Dankwoord

Bij deze wil ik mijn promotor, Philip Dutré, bedanken om mij de mogelijkheid te geven tot deze eindverhandeling. Hij heeft mij de wondere wereld van computergrafieken leren kennen, waar ik ondertussen mijn informatica-hart ben verloren. Nooit was en werd ik zo gemotiveerd.

Ook dank ik mijn begeleider, Olivier Dumont. Hij heeft mij voldoende ruimte gelaten mijn eigen weg te volgen, maar waar ik uit de bocht ging heeft hij perfect weten bij te sturen. Ik heb enorm veel aan hem gehad.

Mijn ouders wil ik bedanken om mij de kans te geven tot een universitaire studie, om hun steun en hun geloof in mij.

Valerie en Nelis wil ik bedanken voor hun hulp aan deze thesis; Valerie voor het bijsturen van de grammaticale correctheid en structuur van deze tekst, Nelis voor de fantastische ideeën en gesprekken en de technische correctie van deze tekst.

Als laatste wil ik ook Valerie, Nelis, Maarten en Bruno bedanken voor hun aandacht, hulp en interesse wanneer ik weer eens met overenthousiasme, dan weer met frustraties kwam aandraven.

Verder wens ik u veel leesgenot toe.

Inhoudsopgave

Hoofdstuk 1

Inleiding

> *The best way to become acquainted with a subject is to write a book about it.*
>
> - Benjamin Disraeli (1804 - 1881)

Beeldsynthese binnen het domein van computergrafieken is een discipline die toepassingen kent in verscheidene gebieden zoals onder andere de architectuur, de medische wereld, computer animatie en computer spellen.

Radiositeits algoritmen, rasterizers en ray tracing zijn naast andere methoden om aan beeldsynthese te doen. De ray tracing techniek werd voor het eerst voorgesteld door T. Whitted [Whi80] in 1980 en is tot op vandaag onderwerp van actief onderzoek.

Ray tracing maakt onder andere de berekening van de belichting in een synthetische omgeving mogelijk op algoritmisch relatief eenvoudige wijze. Hoewel de de implementatie van de berekening vrij eenvoudig is, vraagt de berekening zelf een grote rekenkracht.

Deze rekenkracht vormt één van de grootste knelpunten van ray tracing. Velerlei technieken bestaan om dit aspect te optimaliseren gaande van ruimtelijke structuren om het verzamelen van de gegevens voor de berekening te vereenvoudigen tot methoden die de berekening zelf vereenvoudigen zodat misschien geen volledig correct, maar wel een visueel plausibel resultaat wordt bekomen. Deze verhandeling behandelt de optimalisatie voor het verzamelen van de gegevens voor de berekening.

Deze verhandeling bestaat uit drie delen.

In een eerste deel zal de werking van het ray tracing algoritme besproken worden, waarna een overzicht van enkele standaard optimalisaties voor de

berekening samen met een aantal recente ontwikkelingen op dat gebied volgt.

Eén van de recente ontwikkelingen, lokale belichtingsomgevingen, vormt de basis voor de techniek die in deze verhandeling uitgewerkt wordt. Het concept van lokale belichtingsomgevingen wordt overgenomen. De wijze waarop de constructie van deze structuur verloopt wordt in deze verhandeling anders bekeken. Daarnaast worden een aantal mogelijkheden voor deze structuur toegevoegd.

Het concept van Local Illumination Environments (LIE's) of lokale belichtingsomgevingen, zoals deze voorgesteld werden in [FBG02], wordt eerst besproken, waarna het nieuwe concept, Preprocessed Local Illumination Environments (P-LIE's), of voorberekende lokale belichtingsomgevingen, voorgesteld wordt.

Het tweede deel behandelt de constructie van de voorberekende lokale belichtingsomgevingen die in detail besproken wordt. Deze constructie omvat een combinatie van verscheidene methoden die reeds werden voorgesteld in het eerste deel.

In het derde deel wordt het ray tracen met behulp van voorberekende lokale belichtingsomgevingen uitvoerig getest. Daarbij is de doelstelling om een zo groot mogelijke versnelling voor de berekening van de belichting te behalen. In het afsluitende hoofdstuk wordt, op basis van de resultaten van deze testen, een evaluatie gegeven van de P-LIE-structuur.

Als laatste wordt een algemeen besluit en enkele mogelijkheden tot verder onderzoek in verband met (al dan niet voorberekende) lokale belichtingsomgevingen gegeven.

Deel I

Ray tracing en directe belichting

Hoofdstuk 2

Ray tracing

> *De vraag die waarschijnlijk het meest gesteld wordt aan een student in het laatste jaar van zijn/haar opleiding is: 'Waarover gaat je thesis?'. Telkens ik deze vraag krijg tracht ik in zo min mogelijk technische taal een korte samenvatting te geven. Daarbij vertel ik eerst kort hoe ray tracing ongeveer werkt.*
>
> *Iemand zei ooit, na mijn uitleg gehoord te hebben: 'Aah, een beetje zoals een schilderij dus.'*

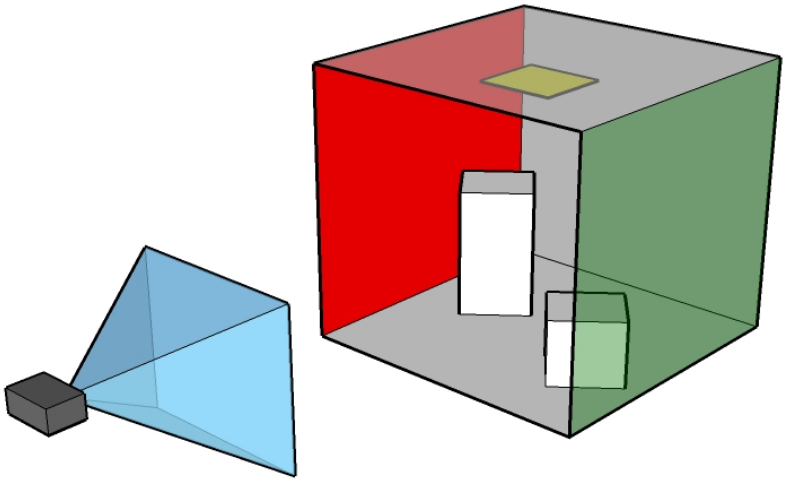
Ray tracing is een techniek die specifiek binnen computergrafieken gebruikt wordt om driedimensionele, synthetische scènes te visualiseren. Dit hoofdstuk geeft een overzicht van de verschillende concepten van deze techniek.

Ten eerste wordt de synthetische scène gedefinieerd, die gevisualiseerd moet worden. Daarna worden de begrippen "licht", "reflectie" en "schaduw" in meer detail bekeken. Als laatste wordt het ray tracing proces om een scène te visualiseren besproken.

De inhoud van dit hoofdstuk is grotendeels gebaseerd op de twee boeken [SM03] en [DBB03]. Een uitgebreide, maar compact beschreven, samenvatting van de in dit hoofdstuk besproken en andere technieken wordt gegeven in het Global Illumination Compendium [Dut03].

2.1 De scène

Een voorbeeld van een mogelijke opzet voor een scène is weergegeven in figuur 2.1. Een scène bestaat uit drie elementen: de geometrie, de lichtbronnen en een camera.



Figuur 2.1: Een voorbeeld van een mogelijke scène.

Figuur 2.2: Een triangle-mesh die een bootje voorstelt.

2.1.1 Geometrie

In een driedimensionele ruimte zijn een aantal objecten bestaande uit één of meerdere meetkundige primitieven geplaatst die samen de geometrie van de scène vormen. Typische objecten zijn bollen, cylinders, kubussen, vlakken en dergelijke. Meer complexe objecten worden aan de hand van meerdere primitieven, meestal driehoeken, samengesteld. Deze zogenaamde "triangle meshes" worden gedefinieerd door een aantal punten, die elk een geassocieerde normaal hebben. Door de connectiviteit tussen de punten te bepalen worden de driehoeken gevormd. Een voorbeeld van een triangle mesh staat afgebeeld in figuur 2.2

2.1.2 Lichtbronnen

In de scène zijn een aantal lichtbronnen geplaatst. Verschillende soorten lichtbronnen worden gebruikt in computergrafieken:

1. *Puntlichtbronnen* zijn bepaald door een positie in de ruimte en stralen hun licht vanuit dit punt in alle richtingen rond.

2. *Spotlichtbronnen*: naast een positie in de ruimte hebben deze een gerichte kegel van richtingen waarin het licht wordt uitgestraald.

3. *Directionele lichtbronnen* nemen geen positie in, maar zijn gedefinieerd aan de hand van een richting. Het licht komt gelijkmatig vanuit één richting ingevallen op scène. Dit licht kan vergeleken worden met het zonlicht.

4. *Lineaire lichtbronnen* zijn bepaald door een lijnstuk in de ruimte, waarbij het licht in alle richtingen rond het lijnstuk word gestraald.

5. *Vlakke lichtbronnen* bestaan uit een polygon en stralen hun licht uit in alle richtingen binnen een hoek van 90 graden met de normaal van de polygon. Veelal zijn dit rechthoeken.

6. *Volume lichtbronnen* hebben een bepaald ruimtelijk object als vorm. De manier waarop licht hierdoor wordt uitgestraald is daardoor afhankelijk van de vorm, maar meestal gebeurt dit op dezelfde wijze als bij vlakke lichtbronnen.

Het is niet noodzakelijk dat een lichtbron zijn licht uniform in alle richtingen uitstraalt. Meer zelfs, in realiteit zijn er maar weinig tot geen lichtbronnen die dit effectief doen. Hoewel verschillende mogelijkheden voor het modelleren van de lichtdistributie bestaan, is het in een computermodel gemakkelijker om te werken met uniform stralende, zogenaamde diffuse lichtbronnen. In deze verhandeling zal uitsluitend gebruik gemaakt worden van diffuse vlakke lichtbronnen.

2.1.3 De camera

Om een beeld te kunnen genereren is een camera aanwezig die bepaalt hoe de scène zal worden beschouwd. Die camera wordt gedefinieerd door een positie in de ruimte en een “venster”. De camera kijkt vanuit het oogpunt door het venster naar de scène. Het venster komt overeen met het beeld dat uiteindelijk wordt gegenereerd.

2.2 Belichting in de scène

In de scène moet er licht aanwezig zijn opdat er iets te zien zou zijn. Daarom is een eenheid nodig waarin kan uitgedrukt worden hoeveel licht een lichtbron uitstraalt.

De objecten in de scene reflecteren het licht dat op hen invalt. Daarom is er nood aan een model die dit kan voorstellen.

Eerst zal de modellering van licht worden besproken, daarna de modellering van reflectie.

2.2.1 Licht

In oudere ray tracing algoritmen en conventionele hardware gebaseerde renderers wordt licht gemodelleerd als een kleur in de RGB-kleurenruimte. Dit betekent dat kleur moet uitgedrukt worden in een tuple van drie natuurlijke waarden tussen 0 en 255. Dit volstaat om mooie afbeelding te genereren. Wanneer fysische realiteit beoogt wordt, schiet RGB tekort; er is nood aan reële getallen.

In computergrafieken wordt over het algemeen het fysische geometrische model van licht gebruikt. Dit model geeft ons enkele eigenschappen over het gedrag van licht in de ruimte:

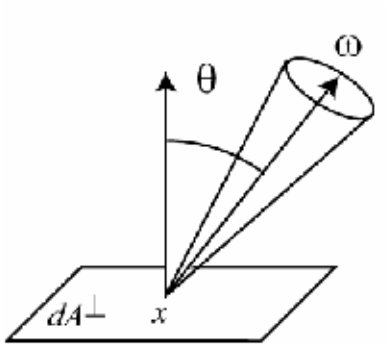
1. Licht kan uitgestraald worden en ondergaat reflectie en transmissie.
2. Licht propageert zich in de ruimte langs rechte lijnen.
3. Het licht wordt beschouwd in een evenwichtstoestand.

Een eenheid waarin lichtenergie kan worden uitgedrukt is radiant vermogen (P), ook wel flux genoemd. De eenheid van flux is Watt(W) en staat voor de hoeveelheid energie dat wordt uitgestraald/gereflecteerd door een oppervlak per eenheid tijd (sec).

Irradiantie (E) stelt de hoeveelheid invallend radiant vermogen op een oppervlak voor, per eenheid oppervlak (m^2).

$$E = \frac{dP}{dA}$$

Radiantie (L) stelt de hoeveelheid invallende of uitgaande flux, per eenheid ruimtehoek (sterradian), per eenheid oppervlak ($W/(sterradian * m^2)$), in een punt voor. In figuur 2.3 staat de configuratie voor radiantie afgebeeld.



Figuur 2.3: Radiantie in een punt x op een oppervlak A. Bron: [DBB03],

$$L = \frac{d^2P}{dA\, d\omega\, cos\theta} \tag{2.1}$$

In principe staat het oppervlak A loodrecht op de richting waarvoor de radiantie wordt uitgedrukt. Indien de richting niet onder een rechte hoek staat met het oppervlak wordt de flux verspreid over een iets groter oppervlak. Om dit in rekening te nemen is de cosinusterm toegevoegd in formule 2.1.

Wanneer men over de radiantie vanuit een punt x richting een ander punt y spreekt, dan wordt dat als volgt genoteerd:

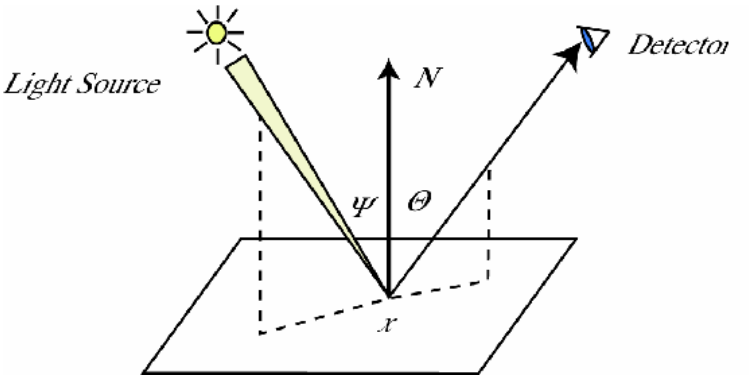
$$L(x \rightarrow y)$$

Radiantie vanuit een punt x in een richting Θ noteert als volgt:

$$L(x \rightarrow \Theta)$$

Radiantie heeft verder de belangrijke eigenschap dat het invariant blijft langs rechte paden. De radiantie met andere woorden, die vertrekt vanuit een bepaald punt in een bepaalde richting, is dezelfde als de radiantie die toekomt in een ander punt langs dezelfde richting.

$$L(x \rightarrow \Theta) = L(\Theta \rightarrow y)$$



Figuur 2.4: De BRDF in een punt x. Bron: [DBB03],

2.2.2 Reflectie

Licht, dat via een bepaalde richting Ψ in een bepaald punt op een object invalt, wordt vanuit een punt langs een bepaalde richting Θ terug gereflecteerd. In realiteit is het punt waar het licht invalt op een object niet noodzakelijk hetzelfde punt van waaruit het licht terug wordt gereflecteerd. Hier wordt aangenomen dat dit toch het geval is. Licht verlaat een oppervlak daar waar het toegekomen is.

De fractie van het licht, invallend in een punt in een bepaalde richting, dat vanuit datzelfde punt in een een bepaalde richting wordt gereflecteerd, is gegeven door de bidirectionele reflectie distributie functie (BRDF). Dit is de ratio van de differentieel uitgaande radiantie langs een richting Ψ en de differentieel invallende irradiantie langs een richting Θ.

$$\begin{aligned} f_r(x, \Psi \rightarrow \Theta) &= \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} \\ &= \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi} \end{aligned}$$

De term N_x staat voor de normaal van het oppervlak in het punt x.

Een aantal eigenschappen zijn:

- De waarde van de BRDF is groter dan of gelijk aan nul.
- Reciprociteit: de waarde van de BRDF verandert niet indien de twee gegeven richtingen worden omgewisseld;

$$f_r(x, \Psi \rightarrow \Theta) \quad = \quad f_r(x, \Psi \leftarrow \Theta)$$

Omwille van deze eigenschap wordt ook vaak de volgende notatie gebruikt:

$$f_r(x, \Psi \leftrightarrow \Theta) \tag{2.2}$$

Er bestaan verschillende types BRDF's. De waarschijnlijk meest gebruikte is de BRDF voor diffuse oppervlakken. Diffuse materialen zullen het invallende licht gelijkmatig reflecteren over alle omliggende richtingen. Dit maakt dat de BRDF een constante wordt:

$$f_r(x, \Psi \leftrightarrow \Theta) \quad = \quad \frac{\rho_d}{\pi}$$

Hierbij geeft ρ_d de fractie aangeeft van het invallend licht dat wordt gereflecteerd.

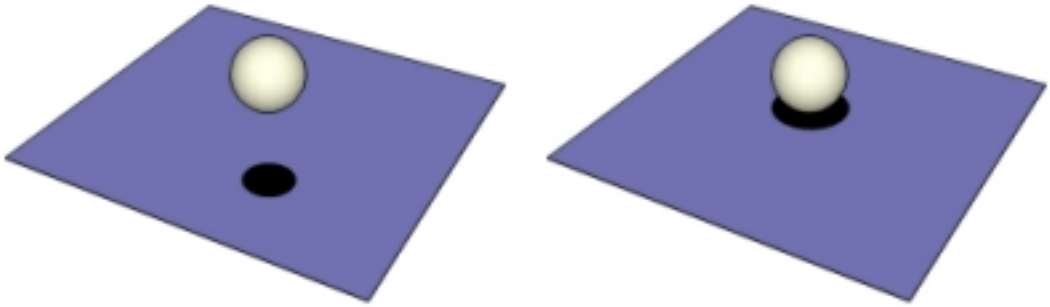
Diffuse materialen zijn eenvoudig om mee te werken, maar in realiteit komen ze niet voor in de pure vorm van een constante. Er bestaan wel materialen die dicht in de buurt komen, bijvoorbeeld een mat vel papier.

Andere modellen worden hier verder niet besproken; er zal in deze verhandeling enkel gewerkt worden met de diffuse BRDF. Voor een uitgebreide beschrijving van de verschillende mogelijke BRDF's, zie [DBB03] en het Global Illumination Compendium [Dut03].

2.2.3 Schaduwen

Schaduwen zijn in synthetische beelden van essentieel belang. Een beeld is niet volledig als er geen schaduwen aanwezig zijn. Ze geven belangrijke visuele hints over de relatieve positie van objecten ten opzichte van elkaar. Een voorbeeld is weergegeven in figuur 2.5. In de linker afbeelding lijkt de bol boven het vlak te zweven, terwijl in de rechter afbeelding de bol ertegen is geplaatst. Het vlak en de bol hebben nochtans dezelfde positie in het beeld. Het is de schaduw die aangeeft waar ze zich in drie dimensies ten opzichte van elkaar bevinden.

Er worden twee soorten schaduwen onderscheiden:



Figuur 2.5: Het belang van schaduwen in een scène: links lijkt de bol te zweven, terwijl rechts de bol op het vlak lijkt te liggen.

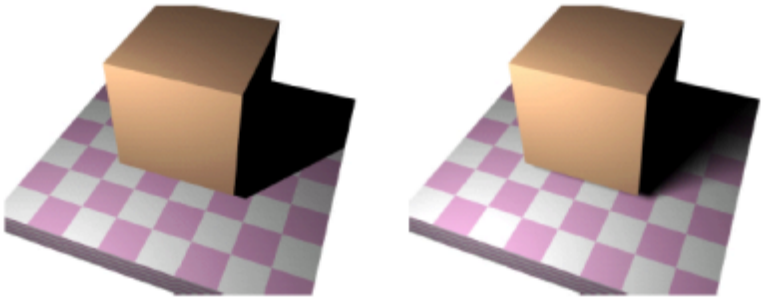
1. *Harde schaduwen*; bijvoorbeeld links in figuur 2.6. Deze schaduwen hebben een abrupte grens tussen het belichte en het in de schaduw liggende deel van een oppervlak. Het deel van een volledige schaduw dat de harde schaduw vormt wordt de umbra regio genoemd.

2. *Zachte schaduwen*; bijvoorbeeld rechts in figuur 2.6. Deze schaduwen hebben een geleidelijke overgang van het belichte naar het in de schaduw liggende deel van een oppervlak. Het deel van een volledige schaduw dat de zachte schaduw vormt wordt de penumbra regio genoemd.

Alhoewel harde schaduwen in realiteit niet voorkomen, worden ze vaak gebruikt in computergrafiektechnieken en dan vooral bij oudere hardware gebaseerde renderers. Zij werken immers hoofdzakelijk met nul-dimensionale lichtbronnen (punt-, directionele of spotlichtbronnen) die daardoor geen zachte schaduw kunnen afwerpen.

In ray tracing bestaat de mogelijkheid om met één-, twee- en driedimensionale lichtbronnen te werken die wel zachte schaduwen afwerpen.

Harde schaduwen geven al een goede hint over de relatieve positie van de objecten in de scène, maar kunnen in bepaalde situaties subtiele details missen. Dit is bijvoorbeeld het geval in figuur 2.6: de positie van de kubus wordt pas correct weergegeven indien zachte schaduwen gebruikt worden. Wanneer visuele realiteit wordt nagestreefd, is het noodzakelijk ook zachte schaduwen te kunnen visualiseren.



Figuur 2.6: Het belang van zachte schaduwen in een scène, links een scène gevisualiseerd met harde schaduwen, rechts dezelfde scène met zachte schaduwen. Links lijkt de kubus op het vlak te liggen, terwijl de kubus in het rechtse beeld duidelijk boven het oppervlak zweeft lijkt te zweven. Bron: [HDG99]

2.3 Ray tracing en directe belichting.

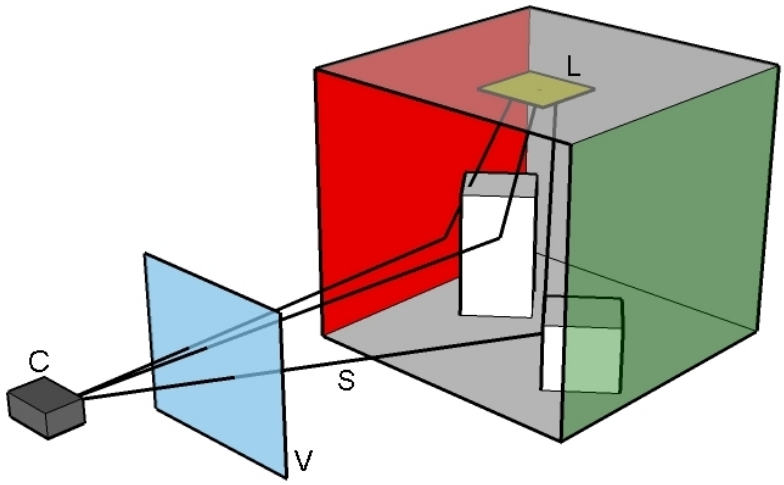
De opzet voor het ray tracing proces staat afgebeeld in figuur 2.7

Wanneer de scène gedefinieerd is, wordt het ray tracing proces gestart. Het venster (V) van de camera (C) wordt gediscretiseerd over een aantal punten. Voor elk punt van het venster, dat overeenkomt met een pixel van het te genereren beeld, wordt vanuit het oogpunt van de camera een zichtstraal (S) in de scène getrokken. Het eerste object in de scène waarmee de zichtstraal snijdt, is het object dat gezien wordt via die straal en dus eveneens in de bijhorende pixel.

Voor elk gevonden snijpunt wordt de radiantie die gereflecteerd wordt naar de camera volgens de richting van de zichtstraal bepaald. Eens de radiantie door een punt van het venster berekend is, kan die radiantiewaarde gemapt worden op een RGB-waarde die gebruikt wordt in de overeenkomstige pixel in het uiteindelijke beeld.

2.3.1 De renderingvergelijking

Om de radiantie vanuit een punt in een bepaalde richting te kunnen berekenen wordt de renderingvergelijking (RE) geëvalueerd. Hiervoor bestaan verschillende formulaties; de voor ons relevante vorm is de renderingvergelijking over de oppervlakken in de scène:



Figuur 2.7: Opzet voor ray tracing.

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_A f_r(x, \Psi \rightarrow \Theta) L(y \leftarrow -\Psi) V(x,y) G(x,y) dA_y \tag{2.3}$$

In woorden: de uitgaande radiantie is de uitgestraalde radiantie plus de gereflecteerde radiantie. De gereflecteerde radiantie vormt een integraal over alle oppervlakten in de scène. Hierin komt onder andere de BRDF f_r voor.

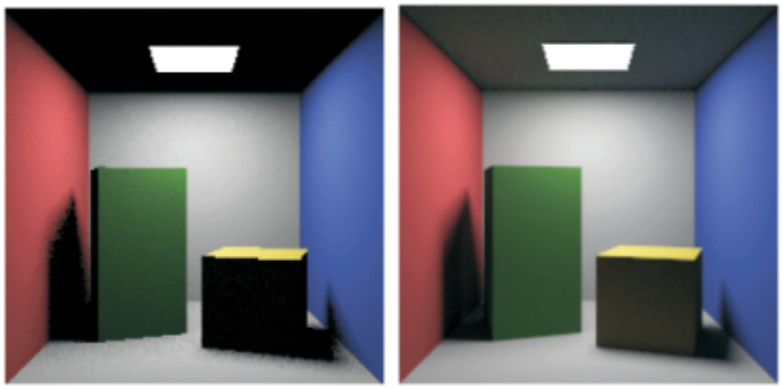
Een andere veranderlijke in de integrand is y: dit is een punt op het oppervlak waarover wordt geïntegreerd. Het punt y wordt gevonden door een straal vanuit x volgens de richting Ψ te trekken en te snijden met het oppervlak waarover wordt geïntegreerd.

De term $L(y \leftarrow -\Psi)$ geeft de invallende radiantie in x vanuit het punt y langs de invallende richting Ψ.

De term $V(x, y)$ is de zichtbaarheidsfunctie. Zoals de naam aangeeft,geeft deze functie de zichtbaarheid tussen de twee punten x en y. Deze geeft de waarde 1 indien beide punten zichtbaar zijn voor elkaar en 0 indien de zichtbaarheid tussen beide punten wordt verhinderd.

De term $G(x, y)$ wordt de geometrie term genoemd en is als volgt gegeven:

$$G(x,y) = \frac{cos(N_x, \Psi) cos(N_y, \Psi)}{r_{xy}^2}$$



Figuur 2.8: Links een scène met enkel directe belichting, rechts met de indirecte belichting erbij.

Deze term brengt de relatieve oriëntatie van de twee vlakken (het vlak waar x in ligt en het vlak waar y in ligt) en hun onderlinge afstand in rekening. De noemer in de deling is het kwadraat van de afstand tussen de twee punten.

Vergelijking 2.3 geeft de radiantie vanuit een punt x in een richting Ψ als een integraal over alle oppervlakken in de scène, en behandelt dus alle lichttransport.

Het volledig invallend licht kan worden opgedeeld in twee types:

1. *directe belichting*: beslaat al het licht dat rechtstreeks van de lichtbron invalt in het punt, en

2. *indirecte belichting*: is het licht dat van de lichtbron, reflecterend via objecten in de scène, uiteindelijk toekomt in het punt.

In figuur 2.8 staan twee beelden van scènes, één waarbij enkel de directe belichting werd bepaald, de ander met.

De indirecte belichting wordt vaak nog niet in rekening genomen in hedendaagse renderers. Indirect licht wordt eerder gesimuleerd door “ambient” licht toe te voegen; dit is licht dat vanuit elk punt in de ruimte vertrekt, in elke richting. Globale belichtingsalgoritmen daarentegen evalueren effectief alle lichttransport en dus ook de indirecte belichting.

In deze verhandeling ligt de focus enkel op de directe belichting. Later in deze verhandeling zullen de mogelijkheden om indirecte belichting mee te rekenen behandeld worden.

Om de directe belichting los van de indirecte te kunnen evalueren moet de rendering vergelijking herschreven worden:

$$L(x \rightarrow \Theta) \quad = \quad L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta)$$

Waarbij $L_r(x \rightarrow \Theta)$ de gereflecteerde radiantie is in x langs Θ:

$$\begin{aligned} L_r(x \rightarrow \Theta) &= \int_A f_r(x, \Psi \rightarrow \Theta) L(y \leftarrow -\Psi) V(x,y) G(x,y) dA_y \\ &= L_{direct} + L_{indirect} \end{aligned}$$

L_{direct} en $L_{indirect}$ zijn als volgt gegeven:

$$L_{direct} = \int_A f_r(x, \overrightarrow{xy} \rightarrow \Theta) L_e(y \rightarrow \overrightarrow{yx}) V(x,y) G(x,y) dA_y \tag{2.4}$$

$$L_{indirect} = \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L_i(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_y \tag{2.5}$$

met

$$L_i(x \leftarrow \Psi) \quad = \quad L_r(r(x, \Psi) \rightarrow -\Psi)$$

De indirecte belichting wordt berekend door middel van een integraal over de hemisfeer rond het punt x (formule 2.5). De term $r(x, \Psi)$ staat voor de zogenaamde “ray-cast” functie. Die geeft het dichtste punt voor x in de scène langs de richting Ψ. In de integraal voor de indirecte belichting wordt enkel de gereflecteerde radiantie in rekening gebracht.

De vergelijking voor de directe belichting (formule 2.4) daarentegen is een integraal over alle oppervlakken in de scène, waarbij enkel de uitgestraalde radiantie in rekening wordt genomen.

2.3.2 Evaluatie van de renderingvergelijking

De evaluatie van de renderingvergelijking voor de directe belichting omvat het evalueren van een integraal. Onder bepaalde beperkingen voor de scène kan deze analytisch geëvalueerd worden. Indien deze beperkingen niet kunnen worden opgelegd kan Monte Carlo integratie een uitweg bieden.

Monte Carlo integratie

Evaluatie van de integraal over een functie $f(x)$, met behulp van Monte Carlo, gebeurt door een aantal monsters (N) te nemen uit het domein (D) van de functie. Deze willekeurige waarden uit het domein worden genomen volgens een kansdichtheidsfunctie (PDF) $p(x)$. Deze is gedefinieerd zodat de kans dat een monster de waarde x heeft, gegeven is door $p(x)dx$.

De integraal

$$I = \int_D f(x)dx$$

wordt nu bepaald met behulp van de schatter

$$< I >= \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{2.6}$$

Een meer volledige behandeling over het gebruik van Monte Carlo methoden voor het evalueren van integralen wordt gegeven in [KW86].

Monte Carlo integratie van de directe belichting.

Aangezien meerdere lichtbronnen in de scène aanwezig kunnen zijn (N_L), zullen er twee PDF's bepaald moeten worden, namelijk één voor het selecteren van de lichtbron die zal bemonsterd worden en één voor het monster op die lichtbron zelf. De meest eenvoudige versies daarvoor zijn de uniforme PDF's.

Voor het selecteren van lichtbron i:

$$p_L(i) \quad = \quad 1/N_L$$

Voor het nemen van het monster:

$$p(y) \quad = \quad 1/A_{L_i}$$

waarbij A_{L_i} de oppervlakte is van lichtbron i.

De schatter voor de directe belichting wordt daardoor:

$$< L_{direct}(x \rightarrow \Theta) >= \frac{N_L}{N} \sum_{i=1}^{N} A_{L_i} L_e(y_i \rightarrow \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \leftrightarrow \Theta) G(x, y_i) V(x, y_i) \tag{2.7}$$

Om de rendering vergelijking voor de directe belichting te evalueren genereert men een aantal (N) monsters. Voor elk monster y_i wordt de oppervlakte (A_{L_i}) van de lichtbron bepaald waarop deze zich bevindt, de uitgestraalde radiantie, de BRDF, de geometrieterm en de zichtbaarheidsfunctie geëvalueerd.

De uitkomst is een benadering van de gereflecteerde radiantie door directe belichting in het punt x in de richting Θ.

De PDF voor de lichtbronnen $p_L(i)$ kan ook vervangen worden een maat die evenredig is aan de inverse van het vermogen van de lichtbron;

$$p_L(i) \quad = \quad 1/P_{L_i}$$

Daardoor zullen lichtbronnen die meer licht uitstralen meer in rekening gebracht worden dan lichtbronnen die minder licht uitstralen. Er wordt dan bemonsterd volgens de bijdrage, "importance sampling" genoemd in de literatuur. Gelijkaardige PDF's zijn ook mogelijk voor $p(y)$. Zie [Dut03] en [DBB03].

2.3.3 Focus van deze verhandeling

De termen in de schatter van de directe belichting (formule 2.7) met uniforme PDF's zijn allemaal eenvoudig te evalueren, behalve één: de zichtbaarheidsfunctie [1].

Voor de evaluatie van de zichtbaarheidsfunctie moet nagegaan worden of de twee punten x en y zichtbaar zijn voor elkaar. Dit gebeurt in het ray tracing algoritme door een straal te trekken door de punten x en y en te controleren of deze straal al dan niet snijdt met objecten in de scène tussen x en y. Indien de straal snijdt met een bepaald object en het snijpunt ligt effectief tussen x en y, dan kunnen zij elkaar niet zien; het object ligt dan op het pad tussen de twee punten.

Dit introduceert dus opnieuw een set stralen die gecontroleerd moeten worden op intersectie met de objecten in de scène. Aangezien de punten y op lichtbronnen liggen, spreekt men van schaduwstralen. Als een intersectie met een schaduwstraal gevonden wordt, ligt x in de schaduw ten opzichte van het punt y op de lichtbron.

[1] De BRDF kan ook minder eenvoudig zijn, maar met een diffuse BRDF, en dus een constante waarde, is dat niet het geval.

De intersectietest is echter qua berekeningstijd een dure operatie. Eerst zal men de test een aantal maal moeten uitvoeren voor de zichtstralen. In het ergste geval:

$$W_{venster} \times H_{venster} \times N_{objecten}$$

Hierbij is $W_{venster}$ het aantal punten in het venster van de camera is, waardoor een zichtstraal wordt getrokken in de breedte, $H_{venster}$ het aantal punten in de hoogte, en $N_{objecten}$ het aantal objecten in de scène.

Daarnaast wordt dan nog eens voor elke gevonden snijpunt, N (aantal genomen monsters), schaduwstralen gecontroleerd op intersectie. Beide gecombineerd geven dan in het ergste geval:

$$W_{venster} \times H_{venster} \times N \times N_{objecten}^2 \tag{2.8}$$

Een uitgewerkte scène kan al gemakkelijk bestaan uit tienduizenden driehoeken. Wanneer daarvan een beeld wordt gegenereerd van een behoorlijk resolutie, dan kan het aantal intersecties die uitgevoerd moeten worden hoog oplopen.

Aan de eerste drie getallen in formule 2.8 kan men niets veranderen, aan het aantal objecten dat moet worden gesneden voor een zichtstraal of een schaduwstraal echter wel. Er bestaan verschillende structuren om dit aantal zo veel mogelijk te beperken. Deze versnellingsstructuren vormen het onderwerp van het volgende hoofdstuk. In deze verhandeling zelf wordt een gelijkaardige versnellingsstructuur ontwikkeld.

2.4 Conclusie

In dit hoofdstuk werd een synthetische scène en de componenten daarin gedefinieerd. Daarbij werd ook gedefinieerd hoe licht en reflectie worden gemodelleerd.

Verder werd ray tracing als methode om een synthetische scène te visualiseren behandeld en het belang van schaduwen in de visualisatie van een scène. Daarbij is net het bepalen van een schaduw voor het ray tracing proces het grootste knelpunt. Het is belangrijk dat dit zo snel mogelijk kan gebeuren.

De voorgestelde structuur in deze verhandeling zal het bepalen van de schaduwen trachten te versnellen, specifiek de evaluatie van de zichtbaarheidsfunctie.

Hoofdstuk 3

Het RT proces versneld

Zoals besproken werd in het vorige hoofdstuk, zijn schaduwen in computergrafieken van groot belang om aan realistische beeld synthese te kunnen doen. Een algoritme dat zich daartoe leent is ray tracing.

De berekeningskost van schaduwen in standaard ray tracing ligt echter hoog. Dit is onder meer te wijten aan de intersectietesten die moeten gebeuren bij de evaluatie van de belichting. Het verbaast dan ook niet dat er veelvuldig onderzoek werd en wordt gedaan naar structuren die deze kost, het aantal intersectietesten dus, kunnen drukken. In dit deel wordt een overzicht gegeven van de daartoe ontwikkelde structuren.

De algoritmen, methoden en structuren, die schaduwen in synthetische beelden modelleren, uit de literatuur, kunnen we in twee categorieën opdelen:

1. "Blokkerdetectoren": zijn systemen die specifiek het detecteren van de mogelijke blokkers, objecten die schaduwen afwerpen, in de scene gaan optimaliseren. Dit vermijdt de noodzaak om tijdens het ray tracen te itereren over *alle* objecten in de scène. In plaats daarvan moet enkel een subset van de objecten in de scène behandeld worden voor het bepalen van de belichting.

2. "Schaduw algoritmen": zijn systemen die, gegeven de blokkers, het berekenen van de belichting zelf trachten te optimaliseren door andere bemonstering schema's of analytische methoden te gebruiken.

Voor schaduwalgoritmen kan steeds een afweging gemaakt worden: correctheid van de oplossing versus de snelheid waarmee de oplossing berekend wordt.

Bijvoorbeeld de techniek uit [PSS98] benaderd een zachte schaduw, terwijl per lichtbron slechts één schaduwstraal nodig is. De schaduwen hebben achter altijd een umbra regio, wat in realiteit niet noodzakelijk het geval is.

De keuze die hierin wordt gemaakt is sterk afhankelijk van de toepassing. Zo zal in computerspelletjes eerder voor snelheid gekozen worden dan voor fysisch correcte beelden. In computergeanimeerde films zal de keuze ongeveer in het midden liggen.

Schaduwalgoritmen kunnen de structuren voor het bepalen van de blokkers goed gebruiken, want de blokkers moeten nog steeds bepaald worden vooraleer de belichting kan bepaald worden.

In dit hoofdstuk wordt een overzicht gegeven van blokker detectoren. De structuur die werd ontwikkeld voor deze verhandeling bevat verschillende elementen van de hier besproken algoritmen. Het is daarom interessant om de verschillende methoden wat meer in detail te bekijken.

Een overzicht van de meeste bestaande technieken omtrent het behandelen van de schaduwen en blokkers, kan worden teruggevonden in [WPF90] en [HLHS03].

3.1 Technieken op basis van ruimtelijke opdeling

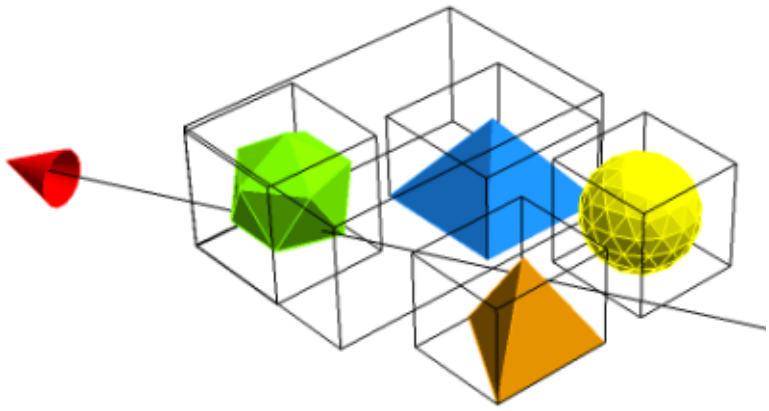
In deze vaak gebruikte categorie van systemen voor het detecteren van schaduwen zal de ruimte volgens een bepaalde structuur opgedeeld worden.

3.1.1 Begrenzende volumes

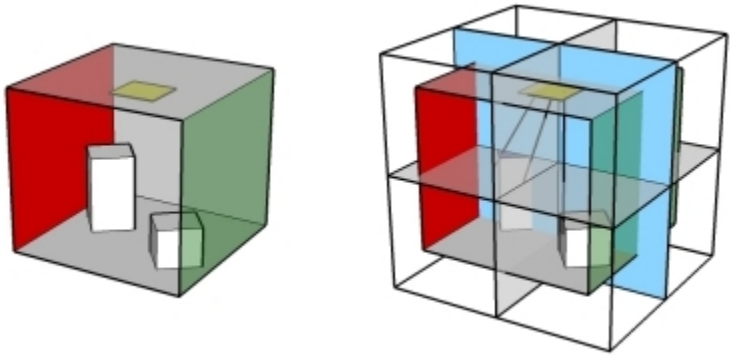
Een eerste ruimtelijke structuur die helpt bij het bepalen van de blokkers zijn zogenaamde "bounding volumes". Rond elk object, dat uit meerdere geometrische eenheden zoals bijvoorbeeld driehoeken bestaat, wordt een balk gedefinieerd die het hele object omvat. Een voorbeeld van een mogelijke structuur is weergegeven in figuur 3.1.

Bij het snijden van een straal met objecten uit de scène, wordt eerst gecontroleerd of de straal wel snijdt met het begrenzende volume van het object. Wanneer dit niet het geval is, zal de straal ook niet snijden met het object zelf en dus wordt het controleren op intersectie met de geometrische eenheden, waaruit het object bestaat, uitgespaard.

Het begrenzende object hoeft niet noodzakelijk een balk te zijn, ook bijvoorbeeld een bol kan gebruikt worden. De afweging bij de keuze is de ratio van de begrensde ruimte ten opzichte van de ruimte die ingenomen wordt door het object zelf. Indien te veel lege ruimte meegerekend wordt in de



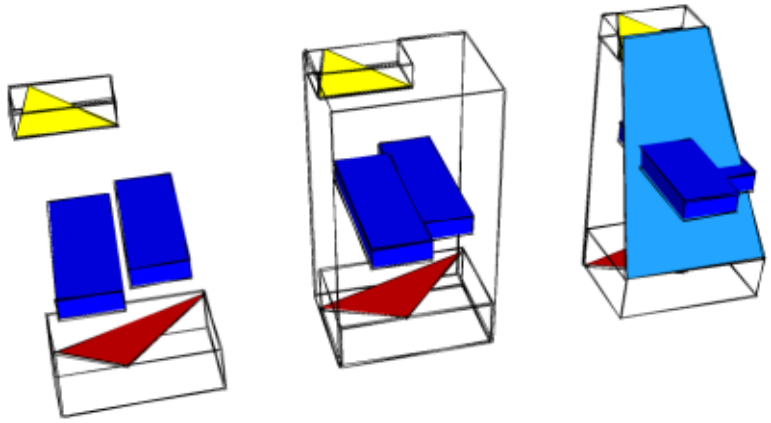
Figuur 3.1: Intersectie van een straal met een hiërarchie van begrenzende volumes.



Figuur 3.2: Links de scène, rechts de scène in een octree.

begrenzing, zal er vaker onterecht een intersectie voorkomen met het begrenzende object, waardoor men nodeloos het hele object moet controleren. Hoe nauwer de begrenzing aansluit bij het te begrenzen object, hoe beter.

Verder kan ook een hiërarchie opgesteld worden van deze volumes, waarbij een groep begrenzende volumes zelf in een begrenzend volume worden geplaatst. De structuur van de hiërarchie zelf kan verschillende vormen aannemen, binaire bomen, avl-bomen en dergelijke zijn mogelijk Zie ook [SM03].



Figuur 3.3: Het opzetten van de schacht in shaft culling. a) de begrenzende volumes voor object en lichtbron b) het volume dat beide omvat, c) de vlakken tussen beide.

3.1.2 Octrees

Een andere, vaak gebruikte opdeling van de ruimte is door middel van een octree. Dit is een boomstructuur waarbij elke knoop acht kinderen bevat, die de ruimte van de knoop opdelen in acht gelijke kubussen. De wortel van de boom is zelf een kubus die de hele scene omvat. Een weergave van een mogelijke opdeling voor een scène staat weergegeven in figuur 3.2.

Wanneer een straal moet snijden met de objecten wordt eerst de cel bepaald waar de straal in begint. Indien de oorsprong van de straal buiten de boom zelf ligt, wordt gestart in de cel waar de straal de structuur binnengaat. De straal wordt dan gecontroleerd op intersectie met de objecten die zich bevinden in die eerste cel.

Indien het dichtstbijzijnde snijpunt binnen deze cel gevonden wordt, dan weet men ook dat dit snijpunt het dichtstbijzijnde snijpunt zal zijn voor de hele scène en moet niet verder gezocht worden.

Indien geen snijpunt werd gevonden, wordt de buur-cel opgevraagd die grenst aan de zijde van de cel waar de straal de cel verlaat. De objecten in de buurcel worden dan op hun beurt gecontroleerd op intersectie, enzovoort, totdat een snijpunt gevonden is of de straal de volledige structuur verlaat.

3.1.3 Shaft Culling

Shaft culling is een andere methode om het aantal blokkers, die getest worden op intersectie, te verminderen. Deze techniek wordt uitvoerig beschreven in [HW94].

Het algoritme veronderstelt een zekere hiërarchische structuur van begrenzende volumes als gegeven. Het doorloopt dan, eenmaal men de zichtbaarheid tussen een lichtbron en punt wil bepalen, de volgende stappen:

1. Ten eerste wordt het begrenzende volume opgezet rond de lichtbron en rond de geometrische eenheid waarop het punt zich bevindt, bijvoorbeeld zoals in figuur 3.3(a).

2. Daarna construeert men het begrenzende volume rond de eerste twee volumes, zoals figuur 3.3(b).

3. Daarna wordt een set vlakken bepaald die de twee volumes met elkaar verbinden, zoals in figuur 3.3(c).

De verschillende volumes in de gegeven hiërarchie worden dan overlopen en elk getest op overlapping met de opgezette structuur. Eerst wordt gecontroleerd op overlapping met het algemeen begrenzende volume, daarna met de twee individuele volumes van lichtbron en object en als laatste met de vlakken.

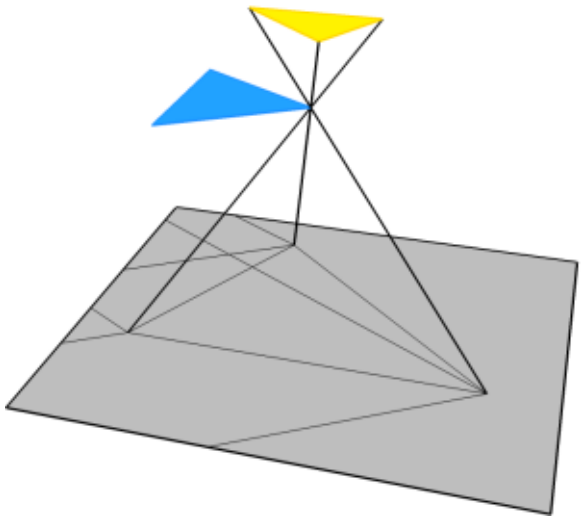
De berekeningskost van de controle op overlapping van twee volumes is goedkoper dan de berekeningskost van de controle op overlapping van een aantal vlakken met een volume, vandaar dat de overlap testen met de volumes als eerste worden uitgevoerd.

Indien een van deze drie testen aangeeft dat een volume uit de hiërarchie niet overlapt met de schacht (dus er niet mee snijdt of in ligt), dan bevat deze en zijn kinderen geen blokkers. In het andere geval worden de kinderen toegevoegd aan de lijst van de volgende items die gecontroleerd moeten worden.

Uiteindelijk zal men een lijst van objecten overhouden die de potentiële blokkers vormen tussen een lichtbron en een geometrische eenheid.

De twee blauwe objecten in figuur 3.3 bijvoorbeeld liggen beide in de schacht en zullen als potentiële blokkers herkend worden.

In [Woo93] wordt een gelijkaardige methode beschreven. Voor een punt waarvoor de belichting berekend wordt, worden blokkers eerst gecontroleerd



Figuur 3.4: Een voorbeeld van een discontinuity mesh.

of ze tussen de vlakken liggen die gevormd worden door het punt en de zijden van de lichtbron, alvorens ze worden gecontroleerd op intersectie met schaduwstralen.

3.2 Discontinuity meshes

Een heel andere categorie zijn algoritmes die een zogenaamde discontinuity mesh opstellen. Een beperking bij deze algoritmen is dat de scène uitsluitend uit polygonen opgebouwd moet zijn .

Voor elke polygon in de scène kan een discontinuity mesh opgebouwd worden, die de polygon in kleinere polygons opdeelt. De kleinere polygonen bepalen regio's waarvoor de van daaruit zichtbare structuur van de lichtbron gelijk is. In elk van deze regio's zijn dezelfde punten en zijden van de lichtbron zichtbaar.

Deze structuur over de polygons in de scène geeft gedetailleerde informatie over de zichtbaarheid van de lichtbronnen. Verschillende andere algoritmen maken hier gebruik van.

Een voorbeeld van een algoritme dat gebruik maakt van discontinuity meshing zijn zogenaamde *backprojections*, beschreven in [SG94] en [DF94]. Daarbij wordt vanuit het punt, waarvoor de belichting moet berekend worden, de discontinuity mesh geprojecteerd op de lichtbron. De projectie geeft

het zichtbare deel van de lichtbron. Op basis daarvan kan dan de belichting berekend worden.

Een ander algoritme dat een discontinuity mesh opstelt is de *visibility skeleton*, beschreven in [DDP97]. Aan de hand daarvan kunnen verschillende zichtbaarheidqueries beantwoord worden, zo kan onder andere de exacte lijst van blokkers tussen twee polygonen opgevraagd worden.

In [DDP02] wordt een "3D visbility complex" ontwikkeld, waarbij de volledige visibiliteit tussen convexe objecten wordt bepaald. Dit algoritme is niet beperkt, in tegenstelling tot een discontinuity mesh, tot scènes opgebouwd uit polygonen.

3.3 Lazy Visibility Evaluation

Een laatste methode om blokkers te detecteren is directe belichting met "lazy visibility evaluation", die beschreven wordt in [HDG99]. In deze techniek wordt eerst een blokker map opgezet waarmee in een tweede stap de belichting geëvalueerd wordt.

Voor elke pixel in het venster van de camera wordt het zichtbare punt in de scène bepaald. Dan wordt elke lichtbron voor elk zichtpunt bemonsterd door middel van schaduwstralen. Op deze wijze worden blokkers gedetecteerd en opgeslagen als blokker-licht paren bij de pixel. Eenmaal een blokker is gedetecteerd voor een bepaalde pixel wordt via een flood-fill algoritme nagegaan of hetzelfde blokker-licht paar ook voorkomt in de naburige pixels.

Als de blokkermap geconstrueerd is, wordt de belichting geëvalueerd, met dus de voorkennis van de blokkers. Naast Monte Carlo als methode om de belichting te evalueren wordt hier ook de mogelijkheid besproken de evaluatie analytisch uit te voeren.

Voor de analytische evaluatie zijn enkele beperkingen gegeven die aan de scène worden opgelegd:

- De lichten bestaan uit polygonen.
- De lichtbronnen zijn diffuus: $L_e(x \rightarrow \Theta) = L_{ei}$.
- De belichte oppervlakken in de scène hebben een diffuse BRDF (dus ook een constante).

Door projectie van de blokkers in een pixel op de bijhorende lichtbron, kunnen de grenzen van het deel van de lichtbron dat zichtbaar is in de pixel,

worden bepaald. De continue integraal over het oppervlak van het zichtbare deel van de lichtbron kan worden omgezet naar een discrete integraal over de zijden van het zichtbare deel van de lichtbron. Op deze wijze heeft men een analytische evaluatie van de belichting in een punt.

In [SR00] wordt een algoritme gegeven dat onder dezelfde beperkingen een analytische evaluatie maakt van de globale belichting. Hier gebeurt dit echter in termen van de punten van het zichtbare deel van de lichtbron in een zichtpunt. De oplossing kan incrementeel, onafhankelijk van de volgorde waarin de punten gegeven zijn, berekend worden. Dit biedt de mogelijkheid om, telkens een punt van het zichtbare deel van de lichtbron is bepaald, het deel van de berekening voor dat punt direct uit te voeren. Projectie in dit algoritme gebeurt op een vlak dat zich boven en evenwijdig aan de polygon ligt waarvan het zichtpunt deel uitmaakt. Voor dit algoritme moeten eveneens de blokkers gekend zijn.

In deze verhandeling worden dezelfde beperkingen opgelegd aan de scène. De mogelijkheid om analytische evaluatie te gebruiken bij het renderen met de techniek beschreven in deze verhandeling komt later uitgebreider aan bod.

James Arvo beschrijft eveneens in [Arv95] een analytische methode om de renderingvergelijking te evalueren, maar dit wanneer de BRDF of de radiantie van de lichtbron is gegeven als een lineaire combinatie van phong-lobes (zie ook [Pho75].

3.4 Local Illumination Environments

Een zogenoemd “local illumination environment” is een zeer recent ontwikkelde structuur. Zulke structuren vormen de basis voor de uitgewerkte techniek voor deze verhandeling en worden daarom uitgebreid in het volgende hoofdstuk besproken.

3.5 Conclusie

In dit deel werden verschillende structuren en algoritmen, die worden gebruikt om het ray tracen te versnellen, besproken. Elke methode die hier vermeld werd, wordt ergens in de structuur die ontworpen werd voor deze verhandeling gebruikt.

Hoofdstuk 4

Local Illumination Environments

"Valt eventueel nog iets mee te doen."

Uit de weblog van mijn thesis. Zoveel maand later: dit hoofdstuk.

Lokale belichtingsomgevingen (local illumination environments) werden voor het eerst geïntroduceerd door Kavita Bala, Sebastian Fernandez, Moreno A. Piccolotto en Donald P. Greenberg in [FBPG00]. Twee jaar later verscheen een artikel van dezelfde auteurs die de lokale belichtingsomgevingen gedetailleerder besprak [FBG02].

In dit hoofdstuk wordt eerst de in deze papers beschreven local illumination environments in detail besproken.

Daarna wordt een nieuwe, gelijkaardige structuur voorgesteld: *Preprocessed Local Illumination Environments*, of voorberekende lokale belichtingsomgevingen. Enkel de nieuwe structuur zelf zal hier worden besproken. In het hier op volgende hoofdstuk zal dieper worden ingegaan op de constructie van deze omgevingen en de implementatie daarvan.

In wat volgt wordt de term "Local Illumination Environment" algemeen afgekort tot "LIE". De voorgestelde structuur "Preprocessed Local Illumination Environments" tot "P-LIE".

4.1 LIE's

In dit onderdeel wordt het concept van LIE's, beschreven in [FBG02] uitvoerig besproken.

Hierin wordt het basisidee van LIE's gegeven, hoe deze gebruikt worden in ray tracing en hoe de constructie van de LIE's verloopt.

4.1.1 Basisidee voor een LIE

Samen met een scène is een octree structuur gegeven. Voor elke cel worden een aantal gegevens opgeslagen, die samen met de cel de lokale belichtingsomgeving vormen.

- Lichtbronnen die voor elk punt in de cel volledig zichtbaar zijn worden bijgehouden in de LIE. Tijdens het renderen met een ray tracer is het overbodig om met schaduwstralen de volledig zichtbare lichtbronnen te controleren op schaduwen, er zijn immers geen blokkers.

- Voor lichtbronnen die deels zichtbaar zijn in een subset van de punten in de cel, wordt een minimale lijst van potentiële blokkers bijgehouden, die geassocieerd zijn met de lichtbron waarvoor de blokker schaduwen afwerpt. Tijdens het ray tracen moeten enkel deze blokkers gecontroleerd worden op intersectie bij het evalueren van de belichting.

- Lichten die niet zichtbaar zijn voor de punten in de cel worden niet bijgehouden. Onzichtbare lichtbronnen zijn diegene die ofwel geen licht afwerpen in de richting van de cel ofwel volledig door blokkers van het zicht onttrokken zijn. Bij het renderen moeten deze lichtbronnen niet in rekening worden genomen bij het evalueren van de belichting.

4.1.2 Het RT proces met LIE's

De LIE structuur versnelt het renderen; in de paper wordt een versnelling van 10x tot 30x gemeld ten opzichte van een conventionele raytracer.

Het renderen van een scène verloopt als volgt:

- Door elk punt van het scherm van de camera wordt vanuit het oogpunt van de camera een zichtstraal getrokken. De snijpunten tussen de zichtstralen en de objecten in de scène, die het dichtst bij het oogpunt van de camera liggen, worden bepaald.

- Voor elk van die snijpunten wordt de cel van de octree bepaald waarin dat punt zich bevindt.

- De opgeslagen gegevens (blokkers en lichtbronnen) in de cel worden gebruikt bij het berekenen van de belichting. Specifiek worden zij gebruikt bij het evalueren van de zichtbaarheidsfunctie uit de renderingvergelijking:

  - Als voor een lichtbron geen blokkers opgeslagen zijn in de cel, dan is er geen nood aan intersectietesten. Er werden geen gegevens opgeslagen omdat ofwel de lichtbron volledig afgeschermd werd door blokkers, ofwel de lichtbron geen licht afwerpt in de richting van de cel.
  - Lichtbronnen die niet zichtbaar zijn in de cel worden niet bijgehouden en dus niet meegerekend.
  - Voor een lichtbron die geassocieerde blokkers heeft, moeten er toch intersectietesten uitgevoerd worden. Deze blokkers zijn echter de enige waarop de testen moeten worden uitgevoerd.

4.1.3 Constructie van een LIE

De constructie van de LIE structuur gebeurt op basis van de positie van de camera [1]. Dit gebeurt *parallel* met het renderen van de scène zelf. Tijdens het renderen wordt de structuur steeds verder aangevuld. Dit impliceert de beelden in het begin fouten zullen vertonen. Hoe volledi ger de LIE-structuur wordt, des te minder fouten zichtbaar zijn.

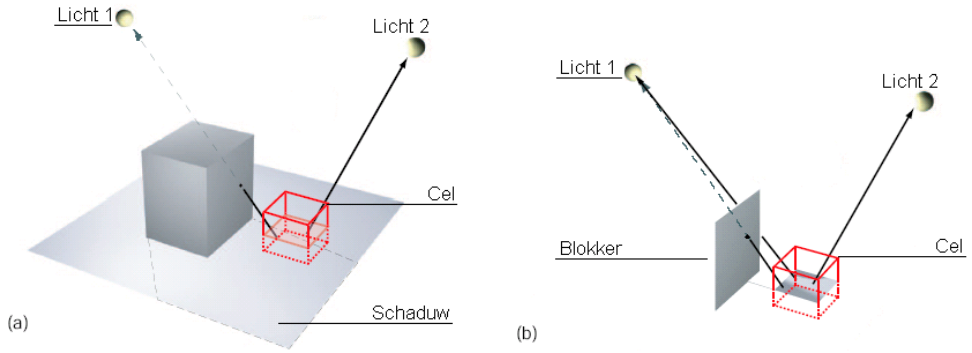
Het proces dat de LIE structuur construeert trekt vanuit het oogpunt van de camera een willekeurige zichtstraal in de scène. Het snijpunt van de zichtstraal met de objecten in de scène (dichtst bij het oogpunt van de camera) wordt bepaald.

Als er nog geen LIE bestaat voor de kleinste octreecel die het snijpunt bevat, wordt er een LIE aangemaakt. Als de LIE in de cel wel al bestaat, wordt deze verder aangevuld. Dit gebeurt door vanuit het gevonden snijpunt schaduwstralen naar de verschillende lichtbronnen te controleren op intersectie met blokkers.

In figuur 4.1 bijvoorbeeld wordt naar elke lichtbron een schaduwstraal gericht. Voor lichtbron 1 wordt een blokker gevonden, voor lichtbron 2 niet.

- Als de schaduwstraal geen objecten snijdt voor een bepaalde lichtbron en de LIE bevat de lichtbron niet, wordt de lichtbron als niet “geblokt” toegevoegd.

[1] view-driven constructie



Figuur 4.1: Detectie van de blokkers. (a) detectie, (b) de LIE voor de cel. Afbeelding uit [FBG02].

- Als de schaduwstraal wel een object snijdt voor een bepaalde lichtbron en de LIE bevat de lichtbron niet, verandert niets aan de LIE.
- Als de schaduwstraal geen objecten snijdt voor een bepaalde lichtbron en de LIE bevat de lichtbron wel, verandert niets aan de LIE.
- Als de schaduwstraal wel een object snijdt voor een bepaalde lichtbron en de LIE bevat de lichtbron wel, dan wordt het object waarmee de schaduwstraal snijdt, toegevoegt aan de set van blokkers voor de lichtbron.

Doordat het constructieproces parallel wordt uitgevoerd met de renderer zullen de eerste gegenereerde beelden enkele foutjes bevatten. Die verdwijnen wanneer het constructieproces een tijd lang heeft kunnen werken aan de structuur.

In de hierboven vermelde artikels werd de LIE structuur op voorhand geconstrueerd, wat belang is voor de tijdsduur van het renderen. De duur van de constructie verliep van enkele seconden tot een uur, afhankelijk van de scène, de positie en het venster van de camera.

Verfijnen van de octree

Indien de LIE voor een cel te complex wordt zal deze worden opgesplitst in acht kindercellen. Aanvankelijk starten deze kinderen zonder toegevoegde gegevens. Het is aan het constructieproces om de LIE's in de kinderen van nul terug op te bouwen.

Als criterium voor complexiteit wordt in het artikel het totale aantal blokkers in een LIE gebruikt. Wanneer dit aantal een zekere grens overschrijdt wordt de cel opgesplitst. Er wordt echter niet vermeld welke waarde deze grens zou moeten hebben.

Daarnaast wordt een specifieke grens opgelegd op de diepte van de boom. De auteurs vermelden hierbij dat de optimale vaste diepte voor het construeren en renderen afhankelijk is van de scène zelf. Het is daardoor moeilijk een eenduidige richtlijn te geven voor de maximale diepte van de boom.

4.1.4 Detectie van de blokkers

De blokkers voor een bepaalde lichtbron in een LIE worden gedetecteerd door middel van het bemonsteren van de lichtbron zelf.

In het artikel wordt vermeld dat dit eventueel door middel van shaft culling [HW94] kan gebeuren. Deze methode wordt echter te conservatief bevonden omdat ze alle blokkers bijhoudt, terwijl een subset van deze blokkers potentieel niets bijdraagt. Wanneer namelijk een object, binnen de schacht tussen lichtbron en punt, achter een ander object verborgen zit, zal deze niet bijdragen tot de schaduw in de LIE.

Het detecteren via bemonstering daarentegen zal een potentieel kortere lijst van blokkers genereren; enkel de blokkers die in het zicht van de LIE ten opzichte van de lichtbron vallen zullen worden bijgehouden.

Volledig optimaal is detectie via bemonstering niet. Indien een aantal kleinere blokkers zich bevinden voor een grotere blokker, zou het toch beter zijn enkel die grotere blokker bij te houden; hoe minder blokkers er worden bijgehouden, hoe minder intersectietesten moeten uitgevoerd worden tijdens het renderen.

4.2 P-LIE's

Het concept van lokale belichtingsomgevingen vormt het basis idee van de structuur die in dit deel wordt voorgesteld.

Eerst wordt de nieuwe structuur voorgesteld. Daarna worden de doelstellingen besproken die voor deze structuur gesteld worden.

4.2.1 Basisidee voor een P-LIE

Het doel van LIE's en P-LIE's is het versnellen van het ray tracing proces. Dit wordt bekomen door een octree structuur aan te vullen met zichtbaarheidsinformatie in de cellen. Met behulp van deze informatie kan de zichtbaarheidsfunctie sneller worden geëvalueerd.

De versnelling moet van zulke grootte-orde zijn dat ze de constructiekost van de structuur op zijn minst compenseert. Met andere woorden: (de constructieduur van de P-LIE + de tijd nodig om met de P-LIE gegevens te renderen) moet lager liggen dan (de constructieduur van een octree + het renderen met enkel de octree), zodat het bepalen van de extra zichtbaarheidsinformatie in de cellen effectief tijd uitspaart.

Daarbij moet opgemerkt worden dat de P-LIE structuur onafhankelijk is van het oogpunt van de camera en dus geconstrueerd wordt voor elk mogelijk zichtpunt. De structuur moet slechts éénmaal aangemaakt worden, waarna meerdere beelden op basis van dezelfde P-LIE gegenereerd kunnen worden. Het bijhorende doel is dat de constructie van P-LIE's niet meer tijd vergt dan de constructie van LIE's.

Dit geldt natuurlijk ook voor een octree, maar het geldt niet voor de eerder voorgestelde LIE structuur; het constructieproces moet voor elk oogpunt van de camera aan de gang zijn en blijven.

Doordat de constructie van LIE's parallel met het renderen plaats vindt is het echter wel moeilijk een vergelijking te maken tussen LIE's en P-LIE's op vlak van duur van de constructie en het renderen, aangezien deze tijden voor LIE's met elkaar verweven zijn. Een uitgebreide vergelijking van de P-LIE structuur met de LIE structuur en de octree structuur zal worden gemaakt in hoofdstuk 6.

Naast het versnellen van het ray tracen kan deze structuur eventueel ook gebruikt worden in andere schaduwalgoritmen die gebruik maken van de kennis van blokkers tussen een punt en een lichtbron in de scène. De mogelijkheden die P-LIE's kunnen bieden voor andere doeleinden dan versnelling komen later in deze verhandeling uitgebreid aan bod.

4.2.2 Gelijkenissen tussen LIE's en P-LIE's

Voorberekende lokale belichtingsomgevingen hebben dezelfde basis als de lokale belichtingsomgevingen zoals ze werden voorgesteld in [FBG02]. Ze zijn aan elkaar gelijk wat betreft het volgende:

- Een P-LIE (Preprocessed Local Illumination Environment) wordt net

als een LIE gevormd door extra gegevens op te slaan in de cellen van de octree van de scène.

- Deze gegevens in een P-LIE bevatten net als de gegevens in een LIE informatie over objecten in de scène die het licht dat toekomt in de cel afschermen.

4.2.3 Onafhankelijkheid van het zichtpunt

P-LIE's verschillen van de eerder voorgestelde LIE structuur in de constructie. P-LIE's worden volledig onafhankelijk van de positie van de camera geconstrueerd.

Dit brengt enkele implicaties met zich mee:

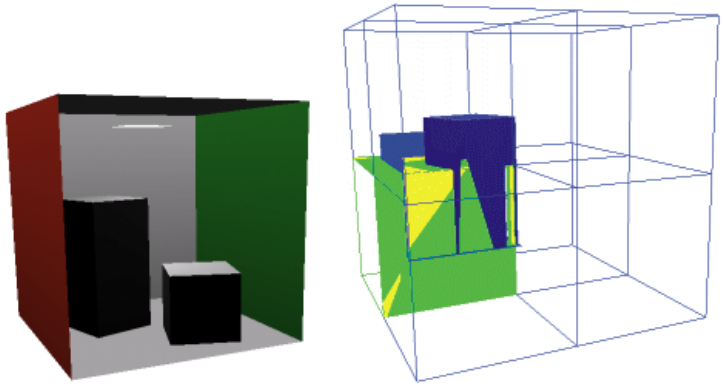
- Niet enkel de cellen van de octree in het zichtpunt van één of andere camera, maar alle cellen van de octree moeten voorzien worden van een LIE; er is immers geen indicatie welke cellen tijdens het renderen zullen worden gebruikt en welke niet.
- De structuur hoeft niet meer geconstrueerd te worden tijdens het renderen. De constructie van de structuur kan gebeuren voordat aan het renderen wordt begonnen.

Enkele voordelen:

- Het renderproces moet geen processortijd en/of geheugentoegang delen met het constructie proces, aangezien de constructie nu op voorhand gebeurt.
- De structuur kan opgeslagen worden en meerdere malen gebruikt worden om een afbeelding te renderen, vanuit alle mogelijke cameraposities. De constructie is dus een eenmalige uitvoering, waarvan het resultaat blijvend gebruikt kan worden.
- De constructie "mag" (liefst niet natuurlijk) wat langer duren, als dit opweegt tegen de tijdswinst voor het renderen van de scène.

Enkele nadelen:

- Voor alle cellen een P-LIE construeren impliceert dat dit ook voor cellen gebeurt die eventueel tijdens het renderen nooit gebruikt zullen worden.
- De nodige geheugenruimte zal groter zijn wannneer in plaats van enkel voor de cellen die nodig zijn tijdens het renderen, voor elke cel een LIE wordt geconstrueerd.



Figuur 4.2: Links de scène, rechts de octree met een visualisatie van de gegevens in de linker-onder-achter cel. De blokkers in blauw, de umbra regio in groen en de penumbra regio in geel.

4.2.4 Gegevens in een P-LIE

De gegevens die worden opgeslagen in een P-LIE verschillen van een LIE:

Voor elke zijde van elke cel wordt voor elke lichtbron in de scène het volgende bewaard:

1. de objecten die zich bevinden tussen de zijde en de lichtbron, namelijk de zogenaamde "blokkers". Dit zijn dezelfde gegevens als in een LIE worden opgeslagen.

2. de umbra regio van de schaduw die de blokkers veroorzaken op de zijde.

3. de penumbra regio van de schaduw die de blokkers veroorzaken op de zijde.

In figuur 4.2 worden de opgeslagen regio's op een cel en de bijhorende blokkers gevisualiseerd.

4.2.5 Het RT proces met P-LIE's

De gegevens die in de P-LIE structuur opgeslagen worden, zorgen voor een aantal extra mogelijkheden tijdens het renderen met behulp van ray tracing. Wanneer de visibiliteitsfunctie geëvalueerd moet worden, zal dit in standaard ray tracing gebeuren door stralen (vanuit het punt waarvoor de belichting wordt geëvalueerd (x) naar een punt (y) op een lichtbron (L_i)) te controleren

op intersectie met de objecten in de scène. Dit kan nu anders.

Ten eerste zal bepaald worden in welke cel het punt x zich bevindt, daarna door welke zijde van de cel de straal richting een punt y op een lichtbron L_i gaat. De gegevens voor deze zijde en de lichtbron L_i waar y op ligt worden hiervoor gebruikt.

1. Als er geen gegevens gevonden worden, dan valt er geen licht van L_i in het punt x. Het resultaat van de visibiliteitsfunctie is 0.

2. Wanneer er wel gegevens gevonden worden, maar geen blokkers zijn opgeslagen, bevinden er zich geen blokkers tussen de zijde en de lichtbron. De drie hierna volgende stappen kunnen in dat geval overgeslagen worden.

3. De straal wordt gecontroleerd op intersectie met de umbra regio van de zijde. Als de straal daarmee effectief snijdt, dan ligt het punt x in een harde schaduw en is het resultaat van $V(x, y)$ dus 0.

4. De straal wordt opnieuw gecontroleerd op intersectie, ditmaal met de penumbra regio van de zijde. Als de straal daarmee niet snijdt, ligt het punt in het volle licht van L_i en moeten de blokkers niet meer gecontroleerd worden.

5. Als de straal wel snijdt met de penumbra regio, wordt de straal gecontroleerd op intersectie met de blokkers tussen de zijde en L_i. Als een intersectie wordt gevonden, dan ligt x in de schaduw. Het resultaat is 0.

6. Wanneer blijkt dat

   - de straal niet snijdt met de penumbra regio,
   - of de straal niet snijdt met de blokkers,
   - of de zijde voor lichtbron L_i geen blokkers heeft,

   moet de straal nog gesneden worden met de objecten die zich in de cel zelf bevinden. Als daarmee een intersectie gevonden wordt, is het resultaat opnieuw 0, in het andere geval is het 1; x en y kunnen elkaar dan “zien”.

4.3 Conclusie

In dit hoofdstuk werd het concept van zogenaamde "Local Illumination Environments" geïntroduceerd. Deze structuur bevat informatie over welke objecten schaduwen zullen afwerpen en welke lichtbronnen licht stralen in een cel. Het ray tracing proces wordt hiermee met een factor 10 tot 30 versneld.

Op basis van dit concept werd de nieuwe structuur, "Preprocessed Local Illumination Environments" voorgesteld. Het grote verschil tussen beide zit in de constructie, die bij P-LIE's onafhankelijk van de positie van de camera gebeurt. Verder bevat deze structuur nog meer informatie over de visibiliteit.

Het doel is het renderen met ray tracing te versnellen. Aangezien de P-LIE's onafhankelijk van de camerapositie geconstrueerd worden, kunnen ze zonder nieuwe berekeningen vanuit verschillende cameraposities hergebruikt worden.

Deel II

Preprocessed LIE's

Hoofdstuk 5

Constructie van Preprocessed LIE's

Voorberekende lokale belichtingsomgevingen vormen een structuur die voor een synthetische scène wordt geconstrueerd. Op basis van deze structuur kan het renderen van de scène met ray tracing versneld worden. Deze versnelling vindt plaats bij de evaluatie van de zichtbaarheidsfunctie uit de renderingvergelijking voor de directe belichting, die wordt gebruikt om de radiantie in een punt in een bepaalde richting te berekenen.

In dit hoofdstuk wordt beschreven hoe voorberekende lokale belichtingsomgevingen, of preprocessed local illumination environments (P-LIE's), worden geconstrueerd.

De constructie gebeurt in een aantal stappen:

1. Een octreestructuur voor de scène wordt opgezet.
2. De data-objecten worden geïnitialiseerd. Voor elke zijde van elke cel die een lichtbron "ziet" wordt een gegevensobject aangemaakt. Daarbij moet "zichtbaarheid" tussen zijde en lichtbron gedefinieerd worden.
3. Omvattende volumes worden opgesteld rond de zijden en lichtbronnen.
4. De vlakken, de hoeken van de vlakken en de oriëntaties van de hoeken worden bepaald tussen zijden en lichtbronnen. Deze worden later gebruikt bij het clippen van de blokkers en het verwijderen van gegevens.
5. De umbra en penumbra regio op de zijden, veroorzaakt door de lichtbronnen met bijhorende blokkers, worden bepaald. Zij worden later voor het verwijderen van gegevens en tijdens het renderen zelf gebruikt.

6. Gegevens worden verwijderd. Zo zullen bijvoorbeeld zijden die volledig in een umbra regio liggen verwijderd worden.

7. Onder bepaalde voorwaarden kan de octree verder worden opgesplitst. Dit zorgt voor een betere aanpassing van de structuur van de octree aan de belichting die aanwezig is in de scène.

Deze stappen komen in wat volgt in detail aan bod. Als laatste zullen een aantal bedenkingen in verband met de parameters voor dit constructieproces geformuleerd worden.

5.1 Opdeling van de Octree

Als eerste stap wordt voor de scène een octree gecreëerd.

De objecten in de scène worden niet in hun geheel in de octree geplaatst. Het zijn de geometrische primitieven waaruit de objecten bestaan waarvoor bepaald zal worden in welke cel(len) ze thuishoren.

Dit impliceert dat objecten uit geometrische primitieven moeten bestaan. Voor deze verhandeling werd gekozen voor driehoeken, maar andere, zoals algemene polygonen of NURBS zijn mogelijk. Dit impliceert eveneens dat andere geometrische objecten zoals bijvoorbeeld bollen, die met gemak konden worden gemodelleerd door middel van een centrum en een radius, nu ook benaderd moeten worden met driehoeken.

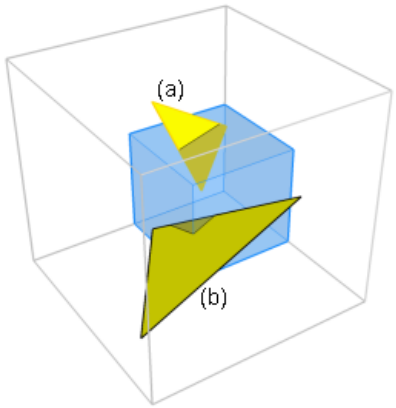
Een boomstructuur bestaat uit knopen en bladeren; dit geldt ook voor een octree. In de knopen van een octree worden geen driehoeken van de objecten in de scène bewaard, enkel in de bladeren van de boom. De driehoeken in een knoop zijn dan de driehoeken in zijn kinderen.

In elk blad wordt een lijst bijgehouden van de driehoeken die zich in dat blad bevinden. Een driehoek bevindt zich in een blad wanneer

1. minimum één van de punten van de driehoek zich in het blad bevindt (bijvoorbeeld driehoek a in figuur 5.1) en/of

2. minimum één van de zijden van de driehoek door het blad gaat (bijvoorbeeld driehoek b in figuur 5.1).

Dit impliceert dat een driehoek in meerdere bladeren tegelijk kan worden bijgehouden. Als de driehoeken van verschillende bladeren opgevraagd worden, moet daar rekening mee gehouden worden.



Figuur 5.1: Twee driehoeken in een blad. Driehoek (a) heeft twee punten in de cel, driehoek (b) heeft een zijde die door de cel gaat.

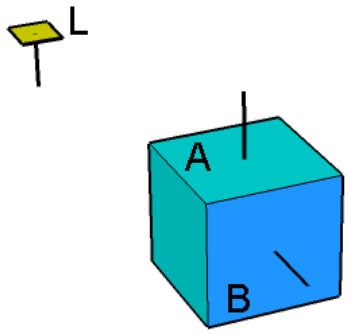
Eerst moet de kubus die de hele scène omvat (de "wortel" van de boom) bepaald worden. Dit gebeurt door het centrum van de scène en de maximale afstand van de punten tot dit centrum te bepalen. Dit centrum en de afstand bepalen de kubus.

De wortel is altijd een knoop. Aangezien een knoop geen bladeren bevat, worden acht bladeren aan de wortel als kinderen toegekend. Elk van deze kinderen is opnieuw een kubus. Voor alle driehoeken in de scène wordt bepaald in welke van deze acht kinderen ze thuishoren.

Voor de octree is bepaald wat het maximum aantal driehoeken in een blad kan zijn. Indien voor één van de bladeren blijkt dat deze grens wordt overschreden, wordt het blad een knoop, waaraan opnieuw acht kinderen worden toegekend, en worden de driehoeken van het originele blad verdeeld over de nieuwe kinderen. Dit principe wordt recursief herhaald tot elk blad in de boom een correct aantal driehoeken bevat.

Een complexe scène kan veel op één plaats geconcentreerde driehoeken bevatten. Deze zorgen ervoor dat de boom te complex wordt (te diep). Daarom wordt een maximum grens aan de diepte van de boom opgelegd. Indien een blad te veel driehoeken bevat, maar splitsen de boom te diep maakt, dan wordt niet verder opgedeeld.

Voor de implementatie van een octree kan best gebruik gemaakt worden



Figuur 5.2: Zijde A kan lichtbron L zien, de normalen wijzen naar elkaar. Zijde B kan lichtbron L niet zien.

van het Composite pattern uit [GHJV95].

5.2 Initialisatie van de gegevensobjecten

Wanneer de octree opgesteld is, worden de gegevensobjecten in de bladeren gecreëerd.

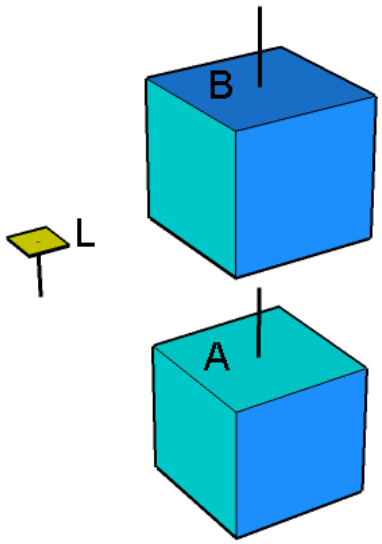
In elk blad zal een gegevensobject worden aangemaakt *per zijde per lichtbron* die "zichtbaar" is in die zijde. Een lichtbron is zichtbaar voor een zijde als deze twee voorwaarden voldaan zijn:

- Minimum één van de normalen van de driehoeken die de lichtbron vormen, wijst naar de zijde en de normaal van de zijde wijst naar de lichtbron. Beide normalen moeten met andere woorden naar elkaar gericht staan. Dit kan bepaald worden met volgende vergelijking:

  $$dotproduct(zijde.normaal, lichtbron.normaal) <= 0.0$$

  In figuur 5.2 bijvoorbeeld is voor zijde B de lichtbron L niet zichtbaar, maar voor zijde A wel.

- De zijde moet zich voor de lichtbron bevinden, niet achter de lichtbron. Voor en achter worden bepaald door de oriëntatie van de normaal van de lichtbron. Bijvoorbeeld in figuur 5.3 staat zijde A voor de lichtbron en zijde B achter de lichtbron.



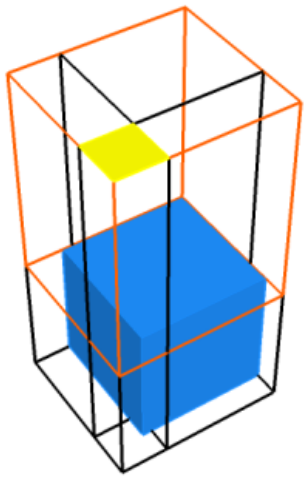
Figuur 5.3: Zijde A kan lichtbron L zien, de normalen wijzen naar elkaar en A ligt voor L. Zijde B kan lichtbron L niet zien, de normalen wijzen wel naar elkaar maar B light achter L.

In elke cel wordt voor elke zijde die een lichtbron kan zien een gegevensobject aangemaakt. In die objecten zullen dan de blokkers en de umbra en penumbra regio's worden bijgehouden.

Voor de constructie wordt in de gegevensobjecten nog meer informatie bijgehouden (cfr. infra). Wanneer de constructie voltooid is worden deze terug verwijderd.

Een gegevensobject kan binnen de cel geïdentificeerd worden door de tuple $(zijde, lichtbron)$. Over de hele octree kan de identificatie gebeuren met de tuple $(cel, zijde, lichtbron)$. De wijze waarop de gegevensobjecten worden bijgehouden, kan dus verschillen:

- Ofwel wordt binnen één cel per zijde een lijst van bijhorende gegevensobjecten bijgehouden (zes lijsten).
- Ofwel wordt binnen één cel een lijst van gegevensobjecten bijgehouden die opvraagbaar zijn met de tuple $(zijde, lichtbron)$.
- Ofwel kan over de hele octree een lijst van gegevensobjecten worden bijgehouden die opvraagbaar zijn met de tuple $(cel, zijde, lichtbron)$.



Figuur 5.4: De drie omvattende volumes voor een lichtbron en een cel. Het volume voor de bovenzijde staat in oranje.

In de implementatie werd voor de tweede optie gekozen, omdat een zijde van een cel gemakkelijk kan worden opgevraagd aan de hand van een *richting*.

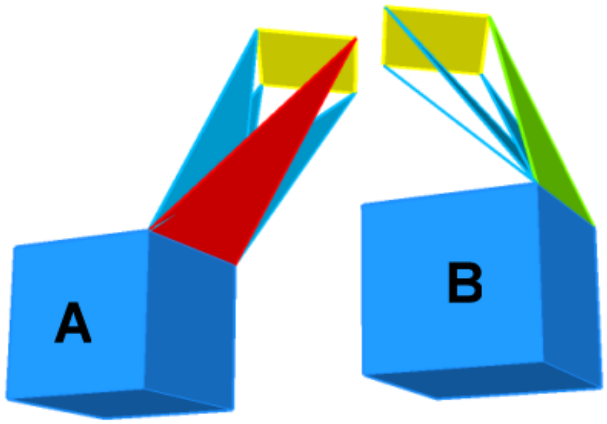
5.3 Omvattende volumes

Rond de lichtbron en de zijde van elk gegevensobject wordt een balk bepaald en bewaard. Dit omvattend volume wordt later bij het clippen van de blokkers gebruikt. In figuur 5.4 bijvoorbeeld staan de drie omvattende volumen voor een lichtbron en de drie zijden, waar de lichtbron zichtbaar is, afgebeeld.

Indien de lichtbron geen vlak is of niet parallel met één van de vlakken van de cel staat, wordt het omvattend volume rond de zijde en het omvattend volume van de lichtbron bepaald.

5.4 Vlakken, hoeken en oriëntaties

Voor elk gegevensobject worden vier vlakken opgezet die de schacht van de lichtbron naar de zijde bepalen. Naast de vlakken worden de hoeken die de vlakken met de normaal van de zijde maken, en de oriëntaties (links of rechts) van de hoeken in het gegevensobject bewaard. De vlakken worden later gebruikt bij het clippen van de blokkers, de hoeken en de oriëntaties bij het verwijderen van gegevensobjecten.



Figuur 5.5: De vlakken voor een ribbe. Voor cel A is het rode vlak het resultaat, voor cel B het groene vlak.

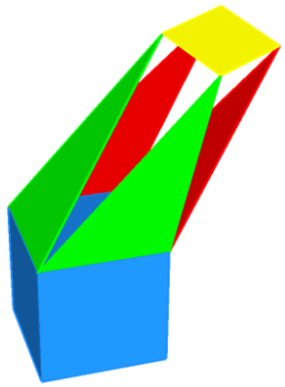
De vier vlakken tussen een zijde en een lichtbron gaan elk door één van de vier ribben van de zijde (aangezien een cel een kubus is, is een zijde een vierkant) en door een punt van de lichtbron.

- Indien de lichtbron volledig links van een ribbe van de zijde ligt, dan zal het vlak door die ribbe snijden met een punt op de lichtbron, zodat het vlak een *minimale* hoek maakt met de normaal van de zijde.
- Indien de lichtbron gedeeltelijk of volledig rechts van een ribbe van de zijde ligt, dan zal het vlak door die ribbe snijden met een punt op de lichtbron, zodat het vlak een *maximale* hoek maakt met de normaal van de zijde.

Links en rechts worden hierbij bepaald door de richting waarin de ribben liggen. De ribben zijn gegeven tegen de wijzers van de klok in, ten opzichte van de normaal van de zijde.

Het bepalen van de vlakken gebeurt door voor elke ribbe van de zijde te itereren over alle coördinaten van de lichtbron. Voor elk coördinaat wordt het vlak door dat punt en de ribbe bepaald en de hoek van dat vlak met de normaal van de zijde berekend. Op basis van de twee bovenstaande regels wordt dan het correcte vlak bepaald.

In afbeelding 5.5 ligt voor cel A de lichtbron volledig rechts van de zijde waarvoor de vlakken worden bepaald. Het rode vlak is het juiste. Voor cel B



Figuur 5.6: De uiteindelijke vlakken voor een zijde. Rode vlakken zijn rechts geörienteerd, groene vlakken links.

in de afbeelding ligt de lichtbron voor de zijde waar de vlakken voor worden bepaald volledig links. Het groene vlak is hierbij het juiste.

Naast de vlakken worden ook de bijhorende hoeken met de normaal van de zijde opgeslagen in het gegevensobject. Voor elke hoek wordt daarbij de oriëntatie opgeslagen; *Links* indien de lichtbron links van de ribbe lag, *Rechts* indien de lichtbron deels of volledig rechts van de ribbe lag.

In afbeelding 5.6 zijn de vier vlakken voor de zijde bepaald. De rode vlakken zijn rechts geörienteerd, de groene links.

5.5 Clippen van de blokkers

Wanneer voor een gegevensobject het omvattend volume en de vier vlakken bepaald zijn (stap 1.3 en 1.4), kunnen de blokkers bepaald worden. Dit verloopt in een aantal stappen:

1. Eerst wordt bepaald welke bladeren van de octree in het omvattende volume rond de zijde en de lichtbron zitten. Deze bladeren bevatten alle mogelijke objecten die het uitgaande licht van de lichtbron op de zijde kunnen afschermen.

   Een blad zit in het volume wanneer die ermee overlapt of er volledig in ligt.

2. Als tweede stap worden de driehoeken in de bepaalde bladeren in een lijst geplaatst zodat geen dubbels meer voorkomen. Een driehoek kan

in meerdere cellen voorkomen, dus een simpele samenvoeging (concat) van alle lijsten van driehoeken die bijgehouden worden in de bladeren kan niet gebruikt worden; driehoeken die dubbel voorkomen zouden dubbel als blokker herkend worden.

In plaats daarvan worden de driehoeken uit de bladeren één voor één in een hashtabel geplaatst, zodat bij toevoeging snel kan worden nagegaan of een bepaalde driehoek al eerder werd toegevoegd of niet.

3. Bij de derde stap begint het eigenlijke werk: elke driehoek uit de hashtabel wordt geclipt tegen de vier eerder bepaalde vlakken tussen de zijde en de lichtbron. Omdat de schacht die de vier vlakken vormen bovenaan en onderaan een opening heeft, worden de driehoeken ook geclipt tegen de vlakken van het omvattende volume. Elke driehoek waarvan na het clippen minstens een deel overblijft, is een blokker die in de schacht ligt.

   In figuur 5.7 wordt het clippen van twee blokkers voor een zijde afgebeeld.

   Voor het clippen werd het Sutherland-Hodgman clippingalgoritme gebruikt, beschreven in [FvDFH90]. Dit is in principe een 2D polygon clipping algoritme, waarbij een 2D polygon tegen een lijn wordt geclipt. Zoals in het boek wordt vermeld kan dit algoritme omgezet worden naar een 3D polygon clipping algoritme, waarbij een 3D polygon (in ons geval een 3D driehoek) tegen een vlak wordt geclipt.

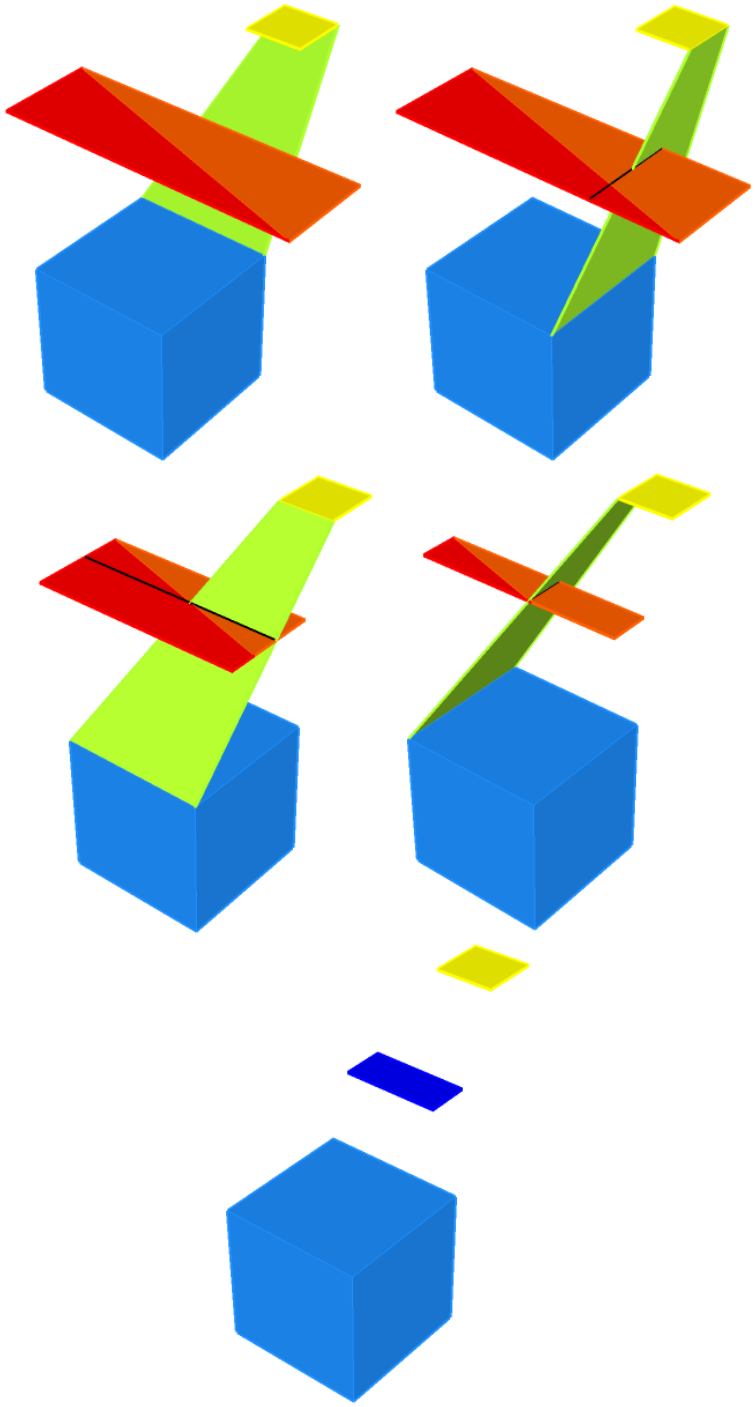
In figuur 5.8 worden de geclipte blokkers in een cel van de octree van de Cornell Cube 2 scène afgebeeld.

5.6 Bepaling van umbra en penumbra

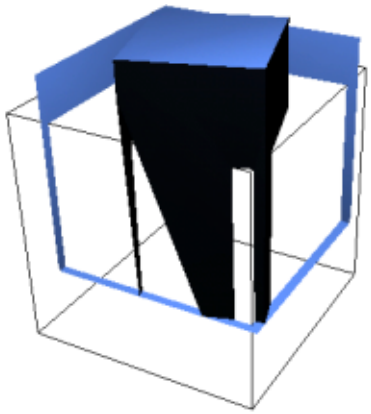
Wanneer alle blokkers tussen een zijde en een lichtbron gekend zijn, kan de umbra en penumbra regio voor een zijde bepaald worden. Dit gebeurt in een aantal stappen:

1. Voor elke geclipte blokker wordt vanuit elk coördinaat van de lichtbron een projectie op de zijde bepaald.

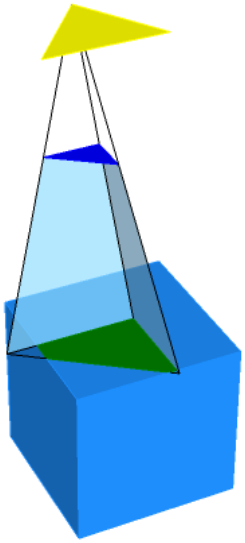
   Voor de projectie wordt opnieuw gebruik gemaakt van de Sutherland-Hodgman clipper. Voor elke blokker worden vlakken bepaald die door het punt van de lichtbron en een zijde van de blokker gaan. Het vierkant van de zijde wordt tegen de bepaalde vlakken geclipt. Wat overblijft



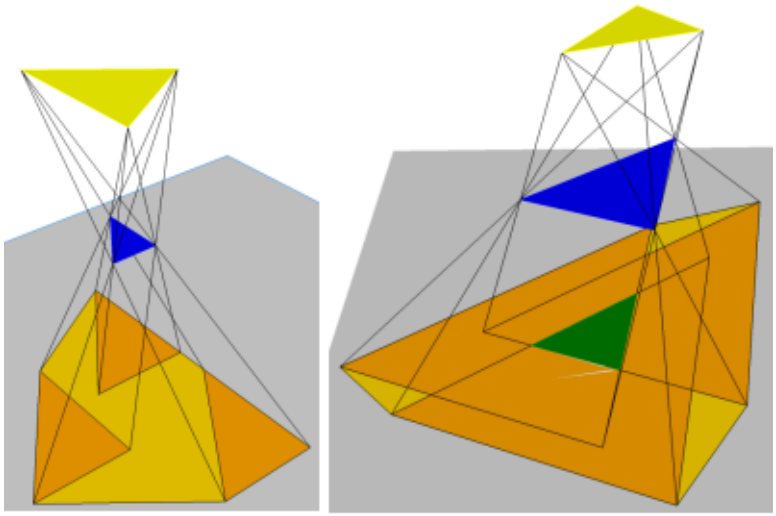
Figuur 5.7: Het clippen van twee blokkers voor de bovenzijde van een cel.



Figuur 5.8: De blokkers voor de linker-achter-onder cel in de Cornell Cube 2 scène. (Voor de definitie van de scène zie bijlage 7.)



Figuur 5.9: De projectie (groene regio) van een blokker (blauw driehoekje) vanuit een coördinaat van de lichtbron op een zijde van een cel.



Figuur 5.10: Links een projectie zonder umbra regio, rechts een projectie met in het groen een umbra regio. De oranje driehoeken zijn de projecties vanuit de coördinaten van de lichtbron. Daarrond wordt de convex hull bepaald.

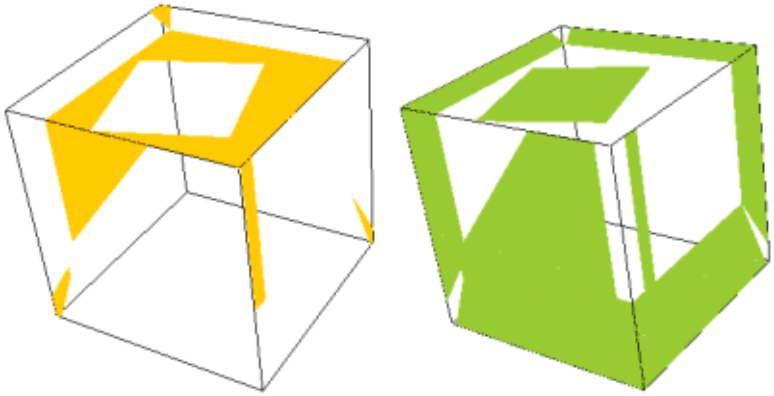
van het vierkant, is de projectie van de blokker op de zijde vanuit een coördinaat van de lichtbron (zie bijvoorbeeld figuur 5.9.

2. Bij het begin van de tweede stap zijn voor alle blokkers alle projecties op de zijde bepaald (de oranje driehoeken in figuur 5.10). Om de umbra regio te bepalen van een blokker op de zijde moet de intersectie van alle projecties van die blokker genomen worden (de groene driehoek in figuur 5.10).

   Dit gebeurt met de algemene polygonclipper uit [Vat92]. Dit is een 2D polygonclippingalgoritme dat polygonen tegen polygonen clipt. Het algoritme kan de unie, het verschil, de intersectie en de logische XOR van twee polygonen bepalen. Hier is de intersectie van toepassing.

   Merk op dat het algoritme werkt in 2D. Dit levert geen probleem gezien de projecties in het vlak van de zijde liggen; aangezien het vlak van de zijde evenwijdig is aan twee van de assen kunnen de 3D punten van de projecties op eenvoudige wijze als 2D punten gebruikt worden.

   Wanneer bijvoorbeeld de zijde evenwijdig is aan het XY-vlak dan zullen alle punten van de projecties dezelfde z coördinaat hebben. Er kan dan gewerkt worden in 2D met de x en y coördinaten.



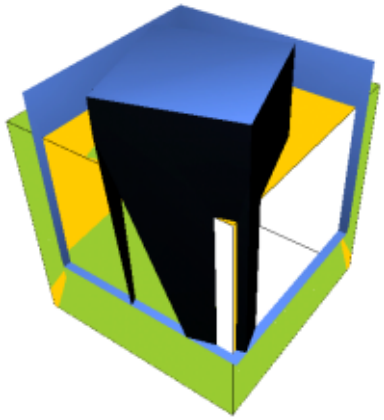
Figuur 5.11: Links de penumbra, rechts de umbra regio voor de linker-achter-onder cel in de Cornell Cube 2 scène. (Voor de definitie van de scène zie bijlage 7.)

3. Daarna wordt de unie van de verschillende umbra regio's genomen, zodat alle umbra regio's op de zijde één polygon vormen. Omdat bij het ray tracen enkel van driehoeken gebruik gemaakt wordt, wordt een triangularisatie van de polygon opgesteld. Dit wordt opnieuw gedaan met het algoritme uit [Vat92]. Een triangularisatie opstellen van een polygon is immers eveneens een van de mogelijkheden van dit algoritme.

4. In de laatste stap wordt de penumbra regio bepaald. Van dezelfde projecties uit de eerste stap wordt per blokker de unie genomen met het algemeen polygonclippingalgoritme. Dit geeft de schaduwregio (umbra en penumbra regio in één) van de blokker vanuit de projectiepunten op de lichtbron. Om de regio voor alle punten op de lichtbron uit te breiden, wordt van de schaduwregio de convex hull genomen (De oranje plus de gele zones in figuur 5.10). Daarvoor wordt het "gift wrap" algoritme gebruikt beter bekend als "Jarvis' March" [Jar73].

   Van alle schaduwregio's van de blokkers wordt opnieuw de unie genomen. Van deze unie wordt het verschil genomen met de in de vorige stap bepaalde umbra regio. Dit verschil vormt de penumbra regio voor de zijde.

Wanneer de umbra regio, de penumbra regio en de blokkers (niet de geclipte, maar de oorspronkelijke) bepaald zijn, worden ze opgeslagen in het gegevensobject. In figuur 5.11 worden de umbra en penumbra regio van een cel uit de Cornell Cube 2 scène afgebeeld.



Figuur 5.12: Alle gegevens in de linker-achter-onder cel van de Cornell Cube 2 scène. (Voor de definitie van de scène zie bijlage 7.)

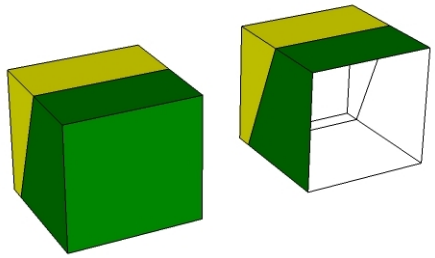
In figuur 5.12 staan alle de blokkers samen met de bijhorende umbra en penumbra regio in een cel afgebeeld.

5.7 Verwijderen van gegevens

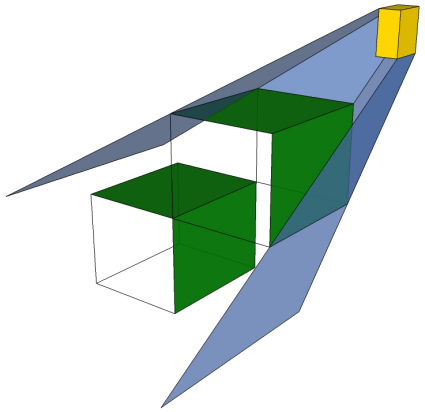
Een zijde die volledig in een umbra regio ligt, ontvangt geen licht meer van de bijhorende lichtbron. Schaduwstralen die vanuit de cel door die zijde gaan, zullen dus gegarandeerd snijden met een blokker. Voor die stralen is het resultaat van de zichtbaarheidsfunctie gegarandeerd 0. Gezien er geen testen op intersectie meer nodig zijn, zijn de gegevens die worden bijgehouden, bijgevolg overbodig geworden.

Bij het aanmaken van de gegevensobjecten voor de zijden van een cel werd enkel een object gecreëerd wanneer er licht van de lichtbron op de zijde viel. Zijden die volledig in een umbra regio vallen, ontvangen geen licht, dus kunnen de gegevensobjecten voor die zijden eveneens verwijderd worden. Voor een schaduwstraal door een zijde naar een lichtbron waarvoor geen gegevensobject bestaat, is onmiddellijk geweten dat er langs die straal geen licht invalt.

Wanneer het oppervlak van de umbra regio op een zijde gelijk is aan het oppervlak van de zijde, ligt de zijde volledig in de umbra regio en kan het bijhorende gegevensobject verwijderd worden. In figuur 5.13 ligt de zijde vooraan volledig in een umbra regio. Het gegevensobject voor die zijde wordt verwijderd.



Figuur 5.13: De zijde vooraan in de cel ligt volledig in een umbra regio (links), het bijhorende gegevens object kan worden verwijderd (rechts).

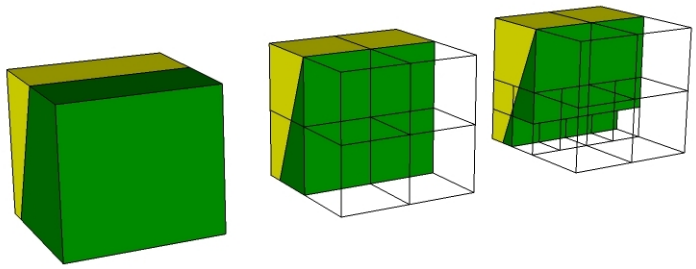


Figuur 5.14: De bovenste cel licht voor de lichtbron langs alle zijden in een umbra regio, bijgevolg ook alle onderliggende cellen.

Wanneer een aantal zijden van een cel volledig in een umbra regio liggen voor een lichtbron, zullen, onder bepaalde voorwaarden, zijden van buren van deze cel eveneens volledig of deels in een umbra regio liggen. De gegevensobjecten van deze naburige zijden kunnen dan, zonder dat blokkers of umbra regio's worden berekend, verwijderd worden. In figuur 5.14 bijvoorbeeld ligt een cel voor alle zijden in een umbra regio van een lichtbron. De cellen die binnen de aangegeven vlakken liggen, worden evenmin door de lichtbron belicht.

De propagatie van informatie over de umbra regio levert een tijdwinst in het constructieproces. Tijdens het renderen wordt hiermee geen extra voordeel behaald.

De propagatie wordt uitgebreider besproken in het volgende hoofdstuk.



Figuur 5.15: Wanneer de linker cel wordt gesplitst, liggen vier kinderen volledig in een umbra regio, waardoor de gegevensobjecten kunnen worden verwijderd (midden). Dit kan nogmaals herhaald worden. (rechts).

5.8 Verfijnen van de octree.

Wanneer een cel slechts gedeeltelijk in een umbra regio ligt, kan het interessant worden om de cel op te splitsen.

Een voorbeeld: in een cel wordt de helft van de geometrie in de cel belicht door een lichtbron, terwijl de andere helft in de schaduw ligt (zie figuur 5.15). Enkel voor de punten die in de schaduwhelft liggen, is het nodig om de blokkers te controleren op intersectie bij het evalueren van de directe belichting. Voor punten in de andere helft is er geen schaduw, dus zullen schaduwstralen geen enkele blokker snijden.

Wanneer die cel opgesplitst wordt, zal in vier van de kinderen geen blokkers gevonden worden en dus ook geen onnodige intersectietesten uitgevoerd worden. Daarnaast kan het gebeuren dat de kinderen in de schaduwhelft volledig in een umbra regio liggen, waardoor gegevensobjecten verwijderd kunnen worden. Zie bijvoorbeeld figuur 5.15. Dit versnelt terug het ray tracing proces.

Tijdens de P-LIE constructie kan er na het bepalen van de blokkers en de umbra en penumbra regio opgesplitst worden. De driehoeken in het oorspronkelijke blad worden opnieuw verdeeld over de nieuwe kinderen. Voor deze kinderen worden op hun beurt de blokkers en de umbra en penumbra regio's bepaald.

Er zijn verschillende criteria mogelijk om te beslissen of een cel opgesplitst dient te worden of niet. Eén criterium werd al vernoemd bij de beschrijving van een LIE: het aantal gedetecteerde blokkers in een cel. Indien dit aantal

een bepaalde grens overtrad werd de cel verfijnd.

De beslissing op basis van meerdere criteria of een cel al dan niet moet worden gesplitst, werd geïmplementeerd door middel van het pattern "Chain of responsibility" uit [GHJV95]. De eventueel te splitsen cel wordt doorgegeven aan de ketting van verschillende objecten die elk volgens een bepaald criterium moeten beslissen of de cel dient gesplitst te worden of niet.

In de nu volgende onderdelen worden twee gebruikte criteria besproken.

5.8.1 Verfijnen volgens aantal blokkers

In de beschrijving van de LIE's werd al aangegeven dat een cel wordt opgesplitst, als voor deze cel te veel blokkers werden gevonden. Als de cel opgesplitst wordt, zal de lijst van blokkers voor elk kind kleiner zijn dan de lijst voor de oorspronkelijke cel. Dit vermindert het aantal intersectietesten die nodig zijn tijdens het ray tracen bij het evalueren van de visibiliteitsfunctie in de renderingvergelijking voor de directe belichting
.
Met een P-LIE zijn er twee mogelijkheden:

- Een grens opleggen aan het totale aantal blokkers in een cel voor een bepaalde lichtbron. Dit komt overeen met het criterium voor LIE's.
- Een grens opleggen aan het aantal blokkers voor een zijde van een cel en een lichtbron.

De tweede mogelijkheid laat toe om op een fijnere wijze een grens op te leggen. In de implementatie werd deze gebruikt.

5.8.2 Verfijnen volgens umbra regio

Zoals eerder vermeld kan het nuttig blijken een cel op te splitsen als de cel slechts voor de helft in de schaduw van een lichtbron ligt. Het voordeel is maximaal wanneer blijkt dat na splitsing slechts één van de kindercellen in een schaduw ligt en de andere zeven niet.

Aan de hand van de umbra regio kan beslist worden om een cel op te splitsen volgens één van deze richtlijnen:

1. Splits als minstens één van de kinderen na de splitsing een zijde zal hebben die volledig in een umbra regio ligt.

2. Splits als minstens één van de kinderen na de splitsing een zijde heeft die noch in een umbra regio, noch in een penumbra regio ligt, of met andere woorden geen blokkers heeft.

Om de eerste richtlijn na te gaan wordt van de umbra regio van de cel de intersectie genomen met de zijde van het kind. Als deze intersectie hetzelfde oppervlak heeft als de zijde van het kind, dan ligt die zijde volledig in een umbra regio.

De tweede richtlijn nagegaan worden door te testen of de zijde van het kind niet overlapt met de umbra of penumbra regio van de cel.

5.9 Parameters en afwegingen

Bij de constructie zijn een aantal parameters gegeven die het constructieproces sturen.

5.9.1 Maximaal aantal driehoeken in een cel

Een eerste parameter is het aantal driehoeken die worden bijgehouden in een blad van de octree. Een groter maximum impliceert minder bladen, en vice versa.

Minder bladen geeft minder werk bij de constructie , maar zorgt er ook voor dat de octreestructuur minder goed aansluit bij de opbouw van de scène: de cellen bevatten meer driehoeken, dus moeten meer driehoeken getest worden op intersectie met een schaduwstraal.

Een afweging moet gemaakt worden tussen enerzijds de resolutie van de octree en anderzijds de complexiteit van de octree voor de constructie. Hoe complexer de octree immers is, des te meer bladeren behandeld moeten worden.

Dit is een moeilijk te bepalen parameter. Ze is sterk afhankelijk van de constitutie van de scène. Over het algemeen gaf een maximum van 200 blokkers goede resultaten, bij één scène was 300 beter.

5.9.2 Maximale diepte van de boom

Een andere parameter die de complexiteit van de boom binnen de perken houdt, is de maximale diepte. Als de boom dieper kan worden, zijn er meer bladeren en is de boom bijgevolg complexer. De waarde voor deze parameter is opnieuw sterk afhankelijk van de scène.

5.9.3 Maximale splitsdiepte

De maximale splitsdiepte geeft het aantal keer dat een cel maximaal mag splitsen door het verfijnen volgens aantal blokkers of umbra regio. Deze grens is zeker nodig bij het verfijnen volgens de umbra regio; het kan gebeuren dat deze verfijning een cel tientallen keren splitst, of zelfs niet stopt met splitsen. Ook bij het verfijnen volgens het aantal blokkers kan er heel diep worden opgesplitst. Een grens is dus zeker nodig.

5.9.4 Maximaal aantal blokkers

Voor de verfijning van de boom volgens het aantal blokkers in een cel moet ook een grens bepaald worden. Opnieuw blijkt dat deze moeilijk te bepalen is. In één scène bijvoorbeeld met een grens van 700 blokkers werden 2000 bladeren bijgemaakt (zelfs al was de maximale splitsdiepte 2), terwijl voor dezelfde scène met een grens van 900 blokkers, slechts 100 bladeren werden bijgemaakt. Voor andere scènes waren dan weer andere waarden beter van toepassing.

Doordat deze grens door "trial and error" bepaald moet worden, lijkt het verfijnen volgens het aantal blokkers ons geen goede methode. Het verfijnen volgens de umbra regio laat zich veel gemakkelijker afstellen en geeft goede resultaten.

Een grens op het aantal blokkers wordt nochtans wel gebruikt bij de constructie van LIE's, maar omdat enkel cellen die in het zicht van de camera vallen kunnen gesplitst worden, zal een te groot aantal opsplitsingen minder voorkomen.

5.10 Conclusie

In dit deel werd beschreven hoe de constructie van een P-LIE in zijn werk gaat. De belangrijkste stappen hierbij zijn het bepalen van de blokkers, en het bepalen van de umbra en penumbra regio's.

De constructie werd geïmplementeerd in C#, zoals ze hier werd voorgesteld. Daarbij bleek dat het clippen van de driehoeken met de algemene polygonclipper de duurste operatie vormt. De constructie kan versneld worden door via de propagatie van umbrainformatie deze operaties zo veel mogelijk te vermijden. Deze propagatie is het onderwerp van het volgende hoofdstuk.

Hoofdstuk 6

Propagatie van gegevens over de umbra

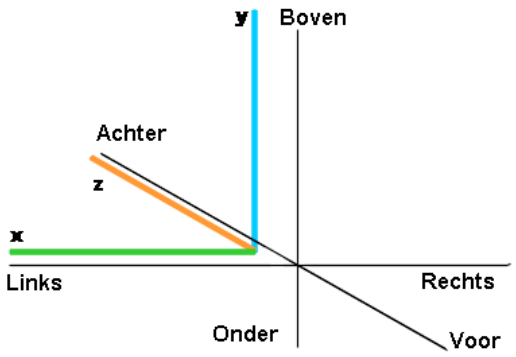
Bij het construeren van de P-LIE wordt voor elk blad van de octree, voor elke zijde en voor elke lichtbron bepaald welke objecten in de scène het licht afschermen, dat van de lichtbronnen toekomt in de zijden. Naast de blokkers wordt ook de umbra regio en de penumbra regio bepaald die de blokkers op de zijde veroorzaken.

Wanneer een zijde volledig in een umbra regio ligt, wordt het gegevensobject voor die lichtbron en die zijde verwijderd; tijdens het renderen is geweten dat, wanneer voor een zijde en lichtbron geen gegevensobject bestaat, er ook geen licht van die lichtbron in die zijde toekomt.

Eenmaal de blokkers en de umbra en penumbra regio voor een zijde bepaald zijn, wordt nagegaan of het gegevensobject al dan niet verwijderd moet worden. Daartoe wordt de oppervlakte van de zijde vergeleken met de oppervlakte van de umbra regio. Als die gelijk zijn, ligt de zijde volledig in de umbra regio en kan het gegevensobject verwijderd worden.

Als zijden volledig in een umbra regio liggen, kan niet alleen het bijhorende gegevensobject verwijderd worden, maar zijn ook een aantal zaken geweten over de zichtbaarheid van de lichtbron in naburige cellen. Als die informatie naar die naburige cellen gepropageerd wordt, kan op voorhand van de naburige zijden bepaald worden of ze al dan niet volledig in een umbra regio liggen. De daarbij horende gegevensobjecten kunnen onmiddellijk zonder verdere berekening van blokkers verwijderd worden.

Deze propagatie van de informatie over de umbra is dus een optimalisatie voor het construeren van de P-LIE, voor het renderen zelf verandert dit niets. Onder propagatie wordt het bepalen van umbra regio's in buren op basis van de umbra configuratie van een cel zelf verstaan.



Figuur 6.1: De zes richtingen langs de x, y en z as.

Afhankelijk van de positie van de lichtbron ten opzichte van de cel en het aantal zijden van de cel die de lichtbron "zien", wordt de informatie over de umbra naar de naburige cellen gepropageerd. In de hierop volgende onderdelen wordt beschreven hoe de verschillende configuraties van onderling relatieve posities aangepakt worden. Daarbij zijn eerst enkele definities nodig.

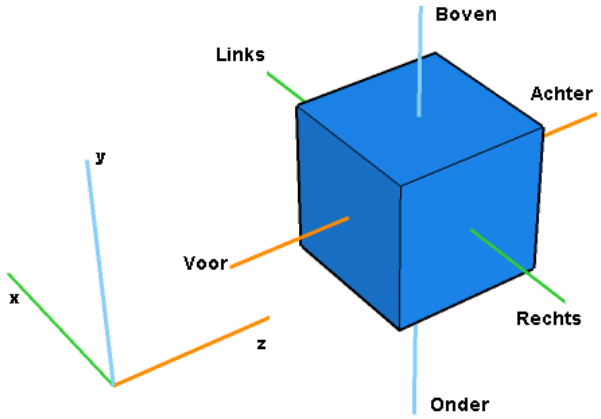
6.1 Definities

Er zijn zes richtingen gedefinieerd langs de assen (zie ook figuur 6.1):

$$\begin{aligned} &Links \\ &Rechts \\ &Voor \\ &Achter \\ &Boven \\ &Onder \end{aligned}$$

Van deze richtingen kan de omgekeerde richting worden gevraagd:

$$\begin{aligned} Links.invers &= Rechts \\ Rechts.invers &= Links \end{aligned}$$



Figuur 6.2: De zes zijden geïdentificeerd aan de hand van de richtingen.

$$\begin{aligned} Voor.invers &= Achter \\ Achter.invers &= Voor \\ Boven.invers &= Onder \\ Onder.invers &= Boven \end{aligned}$$

Elke zijde van een cel kan ten opzichte van de assen geïdentificeerd worden aan de hand van een richting (zie figuur 6.2).

Van een cel kan een buur worden opgevraagd:

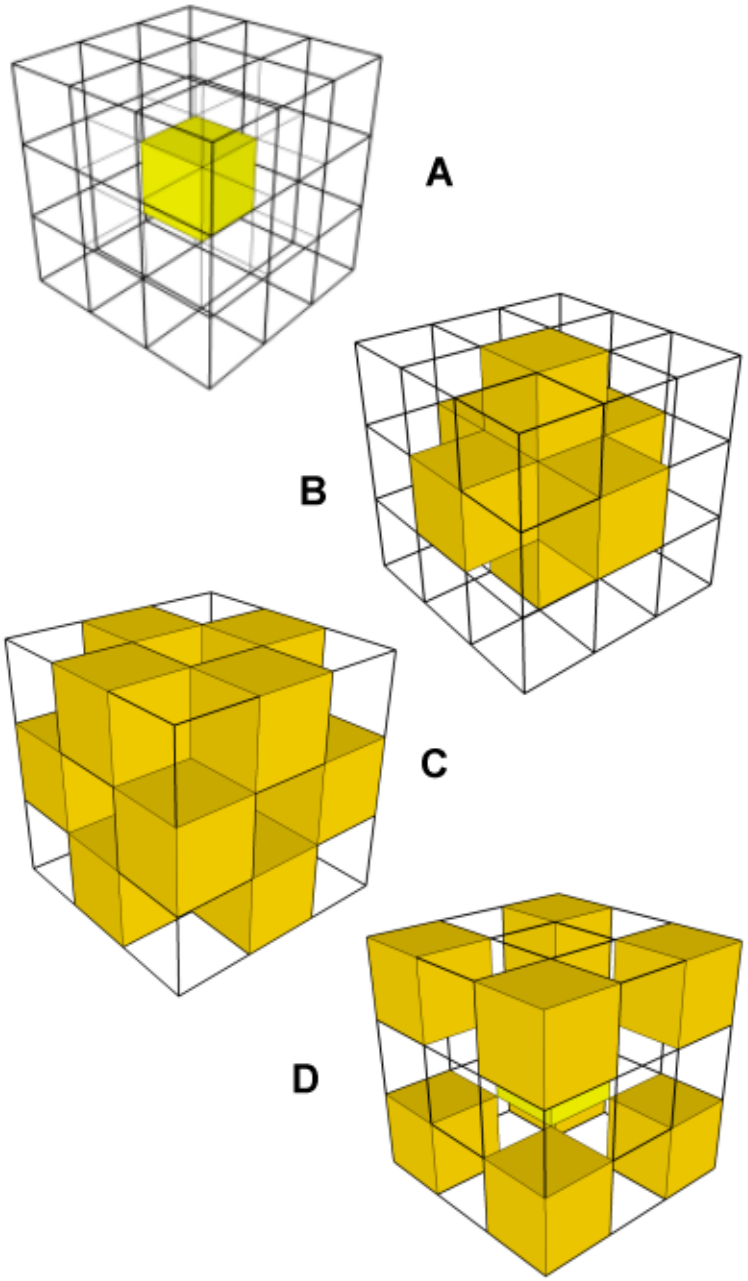
$$cel.buur(Richting) \tag{6.1}$$
$$cel.buur(RichtingA, RichtingB) \tag{6.2}$$
$$cel.buur(RichtingA, RichtingB, RichtingC) \tag{6.3}$$

Als parameters krijgt deze functie één, twee of drie richtingen mee. Zij bepalen welke buur bedoeld wordt. Een cel heeft 26 buren (zie figuur 6.3.A).

1. Van deze 26 buren kunnen 6 buren die een zijde delen met de cel worden opgevraagd met functie 6.1 (zie figuur 6.3.B).

2. 12 van de 26 buren delen een ribbe met de cel en kunnen worden opgevraagd met functie 6.2 (zie figuur 6.3.C).

3. De overblijvende 8 buren delen een hoekpunt met de cel en kunnen worden opgevraagd met functie 6.3 (zie figuur 6.3.D).



Figuur 6.3: A) De 26 buren rond een cel. B) De 6 buren die een zijde delen. C) De 12 buren die een ribbe delen. D) De 8 buren die een knooppunt delen.

Van een zijde kan de overeenkomende richting gevraagd worden:

$$zijde.richting \tag{6.4}$$

Bijvoorbeeld de bovenste zijde zal *Boven* as resultaat geven, de linkse zijde *Links*.

6.2 Algemene propagatie

Stel dat een lichtbron zichtbaar is in één zijde van een cel. De lichtbron wordt echter afgeschermd door blokkers. De zijde ligt dus in een umbra regio. Dan zal in de buur aan de tegenovergestelde zijde, de zijde in dezelfde richting ook in een umbra regio liggen.

Dit kan veralgemeend worden: wanneer een zijde volledig in een umbra regio ligt, dan zal in de buur die de tegenovergestelde zijde deelt, de zijde in dezelfde richting ook in een umbra regio liggen, ongeacht het aantal zijden die de lichtbron kunnen zien, in die buur, of in de eerste cel. De enige nodige voorwaarde is dat alle zijden van de eerste cel waarvoor de lichtbron zichtbaar is, in een umbra regio liggen.

Dit geldt in alle gevallen, en zal daarom in de aparte bespreking niet herhaald worden.

6.3 Lichtbron zichtbaar in één zijde

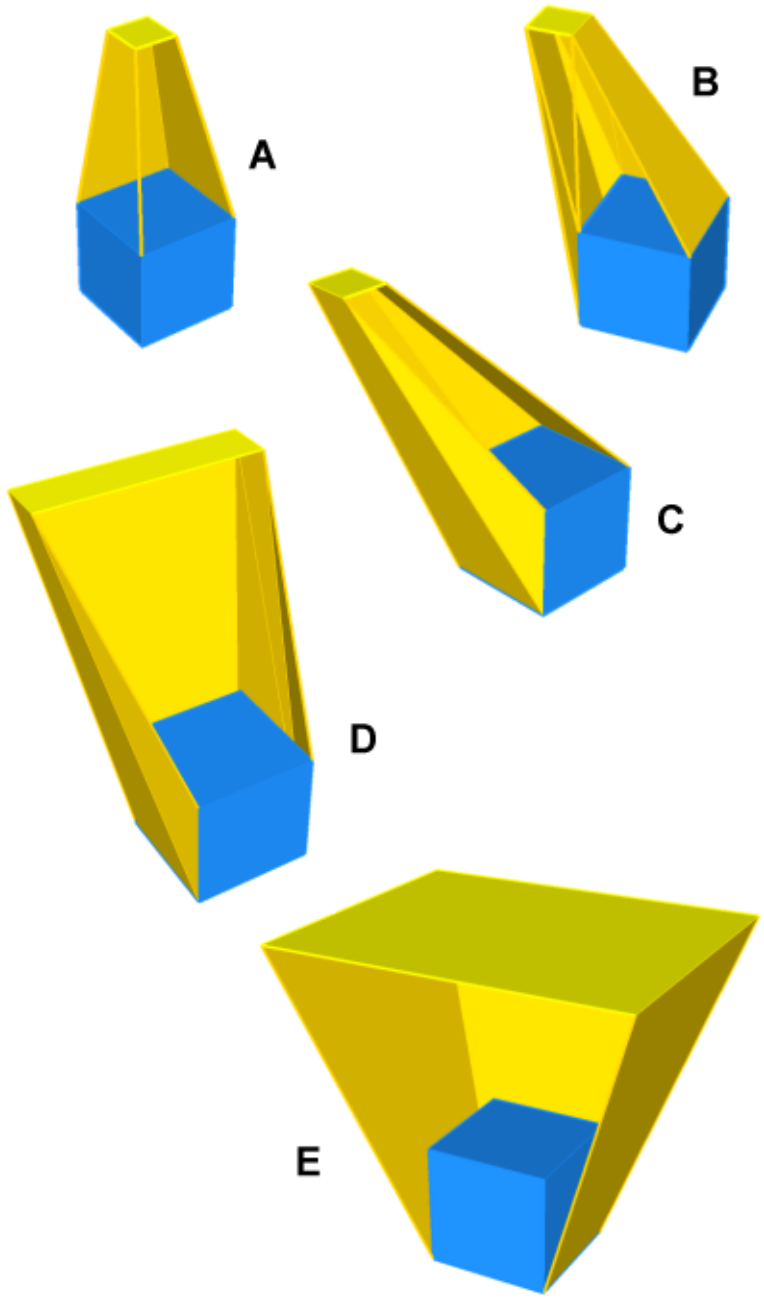
Als de lichtbron op slechts één zijde van een cel licht afwerpt (zie figuur 6.4.A) en de zijde door blokkers volledig in de umbra regio ligt, dan zullen alle zijden van de buur, grenzend aan de overstaande zijde van de cel, ook in een umbra regio liggen (zie figuur 6.5).

Bijvoorbeeld: de ene zijde van de cel in de umbra regio is de bovenzijde (*Boven*), dan kunnen alle gegevensobjecten voor dezelfde lichtbron in $cel.buur(Boven.invers)$ verwijderd worden.

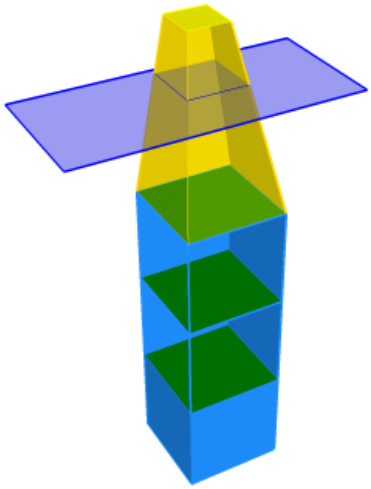
Als één van deze buren geen blad is, maar een knoop, dan kunnen alle gegevensobjecten voor die lichtbron in alle kinderen van die knoop recursief verwijderd worden.

Met functie 6.5 kan de buur die grenst aan de overstaande zijde opgevraagd worden. Hiermee kan het verwijderen recursief over alle buren uitgevoerd worden.

$$cel.buur(zijde.richting.invers) \tag{6.5}$$



Figuur 6.4: A) De lichtbron is zichtbaar in één zijde. B) De lichtbron is zichtbaar in twee zijden. C) In drie zijden. D) In vier zijden. E) In vijf zijden.



Figuur 6.5: De enige zijde van de bovenste cel die de lichtbron ziet, valt in een umbra regio, dus ook dezelfde zijden in de onderliggende cellen.

De recursie stopt wanneer de grens van de octree bereikt wordt of een buur een blad is van een hoger niveau in de boom.

6.4 Lichtbron zichtbaar in twee zijden

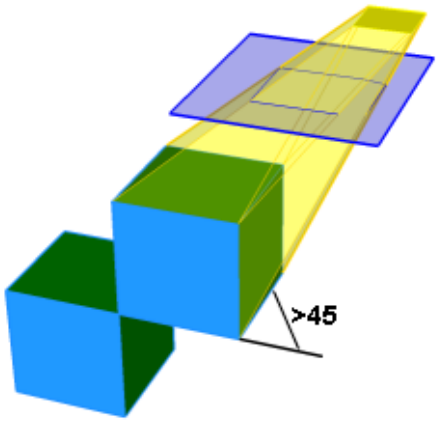
Wanneer een lichtbron op twee zijden van een cel licht afwerpt (zie bijvoorbeeld figuur 6.4.B), zijn er twee mogelijkheden.

6.4.1 Twee zijden in umbra

Stel dat de richtingen van de twee zijden R_A en R_B zijn. Beide zijden liggen volledig in een umbra regio. Voor beide zijden zijn er in het bijhorende gegevensobject drie vlakken die een links georiënteerde hoek maken met de normaal van de zijde en één vlak dat een rechts georiënteerde hoek maakt met de normaal van de zijde.

In de tegenovergestelde buur kunnen alle gegevensobjecten voor de lichtbron verwijderd worden, als de links geöriënteerde vlakken tegenover de rechts geöriënteerde vlakken een hoek groter dan 45 graden met de normaal van de zijde maken.

De tegenovergestelde buur is diegene die met de cel de ribbe deelt die



Figuur 6.6: Beide cellen die de lichtbron zien liggen in de umbra van de blokker, dus ook de cellen aan de overstaande zijde.

tegenover de ribbe die de twee zijden delen staat (zie figuur 6.6). Deze buur kan worden opgevraagd met:

$$cel.buur(R_A.invers, R_B.invers) \tag{6.6}$$

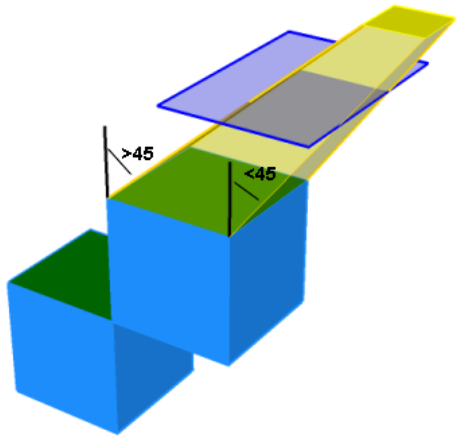
Indien de buur een knoop is in plaats van een blad, dan kan in alle kinderen van de knoop de gegevensobjecten voor de lichtbron recursief verwijderd worden.

Opnieuw kan de verwijdering recursief worden toegepast, door telkens de overstaande buur met functie 6.6 op te vragen. De recursie stopt wanneer de grens van de octree wordt bereikt of een buur zich hoger in de boom bevindt dan de huidige cel.

6.4.2 Eén van de twee zijden in umbra

Stel dat de richting van de zijde die in de umbra regio ligt, R_A is. De richting van de andere zijde die deels of niet in een umbra regio ligt is R_B. De tegenoverstaande buur heeft dezelfde zijde in umbra regio als:

- de rechts georiënteerde hoek met de normaal van de zijde die in de umbra regio ligt kleiner is dan 45 graden
- en de overstaande links georiënteerde hoek met de normaal van de zijde groter is dan 45 graden.



Figuur 6.7: Eén van de cellen die de lichtbron zien ligt in de umbra van de blokker, de andere niet.

Onder deze voorwaarden zal de zijde die in richting R_A van de buur functie 6.6 bekomen wordt, ook in een umbra regio liggen voor de lichtbron (zie bijvoorbeeld figuur 6.7).

6.5 Lichtbron zichtbaar in drie zijden

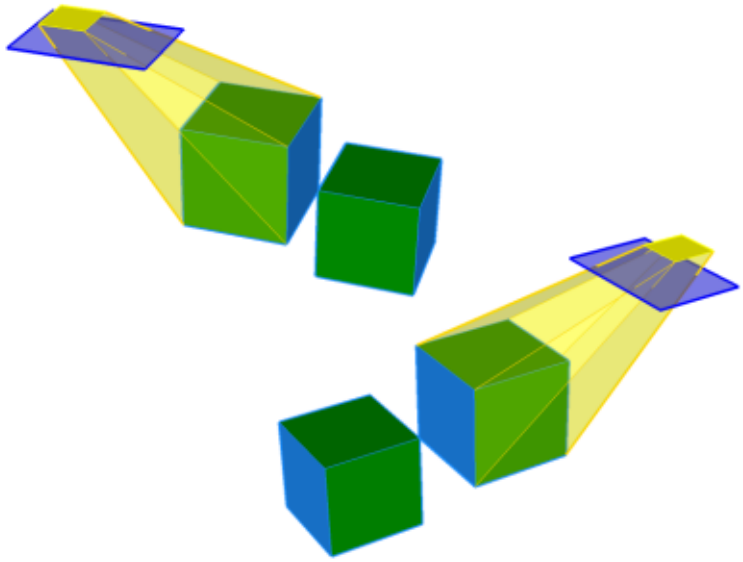
Wanneer een lichtbron op drie zijden van een cel licht afwerpt (zie bijvoorbeeld figuur 6.4.C) zijn er drie mogelijkheden. De richtingen van de drie zijden worden aangeduid met R_A, R_B en R_C.

6.5.1 Drie zijden in umbra

Wanneer alle drie de zijden volledig in een umbra regio liggen, kan in de tegenoverstaande buur alle gegevensobjecten voor de bijhorende lichtbron worden verwijderd. De tegenoverstaande buur is in dit geval de buur die met de cel het hoekpunt deelt die tegenover het hoekpunt dat de drie zijden delen staat. Deze kan worden opgevraagd met:

$$cel.buur(R_A.invers, R_B.invers, R_C.invers) \tag{6.7}$$

Ook hier moeten alle links georiënteerde hoeken van de vlakken met de normalen van de respectievelijke zijden groter zijn dan 45 graden. (zie figuur 6.8).



Figuur 6.8: Drie zijden in een umbra regio

Opnieuw geldt dat, als de buur een knoop is in plaats van een blad, in alle bladeren van de knoop de gegevensobjecten voor de lichtbron verwijderd kunnen worden.

Bovendien kan opnieuw de verwijdering recursief toegepast worden door telkens de overstaande buur met functie 6.7 op te vragen.

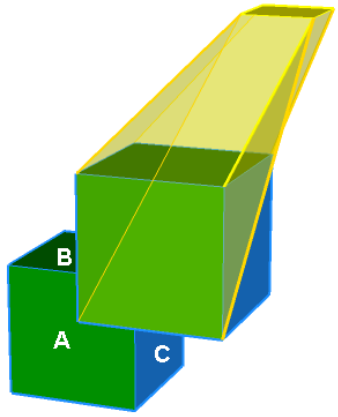
6.5.2 Twee van de drie zijden in umbra

Stel dat de zijde in richting R_C niet of deels in een umbra regio ligt en de twee andere zijden R_A en R_B volledig in een umbra regio liggen.

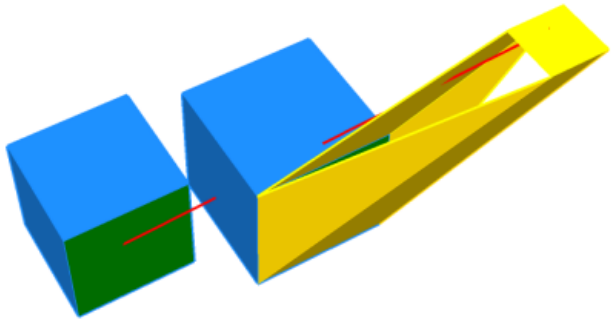
In dit geval geldt ongeveer hetzelfde wanneer twee zijden de lichtbron zien en één ervan in de umbra regio ligt, namelijk:

- De links georiënteerde hoeken van de vlakken met de normalen van de zijden R_A en R_B moeten groter zijn dan 45 graden.
- De rechts georiënteerde hoeken van de vlakken, die door de ribbe gaan die gedeeld wordt met de zijde R_C, moeten kleiner zijn dan 45 graden.

Als bovenstaande twee voorwaarden gelden, zullen dezelfde zijden in de buren, recursief opgevraagd met functie 6.7, ook in een umbra regio liggen voor de lichtbron (zie figuur 6.9).



Figuur 6.9: Twee zijden van de drie in een umbra regio.



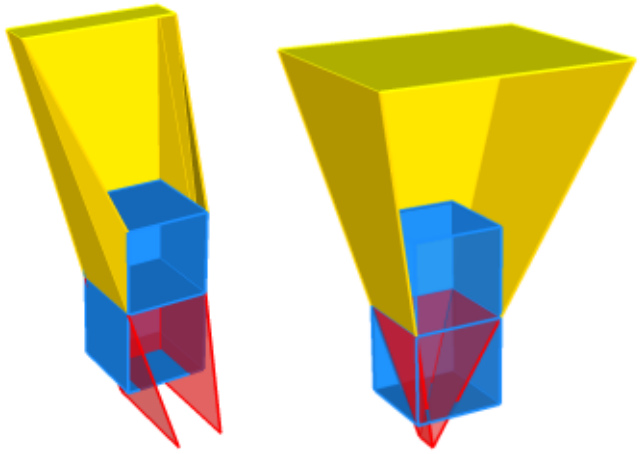
Figuur 6.10: Eén van de drie zijden in een umbra regio, de andere twee niet. De rechts geöriënteerde hoeken zijn echter te groot; de rode straal vanuit de buur raakt tot de lichtbron.

6.5.3 Eén van de drie zijden in umbra

In dit geval ligt één van de drie zijden van een cel, waarin de lichtbron zichtbaar is, volledig in een umbra regio voor de lichtbron, de andere twee niet (zie figuur 6.10). Deze zijde zal twee links en twee rechts georiënteerde hoeken voor de vlakken hebben (de andere twee ook).

De buur opgevraagd met functie 6.7 zal dezelfde zijde in een umbra regio hebben liggen als:

- de twee links georiënteerde hoeken van de vlakken met de normaal van de zijde in umbra groter zijn dan 45 graden.
- de twee rechts georiënteerde hoeken van de vlakken met de normaal



Figuur 6.11: Hoewel alle zijden die de lichtbron zien eventueel in de umbra regio liggen, kan niets gezegd worden over de umbra regio in naburige cellen.

van de zijde in umbra kleiner zijn dan 45 graden (zie figuur 6.10).

6.6 Lichtbron zichtbaar in vier en vijf zijden

Als een lichtbron zichtbaar is in een cel voor vier of vijf zijden (zes zijden kan niet) kan geen uitspraak gemaakt worden over umbra regio's in de naburige cellen. De buren vallen immers niet meer in de "scope"van de lichtbron: de vlakken gaan door de naburige cellen, ongeacht de hoek met de normaal van de zijde, in plaats van er rond te liggen (zie figuur 6.11).

Het algemene geval blijft wel geldig.

6.7 Propagatieregels samengevat

Op basis van voorgaande beschrijving van de voorwaarden om informatie over de umbra te kunnen propageren, kan een volgende samenvatting van deze regels opgesteld worden:

- *Algemene regel* : Alle zijden voor een lichtbron in umbra : dezelfde zijde van de overstaande buur ook.
- *Eén zijde* : dezelfde zijde van de overstaande buur ook.
- *Twee zijden* :

  - *Beide zijden in umbra* : de linkse hoeken tegenover de rechtse hoeken zijn groter dan 45 graden : dezelfde zijden van de overstaande buur liggen ook in een umbra regio.
  - *Eén van de twee zijden in umbra* : de linkse hoek van de umbra zijde is groter dan 45 graden en de rechtse hoek van de umbra zijde is kleiner dan 45 graden : dezelfde zijde in de overstaande buur ligt ook in een umbra regio.

- *Drie zijden* :
  - *Drie zijden in umbra* : de linkse hoeken zijn groter dan 45 graden: dezelfde zijden in de overstaande buur liggen ook in een umbra regio.
  - *Twee van de drie zijden in umbra* : de linkse hoeken groter dan 45 graden in de twee umbra zijden en de rechtse hoeken voor de ribbe die gedeeld worden met de zijde die niet in de umbra regio ligt kleiner dan 45 graden : dezelfde zijden in de overstaande buur liggen ook in een umbra regio.
  - *Eén van de drie zijden in umbra* : twee linkse hoeken groter dan 45 graden en twee rechtse kleiner dan 45 graden : dezelfde zijde in de overstaande buur ligt ook in een umbra regio.

- *Vier zijden* : Naast de algemene regel geen propagatie.
- *Vijf zijden* : Naast de algemene regel geen propagatie.

6.8 Conclusie

In dit hoofdstuk werd de mogelijkheid tot propagatie van informatie over de umbra over de octree besproken en toegelicht.

Wanneer bepaalde zijden van een cel volledig in een umbra regio liggen, kunnen onder bepaalde voorwaarden voor de hoeken tussen de zijde en de lichtbron uitspraken gemaakt worden over de umbra regio's voor zijden in naburige cellen. Hierdoor moeten in die zijden geen blokkers bepaald worden, wat de constructie versnelt.

De propagatie is een optimalisatie voor de constructie van de P-LIE, maar heeft geen invloed op het renderen.

Wanneer de voorwaarden meer worden gespecificeerd, kunnen deze regels uitgebreid worden. Als bijvoorbeeld de lichtbron in één zijde van een cel

zichtbaar is, kan gesteld worden dat, als de hoeken van vlakken groter zijn dan 45 graden, ook naburige cellen van de tegenoverstaande buur in de umbra regio liggen.

Dit is slechts één voorbeeld, nog veel andere zijn te geven. Die zullen specifieke voorwaarden op de waarden van de hoeken leggen. Hoe specifieker echter de voorwaarden, hoe minder frequent de situatie zich zal voordoen in de scène, waardoor op een zeker niveau het nagaan van de voorwaarden niet meer zal opleveren.

Deel III

Resultaten

Hoofdstuk 7

Test scènes

In deze bijlage worden kort de 10 scènes die gebruikt werden bij de testen voorgesteld. De afbeeldingen van de scènes zijn telkens resultaten van de ray tracer, met de volgende instellingen:

- 398×398 pixels.
- 4 zichtstralen per pixel, regulier verdeeld.
- 25 schaduwstralen per lichtbron, verdeeld volgens het N-Rooks schema.
- Het ray tracen gebeurde met de P-LIE structuur.

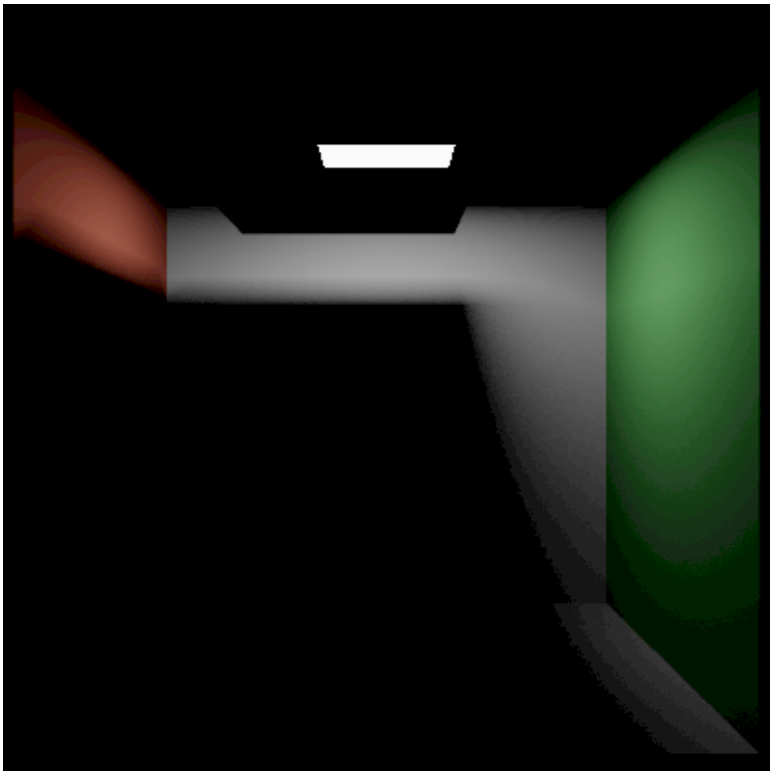
7.1 Cornell Cube 1

De eerste scène is een cornell cube (figuur 7.1, met een vlak dat net onder de lichtbron zweeft. De cellen van de octree zullen dus allemaal dit vlak als één van de blokkers hebben.

Deze scène werd in de eerste plaats gemaakt om tijdens het implementeren gemakkelijk de bepaling van de umbra en de penumbra regio te kunnen testen. Zo ligt bijvoorbeeld de rechter-onder-voorste cel voor elke zijde in een umbra regio.

Specificaties:

- 16 driehoeken
- 1 lichtbron

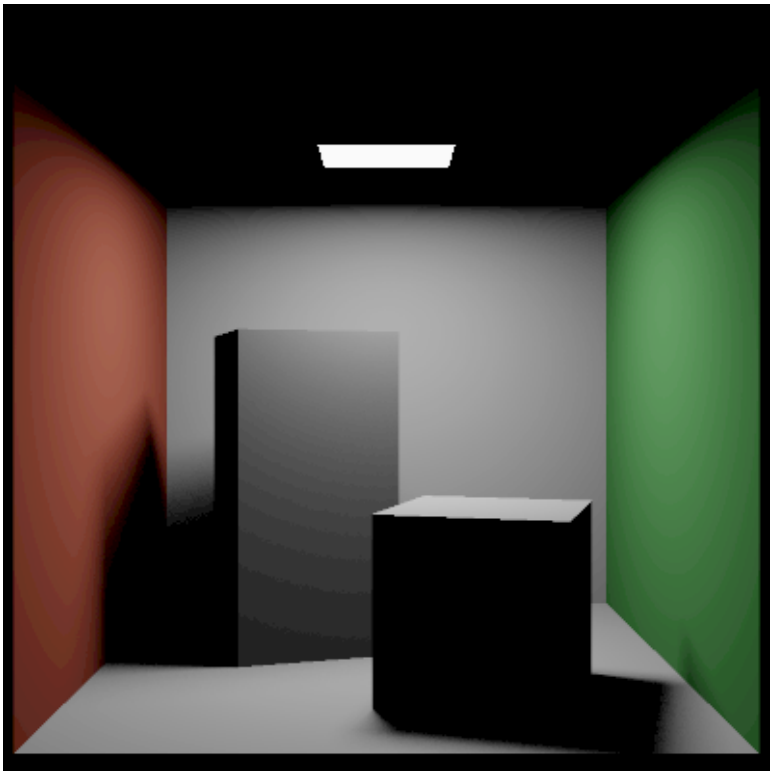


Figuur 7.1: Cornell Cube 1

7.2 Cornell Cube 2

Cornell cube 2 is de klassieke cornell cube (figuur 7.2). Ook deze scène werd vooral gebruikt bij het testen tijdens de implementatie. Doordat er weinig driehoeken zijn bestaat de octree voor de scène uit acht cellen. Wanneer de verfijning volgens umbra regio worden enkel de rechter-onder-voor en de linker-onder-achter cel verfijnd. De twee andere cellen hebben meer verspreide umbra regio's waardoor niet gesplitst wordt.
Specificaties:

- 32 driehoeken
- 1 lichtbron



Figuur 7.2: Cornell Cube 2

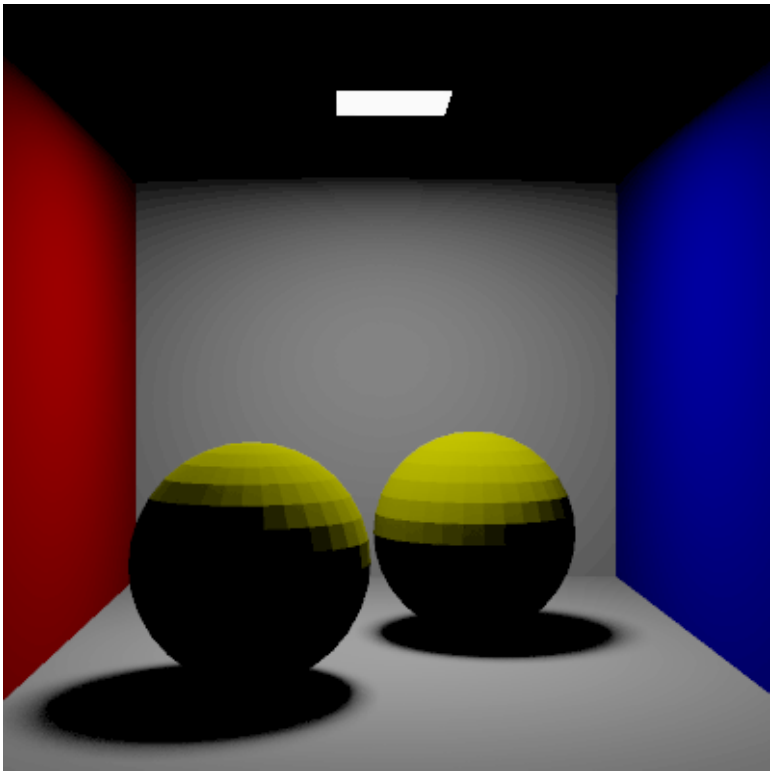
7.3 Cornell Cube 3

Cornell cube 3 is een cornell cube met twee bollen die worden benaderd met driehoeken (figuur 7.2). De cellen van de octree concentreren zich vooral op de twee bollen. Met verfijning volgens umbra regio worden extra cellen bijgemaakt in de schaduw regio's van deze bollen.
Specificaties:

- 1932 driehoeken
- 1 lichtbron

7.4 Cornell Cube 4

Cornell cube 4 is Cornell cube 2 met twee lichtbronnen (figuur 7.2). De splitsing volgens umbra regio heeft nu enkel effect op de linker-onder-voor en de linker-onder-achter cel.



Figuur 7.3: Cornell Cube 3

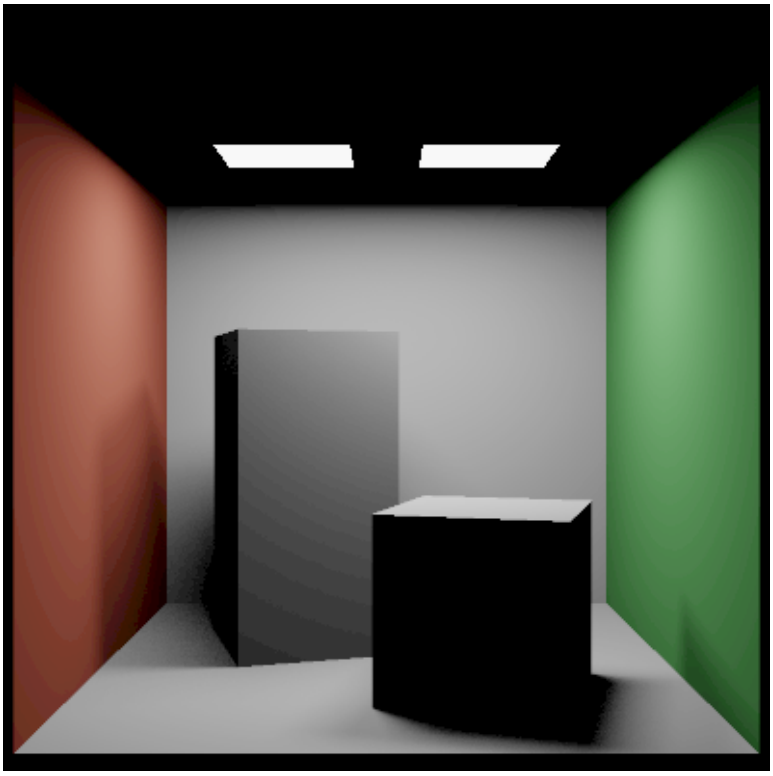
Specificaties:

- 34 driehoeken
- 2 lichtbronnen

7.5 Cornell Cube 5

Cornell cube 5 is Cornell cube 1 met twee lichtbronnen (figuur 7.2). Bij de verfijning worden in vergelijking met Cornell Cube 1 veel meer cellen aangemaakt rond de muren van de kubus.
Specificaties:

- 18 driehoeken
- 2 lichtbronnen



Figuur 7.4: Cornell Cube 4

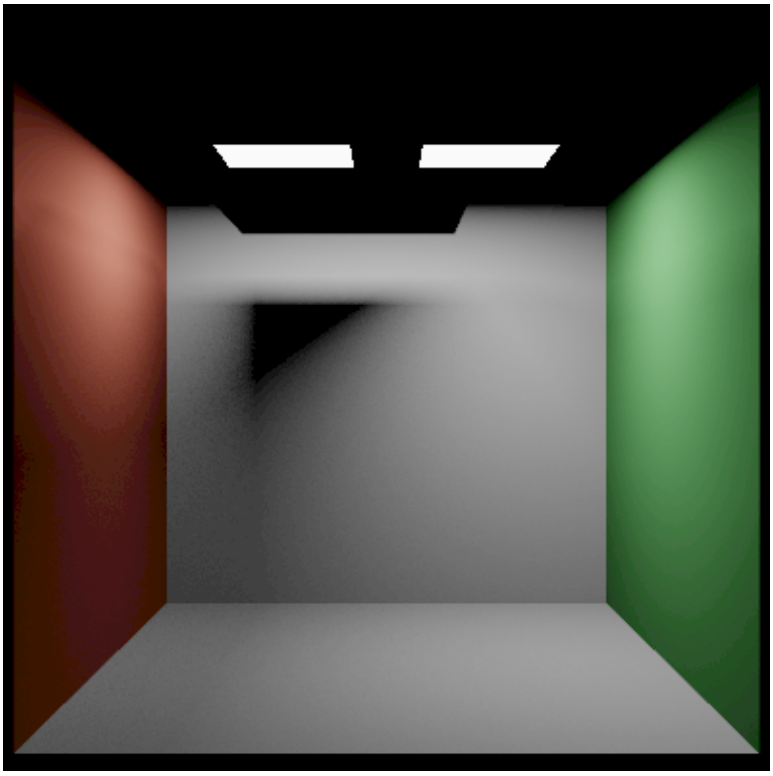
7.6 Huis A

Huis A heeft één centrale lichtbron in een kamer die ingericht is als een bureau. Tegen de muur staan twee boekenkasten, in de kamer staan twee tafels en een bureaustoel. Complexe schaduwen bevinden zich vooral onderaan de stoel en tussen de planken van de boekenkast. Op die plaatsen is de concentratie cellen van de octree het hoogst.
Specificaties:

- 4913 driehoeken
- 1 lichtbron

7.7 Huis B

Deze scène komt overeen met Huis A, de kleuren werden aangepast, de kasten bestaan uit een meer eenvoudige mesh en op het bureau werd een flatscreen



Figuur 7.5: Cornell Cube 5

geplaatst. Door de 2 lichtbronnen verdubbelt het werk voor de constructie. Specificaties:

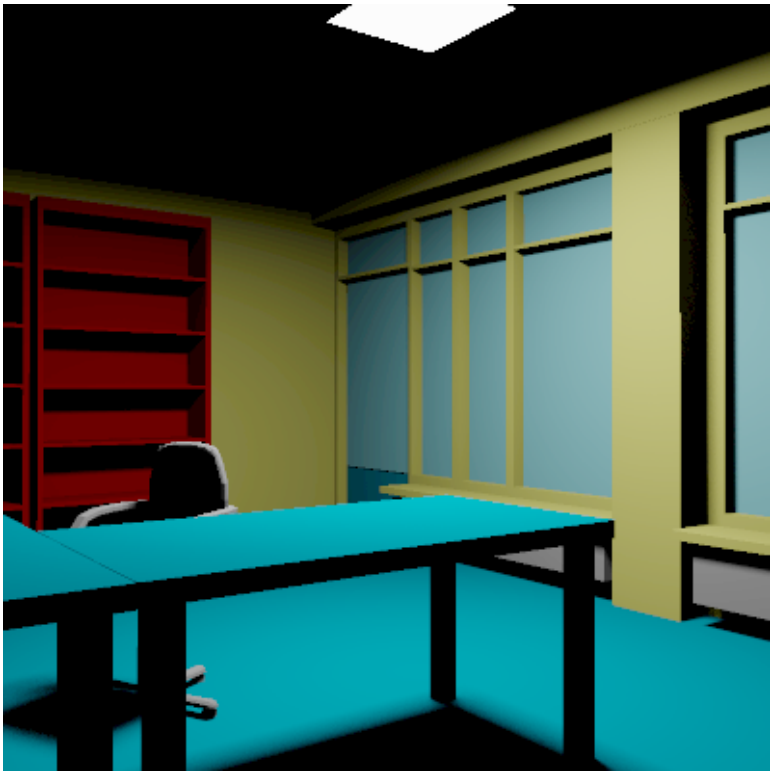
- 5789 driehoeken
- 1 lichtbron

7.8 Huis C

Huis C is dezelfde scène als Huis B (figuur 7.8), maar met twee lichtbronnen. Slechts één lichtbron valt binnen het zicht van de camera.

Specificaties:

- 5791 driehoeken
- 2 lichtbronnen



Figuur 7.6: Huis A

7.9 Dragon

Deze scène komt grotendeels overeen met Cornell Cube 2. Hier werd een draakje geplaatst op de rechter balk in de scène. Aangezien de kubus en de balken gevormd zijn met 32 driehoeken, bestaat het draakje uit de overige 2734 driehoeken. Hierdoor concentreren de cellen van de octree zich vooral rond de draak.

Specificaties:

- 2766 driehoeken
- 1 lichtbron

Figuur 7.7: Huis B

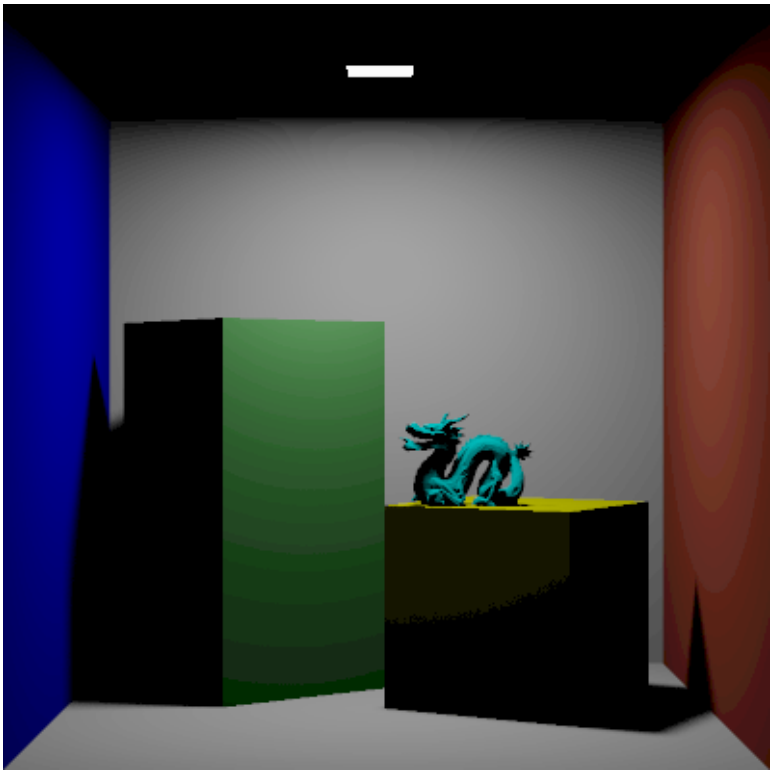
7.10 Boot

Deze scène bevat een mesh van een boot en één relatief veraf geplaatste lichtbron die niet in beeld valt. In deze scène zijn nauwelijks schaduwen aanwezig, zodat de bovenzijden van de cellen in de octree over het algemeen geen blokkers zullen hebben. De andere zijden van de cellen snijden met de boot en hebben zodoende een umbra en penumbra regio.

Specificaties:

- 3974 driehoeken
- 1 lichtbron

Figuur 7.8: Huis C



Figuur 7.9: Dragon

Figuur 7.10: Boot

Hoofdstuk 8

Resultaten

In het voorgaande deel werd een voorberekende lokale belichtingsomgeving (P-LIE) voorgesteld, die het renderen van een scène met ray tracing kan versnellen. De visibiliteitsfunctie uit de renderingvergelijking voor de directe belichting wordt sneller geëvalueerd. In dit hoofdstuk wordt onder andere deze versnelling getest.

Voor de testen te kunnen uitvoeren werd een ray tracer geschreven. In het eerste onderdeel van dit hoofdstuk wordt een hoog-niveau beschrijving gegeven van deze implementatie.

Daarna volgt de bespreking van de drie testen die werden uitgevoerd, samen met enkele conclusies op basis van de resultaten. Een uitgebreide conclusie op basis van deze testen is aan de orde in het hierop volgende hoofdstuk.

8.1 Ray tracen met P-LIE's

De implementatie van de ray tracer werd net als de constructie van de P-LIE geschreven in C#. De testen met deze implementatie werden uitgevoerd op een Pentium IV 1.9GHz, met 256Mb werkgeheugen.

8.1.1 De ray tracer

Het renderen van een scène gebeurt in een aantal stappen:

1. Een "hitbuffer" wordt gecreëerd.

   Vanuit het oogpunt van de camera worden een aantal zichtstralen gesneden met de objecten in de scène. Om de testen op intersectie te

versnellen wordt de octree gebruikt die gegeven met P-LIE-structuur. Het snijpunt van een zichtstraal met de objecten uit de scène dat het dichtst bij het oogpunt van de camera ligt wordt samen met enkele andere gegevens bijgehouden in een hitbuffer:

- De coördinaten van het snijpunt.
- De normaal in dat snijpunt, die later zal gebruikt worden bij het evalueren van de renderingvergelijking.
- De richting van de zichtstraal, die eveneens later zal gebruikt worden bij het evalueren van de renderingvergelijking.
- Een referentie naar het object waarmee werd gesneden, zodat bepaalde de materiaaleigenschappen bij de evaluatie van de renderingvergelijking kunnen opgevraagd worden.
- Een referentie naar het blad van de octree waarin het snijpunt zich bevindt.

2. De belichting wordt geëvalueerd. Dit is de zogenaamde "shading" stap.

   Voor elk snijpunt in de hitbuffer wordt de gereflecteerde radiantie vanuit dat punt in de omgekeerde richting van de zichtstraal berekend. Daartoe moet de renderingvergelijking voor de directe belichting berekend worden. Die werd eerder al gegeven in hoofdstuk 2.

   De rendering vergelijking voor de directe belichting is een integraal over de oppervlakte van alle lichtbronnen in de scène. Voor de evaluatie van deze integraal wordt Monte Carlo gebruikt.

   Voor elke lichtbron (N_L lichtbronnen in totaal) worden N punten (y_i) op de lichtbron volgens een bemonsteringsschema bepaald. Voor elk punt wordt de volgende term berekend:

$$A_{L_i} L_e(y_i \rightarrow \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \leftrightarrow \Theta) G(x, y_i) V(x, y_i) \tag{8.1}$$

   Bij de evaluatie van de visibiliteitsfunctie $V(x, y_i)$ wordt de P-LIE structuur gebruikt.

   Vanuit het snijpunt x wordt een schaduwstraal richting y_i gecontroleerd op intersectie met de objecten in de scène. Als deze straal inderdaad snijdt met een object dan komt geen licht vanuit het punt y_i toe in het punt x. Het resultaat van de visibiliteitsfunctie is in dat geval 0. Wanneer geen intersectie gevonden wordt, is het resultaat 1.

De evaluatie van de visibiliteitsfunctie kan met de P-LIE structuur op vier mogelijke wijzen uitgevoerd worden:

(a) Controleer de driehoeken in de bladeren waar de schaduwstraal door gaat op intersectie met die straal. Dit kan gebeuren met de octree, zonder dat gegevens uit de P-LIE structuur gebruikt moeten worden.

(b) Controleer de driehoeken in de lijst van blokkers voor de zijde waar de straal doorheen gaat en de lichtbron waarop y_i zich bevindt. Als geen intersectie gevonden wordt, test dan ook de driehoeken in het blad waar x in ligt.

(c) Test eerst of de schaduwstraal snijdt met de umbra regio van de zijde voor de lichtbron waarop y_i ligt. Als een intersectie gevonden wordt is het resultaat van de visibiliteitsfunctie 0, anders moet mogelijkheid (b) uitgevoerd worden.

(d) Als geen intersectie gevonden wordt voor de schaduwstraal en de umbra regio kan eerst nog gecontroleerd worden of de schaduwstraal snijdt met de penumbra regio. Indien dit niet het geval is moet in mogelijkheid (b) niet meer gecontroleerd worden of de straal snijdt met de blokkers. In het andere geval moet mogelijkheid (b) volledig uitgevoerd worden.

3. Wanneer de radiantie in alle punten van de hitbuffer gekend is, kan van deze waarden een visualisatie gegenereerd worden.

8.1.2 Instellingen voor de constructie

Voor het construeren van de P-LIE moeten een aantal instellingen bepaald worden, namelijk:

- De maximale diepte van de boom.
- Het maximaal aantal driehoeken in een blad van de octree.
- Het al dan niet verfijnen van de P-LIE aan de hand van de umbra regio. Als hiervoor gekozen wordt, moet ook de maximale splitsdiepte bepaald worden.
- Het al dan niet verfijnen van de P-LIE aan de hand van het aantal blokkers voor een zijde. Als hiervoor gekozen wordt, moet de maximale splitsdiepte én het maximaal aantal blokkers voor een zijde bepaald worden.

De invloed van de maximale diepte voor de boom op de snelheid van het renderen werd niet getest. De diepte van de boom ging in geen van de gebruikte testscènes boven acht vertakkingniveau's. Het is vooral het maximaal aantal driehoeken in een blad van de octree die de diepte van de boom beïnvloed. Wanneer minder driehoeken toegelaten worden zullen er in de eerste plaats meer bladeren zijn (en de boom bijgevolg dieper) en zal de structuur van de octree beter aansluiten bij de geometrie van de scène.

8.2 Test op de snelheid van de P-LIE

Als eerste werd de snelheid van het renderen met P-LIE's getest en vergeleken met de snelheid van het renderen met een octree.

8.2.1 Hoe getest werd

Voor de test werden in totaal 10 scènes gebruikt. Voor elke scène werd een P-LIE geconstrueerd met volgende instellingen:

- maximaal 200 driehoeken in een blad,
- geen verfijning.

Voor elke scène werd de *duur van het evalueren van de belichting* in alle snijpunten van de hitbuffer gemeten. De duur van het bepalen van de snijpunten in de hitbuffer wordt niet meegerekend, dit gebeurt immers los van de P-LIE waardoor de duur bijgevolg gelijk zal zijn.

Voor elke scène worden vier waarden gemeten, namelijk de tijd voor de evaluatie van de radiantie voor alle snijpunten in de hitbuffer met behulp van

1. de octree (de OCTREE tijd),
2. de voorberekende blokkers (de P-LIE tijd),
3. de voorberekende blokkers en umbra regio (de UMBRA tijd),
4. de voorberekende blokkers, umbra en penumbra regio (de PENUMBRA tijd).

Dit telkens voor:

1. 126 op 126 zichtstralen en
2. 254 op 254 zichtstralen.

	4 schaduwstralen	9 schaduwstralen
126*126 zichtstralen	46,84%	45,05%
254*254 zichtstralen	44,38%	46,28%

Tabel 8.1: Gewogen gemiddelde van de procentuele versnelling met de P-LIE structuur

Met telkens:

1. 4 schaduwstralen per lichtbron en
2. 9 schaduwstralen per lichtbron.

De testen van de UMBRA- en PENUMBRA-tijden werden enkel uitgevoerd voor 4 schaduwstralen per lichtbron en niet voor 9 schaduwstralen. De reden daartoe wordt gegeven in het volgende onderdeel.

Elk van deze metingen werd in vijfvoud gedaan en daarna uitgemiddeld, zodat de invloed van de garbage collector op de resultaten vermeden wordt. Meestal lag het verschil tussen de vijf metingen niet hoger dan tienden van seconden.

8.2.2 Resultaten van de test

De tijden die opgemeten werden voor de berekening van de belichting met behulp van de octree, worden gezien als referentie om de versnelling te bepalen voor het renderen met de P-LIE. Verwacht wordt dat de berekening van de radiantie in de snijpunten met de drie andere methoden sneller gaat.

Uit de metingen (zie tabellen in bijlage A.1, A.2 en A.3) blijkt dat de UMBRA- en de PENUMBRA-tijden groter zijn dan de P-LIE-tijden. Voor enkele scènes waren ze zelfs groter dan de OCTREE-tijden.

De oorzaak is vermoedelijk de triangularisatie van de umbra en de penumbra regio's. Door de triangularisatie kan het gebeuren dat er meer intersectietesten uitgevoerd worden in vergelijking met de octree.

Als er bijvoorbeeld drie blokkers zijn die gecontroleerd worden op intersectie met de octree, maar de umbra en penumbra regio bestaat uit een tiental driehoeken, zal dit meer tijd vragen wanneer de straal niet door de umbra regio en wel door de penumbra regio gaat.

Van de P-LIE-tijden voor de scènes werd een gewogen gemiddelde genomen van de procentuele versnelling met als gewichten de OCTREE-tijden

(Zie tabel 8.1). Op die manier wordt een versnelling voor een scène, waarvan het renderen met de octree langer duurt, meer in rekening genomen dan de versnelling voor een scène met een korte evaluatieduur. De scènes met een lange evaluatieduur hebben immers het meeste baat aan versnelling.

Het gewogen gemiddelde van de waarden uit tabel 8.1 geeft een waarde van 45,63%. We kunnen stellen dat het gemiddelde verschil tussen de OCTREE-tijden en de P-LIE-tijden 45% van de gemiddelde OCTREE-tijd bedraagt.

8.3 De invloed van het verfijnen van de P-LIE

Als tweede werd de invloed van het verfijnen van de P-LIE volgens umbra regio bij de constructie op de rendertijd getest.

8.3.1 Hoe getest werd

Als tweede test werden dezelfde metingen gedaan, met een andere configuratie voor de constructie van de P-LIE's:

- maximaal 200 driehoeken in een blad,
- verfijning aan de hand van de umbra regio,
- maximaal 2 opsplitsingen.

De metingen voor de OCTREE-tijd werden echter niet uitgevoerd. Enkele testen gaven kortere tijden dan de metingen uit de vorige test. De oorzaak is de betere resolutie van de octree. Het is echter niet mogelijk om een octree te construeren met dezelfde resolutie zonder daarvoor de gegevens uit de P-LIE's te gebruiken.

8.3.2 Resultaten van de test

De metingen van deze test werden opgenomen in bijlage in tabel A.1, A.2 en A.3. Als referentie wordt hier de OCTREE-tijden uit de vorige test gebruikt om de relatieve versnelling van de tijden te berekenen.

Van de P-LIE-tijden werd opnieuw een gewogen gemiddelde van de procentuele versnelling over de tien scènes met als gewichten de OCTREE-tijden bepaald. De resultaten hiervan werden opgenomen in 8.2.

	4 schaduwstralen	9 schaduwstralen
126*126 zichtstralen	52,28%	50,98%
254*254 zichtstralen	50,21%	53,03%

Tabel 8.2: Gewogen gemiddelde van de procentuele versnelling met de P-LIE structuur, verfijnd aan de hand van de umbra regio's

Het gemiddelde van de waarden uit tabel 8.2 bedraagt 51,62%. De P-LIE-tijden liggen dus gemiddeld 51% van de OCTREE-tijden lager. Dit is 6% meer dan de gemiddelde versnelling uit de vorige test.

8.3.3 Verfijnen van de P-LIE volgens aantal blokkers

P-LIE's die werden geconstrueerd met een maximum op het aantal blokkers tussen een zijde en een lichtbron werden niet uitvoerig getest. Enkele korte testen gaven aan dat een grens moeilijk valt te bepalen en verschilt naargelang de geometrie in de scène.

Bijvoorbeeld bij de Dragon scène met een grens van maximum 700 blokkers in een zijde werden bij de constructie meer dan 2000 cellen bijgemaakt. Met een grens van maximaal 900 blokkers werden slechts 200 cellen bijgemaakt.

Er kan ook aan adaptieve grens gebruikt worden, die lager wordt naarmate het blad dieper in de boom ligt. Hiervoor moet de startgrens en de verminderingsfactor bepaald worden.

De startgrens moet opnieuw voldoende hoog liggen om een overdreven aantal splitsingen te vermijden. Uit enkele testen bleek het aantal driehoeken in de scène als startgrens en als verminderingsfactor $\frac{1}{2 \times blad.niveau}$ een aanvaardbaar aantal splitsingen te geven (gelijklopend aan het aantal splitsen bij het verfijnen volgens umbra). Het renderen werd echter niet extra versneld.

Andere instellingen voor de startgrens en de verminderingsfactor konden voor bepaalde scènes de rendertijd verbeteren, voor andere scènes verbeterde de rendertijd niet, of werd zelfs niet verfijnd doordat de grens nooit bereikt werd.

8.4 Duur van de constructie

Naast de tijden voor de evaluatie van de radiantie werden ook de tijden voor de constructie van de P-LIE's voor deze testen gemeten.

Naam	**Tijd**	**Verschil**	**# Bladeren**	**# Driehoeken**	**% Langer**	**Tijd / # Driehoeken**
Cornell Cube 1	0.26		8	16		0.02
	0.56	0.30	17	16	186.67	0.04
Cornell Cube 2	0.21		8	32		0.01
	1.98	1.77	65	32	111.86	0.06
Cornell Cube 3	23.06		51	1932		0.01
	41.76	18.70	261	1932	223.32	0.02
Cornell Cube 4	0.82		8	34		0.02
	5.74	4.92	86	34	116.67	0.17
Cornell Cube 5	0.58		8	18		0.03
	6.06	5.48	126	18	110.58	0.34
Dragon	176.50		159	2766		0.06
	575.68	399.18	2557	2766	144.22	0.21
Huis A	75.64		157	4913		0.02
	97.39	21.75	222	4913	447.77	0.02
Huis B	102.24		196	5789		0.02
	170.21	67.97	436	5789	250.42	0.03
Huis C	286.25		195	5789		0.05
	325.81	39.56	245	5789	823.58	0.06
Boot	73.92		75	3974		0.02
	440.89	366.97	1032	3974	120.14	0.11

Tabel 8.3: Constructie gegevens van de P-LIE's. Per scène zijn twee rijen gegeven, respectievelijk voor de eerste test en de tweede test. Zie het vorige hoofdstuk voor een beschrijving van de bijhorende scènes.

Naam	Tijd (sec)	Tijd / Driehoeken
Cornell Cube 1	0.09	0.00563
Cornell Cube 2	0.03	0.00094
Cornell Cube 3	1.07	0.00055
Cornell Cube 4	0.05	0.00147
Cornell Cube 5	0.03	0.00167
Dragon	8.62	0.00312
Huis A	6.94	0.00141
Huis B	5.30	0.00092
Huis C	11.83	0.00204
Boot	2.73	0.00069

Tabel 8.4: De constructie tijden van de P-LIE's, zonder de umbra of de penumbra regio's te bepalen. Zie het vorige hoofdstuk voor een beschrijving van de bijhorende scènes.

In tabel 8.3 werden de tijden voor de constructie van de P-LIE's, gebruikt in de eerste en de tweede test, opgenomen samen met onder andere het aantal driehoeken in de respectievelijke scènes en het aantal bladeren in de bijhorende P-LIE's.

Gemiddeld genomen (gewogen met het aantal driehoeken van de scene) duurt de constructie met verfijning volgens umbra regio 2.35 keer zo lang als de constructie zonder de verfijning. De gemiddelde duur verdubbelt dus.

De constructie van de P-LIE duurt gemiddeld $0.03 \times aantal_{driehoeken}$ seconden wanneer geen verfijning gebruikt wordt en gemiddeld $0.1 \times aantal_{driehoeken}$ seconden wanneer deze wel gebruikt wordt.

8.5 Constructie zonder umbra of penumbra

Uit de vorige testen blijkt dat het renderen met de umbra en/of penumbra regio's minder snel gaat dan het renderen waarbij enkel de blokkerinformatie gebruikt wordt. Dit betekent dat de umbra en penumbra regio bij de constructie niet "moeten" berekend worden, zodat de constructietijd vermindert. Dit impliceert echter wel dat het verfijnen van de P-LIE volgens de umbra regio niet meer mogelijk zal zijn en de 6% winst die hiermee geboekt werd verloren gaat.

In tabel 8.4 staan de tijden voor de constructie van de P-LIE's zonder de bepaling van de umbra en penumbra regio.

Resultaat: de constructieduur wordt gereduceerd tot 5% van de constructieduur wanneer de umbra en penumbra regio's wel bepaald werden. De constructie duurt nu gemiddeld $0.002 \times aantal_{driehoeken}$ seconden.

Dit is beduidend minder. Voor de Dragon scène duurde het bijvoorbeeld ongeveer 3 minuten alvorens de P-LIE was geconstrueerd. Zonder de umbra en penumbra regio's duurde de constructie slechts 9 seconden.

In bijlage staan in tabel A.4 de tijden voor het renderen met dezelfde configuratie uit de eerste test, maar nu opgemeten voor de scènes met de P-LIE's zonder umbra en penumbra regio. De gewogen gemiddelde procentuele versnelling bedraagt hier 48%. Dit is 3% beter dan de versnelling uit de eerste test. Deze extra versnelling is te wijten aan het feit dat de garbage collector minder werk heeft en daardoor minder interfereert met de ray tracer.

8.6 Conclusie

In dit deel werden een aantal uitgevoerde testen en de resultaten ervan gepresenteerd. Er kan gesteld worden dat het renderen met P-LIE's 45% minder tijd vraagt dan het renderen met een octree. Wanneer de P-LIE's geconstrueerd werden met verfijning volgens umbra-regio is het 50% minder. Daartegenover staat de constructieduur van de P-LIE's, die behoorlijk lang is en die bijna verdubbelt als de verfijning gebruikt wordt.

Een uitgebreide conclusie op basis van de resultaten van de testen vormen samen met enkele bedenkingen en voorgestelde verbeteringen het onderwerp van het volgende hoofdstuk.

Hoofdstuk 9

Algemeen Besluit

Dit hoofdstuk vormt het besluit van deze verhandeling.

In het eerste onderdeel wordt een evaluatie gegeven van de P-LIE's die in deze verhandeling geïntroduceerd werden. Daarna wordt kort aangegeven welke verdere mogelijkheden en/of verbeteringen bestaan voor de P-LIE's.

9.1 Preprocessed LIE's

Het doel van de voorberekende lokale belichtingsomgevingen is het versnellen van het renderen van een synthetische scène met behulp van ray tracing. Van deze structuur wordt verwacht de evaluatie van de visibiliteitsfunctie in de rendering vergelijking voor de directe belichting te kunnen versnellen.

In dit eerste onderdeel van het besluit wordt op basis van de resultaten van de testen uit het vorige hoofdstuk een aantal conclusies gemaakt, samen met een aantal bedenkingen die hierbij moeten gemaakt worden.

9.1.1 De versnelling met P-LIE's

Uit de eerste test in het vorige hoofdstuk blijkt dat het renderen met P-LIE's gemiddeld 45% sneller gaat dan het renderen met een octree. De de duur van het renderen wordt bijna gehalveerd.

Het enige bezwaar op deze versnelling is de extra tijd die de constructie van een P-LIE vraagt. Dit bedraagt gemiddeld drie honderdste van het aantal driehoeken in seconden. Voor een scène met 10.000 driehoeken duurt de constructie *gemiddeld* vijf minuten. Bij de testen werden langere constructietijden gemeten voor kleinere scènes.

Wanneer het renderen met een octree van een scène met 10.000 driehoeken een 30-tal minuten in beslag neemt (wat zeer goed mogelijk is), zal dit met de P-LIE's een klein kwartier minder lang duren. De vijf minuten extra constructietijd maakt dat er minder, maar nog steeds, tijd wordt gewonnen.

Verder moet ook in rekening genomen worden dat de constructie van de P-LIE eenmalig uitgevoerd moet worden. Daarna kan deze meermaals, met verschillende cameraposities en verschillend detail, gebruikt worden voor een rendering.

Bij het gebruik van P-LIE's voor het renderen moet een afweging gemaakt worden tussen de constructietijd en de tijd die bij het renderen gewonnen wordt. Bij deze afweging moet rekening gehouden worden met het aantal afbeeldingen die gegenereerd zullen worden op basis van de P-LIE's en het detail (aantal schaduwstralen per lichtbron) dat daarbij vereist is.

9.1.2 Umbra en penumbra

Op basis van de resultaten uit het vorige hoofdstuk kan gesteld worden dat het bijhouden van de umbra en de penumbra regio in de gegevensobjecten niet oplevert bij het renderen. In de meeste gevallen werd een versnelling gemeten wanneer beide regio's gebruikt werden, maar die was echter steeds minder dan de versnelling die met het gebruik van enkel de voorberekende blokkers geboekt werd. In sommige gevallen was er zelfs sprake van een vertraging in vergelijking met de octree.

Van de umbra en de penumbra regio's op de zijden, die niet convexe, zelfsnijdende polygonen kunnen zijn, wordt een triangularisatie opgesteld. Hierdoor kan het gebeuren dat een groot aantal extra driehoeken wordt gecreëerd, bovenop de driehoeken die al aanwezig zijn in de scène. In het slechtste geval moeten al deze driehoeken gecontroleerd worden op intersectie met een schaduwstraal. Als het aantal umbra en penumbra driehoeken plus het aantal blokkers hoger ligt dan het aantal driehoeken die gecontroleerd worden op intersectie wanneer een octree gebruikt wordt, zal de evaluatie van de visibiliteitsfunctie in vergelijking langer duren.

Dit kan misschien opgelost worden door geen triangularisatie op te stellen van de umbra en penumbra regio's, waardoor een schaduwstraal enkel op intersectie met maximaal twee extra polygonen gecontroleerd moet worden. Vermoedelijk zal dit echter toch geen oplossing bieden.

Om een straal te testen op intersectie met een polygon moet eerst het intersectiepunt met het vlak waarin de polygon ligt bepaald worden. Dit kan op vrij eenvoudige wijze gebeuren doordat de zijden parallel liggen aan twee

assen. Maar dan moet om na te gaan of het punt in de polygon ligt, voor een rechte door het intersectiepunt het aantal intersecties met de ribben van de polygon bepaald worden. Aangezien de polygon meer ribben zal hebben dan driehoeken in zijn triangularisatie geeft dit opnieuw meer intersectie testen.

De mogelijkheid bestaat dat het modelleren van de umbra en penumbra regio's als polygonen voor de intersectietest niet oplevert, maar dit zou getest moeten worden. Onder voorbehoud kan gesteld worden dat de umbra en penumbra regio's beter niet gebruikt worden bij het evalueren van de visibiliteitsfunctie.

9.1.3 Verfijnen volgens umbra

Bij de constructie van de P-LIE's blijken de umbra regio's wel van nut. Uit de resultaten in het vorige hoofdstuk blijkt dat als een P-LIE structuur wordt geconstrueerd en verfijnd volgens de umbra regio's, de versnelling voor het renderen met P-LIE's gemiddeld 6% hoger ligt. In dit geval is effectief sprake van een gehalveerde rendertijd.

Wanneer de P-LIE wordt verfijnd volgens de umbra regio, duurt het gemiddeld tot tweemaal zo lang om de P-LIE's te construeren. Langs de andere kant moet de constructie van de P-LIE's voor een scène slechts eenmalig uitgevoerd worden, waarna de P-LIE's meermaals kunnen gebruikt worden. De totale tijdwinst bij het renderen kan de extra constructietijd compenseren.

Wanneer gekozen wordt om de P-LIE te verfijnen om die 6% extra versnelling te bekomen, moet dus de extra tijd die nodig is voor de constructie liefst lager liggen dan de tijd die de extra versnelling wint. Indien bijvoorbeeld een snelle eenmalige visualisatie moet gemaakt worden van een scène bij wijze van test kan het voordeliger blijken de verfijning niet te gebruiken. Als een rendering vanuit verschillende camerastandpunten met hoog detail moet worden uitgevoerd, kan de verfijning voordeel bieden.

9.1.4 Geen umbra regio's?

Voor het renderen bleken de umbra en penumbra regio's geen voordeel te bieden. Enkel bij de verfijning van de P-LIE zijn de umbra regio's nog van nut.

Uit de resultaten is verder gebleken dat met verfijning volgens de umbra regio's de constructie tot twee maal zo lang duurt. De oorzaak is de extra hoeveelheid cellen die verwerkt moeten worden.

Wanneer geen verfijning gebruikt wordt, is het bijgevolg voordeliger de umbra regio's *niet* te bepalen. De resultaten uit het vorige hoofdstuk gaven aan dat de constructieduur dan lager ligt (een twintigste van de duur van de constructie wanneer de umbra regio's wel worden bepaald). Doordat er minder objecten in het geheugen geladen worden en de garbage collector bijgevolg minder werk heeft stijgt de versnelling van het renderen met een P-LIE (evenwel zonder verfijning) met 3%. Dus wordt de winst die geboekt werd met de verfijning gehalveerd.

Het is opnieuw afhankelijk van de rendering die uitgevoerd moet worden of al dan niet de P-LIE-structuur beter verfijnd wordt volgens de umbra regio. Als verfijning gebruikt wordt en umbra regio's bepaald moeten worden, duurt de constructie gemiddeld 33 maal zo lang in vergelijking met de constructie zonder verfijning, en bepaling van de umbra regio's. Dit maakt de verfijning minder aantrekkelijk.

9.1.5 P-LIE's versus LIE's

Een vergelijking van P-LIE's met de LIE's uit [FBG02] is moeilijk te maken. Om een correcte vergelijking te kunnen maken moeten beide structuren in een zelfde omgeving getest worden.

Voor LIE's werden tijden gemeten die tot 10 maal kleiner waren dan tijden voor een standaard structuur. De structuur waarmee vergeleken werd verschilt echter van de in deze verhandeling gebruikte octree. Daarnaast werd het renderen van de scènes met LIE's uitgevoerd op 24 processoren in parallel. Verder speelt ook de gebruikte taal waarin de ray tracers worden geschreven een rol, bijvoorbeeld een C++ implementatie wordt niet gestoord door een garbage collector.

9.1.6 Samenvattend besluit

Op basis van bovenstaande delen kunnen de volgende conclusies gemaakt worden:

- Het gebruik van een P-LIE bij het renderen halveert de rendertijd in vergelijking met het renderen op basis van een octree.
- De umbra en penumbra regio's bieden geen voordeel bij het renderen.
- De verfijning volgens umbra regio verhoogt de versnelling, maar ook de constructieduur.

De laatste conclusie houdt in dat bij de constructie van de P-LIE een afweging gemaakt moet worden tussen de gewenste render- en constructietijd, afhankelijk van de scène, het aantal beelden die met de P-LIE gemaakt moeten worden en met welk detail dit gebeurt.

9.2 Mogelijkheden voor de toekomst

In deze verhandeling werden nog niet alle mogelijkheden besproken van (al dan niet voorberekende) lokale belichtingsomgevingen. In dit deel worden kort enkele denkpistes gegeven voor eventueel verder onderzoek.

9.2.1 Dynamische scènes

De P-LIE structuur die in deze verhandeling werd voorgesteld is van toepassing op statische scènes. Lichtbronnen blijven op een vaste positie en ook de andere objecten verplaatsen niet. In [FBPG00] worden LIE's beschreven die gebruikt worden in dynamische scènes.

Op zelfde wijze kunnen P-LIE's gebruikt worden voor dynamische scènes. Bij verplaatsing van een lichtbron of een object moeten alle beïnvloedde cellen worden bijgewerkt, door met bijvoorbeeld clipping de blokkers opnieuw te bepalen. Eventueel kan hierbij ruimtelijke coherentie gebruikt worden.

9.2.2 Umbra en penumbra polygonen

Zoals eerder al vermeld kan het interessant zijn om de triangularisatie van de umbra en penumbra regio's achterwege te laten en met polygonen te werken, zodat er eventueel toch een versnelling optreed.

Daarnaast kan ook worden nagegaan of de bepaling van deze regio's niet efficiënter kan gebeuren. Bij de implementatie werd meermaals de algemene polygonclipper uit [Vat92] gebruikt. Die werd echter niet zelf geschreven. Een C++ versie werd gebruikt waarvoor een C# wrapper geschreven werd. Hierdoor moeten enkele conversies van objecten (Driehoeken in C# naar polygonen in C++) meermaals uitgevoerd worden. Dit was niet nodig wanneer een rechtstreekse C# implementatie zou zijn gebruikt, of de implementatie van de constructie in C++ zou zijn geschreven. Dit zou de constructieduur ten goede komen, zodat het verfijnen volgens umbra regio terug aantrekkelijker wordt.

9.2.3 Andere methoden

Blokkers werden in deze verhandeling geïdentificeerd door middel van een lichte variant op shaft culling. In [FBG02] gebeurde dit aan de hand van intersectietesten. Eventueel zouden ook andere mogelijkheden kunnen worden overwogen.

Verder bleek ook dat, wanneer de structuur verfijnd wordt, een grotere versnelling wordt geboekt. Andere splitscriteria dan de umbra regio of het aantal blokkers voor een zijde kunnen worden getest. Voor de verfijning volgens het aantal blokkers kan nog nagegaan worden of een richtlijn voor de grens kan opgesteld worden.

9.2.4 Veel lichtbronnen

Naast een versnelling van de evaluatie van de directe belichting, zijn lokale belichtingsomgevingen zoals ze worden voorgesteld in [FBG02] ook een structuur om de berekening van de belichting in scènes waarin een groot aantal lichtbronnen aanwezig zijn te optimaliseren. Zo zal de bijdrage van lichtbronnen op een grote afstand van een cel genegeerd worden wanneer er geen zichtbare invloed is op het resulterende beeld van het renderen.

P-LIE's zouden voor hetzelfde doel gebruikt kunnen worden.

9.2.5 Globale belichting

De P-LIE structuur wordt gebruikt bij het versnellen van de evaluatie van de visibiliteitsfunctie in de renderingvergelijking voor de directe belichting. Mogelijkheden om deze ook te gebruiken bij globale belichting kunnen worden onderzocht.

Bijvoorbeeld bij een path tracer of een radiositeitsalgoritme (zie [DBB03]) waarbij een aparte evaluatie van de directe belichting wordt gebruikt, kan een P-LIE van pas komen.

Daarnaast kan ook worden nagegaan wat de mogelijkheden zijn indien niet alleen zichtbare lichtbronnen met bijhorende blokkers worden bijgehouden in de cellen, maar ook de omliggende objecten die zichtbaar zijn in de cel.

Voor een cel zouden alle zichtbare objecten en de lichtbronnen geïdentificeerd kunnen worden door middel van clipping of stralen. De blokkers voor de lichtbronnen zullen eveneens tot de lijst van zichtbare objecten behoren. Een shaft culling (zie [HW94]) algoritme zou snel de blokkers uit die lijst kunnen identificeren.

9.2.6 De BRDF en de BTDF

Voor deze verhandeling lag de focus voor P-LIE's op de versnelling die voor het renderen werd behaald. Daarom werd geen aandacht geschonken aan andere materialen voor de objecten in de scène dan diffuse. Andere materialen zouden gebruikt kunnen worden, zonder veel te moeten veranderen aan de P-LIE's.

Indien refractie wordt gemodelleerd aan de hand van de bidirectionele transmissie distributie functie (BTDF), zullen er caustics verschijnen in de scène. Er kan nagegaan worden of ook caustic informatie zou kunnen opgeslagen worden in de P-LIE's. Bijvoorbeeld kunnen de cellen waarin een caustic valt samen met het object en de lichtbron die de caustic veroorzaken geïdentificeerd worden, om zo optimaal de belichting van de caustic te kunnen berekenen.

9.3 Ten laatste...

Deze eindverhandeling heeft als primair onderwerp de evaluatie van de directe belichting in ray tracing. Dit is een onderwerp dat nog steeds actief onderzocht wordt. Eén van de meest recente ontwikkelingen in dit gebied zijn de lokale belichtingsomgevingen.

Deze techniek werd in deze verhandeling nader bekeken. De mogelijkheid om de lokale belichtingsomgevingen te berekenen vóór aan het renderen begonnen wordt, werd uitgewerkt. Doordat de constructie onafhankelijk wordt van de positie van de camera, kan de P-LIE-structuur meermaals gebruikt worden zonder een her- of extra berekening. Verder werd ook nagegaan of het bepalen van umbra en penumbra regio's voordelen boden bij het renderen.

Van de P-LIE structuur wordt verwacht dat deze de evaluatie van de directe belichting versneld. Met een halvering van de rendertijd kan gesteld worden dat dit inderdaad het geval is. De umbra en de penumbra regio's echter bleken hierin geen aandeel te hebben.

Daarnaast moet ook de constructie van een P-LIE in een zo kort mogelijke tijd gebeuren. Enkel indien umbra en penumbra regio's niet worden berekend, kan worden gesteld dat ook deze tijd voldoende laag ligt.

De doelstelling voor deze verhandeling lijkt dus bereikt te zijn. Evenwel zijn er nog vele andere mogelijkheden die een eventueel meer efficiënte structuur kunnen vormen.

Bijlage A

Tijden voor de testscènes.

		126	126 + u	254	254 + u
Huis B	OCTREE	12.23	12.23	45.86	45.86
	LIE	5.91	5.61	22.84	21.6
	UMBRA	14.72	17.29	57.23	58.13
	PENUMBRA	19.78	23.09	76.97	80.33
Huis A	OCTREE	10.89	10.89	39.67	39.67
	LIE	5.91	6.13	21.4	23.3
	UMBRA	10.44	10.89	40.59	41.9
	PENUMBRA	13.97	14.52	54.29	52.1
Dragon	OCTREE	4.09	4.09	14.19	14.19
	LIE	3.29	1.66	12.93	6.07
	UMBRA	3.9	2.1	15.31	8.03
	PENUMBRA	3.04	2.12	11.92	8.03
Cornell Cube 1	OCTREE	1.37	1.37	4.79	4.79
	LIE	0.95	0.94	3.56	3.33
	UMBRA	0.99	0.96	3.69	3.51
	PENUMBRA	1.01	0.98	3.75	3.53
Cornell Cube 2	OCTREE	2.27	2.27	7.42	7.42
	LIE	1.87	1.58	6.26	5.76
	UMBRA	1.94	1.6	6.5	5.92
	PENUMBRA	1.98	1.64	6.72	6.01

Tabel A.1: Tijden voor de evaluatie van de directe belichting in de eerste 5 van de 10 testscènes, door middel van a) de octree (OCTREE), b) de voorberekende blokkers (LIE), c) de voorberekende blokkers en umbra regio (UMBRA) en d) de voorberekende blokkers en umbra en penumbra regio (PENUMBRA). Dit werd gedaan voor afbeeldingen van 126 op 126 en 254 op 254 pixels, elk eenmaal zonder verfijning van de P-LIE's, eenmaal met. Bij de berekening werden 4 schaduwstralen per lichtbron gebruikt.

		126	126 + u	254	254 + u
Cornell Cube 3	OCTREE	5.32	5.32	16.43	16.43
	LIE	2.93	2.86	9.07	10.43
	UMBRA	7.92	7.44	26.5	27.22
	PENUMBRA	8.75	8.26	29.36	29.26
Cornell Cube 4	OCTREE	3.97	3.97	15.77	15.77
	LIE	2.56	2.52	10.35	8.26
	UMBRA	2.96	2.74	11.43	9.13
	PENUMBRA	3.02	2.89	11.7	9.45
Huis C	OCTREE	25.75	25.75	93.31	93.31
	LIE	12.4	12.13	48.3	46.04
	UMBRA	39	37.32	148.87	144.61
	PENUMBRA	50.68	49.72	197.47	193.24
Cornell Cube 5	OCTREE	3.08	3.08	11.15	11.15
	LIE	1.74	1	6.73	3.71
	UMBRA	2.23	1.21	8.73	4.52
	PENUMBRA	2.49	1.27	9.41	4.69
Boot	OCTREE	5.81	5.81	18.46	18.46
	LIE	2.19	1.25	7.09	4.44
	UMBRA	4.84	2.97	17.12	10.93
	PENUMBRA	4.75	2.89	16.64	10.63

Tabel A.2: Tijden voor de evaluatie van de directe belichting in de andere 5 van de 10 testscènes, door middel a) van de octree (OCTREE), b) de voorberekende blokkers (LIE), c) de voorberekende blokkers en umbra regio (UMBRA) en d) de voorberekende blokkers en umbra en penumbra regio (PENUMBRA). Dit werd gedaan voor afbeeldingen van 126 op 126 en 254 op 254 pixels, elk eenmaal zonder verfijning van de P-LIE's, eenmaal met. Bij de berekening werden 4 schaduwstralen per lichtbron gebruikt.

		126	126 + u	254	254 + u
Huis B	OCTREE	28.82	28.82	104.02	104.02
	LIE	14.07	13.94	52.98	51.69
Huis A	OCTREE	23.78	23.78	87.88	87.88
	LIE	13.34	13.13	47.57	44.83
Dragon	OCTREE	8.47	8.47	31	31
	LIE	7.14	3.45	27.61	13.04
Cornell Cube 1	OCTREE	2.74	2.74	10.58	10.58
	LIE	2.02	1.98	7.89	7.43
Cornell Cube 2	OCTREE	4.53	4.53	16.14	16.14
	LIE	3.73	3.46	13.53	12.41
Cornell Cube 3	OCTREE	11.49	11.49	39	39
	LIE	6.05	5.58	21.41	22.42
Cornell Cube 4	OCTREE	8.96	8.96	30.1	30.1
	LIE	5.62	5.38	19.04	18.24
Huis C	OCTREE	58.74	58.74	226.42	226.42
	LIE	28.4	27.18	107.9	101.72
Cornell Cube 5	OCTREE	6.17	6.17	25.86	25.86
	LIE	3.38	2.4	14.71	7.6
Boot	OCTREE	13.15	13.15	42.5	42.5
	LIE	4.95	2.33	16.92	8.73

Tabel A.3: Tijden voor de evaluatie van de directe belichting in de 10 testscènes, door middel van de octree (OCTREE) en de voorberekende blokkers (LIE). Dit werd gedaan voor afbeeldingen van 126 op 126 en 254 op 254 pixels, met telkens 9 schaduwstralen per lichtbron.

		128	256
Huis B	OCTREE	12.23	45.86
	LIE	5.99	21.32
Huis A	OCTREE	10.89	39.67
	LIE	5.3	20.7
Dragon	OCTREE	4.09	14.19
	LIE	3.29	12.93
Cornell Cube 1	OCTREE	1.37	4.79
	LIE	1.03	3.75
Cornell Cube 2	OCTREE	2.27	7.42
	LIE	1.77	6.72
Cornell Cube 3	OCTREE	5.32	16.43
	LIE	2.5	9.66
Cornell Cube 4	OCTREE	3.97	15.77
	LIE	2.35	10.1
Huis C	OCTREE	25.75	93.31
	LIE	12.08	41.7
Cornell Cube 5	OCTREE	3.08	11.15
	LIE	1.65	6.2
Boot	OCTREE	5.81	18.46
	LIE	2.16	7

Tabel A.4: Tijden voor de evaluatie van de directe belichting in de 10 testscènes, door middel van de octree (OCTREE) en de voorberekende blokkers (LIE) *zonder dat de umbra en penumbra regio werden voorberekend.* Dit werd gedaan voor afbeeldingen van 126 op 126 en 254 op 254 pixels, met telkens 4 schaduwstralen per lichtbron.

Bibliografie

[Arv95] James Arvo. Applications of irradiance tensors to the simulation of non-lambertian phenomena. In *SIGGRAPH*, pages 335–342, 1995.

[DBB03] Philip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. A. K. Peters, Natick, USA, 2003.

[DDP97] Frédo Durand, George Drettakis, and Claude Puech. The visibility skeleton: A powerful and multi-purpose global visibility tool. In *ACM SIGGRAPH 97*, Aug 1997. http://w3imagis.imag.fr/Membres/Fredo.Durand/PUBLI/siggraph97/index.htm.

[DDP02] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex. *ACM Trans. Graph.*, 21(2):176–206, 2002.

[DF94] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 223–230, New York, NY, USA, 1994. ACM Press.

[Dut03] Philip Dutré. The global illumination compendium, August 2003. `http://www.cs.kuleuven.ac.be/~phil/GI/`.

[FBG02] Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Local illumination environments for direct lighting acceleration. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 7–14, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[FBPG00] Sebastian Fernandez, Kavita Bala, Moreno A. Piccolotto, and Donald P. Greenberg. Interactive direct lighting in dynamic scenes. Technical report PCG-00-2, Program of Computer Graphics, Cornell University, Jan 2000.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[HDG99] David Hart, Philip Dutré;, and Donald P. Greenberg. Direct illumination with lazy visibility evaluation. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 147–154, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*. Eurographics, Eurographics, 2003. State-of-the-Art Report.

[HW94] Eric A. Haines and John R. Wallace. Shaft culling for efficient ray-traced radiosity. In *Second Eurographics Workshop on Rendering (Photorealistic Rendering in Computer Graphics)*, 1994.

[Jar73] R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inform. Proc. Lett.*, 2:18–21, 1973.

[KW86] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo methods. Vol. 1: basics*. Wiley-Interscience, New York, NY, USA, 1986.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[PSS98] S. Parker, P. Shirley, and B. Smits. Single sample soft shadows. Technical Report UUCS-98-019, University of Utah, October 1998.

[SG94] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 231–238, New York, NY, USA, 1994. ACM Press.

[SM03] Peter Shirley and R. Keith Morley. *Realistic Ray Tracing*. A. K. Peters, Ltd., Natick, MA, USA, 2003.

[SR00] Michael M. Stark and Richard F. Reisenfield. Exact illumination in polygonal environments using vertex tracing. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 149–160, London, UK, 2000. Springer-Verlag.

[Vat92] Bala R. Vatti. A generic solution to polygon clipping. *Commun. ACM*, 35(7):56–63, 1992.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[Woo93] Andrew Woo. Efficient shadow computations in ray tracing. *IEEE Comput. Graph. Appl.*, 13(5):78–83, 1993.

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.

Lijst van figuren

Lijst van tabellen