorbeeld $m = 6$ nemen en het aantal richtingen $n = 40$ hebben we $40^6 = 4{,}096 \cdot 10^9$ ray traces per beeldpunt. De complexiteit van deze aanpak is namelijk $O(n^m)$.[1] Met huidige computers zou het dus veel te lang duren voordat er een beeld ontstaat.

Veel van deze berekeningen zullen overbodig zijn omdat ze al eerder zijn uitgevoerd of omdat ze een onwaarneembaar kleine bijdrage leveren aan het uiteindelijke beeld. Om toch correcte lichtinvalsberekeningen te kunnen doen zal er dus een andere aanpak nodig zijn.

[1] In de informatica wordt de notatie $O(x)$ gebruikt om een indicatie van de ergst mogelijke tijdsduur aan te geven. De functie O is dan als het ware verantwoordelijk voor alle afwijkingen van de inschatting x. Door deze functie te gebruiken is het duidelijk dat x een indicatie is en dat er niet gerekend kan worden met x of de uitkomst van $O(x)$.

4.2 Recursie beperken

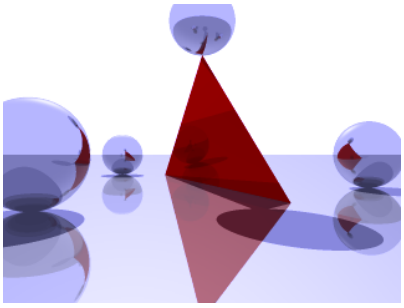
Een betere manier om dit probleem op te lossen is om de resultaten van de ray traces op te slaan, zodat deze hergebruikt kunnen worden. Dit is echter een moeilijk probleem om op te lossen, dus zullen wij een eenvoudigere aanpak bespreken.

We beperken onze blik door de volgende aannamen to doen:

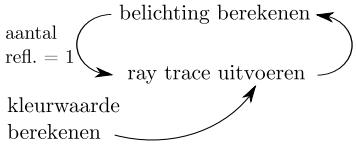
- Licht kan alleen verhinderd worden door een voorwerp tussen het snijpunt en de lichtbron waaraan gerekend wordt.
- De enige weerkaatsing van licht van oppervlakken (niet van lichtbronnen) op andere oppervlakken is weerspiegeling.

Licht van indirecte lichtbronnen wordt dus niet in de berekening meegenomen. Normaalgesproken belicht een rode wand die met een wit licht beschenen wordt ook een nabije witte wand: die witte wand lijkt dan ook rood.

De duur van de berekening is dan nog maar afhankelijk van het aantal directe lichtbronnen en de recursiediepte. De complexiteit is in het ergste geval $O(m)$. Recursiediepte m is namelijk een maximum. Als we tijdens de berekening met een niet-weerspiegelend oppervlak te maken krijgen, houdt de recursie op.



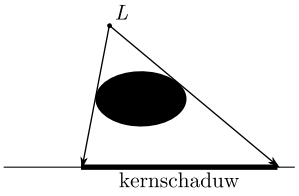
Figuur 4.2 Resultaat van een ray trace met spiegeling en schaduw.



Figuur 4.3 De recursie is minder omvangrijk geworden.

4.3 Schaduwen

Een schaduw ontstaat als er een voorwerp tussen een lichtbron en een oppervlak staat. We kunnen schaduwen nabootsen door een straal te schieten van uit het punt op het oppervlak, waaraan wij rekenen, in de richting van een lichtbron. Als deze straal een voorwerp snijdt dat dichterbij dit punt is dan de lichtbron, wordt het licht van de bron verhinderd. Deze wordt dan niet meegenomen in de (Phong) belichtingsberekening.

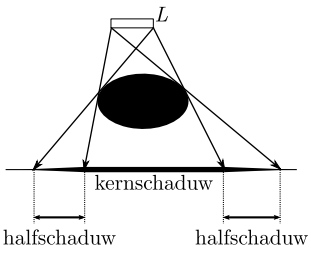


Figuur 4.4 Schaduw van een puntlicht

4.3.1 Halfschaduwen

Een probleem met deze methode is het feit dat het alleen correct werkt met puntlichten. Alle schaduwen hebben dus een scherpe rand, geen halfschaduw (zie figuur 4.4). In werkelijkheid bestaan er geen lichtbronnen zonder grootte. Als de lichtbron wel een grootte heeft, ontstaat er een halfschaduw (zie figuur 4.5).

Halfschaduwen zijn een stuk moeilijker om te benaderen dan schaduwen van een puntlicht. Er komen veel factoren bij kijken die de schaduw beïnvloeden, zoals de afstand van het oppervlak naar het het voorwerp dat de schaduw veroorzaakt en de afstand naar de lichtbron. We zullen hier dus verder niet op in gaan.



Figuur 4.5 Halfschaduw bij een ruimtelijke lichtbron

Hoofdstuk 5

Snijpuntsberekeningen

Zonder snijpuntsberekeningen kan een raytracer natuurlijk niets tekenen. Deze berekeningen worden erg vaak gedaan, dus is het zaak om de snelste methoden te vinden. Hierin verschilt de praktijk met de wiskundige theorie. Vaak kan op verschillende manieren berekend worden of er een snijpunt is. Wiskundig gezien leveren deze manieren precies dezelfde antwoorden op, maar als we deze gaan uitwerken op een computer, kunnen er subtiele verschillen tevoorschijn komen. Bovendien is de ene methode in sommige gevallen sneller en in andere gevallen weer langzamer is dan een andere methode, of levert onjuiste antwoorden.

5.1 Bollen

Willen we weten of een straal R een bol snijdt, moeten we eerst weten hoe een bol gedefinieerd is:

$$f(\mathbf{x}) = (\mathbf{x} - \mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}) - r^2 = 0 \tag{5.1}$$

$\mathbf{c}$: middenpunt

r: radius

5.1.1 Algebraïsche methode

De vergelijking is gemakkelijk met de abc-formule op te lossen. Vergelijking 5.1 lijkt echter helemaal niet op een vergelijking in de vorm van $ax^2 + bx + c$. We moeten weten wanneer $(R \circ f)(t) = 0$. Dat wordt in dit geval:

$$(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - r^2 = 0 \tag{5.2}$$

Willen we de abc-formule kunnen gebruiken, moeten we het in de vorm van $at^2 + bt + c$ krijgen:

$$\begin{aligned} a &= \mathbf{d} \cdot \mathbf{d} \\ b &= 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d} \\ c &= (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 \end{aligned}$$

Als $\mathbf{d}$ genormaliseerd is, is a in alle gevallen 1, dus hoeft die berekening niet meer uitgevoerd te worden. Als de discriminant negatief is, zijn er geen snijpunten en hoeft de berekening verder niet meer uitgevoerd te worden. In de andere gevallen is de oplossing t:

$$t = \frac{-b \pm \sqrt{d}}{2a} \tag{5.3}$$

Met $d = b^2 - 4ac$. Als we twee oplossingen krijgen, zijn we alleen geïnteresseerd in de kleinste t, de t het dichtst bij de camera. Bovendien is alleen de $t \geq 0$ interessant. We kunnen deze t invullen in de functie van de straal $R(t) = \mathbf{o} + \mathbf{d}t$ en krijgen zo het snijpunt.

5.1.2 Meetkundige methode

Uiteraard is er meer dan eén aanpak voor dit probleem. Het snijpunt van een straal met een bol kan ook bepaald worden met behulp van eenvoudige meetkunde.

Allereerst bepalen we het lijnstuk O_c (zie figuur 5.1) tussen de oorsprong van de straal ($\mathbf{o}$) en het middenpunt van de bol ($\mathbf{c}$). Als $|O_c|^2 \leq r^2$ kunnen we de berekening afbreken[1], omdat we dan in de bol zitten. Tevens geldt $\mathbf{c} - \mathbf{o} < 0$ als de bol achter de oorsprong van de straal zit. In dat geval kan de berekening ook afgebroken worden.

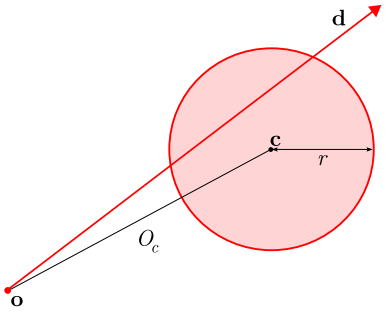
Vervolgens bepalen we t_d: de t die, ingevuld in $R(t)$, het punt oplevert dat het dichtst bij $\mathbf{c}$ zit (zie figuur 5.2). De afstand tussen deze twee punten is d. We weten O_c en de richting van de straal $\mathbf{d}$. Deze vormen een rechthoekige driehoek, dus kunnen we t_d met de stelling van Pythagoras berekenen. $\mathbf{d}$ moet genormaliseerd zijn:

$$t_d = O_c \cdot \mathbf{d} \tag{5.4}$$

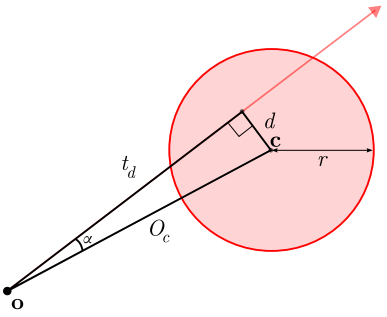
Als $t_d = 0$, is er geen snijpunt met de bol ($\angle\alpha = \frac{\pi}{2}$). Als $t_d < 0$, zit de bol achter de camera. In beide gevallen kunnen we de berekening dan afbreken: er is geen (bruikbaar) snijpunt. In de andere gevallen bepalen we vervolgens de afstand d:

$$d^2 = |O_c|^2 - t_d^2 \tag{5.5}$$

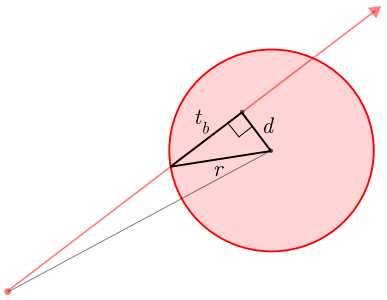
[1] We vergelijking hier de kwadraten, omdat lengten meestal een wortelberekening nodig hebben. Wortelberekeningen zijn relatief erg traag, dus worden het liefst vermeden.



Figuur 5.1 Lijnstuk O_c tussen $\mathbf{o}$ en $\mathbf{d}$.



Figuur 5.2 Bepalen van t_d.



Figuur 5.3 Bepalen van t_b.

Indien $d^2 > r^2$ valt de straal buiten de bol en is er geen snijpunt. Als dat niet het geval is, moet t_b uitgerekend worden, dus de afstand op de straal van het snijpunt met de bol tot het punt $R(t_d)$ (zie figuur 5.3):

$$t_b^2 = r^2 - d^2 \tag{5.6}$$

Nu moeten we nog de t van het snijpunt zelf berekenen: $t = t_d - t_b$.

5.1.3 Berekenen van de normaal

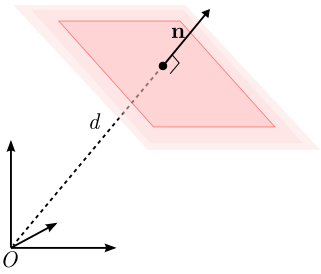
Als we het snijpunt weten, kunnen wij gemakkelijk de normaal bepalen. De vector loodrecht op de bol in het snijpunt $\mathbf{p}$ heeft dezelfde richting als de lijn vanuit het centrum (vergelijking 5.7). De resulterende vector moet uiteraard genormaliseerd worden. De lengte van deze vector hoeven we niet uit te rekenen: de lengte is gelijk aan de radius van de bol.

$$\mathbf{n} = \frac{\mathbf{p} - \mathbf{c}}{r} \tag{5.7}$$

5.1.4 Voor- en nadelen van beide methoden

De meetkundige methode lijkt langer dan de algebraïsche methode, maar het voordeel van de meetkundige is wel dat we sneller kunnen weten of er geen snijpunt is. In de meeste gevallen raken het grootste deel van de stralen de bol niet, omdat de bol anders het hele scherm in beslag neemt. De berekeningen kunnen dus versneld worden als er zonder langzame berekeningen bepaald kan worden of er snijpunt is of niet.

5.2 Vlakken



Figuur 5.4 Een vlak vastgelegd met $\mathbf{n}$ en d.

Een vlak staat vast als we de normaal (deze hoeven we later dus ook niet meer uit te rekenen) weten en de afstand tot de oorsprong, d (zie figuur 5.4).

Een vlak is oneindig lang en breed, maar er is slechts eén punt het dichtst bij de oorsprong. In plaats van het opgeven van een punt voor de plaats van het vlak, voldoet de afstand (d) tot de oorsprong om het vlak vast te leggen.

De vergelijking voor een vlak is dus:

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{n} - d = 0 \tag{5.8}$$

Om het snijpunt te gaan berekenen, moeten we de straal invullen:

$$f(R(t)) = (\mathbf{o} + \mathbf{d}t) \cdot \mathbf{n} - d = 0 \tag{5.9}$$

Als we de termen herschikken, krijgen we:

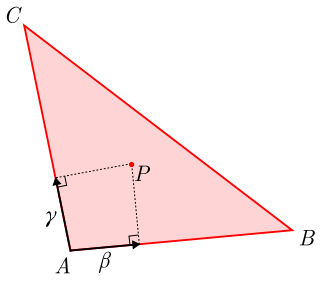
$$t = \frac{d - \mathbf{n} \cdot \mathbf{o}}{\mathbf{n} \cdot \mathbf{d}} \tag{5.10}$$

t moet uiteraard positief zijn. Als $\mathbf{n} \cdot \mathbf{d} = 0$ ($\mathbf{n}$ en $\mathbf{d}$ staan loodrecht op elkaar), krijgen we een deling door 0, dus is er geen snijpunt gedefinieerd.

5.3 Driehoeken

Driehoeken lijken op zichzelf niet bijster interessant, maar in computer graphics worden driehoekjes veel gebruikt om complexere objecten uit op te bouwen. Deze objecten kunnen uit duizenden tot miljoenen driehoekjes bestaan. Hoe je zo snel mogelijk een snijpunt met een driehoek kunt bepalen is dus een veelvuldig bestudeerd onderwerp. We zullen hier alleen de algebraïsche methode bespreken.

Een driehoek heeft drie hoekpunten, of *vertices* (Latijn voor 'toppen'). Deze vertices mogen natuurlijk niet op dezelfde lijn liggen.



Figuur 5.5 Punt P in driehoek $\triangle ABC$

5.3.1 Parametrische methode

Een punt P binnen een driehoek $\triangle ABC$ (zie figuur 5.5) kunnen we met de volgende vergelijking beschrijven:

$$P = A + \beta(B - A) + \gamma(C - A) \tag{5.11}$$

Hierbij moet gelden $\beta \geq 0$, $\gamma \geq 0$ en $\beta + \gamma < 1$. Als aan deze voorwaarde niet voldaan wordt, ligt punt P buiten $\triangle ABC$. We moeten dus β en γ berekenen. We krijgen dan de volgende vergelijking:

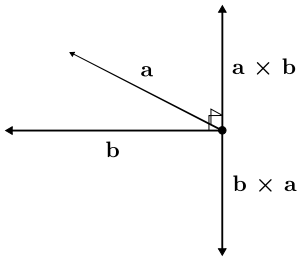
$$\mathbf{o} + \mathbf{d}t = A + \beta(B - A) + \gamma(C - A) \tag{5.12}$$

We kunnen dit systeem van lineaire vergelijkingen herschrijven in matrixvorm:

$$\begin{bmatrix} x_B - x_A & x_C - x_A & x_d \\ y_B - y_A & y_C - y_A & y_d \\ z_B - z_A & z_C - z_A & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_A - x_o \\ y_A - y_o \\ z_A - z_o \end{bmatrix} \tag{5.13}$$

Dit systeem kan opgelost worden met behulp van de regel van Cramer.[2] We kunnen de berekening vroegtijdig afbreken als β, γ of de som hiervan buiten $[0, 1]$ valt. Het is dus handig deze variabelen als eerste te berekenen.

[2] De uitwerking hiervan is te omvangrijk en behoeft te veel uitleg over matrices en hun eigenschappen om hier verder op in te gaan.



Figuur 5.6 Het uitproduct van **a** en **b** levert vector **n** op.

5.3.2 Berekenen van de normaal

Het uitproduct van twee vectoren levert een nieuwe vector op die loodrecht staan op het vlak dat wordt beschreven door de twee vectoren. Zo kunnen wij aan de hand van de vertices van de driehoek de normaal bepalen. In de praktijk moet deze echter nog wel genormaliseerd worden.

$$\mathbf{n} = (B - A) \times (C - A) \tag{5.14}$$

Het uitproduct is niet commutatief ($\mathbf{a} \times \mathbf{b} \neq \mathbf{b} \times \mathbf{a}$), maar anticommutatief ($\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$), zoals in figuur 5.6 te zien is. Het maakt dus wèl uit in welke volgorde wij de punten van de driehoek opslaan. Als de punten in de volgorde met de klok mee zijn opgeslagen, is de normaal tegenovergesteld aan de normaal van de driehoek waarvan de punten tegen de klok in zijn opgeslagen.

Hoofdstuk 6

Antialiasing

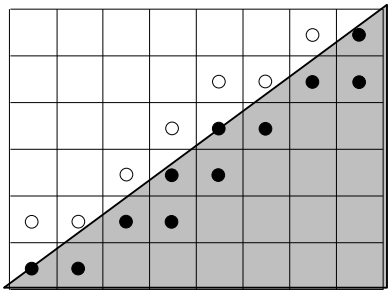
Aliasing is het verschijnsel dat door bemonstering de bron niet juist wordt weergegeven. Het meest duidelijke voorbeeld hiervan is de vorming van kartelige randjes bij schuine randen als deze in de vorm van een raster worden bemonsterd.

Dit wordt veroorzaakt door het feit dat we van een continu veld naar een discreet veld (hier een rasterafbeelding) gaan (zie figuur 6.1).

In programma 6 nemen we maar eén monster per beeldpunt. We schieten dus maar eén straal.

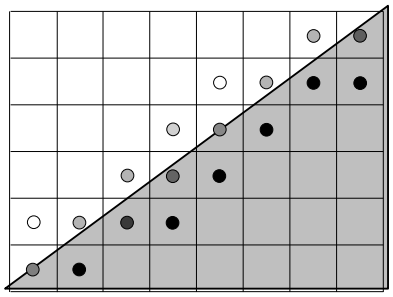
Programma 6.1 Reguliere bemonstering

voor elk *beeldpunt* (x, y) **in** *raster* **doe**
  kleur beeldpunt := $raytrace(x, y)$



Figuur 6.1 Aliasing artefacten

6.1 Overbemonstering



Figuur 6.2 Antialiasing

Het is onvermijdelijk dat we een blokkerig beeld krijgen, maar we kunnen het wel minder blokkerig laten lijken. We kunnen bijvoorbeeld meerdere monsters per beeldpunt nemen en dan het gemiddelde gebruiken. Bij een overgang van zwart naar wit, krijgen we grijze beeldpunten op de grens (zie figuur 6.2).

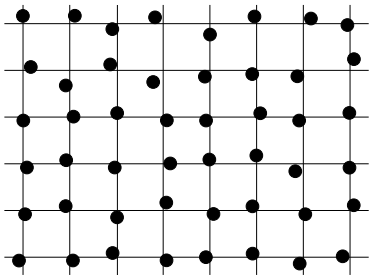
Als we meerdere monsters per beeldpunt willen nemen, ligt het voor de hand dat we deze monsters regelmatig binnen het vierkant van het beeldpunt nemen. Het beeldpunt wordt weer in een nieuw raster verdeeld met n rijen en n kolommen. Van alle n^2 monsters per beeldpunt moeten we dan de kleur optellen en delen door n^2:

Programma 6.2 Reguliere overbemonstering

voor elk *beeldpunt* (x, y) **in** *raster* **doe**
  begin
  $kleur := zwart$
    voor $i := 0$ **in** $[0, n)$ **doe**
      voor $j := 0$ **in** $[0, n)$ **doe**
        $kleur := kleur + raytrace(x + i/n, y + j/n)$
    $kleur\ beeldpunt := kleur\ /\ n^2$
  eind

6.2 Stochastische overbemonstering

Reguliere overbemonstering heeft ook nadelen omdat het regelmatig is. Daardoor kunnen er Moiré patronen ontstaan. Dit zie je bijvoorbeeld op televisie, wanneer er vage lijnen over een dicht gestreepte blouse lijken te bewegen. We kunnen dit oplossen door niet vaste plaatsen te kiezen voor de monsters, maar deze willekeurige plaatsen. Als deze volstrekt willekeurig zijn, zullen punten te dicht bij elkaar liggen of juist te ver van elkaar. We kunnen beter een Poisson-schijf patroon gebruiken. In een dergelijk patroon hebben alle punten een minimum afstand van elkaar. Het nadeel hiervan is dat het veel tijd kost om zo'n patroon te genereren. Het is echter gemakkelijk na te bootsen door het reguliere patroon te nemen en voor elk punt een kleine willekeurige waarde bij i en j op te tellen of af te trekken.



Figuur 6.3 Stochastische bemonstering

6.3 Adaptieve overbemonstering

Een andere oplossing is om het aantal monsters aan te passen aan de situatie. Er zijn niet altijd evenveel monsters nodig om een goed beeld van de gemiddelde kleur van een beeldpunt te krijgen.

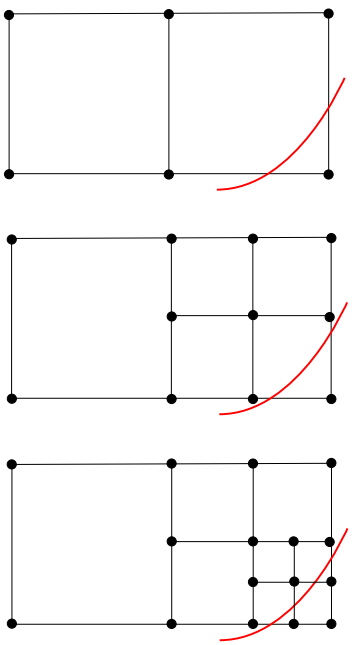
Adaptieve bemonsteringsalgoritmen maken gebruik van de kleurwaarden om een inschatting te maken over de hoeveelheid detail in

de bron. Ze nemen dus geen kennis van de eigenlijke details van bron, maar kijken eerst naar het verschil in kleurwaarde van de omliggende monsters voordat ze naar meer detail zoeken.

Een dergelijk algoritme blijft bemonsteren totdat het nemen van extra monsters geen zichtbaar effect meer heeft op het eindresultaat.

Het is zelfs mogelijk om niet eens voor elk beeldpunt een monster te nemen. Dan worden de beeldpunten waar geen monster voor is genomen berekend door de omliggende monsters te interpoleren. Als het algoritme echter begint met een te grote afstand tussen de monsters, kan het zo zijn dat een detail, bijvoorbeeld een kleine bol, tussen de monsters niet weergegeven wordt. Uit de omliggende monsters kon dan niet opgemaakt worden dat ertussen nog een detail lag.

Adaptieve overbemonstering geeft dus perfecte resultaten voor scènes zonder erg kleine details. Wanneer het wel nodig is om kleine details correct weer te geven moet de minimale afstand tussen bemonstering net iets kleiner zijn dan het kleinste significante detail. Hierdoor kan het zijn dat adaptieve bemonstering niet meer zo veel gunstiger is dan reguliere bemonstering.

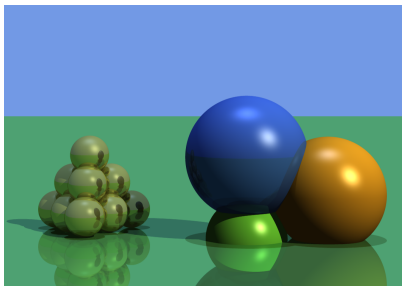


Figuur 6.4 Het raster past zich aan de situatie aan.

Hoofdstuk 7

Bespreking

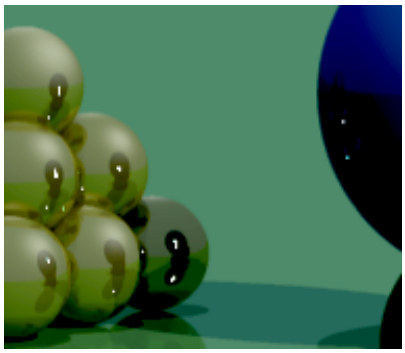
7.1 Conclusie



Figuur 7.1 Rendering gemaakt met de Huygens raytracer.

In de voorgaande hoofdstukken hebben we gezien hoe we met ray tracing technieken de rendervergelijking kunnen benaderen. Uiteraard zijn er nog meer verschijnselen die we niet gemodelleerd hebben. Denk hierbij aan breking van licht bij doorzichtige oppervlakken en andere verschijnselen veroorzaakt door het golfkarakter van licht. Het benaderen van indirecte belichting hebben we helaas vrijwel niet kunnen behandelen.

7.2 Huygens raytracer



Figuur 7.2 Niet alles gaat meteen goed.

Wij hebben zelf een raytracer geschreven, en deze tot Huygens raytracer gedoopt. Christiaan Huygens heeft ook op het gebied van de optica veel onderzoek gedaan. Ironisch is het echter dat de door hem geïntroduceerde golftheorie juist niet gebruikt wordt in deze raytracer. Van het schrijven van de raytracer hebben we veel geleerd, omdat je de theorie goed moet begrijpen als je het wilt implementeren. Als er fouten in de rendering tevoorschijn komen, is het niet altijd even makkelijk om te achterhalen waardoor het komt. Deze fouten hebben echter wel geleid tot een beter begrip van de theorie.

Als aanvulling hebben we online het volgende staan:

- Broncode van de Huygens raytracer,
- Renderingen en filmpjes in kleur,
- Beschrijvingen van de gerenderde scènes,
- Digitale versie van dit document.

http://profielwerkstuk.roberthensing.nl/raytracing.zip

Lijst van begrippen

aliasing
Het verschijnsel dat voorkomt als een continu veld met een discreet veld benaderd wordt. Een voorbeeld hiervan zijn schuine lijnen die blokkerig lijken.

algoritme
Een eindige reeks instructies om te beschrijven hoe uit een begintoestand een bepaald doel bereikt wordt.

antialiasing
Methoden die het aliasingprobleem proberen te verdoezelen.

genereren
Het programmatisch creëren van informatie.

eenheidsvector
Vector met een lengte gelijk aan 1.

inproduct
Som van producten van componenten, dus voor drie dimensies: $x_1x_2 + y_1y_2 + z_1z_2$. Resultaat is eéndimensionaal.

normaal
De vector die loodrecht staat op het (raak)vlak.

radius
De helft van de diameter.

realtime
Een proces is realtime als het zo snel gedaan wordt dat het resultaat direct beschikbaar is.

recursie
Wanneer een waarde afhankelijk is van een eerder berekende waarde of waarde die nog berekend moet worden, is er sprake van recursie.

voortgezet op de volgende pagina

rendering
: Het maken van een tweedimensionale afbeelding met behulp van de beschrijving van een scène.

scène
: Verzameling objecten en lichtbronnen in een driedimensionale ruimte.

straal
: Een lijn die aan eén kant is begrensd.

uitproduct
: Product van twee vectoren $\mathbf{a} \times \mathbf{b}$ dat een vector loodrecht op $\mathbf{a}$ en $\mathbf{b}$ oplevert.

weerkaatsen
: Het veranderen van de richting van een lichtstraal, waarbij de intensiteit veranderd wordt, door gedeeltelijke absorptie.

weerspiegelen
: Het veranderen van de richting van een lichtstraal, waarbij de intensiteit onveranderd blijft, zoals bij een spiegel.

Lijst van figuren

Bibliografie

Artikelen en boeken

- "Introduction to Realtime Ray Tracing", Peter Shirley. p. 158-160 (2005)
- "A Practical Guide to Global Illumination using Ray Tracing and Photon Mapping", Henrik W. Jensen. p. 25-27 (2002)
- "From Local to Global Illumination and Back", Alain Fournier. (1995)
- "Monte Carlo Ray Tracing", Henrik W. Jensen et al. p. 89-93 (2003)
- "The Rendering Equation", James T. Kajiya. (1986)
- "Computer Graphics: Principles and Practice", James D. Foley et al. p. 721-792 (1991)
- "Fast, Minimum Storage Ray/Triangle Intersection", Thomas Möller, Ben Trumbore. (1997)
- "An Efficient Ray-Polygon Intersection", Didier Badouel. (1990)
- "Ray-Triangle Intersection Using Binary Recursive Subdivision". Douglas Voorhies, David Kirk. (1991)

Webpagina's

- http://en.wikipedia.org/wiki/Lambertian_reflectance
- http://en.wikipedia.org/wiki/Rendering
- http://en.wikipedia.org/wiki/Dotproduct
- http://en.wikipedia.org/wiki/Cross_product
- http://en.wikipedia.org/wiki/System_of_linear_equations
- http://www.devmaster.net/wiki/Ray-sphere_intersection
- http://www.siggraph.org/education/materials/HyperGraph/raytrace/rayplane_intersection.htm

Index