KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

realtime renderen van vele lichtbronnen

Martinus Wilhelmus Tegelaers

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Mens-machine communicatie

Promotor:
Prof. dr˙ ir. P. Dutre

Assessor:
T. Do

Begeleider:
T.O. Do

Academiejaar 2016 – 2017

Voorwoord

Dit is mijn dankwoord om iedereen te danken die mij bezig gehouden heeft. Hierbij dank ik mijn promotor, mijn begeleider en de voltallige jury. Ook mijn familie heeft mij erg gesteund natuurlijk.

Martinus Wilhelmus Tegelaers

Inhoudsopgave

Samenvatting

In dit environment wordt een al dan niet uitgebreide samenvatting van het werk gegeven. De bedoeling is wel dat dit tot 1-bladzijde beperkt blijft.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

Dit is de algemene inleiding placeholder

1.1 Probleemstelling

probleem stelling placeholder

1.2 Overzicht Thesis

Wat volgt is een kort overzicht van de hoofdstukken binnen deze thesis:

Chapter 2: Theorie Het theorie hoofdstuk zal een korte inleiding geven tot het renderen van 3d computer graphics en de basis technieken gebruikt binnen deze thesis.

Chapter 3: Literatuur Studie De literatuur studie bouwt verder op de brede inleiding van chapter 2, en behandelt de specifieke papers waarop deze thesis verder bouwt. Hierbij worden zowel de rendering technieken besproken die in de thesis geimplementeerd zijn, als de datastructuren gebruikt in het voorgestelde algoritme.

Chapter 4: Methode: Eigen Algoritme Hier wordt het eigen algoritme besproken

Chapter 5: Methode: Implementatie Hier wordt de implementatie besproken

Chapter 6: Resultaten Hier worden de resultaten besproken

Chapter 7: Discussie en conclusie Hier worden de resultaten bediscussieerd

Hoofdstuk 2

Theorie

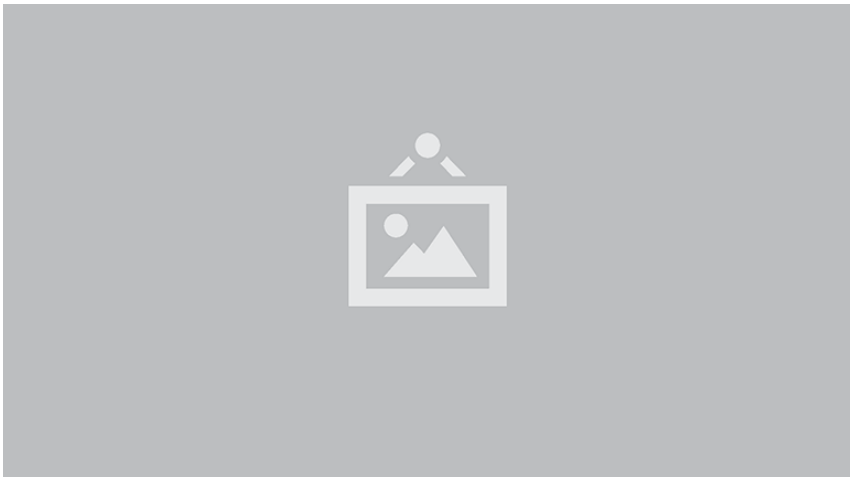
Zoals besproken in de inleiding draait de thesis om het renderen van drie dimensionale scenes in real-time. Het doel is hierbij veelal om geloofwaardige afbeeldingen te creeeren uit een bepaalde drie dimensionale scene beschrijving. In veel gevallen betekent dit dat de scene fotorealistisch dient afgebeeld te worden, echter andere stylistische keuzes zijn tevens gebruikelijk. In alle gevallen echter is het concept van geloofwaardigheid in grote mate afhankelijk van de manier hoe mensen de wereld om zich heen waarnemen. Voordat dan ook verder ingegaan wordt op de algoritmes om dergelijke afbeelding te produceren, zal eerst ingegaan worden op deze perceptie. Nadat vastgesteld is wat bereikt dient te worden met renderen, zal ingegaan worden op hoe dit mathematisch voor te stellen, en welke algoritmes gebruikt worden om deze problemen op te lossen. Als laatste zal besproken worden hoe dit binnen huidige generatie videokaarten op hardware niveau geimplementeerd is.

2.1 Fysische werkelijkheid

De fysische wereld waarin de mens zich bevindt wordt gedicteerd door alle fysische wetten. De mens neemt deze wereld door middel van zintuigen. Voor computer graphics, waarneming is het belangrijkste zintuig. Door middel van waarneming wordt de drie dimensionale wereld om de mens heen geinterpeteerd. Deze interpretatie zal de fysische werkelijkheid genoemd worden binnen deze thesis. Zowel de fysische wereld als de manier waarop deze waargenomen wordt bepaald dus de fysische werkelijkheid.

2.1.1 Waarneming

De mens neemt de wereld waar door het oog. Het menselijk oog interpreteert de drie dimensionale wereld door stralen van licht te focussen op een enkel punt, doormiddel van een lens. Het enkele punt dat licht omzet naar neurosignalen wordt de retina genoemd. Een camera bootst het oog na, en projecteert licht op een electronische photosensor, die het signaal op zet naar een digitaal signaal. Dit is weergegeven in figuur 2.1.



Figuur 2.1: Waarneming doormiddel van het oog en camera.

Deze manier van projectie heeft twee belangrijke gevolgen:

- Objecten worden als kleiner waargenomen naarmate ze verder van de waarnemer af staan.

- Objecten worden waargenomen met Foreshortening, i.e. de dimensies van een object parallel aan het gezichtsveld, worden als kleiner waargenomen dan dimensies van hetzelfde object loodrecht aan het gezichtsveld.

De mens verwacht dat deze eigenschappen aanwezig zijn, om beelden te interpeteren.

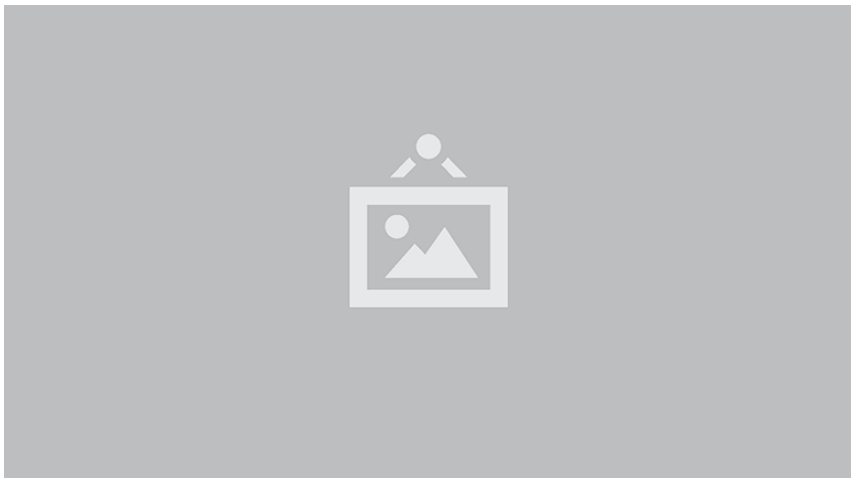
2.1.2 Licht

Het tweede belangrijke inzicht bij waarneming is dat de wereld wordt waargenomen door middel van licht. Dit betekent dat bij afwezigheid van licht, het niet mogelijk is om iets waar te nemen. Verder betekent dat ook dat het gedrag van licht een grote invloed heeft op de manier hoe de wereld wordt waargenomen.

Licht is een electromagnetische straling. Hierdoor zal licht onder normale omstandigheden zich altijd in een rechte lijn voortplanten zolang het binnen hetzelfde medium blijft. Indien het licht in contact komt met een nieuw medium zijn er verschillende fenomen die kunnen gebeuren:

Absorptie Het licht wordt geabsorbeerd door de atomen van het nieuwe medium, en uitgestoten als warmte. Hierbij gaat het licht verloren.

Reflectie Het licht wordt gereflecteerd op het oppervlakte van het nieuwe medium. Hierbij wordt het licht terug de scene ingestuurd. De hoek van reflectie hangt af van het type medium. Indien het materiaal zich gedraagd als een spiegel, en



FIGUUR 2.2: Absorptie, reflectie en transmissie van licht.

zal het licht teruggekaatst worden met dezelfde hoek gespiegeld om de normaal. Indien het materiaal licht diffuus weerspiegelt betekent dat de hoek van inval niet uitmaakt voor de reflectie en deze min of meer willekeurig is.

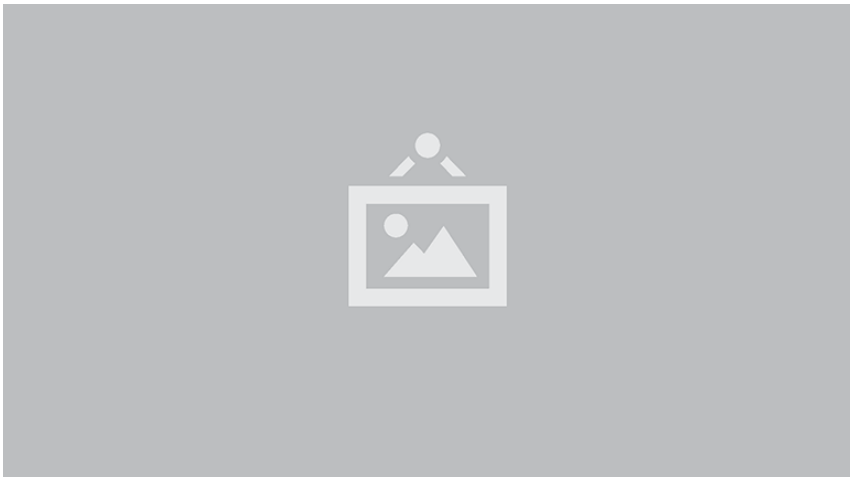
Transmissie Het licht plant zich verder voort door het nieuwe medium, opnieuw in een rechte lijn, met mogelijk een iets andere richting op basis van de brekingsindex van het nieuwe en het oude medium.

Deze fenomenen zijn verder geillustreerd in figuur 2.2. Ze zijn niet exclusief aan elkaar. Een medium kan dus bijvoorbeeld een gedeelte van het licht absorberen en een ander gedeelte reflecteren.

Zoals eerder vermeldt, neemt het oog de wereld waar door licht op te vangen. Het merendeel van het licht dat opgevangen wordt is gereflecteerd via een of meerdere oppervlaktes. Een belangrijke constatering is dat objecten slechts zichtbaar zijn als er binnen het medium geen (onderzichtige) andere media liggen tussen het object en de lens. Dit is een triviale constatering in de fysische werkelijkheid echter dit zal niet triviale consequenties hebben binnen de computer graphics zoals later zal worden beschreven.

2.1.3 Kleur

Een tweede belangrijk aspect van licht voor computer graphics is het concept kleur. Het concept kleur is niet een fysisch verschijnsel, maar een consequentie van hoe ogen licht interpeteren. De mens neemt slechts een gedeelte van al het licht waar. Dit wordt het zichtbare licht genoemd. Het menselijk oog interpeteert het licht Door het zowel een intensiteit als een kleur toe te kennen. De kleur die waargenomen wordt van een lichtstraal is afhankelijk van het licht. Een gemiddeld persoon is in staat om 3 verschillende primaire kleuren waar te nemen, rood, blauw en groen. Elke zichtbare kleur kan voorgesteld worden als een mix van deze primaire kleuren. De



Figuur 2.3: Het mengen van kleuren volgens een additief model.

manier om deze kleuren te mengen is afgebeeld in figuur 2.3 Belangrijk om hierbij te vermelden is dat licht zich gedraagd als additieve kleurmenging. Dit houdt in dat indien verschillende kleuren licht op het zelfde punt worden afgebeeld, dit punt zal worden waargenomen als de kleur gelijk aan de optelling van deze lichten.

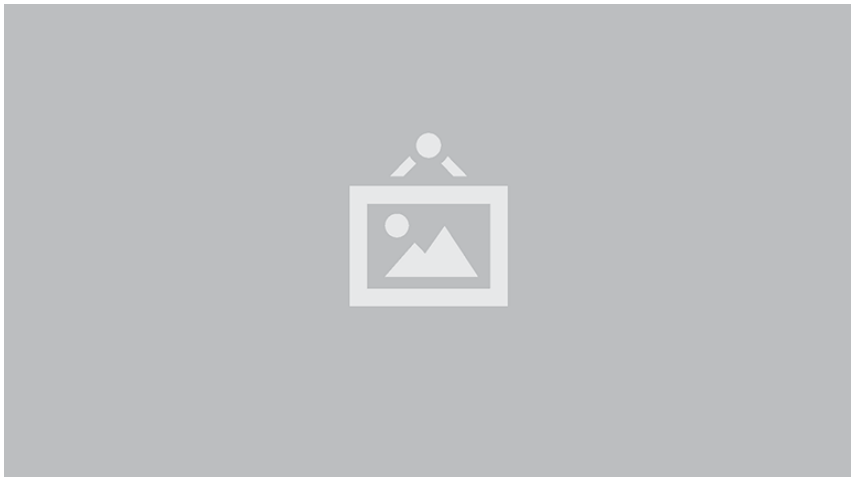
Objecten kunnen tevens een kleur hebben. Reeds is besproken dat objecten worden waargenomen door de reflectie van hun licht. De kleur van een object is dan ook het gevolg van de gedeeltelijk absorptie van licht. In het geval dat een gekleurd object wordt verlicht met puur wit licht, zal slechts het licht dat overeenkomt in frequentie met de kleur van het object weerspiegelt worden. De frequenties tegenovergesteld aan de kleur van het object, zullen worden geabsorbeerd door het object.

2.1.4 Simulatie

Computer graphics heeft als doel om de fysische werkelijkheid te benaderen. Echter hiervoor is het niet nodig om de volledige fysische werkelijkheid te benaderen. Het simuleren van de fysische werkelijkheid om een afbeelding te verkrijgen, het renderen, kan dus in grofweg in twee problemen ingedeeld worden:

- Wat is zichtbaar binnen een scene vanuit het huidige gezichtspunt.
- Hoe ziet datgene wat zichtbaar is er uit binnen onze afbeelding.

Wat afgebeeld wordt op een afbeelding, hangt af van twee aspecten, hoe wordt de 3d scene op het 2d beeld geprojecteerd. En wat van elk object dat geprojecteerd kan worden is daadwerkelijk zichtbaar op de uiteindelijk afbeelding. Het eerste wordt perspectief projectie genoemd. Het tweede probleem wordt het visibiliteitsprobleem probleem genoemd.



FIGUUR 2.4: Voorstelling van objecten doormiddel van driehoeken.

Hoe hetgene wat afgebeeld wordt, er uiteindelijk uit ziet binnen onze afbeelding, wordt in de tweede stap bepaald. Deze stap wordt shading genoemd, en alle berekeningen gerelateerd aan kleur, absorptie, weerspiegeling etc, vallen hier onder.

2.2 Notaties en definities

Voordat de achterliggende theorie verder behandelt wordt, zal eerst kort ingegaan worden op de verschillende notaties en definities binnen deze thesis.

2.2.1 Geometrische definities

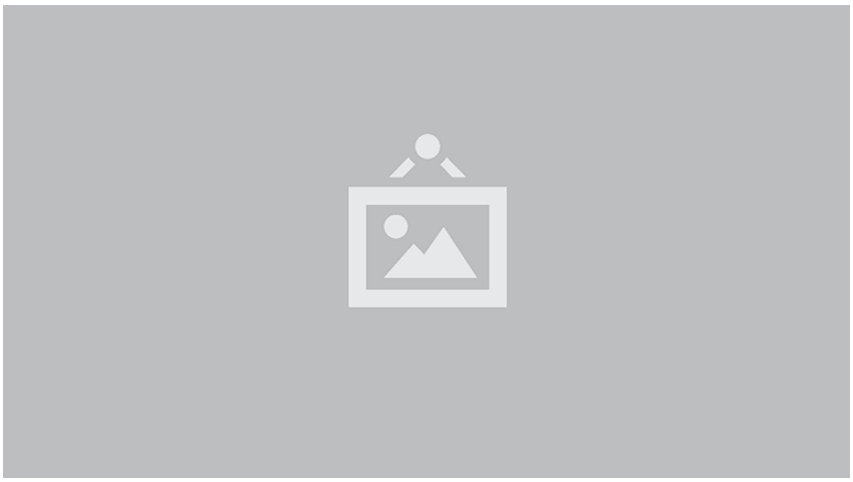
Voordat het mogelijk is om drie dimensionale omgevingen om te zetten naar afbeeldingen is het nodig om een beschrijving van deze scenes te hebben. De basis render primitieven die gebruikt worden door de meeste grafische hardware zijn punten, lijnen en driehoeken. Een punt wordt beschreven met homogene coordinaten:

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix}$$

Indien gesproken wordt van primitieven zal, als niet anders is aangegeven, gedoeld worden op driehoeken. De notatie voor een driehoek is:

$$\triangle \mathbf{v_1 v_2 v_3}$$

Een *mesh* is een collectie van driehoeken die samen de vorm van een object vormen. Een *object* bevat zowel een *mesh* als een transformatie matrix die die de locatie, schaal, en rotatie van het *object* vastlegt. De verschillende geometrische



Figuur 2.5: Het standaard camera model.

definities zijn geillustreerd in figuur 2.4. De volledige beschrijving van een virtuele omgeving zal een *scene* genoemd worden. Deze bevat een set van objecten, de lichten, en eventuele definities van gezichtspunten.

2.2.2 Camera model

Het gezichtspunt, of *camera* binnen deze thesis is het standaard camera model binnen computer graphics. Zoals geillustreerd in figuur
2.5.

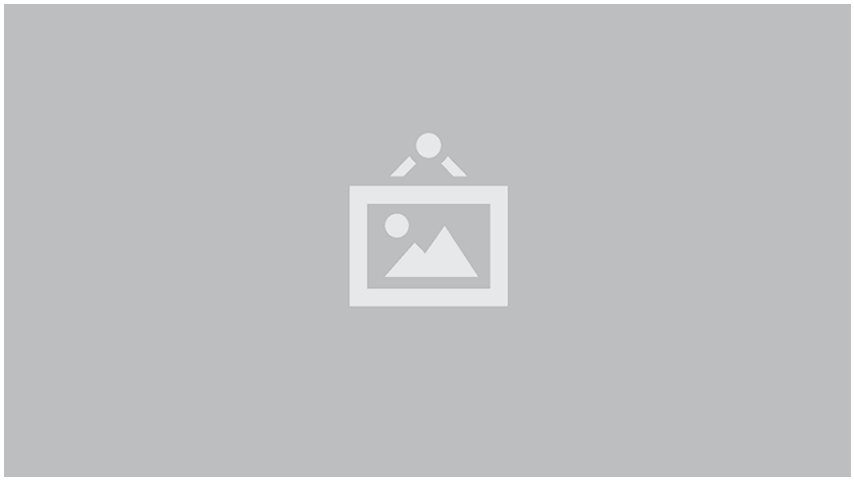
De volgende vectoren en punten zijn hiervoor gedefinieerd

Up De locale y-as van de camera.
Eye (x, y, z) positie van de camera in wereld coordinaten.
Centre (x, y, z) positie waarnaar de camera kijkt
Z-near de near z plane waarop het beeld wordt geprojecteerd.
Z-far Het vlak waarachter fragmenten niet meer worden weergegeven.

De *Z-near* en *Z-far* in combinatie met *Eye* creeert de view frustrum. Slechts Primitieven binnen dit view frustum zullen worden gerenderd.

2.3 Perspectief projectie en het visibileitsprobleem

In de sectie ... zijn de twee problemen vastgesteld die opgelost dienen te worden om geloofwaardige afbeeldingen te generen. In deze sectie zal de eerste geadresseerd worden, wat is zichtbaar binnen een scene vanuit het huidige gezichtspunt. Om te bepalen wat zichtbaar is dient zowel het perspectief gesimuleerd te worden, als bepaald te worden welk van de objecten in perspectief als eerste zichtbaar is.



FIGUUR 2.6: Perspectief projectie.

2.3.1 Perspectief projectie

Zoals eerder besproken zijn de twee eigenschappen van perspectief:

- Objecten worden als kleiner waargenomen naarmate ze verder van de waarnemer af staan.

- Objecten worden waargenomen met Foreshortening, i.e. de dimensies van een object parallel aan het gezichtsveld, worden als kleiner waargenomen dan dimensies van hetzelfde object loodrecht aan het gezichtsveld.

Deze effecten kunnen gesimuleerd worden door de 3d scene te projecteren naar het oogpunt en af te beelden op het canvas. Zoals weergegeven in figuur 2.6.

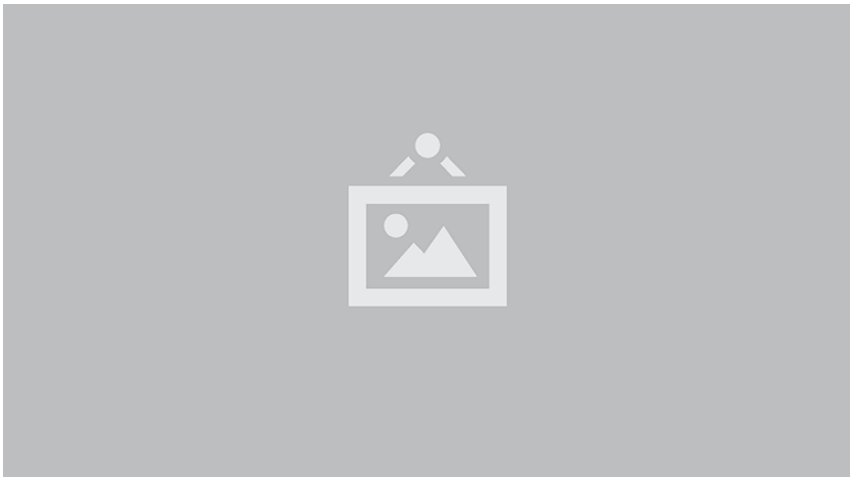
Zoals eerder beschreven bestaan de objecten binnen de scenes uit meshes van primitieven. Omdat elk van deze driehoeken gedefinieeerd kan worden door zijn drie vertices, is het niet nodig om elke mogelijk punt binnen de driehoek af te beelden op de canvas, maar is het genoeg om slechts deze drie vertices te projecteren.

In figuur 2.7 is de projectie $\mathbf{p}'$ van een enkel punt $\mathbf{p}$ op een enkele dimensie van de canvas weergegeven. De hoek $\angle\mathbf{abc}$ Hier is te zien dat de hoek tussen C en AB'C' gelijk is. Dit betekent dat we het punt C' kunnen bereken doordat de verhouding geldt

$$\frac{BC}{AB} = \frac{B'C'}{AB'}$$

Verder is de afstand van het oogpunt tot de canvas, AB', bekend. Wat ertoe leidt dat we het geprojecteerde punt gemakkelijk kunnen berekenen.

$$p'_x = d\frac{p_x}{p_z}$$



Figuur 2.7: Projectie van een enkel punt.

$$p'_y = d\frac{p_y}{p_z}$$
$$p'_z = d$$
$$p'_w = 1$$

waar d de afstand van het oogpunt tot de canvas is. Binnen computer graphics wordt deze stap de perspectief deling genoemd. We kunnen deze berekeningen samenvoegen tot een enkele matrix $\mathbf{P}$ die een punt in een specifiek coordinaten stelsel omzet naar een punt geprojecteerd op de canvas. Wat leidt tot de perspectief projectie van punten.

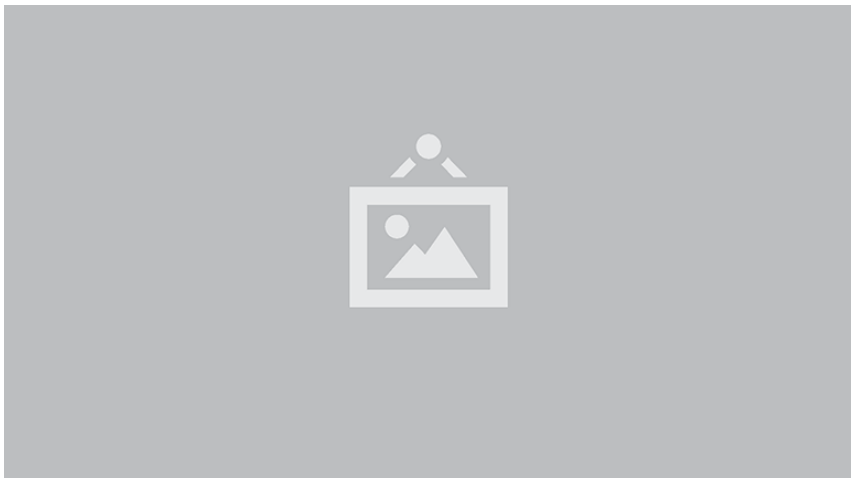
$$\mathbf{Pp} = \mathbf{p}'$$

Of deze projectie daadwerkelijk nodig is, is afhankelijk van de gekozen rendering techniek. Binnen rasterisatie is het nodig om deze stap expliciet uit te voeren. Raytracing neemt de perspectief projectie impliciet mee.

2.3.2 Visibiliteitsprobleem

Er is nu vastgesteld hoe objecten in perspectief afgebeeld op de canvas kunnen worden. Echter, hiermee is nog niet volledig vastgesteld wat daadwerkelijk zichtbaar gaat zijn op het canvas, zoals weergegeven in figuur 2.8. Hiervoor is het tevens nodig om te bepalen welke delen van objecten zichtbaar zijn, en welke verborgen zijn achter andere objecten. Dit probleem wordt onder andere het visibiliteitsprobleem genoemd, en was een van de eerstegrote problemen binnen computer graphics.

De oplossing voor dit probleem is de realisatie dat het visibiliteitsprobleem intrinsiek een sorteerprobleem is. Stel er bestaat een minimale oppervlakte O op het canvas, waarvoor gekeken wordt welk deel zichtbaar is. Door middel van perspectief projectie is het mogelijk om te bepalen welke objecten op O worden afgebeeld. Er van



FIGUUR 2.8: Visibiliteitsprobleem in een scene met meerdere primitieven.

uitgaande dat er een object A bestaat die afgebeeld wordt op O. Dan is dit object daadwerkelijk zichtbaar in O als er geen andere objecten op O worden geprojecteerd die dichterbij het oogpunt liggen dan object A. Wanneer alle objecten gesorteerd zijn is het per punt of het canvas mogelijk om het dichtstbijzijnde object te selecteren, en deze weer te geven op het scherm.

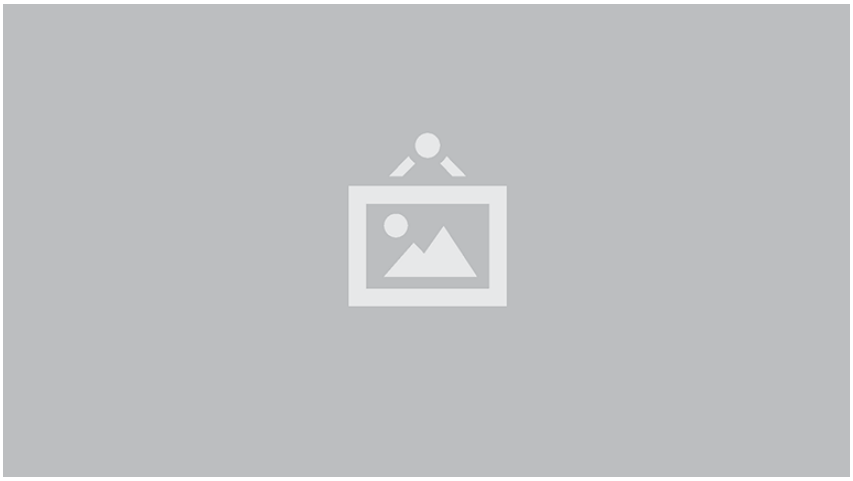
De algoritmes om dit efficient te doen worden verborgen oppervlakte bepalings (hidden surface determination) algoritmes genoemd. Deze kunnen grofweg ingedeeld worden in twee categorien, raytracing en rasterisatie. Hierbij zou, in theorie, geen verschil in resultaat hoeven zijn. Beide klasses van algoritmes hetzelfde doel hebben, het produceren van realistische beelden op basis van een 3d scene.

Raytracing werkt op basis van het trekken van zogenoemde stralen door $\mathbf{p}'$ waarbij de eerste $\mathbf{p}$ wordt bepaald. Rasterisation daarentegen, bepaald alle mogelijke $\mathbf{p}'$ op basis van alle mogelijke $\mathbf{p}$ die geprojecteerd worden op $\mathbf{p}$. Vervolgens wordt bepaald welke $\mathbf{p}'$ daadwerkelijk zichtbaar is.

In de volgende secties zal er kort ingegaan worden op beide technieken.

2.3.3 Raytracing

Raytracing simuleert de werking van licht en het menselijk ook, en lost hiermee zowel het visibiliteitsprobleem als het perspectief op. Er is reeds vastgesteld dat menselijke waarneming berust op het waarnemen van licht dat valt op de lens en geprojecteerd wordt op de retina, de lichtsensor. In theorie is het mogelijk om beelden op een zelfde manier op te bouwen, zoals dit gebeurd in het oog. In dit geval zouden vanuit lichten willekeurige stralen geschoten kunnen worden. Hierbij is een straal gedefinieerd als zijnde een vector met een beginpunt en een richting. Wanneer een dergelijke straal door de canvas op het oogpunt valt, wordt deze meegeteld. Dit is geillustreerd in figuur 2.9. Deze techniek wordt forwaards tracen genoemedt. Echter vaak is het canvas van een camera velen malen kleiner dan de scene in kwestie. Dit zorgt ervoor



Figuur 2.9: Forwaards raytracen.

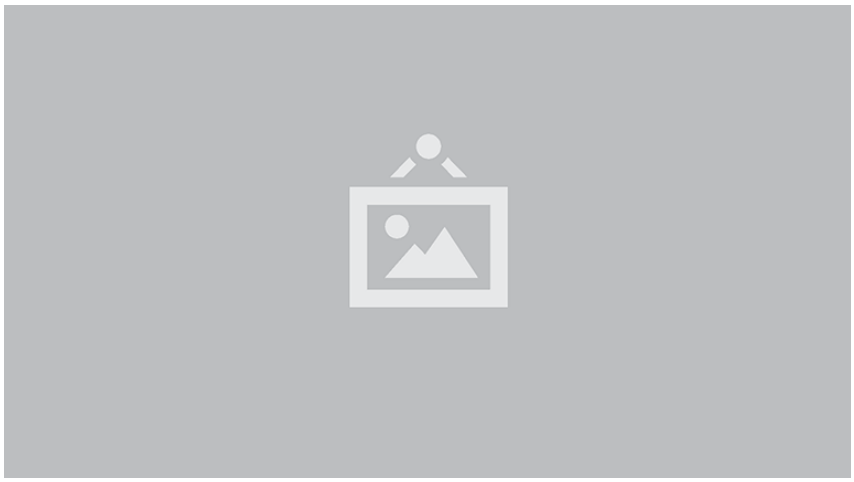
dat de kans dat de canvas geraakt wordt, uitermate klein is. Hierdoor is een groot aantal lichtstralen nodig, voordat een geloofwaardige afbeelding wordt verkregen.

Met de realisatie dat we uiteindelijk slechts de stralen nodig hebben, die door het canvas op het oogpunt vallen kunnen we de techniek omdraaien. In plaats van willekeurige stralen te schieten vanuit lichten, schieten we, per punt dat we willen weten op de canvas, een straal. Deze straal zal dus altijd het oogpunt raken, en door punt $\mathbf{p}'$ gaan. Vervolgens dient gekeken te worden welk object, als er een bestaat, deze straal raakt. Hiermee wordt punt $\mathbf{p}$ gevonden. Vanaf $\mathbf{p}$ kunnen we bepalen welke lichten dit punt raken, en dus hoe het punt $\mathbf{p}'$ gekleurd dient te worden. Dit zal verder besproken worden in de sectie over shading.

Deze techniek, waarbij gestart wordt vanuit de camera wordt, achterwaardse tracing genoemd. Belangrijk om hierbij op te merken is dat ray tracing, dus voor elk punt $\mathbf{p}'$ een punt $\mathbf{p}$ vindt. Wanneer gesteld wordt dat punt $\mathbf{p}'$ een (sub)pixel is, zou een algoritme dus bestaan uit twee loops. In de eerste plaats wordt per pixel een straal gegenereerd. Vervolgens wordt per straal gekeken over alle objecten welke object zowel door de straal geraakt wordt en het dichtstbij ligt. Doordat de buitenste loop over de pixels loopt, worden raytracing algoritmes dan ook wel beeldcentrische algoritmes genoemd. De pseudo code zal er als volgt uitzien:

```
for pixel in canvas:
    ray = construct_ray(eye, pixel)

    for object in scene:
        closest = None
        if (ray.hits(object) and
           (closest == None or distance(object, eye) < distance(closest, eye))):
            closest = object
```



FIGUUR 2.10: Raytrace algoritme.

```
    do_shading(closest, ray)
```

Waarbij `do_shading` gebruikt wordt om de kleur te bepalen van de specifieke pixel. Dit is verder geillustreerd in figuur 2.10

Dit concept is de basis voor alle raytracing algoritmes. Merk hierbij verder op, dat er geen expliciete perspectief projectie plaats vindt. Doordat stralen opgebouwd zijn beginnend in het oogpunt door punt $\mathbf{p}'$, wordt het perspectief impliciet gedefineerd.

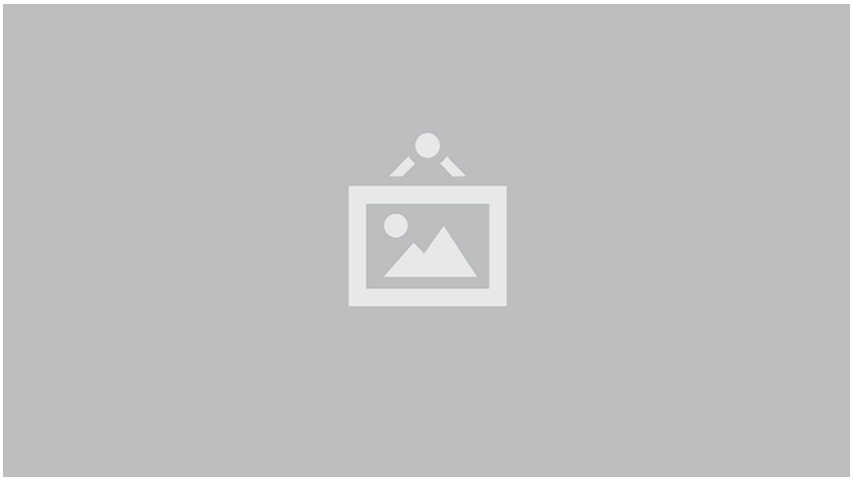
2.3.4 Rasterisatie

Rasterisatie algoritmes lossen perspectief projectie en het visibiliteitsprobleem op een verschillende volgorde op dan hoe het binnen raytracing algoritmes wordt opgelost. Waar raytracing uitgaat van het punt $\mathbf{p}'$ en kijkt welk object hier op valt, begint een rasterisatie algoritme met het afbeelden van alle objecten op de canvas, om vervolgens te bepalen op welke pixels deze objecten invloed hebben. In dit geval wordt uitgegaan van de punten $\mathbf{p}$ en worden de punten $\mathbf{p}'$ gevonden. Waar raytracing algoritmes dus beeld centrisch zijn, zijn rasterisatie algoritmes object centrisch. Hierbij wordt in de buitenste loop over alle objecten gelopen. En daarna per object gekeken welke pixels door dit object worden beinvloedt. Dit leidt tot de volgende pseudo code:

```
for object in canvas:
    projection = project(object, eye)

    for pixel in projection:
        do_shading(pixel, object)
```

Dit is tevens afgebeeld in figuur 2.11.

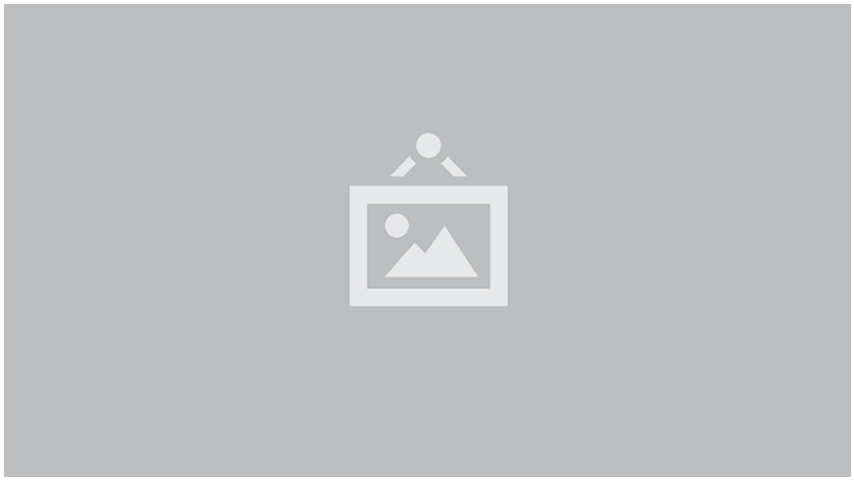


Figuur 2.11: Het rasterisatie algoritme.

Om een enkele primitief dus af te beelden op het canvas dient eerst, voor elke hoek van dit primitief de perspectief deling uitgevoerd te worden. Hierna dient het resultaat omgezet te worden naar raster-ruimte, zodat de punten binnen pixels vallen. Vervolgens dienen de pixels overlopen te worden, om na te gaan of deze binnen of buiten het object valt of niet. Dit leidt uiteindelijk tot een set van $\mathbf{p}'$, i.e. een set van pixels, die behoren tot het object. Om deze pixels efficient te overlopen, wordt meestal een bounding box in raster-ruimte gecreeerd. Slechts voor de pixels binnen deze bounding box wordt nagegaan of het object behoort tot hen of niet. Dit is weer gegeven in figuur ...

Hiermee is vastgesteld dat de oplossing voor de perspectief projectie bestaat uit twee simpele stappen, die goedkoop uit te rekenen zijn. Echter, dit lost nog niet het visibiliteitsprobleem op, doordat het mogelijk is dat verschillende objecten op het punt $\mathbf{p}'$ worden afgebeeld. Om het visibiliteitsprobleem op te lossen zijn verschillende algoritmes voorgesteld. In het volgende stuk zal het z-buffer algoritme besproken worden. Dit is het algoritme waar grafische kaarten gebruik van maken, en is daarom van belang.

Zoals opgemerkt bij de bespreking van het visibiliteitsprobleem, is dit intrinsiek een sorteer probleem, waarbij objecten geordend dienen te worden ten opzichte van de kijkas, de **z**-as. Om het zichtbare object binnen een punt $\mathbf{p}'$ te bepalen, dient dus bepaald te worden welk object de kleinste **z**-as waarde heeft ten opzichte van het oogpunt. De oplossing voor dit probleem is dan ook simpel. Voor elke pixel wordt de kleinste gevonden **z**-as waarde bijgehouden in een corresponderende twee dimensionale array. Deze array wordt een z-buffer, of een diepte-buffer (depth-buffer) genoemd. Wanneer een pixel gevonden wordt met een kleinere **z**-waarde, wordt zowel het object in punt $\mathbf{p}'$ als de nieuwe diepte bijgewerkt. Wanneer alle objecten overlopen zijn zal er dus per pixel bekend zijn welke objecten gebruikt dienen te worden om de shading berekening uit te voeren.



FIGUUR 2.12: Uitstraling van radiantie over ω_o vanuit $\mathbf{p}$

2.4 Shading

In de voorgaande secties is besproken hoe visibiliteit opgelost kan worden. Echter dit is slechts de eerste stap in het genereren van beelden. Nu vastgesteld is welke vorm objecten in de scene hebben, en welke delen van objecten daadwerkelijk zichtbaar zijn, is het tevens nodig om te bepalen hoe deze objecten er uit zien. Shading is het proces waarbij vergelijkingen worden gebruikt om te bepalen welke kleur punten dienen te hebben. Hierbij wordt verder gebouwd op de kennis van sectie .... In deze sectie zal een mathematische beschrijving worden gegeven van shading.

2.4.1 Mathematische modelering

De belangrijkste vergelijking binnen computer graphics is de render vergelijking (rendering equation):

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{2\pi^+} f_r(\mathbf{p}, \omega_i, \omega_o) L_i(\mathbf{p}, \omega_i) \cos\theta_i d\omega_i$$

Hier weergegeven in hemisfeer vorm. Deze vergelijking toont stabiele toestand van de stralingsenergie balans binnen een scene. Hierbij is $L_o((p), \omega_o$ de radiantie uitgezonden vanuit punt $\mathbf{p}$ over ω_o. Deze radiantie kan gedefinieerd worden aan de hand van de som van gereflecteerde radiantie, en de radiantie die door het punt $\mathbf{p}$ zelf wordt uitgestraald. De uitgestraalde radiantie is $L_e(\mathbf{p}, \omega_o)$. De reflectie van radiantie wordt beschreven door het tweede deel van de render vergelijking:

$$L_o(\mathbf{p}, \omega_o) = \int_{2\pi^+} f_r(\mathbf{p}, \omega_i, \omega_o) L_i(\mathbf{p}, \omega_i) \cos\theta_i d\omega_i$$

Dit wordt de reflectie vergelijking genoemd. Hierbij wordt geintegreerd over de gehele hemisfeer om de volledige binnenkomende radiantie te berekenen. Vervolgens

specificeert een zogenoemde bidirectionele reflectie distributie functie (bidirectional reflectance distribution function BRDF), hoe de radiantie over een bepaalde ruimtehoek ω_i bijdraagt aan de uitgaande radiantie in punt $\mathbf{p}$ over ruimtehoek ω_o. Hiermee kan precies worden vastgelegd wat de uitgaande radiantie is in punt $\mathbf{p}$ over ruimtehoek ωo. Dit is verder geillustreerd in fig. **??**.

De BRDF in essentie is de wiskundige functie die beschrijft hoe een materiaal zich gedraagd ten opzichte van licht. Deze functies hebben een aantal eigenschappen die gebruikt kunnen worden bij de berekening van de kleur van een punt.

Helmholtz reciprociteit De waarde van een BRDF blijft gelijk indien ω_i en ω_o worden omgedraaid.

$$f_r(\mathbf{p}, \omega_i, \omega_o) = f_r(\mathbf{p}, \omega_o, \omega_i)$$

Lineariteit De totale gereflecteerde radiantie is gelijk aan de som van alle BRDFs op dit specifieke punt. Hierdoor wordt het mogelijk om een materiaal voor te stellen met meerdere BRDFs in hetzelfde punt.

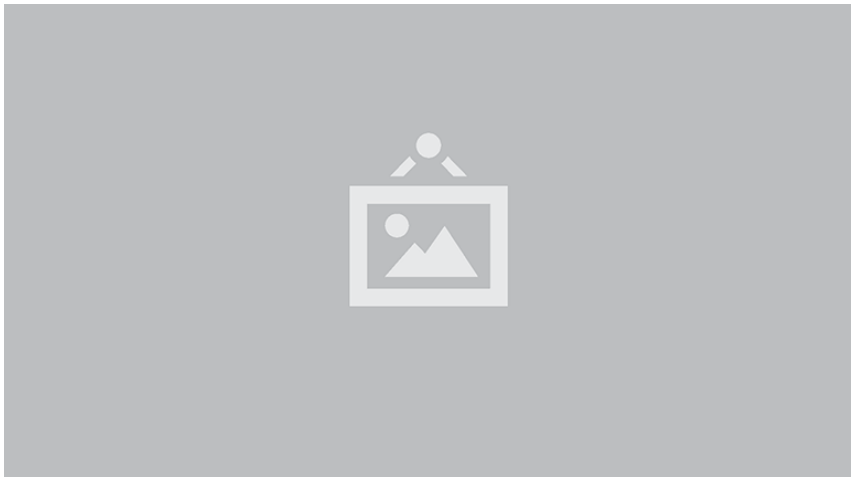
Conservatie van energie De totaal ingevallen radiantie is gelijk aan de som van het uitgezonden licht, en het geabsorbeerde licht. Dit houdt in dat L_o niet groter dan 1 kan zijn over alle ω_o.

Het is nu mogelijk om een beschrijving van de kleur in elk punt te geven aan de hand van de radiantie die berekend kan worden met de rendering vergelijking. Hierbij zullen de materialen van objecten beschreven zijn met BRDFs. Echter er is hier nog wel een groot probleem. De radiantie die uitgezonden wordt vanuit $\mathbf{p}$ over ω_o is afhankelijk van alle radiantie binnenkomend over de gehele hemisfeer in punt $\mathbf{p}$. De binnenkomende radiantie is gelijk aan de radiantie uitgezonden vanuit alle punten op de hemisfeer, volgens de Helmholtz reciprociteit. Als gevolg heeft dit dat om de radiantie te berekenen, het nodig is om alle radiantie in de scene al van te voren te weten. Dit is niet mogelijk, en dus zullen alle shading algoritmes pogen een benadering te geven van de daadwerkelijke oplossing van de rendering vergelijking. De kwaliteit van de benadering hangt af van meerdere aspecten, een grote beperkende factor binnen real-time graphics is de beschikbare rekentijd.

2.4.2 Lambertiaanse Bidirectionele Reflectie Distributie Functie

Materialen kunnen gedefinieerd worden als set van BRDFs, die het gedrag van het licht beschrijven indien het in contact komt met een object. De simpelste BRDF is de lambertiaanse BRDF. Deze BRDF beschrijft een puur diffuus oppervlakte, wat inhoudt dat de richting waarin een binnenkomende straal licht wordt gereflecteerd puur willekeurig is. Dit is weergegeven in fig. **??**. Deze BRDF heeft als uitkomst een constante waarde. Deze constante waarde wordt veelal gedefinieerd als de de *diffuse kleur* $c_{\texttt{dif}}$ van dit object. Dit leidt tot de volgende functie:

$$f(\omega_i, \omega_o) = \frac{c_{\texttt{dif}}}{\pi}$$



FIGUUR 2.13: Lambertiaanse BRDF.

Hierbij is de deling door π een gevolg van de integratie van de cosinus factor over de hemisfeer.

De lambertiaanse BRDF is als standaard materiaal gebruikt binnen deze thesis. Indien niet anders vermeld zullen afbeeldingen en testen gegeneerd zijn met deze functie.

2.4.3 Definitie van licht

Zoals eerder benoemd, draait de kern van deze thesis om het optimaliseren van het aantal lichtberekeningen in real-time toepassingen. Om deze reden is het belangrijk om het concept licht zoals gebruikt in deze thesis te definieren. Wanneer er gesproken wordt van een licht, of een lichtbron zal altijd gedoeld worden op een eindige puntlichtbron die zich bevindt op punt $\mathbf{p}$ binnen de scene.

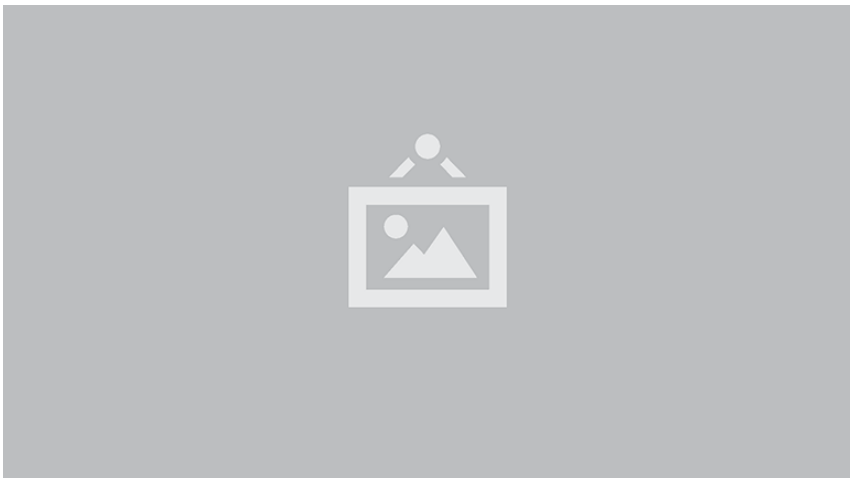
In de fysische wereld zijn lichten nooit eindig, echter de invloed die ze hebben op de omringende wereld zal bij grotere afstand 0 benaderen. Binnen de fysische wereld is dit een gevolg van absorptie door het medium waardoor het licht zich beweegt. Dit proces wordt afstands demping genoemd. Binnen de physica wordt deze relatie vastgelegd met de wet van Lambert-Beer gedefinieerd voor uniforme demping als.

$$T = e^{-\mu l}$$

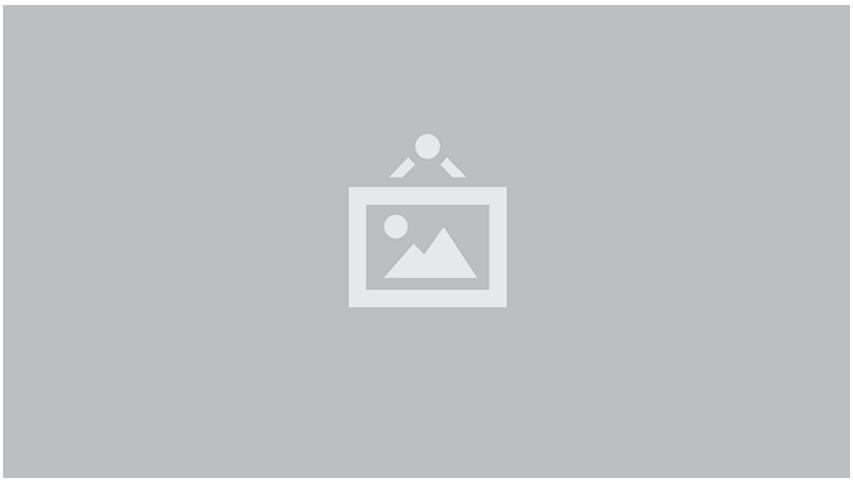
waar l de padlengte van de straal licht door het medium is en μ de dempingscoefficient is.

Hierbij wordt de demping van het licht gerelateerd aan het medium waardor het zich beweegt. Dit leidt tot afstandsdempings (distance attenuation) curves zoals weergegeven in figuur 2.14.

Binnen veel real-time rendering toepassingen wordt afgestapt van dit fysische model. Er wordt gebruikt gemaakt van een eindige benaderingen van deze lichtbronnen. Waarbij wordt gesteld dat het licht geen invloed meer heeft na afstand r.



Figuur 2.14: Afstandsdempings curves.



Figuur 2.15: Voorstelling van licht.

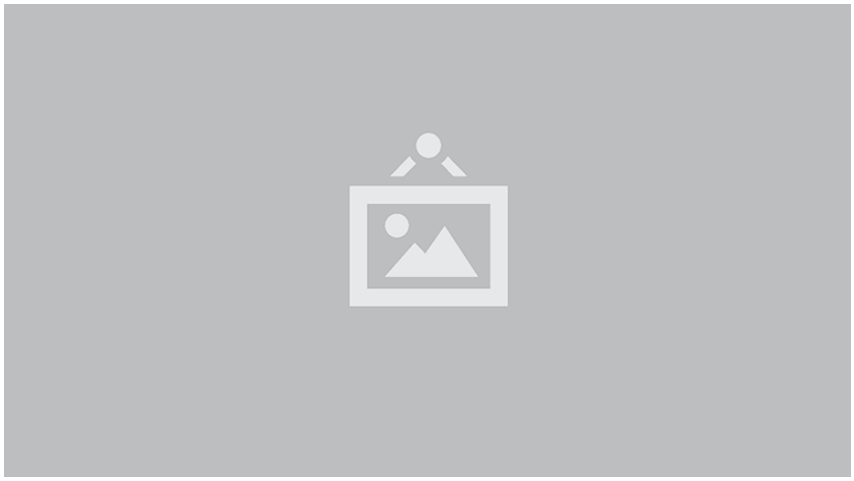
Dit leidt tot een voorstelling als weergegeven in figuur 2.15.
Hierbij is de invloed in de oorsprong gelijk aan 1, en op afstand r en groter 0.

Echter om de illusie te wekken dat de lichten fysiek accuraat zijn, dient tevens een benadering gemaakt te worden van de afstandsdempings functies. Deze functies dienen te voldoen aan de eerder gestelde voorwaarde, waarbij de invloed een is in de oorsprong, en nul op afstand r. Enkele veel gebruikte benaderingen als wel de daadwerkelijke afstandsdemping zijn gegeven in figuur 2.16

Binnen de thesis zelf is gekozen voor de benadering:

$$(\frac{l}{r}\Big|_{[0,1]})^2$$

Als laatste dienen we intensiteit van de lichtbron vast te leggen. Zoals gebruikelijk



FIGUUR 2.16: Afstandsdempings curves voor eindige lichtbronnen.

binnen computer graphics, is hier gekozen voor een rgb voorstelling. Waarbij de waardes zich bevinden in het bereik van 0 tot en met 1.

Dit alles leidt ertoe dat we een lichtbron kunnen definieren als de set van de volgende eigenschappen:

- De positie **p** van het licht ten opzichte van een coordinatenstelsel met oorsprong O
- Een afstand r die de invloed van de lichtbron bepaald
- Een afstandsdempingsfunctie f die het verval van invloed moduleert
- Een intensiteit **i** die de kleur en kracht van de lichtbron bepaald

2.5 Moderne Grafische Pipeline

Een belangrijk aspect in het ontwerpen van efficiente real-time grafische algoritmes is de verwerking van beelden op de grafische kaart. Moderne grafische kaarten maken gebruik van een zogenoemde programmeerbare pipeline (programmable pipeline). Dit houdt in dat verscheidene onderdelen van de grafische pipeline gedefinieerd kunnen worden door de programmeur.

Huidige pipelines kunnen gebruik maken van verschillende `APIs` om deze programmeerbare pipeline aan te spreken. De twee meest gebruikte industrie standaarden zijn `openGL` en `Direct3D`.

In deze sectie zal ingegaan worden op de verschillende aspecten van de moderne grafische pipeline. Eerst zal conceptueel ingegaan worden op de conceptuele stappen van de pipeline. Daarna zal gekeken worden naar hoe deze conceptuele stappen daadwerkelijk onderverdeeld zijn. Hiervoor zal gebruik gemaakt worden van de naamgeving zoals deze binnen `openGL` is gedefinieerd.

2.5.1 Pipeline

De `openGL` pipeline bestaat uit 7 stappen, die achtereen volgens worden uitgevoerd. Deze stappen zijn weergegeven in figuur. Hierbij zijn de vaste-functie stappen weergegeven in De programmeerbare stappen zijn weergegeven in

2.5.2 Vertex Specificatie

Vertex Shader

Tessellation

Geometrie Shader

2.5.3 Vertex Verwerking

2.5.4 Vertex Nabewerking

2.5.5 Opstelling Primitieven

2.5.6 Rasterisatie

2.5.7 Fragment Verwerking

2.5.8 Per-Sample Operaties

Hoofdstuk 3

Methode

Hoofdstuk 4

Implementatie

Hoofdstuk 5

Resultaten

Hoofdstuk 6

Discussie

Hoofdstuk 7

Conclusie

Fiche masterproef

Student: Martinus Wilhelmus Tegelaers

Titel: realtime renderen van vele lichtbronnen

Engelse titel: Realtime rendering of many light sources

UDC: 621.3

Korte inhoud:

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Mens-machine communicatie

Promotor: Prof. dr˙ ir. P. Dutre

Assessor: T. Do

Begeleider: T.O. Do