Masterthesis

Mooie resultaten met weinig moeite: Dieptemappen met stroke-based invoer

Academiejaar 2007-2009

Patrik Goorts

Philippe Bekaert
Mark Gerrits
Bert De Decker

Abstract

In deze thesis wordt het omzetten van 2D afbeeldingen naar 3D afbeeldingen onderzocht, meer bepaald het omzetten naar een dieptemap. Dit is een grijswaardeafbeelding, waarbij elke pixel een diepte van de overeenkomstige pixel voorstelt. De dieptemap wordt berekend met behulp van extra invoer van de gebruiker. Deze kan strokes tekenen op de afbeelding die de eigenschappen op deze plaats aangeeft, bijvoorbeeld absolute diepte of oriëntatie. De oplossing wordt bekomen door voor een gegenereerde oplossing te berekenen hoe fout deze is, gegeven de strokes van de gebruiker. Door deze fout iteratief te verminderen, kan een oplossing berekend worden.

In het eerste deel wordt een literatuurstudie gemaakt over het maken van dieptemappen door het annoteren van afbeeldingen. Er wordt gekeken naar technieken voor het minimaliseren van functies, met aandacht voor het minimaliseren via graph cuts. Enkele graph cut algoritmen worden besproken. Als tweede worden toepassingen behandeld die gebruik maken van annotaties op afbeeldingen, zoals het inkleuren, herbelichten en corrigeren van afbeeldingen en het segmenteren in lagen. Deze toepassingen maken vaak gebruik van de besproken minimalisatietechnieken. Als derde worden algoritmen besproken voor het maken van dieptemappen en modellen. Deze bevatten stereo afbeeldingen, afbeeldingen met speciale belichting, ... Tenslotte wordt gekeken naar het creëren van dieptemappen met invoer van de gebruiker. Deze technieken zijn momenteel nog jong en er zijn veel uitbreidingen mogelijk.

In het tweede deel wordt een nieuwe methode voorgesteld. In deze methode kan de gebruiker strokes tekenen op een afbeelding. Deze strokes stellen absolute diepte voor of een oriëntatie. Deze strokes worden vervolgens gepropageerd naar de rest van de pixels in hetzelfde segment in de afbeelding. De gebruiker kan verschillende dieptes tekenen in één segment, zodat er ook gradiënten getekend kunnen worden in plaats van alleen lijnen van één kleur. Dit soort invoer is nog niet eerder gebruikt.

Voor een bepaalde dieptemap kunnen we nu berekenen hoe fout deze is. Deze fout noemt men de energie en moet zo minimaal mogelijk zijn. Het berekenen van deze energie gebeurt met behulp van de afbeelding, de ingegeven strokes en extra berekende informatie. Deze

informatie bestaat uit de randen in de afbeelding en de afstand tot de dichtsbijzijnde stroke. De randen zijn nodig om de afbeelding in te delen in segmenten. Er worden geen absolute randen gebruikt, maar een kans op een rand, zodat ook vage randen worden herkend. De afstanden zijn nodig als er verschillende dieptes zijn getekend in één segment.

Gegeven deze informatie, kan er een nieuwe dieptemap berekend worden. Dit gebeurt door iteratief pixels van diepte te wisselen totdat er geen pixels meer gewisseld kunnen worden. Om te bepalen welke pixels mogen wisselen, wordt de afbeelding voorgesteld als een gelabelde graaf met twee extra knopen die twee verschillende dieptes voorstellen. De labels op de bogen staan in functie van de energiefunctie. Door het berekenen van een min-cut van de graaf zullen de pixels in twee groepen worden gedeeld, waarbij elke knoop bij één van de twee dieptes hoort. Hieruit kan bepaald worden welke pixels van dieptewaarde kunnen wisselen. Door dit repetitief voor elk paar dieptes te doen, zal er een optimale oplossing bekomen worden.

Voorwoord

Niemand houdt van saai en repititief werk. Daarom is er veel werk gestoken in het automatiseren van veel taken, maar er is nog een lange weg te gaan. Het lijkt mij zeer interessant hier mijn bijdrage aan te leveren. Vooral de techniek door *gewoon wat lijntjes te tekenen* op een afbeelding voor goede resultaten is aantrekkelijk. Het is een uitdaging om simpele invoertechnieken te gebruiken en toch een bruikbaar eindresultaat te hebben.

Deze thesis is niet zonder hulp gerealiseerd. Mijn dank gaat uit naar mijn promotor Prof. Dr. Philippe Bekaert en mijn begeleider Mark Gerrits en Bert De Decker voor alle hulp en aanwijzingen die mij vooruit hielpen.

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

Het creëren van een 3D-representatie van een scene is een vrij complex probleem. Veel oplossingen zijn al voorgesteld, maar de meeste vereisen gecontroleerde omstandigheden. Hierbij denken we aan meerdere camera's, speciale belichting of geavanceerde tastapparatuur (Levoy et al. 2000). Een aantal populaire technieken zijn stereovisie (gebruik makend van twee of meer afbeeldingen die vanop verschillende posities zijn vastgelegd, Scharstein & Szeliski (2002) en Seitz et al. (2006)) en *structured light* (gebruik makend van speciale lichtpatronen op de scene geprojecteerd (Scharstein et al. 2003)). De in deze thesis voorgestelde techniek werkt met slechts één afbeelding, zonder speciale eisen qua omstandigheden. Het gebruik van een enkele 2D-afbeelding is een nog weinig onderzocht gebied.

Diepteinformatie berekenen uit afbeeldingen is nochtans een nuttig probleem. Met deze informatie is het mogelijk objecten te verwijderen, toe te voegen of te wijzigen. Denk bijvoorbeeld aan het naar voren halen van een object. Dan is het belangrijk te weten welke objecten dan verborgen worden door de nieuwe positie en welke niet. Dit is puur afhankelijk van de diepte. Een andere toepassing is het gebruik van animatie of het verplaatsen van het standpunt. Als de camera verplaatst zou worden, zullen de objecten die vooraan staan sterk moeten verschuiven en objecten achteraan minder. Dit is enkel mogelijk als de diepte van elk object bekend is.

Een 3D-representatie uit een enkele, willekeurige 2D-afbeelding genereren is niet triviaal. Dit is omdat het probleem *underconstrained* is: de pixels van een afbeelding op zich geven niet genoeg informatie om een unieke en ideale oplossing te bekomen. Extra informatie is dus vereist. Een volledig geautomatiseerde methode die slechts één afbeelding gebruikt moet deze informatie via bijvoorbeeld computervisie bekomen: er moet worden gezocht naar bepaalde *features* in de afbeelding die informatie kunnen geven over de diepte. We denken bijvoorbeeld aan belichting, schaduw, weerspiegeling, ... Er bestaan nog andere technieken die worden gebruikt voor specifieke scenes, omdat deze features bekend zijn. Een voorbeeld hiervan is het modelleren van gezichten (Blanz et al. 1999) of een automatische photo pop-up (Hoiem

et al. 2005).

In de voorgestelde oplossing bekijken we het probleem van een andere kant. Met behulp van een nieuwe methode gaan we een 3D-representatie maken uit één willekeurige afbeelding. De 3D-informatie wordt voorgesteld door een zogenaamde dieptemap. Dit is een discrete grijswaardeafbeelding, waarbij elke pixel de diepte voorstelt. Wit is ver weg, zwart is dichtbij. Tussenliggende waarden zijn tussenliggende diepten. Het eindresultaat zijn dus twee afbeeldingen: de originele afbeelding en de dieptemap.

Om deze dieptemap te bepalen, kan de gebruiker *strokes* maken op de originele afbeelding. Deze strokes bepalen wat de diepte of de oriëntatie is op deze plaats. De diepte komt rechtstreeks overeen met de diepte die gebruikt wordt in de dieptemap. De oriëntatie geeft aan hoe de vlakken in de afbeeldingen hellen. Hieruit kan eveneens diepte bepaald worden. Deze strokes mogen niet te gedetailleerd of te precies worden, anders is het gebruik niet meer praktisch. De informatie van de strokes wordt vervolgens verspreid over de rest van de afbeelding. Andere toepassingen van deze invoertechniek zijn bijvoorbeeld herinkleuren van afbeeldingen (Levin et al. 2004) of regiosegmentatie (Boykov & Jolly 2001).

De dieptemap kan gemakkelijk bepaald worden voor de pixels waar de gebruiker strokes heeft geplaatst. Om te weten hoe de andere pixels ingevuld moeten worden kan er voor elke mogelijke dieptemap berekend worden hoe fout deze is. Deze fout noemen we de energie. Het is de bedoeling deze energie te minimaliseren. De energie wordt iteratief berekend uit de strokes gemaakt door de gebruiker, de huidige dieptemap en de originele afbeelding. Elke iteratie heeft als uitvoer een dieptemap die elke keer iets beter overeenstemt met de strokes ingegeven door de gebruiker. De berekeningen gebeuren op een discrete manier, dus er wordt slechts een eindige precisie bereikt.

De energiefunctie is zo opgesteld dat gebieden met ongeveer dezelfde kleur ook dezelfde diepte krijgen. Aan de randen van een segment in de afbeelding wordt er gestopt. De segmenten worden berekend met behulp van een edge detection algoritme. Deze edge detection moet niet noodzakelijk bepalen of de ruimte tussen 2 pixels een rand is of niet. Omdat er gebruik wordt gemaakt van energieminimalisatie kan er ook een waarde tussen 0 en 1 worden berekend, waarbij 0 zeker geen rand is en 1 zeker wel. Een waarde van $0,5$ wordt dan als een rand beschouwd als de gebruiker dit als rand beschouwt, met andere woorden als er strokes zijn geplaatst. Als er geen strokes zijn geplaatst is dit beschouwd als geen rand. Dit maakt randdetectie minder afhankelijk van een correcte detectie.

Als tweede houden we rekening met afstanden tot ingegeven informatie om eventuele fouten in de strokes te camoufleren en om meerdere dieptes in een segment toe te laten. Zo kun-

nen ook overgangen in diepte getekend worden. De gebruiker kan een lijn met een gradiënt tekenen. Deze gradient geeft dan ook een helling aan. Als er geen gebruik wordt gemaakt van de afstand, wordt dit beschouwd als een fout in de strokes, omdat een enkel segment in de afbeelding meerdere dieptes moet hebben. Deze manier van strokes tekenen is nog niet toegepast in andere toepassingen.

Het minimaliseren van de energie gebeurt met behulp van grafen. Het principe is dat er een graaf wordt opgesteld van een deel van de afbeelding. Elke pixel stelt een knoop voor (de pixelpknopen) en tussen elke pixelknoop wordt een boog getrokken als deze pixels naast elkaar liggen. Aan deze graaf worden twee knopen toegevoegd die elk een diepte voorstellen en deze twee knopen worden met elke pixelknoop verbonden. Vervolgens krijgt elke boog een label die in functie staat van de energiefunctie. Deze graaf wordt vervolgens in twee stukken geknipt, zodat de som van de labels van de doorgeknipte bogen zo klein mogelijk is. Deze bewerking noemt men een *graph-cut* en bepaalt welke knoop (en dus welke overeenkomende pixel) welke diepte krijgt. Als de diepte voor elke pixel wordt geplaatst in een afbeelding, verkrijgt men een dieptemap. Door iteratief elk paar mogelijke dieptes te beschouwen, zal de hele afbeelding een juiste diepte krijgen. Wanneer er geen pixels meer van diepte veranderen, is een minimale energie en dus een optimale oplossing bereikt. Deze techniek is eerder gebruikt voor het opslitsen van afbeeldingen in segmenten en is beschreven door Boykov et al. (2001).

De *graph-cut* zelf gebruikt een techniek die speciaal is afgestemd op afbeeldingen. Er wordt gebruik gemaakt van een graaf die er uitziet als een raster. Omdat elke knoop een beperkt aantal buren heeft, is hier speciaal rekening mee gehouden. Bovendien wordt er geen rechtstreekse graph-cut berekend. In plaats daarvan wordt er gezocht naar een zo groot mogelijke *stroom* die door de graaf kan gestuurd worden. Dit kan worden geïnterpreteerd als de hoeveelheid water dat er door een netwerk van buizen kan stromen als de kost van de bogen de breedte van de buizen voorstelt. Deze stroom noemt men de max-flow. Hieruit kan de min cut worden afgeleid.

Deze methode is uitermate geschikt als er geen veronderstellingen over de afbeelding of de scene gemaakt kunnen worden. Er is geen controle over de belichting, het aantal en de plaats van camera's, ... Dit is het geval voor afbeeldingen die niet zijn gemaakt om te worden omgezet naar 3D. Deze techniek is daarentegen minder geschikt voor scenes met veel kleine vlakken of grillige vormen (bijvoorbeeld een bos). Deze vereisen zeer veel invoer van de gebruiker. Ook afbeeldingen met vage randen (bijvoorbeeld wolken of glas) zijn moeilijk te verwerken, omdat de segmenten niet goed kunnen worden bepaald.

Deze thesis bestaat uit twee delen. In het eerste deel wordt een literatuurstudie gemaakt

over technieken die nodig zijn voor de berekening van de dieptemappen, andere toepassing van het annoteren van afbeeldingen voor diverse doeleinden (eveneens gebruik makend van energieminimalisatie en graph cuts) en andere methoden voor het maken van dieptemappen, zoals stereo-afbeeldingen, speciale lichtpatronen, aftasten van objecten, ... Als laatste wordt er een bespreking gegeven van eerdere methoden om dieptemappen te maken met behulp van user input. In het tweede deel wordt de nieuwe methode voorgesteld, samen met de resultaten. Ook worden de gebruikte algoritmen in detail besproken.

Hoofdstuk 2

Literatuurstudie

In dit hoofdstuk zal er een literatuurstudie gegeven worden over de verschillende technieken die nauw samenhangen met het omzetten van 2D naar 3D. In sectie 2.1 geven we een overzicht van een aantal algemene technieken die gebruikt worden in verderliggende secties, meer bepaald energieminimalisatie en graph cuts. In sectie 2.2 wordt een overzicht gegeven van eerdere toepassingen met stroke-based input. Deze invoermethode komt in een aantal andere toepassingen terug. Sectie 2.3 beschrijft andere methoden om dieptemappen of 3D-modellen te maken. Deze methoden zijn meestal niet gericht op één afbeelding als invoer. Er wordt bijvoorbeeld gebruik gemaakt van stereovisie, speciale belichting, aftasten van objecten, ... In de laatste sectie, sectie 2.4 is een overzicht gegeven van andere invoertechnieken waarbij de gebruiker extra invoer kan geven. Dit zijn niet noodzakelijk strokes, maar ook bijvoorbeeld aanduiden van verdwijnpunten, van vlakken, van de horizon, ...

2.1 Achterliggende technieken

2.1.1 Energieminimalisatie

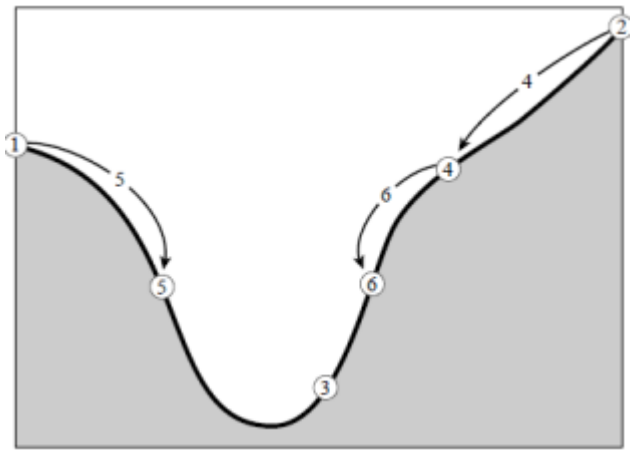
In deze sectie worden energieminimalisatieproblemen besproken, samen met een aantal oplossingen. Deze technieken kunnen gebruikt worden om andere problemen op te lossen, zoals bijvoorbeeld die in sectie 2.2 en 2.3.

Bij deze techniek is het de bedoeling een functie f met een aantal onafhankelijke variabelen a_1 tot a_n te minimaliseren, dit wil zeggen de variabelen zo kiezen dat de waarde van $f(a_1, \ldots, a_n)$ zo klein mogelijk is. Dit minimum kan globaal of lokaal zijn. Globaal is het echte minimum van de hele functie; lokaal is enkel in een eindig interval van de functie. Het probleem van het maximum vinden is analoog (bereken het minimum van $-f$) en beide problemen worden vaak aangeduid als optimalisatie.

De meeste algoritmen vinden enkel een lokaal minimum. Voor het oplossen van globale minimalisatie zijn er twee heuristieken vaak gebruikt (Press et al. 2002):

1. Kies een groot aantal (willekeurige) intervallen en bepaal het minimum van elk interval. Het eindresultaat is het minimum van deze minima
2. Zoek een lokaal minimum, kies een punt een eindige hoeveelheid verder en bepaal een (beter) nieuw lokaal minimum. Dit kan hetzelfde punt zijn.

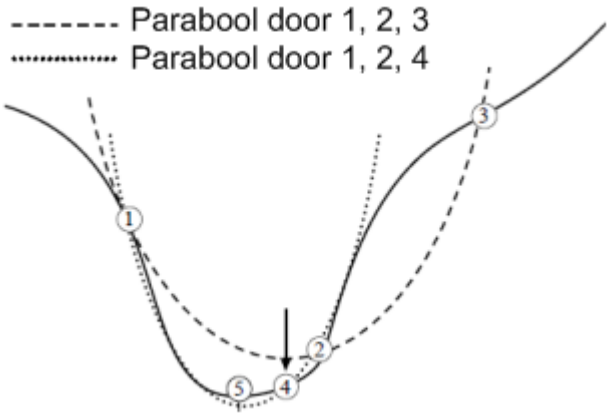
De gulden snede methode



Figuur 2.1: De gulden snede methode: Het initiële interval is [1, 2] met punt 3 het eerste geschatte minimum. Het interval wordt vervolgens verkleind tot [1, 4], [5, 4] en [5, 6] (Press et al. 2002).

Deze methode zoekt een minimum in een interval $[a, c]$ voor ééndimensionale functies in een worst-case scenario (Press et al. 2002). Gegeven is een triplet (a, b, c) waarvoor geldt $f(a) > f(b)$ en $f(b) > f(c)$. We weten dan dat b een minimum is in het interval $[a, c]$. De bedoeling is nu een kleiner interval te vinden. Kies een punt x in het grootste van de intervallen $[a, b]$ en $[b, c]$ op een fractie $0,38197$. Dit is de zogenaamde gulden snedefractie $\frac{3+\sqrt{5}}{2}$. Stel dat $[a, b]$ gekozen is, dan is het nieuwe triplet (a, x, b) als $f(b) > f(x)$, anders (a, b, x). Dit triplet kan gebruikt worden voor de volgende iteratie. Dit gaat door tot het interval kleiner is dan $\sqrt{\epsilon}\,|x|$ met ϵ de precisie van de berekeningen.

Door de fractie van de gulden snede te gebruiken, is gegarandeerd dat het interval verkleint met minstens een fractie $0,6$. Meer informatie en de berekening van deze waarde is te vinden in Press et al. (2002).



Figuur 2.2: Principe van Brent's methode: initieel wordt het interval [1, 3] (= v en w) beschouwd met minimum 2 (= x). Het minimum van de parabool door deze punten is 4, de nieuwe x. Het nieuwe interval wordt dan gevomd door 1 en 2, en het nieuwe minimum is 4. (Press et al. 2002).

Brent's methode

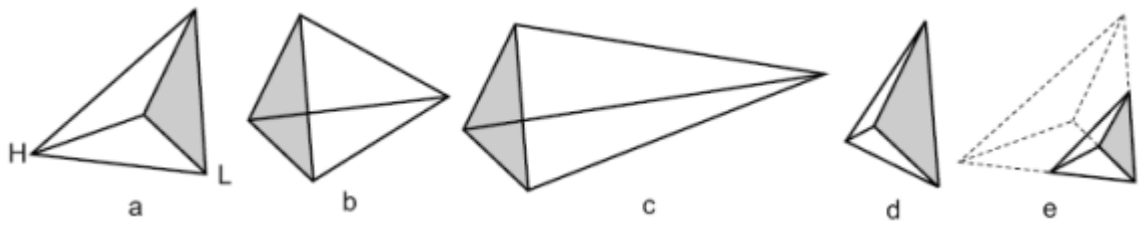
Deze methode zoekt een minimum in een interval $[a, b]$ voor ééndimensionale functies waarbij de omgeving van het minimum ongeveer parabolisch is (Press et al. 2002). Er worden zes punten beschouwd, a, b, u, v, w en x. We kiezen een zoekinterval waar het minimum inligt en begrend door a en b. x is het minimum gevonden tot nu toe, w het tweede kleinste en v houdt de oude waarde van w bij. In u wordt het punt bijgehouden waar f het laatst is geëvalueerd. Er wordt een parabool gevormd tussen x, v en w. Het nieuwe minimum wordt het minimum van deze parabool. Indien de parabool niet geschikt is, wordt de guldensnedemethode gebruikt.

Downhill simplex methode

In dit geval is het mogelijk een meerdimentionale functie te evalueren, maar heeft veel functie-evaluaties nodig. Niettemin is het een simpele en elegante oplossing. Er zijn geen afgeleiden nodig (Press et al. 2002).

Het algoritme maakt gebruik van een simplex. Dit is een geometrische figuur met $N + 1$ punten in dimensie N en alle verbindingen tussen deze punten. Voor het algoritme wordt als input $N + 1$ punten gegeven voor een functie met N parameters, en definieert zo een initiële simplex. Eén van die punten is het startpunt.

Het algoritme loopt in verschillende stappen. In elke stap wordt één van de vier operaties toegepast: reflectie, expansie, samentrekking en 1D-samentrekking (zie figuur 2.3. Hierbij



Figuur 2.3: (a) Initiële simplex (b) Na reflectie (over het grijze vlak) (c) Na reflectie en exapnsie (d) Na 1D-samentrekking (e) Na samentrekking

wordt het *slechtste* (het hoogste) punt X verplaatst. Reflectie verplaatst het hoogste punt naar zijn spiegelbeeld volgens de overige N punten. Expansie is een spiegeling, gevolgd door een verplaatsing over de as $\overline{CX}$, met C het centrumpunt van de overige N punten. Dit gebeurt als het gespiegelde punt beter is dan het beste punt. De verplaatsing is weg van het centrumpunt C.

Als het gespiegelde punt niet beter is, wordt een 1D-samentrekking uitgevoerd. Een 1D-samentrekking is een verplaatsing over de as $\overline{CX}$ naar het centrumpunt toe. Als het nieuwe punt slechter is dan alle andere punten, wordt een samentrekking uitgevoerd over alle dimensies. Hierbij verplaatsen alle punten X_i zich naar het laagste punt X_l toe over de as $\overline{X_i X_l}$.

Het algoritme loopt dus als volgt (*Downhill Simplex method* 2006):

1. Sorteer de punten zodat $f(x_{N+1}) > \ldots > f(x_1)$
2. Zoek het reflectiepunt x_r
3. Als $f(x_1) < f(x_r) < f(x_N)$, stel $x_{N+1} = x_r$.
4. Anders als $f(x_r) < f(x_1)$, genereer een nieuw punt x_e door expansie. Dit is als het gereflecteerde punt het beste is. Als het gegenereerde punt beter is, stel $x_{N+1} = x_e$, anders $x_{N+1} = x_r$.
5. Anders als $f(x_N) < f(x_r)$, genereer een nieuw punt x_c door 1D-samentrekking. Dit is als het gereflecteerde punt niet beter is.
   - Als nu $f(x_c) > f(x_{N+1})$, stel $x_{N+1} = x_c$.
   - Anders trek samen over alle dimensies naar x_{N+1}.

Powell's methode in meerdere dimensies

We definiëren de functie *linmin* (line minimizations) met parameters een fuctie f, een punt p en een richtingsvector n als $linmin(f, p, n) = \lambda$ met $f(p + \lambda n)$ is minimaal. Een simpele

toepassing van deze functie is alle eenheidsvectoren $e_1 \dots e_N$ te gebruiken en deze om beurten te voeden aan linmin tot het resultaat niet meer daalt. Dan is een lokaal minimum bereikt. Helaas is dit een zeer inefficiënte methode voor veel functies. Het is beter een aantal richtingen te kiezen zodat minimaliseren over één richting niet teniet wordt gedaan door te minimaliseren over de andere richtingen. Dit noemt men *conjugate directions*. Een mathematische definitie en afleiding is te vinden in (Press et al. 2002).

Powell's methode voor het vinden van een lokaal minimum loopt als volgt:

1. Initialiseer elke richting u_i als de eenheidvector
2. Herhaal tot de functie niet meer daalt:
   (a) Bewaar de startpositie P_0.
   (b) Voor elke $i = 1 \dots N$, $P_I = linmin(f, P_{i-1}, u_i)$.
   (c) Voor elke $i = 1 \dots N - 1$, stel $u_i = u_{i+1}$.
   (d) Stel $u_N = P_N - P_0$.
   (e) $P_0 = linmin(f, P_N, u_N)$.

Dit algoritme kan foute resultaten geven als de richtingen lineair afhankelijk worden. Daarom wordt een nieuwe richting berekend onder de volgende voorwaarden:

1. $f_E < f_0$ en
2. $2(f_0 - 2f_N + f_E)[(f_0 - f_N) - \Delta f]^2 < (f_0 - f_E)^2 \Delta f$

Met $f_0 = f(P_0)$, $f_N = f(P_N)$, $f_E = f(2P_N - P_0)$ en Δf de grootste daling in een bepaalde richting van de huidige iteratie. De richting met de grootste daling (u_i) wordt dan vervangen door de eenheidsrichting e_i. Meer informatie over deze voorwaarden is te vinden in (Press et al. 2002).

Oplossen met grafen

In Boykov & Jolly (2001) en Boykov & Funka-Lea (2006) wordt een algoritme gegeven, waarbij een functie wordt geminimaliseerd met behulp van een min-cut-algoritme. Dit algoritme is beschreven in sectie 2.1.3. Een voorbeeld van een toepassing is gegeven in sectie 2.2.2.

2.1.2 Min-Cut/Max-Flow

In deze sectie wordt het probleem van een min-cut toegelicht. Dit is een methode om een graaf in twee te splitsen volgens een aantal voorwaarden.

Grafen

Een graaf is een verzameling van knopen, V, en een verzameling van bogen, E. Met elke boog b zijn er twee knopen geassocieerd, x en y; men noemt deze de eindknopen van de boog. Dit duidt men aan met $b = (x, y)$. Als voor elke boog geldt $(x, y) = (y, x)$ noemt men dit een ongerichte graaf; anders een gerichte graaf. Aan elke boog kan men een waarde toekennen. Dit noemt men het gewicht en wordt gedefinieerd als $c : E \rightarrow \{1, \ldots, U\}$. $c(b)$ is de capaciteit van de boog b.

We beschouwen een graaf G met twee speciale knopen, de terminalen s en t, en een gewicht op elke boog. Een flow van deze graaf is een functie $f : E \rightarrow \{1, \ldots, U\}$ met $\forall a \in E : f(a) \leq c(a)$ en $\forall j \in V - \{s, t\} : \sum_{j,k} f(j, k) - \sum_{i,j} f(i, j) = 0$ (er moet evenveel *buitenstromen* als dat er *binnenstroomt*). De waarde van een flow is gedefinieerd als $|f| = \sum_{j,t} f(j, t)$.

Het probleem

Een cut van een graaf zijn twee deelverzamelingen, S en T, zodat $S \cap T = 0$, $S \cup T = G$, $s \in S$ en $t \in T$. Dit kan ook worden voorgesteld als een deelverzameling C van de bogen E, zodat S en T gescheiden zijn in de graaf $G' = \{K, E \backslash C\}$. De kost van een cut is de som van alle bogen van de graaf waarbij een knoop van de boog in S zit en de andere in T, met andere woorden in C zit. Deze kost kan gericht zijn, omdat ook de bogen gericht kunnen zijn. Het probleem is een cut te vinden met een minimale kost.

Alternatief kan er ook een maximum flow worden gevonden. In Ford & Fulkerson (1962) is bewezen dat deze problemen equivalent zijn. Het Max-flow probleem kan voorgesteld worden als de maximum hoeveelheid water die door de graaf kan stromen van één terminaal naar de andere, waarbij de kost van de bogen de capaciteit van de *buizen* aangeeft (Boykov & Kolmogorov 2004). Meer bepaald, de gesatureerde bogen vormen een minimale cut van de graaf.

Een maximum flow van een graaf moet voldoen aan de volgende constraints ($f(x, y)$ geeft de flow weer, $c(x, y)$ de capaciteit):

$f(v, w) \leq c(v, w)$	$\forall (v, w) \in V \times V$	(De flow overschrijdt de capaciteit niet.)
$f(v, w) = f(w, v)$	$\forall (v, w) \in V \times V$	(De flows lopen symmetrisch.)
$\sum_{u \in V} f(u, v) = 0$	$\forall (v, w) \in V \backslash \{s, t\}$	(Er stroomt evenveel flow in een knoop dan dat er buitengaat.)

Tabel 2.1: Constraints van het max-flow probleem

In het domein van de beeldverwerking kan dit probleem gebruikt worden om pixels te labelen, namelijk de labels toegekend aan de terminalen. Hierbij is elke andere knoop dan s en t met s

en t verbonden en ze zijn onderling eveneens volgens een orde verbonden. Deze orde kan bijvoorbeeld naburigheid zijn in een afbeelding, waarbij de knopen pixels voorstellen. Aan elke boog in deze graaf wordt een positief reeel gewicht toegekend. De bogen kunnen gericht zijn. Het gewicht bepaalt factoren voor het minimaliseren van een bepaalde functie (zie sectie 2.1.1 voor minimalisatie en sectie 2.2 voor een aantal voorbeelden van minimalisaties met min-cuts).

Oplossingen

De klassieke oplossing is beschreven in Ford & Fulkerson (1962) en genoemd naar de auteurs Ford en Fulkerson. Het principe is dat als er nog een pad is waar een flow kan zijn (met voldoende capaciteit), dan wordt die flow toegevoegd. Omdat het een gerichte graaf is, wordt er als er flow wordt toegevoegd in één richting, dezelfde capaciteit afgetrokken in de andere richting. Als er geen flows meer worden gevonden, is het algoritme beëindigd. Als het zoeken naar paden met depth-first gebeurt, noemt men dit het algoritme van Ford-Fulkerson, indien met breadth-first het algoritme van Edmonds-Karp (Edmonds & Karp 1972). Ford-Fulkerson heeft een complexiteit van $O(bogen \times maxcapaciteit)$, bij Edmonds-Karp is dit $O(knopen \times bogen^2)$.

Dinitz (1970) stelt een oplossing voor met complexiteit $O(knopen^2 \times bogen)$. Dit algoritme bestaat uit 2 fasen: In de eerste fase wordt een gelaagd netwerk opgebouwd waarbij alleen de bogen worden gebruikt waarvan de flow kleiner is dan de capaciteit (er kan nog meer flow worden toegevoegd). Laag x bestaat uit alle knopen waarvan de afstand tot de eerste terminaal, s, x is (laag 0 bestaat enkel uit s). Wanneer de tweede terminaal, t, bereikt wordt, wordt er gestopt. In de tweede fase worden deze lagen doorzocht op een pad naar de terminaal t. Indien deze gevonden is, wordt deze toegoevoegd aan de flow. Het algoritme stopt als t niet meer te bereiken is. Indien het algoritme uitgebreid wordt met een dynamische boom, wordt de complexiteit $O(knopen \times bogen \times log(knopen))$.

In Goldberg & Tarjan (1988) wordt een algoritme beschreven van complexiteit $O(knopen^2 \times bogen)$, of $O(knopen^2)$ met een optimalisatie. Dit wordt ook het push-relabel algoritme genoemd. We definiëren de afstand tussen de knopen v en w als het aantal bogen van het kleinste pad tussen v en w; $d(v)$ is een labeling gedefinieerd als de afstand tot t, met $d(s) = n$, $d(t) = 0$ en $d(v) \leq d(w) + 1$ voor elke boog (v, w). Bovendien definiëren we $|f|$ als $\sum_{v \in V} f(v, t)$, oftewel de netto flow in t. Het algoritme bestaat uit het manipuleren van preflows. Dit is een flow waarbij de hoeveelheid die binnenstroomt de hoeveelheid die naar buiten kan mag overschrijden. Dit betekent dat de derde formule in tabel 2.1 wordt gewijzigd door: $\sum_{u \in V} f(u, v) \geq 0 (\forall (v, w) \in V \setminus \{s\})$. We definiëren de graaf G zoals in de probleemstelling. Voor elke boog die we kunnen vormen tussen twee knopen en geen deel is van G, zeggen we dat het gewicht nul is. We definiëren de *flow excess* van v als de totale flow die v binnenkomt ($excess(v) = \sum_{u \in V} f(u, v)$). Het algoritme zoekt knopen v die een positieve excess hebben

en duwt deze flow richting t, waarbij het de bedoeling is een flow een stap dichter naar t te duwen. Als t niet bereikbaar is, wordt de flow naar s geduwd (de onbruikbare flows lopen zo terug naar s). Wanneer elke excess nul is, is er een maximum flow gevonden en zijn er geen preflows meer over.

Het pushen van flow van v naar w gebeurt als volgt: Als $d(v) = d(w) + 1$, v een positieve excess heeft en $c(v, w) - f(v, w)$ (het residu) is positief, dan wordt een hoeveelheid flow δ van v naar w verplaatst (opgeteld bij $f(v, w)$), met $\delta = min(excess(v), c(v, w) - f(v, w))$ (en af te trekken van $f(w, v)$). Het residu kan positief zijn als de flow nog niet gesatureerd is, of als de omgekeerde flow positief is (en de flow dus negatief is). Het relabelen zorgt dat de labels $d(v)$ van de knopen v voldoet aan de voorwaarden, namelijk door te stellen dat $d(v) = min(d(w) + 1|(v, w) \in Knopen)$. Als er geen knopen zijn, is het label ∞. Een initialisatie is $d(s) = n$ en $d(v) = 0$. Andere initialisaties zijn mogelijk.

Het algoritme loopt dan als volgt:

1. Initialiseer: $f(s, v) = c(s, v)$; $f(v, s) = -c(v, s)$; Andere: $f(v, w) = 0$
2. Initialiseer: $e(v) = f(s, v)$, $d(s) = n$; Andere: $d(v) = 0$
3. Zolang er gepushed of gerelabeled kan worden, push of relabel.

Correctheid en terminatie wordt bewezen in In Goldberg & Tarjan (1988).

Dit algoritme maakt gebruik van een afstandsfunctie waarbij elke boog één is in lengte. In Goldberg & Rao (1998) wordt dit concept gegeneraliseerd. De symbolen f en c houden hun definitie. We definiëren hier het residu als $c(i, j) - f(i, j) + f(j, i)$. Een residuboog is een boog met een strikt positief residu. De verzameling van alle residubogen noemen we E_f. De residugraaf is (V, E_f). Een blokkerende flow is een flow van s naar t waarbij elk pad een boog bevat met nul als residu. Een lengtefunctie is een functie $L : A \rightarrow R^+$. Een lengtelabeling is een functie $d : v \rightarrow R^+$ met $d(t) = 0$ en voor elke boog $(i, j) \in A : L_d(i, j) = L(i, j) + d(j) - d(i)$. $d_L(i)$ noemen we de afstand van t tot i. We noemen het volume $vol_{f,l} = c_f(a)L(a)$ als $a \in E_f$, anders 0.

Nu wordt het *binary blocking* algoritme voorgesteld. Hierbij is de lengte nul op bogen met een grote capaciteit en één anders. Hiervoor houden we een bovenlimiet F voor de residue flow (het verschil tussen de ideale flow en de huidige flow). Als F kleiner wordt dan één, is een ideale flow bereikt. De limiet wordt elke fase aangepast en elke fase is opgedeeld in update stappen. In een update stap wordt een parameter Δ gebruikt, die afhankelijk is van F. Deze is de flow die we in t willen duwen in één stap en bepaalt de lengtefunctie L:

$$L(a) = \begin{cases} 0 & c_f(a) \geq 3\Delta \\ 1 & anders \end{cases}$$

Nu wordt in elke stap een blocking flow of een flow van waarde Δ gezocht. Deze flow wordt elke keer toegevoegd aan het resultaat.

2.1.3 Energieminimalisatie met behulp van min-cuts

In deze sectie worden de min-cut en de minimalisatiealgoritmen besproken die zijn gebruikt in de implementatie. Voor de energieminimalisatie werden twee algoritmen vergeleken die gebruik maken van grafen. De bespreking is te vinden in sectie 3.4.

Min cut

Voor het minimalisatiealgoritme wordt gebruik gemaakt van een min-cut van een graaf. Hierbij gaan we een gelabelde gaaf opstellen met twee speciale knopen, s en t. Vervolgens gaan we deze graaf in twee grafen splitsen zodat elke knoop bij s of bij t hoort en dat de som van de labels van de bogen die deze twee grafen verbonden zo minimaal mogelijk is. Om deze twee grafen te berekenen wordt gebruik gemaakt van de methode in Boykov & Kolmogorov (2004). Dit algoritme is geoptimaliseerd voor afbeeldingen. Een uitleg over min cuts en gelabelde grafen is te vinden in 2.1.2.

Er wordt een gerichte graaf G beschouwd met de terminalen s en t. Het algoritme loopt in drie stappen: grow, augment en adopt. Ten alle tijden worden drie verzamelingen bijgehouden: S, T, en F. S en T vormen twee disjuncte bomen met wortel s en t. F zijn alle knopen van G die nog niet tot een boom behoren. Deze knopen noemen we vrije knopen. Initieel zijn S en T enkel gevuld met s en t, respectievelijk. F bevat alle andere knopen. De knopen in S en T zijn ofwel actief, ofwel passief. Actieve knopen zitten aan de rand, de bladeren van de boom. De passieve bevinden zich aan de binnenkant. Zie figuur 2.4 voor een graaf en de twee bomen S en T.

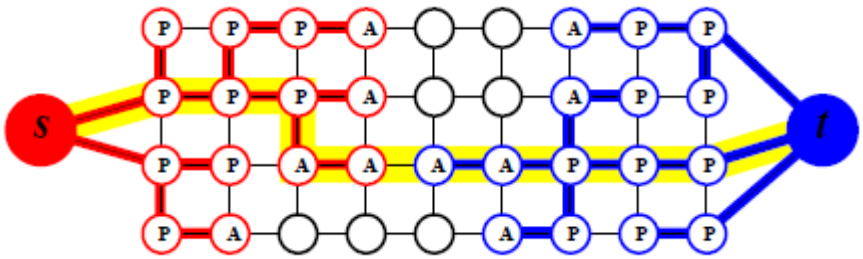
In de groeifase (grow) wordt er een pad gezocht van de source s naar de sink t. Als een knoop p verbonden is met een knoop in S of T, kan deze knoop aan S of T respectievelijk worden toegevoegd. Er is een pad gevonden als S en T elkaar raken op een bepaalde plaats.

Over dit pad wordt een flow gestuurd (augment). Elke boog in de graaf heeft een label. Dit label noemen we de capaciteit. De kleinste capaciteit op een pad noemen we de bottleneck-value. Een flow over een pad sturen gebeurt door de capaciteit op elke boog te verminderen met de bottleneckwaarde van het pad. Bogen waar de capacteit nu nul is zijn verzadigd.

Deze zijn niet meer bruikbaar en worden niet meer beschouwd. Alle knopen die nu niet meer verbonden zijn met s of t moeten weer verbonden worden (adopt).

Dit blijft doorgaan tot er geen pad meer wordt gevonden. Dan is de max flow gevonden. De splitsing bestaat uit de verzamelingen S en T. De cut zelf is de bogen met een knoop in S en een knoop in T. Deze bogen zijn verzadigd.

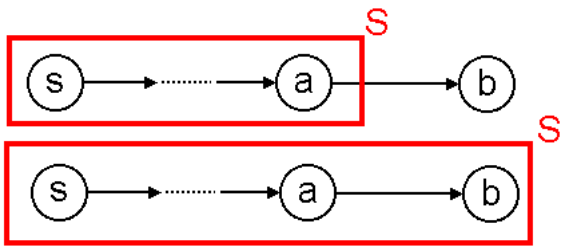
Nu zal er een gedetailleerde beschijving worden gegeven van de verschillende stappen.



Figuur 2.4: Een volledig gegroeide boom. S is rood, T is blauw. Actief is A, passief is P. Vrije knopen zijn zwart. De gele lijn geeft het gevonden pad aan. (Boykov & Kolmogorov 2004)

Grow In de groeifase worden de verzamelingen S en T uitgebreid met knopen uit F. Er wordt gekeken naar een actieve knoop a en er wordt een vrije knoop b aan S of T toegevoegd als a en b zijn verbonden met een niet-verzadigde boog (zie figuur 2.5). Voor verzadiging te testen wordt altijd gekeken vertrekkend vanuit s en verschilt voor S en T (dus a naar b voor boom S en b naar a voor boom T). Als alle buren zijn bekeken wordt b een actieve knoop, het kind genoemd. Deze knoop kan doorgroeien. a, de ouder, wordt passief. Dit proces gebeurt zowel in de S-boom als in de T-boom. Als een actieve knoop van S een actieve knoop van T raakt, is er een pad P gevonden en gaat de volgende fase in. Een pad bestaat uit een verzameling bogen die van S naar T lopen (in boom T zijn deze dus omgekeerd dan de zoekrichting).

Augment In deze fase wordt een flow gestuurd van s naar t over een pad P. Dit wil zeggen dat de capaciteit (het label of het gewicht) van een boog wordt verlaagd als deze op het pad ligt. Omdat het een gerichte graaf is, is er ook een pad in omgekeerde richting. De capaciteit op het omgekeerde pad wordt verhoogd met dezelfde waarde. De hoeveelheid is maximaal, dus de capaciteit van de boog met de kleinste capaciteit wordt gebruikt. Als een boog volledig verzadigd is na dit proces (als een boog capaciteit nul heeft), zijn niet meer alle



Figuur 2.5: De verzameling S groeit met de knoop b

knopen verbonden met s of t via niet-verzadigde bogen en zullen deze knopen niet meer tot S of T behoren. Deze knopen noemt men wezen en worden verwerkt in de volgende fase.

Adopt In deze fase wordt voor elke wees een nieuwe ouder gezocht. In dit proces kunnen nieuwe wezen ontstaan. Een nieuwe ouder voldoet aan de volgende eisen:

1. De ouder behoort tot dezelfde boom (S of T) als waar de wees toe behoorde na de Grow fase.
2. De boog tussen de ouder en de wees is niet verzadigd.
3. De ouder moet verbonden zijn met s of t (er mogen geen verzadigde bogen zijn op het pad tussen de wees en s of t). Dit moet getest worden omdat de ouder ook het kind kan zijn van een wees.

Als ee nieuwe ouder gevonden wordt, blijft de status actief of passief behouden, evenals de boom S of T. Als geen nieuwe ouder wordt gevonden, wordt de wees terug aan F toegevoegd en worden al zijn kinderen wezen.

Minimalisatie

Voor de minimalisatie wordt gebruik gemaakt van de methode zoals beschreven in Boykov et al. (2001). Deze techniek lost minimalisatieproblemen op van de volgende vorm:

$$E(f) = \sum_{p}(\sum_{(p,q)\in N_p} V_{p,q}(f_p, f_q) + D_p(f_p)) \tag{2.1}$$

$$V(a,b) = 0 \Leftrightarrow a = b \tag{2.2}$$

$$V(a,b) = V(b,a) \tag{2.3}$$

$$V(a,b) \leq V(a,c) + V(c,b) \tag{2.4}$$

In formule 2.1 is f een labelingfunctie en f_p en f_q ($\in L$) een label voor de pixels p en q ($\in P$). De labelfunctie geeft aan iedere pixel een uniek label. De verzameling N_p geeft de buren aan

van p. Dit kunnen de 4 buren zijn (horizontaal en verticaal), 8 buren (diagonaal) of zelfs een heel andere relatie. De functie $E(f)$ geeft de energie voor een bepaalde labeling. Een minimalisatiealgoritme zal een labeling construeren zodat $E(f)$ minimaal is.

De functies D en V noemt men respectievelijk de *dataterm* en de *smoothterm*. Deze kunnen afhankelijk zijn van de huidige pixels, de huidige labeling en eventuele extra gegevens ingegeven door de gebruiker. In deze toepassing zijn dit de strokes. D meet het verschil tussen een label f_p en wat het label moet zijn, zoals bijvoorbeeld ingegeven door de gebruiker. Een veel gebruikt voorbeeld is $(f_p - d_p)^2$. V is een maat van hoe *smooth* f is. Een lage waarde voor V betekent dat de labels f_p en f_q voor pixels p en q goed zijn wat betreft de smoothness. Men noemt V metrisch, als voorwaarden 2.2, 2.3 en 2.4 gelden; als alleen de eerste twee gelden is V semi-metrisch. De smoothness is niet noodzakelijk een zo gelijk mogelijke waarde, maar kan bijvoorbeeld ook een zo gelijk mogelijke afgeleide zijn. Voorbeelden van data- en smoothtermen in specifieke toepassingen zijn te vinden in sectie 2.2.

Boykov et al. (2001) beschrijft twee algoritmen voor energieminimalisatie, gebruik makend van graph cuts. Het principe van de eerste techniek, de $\alpha - \beta$ swap, is elk element (of pixel) met label l_1 of l_2 met elkaar te vergelijken. Vervolgens kan er bepaald worden welke elementen van label moeten wisselen zodat de energie van de labeling daalt. Dit wordt uitgevoerd voor elk paar labels l_1 en l_2. Als er niet meer gewisseld kan worden zonder dat de energie daalt, is er een minimum bereikt.

In de tweede techniek, de α expansion, wordt er voor elk label l gezocht welke elementen beter dit label zouden krijgen.

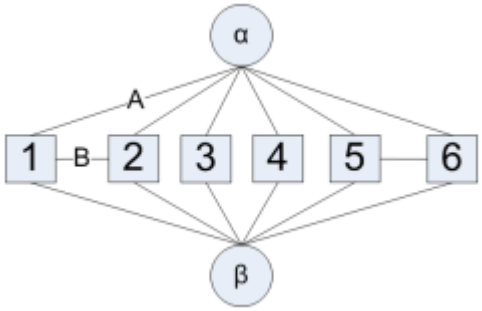
$\alpha - \beta$ swap We definiëren een labeling f als volgt: f is een verzameling $\mathbf{P} = \{P_l | l \in L\}$ met $P_l = \{p \in P | f_p = l\}$ (L is de verzameling van alle labels en P is de verzameling van alle elementen die een label krijgen, bijvoorbeeld pixels). Elk element p met label l zit dus in de verzameling P_l. Als er een swap wordt berekend, wordt dit tussen labels α en β gedaan. De verzameling van alle elementen die label α of β hebben, noemt men $P_{\alpha,\beta} = P_\alpha \cup P_\beta$. Voor een paar labels α, β noemen we een $\alpha - \beta$-swap een verplaatsing van een partitie $\mathbf{P}$ naar een andere partitie $\mathbf{P'}$, waarvoor geldt: $\forall l \neq \alpha, \beta : P_l = P'_l$. Na een swap kunnen een aantal elementen van label zijn gewisseld, dus enkel P_α en P_β kunnen wijzigen.

Het volledige algoritme loopt als volgt:

- Kies een willekeurige f
- Blijf herhalen totdat f niet meer wijzigt.

– Voor elk paar α, β

1. Zoek een f' één $\alpha - \beta$-swap van f, zodat $E(f') \leq E(f)$.
2. Als $E(f') < E(f)$, dan $f = f'$.



Figuur 2.6: Opstelling van de graaf: 1-6 zijn de elementen uit P_α en P_β. Boog A is t_1^α en boog B is $e_{1,2}$. (Boykov et al. 2001)

De stap om een $\alpha - \beta$ swap te vinden wordt opgelost met een min-cut-algoritme (zie sectie 2.1.2 voor een beschrijving van de methode en oplossingstechnieken). De graaf wordt als volgt opgesteld (zie figuur 2.6): De terminalen zijn α en β, de andere knopen de elementen uit P_α en P_β. Elementen die niet het label α of β hebben, worden niet beschouwd. Elk element $p \in P_{\alpha,\beta}$ is verbonden met de twee terminalen via een boog. Deze bogen worden t_p^α en t_p^β genoemd. Elk paar elementen p en q met $(p, q) \in N$ wordt verbonden met een boog, genaamd $e_{p,q}$. Aan elk van deze bogen wordt nu een label toegekend, genaamd de kost. De kosten van de bogen zijn:

$$
\begin{aligned}
c(t_p^\alpha) &= D_p(\alpha) + \sum_{q \in N \wedge q \notin P_{\alpha,\beta}} V(\alpha, f_q) \\
c(t_p^\beta) &= D_p(\beta) + \sum_{q \in N \wedge q \notin P_{\beta,\beta}} V(\beta, f_q) \\
c(e_{p,q}) &= V(\alpha, \beta)
\end{aligned}
$$

Een minimale cut C van deze graaf voorziet in een labeling f_p^C voor elke pixel p:

$$
f_p^C = \begin{cases} \alpha & t_p^\alpha \in C \wedge p \in P_{\alpha,\beta} \\ \beta & t_p^\beta \in C \wedge p \in P_{\alpha,\beta} \\ f_p & p \in P \wedge p \notin P_{\alpha,\beta} \end{cases} \tag{2.5}
$$

We hebben nu voor elk label f_p een nieuw label f_p^C berekend. In Boykov et al. (2001) wordt een formeel bewijs van correctheid gegeven.

Intuïtieve werking van $\alpha - \beta$ swap In deze sectie wordt uitleg gegeven waarom de graaf op deze manier wordt opgesteld. We zullen het algoritme eveneens toepassen op pixels.

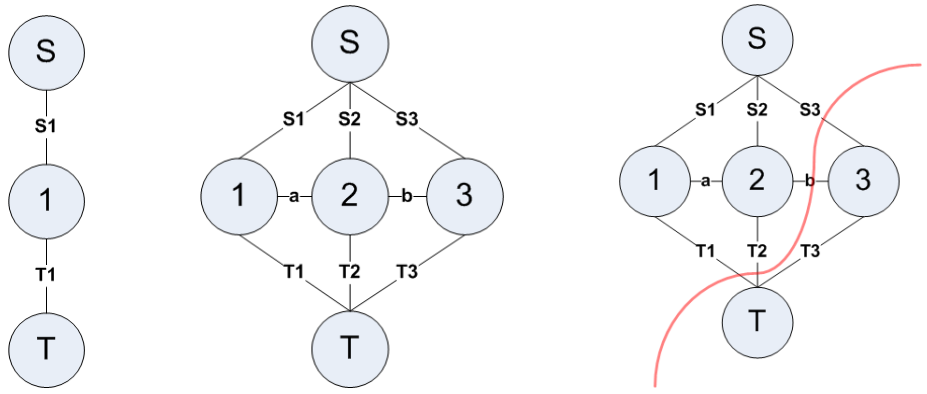
Dataterm: Veronderstel een pixel p, waar de gebruiker een label l_i aan heeft toegekend via bijvoorbeeld een stroke. De pixel heeft momenteel het label $f_p = \alpha$ (zoals in een vorige stap is berekend, of α is de initiële waarde) en het algoritme is met een swap bezig voor de labels α en β. De graaf zoals in figuur 2.7(a) zal dan een deel zijn van de volledige graaf zoals deze wordt opgesteld voor de hele afbeelding. We gaan er vanuit dat er geen buren zijn met een ander label. De kost $S1 = t_p^\alpha$ en $T1 = t_p^\beta$ in de figuur zijn dan:

$$S1 = t_p^\alpha = D_p(\alpha)$$
$$T1 = t_p^\beta = D_p(\beta)$$

Veronderstel nu dat β dichter bij l_i ligt dan α. We weten dan dat $D_p(\alpha) > D_p(\beta)$. In dit voorbeeld zal er een cut zijn op de boog met de laagste kost, namelijk $T1$. Volgens de definitie in formule 2.5 zal de pixel het label krijgen van de de source of de sink waar hij van afgesneden is, dus β. Dit is het gewenste resultaat.

Smooth term tussen pixel $p \in P_{\alpha,\beta}$ ***en*** $q \notin P_{\alpha,\beta}$: Veronderstel nu dat er buren q_n zijn met een label anders dan α of β. Dan moet er een extra energie worden bijgeteld, omdat q_n niet van label kunnen veranderen. Als een pixel van label wil veranderen, zal dat energie kosten, namelijk de smooth term tussen de pixel met label α of β en de buurpixel met een label ongelijk aan α en ongelijk aan β. Deze energie wordt bij de dataterm gevoegd. De energie tussen twee pixel in $P_{\alpha\beta}$ wordt niet beschouwd, omdat deze pixels samen kunnen veranderen. Als deze pixels zeer gelijkaardig zijn, zal het veel energie kosten om ze een ander label te geven, maar weinig energie om ze samen te laten. Dit gedrag is al vervat in andere bogen en moet niet meer worden bijgeteld bij de dataterm.

Smooth term tussen pixel $p, q \in P_{\alpha,\beta}$: De smooth term tussen pixels in $P_{\alpha\beta}$ wordt voorgesteld door de bogen tussen deze pixels zelf. Het is wenselijk dat er een cut is op de plaats waar de energie laag is om de pixels een verschillend label f_p en f_q te geven. Veronderstel bijvoorbeeld figuur 2.7(b). Stel dat pixelknoop 1 en 3 labels hebben gekregen van de gebruiker, l_1 en l_3 respectievelijk. Als er een swap wordt gedaan met de labels $\alpha = l_1$ en $\beta = l_3$, dan is $S1 = t_1^\alpha$ en $T3 = t_3^\beta$ groot. Veronderstel dat pixel 1 en pixel 2 gelijkaardig zijn, en verschillend van pixel 3. Dan zal de energie tussen pixel 1 en pixel 2 groot zijn als deze een verschillend label krijgen en de energie tussen pixel 2 en pixel 3 klein. Dit is precies wat we toekennen aan de bogen, namelijk de energie $V(\alpha, \beta)$. Merk op dat $V(\alpha, \beta) = V(\beta, \alpha)$. De kost $e_{p,q}$ van de boog zal dus groot zijn voor pixels die gelijkaardig zijn (en zullen dus de neiging hebben bij elkaar te blijven) en laag voor verschillende pixels (waar een cut verwacht wordt). In het



(a) Data-term (b) Smoothterm (c) Graphcut

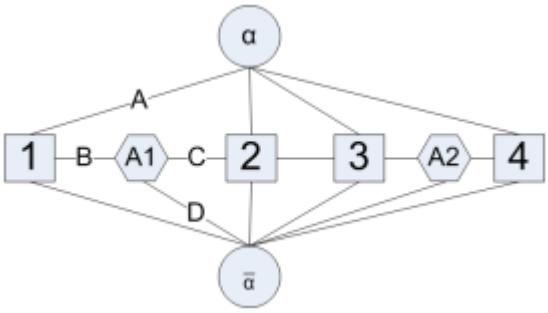
Figuur 2.7: Intuïtieve werking van de $\alpha - \beta$ swap.

voorbeeld is $a = e_{1,2}$ hoog en $b = e_{2,3}$ laag. Het verwachte resultaat is dus een cut zoals in figuur 2.7(c).

α expansion Het tweede algoritme maakt gebruik van de α-expansie en lijkt op het eerste. Een α-expansie is een verplaatsing van een partitie **P** naar een andere partitie **P'**, waarvoor geldt: $P_\alpha \subseteq P'_\alpha$ en $\forall l \neq \alpha : P'_l \subseteq P_l$. Dit houdt in dat P_α groter kan worden, maar niet kleiner.

- Kies een willekeurige f
- Blijf herhalen totdat f niet meer wijzigt.
  - Voor elk label α
    1. Zoek een f' één α-expansie van f, zodat $E(f') \leq E(f)$.
    2. Als $E(f') < E(f)$, dan $f = f'$.

In dit geval wordt verondersteld dat V metrisch is. De minimalisatie wordt weer opgelost met een min-cut-algoritme. De graaf wordt als volgt opgesteld (zie figuur 2.8): De terminalen zijn α en $\overline{\alpha}$, de andere knopen de elementen uit P en assisterende knopen $a_{p,q}$ zodat $(p,q) \in N$ en $f_p \neq f_q$. Elk element p is verbonden met de twee terminalen en de bogen worden t_p^α en t_p^β genoemd. Elk paar elementen p en q met $(p,q) \in N$ en $f_p = f_q$ wordt verbonden met een boog, genaamd $e_{p,q}$. Elk paar elementen p en q met $(p,q) \in N$ en $f_p \neq f_q$ wordt onrechtstreeks verbonden met een koppel bogen via een assisterende boog $a_{p,q}$: dit zijn de bogen $(p, a_{p,q})$ en $(a_{p,q}, q)$. Bovendien wordt er een boog gemaakt tussen $a_{p,q}$ en $\overline{\alpha}$. We noemen deze bogen $e_{p,a_{p,q}}$, $e_{a_{p,q},q}$ en $t_p^{\overline{\alpha}}$. De kosten van de bogen zijn:



Figuur 2.8: Opstelling van de graaf: 1-4 zijn de elementen uit P. Boog A is t_1^{α}, boog B is $e_{1,A1}$, boog C is $e_{A1,2}$ en boog D is $t_A^{\overline{\alpha}}1$. (Boykov et al. 2001)

$$
\begin{aligned}
c(t_p^{\overline{\alpha}}) &= \infty & p &\in P_\alpha \\
c(t_p^{\overline{\alpha}}) &= D_p(f_p) & p &\notin P_\alpha \\
c(t_p^{\alpha}) &= D_p(\alpha) & p &\in P \\
c(e_{p,a}) &= V(f_p, \alpha) & (p,q) &\in N, f_p \neq f_q \\
c(e_{a,q}) &= V(\alpha, f_q) & (p,q) &\in N, f_p \neq f_q \\
c(t_a^{\overline{\alpha}}) &= V(f_p, f_q) & (p,q) &\in N, f_p \neq f_q \\
c(e_{p,q}) &= V(f_p, \alpha) & (p,q) &\in N, f_p = f_q
\end{aligned}
$$

Een minimale cut C van deze graaf voorziet in een labeling f^C:

$$
f_p^C = \begin{cases} \alpha & t_p^{\alpha} \in C \\ f_p & t_p^{\overline{\alpha}} \in C \end{cases}
$$

Er is bewezen dat de optimale α-expansie f' van f gelijk is aan f^C. Bewijzen van correctheid en voorbeelden zijn terug te vinden in Boykov et al. (2001).

2.2 Toepassingen

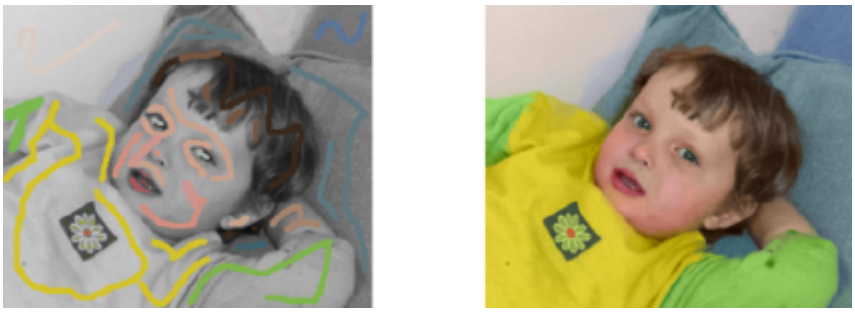
Dit deel beschrijft enkele toepassingen waarbij de gebruiker strokees maakt op een afbeelding. De strokes kunnen kleuren zijn, normalen, dieptes, enzovoort. In alle gevallen wordt er één afbeelding als invoer genomen. Deze technieken vormen dus een goed alternatief voor lang, saai en repititief werk. De belangrijkste achterliggende technieken zijn besproken in sectie 2.1. Het maken van dieptemappen met strokes wordt besproken in sectie 2.4, waar ook andere technieken worden besproken waarbij de gebruiker het algoritme stuurt voor het maken van dieptemappen.

Het merendeel van de technieken is gebaseerd op het minimaliseren van een functie van de vorm:

$$E(f) = E_{smooth}(f) + \lambda E_{data}(f)$$

waarbij $E_{smooth}(f)$ ervoor zorgt dat het resultaat niet te grillig is en $E_{data}(f)$ de ingegeven constraints respecteert. De functie f is een labeling van pixels van een afbeelding (kleur, diepte, ...). De grijswaarden van de afbeelding wordt voorgesteld door I, pixels door $p, q \in P$ en een grijswaarde van een pixel door I_p. Een burenrelatie wordt aangegeven door $N(p) = \{q|q \text{ is een buur van } p\}$.

2.2.1 Inkleuren van zwart-witafbeeldingen



Figuur 2.9: Voorbeeld van het inkleuren (Levin et al. 2004)

Bij deze toepassing wordt een zwart-witafbeelding of video als input genomen. De bedoeling is dat deze afbeeldingen ingekleurd worden, eventueel volledig automatisch. Een gelijkaardig probleem is het veranderen van kleuren van een afbeelding. Deze problemen kunnen door veel en repetitief werk opgelost worden. De photoshoptutorials *Colorize image using shapes* (2008), *Black and white to color* (2008) en *TechEBlog - How to Colorize Black ans White Images in Photoshop* (2008) maken dit duidelijk.

In Levin et al. (2004) wordt een techniek gepresenteerd waarbij de gebruiker enkele gekleurde lijnen moet trekken over de afbeelding. Dit stelt de algemene kleur voor van het aangegeven gebied. Het principe is dat naburige pixels die ongeveer dezelfde intensiteit hebben, ook ongeveer dezelfde kleur hebben. Deze kleuren worden dan gepropageerd over de afbeelding door het oplossen van een energieminimalisatieprobleem. Hierdoor wordt het verschil tussen de kleur van een pixel p en het gewogen gemiddelde van zijn buren geminimaliseerd. De aangegeven kleuren vormen harde constrains. De formule hiervoor is:

$$E(U) = \sum_p \left(U(p) - \sum_{q \in N(p)} w_{pq} U(q) \right)^2$$

waarbij U staat voor één kleurenkanaal in YUV ($Y = I$), $U(p)$ en $U(q)$ geven kleurintensiteiten weer. Deze formule is analoog voor $E(V)$. Er is enkel een dataterm gebruikt, geen smoothing term.
De gewichtsfunctie w_{pq} is groot als $Y(p)$ en $Y(q)$ gelijkaardig zijn en klein als ze sterk verschillen. Er is getest met twee gewichtsfuncties, één gebaseerd op het kwadraat van het verschil en de andere op de genormaliseerde correlaie van de twee intensiteiten:

$$w_{pq} = e^{-(Y(p)-Y(q))^2/2\sigma_p^2}$$
$$w_{pq} = 1 + \frac{1}{\sigma_p^2}(Y(p) - \mu_p)(Y(q) - \mu_p)$$

σ_p en μ_p zijn de variantie en het gemiddelde van de intensiteiten rond p.

De minimalisatie is kwadratisch en de constraints zijn lineair, dus het probleem is een lijst van lineaire vergelijkingen. Het multigrid algoritme (zoals beschreven in Press et al. (2002)) en de matlabs ingebouwde *least squares solver for sparse linear systems* werden gebruikt voor de minimalisatie.

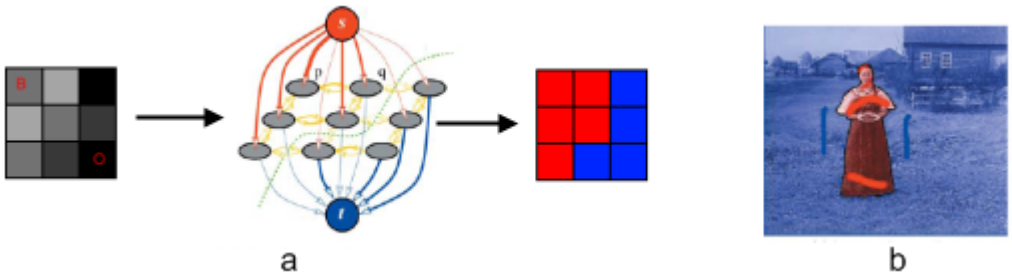
De techniek is ook bruikbaar voor het wijzigen van kleuren of het inkleuren van videobeelden. Een gelijkaardige techniek is beschreven in Nie et al. (2007).

2.2.2 Regiosegmentatie en voorgrondextractie

Dit probleem omvat het opdelen van een 2D-afbeelding in een voor- en een achtergrond, eventueel zelfs meerdere regio's. Dit kan gaan om het verwijderen van storende elementen uit vakantiefoto's, objecten en personen in een andere omgeving plaatsen, medische beelden analyseren, enzovoort. Typisch is dit een zeer tijdrovende bezigheid, ondanks de beschikbaarheid van intelligente tools in fotobewerkingsprogramma's, zoals de *Magic Wand Tool* van Photoshop (*Photoshop CS3 editions* 2008).

In Boykov & Jolly (2001) en Boykov & Funka-Lea (2006) wordt een methode gepresenteerd die gebruik maakt van graph cuts (zie sectie 2.1.2). De afbeelding wordt voorgesteld door een graaf, waarbij de pixels ($\in P$) knopen zijn en de bogen een burenrelatie N aangeven. Dit kunnen de vier buren zijn, acht buren, ... Er worden twee terminalen s en t toegevoegd en elke terminaal wordt met elke pixel verbonden. Vervolgens wordt een energiekostfunctie $E(f) = E_{smooth}(f) + \lambda E_{data}(f)$ gedefinieerd voor de soft constraints, waarbij f een rij is van alle pixels die een labeling (voorgrond of achtergrond) voorstelt.

De dataterm is gegeven door de formule $E_{data}(f) = \sum_{p \in P} R_p(f(p))$. R_p is bijvoorbeeld hoe de intensiteit van een pixel zich verhoudt in een bekend intensiteitsmodel van de afbeelding.



Figuur 2.10: (a) Proces van het segmenteren. De linkse afbeelding is de invoer, waarbij O en B de annotaties van de gebruiker aangeven. De middelste afbeelding is de gegenereerde graaf. De rechtse afbeelding is de uitvoer, met het rode de voorgrond en het blauwe de achtergrond. (b) Voorbeeld van een segmentatie (Boykov & Funka-Lea 2006).

Deze kan bekomen worden uit de harde constraints.

De smooth term wordt gegeven door:

$$E_{smooth}(f) = \sum_{(p,q)\in N} B_{(p,q)}\delta(f(p), f(q))$$

$$\delta(f(p), f(q)) = \begin{cases} 1 & f(p) \neq f(q) \\ 0 & anders \end{cases}$$

$B_{(p,q)}$ is een pelanty voor discontinuïteiten tussen p en q, en is groot als p en q gelijkaardig zijn. Deze kan gebaseerd zijn op afstand, intensiteit of gradientrichting. Als voorbeeld is gegeven: $B_{(p,q)} = exp(-\frac{(I_p - I_q)^2}{2\sigma^2})\frac{1}{dist(p,q)}$.

Er kunnen ook *hard constraints* gedefinieerd worden. Hierbij wordt expliciet aangegeven of een bepaalde pixel voorgrond of achtergrond is. Dit kan manueel via een gebruikersinterface gebeuren, of geautomatiseerd worden. De bedoeling is de energiekostfunctie te minimaliseren, terwijl de harde constraints behouden blijven.

Nadat deze *constraints* zijn gedefinieerd, worden deze gekoppeld aan de graaf. Er is een nieuwe techniek voorgesteld om de graaf op te stellen. De terminalen worden s en t genoemd, twee naburige knopen (geen terminalen) worden p en q genoemd. De gewichten worden dan toegekend als in tabel 2.2.

Wanneer op deze graph nu een min-cut wordt berekend, zullen alle voorgrondpixels met s verbonden zijn, en alle achtergrondpixels met t. De cut wordt berekend met de methode zoals beschreven in .Goldberg & Tarjan (1988) en in sectie 2.1.2. In Boykov & Funka-Lea (2006)

Boog	Gewicht
$\{p, q\}$	Randterm
$\{p, s\}$	∞ als een harde constraint op voorgrond, 0 als een harde constraint op achtergrond, anders regionale term
$\{p, t\}$	∞ als een harde constraint op achtergrond, 0 als een harde constraint op voorgrond, anders regionale term

Tabel 2.2: Gewichtstoekenningen voor regiosegmentatie met behulp van graph cuts

wordt verder nog bewezen dat deze cut een minimalisatie van de energiekostfunctie en het respecteren van de *constraints* representeert.

Als deze methode wordt gekoppeld aan een gebruikersinterface kan dit een krachtige tool opleveren. De gebruiker geeft aan wat zeker voorgrond en wat zeker achtergrond is. Dit vormen dan de harde *constraints* van het algoritme. Bovendien kunnen deze annotaties gemakkelijk worden gewijzigd, zonder dat het hele algoritme opnieuw moet worden uitgevoerd. Dit versnelt de interactie met de gebruiker.

Figuur 2.11: (a) Invoer en annotatie van de voorgrond (b) Extra aanduidingen van voorgrond en achtergrond (c) Resultaat (d) Detail van de voorkant van het resultaat (Rother et al. 2004).

De gebruiker kan ook andere inputtechnieken gebruiken, zoals bijvoorbeeld in Rother et al. (2004). Dit is een uitbreiding op bovenstaande methode. Het aangeven van voorgrond- en achtergrondpixels is niet meer nodig, een rechthoek rond het object tekenen volstaat. Alles binnen de rechthoek wordt gelabeled als voorgrond, de rest is achtergrond. Het is wel nog mogelijk extra informatie toe te voegen na de eerste berekening. Andere verschillen zijn dat het algoritme nu iteratief loopt en dat de kleurwaarden gemixed worden, in plaats van grijswaarden te gebruiken. Bovendien wordt matting gebruikt; er is dus geen segmentatie meer in voorgrond- en achtergrondpixels, maar elke pixel krijgt een α-waarde ($0 \leq \alpha \leq 1$) die bepaalt

hoe doorzichtig deze is: $Kleur = \alpha \times voorgrondkleur + (1 - \alpha) \times achtergrondkleur$. Dit is een veralgemening van een gewone segmentatie, als $\alpha \in \{0, 1\}$.

De energiefunctie wordt op de volgende manier aangepast:

$$E_{smooth} = \sum_{p} -log(h(I_n; \alpha_n))$$

$$E_{data} = \sum_{(p,q) \in N} dist(p,q)^{-1} exp(-\beta(I_q - I_p)^2)$$

$$\delta(\alpha_p, \alpha_q) = \begin{cases} 1 & \alpha_p \neq \alpha_q \\ 0 & anders \end{cases}$$

De functie h stelt een grijswaardendistributie voor, gebaseerd op de door de gebruiker gelabelde pixels. Deze functie werd al voorgesteld in Boykov & Jolly (2001). De factor β vermindert de neiging om de functie smooth te maken in gebieden met hoog contrast.

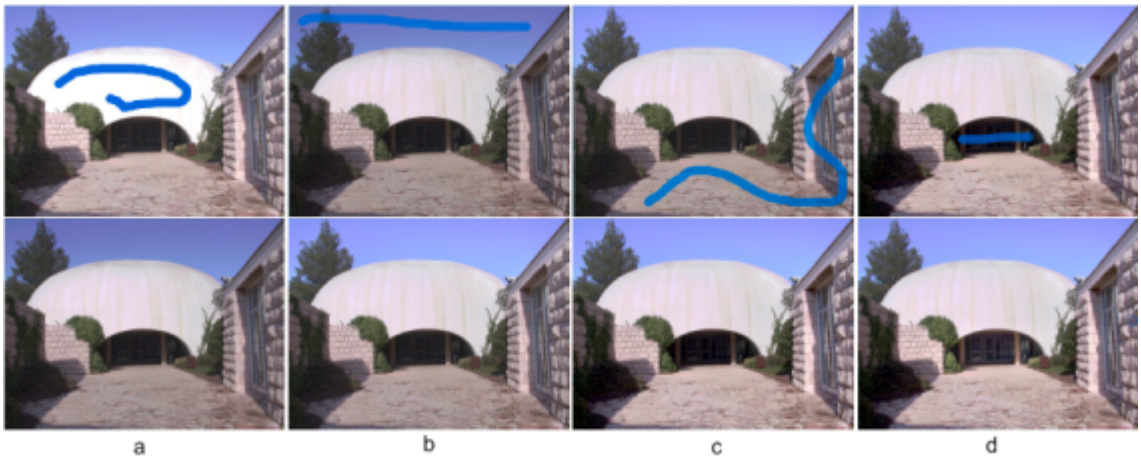
In Wang & Cohen (2005) en Levin et al. (2008) wordt eveneens matting gebruikt. Elke pixel wordt gelabeled met een alpha-waarde α en een onzekerheidswaarde u. De gebruiker kan voorgrond en achtergrond aanduiden, dan is α respectievelijk 1 en 0, en u 0. Andere pixels hebben en α van 0.5 en een u van 1. Het algoritme werkt iteratief en berekent de alpha-waarde voor de pixels rondom de al berekende pixels. De oplossing wordt berekend door het oplossen van een *Conditional Random Field* (Weinman et al. 2004).

Deze methodes staan in contrast met de methodes die geen input vereisen, zoals randdetectie, segmenteren met drempelwaarden, regio-gebaseerde technieken, enzovoort. Deze technieken worden besproken in Gonzalez & Woods (2007), hoofdstuk 10. Andere technieken zijn extractie van blue screens (Smith & Blinn 1996), gebruik van *local kernel histograms* voor extractie met bekende achtergrond met wisselend licht (Noriega & Bernier 2006) en analyse van videobeelden om de achtergrond te bepalen (Kim et al. 2006).

2.2.3 Beeldcorrectie

Hieronder verstaan we het aanpassen van contrast, belichting, saturatie, witbalans en andere eigenschappen van afbeeldingen. Globaal is dit een simpel probleem, maar lokale aanpassingen zijn zeer moeilijk.

In Lischinski et al. (2006) kan de gebruiker een klein gebied aanduiden waar wijzigingen moeten worden aangebracht. Hiervoor worden meerdere soorten brushes aangeboden. Buiten de standaard tekentool is er ook een brush die enkel een bepaalde luminantiewaarde, enkel een kleur of enkel over- en onderverzadigde pixels selecteert. Zo wordt aan elke pixel een gewicht



Figuur 2.12: De bovenste rij geven de annotaties weer; de onderste rij het resultaat. (a) De overbelichting wordt hersteld om details zichtbaar te maken (b) De lucht wordt lichter gemaakt (c) Verhoging in contrast (d) Contrast en onderbelichting onder de koepel worden hersteld. Merk op dat bij elke actie enkel de gewenste delen worden aangepast (Lischinski et al. 2006).

toegekend. De mogelijke wijzigingen zijn: sluitertijd (radiantie) aanpassen, contrast, verzadiging en kleurtemperatuur. Deze wijzigingen worden dan gepropageerd over de afbeelding naar gelijkaardige gebieden. Ook dit gebeurt door het oplossen van een energieminimalisatieprobleem. Hierbij wordt rekening gehouden met de aangegeven constrains (de gebieden om aan te passen).

De energiekostfunctie die in deze methode wordt gebruikt is als volgt gedefinieerd:

$$E(f) = \sum_P w(p)(f(p) - g(p))^2 + \lambda \sum_P h(\nabla f, \nabla L)$$

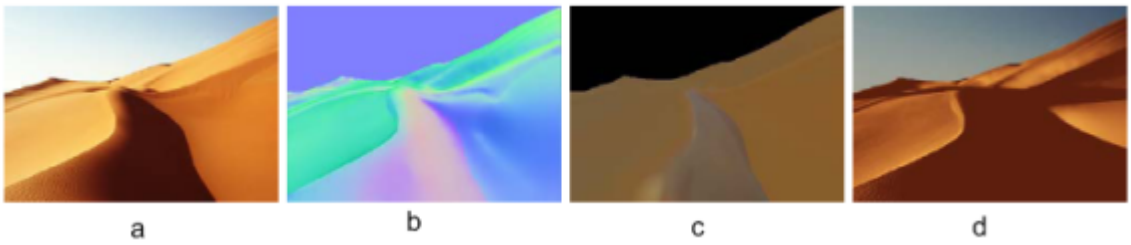
De eerste term is de *data term*, de tweede de *smoothing term*. $w(x)$ geeft het gewicht aan dat de gebruiker heeft bepaald met de brushes. De smoothing term probeert de gradient zo klein mogelijk te maken, behalve daar waar de gradient in de illuminantiewaarden van de afbeelding L groot is (randen bijvoorbeeld). De gebruikte functie is:

$$h(\nabla f, \nabla L) = \frac{|f_x|^2}{|L_x|^\alpha + \epsilon} + \frac{|f_y|^2}{|L_y|^\alpha + \epsilon}$$

α controleert de gevoeligheid en ϵ vermijdt een deling door nul.

2.2.4 Herbelichten van afbeeldingen

Dit is het probleem om de belichting van een echte foto te wijzigen. De belichting is sterk afhankelijk van de riching van de oppervlakten van de objecten, of meer bepaald de normalen. Een mogelijke oplossing is dezelfde scene meerdere keren fotograferen in verschillende belichtingen, zoals voorgesteld in Fuchs et al. (2005), Lin et al. (2002) en Masselus et al. (2003). Deze methoden vereisen echter dat de extra belichting gecontroleerd en bekend is. Een andere oplossing houdt in dat een 3D-model van de scene wordt gemaakt en dat er een renderingstechniek wordt toegepast. Zie hiervoor Marschner et al. (1997) en sectie 2.3. Deze twee oplossingen zijn echter niet praktisch als slechts één foto beschikbaar is.



Figuur 2.13: (a) Invoer (b) De normalenmap (c) De reflectiemap (d) Een herbelichting (Okabe et al. 2006)

In Okabe et al. (2006) wordt een methode voorgesteld die als input één afbeelding neemt. Op deze afbeelding kan de gebruiker aangeven in welke richting de oppervlakten gericht zijn (de normalen). Dit gebeurt op een pen-tablet input systeem.

De eerste stap is de afbeelding opdelen in segmenten. Dit wordt automatisch gedaan waarbij de gebruiker extra input kan geven en de segmentatie kan aanpassen. De tweede stap is de normalen aanduiden per regio. De mogelijkheden zijn: een normale brush (de normaal is de hoek van de pen), een copy-brush (deze kopieert de normaal van een punt naar een ander) en een blur brush (deze maakt de normalen minder grillig). Om grote vlakken op te vullen, wordt een smooth fill gereedschap gebruikt. Deze propageert normaalgegevens over een groter gebied. Dezelfde techniek als in sectie 2.2.1 is gebruikt, waarbij $U = \frac{N_x}{N_z}$ en $V = \frac{N_y}{N_z}$.

Na de invoer wordt de normalenmap verfijnd en wordt geschat waar zich reflectie bevindt. Er wordt uitgegaan van één licht in richting l en van het belichtingsmodel $I(r) = I_a k_a + I_p S_p(k_d \langle l, n \rangle + k_s H_p)$. I_a is de sterkte van het ambient licht, I_p de sterkte van het puntlicht, $S_p(r)$ is een schaduwterm met H_p de intensiteit van spiegelende highlights en n de normaal van het oppervlakte. Nu moet l geschat worden. dit kan gedaan worden door het minimaliseren van de functie (met regiolabel i en albedowaarden k):

$$E(l) = \sum_i \sum_r (I(r) - A_i - D_i \langle l, N(r) \rangle)^2$$
$$A_i = I_a k_i$$
$$D_i = I_p k_i$$

Tijdens de minimalisatie zijn A_i en D_i bepaald per regiolabel i. Nu kunnen schaduw- en speculaire highlightsregio's bepaald worden:

$$S_p(r) = \begin{cases} 1 & I(r) > k_i(I_a + t_s I_p) = A_i + t_s D_i \\ \frac{I(r) - A_i}{t_s D_i} & I(r) > k_i I_a = A_i \\ 0 & anders \end{cases}$$

$$I_p k_s H_p(r) = \begin{cases} I(r) - A_i - D_i & I(r) > A_i + (1 + T_h) D_i \\ \frac{I(r) - A_i - D_i}{4 T_h D_i} & I(r) > A_i + (1 - t_h) D_i \\ 0 & anders \end{cases}$$

Het verbeteren van de normalenmap en de reflectieinschatting wordt gedaan door het verminderen van de fout:

$$e(k_i, N) = I(r) - I_a K_i - I_p S_p(k_i \langle l, n \rangle + k_s H_p)$$

Wanneer dit gedaan is, is het resultaat een gedetailleerde reflectie- en normalenmap, waarmee nieuwe belichtingen bepaald kunnen worden.

2.3 Maken van dieptemappen en 3D-modellen

Deze sectie bespreekt het maken van dieptemappen en 3D-modellen zonder interactie van de gebruiker. Meer bepaald worden het aftasten van objecten, meerdere afbeeldingen van objecten en afbeeldingen met domeinkennis besproken.

2.3.1 Dieptemappen maken door het scannen van reële objecten

De gemakkelijkste methode om een dieptemap te creëren is door een object af te tasten of te scannen en zo de diepte meten. Dit vereist wel dat het object voorhanden en handelbaar (niet te groot, verplaatsbaar, niet te poreus, bereikbaar, ...) is. Bovendien kan de apparatuur duur uitvallen. Niettemin is dit de meest nauwkeurige methode voor het maken van dieptemappen.

Een voorbeeld is een co-ordinate measuring machine (CMM). Dit apparaat heeft een uiteinde, een probe genoemd, die over het object gaat en de positie meet. Door verschillende punten

Figuur 2.14: (a) Renscan5 van Renishaw (*Renishaw - CMM* 2008*a*) (b) Het scannen van een beeld van Michelangelo (Levoy et al. 2000).

af te gaan, kan het object gedigitaliseerd worden. Zie ook *Renishaw - CMM* (2008*a*) en *Renishaw - CMM* (2008*b*).

Contact is niet altijd nodig. Een laserscanner schiet een laserstraal naar het object en meet de tijd tot de straal terug is. De lichtsnelheid is bekend, dus kan de afstand tot het object berekend worden. Omdat ook de positie van het laserapparaat bekend is, kan een dieptemap bepaald worden. Nadeel van deze techniek is dat de tijdsmeting heel accuraat moet zijn. Een andere methode is lasers uitsturen op verschillende frequenties en de faseverschuiving meten. Een betere techniek is een straal van 1 frequentie opsplitsen en een deel naar het object laten gaan. De weerkaatste straal wordt terug samengevoegd met het andere deel. Dit zal een interferentie veroorzaken. Het resultaat laat het verschil tussen de afstand zien die de twee stralen hebben afgelegd.

In Levoy et al. (2000) werd een lasertechniek gebruikt voor het opmeten van verschillende creaties van Michelangelo. Er is een laserscanner gebouwd met een nauwkeurigheid van minder dan een millimeter. Op *Scanning Monticello* (2002) werd er een model gecreëerd van Thomas Jeffersons (*Thomas Jefferson's monticello* 2008) huis.

Guisson (2005) gebruikt evenees een zelfgebouwde scanner. Deze scanner bestaat uit een ronde schijf. Deze schijf moet handmatig gedraaid worden. Op een redelijke afstand staat een digitale camera. Deze camera moet vooraf worden gekalibreerd met behulp van een dambordpatroon. Eerst wordt een foto genomen van de opstelling zonder achtergrond. Daarna wordt het object in het midden van de schijf geplaatst en vanuit een aantal hoeken gefotografeerd. Deze foto's wordt gevoed aan het algoritme in Kutulakos & Seitz (2000).

2.3.2 Reconstructie en dieptemappen uit meerdere afbeeldingen

Maken van dieptemappen met twee afbeeldingen met normale foto's

Bij deze techniek wordt er gezocht naar overeenkomende punten in twee afbeeldingen. Door de verschillende posities van de camera's zullen objecten in de verschillende afbeeldingen op een andere plaats staan. Hoe verder een object van de camera afstaat, hoe minder de verplaatsing tussen verschillende afbeeldingen zal zijn. Op die manier kan de afstand van objecten tot de camera ingeschat worden. In Scharstein & Szeliski (2002) en Seitz et al. (2006) wordt een overzicht gegeven van algoritmes die deze methode gebruiken.

Nu worden enkele aandachtspunten vermeld voor het indelen van de algoritmen voor het maken van dieptemappen en modellen, gebruik makend van twee afbeeldingen. Een dieper overzicht van de taxonomie is te vinden in Scharstein & Szeliski (2002). Hierna volgen voorbeelden van enkele van deze algoritmen.

Een eerste punt is het domeinmodel. In veel algoritmen wordt verondersteld dat de oppervlakten van de gefotografeerde objecten Lambertiaans (*Definition for word(s) lambertian surface - The Photonics Directory* 2008) zijn. Dit wil zeggen dat er geen spiegelende of doorzichtige oppervlakten zijn en dat de illuminantie in alle richtingen hetzelfde is. Een andere kwestie is de camera. Dit kan een orthogonaal of een projectief model zijn en kan belangrijk zijn voor bepaalde aannames.

Een tweede punt is de representatie. Het meest gebruikte is een disparity map $d(x, y)$. Deze geeft het verschil in verplaatsing weer van een punt (x, y) in een referentieafbeelding tegenover een andere afbeelding. Hoe groter de dispariteit, hoe minder de afstand. Het doel van een stereo-matching algoritme is het vinden van een functie d, zodat voor elke waarde van (x, y) slechts één waarde is. Dit kan ook worden gezien als het maken van een oppervlakte in de ruimte (x, y, d), waarbij voorwaarden worden gesteld, bijvoorbeeld hoe smooth dit oppervlakte is. Andere voorstellingen zijn voxels (De Bonet & Viola 1999, Eisert et al. 1999, Szeliski & Golland 1999), polygonen (Faugeras & Keriven 2002, Pons et al. 2003, Soatto & Jin 2003), een visual hull Matusik et al. (2000, 2001), een verzameling lagen, ...

De meeste algoritmen voor stereo-matching zijn gebaseerd op de volgende vier stappen:

1. Een kostfunctie berekenen
2. Kostaggregatie
3. Dispariteitberekening
4. Post-processing

Bovendien zijn er lokale en globale algoritmen. De lokale werken enkel in een klein venster rond een pixel. De globale werken op de hele afbeelding tegelijkertijd.

Stap één bepaalt een kost tussen twee pixels, of hoe deze matchen tussen de twee afbeeldingen. Veel gebruikte voorbeelden zijn de *squared intensity differences* functie $(I_a - I_b)^2$ en de *absolute intensity differences* functie $|I_a - I_b|$. Er kan ook gebruik gemaakt worden van randinformatie (Baker 1980) en de laplaciaan (Nishihara 1987)

Stap twee wordt vooral gebruikt in lokale algoritmen en meestal genegeerd in globale. Hierbij wordt een bewerking gedaan in een deel van de ruimte (x, y, d), bijvoorbeeld het gemiddelde nemen of sommeren. Ook hier kan een onderscheid gemaakt worden tussen 2D (de parameter d is vast) (Arnold 1983, Okutomi & Kanade 1992) of 3D (Grimson 1984).

Stap drie maakt de dispariteitsfunctie zelf aan. In lokale algoritmen is het werk al gedaan in stap twee en deze stap is dan triviaal: kies de waarde met de minimale kost voor het punt (x, y). Voor globale algoritmen begint het werk hier. Het basisprincipe is dat er een functie wordt opgesteld die wordt geminimaliseerd, bijvoorbeeld met de methoden uit sectie 2.1.1. De meest gebruikte is dezelfde functie zoals in sectie 2.2, namelijk:

$$E(f) = E_{smooth}(f) + \lambda E_{data}(f)$$

met f de dispariteitsfunctie d.

Stap vier kan bijvoorbeeld zijn:

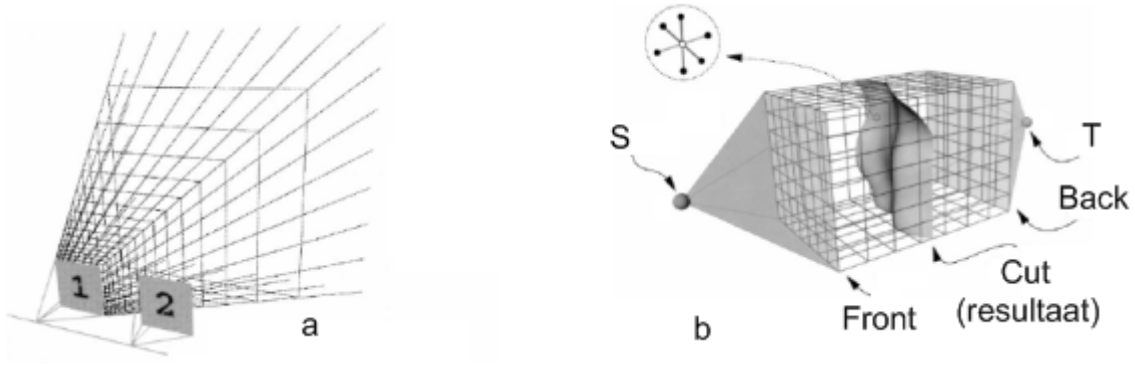
- Een verfijning van de dispariteitsfunctie. Dit kan bijvoorbeeld door te interpoleren, ...
- Een filter om fouten weg te werken.
- Gaten opvullen.

Roy (1999) maakt gebruik van het minimum-cut probleem (sectie 2.1.2) voor het oplossen van de minimalisatiestap van het stereoprobleem. We noemen de matching space de ruimte van alle mogelijke diepteoppervlakten. Deze ruimte is door de klassieke algoritmen begrensd door een afgevlakte piramide, met de top in het camerapunt (zie figuur 2.15 a). De front (het projectievlak) en de back (de far plane) zijn het topvlak en het grondvlak. Een diepte-bepaling is een oppervlakte dat de front scheidt van de back. De ruimte wordt gekwantiseerd volgens diepte (wordt een kubus) of volgens dispariteit (wordt een kegel). Een punt wordt voorgesteld door een vector $\begin{bmatrix} a' & b' & c' & 1 \end{bmatrix}^T$ met assen a, b en c, en deze liggen tussen de kwantisatiewaarden a_s, b_s en d_s. a' en b' komen overeen met de positie op de afbeelding en d' met de diepte. Een punt in de wereld wordt dan voorgesteld door:

$$P_w = Q \begin{bmatrix} a' \\ b' \\ c' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x_s}{a_s - 1} & & & \\ & \frac{y_s}{b_s - 1} & & \\ & & \frac{d_{max} - d_{min}}{d_s - 1} & d_{min} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a' \\ b' \\ c' \\ 1 \end{bmatrix}$$

als de ruimte gekwantiseerd is op dispariteit. x_s en y_s stellen de afbeeldingsgrootte voor en d_{min} en d_{max} het interval van toegelaten dispariteitswaarden. Als kwantisatie op diepte wordt gebruikt, is de formule:

$$P_w = Q \begin{bmatrix} a' \\ b' \\ c' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{a_{max} - a_{min}}{a_s - 1} & & & a_{min} \\ & \frac{b_{max} - b_{min}}{b_s - 1} & & b_{min} \\ & & \frac{d_{max} - d_{min}}{d_s - 1} & d_{min} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a' \\ b' \\ c' \\ 1 \end{bmatrix}$$



Figuur 2.15: (a) Een matching space, gekwantiseerd volgens dispariteit (b) Opstelling van een graaf (Roy 1999).

Het bepalen van de diepte wordt gedefinieerd als het maken van een oppervlakte in de matching space met een zo laag mogelijke kost. De kost wordt als volgt gedefinieerd:

$$cost(a', b', d') = \frac{1}{n} \sum_{i=1}^{n} (v(a', b', d')_i - \overline{v(a', b', d')})^2$$

Het aantal camera's is n en $\overline{v(a', b', d')}$ is het gemiddelde van alle waarden voor elke camera. Deze kost veronderstelt dat de oppervlakten Lambertiaans zijn, oftewel dat elk punt dezelfde illuminantiewaarde heeft vanuit elk standpunt. Daarom ziet elk punt er hetzelfde uit op elke afbeelding. Deze kost wordt opgelost door het min-cut/max-flow probleem hierop toe te passen (zie sectie 2.1.2). Dit is dus een globaal algoritme. De verzameling knopen is:

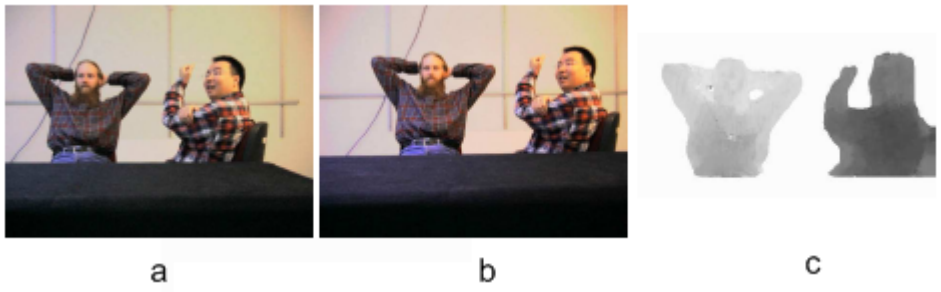
$$V = V^a \cup \{s, t\} = \left\{(a', b', c') | 0 \leq a' \leq a_s, 0 \leq b' \leq b_s, 0 \leq d' \leq d_s + 1\right\} \cup \{s, t\}$$

Hierbij definiëren we s als een knoop verbonden met elk punt van de front (V_{front}) en t verbonden met elk punt van de back (V_{back}). De verbindende bogen noemen we respectievelijk E_{in} en E_{out}. De bogen definiëren een 6-burenrelatie. De bogen tussen buren in de richting a of b noemen we $E_{penalty}$, de bogen in de richting van de diepte E_{label}. Zie figuur 2.15 b. De capaciteiten van de bogen is als volgt gedefinieerd:

$$c(u,v) = \begin{cases} 0 & (u,v) \notin E \\ \infty & (u,v) \in E_{in} \vee (u,v) \in E_{out} \\ cost(u) & (u,v) \in E_{label} \wedge u_d < v_d \\ \infty & (u,v) \in E_{label} \wedge u_d > v_d \\ K & (u,v) \in E_{penalty} \end{cases}$$

K definieert een smoothness term. Hoe hoger K, hoe groter de smoothness. De voorlaatste regel vermijdt dat er twee verschillende punten (a,b,x) en (a,b,y) worden gebruikt, en zo twee dieptes toekennen aan een punt.

Uit deze graaf wordt een minimale cut C_{min} berekend. Meerdere algoritmen zijn getest. De dispariteit kan nu afgeleid worden uit de bogen in de verzameling $C_{min} \cap E_{label}$.



Figuur 2.16: Resultaten van het algoritme (a), (b) Invoerafbeeldingen (c) Dieptemap (Yang & Pollefeys 2003).

Het algoritme in Yang & Pollefeys (2003) maakt gebruik van *squared intensity differences* (SID) als kostfunctie en een lokaal vergelijkingsalgoritme in overeenkomende vensters van aanpasbare grootte voor de kostaggregatie. Het opstellen van de dispariteitsfunctie is een *winner takes all* aanpak, dit wil zeggen dat er een vlak wordt gekozen voor een dieptes, daarop worden de invoerafbeeldingen geprojecteerd en de SID wordt berekend voor elke diepte. De kleinste waarde voor een pixel over al de vlakken is de oplossing. Dit algoritme is ontwikkeld om uitgevoerd te worden op een grafische processor.

Voor het berekenen van de SID wordt een venster beschouwd op het vlak waar de afbeeldingen zijn geprojecteerd. Er worden voor elke pixels verschillende groottes genomen en de

waarden worden opgeteld. Daardoor zal een groot venster worden gebruikt, maar met een groter gewicht in het centrum. Dit heeft als voordeel dat er geen features gemist worden met toch een grote accuraatheid.

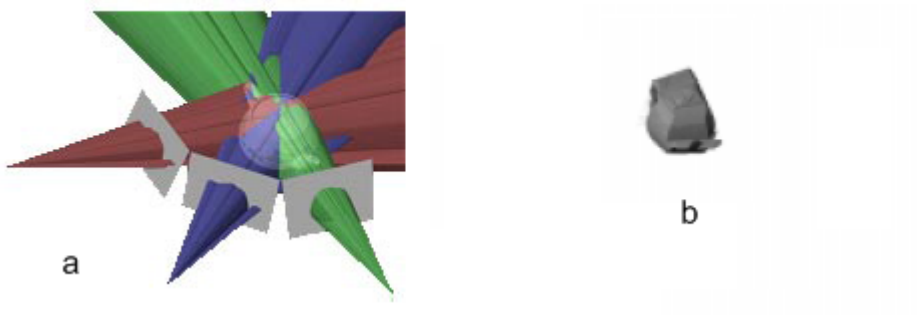
Concreet loopt het algoritme als volgt. Voor elke waarde van d, voor elke pixel (x, y) op projectievlak op afstand d, voor elke venstergrootte, bereken de SID. Tel alle waarden van de vensters op voor een pixel (x, y) op een vlak op afstand d. Kies voor een pixel (x, y) de kleinste som tussen de vlakken op afstand d. De waarde d is de dispariteitswaarde.

Maken van dieptemappen met meedere afbeeldingen zonder minimalisatie

In Fromherz & Bichsel (1995) wordt eerst een vorm afgeleid van de contouren van een object zoals beschreven in Fromherz & Bichsel (1994). Deze contour bevat het volledige object, maar kan verder verfijnd worden. Dit gebeurt met illuminantieinformatie. Het principe is dat een valse match op 2 beelden overeenkomt met 2 verschillende punten op het echte model en deze zullen dan een verschillende illuminantiewaarde hebben. Een juiste match komt overeen met hetzelfde punt op het object voor alle beelden en hebben dus ook dezelfde illuminantiewaarde. Het eindresultaat is een verzameling voxels (waaruit ook een dieptemap afgeleid kan worden). Een grote tekortkoming van dit algoritme is dat het object in kwestie geen spiegelende en doorzichtige oppervlakten mag bevatten, met ander woorden, het oppervlakte moet Lambertiaans zijn (*Definition for word(s) lambertian surface - The Photonics Directory* 2008). Deze punten zullen andere kleuren en illuminantiewaarden hebben voor verschillende camerastandpunten. Bovendien moet de belichting constant blijven.

Een ander algoritme dat een verzameling voxel creeert is beschreven in Seitz & Dyer (1999). Het verschil met Fromherz & Bichsel (1995) is dat dit algoritme van kleuren gebruik maakt in plaats van van illuminantiewaarden. Ook dit algoritme vereist dat de belichting vrij constant blijft en er geen spiegelende en doorzichtige oppervlakten zijn. Hierbij wordt elke voxel opgezocht in de foto's en als de kleuren gelijkaardig zijn, wordt de voxel als onderdeel van het object beschouwd. Dichter gelegen voxels worden eerst behandeld. Op die manier wordt occlusie automatisch behandeld.

Een andere voorstelling van een model is een visual hull. Dit wordt beschreven in Matusik et al. (2000) en Matusik et al. (2001) en is eveneens gerealiseerd in Guisson (2005). Hierbij worden de delen die niet tot het object behoren weggesneden. Er wordt een aantal kegels gevormd met de punten in de camera's. Als de straal van de camera naar de scene tot de kegel behoort, dan snijdt deze straal het object. De *hull* wordt gevormd door de intersecties van deze kegels. Bovendien worden er silhouetten gecreeerd. Dit zijn de snijpunten van de stralen met het projectievlak van de camera, oftewel een deel van de invoerafbeeldingen. Alle berekeningen gebeuren in de image space van de invoerafbeeldingen. Daarom zullen er geen

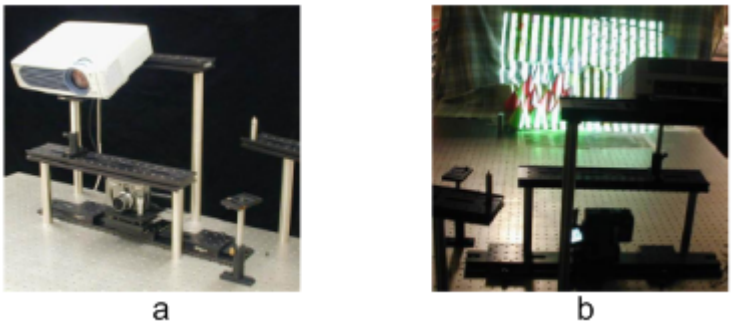


Figuur 2.17: (a) Het snijden van de hull (b) Het resultaat (Matusik et al. 2000)

artefacten van resampling requantificatie zijn.

Het algoritme gaat als volgt: eerst worden de silhouetten gecreeerd. Vervolgens wordt elke *viewing ray* in de invoerafbeeldingen geprojecteerd. Deze projectie wordt gesneden met het silhouette en het resultaat wordt terug geprojecteerd in 3D, waarna deze worden gesneden met de projecties van de andere invoerafbeeldingen. Deze berekeningen kunnen incrementeel gebeuren door het gebruik van epipolaire geometrie. Een andere versnelling is door de silhouetten om te zetten in een verzameling randen. Omdat er randen worden gesneden in plaats van pixels, kan zo het algoritme versneld worden. Het resultaat is een visual hull van het object. Als extra toevoeging kan een texture geplaatst worden over de hull.

Er zijn meerdere gelijkaardige algoritmen die vooraf zijn gegaan aan dit algoritme. Zie bijvoorbeeld Potmesil (1987) en Moezzi et al. (1996). Een introductie van visual hulls is gegeven in Laurentini (1994).



Figuur 2.18: (a) Voorbeeld van een opstelling (b) Een projectie (Scharstein et al. 2003).

In Scharstein et al. (2003) wordt op het te reconstrueren object een lichtpatroon geprojecteerd. Het object met de patronen wordt door twee camera's opgenomen (zie figuur 2.18).

Door het gebruik van verschillende patronen achter elkaar krijgt elke positie op het object een unieke lichtcode. In de paper wordt een binair patroon gebruikt, dus zijn er $log_2(n)$ opnames nodig voor een unieke code per locatie (met n als aantal locaties). Er zijn ook experimenten gedaan met sinusgolven. Door het vergelijken van de codes op de twee beelden kunnen de overeenkomende punten worden bepaald. Met normale epipolaire geometrie kan vervolgens de afstand tot de camera's bepaald worden.

Maken van dieptemappen met meerdere afbeeldingen vanuit één standpunt

In Woodham (1989) worden verschillende opnames gemaakt van een object met verschillende lichtinvallen, terwijl de camera op dezelfde plaats blijft staan. Door het vergelijken van de belichting van de verschillende opnames, kan de gradient van elk punt bepaald worden. Hieruit kan een 3D-model worden gevormd.

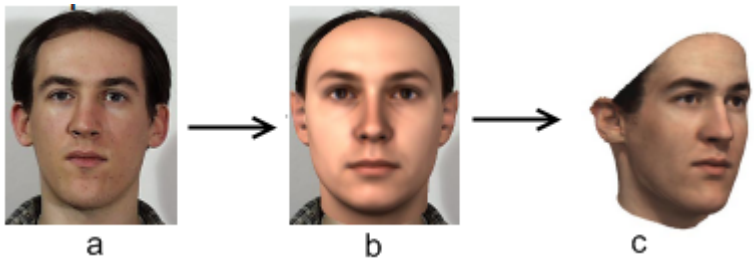
Er wordt aangetoond dat de intensiteiten van een afbeelding enkel afhangen van de gradient van de oppervlakte, als er gebruik wordt gemaakt van een orthogonale projectie (die een perspectieve projectie benadert op korte afstand), van een vaste waarnemer en van een vaste lichtbron met constante radiantie. De gradient wordt verder voorgesteld door twee coördinaten, p en q, waarbij $p = \frac{\delta f(x,y)}{\delta x}$ en $q = \frac{\delta f(x,y)}{\delta y}$. $f(x, y)$ geeft de afstand van het projectievlak tot het object weer. De intensiteit van een punt wordt weergegeven door $R(p, q) = I(x, y)$. Dit noemt men de *reflectance map*. Deze kan, afhankelijk van het model, in een formule worden weerggegeven. Perfecte lambertiaanse oppervlakten kunnen worden weergegeven door: $R(p, q) = \frac{\upsilon(1+pp_s+qq_s)}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}$. $(p_s, q_s, -1)$ wijst in de richting van de lichtbron.

Het probleem bestaat er nu in p en q te bepalen voor elk punt. Dit kan door twee (in het geval van een lineaire R) of meer afbeeldingen te nemen met verschillend invallend licht, en dus ook een verschillende R. Dan zijn er een aantal vergelijkingen van de vorm $R(p, q) = I(x, y)$, waaruit p en q berekend kunnen worden.

2.3.3 Maken van dieptemappen met domeinkennis

Een andere aanpak dan tot nu werd gebruikt, is het gebruiken van kennis van de modellen. Als voorbeeld nemen we het algoritme beschreven in Blanz et al. (1999), dat geschikt is voor het maken van modellen van gezichten uit één foto. Er wordt een database gebruikt van een groot aantal lasergescande gezichten, waar een standaardgezichtsmodel van wordt berekend. Hierop kan een foto worden gematched, waarbij het standaardmodel vervormd wordt zodat dit overeenkomt met de foto. Bovendien is het mogelijk hierop textuur van de foto aan te brengen.

Een andere techniek, zoals beschreven in Hoiem et al. (2005), werkt met scenes bestaande



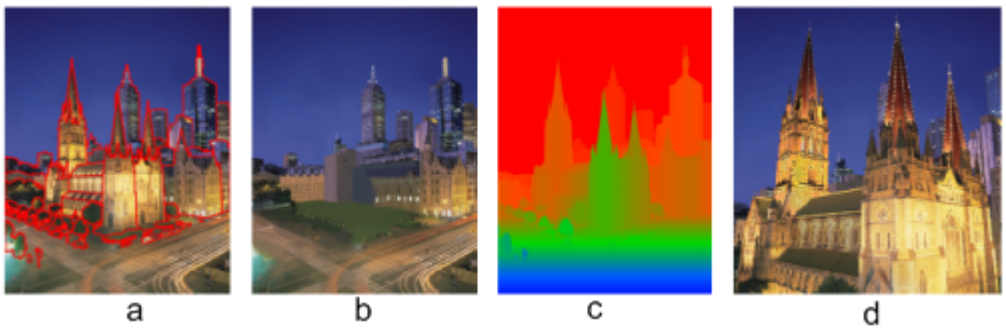
Figuur 2.19: (a) Invoerafbeelding (b) Handmatig plaatsen van het standaardmodel over het gezicht (c) Resultaat berekening van het model (Blanz et al. 1999).

uit de grond, de lucht en objecten die recht omhoog staan. Hiervan kan dan volledig automatisch een *pop-up* worden gemaakt. Elk object dat ruw overeenkomt met een geometrische basisvorm krijgt een diepte. De techniek bestaat eruit grote, gelijkaardige vlakken van pixels samen te nemen tot één gebied en deze een label grond, lucht of object te geven. Het label kan worden bepaald uit features, zoals kleur (blauw (lucht) of groen(gras)), textuur, locatie (lucht bovenaan, grond onderaan), verdwijnpunten, ... Om te bepalen welk label een vlak krijgt, gegeven de features, wordt het algoritme vooraf getrained met afbeeldingen van lucht of grond. Door de features van de training te vergelijken met de invoer, kan er een label worden toegekend. Als de labels bekend zijn, kan worden berekend waar de onderkant van de objecten zich bevinden op de grond. Hierdoor kan elke 3D-positie worden bepaald van de objecten in de scene. Dit veronderstelt dat elk object op de grond is geplaatst. Het resultaat kan eventueel worden omgezet naar een dieptemap.

Figuur 2.20: Voorbeeld van het pop-upalgoritme. De eerste afbeelding is de invoer. De andere twee geven de scene weer vanuit een ander standpunt, waarbij de 3D-eigenschappen zijn berekend uit de originele foto. (Hoiem et al. 2005).

In figuur 2.20 is een voorbeeld gegeven van het algoritme. De eerste afbeelding is de invoer. De drie vereiste objecten, grond, lucht, en rechtopstaande objecten, zijn aanwezig. De objecten staan op de grond en komen loodrecht naar boven. Het resultaat is gedemonstreerd door de scene vanuit een ander oogpunt te bekijken.

2.4 Dieptemappen maken uit één afbeelding met user input



Figuur 2.21: (a) Indeling in lagen (b) aanduiden van verborgen gebieden (c) resultaat van de dieptebepaling (d) vanuit een ander standpunt bekeken (Oh et al. 2007).

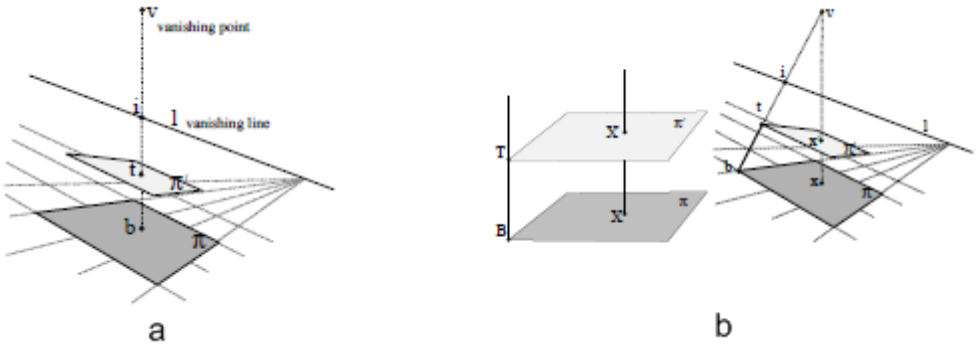
In Oh et al. (2007) wordt een techniek voorgesteld die één afbeelding gebruikt. De afbeelding wordt vooraf handmatig ingedeeld in verschillende lagen. Elke laag bevat een aantal objecten, bijvoorbeeld een stad waar elk gebouw een laag is. In elke laag wordt de diepte bijgehouden, maar ook α-waarden, textuurinformatie, enzovoort kan worden bijgehouden. Bovendien kan de laag vanuit een willekeurig standpunt worden bekeken. Dit is afhankelijk van de berekende diepte, verborgen gebieden, ...

Voor het bepalen van diepte kan de gebruiker de diepte tekenen, zoals er kleuren getekend worden. Dit kan vanuit elk willekeurig standpunt en kan vanuit meerdere standpunten tegelijkertijd bekeken worden. Dit kan absolute diepte zijn of relatieve diepte (een waarde optellen of aftrekken van de huidige diepte). Andere tools zijn het plaatsen van een grondvlak, de laagste pixels van een object aanduiden, geometrische objecten aanduiden en gezichten aanduiden. Als laatste is er nog een selectiegereedschap beschikbaar die het te bewerken gebied beperkt. De resultaten geven lange tijden weer voor het maken van de dieptemappen (13 uur). Vooral het opdelen in lagen is tijdrovend. De paper beschijft verder als toepassing een *clone brush* en een *texture-illuminance decoupling filter*.

Een ander methode voor het tekenen van diepte wordt voorgesteld in Kang (1998). Het is mogelijk de afbeelding in te delen in regio's (voor regionale operaties) en diepte toe te kennen (lokaal of aan een regio). Diepte wordt toegekend door brushes. Buiten weergave op de afbeelding zelf, worden ook de bovenkant, onderkant, linkerkant en rechterkant weergegeven. De gaten die gecreeerd worden kunnen worden opgevuld met textuurinformatie uit de afbeelding. Er worden geen berekeningen uitgevoerd voor automatische dieptebepalingen.

Criminisi et al. (2000) stelt een methode voor voor het maken van dieptemappen van afbeel-

dingen met vlakken en parallelle lijnen. Er wordt uitgegaan van perspectieve camera's. De ruimte wordt voorzien van X-, Y- en Z-assen. Er wordt een referentievlak (vastgelegd over de X- en Y-assen) vastgelegd dat parallel loopt met een *vanishing line* l en eveneens een verdwijnpunt (v) van een andere richting (de Z-as), niet evenwijdig aan het referentievlak. De afbeelding heeft gewone coördinaten (x, y). Een punt U wordt afgebeeld op de afbeelding op punt u door de formule $x = PX = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} X$. x en X zijn homogene kolomvectoren, respectievelijk $(x, y, w)^T$ en $(X, Y, Z, W)^T$. Nu definiëren we de verdwijnpunten voor de X-, Y- en Z-assen als v_x, v_y en v. De verdwijnpunten v_x en v_y liggen op de eerder gekozen *vanishing line* l. Nu is $p_1 = v_x$, $p_2 = v_y$ en $p_3 = v$. p_4 is de projectie van de oorsprong (gelegen op het referentievlak) en ligt niet op l. Nu kunnen we P definiëren als $\begin{bmatrix} v_x & v_y & \alpha v & o \end{bmatrix}$.



Figuur 2.22: (a) Het meten van afstanden tussen twee vlakken (b) het meten van afstanden op een vlak (Criminisi et al. 2000).

Nu kunnen we de afstand tussen twee vlakken meten in de richting van de Z-as (zie figuur 2.22a). De vlakken worden vastgelegd door twee punten t en b op de afbeelding en B en T in de scene. Het punt B ligt op het referentievlak en dus is $B = (X, Y, 0, 1)$ en $T = (X, Y, Z, 1)$. Dan is $b = PB = \rho(Xp_1 + Yp_2 + p_4)$ en $t = PT = \mu(Xp_1 + Yp_2 + Zp_4 + p_4)$, met ρ en μ schaleringsfactoren. Daaruit volgt:

$$\alpha Z = \frac{-\|b \times t\|}{(o.b)\,\|v \times t\|}$$

Hieruit kan een α worden berekend als een referentiehoogte bekend is voor een punt in de afbeelding. Daarna kan elke hoogte berekend worden.

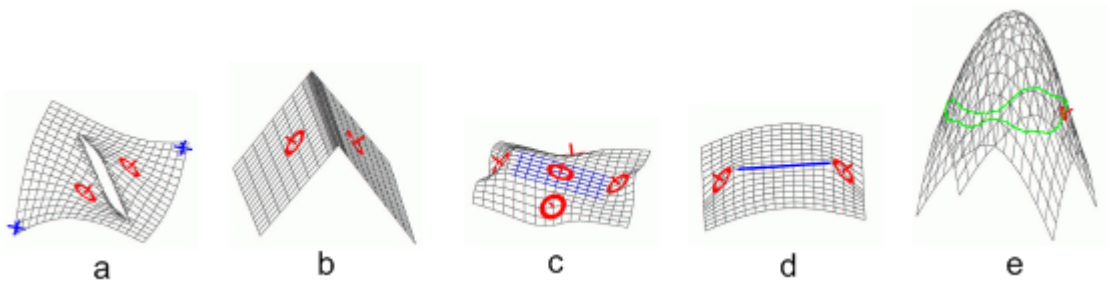
Ook de afstand op een vlak kan gemeten worden (zie figuur 2.22). We weten dat elk vlak parallel aan het referentievlak π dezelfde verdwijnpunten hebben. Als we het coördinatensysteem verplaatsen over de richting van de Z-as op een nieuw vlak π', dan is de nieuwe projectiematrix $\begin{bmatrix} v_x & v_y & \alpha v & \alpha Z v + o \end{bmatrix}$ met Z de afstand tot de vlakken. Hieruit kunnen we een

homologiematrix $\widetilde{H} = I + \alpha Zvo^T$ berekenen die punten van π' transformeert naar π. Hier kunnen affiene metingen gedaan worden, bijvoorbeeld de verhouding tussen twee lijnen. Als een referentieafstand bekend is, kunnen lengtes gemeten worden.

In Zhang et al. (2002) wordt een afbeelding voorgesteld als een functie $f(x, y)$, de dieptemap. De bedoeling is deze functie zo glad mgelijk te maken terwijl deze voldoet aan de door de gebruiker opgelegde beperkingen. Het gladmaken gebeurt door de volgende functie te minimaliseren:

$$\begin{aligned} Q_0(g) = \frac{1}{2d^2} sum_{i,j}(&\alpha_{i,j}(g_{i+1,j} - 2g_{i,j} + g_{i-1,j})^2 \\ &+ 2\beta_{i,j}(g_{i+1,j+1} - g_{i,j+1} - g_{i+1,j} + g_{i,j})^2 \\ &+ \gamma_{i,j}(g_{i,j+1} - 2g_{i,j} + g_{i,j-1})^2) \end{aligned}$$

De functie $g_{i,j} = f(id, jd)$ is een sampling van de functie f met d de afstand tussen twee samples (hetzelfde voor x en y). De gewichten α, β en γ maken het mogelijk discontinuïteiten toe te voegen.



Figuur 2.23: Constraints die de gebruiker kan opleggen. (a) Discontinuïteit in de diepte (b) Discontinuïteit in de normalen (c) Definieren van vlakken (d, e) Minimaliseren ven de curve (Zhang et al. 2002).

De beperkingen die de gebruiker kan opleggen zijn:

- Puntbeperkingen: Zet de diepte of normaal op één punt in de afbeelding.
- Diepte discontinuïteiten: Een curve die aangeeft waar de functie f niet continu is (figuur 2.23a).
- Vouwen: Discontinuïteiten in de normalen, maar niet in de oppervlakten (figuur 2.23b).
- Vlakken (figuur 2.23c).

- Curve controle: Geeft aan hoe vlak of steil een bepaald gebied moet zijn. Dit wordt getekend met een curve. De vlakheid van de curve wordt dan gemaximaliseerd (figuur 2.23d, e).

Nu wordt de functie $Q_0(g)$ geminimaliseerd onder de ingegeven constraints. Dit is een lineair beperkt kwadratische optimatisatie (minimalisatie). $Q_O(g)$ kan dus worden voorgesteld als $g^T Hg$, waar H de Hessiaanse matrix is (zie sectie A.1). Een aantal constraints (punt en vlakken) worden vastgelegd in lineaire vergelijkingen: $Ag = B$; de andere beperkingen definiëren de gewichten α, β en γ in $Q_0(g)$. Nu kunnen deze twee vergelijkingen ($g_0(g)$ en $Ag = B$) worden samengevoegd tot:

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} g \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}$$

Deze vergelijking wordt opgelost met de minimum residue methode (Press et al. 2002).

De techniek *Tour into the Picture*, zoals beschreven in Horry et al. (1997), maakt het mogelijk animaties te maken uit één afbeelding. Dit wordt gedaan door een simpel 3D model te maken met behulp van een *spinnenweb*. Het model is meer een verzameling voorgrondpolygonen die op vijf achtergrondpolygonen geplaatst wordt; er is geen gedetailleerd model of diepteinformatie. De gebruiker moet verdwijnpunten aangeven, de voorgrond- van de achtergrondobjecten scheiden en de objecten opdelen in polygonen. Het spinnenweb zorgt voor deze drie operaties.

Het proces loopt als volgt: Er wordt een masker gemaakt voor de voorgrond en de achtergrond te onderscheiden en de lege plekken in de achtergrond worden opgevuld. Vervolgens wordt het verdwijnpunt en een rechthoek ingegeven (de *achterkant* van de scene) door de gebruiker. Dan wordt de achtergrond gemodelleerd als vijf polygonen, waarbij de gedefinieerde rechthoek de achterkant van een balk is en van daaruit de andere zijden worden bepaald. Voor het modelleren van de voorgrondobjecten wordt er een vierhoek bepaald rond het object (in 2D) en dan wordt deze vierhoek loodrecht gezet op één van de vier zijvlakken van de achtergrond. Zo wordt de diepte van deze vierhoek en dus het object bepaald. Verder wordt beschreven hoe dit model kan gebruikt worden voor het animeren en het renderen van nieuwe beelden.

In Shum et al. (1998) wordt een reeks panoramafoto's omgezet in een 3D-model. Deze foto's zijn genomen vanuit één camerastandpunt. Het model wordt gevormd in een aantal stappen:

1. **Cameraoriëntatie bepalen.** De gebruiker kan een aantal lijnen aangeven waarvoor de richting bekend is, bijvoorbeeld verticale randen van muren zijn parallel aan de Z-as van het coördinatenstelsel. Als twee verzamelingen parallelle lijnen gegeven zijn, kan de cameraoriëntatie bepaald worden.

2. **Normalen van vlakken en richting van lijnen bepalen.** De normalen van vlakken kunnen bepaald worden door het tekenen van een parallellogram op het vlak. Omdat de cameraoriëntatie bekend is, kan de richting van lijnen berekend worden.

3. **Camera translatie bepalen.** Kan bepaald worden als twee of meer punten worden bepaald.

4. **Afstand van vlakken en positie van 3D-punten bepalen.** De eerder ingegeven en berekende informatie wordt ingedeeld in harde en zachte constraints. Bekende punten en lijnen, evenwijdige vlakken en lengtes/elementen berekend uit bekende gegevens zijn harde constraints. Zachte constraints zijn lengtes/elementen uit geschatte gegevens. Zo is een punt op een vlak een harde constraint als de normaal van het vlak gegeven is; als deze berekend is, is dit een zachte constraint. De oplossing van het probleem is het oplossen van $Ax = b$ (zachte constraints), waarbij $Cx = q$ (de harde constraints) bewaard wordt. Dit wordt gedaan door een QR factorizatie (zie sectie A.2.

In Gerrits (2008) wordt een techniek beschreven die gebruik maakt van annotaties op de afbeelding voor het genereren van dieptemappen. De interactietechniek is gelijkaardig aan de technieken zoals beschreven in sectie 2.2. De annotaties kunnen normalen of absolute dieptes zijn. De normalen worden voor de optimalisatie omgezet in gradienten. In de optimalisatiefase wordt een energiefunctie geminimaliseerd. Deze energiefunctie bevat een term die rekening houdt met de aangegeven constraints en een term die een egaliteit in de dieptemap invoert. De gebruikte energiefunctie is dezelfde functie die werd gebruikt in een aantal toepassingen in sectie 2.2:

$$E(f(p)) = \sum_{p \in P} (E_{data}(p) + \lambda E_{smooth}(p))$$

Hierbij wordt een functie f berekend die een diepte toekent aan elke pixel. E_{data} is de dataterm die de aangegeven constraints gebruikt. E_{smooth} gebruikt de gradienten aanwezig in de afbeelding. λ geeft de belangrijkheid van elke term aan.

De dataterm is gegeven door:

$$\begin{aligned} E_{data}(p) &= A(p) + \varphi G(p) \\ A(p) &= w_a(p)(f(p) - k(p))^2 \\ G(p) &= w_g(p) \left[\frac{\delta f(p)}{\delta u} - g_u(p) \right]^2 + w_g(p) \left[\frac{\delta f(p)}{\delta v} - g_v(p) \right]^2 \end{aligned}$$

De term $A(p)$ zorgt ervoor dat de functie dicht bij de harde constraints van de gebruiker blijft. $k(p)$ geeft de diepte op dat punt aangegven door de gebruiker en w_a is niet nul voor

aangeduide gebieden. De term $G(p)$ verwerkt de gradient input. g_u en g_v zijn de aangegeven constraints.

De *smoothness term* v is gegeven door:

$$E_{smooth}(p) = (f(p) - \frac{\sum_{q \in N_4(p)} h(p,q) f(q)}{\sum_{q \in N_4(p)} h(p,q)})^2$$
$$h(p,q) = e^{\frac{(L_p - L_q)^2}{2\sigma^2}}$$

Deze term zal ervoor zorgen dat de Laplaciaan van de functie f groot is als de Laplaciaan van de afbeelding groot is en omgekeerd. Indien de gradient gebruikt zou worden, zou een functie berekend worden die constante diepte verkiest, nu wordt constante helling verkozen.

Voor de minimalisatie is gebruik gemaakt van een simpele hierarchische oplosser. De oplossing wordt verfijnd via interpolatie. Op elk niveau wordt symmetrische LQ gebruikt. Zie ook sectie 2.1.1.

2.5 Conclusie

Er bestaan veel algoritmen voor het oplossen van het dieptemapprobleem, allen met hun sterktes en hun zwaktes. Op het vlak van stereovisie zijn er veel technieken beschikbaar, globaal en lokaal, met veel oplossingsmethoden (bijvoorbeeld minimalisatie). Ook qua technieken die één standpunt kunnen gebruiken, zijn er veel vooruitgangen geboekt. Helaas maken de meeste van deze algoritmen gebruik van een speciale opstelling of worden er eisen opgelegd aan de foto's. De methoden voor het maken van dieptemappen uit één foto zonder speciale eisen zijn nog beperkt. Veel van deze technieken vereisen veel en preciese gebruikersinformatie, zijn niet erg nauwkeurig of zijn niet krachtig genoeg voor voldoende preciese input. Het toekomstig werk zal zich dus richten op het verbeteren, vereenvoudigen en uitbreiden van de beschreven technieken. Hier kunnen dezelfde methoden gebruikt worden als uit de toepassingen.

Hoofdstuk 3

Implementatie

3.1 Overzicht

Het probleem van dieptemappen maken wordt beschouwd als het oplossen van een minimalisatieprobleem met strokes als invoer. De gebruiker kan via strokes aangeven wat de diepte of de oriëntatie moet zijn op deze plaats. Het is mogelijk meerdere dieptes in te geven in één segment in de afbeelding. Dit is niet voorzien in andere stroke-based technieken, wat deze methode uniek maakt.

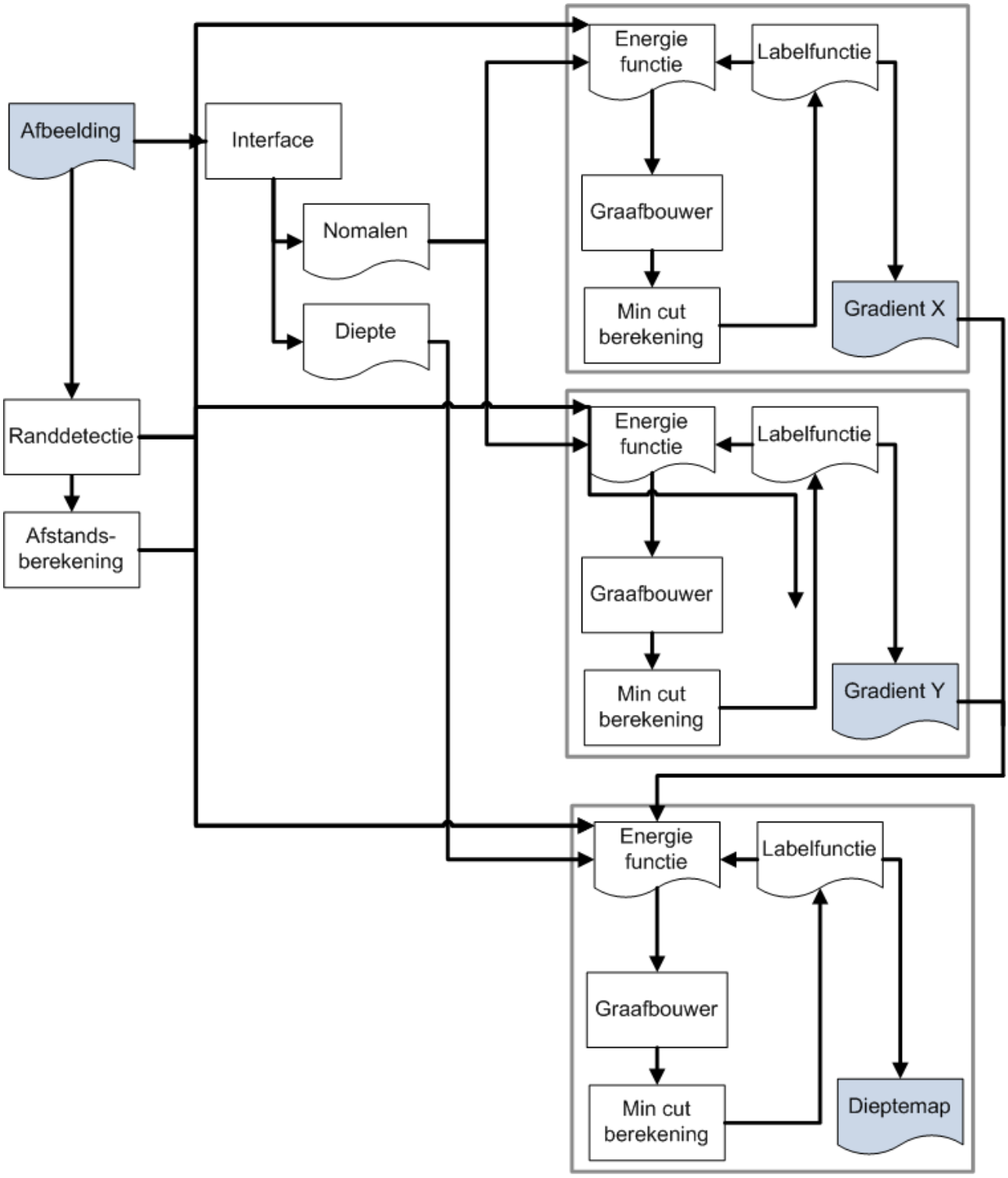
Als de strokes zijn ingegeven, kan er een energiefunctie worden geconstrueerd. Om deze te construeren is er eerst extra informatie nodig. Deze wordt berekend in de voorverwerking. De extra informatie bevat de randen in de afbeelding en de afstand tot de dichtsbijzijnde stroke. De randen worden gebruikt om een kans te berekenen dat er een segment in de afbeelding is. Door geen absolute randen te gebruiken worden effecten van gaten vermeden en kunnen ook vagere randen gebruikt worden. De afstand tot de dichtsbijzijnde stroke wordt gebruikt om meerdere dieptes te kunnen gebruiken in één segment. Hierdoor kunnen hellende vlakken worden aangeduid. Deze worden voorgesteld door kleurovergangen. Deze overgangen worden dan verder gepropageerd volgens de richting van deze overgang.

Met behulp van de strokes en de informatie berekend in de voorverwerking wordt er een energiefunctie gedefinieerd die aangeeft hoe fout de huidige dieptemap is. Het doel is dan deze energie te minimaliseren en op die manier de optimale oplossing te bekomen. Vermits minimalisatie een complex probleem is, zal een benadering worden berekend; in deze implementatie is de afwijking maximaal 2 maal de ideale oplossing (Boykov et al. 2001). De minimalisatie wordt uitgevoerd via graph cuts.

Het minimaliseren gebeurt door iteratief twee labels (dieptes) a en b te kiezen en alle pixels te selecteren die al één van deze 2 labels hebben. Met een min cut wordt berekend welke pixels

eventueel van label wisselen (a wordt b, b wordt a, of het label blijft behouden). Dit gebeurt door de pixels als knopen te beschouwen en de bogen een kost te geven die afhankelijk is van de energie. Zo wordt de labeling aangepast als de totale energie van de labeling minder is dan de vorige. Dit blijft doorgaan totdat er geen wissels meer gedaan kunnen worden die een lagere energie tot gevolg hebben. Dan heeft de laatste labeling de laagste energie en is dus de optimale oplossing. Omdat er altijd twee dieptes worden gekozen, zal de dieptemap discreet zijn. Er is slechts een eindige, maar zelf te bepalen precisie mogelijk.

De minimalisatie wordt drie keer uitgevoerd, twee maal voor de oriëntatie en één maal voor de diepte. De oriëntaties worden apart berekend voor de horizontale en de verticale richting. De diepte maakt gebruik van de oriëntaties berekend in de vorige stappen. Een overzicht van het volledige proces is gegeven in figuur 3.1

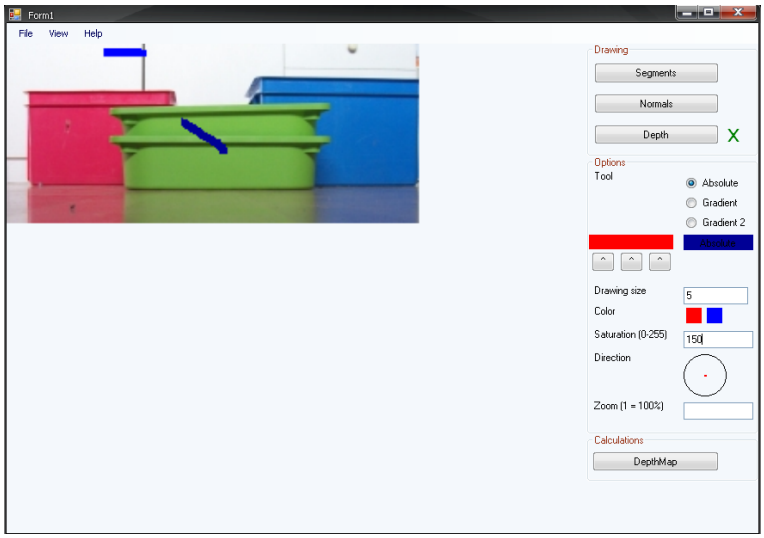


Figuur 3.1: Overzicht van de implementatie

3.2 Invoer

Als het programma gestart wordt, kan de gebruiker een afbeelding inladen. Op deze afbeelding kunnen drie soorten strokes getekend worden: Handmatige randen, oriëntatie en vaste diepten. Deze kunnen getekend worden zoals elk standaard tekenprogramma. Er moet aangegeven worden welke soort strokes er momenteel getekend worden.

Diepte De diepte kan getekend worden met een waarde tussen blauw en zwart (zie figuur 3.2). De diepte kan ook aangegeven worden als een kleurovergang (een gradiënt) met twee of drie kleuren (zie figuur 3.3).

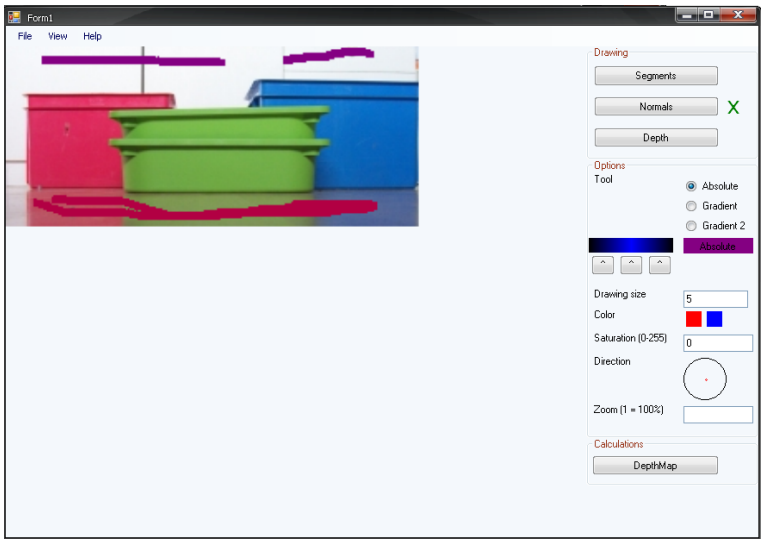


Figuur 3.2: Invoer van absolute diepte.

Oriëntatie De oriëntatie kan ingegeven worden met twee kleuren. Rood is de oriëntatie in de horizontale richting en blauw in de verticale richting. Om deze invoer te vereenvoudigen is er een oriëntatiekiezer gemaakt in de vorm van een cirkel. Deze stelt een bol voor. Het punt in het midden betekent dat de richting recht uit het scherm komt. Hoe verder naar de rand, hoe feller de helling naar die kant. Op de rand van de cirkel is de richting die met het scherm meegaat. Op deze manier is het intuïtiever om een richting te coderen met twee kleuren. Een voorbeeld is gegeven in figuur 3.4.

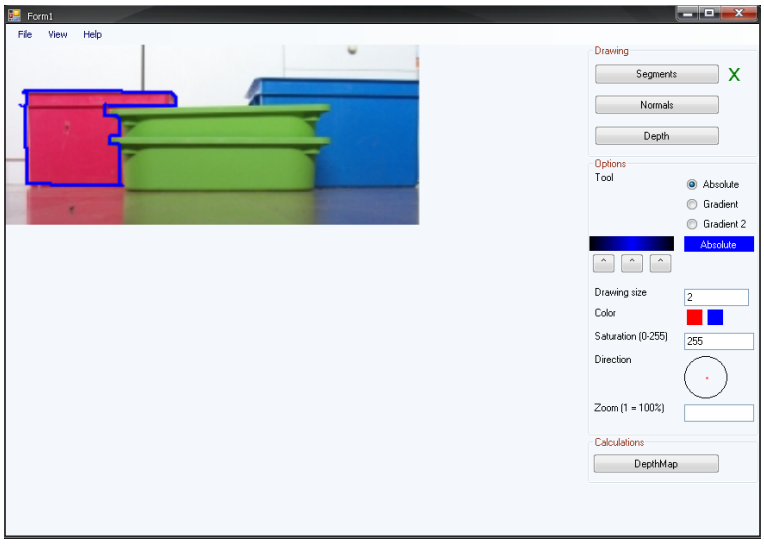
Randen Het is ook mogelijk om randen aan te duiden die de edge detector niet of onvoldoende kan vinden. Randen kunnen in eender welke kleur worden getekend. Deze invoer wordt beschouwd als 100% rand. Zie bijvoorbeeld figuur 3.5.

Figuur 3.3: Invoer van oriëntatie via gradiënten.



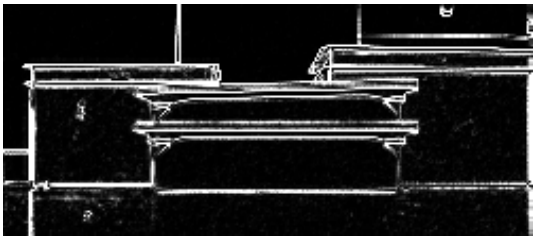
Figuur 3.4: Invoer van oriëntatie via kleur als richting.

Starten Als de nodige strokes zijn gemaakt, volstaat het op DepthMap te klikken. De voortgang en een log worden uitgeprint in de console. Na afloop van het programma worden de berekende gegevens getoond, waaronder de dieptemap, de belangrijke tussenresultaten, de randen, de afstanden, ... Figuur 3.6 geeft een voorbeeld van een typische uitvoer. De betekenis van de tussenresultaten wordt uitgelegd in sectie 3.3.

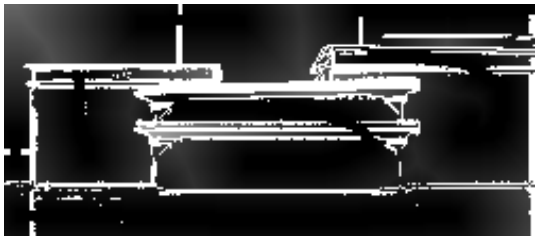


Figuur 3.5: Invoer van handmatige randen in de afbeelding.

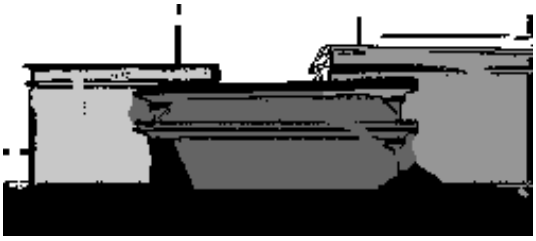
(a) Invoer



(b) Randen



(c) Afstand tot de dichtsbijzijnde stroke



(d) Label van de dichtsbijzijnde stroke

(e) Correctheid van afstanden (zwart=niet correct, grijs=waarschijnlijk, wit=correct)

(f) Uitvoer

Figuur 3.6: Uitvoer van het algoritme.

3.3 Voorverwerking

De energiefunctie die wordt gebruikt, heeft nog extra informatie nodig die kan berekend worden zonder hulp van de gebruiker. Deze informatie bestaat uit de afstand tot de dichtsbijzijnde stroke en de kans dat er een rand is in de afbeelding.

3.3.1 Randdetectie

In deze stap wordt er voor elke pixel berekend hoe groot de kans is dat er op deze plaats in de afbeelding een rand ligt. Op deze waarden wordt vervolgens een drempelwaarde toegepast, zodat extreme waarden afgevlakt worden.

Kans op een rand

Voor het berekenen van een rand wordt er gebruik gemaakt van een algoritme gebaseerd op Di Zenzo (1986).
Eerst berekenen we de discrete afgeleide van de afbeelding I in richting x en y:

$$\begin{aligned}\frac{dR}{dx} &= I_R[x+1][y] - I_R[x][y] \\ \frac{dG}{dx} &= I_G[x+1][y] - I_G[x][y] \\ \frac{dB}{dx} &= I_B[x+1][y] - I_B[x][y]\end{aligned}$$

We definiëren g_x en g_y als volgt:

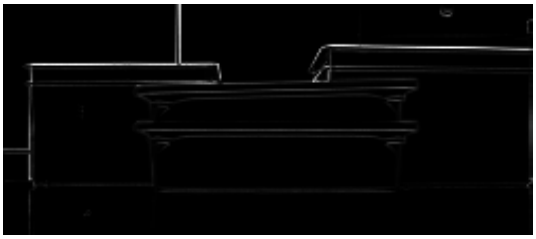
$$\begin{aligned}g_x &= \left|\frac{dR}{dx}\right|^2 + \left|\frac{dG}{dx}\right|^2 + \left|\frac{dB}{dx}\right|^2 \\ g_y &= \left|\frac{dR}{dy}\right|^2 + \left|\frac{dG}{dy}\right|^2 + \left|\frac{dB}{dy}\right|^2\end{aligned}$$

De grootte van de gradiënt of de grootste spatiale stijging in de afbeelding is dan gegeven door:

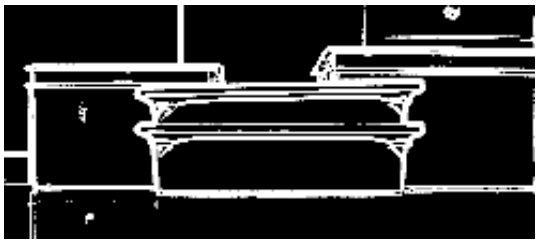
$$S = \sqrt{g_x^2 + g_y^2}$$

Hoe groter S, hoe meer waarschijnlijk dit een rand is. Deze grootte wordt berekend voor elk punt van de afbeelding. Zie bijvoorbeeld figuur 3.7(a). Op deze afbeelding wordt het algoritme toegepast met als resultaat figuur 3.7(b).
Nu worden deze waarden gewijzigd door een vergelijking met een drempelwaarde.

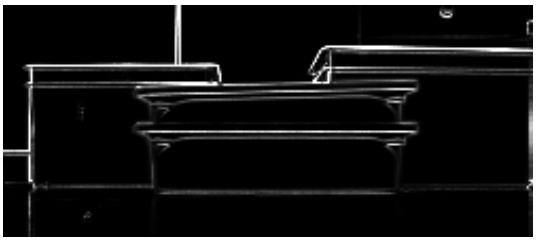
(a) Invoerafbeelding voor randdetectie.



(b) Resultaat na het berekenen van de randen.



(c) Resultaat na het berekenen van de randen (met een logaritmische schaal).



(d) Resultaat na het toepassen van de drempelwaarde.

Figuur 3.7: Stappen in de randdetectie.

Adaptieve drempelwaardeberekening

Omdat het niet zeker is welke drempelwaarde correct is, wordt deze adaptief bepaald. Er worden vier methoden getest:

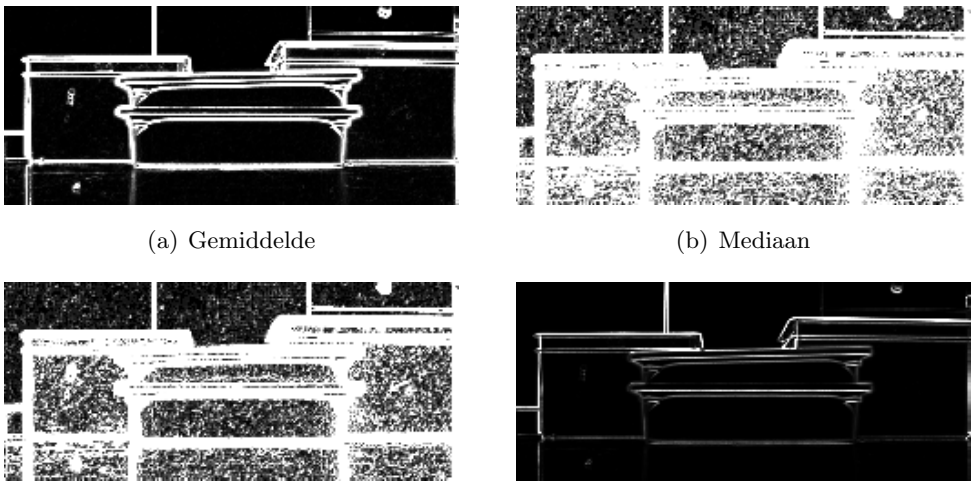
1. Gemiddelde
2. Mediaan
3. Twee keer de mediaan
4. Twee gemiddelden

Het algoritme met twee gemiddelden is afgeleid van een algoritme in Shapiro & Stockman (2001) en werkt als volgt:

1. Kies een initiële drempelwaarde D.
2. Bereken het gemiddelde voor alle waarden onder de drempelwaarde (μ_B) en het gemiddelde voor alle waarden boven de drempelwaarde (μ_F)
3. Bereken het gemiddelde van μ_B en μ_F. Als deze waarde verschillend is van D, herhaal de vorige stap met $D = \frac{(\mu_B+\mu_F)}{2}$.

In dit algoritme worden twee gemiddelden bijgehouden, eentje voor de pixels die een rand zijn volgens de huidige drempelwaarde en eentje voor de pixels die dat niet zijn. Door de drempelwaarde te berekenen vanuit deze twee gemiddelden, is er geen vertekend beeld als de randen extreem hoge waarden hebben. Dit is het geval voor het gewone gemiddelde. Hierbij zullen enkele hoge waarden de drempelwaarde sterk naar boven halen, zodat teveel pixels ten onrechte als rand worden beschouwd. Dit is geïllustreerd in figuur 3.8(a). De randen zijn dik en er zijn te veel segmenten die ten onrechte als segment worden beschouwd.

De mediaan of dubbele mediaan berekenen is eveneens geen oplossing, omdat bij zeer uiteenlopende waarden de helft als rand wordt beschouwd. Dit is meestal geen weerspiegeling van de werkelijke randen. Om deze twee problemen op te lossen, worden er twee gemiddelden gebruikt. Deze techniek heeft goede resultaten, zoals gedemonstreerd in figuur 3.8(d).



(a) Gemiddelde (b) Mediaan

(c) 2*mediaan (d) Twee gemiddelden

Figuur 3.8: Verschillende methoden om de drempelwaarde van randen te bepalen.

Als de drempelwaarde bepaald is, wordt deze toegepast op alle waarden uit de vorige stap. Zo wordt de volgende functie gevormd:

$$Edge(x,y) = \begin{cases} 1 & S(x,y) > D \\ \frac{S(x,y)}{D} & S(x,y) \leq D \end{cases}$$

Deze functie zorgt ervoor dat waarden die boven de drempelwaarde liggen zeker als rand worden gezien. Waarden onder de drempelwaarde worden geschaald tussen 0 en 1, waarbij 0 zeker geen rand is. Deze waarden worden rechtstreeks in de energiefunctie gebruikt, zodat de energieminimalisatie ook rekening houdt met wazige randen. Plaatsen waar geen zekerheid is of er een rand is of niet, krijgen een tussenliggende energie zodat er rekening kan gehouden

worden met andere delen van de afbeelding. Omdat randdetectie niet altijd even goed verloopt (bijvoorbeeld een doos met een egale kleur), kan de gebruiker zelf aanduiden waar er zich randen bevinden. Dit bevordert de correctheid van het algoritme.

Zie bijvoorbeeld figuur 3.8(d). De randen van de dozen zijn duidelijk zichtbaar, maar de randen tussen de dozen zelf kunnen fouten geven. Bovendien weerspeigelt de grond een beetje, waardoor ook hier de randen vager zijn. Daarom is het mogelijk deze randen extra aan te duiden om fouten te voorkomen. Merk op dat er valse positieven zichtbaar zijn. Dit zal geen probleem vormen bij het minimaliseren, omdat er geen zones worden afgebakend en de propagatie als het ware rond deze valse randen gaat.

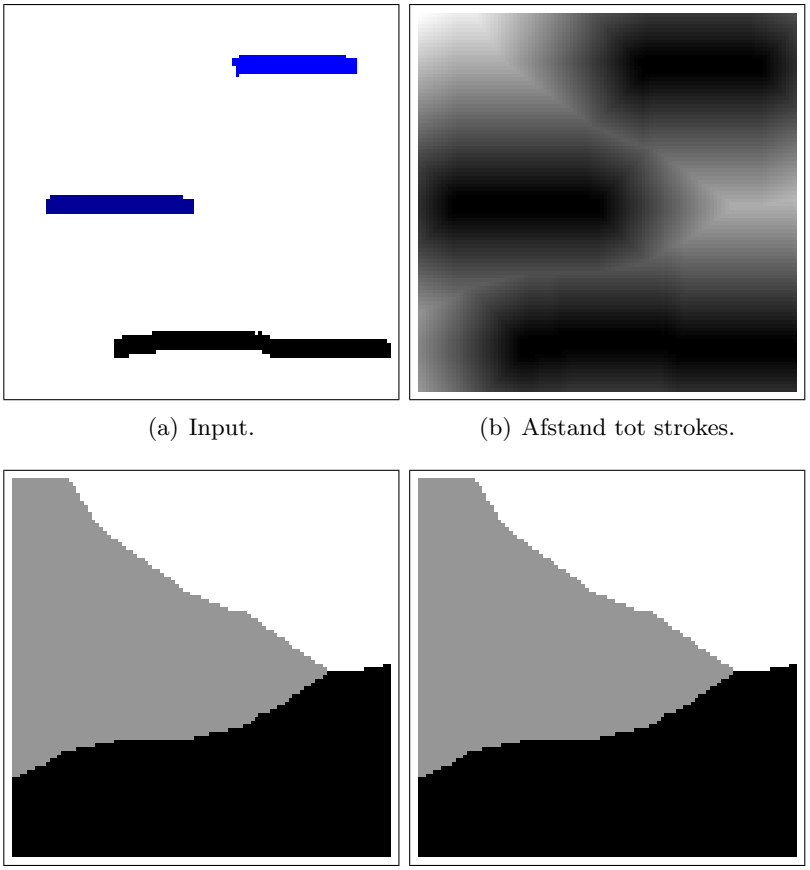
3.3.2 Bepalen van afstanden

In deze stap wordt voor elke pixel de afstand tot de dichtsbijzijnde stroke berekend, waarbij de randen in de afbeelding zoveel mogelijk gerespecteerd worden. Deze informatie wordt gebruikt als er verschillende labels worden gebruikt binnen één segment in de afbeelding. Zo is het mogelijk om niet alleen strokes van één kleur, maar ook gradiënten te tekenen om diepte aan te geven.

Berekening van de afstand is heel rechtlijnig. Voor elke pixel wordt er bijgehouden wat de afstand is tot de dichtsbijzijnde invoer (figuur 3.9(b)), wat deze invoer is (figuur 3.9(c)) en of deze pixel al geupdate is. Elke pixel in de afbeelding wordt bekeken. Als een nabijgelegen pixel gelabeled is als geupdate of als invoer en de nieuwe afstand is korter dan de huidige afstand, dan wordt de huidige pixel gelabeled als geupdate. De afstand wordt berekend als de afstand toegekend aan de buurpixel plus de afstand tussen die buurpixel en de huidige pixel. Dit zal 1 of $\sqrt{2}$ zijn. Uiteraard moet de nieuwe afstand korter zijn dan de vorige waarde (indien beschikbaar). Als er geen pixels meer wijzigen, is elke diepte toegekend. Om te vermijden dat er informatie tussen verschillende spatiale segmenten lekken, wordt er geen vergelijking gemaakt als de twee pixels tot een verschillend segment behoren. Deze informatie wordt afgeleid uit de vorige stap in sectie 3.3.1. Mocht er toch een *lek* zijn, kan dit worden opgevangen door de energiefunctie, zoals uitgelegd in sectie 3.5.3.

3.4 Minimalisatie

Voor de energieminimalisatie maken we gebruik van de algoritmen zoals besproken in sectie 2.1.3. In deze sectie zullen we enkele optimalisaties voor de algoritmen bespreken.



(a) Input.

(b) Afstand tot strokes.

(c) Label van de dichtsbijzijnde stroke.

(d) Resultaat van minimalisatie.

Figuur 3.9: Berekenen van afstanden tot strokes.

3.4.1 Optimalisatie van de bestaande techniek

Voor de minimalisatie is er getest met het $\alpha-\beta$ swap algoritme en het α expansion algoritme. We hebben gekeken naar het aantal iteraties en de grootte van de grafen. De $\alpha-\beta$ swap zal vaker een min-cut berekenen per iteratie, maar de grafen zijn significant kleiner. Dit is omdat bij het swapalgoritme slechts een deel van de pixels gebruikt worden, en bij de expansion alle pixels (en vaak nog extra knopen worden toegevoegd). Dit is de reden dat het $\alpha-\beta$ swap algoritme sneller werkt dan het α expansion algoritme. Dit is ook uit de tests gebleken. Er is getest met een afbeelding van 100 op 100 pixels. Het swap algoritme was 18 maal sneller dan het expansion algoritme. Bovendien is het geheugenverbruik minder, omdat de grafen minder groot zijn. De eindresultaten in de test waren precies hetzelfde.

Het $\alpha-\beta$ swap algoritme kan nog versneld worden om sneller naar een oplossing te komen en de grafen zo snel mogelijk te verkleinen. In de literatuur is geen aanwijzig gegeven voor

de volgorde van het testen van labels. Eigen experimenten hebben aangetoond dat een binair zoekalgoritme het snelste resultaat heeft, in tegenstelling tot sequentieel alle labels aflopen en random labels kiezen. Stel dat de labels tussen 0 en 1 liggen, met tussenstappen van $0,1$. Dan wordt eerst 0 en 1 getest. Vervolgens wordt het midden van dit interval genomen en hetzelfde principe tweemaal toegepast, dus testen tussen 0 en $0,5$, en tussen $0,5$ en 1. Dit gaat door tot het interval lengte nul heeft of kleiner is dan de waarden van de tussenstappen. Deze techniek zorgt ervoor dat de grafen kleiner zijn en de labels heel snel *ongeveer goed* zijn.

Door deze techniek toe te passen wordt er een voorwaarde toegevoegd. Het is nodig dat de startlabeling gelijk is aan 0 of 1. Het algoritme zal dan beginnen met alle pixels die 0 als label hebben indien nodig te verplaatsen naar 1 volgens de energiefunctie. Stel dat ongeveer de helft van de pixels nu label 1 hebben, dan zal de graaf voor elke swap onder $0,5$ onmiddellijk half zo groot zijn. Kleinere grafen zullen sneller gesplitst kunnen worden.

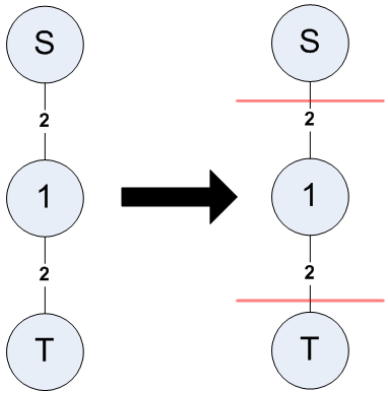
Deze techniek is mogelijk door een eigenschap van de data term van de energiefunctie, namelijk dat de data term groter wordt als de te testen waarde en de waarde van de stroke verder uit elkaar liggen.

Een tweede versnelling is te realiseren in de initialisatie. Als de dieptes allemaal worden geïnitialiseerd op diepte nul, zal er de eerste keer een grote graaf worden gevormd. In plaats daarvan hebben we de dieptes geïnitialiseerd op de waarde van de dichtsbijzijnde stroke. Deze is al berekend in de voorverwerking. Bij deze waarden is de kans groot dat deze al correct zijn. Daarom zal er sneller naar een oplossing worden gegaan. Bovendien zijn de grafen kleiner, zodat ook hierdoor de rekentijd beperkt wordt.

Het algoritme vereist verder dat elke pixelknoop verbonden is met zowel de source s (label α) als de sink t (label β). Dit is strikt gezien niet noodzakelijk. Alle bogen die met s of t zijn verbonden en een gewicht van nul hebben, kunnen weggelaten worden. Dit is omdat alle bogen een positieve kost hebben. De nulboog zal dan altijd tot de min-cut behoren. Het al dan niet cutten op deze boog zal de min-cut niet wijzigen. Een boog van waarde nul kan voorkomen als de energie nul is (als de huidige oplossing perfect overeenkomt met de ingegeven stroke op die pixel) of als er geen stroke is getekend.

Dit kan inhouden dat een bepaalde knoop niet meer rechtstreeks verbonden is aan s of t, enkel via andere knopen. Dit is geen probleem, omdat in de implementatie niet rechtstreeks rekening wordt gehouden met de bogen die zijn doorgesneden. In plaats daarvan worden de verzamelingen S en T gebruikt, zoals beschreven in sectie 2.1.3. Knopen (en dus ook pixels) in S krijgen het label β en knopen in T het label α. Het is zelfs mogelijk dat knopen helemaal niet meer zijn verbonden met s of t, zelfs niet via andere knopen. In dit geval blijft het label

ongewijzigd. Zie bijvoorbeeld figuur 3.10. Beide bogen kunnen tot de min-cut behoren. Het algoritme is zo geïmplementeerd dat er gesneden wordt op beide bogen. In deze toepassing zal het label dan gewoon ongewijzigd blijven, wat een verwacht resultaat is.



Figuur 3.10: Min-cut van een graaf zonder unieke oplossing. De graaf wordt gesneden op de twee bogen. Omdat de knoop nergens meer toe behoort, wijzigt het label niet.

3.5 Energiefunctie

3.5.1 Voorwaarden

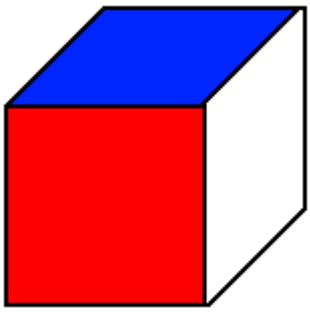
Een goede energiefunctie moet voldoen aan verschillenden voorwaarden. Een triviale voorwaarde is dat de diepte aangegeven door de gebruiker (zoveel mogelijk) gerespecteerd wordt. Dit wordt met de dataterm verwezenlijkt.

Voor de smooth term is het nodig dat pixels die ongeveer dezelfde kleur hebben ook als één gebied worden beschouwd. Dit wil niet zeggen dat deze eenzelfde diepte moeten hebben. Zie bijvoorbeeld figuur 3.11. Het blauwe vlak heeft een geleidelijk wijzigende diepte nodig en het rode vlak heeft overal dezelfde diepte.

In deze implementatie is het mogelijk verschillende dieptes te tekenen in één segment van de afbeelding. Ook hiermee moet rekening worden gehouden.
De energiefunctie die we hebben gebruikt bestaat uit de volgende vorm:

$$E(f) = \sum_{p \in P} (\sum_{(p,q) \in N_p} ((1 - \lambda) \times V_{p,q}(f_p, f_q)) + \lambda \times D_p(f_p)) \tag{3.1}$$

p is een pixel uit de afbeelding P. V noemen we de smooth term en D de dataterm. $\lambda (\in [0, 1])$ bepaalt het gewicht van de verschillende delen. Dit gewicht bepaalt hoeveel de dataterm en de smooth term doorwegen in de berekeningen. Een grote λ zal een groot belang hechten aan



Figuur 3.11: Kubus: De diepte voor het rode vlak kan niet op dezelfde manier gepropageerd worden als het blauwe vlak

de dataterm, zodat de strokes belangrijk worden. Omdat de splitsing van de afbeelding door middel van randen gebruikt wordt in de smooth term, zal er bij een grote λ hier niet veel aandacht aan gegeven worden en daarom zullen segmenten quasi genegeerd worden. Bovendien zal propagatie niet goed verlopen, omdat er ook geen aandacht wordt gegeven aan pixels van gelijke kleur. Alleen de plaatsen waar effectief strokes staan, zullen worden beschouwd. Daarom zal de afbeelding grotendeels uit een enkele diepte bestaan, de initiële waarde. Plaatsen waar strokes zijn getekend, zullen wel zichtbaar zijn.

Als er een te kleine λ wordt gekozen, zal het algoritme de neiging hebben de diepte te veel in vlakken te houden. Dan zullen de strokes worden genegeerd. Ook hier zal de afbeelding bestaan uit een enkele diepte. Het algoritme zal geen propagatie doen volgens de informatie van de strokes, omdat deze niet genoeg worden geforceerd. Het zal minder energie kosten de hele afbeelding een uniforme diepte te geven.

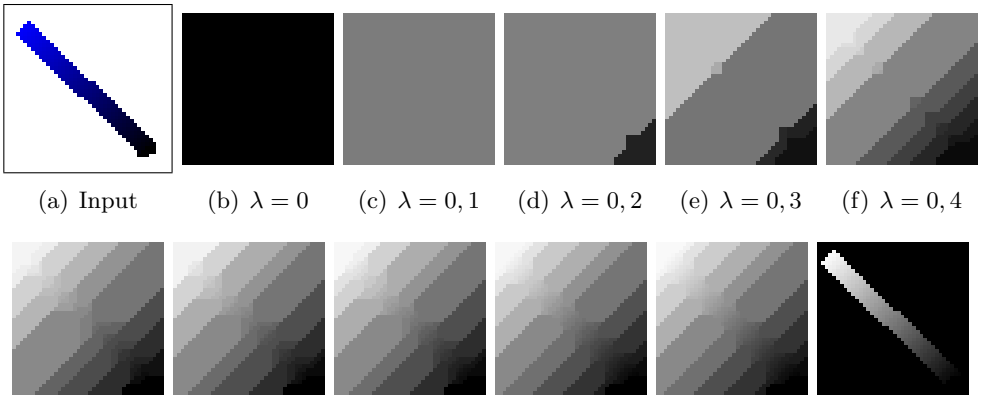
We hebben de functie getest met verschillende waarden en de waarde voor λ is vastgelegd op $0,7$. Verschillende effecten voor verschillende waarden van λ zijn te zien in figuur 3.12, figuur 3.13 en figuur 3.14.

3.5.2 Dataterm

Voor de dataterm van een pixel p zijn de volgende gegevens nodig:

- De labeling waar op wordt getest, f_p.
- Wat de waarde van de strokes op deze plaats is, indien aanwezig. Deze waarde noemen we d_p.

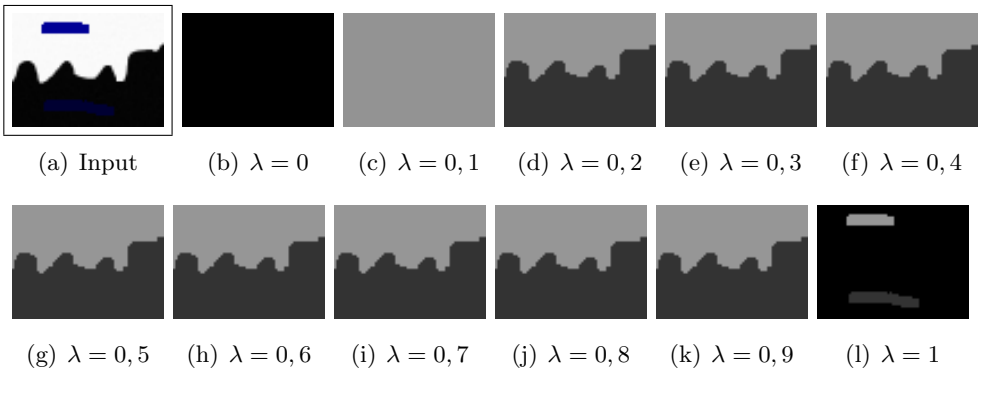
In deze functie wordt dan getest of de waarde f_p overeenkomt met d_p. De functie wordt dan een van:



(a) Input (b) $\lambda = 0$ (c) $\lambda = 0,1$ (d) $\lambda = 0,2$ (e) $\lambda = 0,3$ (f) $\lambda = 0,4$

(g) $\lambda = 0,5$ (h) $\lambda = 0,6$ (i) $\lambda = 0,7$ (j) $\lambda = 0,8$ (k) $\lambda = 0,9$ (l) $\lambda = 1$

Figuur 3.12: Resultaat van de berekening voor verschillende λ



(a) Input (b) $\lambda = 0$ (c) $\lambda = 0,1$ (d) $\lambda = 0,2$ (e) $\lambda = 0,3$ (f) $\lambda = 0,4$

(g) $\lambda = 0,5$ (h) $\lambda = 0,6$ (i) $\lambda = 0,7$ (j) $\lambda = 0,8$ (k) $\lambda = 0,9$ (l) $\lambda = 1$

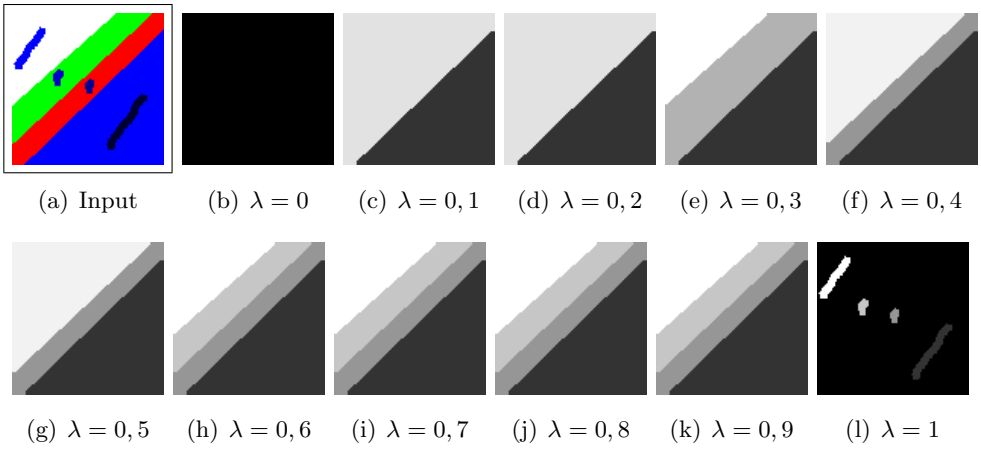
Figuur 3.13: Resultaat van de berekening voor verschillende λ

$$D_p(f_p) = |f_p - d_p|$$
$$D_p(f_p) = (f_p - d_p)^2$$

Beide functies hebben in deze implementatie gelijkaardige resultaten, geen van beide heeft een voordeel. Daarom is er voor de eerste gekozen, omdat deze minder rekenintensief is. De functie is nul als er geen waarde is voor d_p.

3.5.3 Smooth term

De smooth term meet de energie tussen twee pixels p en q. De waarden die we willen testen zijn f_p en f_q. In deze functie gebruiken we de informatie die we hebben berekend in de voorverwerking in sectie 3.3. De functie ziet er als volgt uit:



(a) Input (b) $\lambda = 0$ (c) $\lambda = 0,1$ (d) $\lambda = 0,2$ (e) $\lambda = 0,3$ (f) $\lambda = 0,4$

(g) $\lambda = 0,5$ (h) $\lambda = 0,6$ (i) $\lambda = 0,7$ (j) $\lambda = 0,8$ (k) $\lambda = 0,9$ (l) $\lambda = 1$

Figuur 3.14: Resultaat van de berekening voor verschillende λ

$$V(p,q) = E_d \times E_e$$

In E_d wordt het verschil tussen de labels f_p en f_q getest en dit verschil wordt vergeleken met de gradiënt $g(p,q)$. Deze gradiënt wordt berekend uit de invoer van de oriëntatiestrokes. Deze informatie is alleen beschikbaar als de diepte wordt berekend. Gradiënten van gradiënten worden niet berekend. Als deze niet bekend is, is $g(p,q) = 0$.

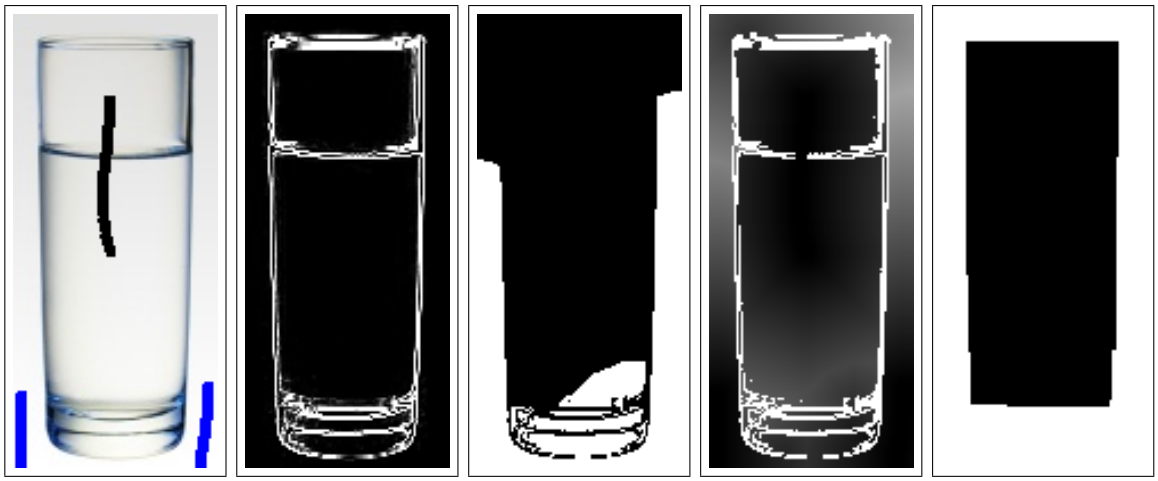
In E_e worden de randen van de afbeelding beschouwd. De energie moet minder zijn tussen pixels waar er zich ook een rand bevindt. Daarom zal de energie verminderd worden in functie van de functie $Edge(p,q)$ van sectie 3.3.1. De volledige functie is dan:

$$V(p,q) = E_d \times E_e = (f_p - f_q - g(p,q)) \times (1 - Edge(p,q))^\gamma$$

Merk op dat $Edge(p,q)$ een waarde tussen 0 en 1 teruggeeft. Als er zeker een rand is op deze plaats, zal de energie nul worden. Dit is omdat het niet uitmaakt welke labels de pixels hebben, omdat deze toch niet aan elkaar gerelateerd zijn. De factor die de rand bepaald wordt een aantal maal toegepast, zodat ook kleinere randen een kans maken belang te hebben. Als er geen macht werd gebruikt, zouden vage randen praktisch genegeerd worden. Nu wordt er meer aandacht besteed aan de randen. Er is gekozen voor $\gamma = 25$.

Nu moet er nog een factor worden toegevoegd die rekening houdt met verschillende waarden in een enkel segment in de afbeelding. Hiervoor kan de informatie uit de afstanden tot de dichtsbijzijnde stroke worden gebruikt, zoals berekend in sectie 3.3.2. Er is berekend wat de afstand is tot de dichtsbijzijnde stroke vanaf p en wat het label c_p van deze stroke is. Analoog

voor c_q. Nu weten we dat als c_p en c_q verschillend zijn, de pixels p en q bij een andere stroke horen. Daarom moet de energie V lager zijn tussen deze pixels, omdat het niet gewenst is dat deze hetzelfde label krijgen. Aan de andere kant kunnen we de energie niet nul maken, omdat er misschien andere informatie correcter is. De afstand kan bijvoorbeeld niet helemaal correct zijn, omdat deze niet voldoende rekening heeft gehouden met randen. Zie bijvoorbeeld figuur 3.15(c). De afstanden worden voor een deel berekend over de randen heen, maar omdat de energiefunctie dit niet als absolute informatie beschouwt, zal het resultaat juist zijn (zoals te zien in figuur 3.15(e), er zijn geen valse randen gebruikt door de afstandsfunctie). Merk op dat de onderkant van het glas niet meegenomen is in de voorgrond, omdat hier duidelijk een rand zichtbaar is. Deze kan ook weggewerkt worden door een extra stroke te tekenen. Om deze informatie toe te passen wordt $V(p, q)$ gedeeld door 5 als de labels verschillend zijn. De energie zal verlagen om de pixels een verschillende waarde te geven, maar zal nooit nul worden (tenzij $V(p, q)$ al nul was).



(a) Input (b) Randen (c) Afstandslabels (d) Afstanden (e) Resultaat

Figuur 3.15: Afstanden tot de strokes zonder scherpe randen.

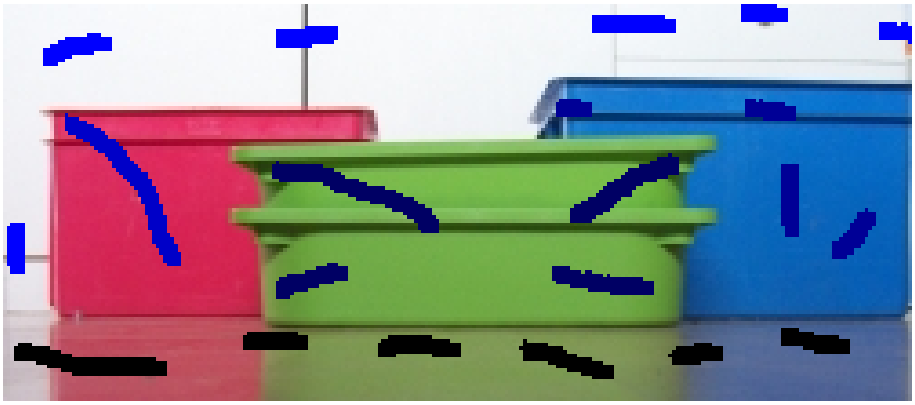
Hoofdstuk 4

Resultaten

In deze sectie worden enkele resultaten van het algoritme weergegeven en besproken. Niet alleen de werkende voorbeelden worden getoond, maar ook de situaties waar het algoritme niet werkt. Waarom iets werkt of niet werkt wordt ook in deze sectie besproken.

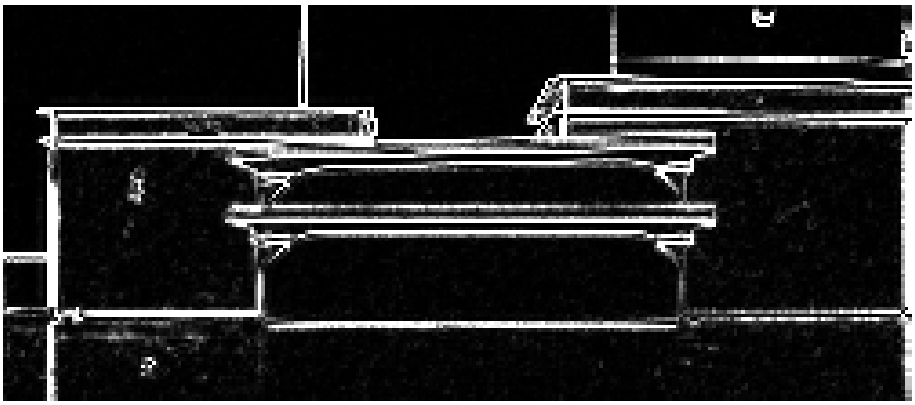
4.1 Vlakke diepte

In dit voorbeeld is er een achtergrond met drie objecten voor geplaatst. De objecten zijn van een verschillende kleur. De invoer is te zien in figuur 4.1(a). De randen zijn te zien in figuur 4.1(c) en het eindresultaat in figuur 4.1(b). Het resultaat laat een goed resultaat zien, op een paar plaatsen na. Rechtsboven van de eerste bak en linksboven van de derde bak is de diepte niet goed gelopen. Dit komt omdat op deze plaats veel ruis is in de randdetectie, zoals te zien in figuur 4.1(c). Daardoor is het onzeker welke diepte het algoritme zal beschouwen als het juiste. Om ervoor te zorgen dat het algoritme wel correct werkt, kan men ofwel zelf de randen tekenen op deze plaats, ofwel een stroke zetten op de probleemgebieden. Dit is gedemonstreerd in figuur 4.2(a). Hier zijn extra randen aangeduid in de probleemgebieden. Deze zijn duidelijk te zien in figuur 4.2(c). Merk op dat er op de grond veel strokes zijn getekend. Deze zijn nodig omdat er een lichte weerspiegeling is. Daarom kunnen de strokes op de bakken te veel naar onder worden gepropageerd. Om dit te vermijden worden deze gebieden gewoon nog eens apart aangeduid als grond.



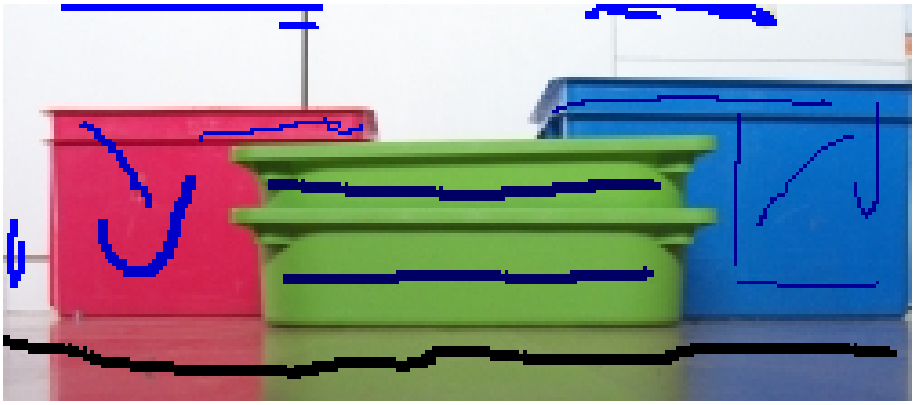
(a) Input

(b) Resultaat



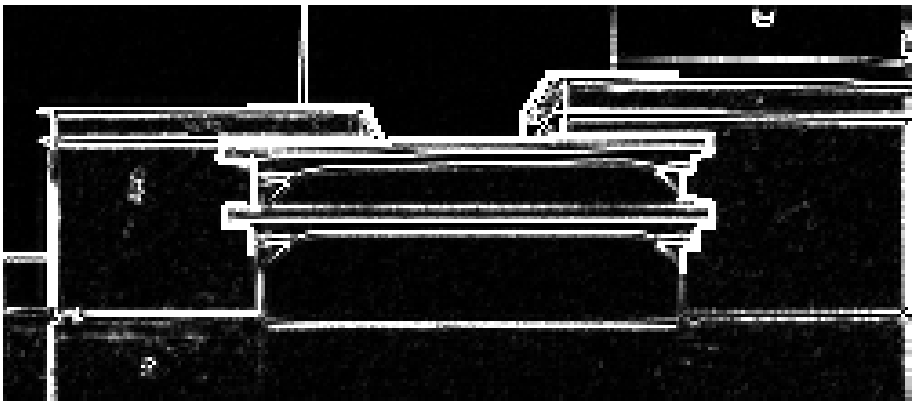
(c) Randen

Figuur 4.1: Voorbeeld van een afbeelding met vlakke oppervlakten.



(a) Input

(b) Resultaat



(c) Randen

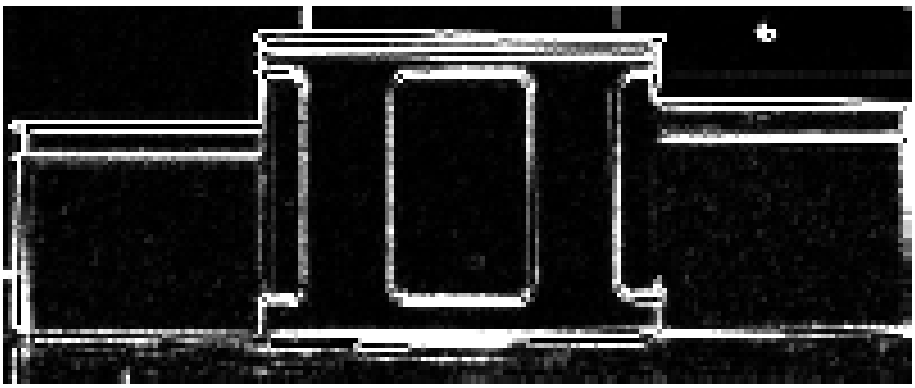
Figuur 4.2: Voorbeeld van een afbeelding met vlakke oppervlakten.

4.2 Gelijke keuren

Dit voorbeeld is gelijkaardig aan het vorige, maar de voorgrondobjecten hebben bijna dezelfde kleur. Daarom moeten er extra strokes getekend worden, zodat onzekere randen toch nog gebruikt worden en valse randen worden genegeerd (zie figuur 4.3(a)). In de invoer zijn er strokes getekend over de randen die niet mogen worden beschouwd. Bovendien staan de strokes dicht bij vage randen die wel moeten worden beschouwd. Uiterst links is de rand te vaag en wordt er niet correct berekend. Ook onderaan tussen de tweede en derde bak zijn er niet genoeg strokes getekend voor een voldoende correct resultaat. Ook deze fouten kunnen worden opgelost door randen bij te tekenen of door extra strokes te trekken. Merk op dat in dit voorbeeld geen extra randen zijn aangegeven. In figuur 4.4 zijn er meer strokes getekend. Het resultaat is nu beter.

(a) Input

(b) Resultaat



(c) Randen

Figuur 4.3: Voorbeeld van een afbeelding met gelijke kleuren.

(a) Input

(b) Resultaat

Figuur 4.4: Voorbeeld van een afbeelding met gelijke kleuren.

4.3 Gekromde vlakken

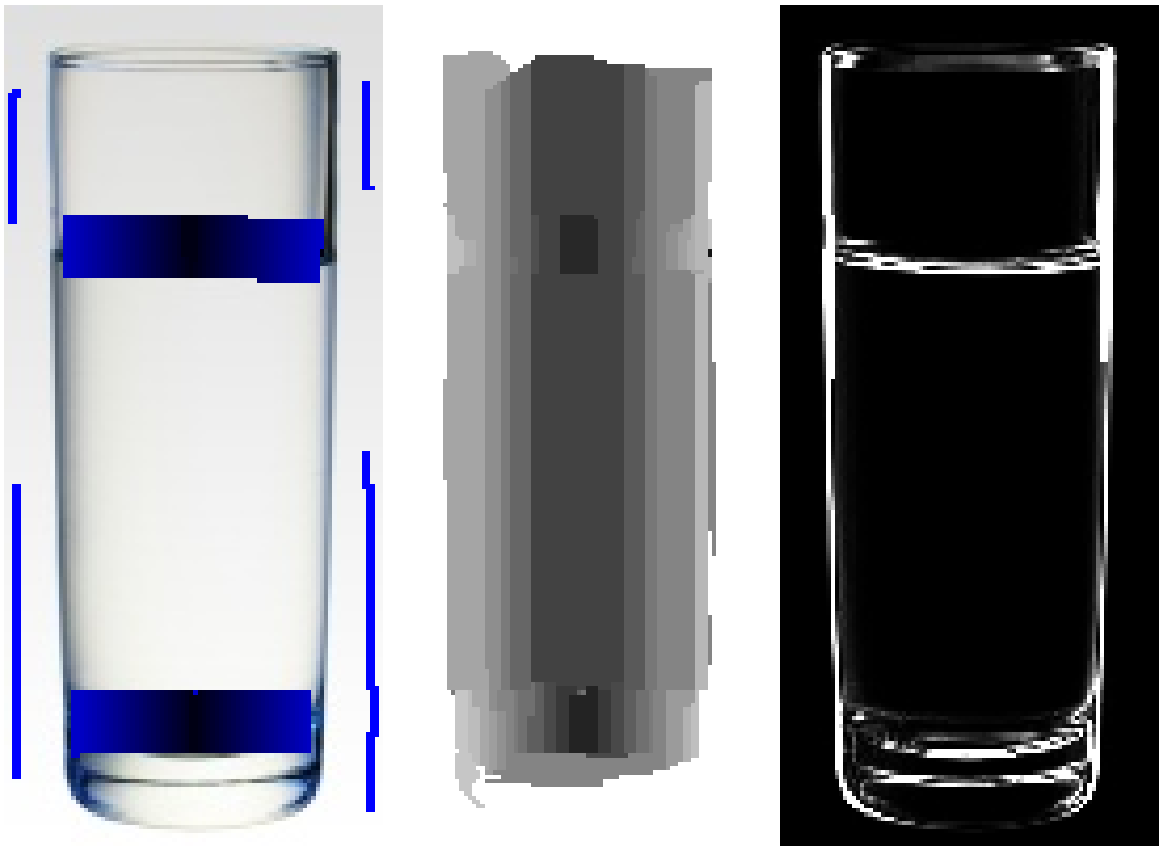
In het volgende voorbeeld in figuur 4.5 is er een ronde bak gebruikt. De diepte is ingegeven als een gradiënt, de ruimte rondom de bak als vaste waarden. De rechterrand van de bak is extra versterkt door de rand handmatig aan te duiden. De andere randen zijn niet gewijzigd. We zien op het resultaat dat de binnenste strook op de bak donker is en lichter wordt naar de rand toe. Dit is het resultaat dat verwacht wordt. Merk op dat de stroken niet een breedte van 1 pixel hebben. Dit komt omdat het minimalisatiealgoritme de neiging heeft zoveel mogelijk pixels in vlakken bijeen te houden, ondanks het feit dat dit geen minimale energie oplevert. Daarom zullen de stroken breder zijn dan de bedoeling. Dit is eigen aan het algoritme en kan opgelost worden door een ander algoritme te kiezen. Andere algoritmen zullen andere sterktes en tekortkomingen hebben, dus is dit resultaat zeker aanvaardbaar voor deze techniek.

(a) Input (b) Resultaat

(c) Randen

Figuur 4.5: Voorbeeld van een afbeelding met ronde oppervlakten.

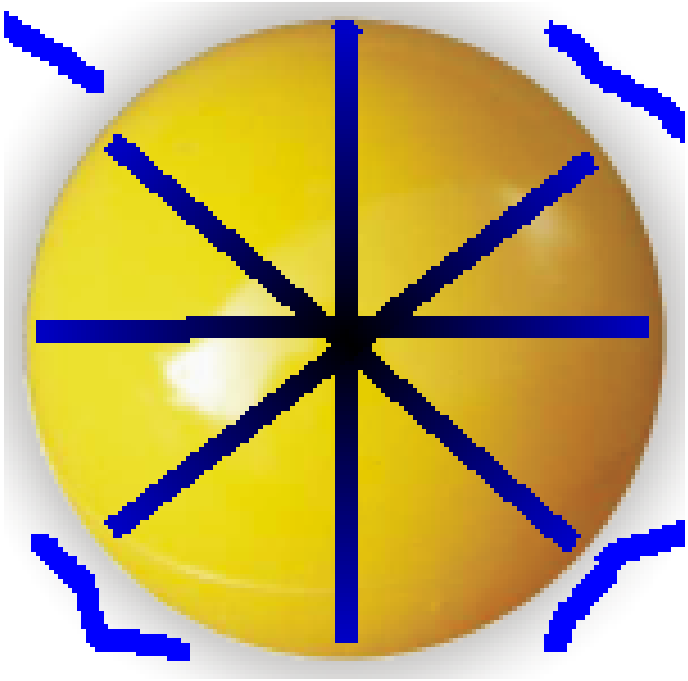
In deze afbeelding is een glas gebruikt. Zoals te zien in het resultaat in figuur 4.6(b), is deze diepteberekening minder goed verlopen. Omdat er veel valse randen horizontaal lopen, moeten hier strokes over getekend worden, zowel boven als onder. Deze strokes zullen nooit perfect bij elkaar uitkomen, dus zal er tegenstrijdige informatie gebruikt worden. Dit heeft zijn weerslag op de correctheid van het resultaat. Er zijn veel gekartelde randen te zien in het midden van het glas. Bovendien wordt er niet voldoende gepropageerd bovenaan en onderaan het glas. Ook dit komt omdat er veel valse randen zijn op deze plaats.



(a) Input (b) Resultaat (c) Randen

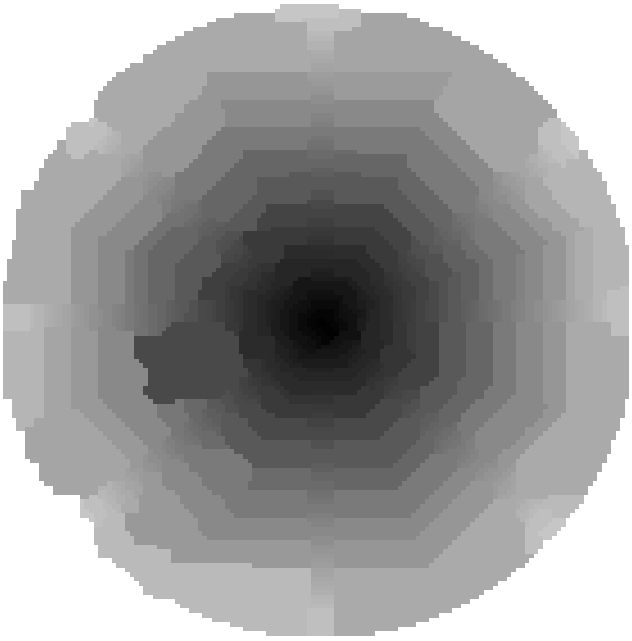
Figuur 4.6: Voorbeeld van een afbeelding met ronde oppervlakten (`http://www.h4x3d.com/feat/themes/glass2.jpg`).

In het volgende voorbeeld is voor een bol gekozen. De invoer is te zien in figuur 4.7(a). Zoals te zien in het resultaat in figuur 4.8(b), is het algoritme er niet in geslaagd mooie ronde vormen te creëren. Dit komt omdat er alleen rekening wordt gehouden met directe afstanden tot de strokes. Daarom wordt er een achthoek gemaakt in plaats van een echte cirkel. Het algoritme is dus niet ideaal voor dit soort afbeeldingen. Het resultaat kan verbeterd worden door extra strokes te tekenen, zoals in figuur 4.9(d). Nu laat figuur 4.9(e) ook geen echt perfecte cirkels zien. Dit komt omdat de strokes nooit perfect getekend kunnen worden. Hierdoor zullen er verspringingen zijn in de diepte als het algoritme een zo goed mogelijke oplossing berekent voor de gegeven strokes.

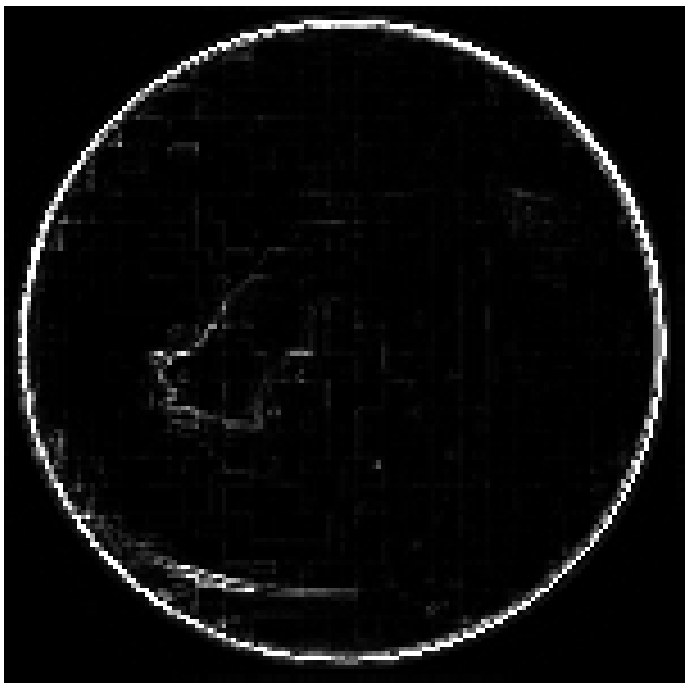


(a) Input

Figuur 4.7: Voorbeeld van een afbeelding met ronde oppervlakten (`http://www.pollypig.com/Images/Spheres/Yellow_solid_sphere.png`).

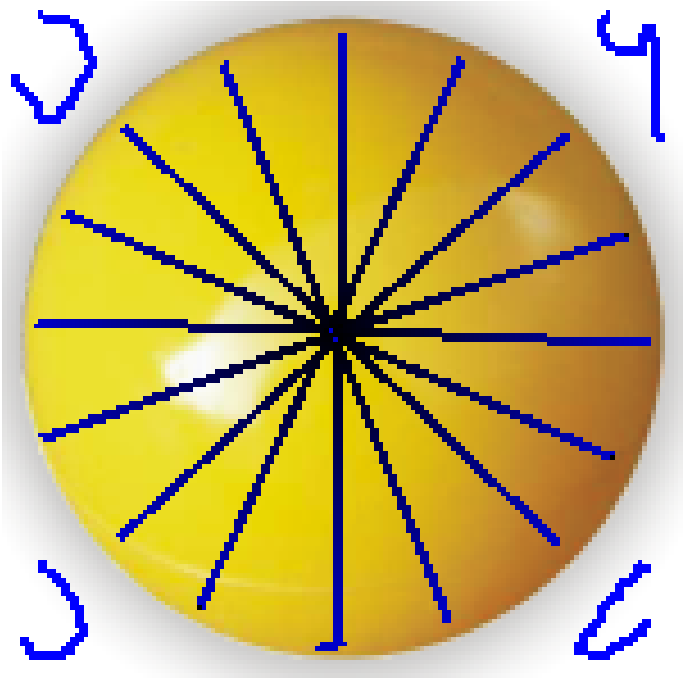


(b) Resultaat

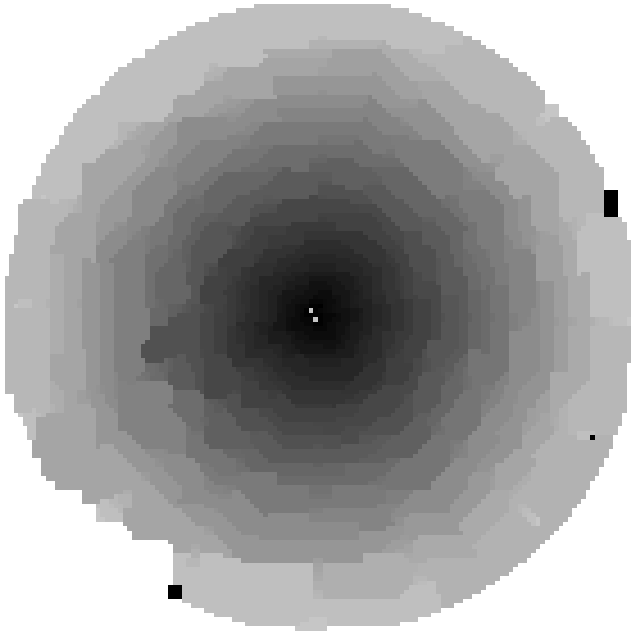


(c) Randen

Figuur 4.8: Voorbeeld van een afbeelding met ronde oppervlakten.



(d) Input



(e) Resultaat

Figuur 4.9: Voorbeeld van een afbeelding met ronde oppervlakten.

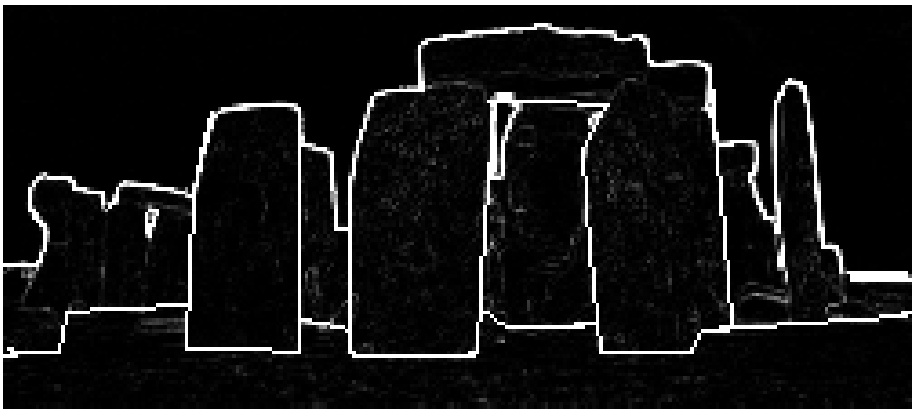
4.4 Echte foto

Hier is een foto van Stonehenge genomen. De randen tussen de stenen en de horizon uiterst links en uiterst rechts zijn niet duidelijk genoeg om automatisch herkend te worden en zijn bijgetekend. De andere randen waren duidelijk genoeg voor het algoritme. Figuur 4.10(b) geeft een vrij goed resultaat. In de achtergrond zijn wat te veel objecten herkend die lucht zouden moeten zijn. Voor de rest is het berekenen van de dieptemap zeer goed verlopen.

In het tweede voorbeeld in figuur 4.11 zijn de resultaten minder goed. Dit komt omdat de randen niet duidelijk worden herkend en daarom is de diepte ook niet goed berekend. De randen in de afbeelding zijn gekarteld en in de dieptemap komt dit terug. Er zijn zelfs enkele objecten die helemaal niet worden herkend.

(a) Input

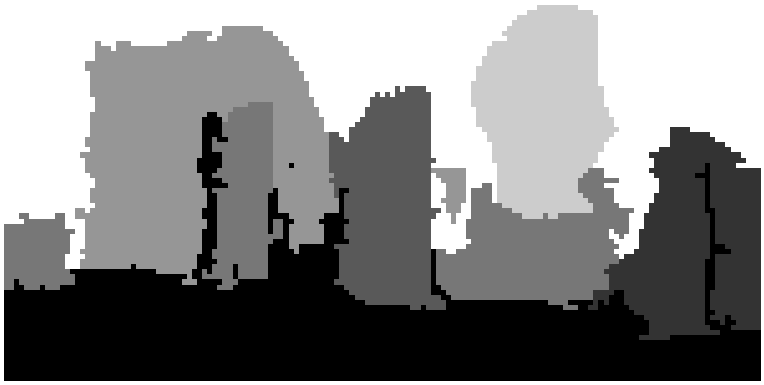
(b) Resultaat



(c) Randen

Figuur 4.10: Voorbeeld van een echte foto (`http://www.itsnature.org/Natural_Wonders/images/article-images/Stonehenge.jpg`).

(a) Input



(b) Resultaat

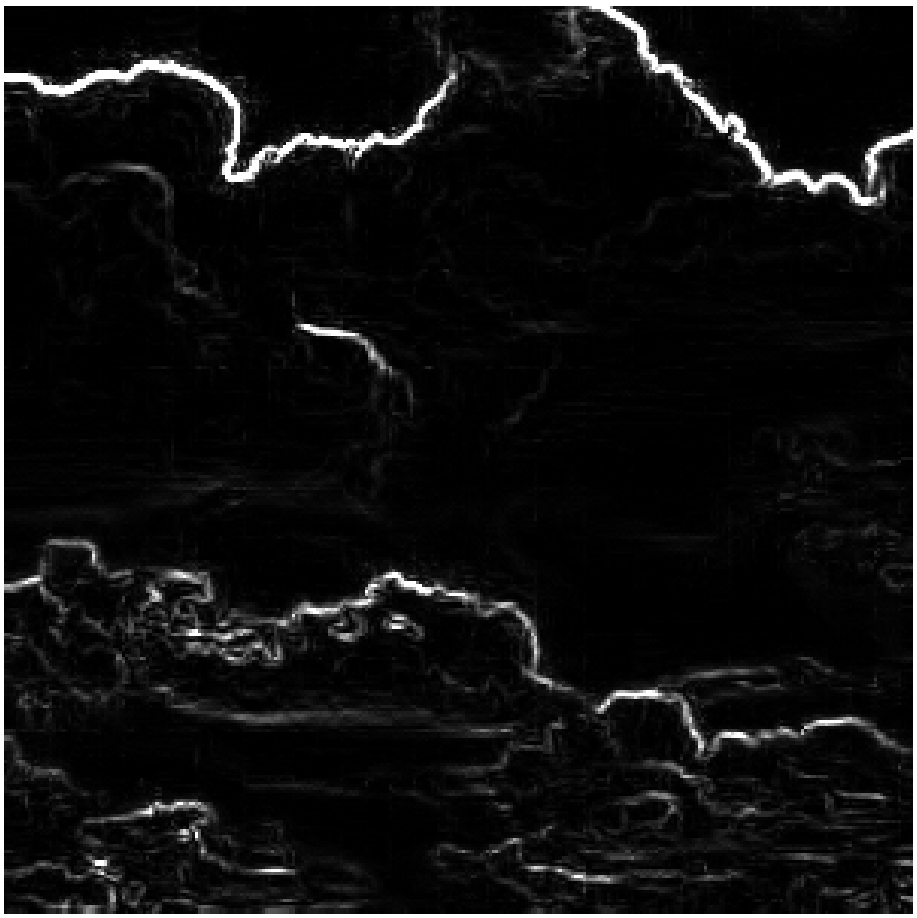
(c) Randen

Figuur 4.11: Voorbeeld van een echte foto (`http://www.sacredsites.com/europe/england/images/stone-ring-avebury-500.jpg`).

4.5 Vage randen

Als het algoritme gevoed wordt met wolken, zullen de segmenten niet goed gedetecteert worden. Dit is goed te zien in figuur 4.13. Niet alle randen worden gevonden, omdat de randen in de afbeelding heel vaag zijn. Als een rand niet gevonden is, zal de energiefunctie hier geen rekening mee kunnen houden en zal de diepte *lekken* over de randen heen. De diepte zal dan bepaald worden door de afstand tot de strokes en deze strokes worden behandeld als verschillende data in een enkel segment in de invoer. Het resultaat is sterk afhankelijk van de positie van de strokes en zal zelden correct zijn.

Figuur 4.12: Voorbeeld van een afbeelding met vage randen (`http://blogs.idc.com/ie/wp-content/uploads/2008/09/cloud-300x300.jpg`).

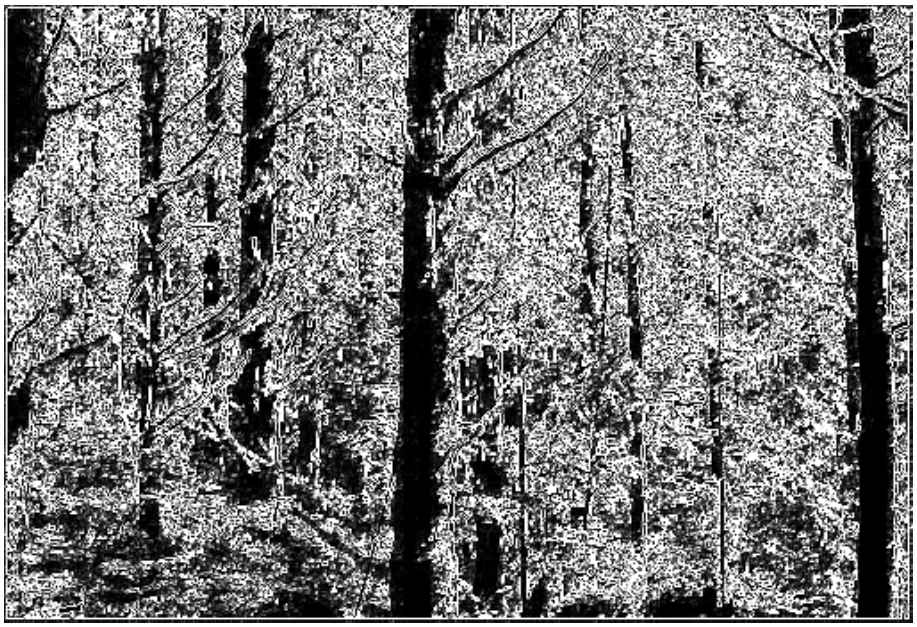


Figuur 4.13: Resultaat van de edge detector.

4.6 Te veel randen

In dit voorbeeld zijn er te veel randen om goed te kunnen segmenteren. Door de overdaad zal de propagatie veel te snel stoppen aan een rand. Segmenten waar geen strokes zijn getekend zullen een quasi willekeurige diepte krijgen, omdat ook de afstandsberekening niet meer geldig is. Deze stopt namelijk ook aan randen.

Figuur 4.14: Voorbeeld van een afbeelding met veel randen (`http://www.sfmcanada.org/english/images/BCCoastalForest.jpg`).



Figuur 4.15: Resultaat van de edge detector.

4.7 Oriëntatie tekenen

Omdat het algoritme de neiging heeft zoveel mogelijk vlakken van pixels bij elkaar te houden, geeft het tekenen van de oriëntatie minder goede resultaten dan het tekenen van een gradiënt als diepte. Meestal zal de oriëntatie gewoon genegeerd worden, omdat het algoritme geen vlakken ineens kan draaien. Dit moet pixel per pixel gebeuren. Omdat alle pixels tegelijkertijd van waarde moeten wisselen, zal het verschil, en dus de energie, tussen individuele pixels hetzelfde blijven of zelfs stijgen en zal er gewoon niets veranderen. Dit is geïllustreerd in figuur 4.16. In 4.16(a) wordt de huidige waardes voor drie pixels gegeven. De gradiënt is berekend als 10. De gewenste waarden worden dan gegeven in figuur 4.16(b). Maar omdat het algoritme berekeningen doet per pixel, zal de energie worden berekend voor de situatie in figuur 4.16(c). De energie tussen pixel 1 en pixel 2 zal dan nul zijn, maar de energie tussen pixel 2 en pixel 3 zal stijgen. Daarom zullen de pixels niet van waarde wisselen.

Daarom is het gebruik van absolute diepte eerder aangeraden dan de oriëntatie te tekenen.

(a) Huidige situatie (b) Gewenste situatie (c) Werking van het algoritme

Figuur 4.16: Het tekenen van oriëntatie: het algoritme berekent niet de correcte energie tussen pixels. Als de gradiënt 10 is moet figuur (a) worden omgezet naar figuur (b). In plaats daarvan wordt de energie berekent voor figuur (c). Omdat de energie tussen pixel 2 en pixel 3 stijgt, zal er niets wijzigen aan de labels van de pixels.

4.8 Performantie

De gebruikte algoritmen werken niet in real-time. Voor een afbeelding van 100 op 100 pixels duurt het ongeveer 1 minuut om één iteratie te berekenen. Een afbeelding van 130 op 130 duurt al een 10-tal minuten per iteratie, een afbeeding van 200 op 200 pixels een half uur. De tijd van de iteratie is afhankelijk van het aantal graph-cuts dat er moet worden uitgevoerd en de grootte van de graaf per cut. De meeste afbeeldingen hebben maar een paar iteraties nodig, maar voor ronde objecten in de afbeelding kan dit oplopen tot een tiental iteraties. Ronde objecten zullen wel veel verschillende diepten hebben in de dieptemap, zodat de grafen kleiner worden naarmate de iteraties vorderen. Daarom heeft het soort afbeelding, de verdeling van de diepte en het aantal iteraties weinig invloed op de lengte van het berekenen. De verschillende factoren heffen elkaar op.

Hoofdstuk 5

Verbeteringen

5.1 Energieminimalisatie

5.1.1 Andere algoritmen

We hebben een implementatie gebruikt die gericht is op grafen. Zoals in sectie 2.1.1 is aangegeven, bestaan er verschillende manieren en aanpakken om een functie te minimaliseren. De gebruikte techniek is speciaal gericht op afbeeldingen. Het nadeel van deze methode is dat pixels zoveel mogelijk in vlakken worden gehouden. Daarom zullen er veel effen vlakken zijn die dezelfde diepte hebben gekregen. Het berekenen van hellingen en smalle stroken kan dan minder goed verlopen, zelfs als de energiefunctie dit oplegt. Vermits er altijd een benadering wordt berekend, wordt de energiefunctie niet perfect gerespecteerd en de energie dus niet maximaal geminimaliseerd. Andere energieminimalisatiealgoritmen doen dit weliswaar ook niet, maar het kan zijn dat door de gebruikte methode er wel betere hellingen kunnen gevormd worden. Het in Gerrits (2008) gebruikte algoritme maakt betere hellingen, maar het resultaat is vager en de randen zijn minder duidelijk zichtbaar.

5.1.2 Andere heuristieken

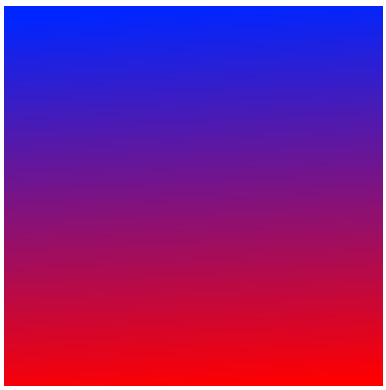
Momenteel wordt het algoritme geïnitialiseerd met de waarde van de dichtsbijzijnde stroke, rekening houdend met harde randen. Mogelijk kan het algoritme geïnitialiseerd worden met andere waarden die ook rekening houden met zachte randen. Deze initialisatie kan niet te precies zijn, anders zal deze hetzelfde resultaat hebben als de minimalisatie zelf en zal de rekentijd daarom ook toenemen.

Een tweede mogelijkheid is de afbeelding in stukken verdelen of de resolutie verkleinen. We hebben tests uitgevoerd door kleine blokken uit de afbeelding apart te berekenen en deze als initialisatie te nemen, maar de resultaten waren niet altijd correct. Dit wordt veroorzaakt door het feit dat het algoritme zoveel mogelijk vlakken bij elkaar houdt. Omdat de blokken

totaal verschillende waarden kunnen krijgen (afhankelijk van waar de strokes staan), kan er verder van een oplossing worden gegaan in plaats van dichterbij. Daarom zullen sommige resultaten langer nodig hebben om te rekenen of zelfs fout zijn. Speciale voorzorgsmaatregelen moeten genomen worden om dan toch correcte of snellere resultaten te bekomen.

5.2 Segmentatie en edge detection

In de huidige implementatie wordt er enkel gebruik gemaakt van de randen die in de afbeelding worden gevonden en eventuele extra randen die door de gebruiker worden aangegeven. Het is ook mogelijk een stroke-based segmentatiealgoritme te gebruiken die de afbeelding splitst. We denken bijvoorbeeld aan het algoritme zoals in Boykov & Jolly (2001), Boykov & Funka-Lea (2006) en Wang & Cohen (2005). Bovendien bestaan er andere segmentatie- en randdetectietechnieken die eveneens gebruikt kunnen worden. Een andere mogelijkheid is het zoeken naar kleurvlakken. Momenteel wordt alleen gezocht naar (duidelijke) randen tussen verschillende kleuren, maar dit is niet altijd goed te berekenen. Neem bijvoorbeeld figuur 5.2. De bovenkant en de onderkant hebben duidelijk een verschillende kleur, maar er wordt geen eenduidige rand gevonden. Dit kan worden opgelost door meer naar kleurvlakken in het geheel te kijken in plaats van naar echte randen.



Figuur 5.1: Een afbeelding met een rand?

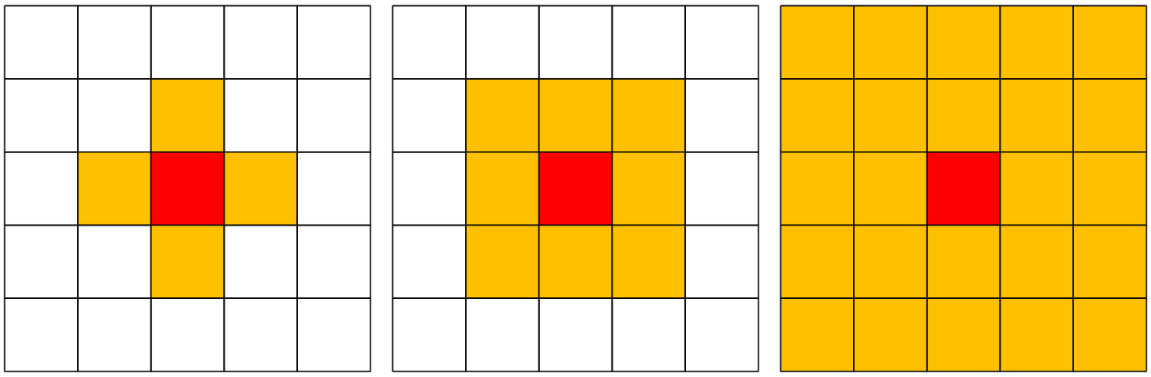
5.3 Energiefunctie

5.3.1 Gebruikte informatie

De huidige energiefunctie kan enkel energie tussen naburige pixels berekenen en gebuikt dus ook enkel informatie tussen naburige pixels. Het is mogelijk extra informatie te beschouwen. We denken bijvoorbeeld aan textuurinformatie, belichting, schaduwen, ...

5.3.2 Buren

Momenteel worden alleen buurpixels beschouwd die effectief langs de pixels in de afbeelding liggen in de horizontale of de verticale richting. Het is ook mogelijk diagonalen te beschouwen of zelfs pixels die verder weg liggen. Hierdoor zal de correctheid verbeterd worden, omdat er meer gegevens kunnen worden gebruikt. Ook de randdetectie kan beter verlopen. Het gebruik van meer buren zal wel de rekentijd verhogen, omdat er meer bogen gevormd zullen worden bij de min-cut.



(a) 4 buren. (b) 8 buren. (c) Buren buiten de echte buren.

Figuur 5.2: Verschillende manieren om buren de definiëren.

5.4 Interface

Het zou praktischer werkbaar zijn als de minimalisatie niet bij elke nieuwe invoer opnieuw berekend moet worden, maar dat er tussenresultaten onthouden kunnen worden. Op die manier kan er meer interactief worden gewerkt: er worden strokes bijgetekend, nieuwe resultaten worden gegenereerd, er worden extra strokes getekend om de oplossing te verfijnen, de resultaten worden aangepast, ... Het incrementeel verfijnen van de oplossing is alleen praktisch haalbaar als het algoritme snel genoeg is of het toelaat tussenresultaten aan te passen aan nieuwe invoer. Dit is niet het geval met de momenteel gebruikte algoritmen.

Hoofdstuk 6

Conclusie

In deze thesis is de creatie van dieptemappen met behulp van stroke-based input onderzocht. Er is gebruik gemaakt van energieminimalisatie om iteratief tot een oplossing te komen. De energieminimalisatie zelf wordt berekend met behulp van graph cuts. Deze techniek heeft vele voordelen, maar ook vele nadelen die ook in het onderzoek naar voren zijn gekomen. Het belangrijkste probleem is dat het algoritme vlakken van pixels bij elkaar probeert te houden, zelfs als dit niet een minimale energie tot gevolg heeft. Daarom is het niet praktisch oriëntaties te tekenen, omdat dit foute resultaten kan geven. Het is praktischer absolute dieptes te tekenen, ook in het geval van een helling. Door de nieuwe methode kunnen er verschillende dieptes getekend worden in één segment van de afbeelding. Hierdoor kunnen toch goede resultaten bekomen worden.

Een tweede probleem is de detectie van de verschillende segmenten. De correctheid van de resultaten is hiervan sterk afhankelijk. Door de gebruiker verschillende algoritmen of een stroke-based methode aan te bieden kan de correctheid worden verbeterd.

Ondanks de potentiële problemen in de methode is de techniek een stap vooruit in het oplossen van het probleem van het omzetten van 2D naar 3D. De gebruiker kan op een snelle en eenvoudige methode dieptemappen creëren voor verschillende toepassingen.

Bijlage A

Wiskundige concepten

A.1 Hessiaanse matrix

Dit is een vierkante matrix van de tweede partiële afgeleides van een functie $f(x_1, \ldots, x_n)$ (Valente 2008).

$$H(f)_{i,j}(x) = D_i D_j f(x)$$

$$\begin{bmatrix} \frac{\delta^2 f}{\delta x_1^2} & \frac{\delta^2 f}{\delta x_1 \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 f}{\delta x_n \delta x_1} & \frac{\delta^2 f}{\delta x_n \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_n^2} \end{bmatrix}$$

A.2 QR factorisatie

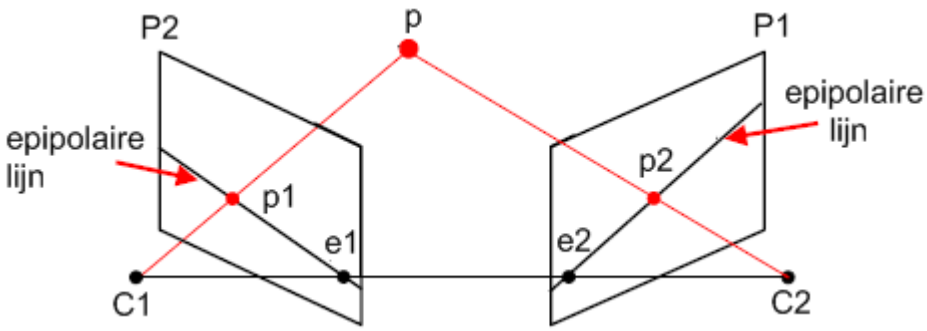
De QR-factorisatie van een matrix A ($m \times n$ en $m \geq n$) is:

$$A = QR$$

met Q een $m \times m$ reële orthogonal matrix en R een $m \times n$ reële bovendriehoeksmatrix. De eerste n kolommen van Q vormt een orthogonale basis voor $ran(A) = \{y \in R^m : y = Ax \wedge x \in R^n\}$, als $dim(ran(A)) = n$. Bewijzen en oplossingsmethoden zijn te vinden in Golub & Van Loan (1996).

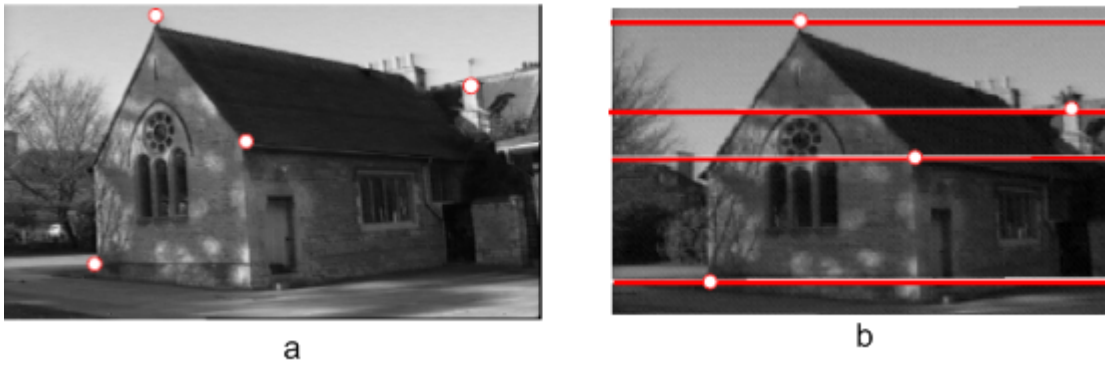
A.3 Epipolaire geometrie

We beschouwen twee projectieve camera's C_1 en C_2 met de bijhorende projectievlakken P_1 en P_2. We beschouwen een punt $p = (x, y, z)$ in de ruimte geprojecteerd op de vlakken als $p_1 = (x_1, y_1)$ en $p_2 = (x_2, y_2)$. De punten p, C_1 en C_2 vormen een epipolair vlak. De snijlijn tussen dit vlak en de projectievlakken heten de epipolaire lijnen. Het snijpunt van



Figuur A.1: Overzicht van de notatie (Andrew Zisserman 2008).

de projectievlakken en de lijn tussen C_1 en C_2 noemt men de epipolen e_1 en e_2. Zie figuur A.1.



Figuur A.2: Overeenkomstige punten met 2 parallelle projectievlakken (a) Vlak 1 met 4 punten (b) Vlak 2 met de overeenkomstige epipolaire lijnen (Andrew Zisserman 2008).

Met een punt in een projectievlak komt een lijn overeen in het andere projectievlak (indien $C_1 \neq C_2$). Het overeenkomstige punt ligt op deze lijn. Dit beperkt het zoeken naar overeenkomstige punten. Zie figuur A.2.

Om reconstructie van 3D-punten te doen, beschouwen we $p_1^T F p_2 = 0$, waarbij F de fundamentele matrix is. Deze matrix heeft rank 2 ($det(F) = 0$) en heeft 7 vrijheidsgraden. Deze zorgt voor een koppeling tussen de twee projectievlakken, en bevat de eigenschappen van de camera's. Als F bekend is, kunnen de 3D-coördinaten van een punt in de ruimte berekend worden. Als acht punten bekend zijn, kan de fundamentele matrix berekend worden door het oplossen van acht van de volgende lineaire vergelijkingen:

$$x_1x_2F_{11} + x_1y_2F_{12} + x_1F_{13} + y_1x_2F_{21} + y_1y_2F_{22} + y_1F_{23} + x_2F_{31} + y_2F_{32} + F_{33} = 0$$

Bibliografie

Andrew Zisserman, M. P. (2008), 'Multiple view geometry'.

Arnold, R. (1983), 'Automated Stereo Perception'.

Baker, H. (1980), 'Edge Based Stereo Correlation.'.

Black and white to color (2008). `http://www.graphic-design.com/Photoshop/Seminars/colorize/index.html`, laatst bekeken op 7 mei 2008.

Blanz, V., Vetter, T. & Tubingen, G. (1999), 'A Morphable Model For The Synthesis Of 3D Faces'.

Boykov, Y. & Funka-Lea, G. (2006), 'Graph Cuts and Efficient NDImage Segmentation', *International Journal of Computer Vision* **70**(2), 109–131.

Boykov, Y. & Jolly, M. (2001), 'Interactive graph cuts for optimal boundary and region segmentation of objects in nd-images', *International Conference on Computer Vision* **1**, 105–112.

Boykov, Y. & Kolmogorov, V. (2004), 'An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision', *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp. 1124–1137.

Boykov, Y., Veksler, O. & Zabih, R. (2001), 'Fast Approximate Energy Minimization via Graph Cuts'.

Colorize image using shapes (2008). `http://www.swish-db.com/tutorials/view.php/tid/629`, laatst bekeken op 7 mei 2008.

Criminisi, A., Reid, I. & Zisserman, A. (2000), 'Single View Metrology', *International Journal of Computer Vision* **40**(2), 123–148.

De Bonet, J. & Viola, P. (1999), 'Poxels: Probabilistic voxelized volume reconstruction', *Proceedings of International Conference on Computer Vision (ICCV)* pp. 418–425.

Definition for word(s) lambertian surface - The Photonics Directory (2008). `http://www.photonics.com/directory/dictionary/lookup.asp?url=lookup&entrynum=2801&letter=l`, laatst bekeken op 22 mei 2008.

Di Zenzo, S. (1986), 'A note on the gradient of a multi-image', *Computer Vision, Graphics, and Image Processing* **33**(1), 116–125.

Dinitz, E. (1970), 'Algorithm for solution of a problem of maximum flow in networks with power estimation', *Soviet Math. Dokl* **11**, 1277–1280.

Downhill Simplex method (2006). `http://mikilab.doshisha.ac.jp/dia/research/report/2006/0806/006/report20060806006.html`, laatst bekeken op 31 juli 2008.

Edmonds, J. & Karp, R. (1972), 'Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems', *Journal of the ACM (JACM)* **19**(2), 248–264.

Eisert, P., Steinbach, E. & Girod, B. (1999), 'Multi-hypothesis, volumetric reconstruction of 3-D objects frommultiple calibrated camera views', *Acoustics, Speech, and Signal Processing, 1999. ICASSP'99. Proceedings., 1999 IEEE International Conference on*.

Faugeras, O. & Keriven, R. (2002), 'Variational principles, surface evolution, PDE's, level set methods and the stereo problem', *Biomedical Imaging, 2002. 5th IEEE EMBS International Summer School on*.

Ford, L. & Fulkerson, D. (1962), *Flows in networks*, Princeton University Press Princeton, NJ.

Fromherz, T. & Bichsel, M. (1994), 'Shape from Contours as Initial Step in Shape from Multiple Cues', *Proceedings of the ISPRS Commission III Symposium on Spatial Information from Digital Photogrammetry and Computer Vision* pp. 249–256.

Fromherz, T. & Bichsel, M. (1995), 'Shape from Multiple Cues: Integrating Local Brightness Information', *Proceedings of the Fourth International Conference for Young Computer Scientists, ICYCS* **95**, 855–862.

Fuchs, M., Blanz, V. & Seidel, H. (2005), 'Bayesian relighting', *Rendering Techniques* pp. 157–164.

Gerrits, M. (2008), 'Stroke-based creation of depth maps'.

Goldberg, A. & Rao, S. (1998), 'Beyond the flow decomposition barrier', *Journal of the ACM* **45**(5), 783–797.

Goldberg, A. & Tarjan, R. (1988), 'A new approach to the maximum-flow problem', *Journal of the ACM (JACM)* **35**(4), 921–940.

Golub, G. & Van Loan, C. (1996), *Matrix Computations*, Johns Hopkins University Press.

Gonzalez, R. & Woods, R. (2007), *Digital Image Processing*, Prentice Hall.

Grimson, W. (1984), 'Computational Experiments with a Feature Based Stereo Algorithm'.

Guisson, J. (2005), Image-based 3D scanner, Master project, Universiteit Hasselt, School voor informatietechnologie.

Hoiem, D., Efros, A. & Hebert, M. (2005), 'Automatic photo pop-up', *Proceedings of ACM SIGGRAPH 2005* **24**(3), 577–584.

Horry, Y., Anjyo, K. & Arai, K. (1997), 'Tour into the picture: using a spidery mesh interface to make animation from a single image', *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* pp. 225–232.

Kang, S. (1998), 'Depth painting for image-based rendering applications'.

Kim, H., Sakamoto, R., Kitahara, I., Toriyama, T. & Kogure, K. (2006), 'Robust Foreground Segmentation from Color Video Sequences Using Background Subtraction with Multiple Thresholds', *Proc. KJPR* pp. 188–193.

Kutulakos, K. & Seitz, S. (2000), 'A Theory of Shape by Space Carving', *International Journal of Computer Vision* **38**(3), 199–218.

Laurentini, A. (1994), 'The Visual Hull Concept for Silhouette-Based Image Understanding', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(2), 150–162.

Levin, A., Lischinski, D. & Weiss, Y. (2004), 'Colorization using optimization', *International Conference on Computer Graphics and Interactive Techniques* pp. 689–694.

Levin, A., Lischinski, D. & Weiss, Y. (2008), 'A Closed-Form Solution to Natural Image Matting', *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp. 228–242.

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J. et al. (2000), 'The Digital Michelangelo Project: 3D Scanning of Large Statues', *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* pp. 131–144.

Lin, Z., Wong, T. & Shum, H. (2002), 'Relighting with the Reflected Irradiance Field: Representation, Sampling and Reconstruction', *International Journal of Computer Vision* **49**(2), 229–246.

Lischinski, D., Farbman, Z., Uyttendaele, M. & Szeliski, R. (2006), 'Interactive local adjustment of tonal values', *ACM Transactions on Graphics (TOG)* **25**(3), 646–653.

Marschner, S., Greenberg, D. & Ithaca, N. (1997), 'Inverse Lighting for Photography', *Presented at the IS&T/SID Fifth Color Imaging Conference*.

Masselus, V., Peers, P., Dutré, P. & Willems, Y. (2003), 'Relighting with 4D incident light fields', *ACM Transactions on Graphics (TOG)* **22**(3), 613–620.

Matusik, W., Buehler, C. & McMillan, L. (2001), 'Polyhedral Visual Hulls for Real-Time Rendering', *Rendering Techniques 2001: Proceedings of the Eurographics Workshop in London, United Kingdom, June 25-27, 2001*.

Matusik, W., Buehler, C., Raskar, R., Gortler, S. & McMillan, L. (2000), 'Image-based visual hulls', *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* pp. 369–374.

Moezzi, S., Katkere, A., Kuramura, D. & Jain, R. (1996), 'Reality modeling and visualization from multiple video sequences', *Computer Graphics and Applications, IEEE* **16**(6), 58–63.

Nie, D., Ma, Q., Ma, L. & Xiao, S. (2007), 'Optimization based grayscale image colorization', *Pattern Recognition Letters* **28**(12), 1445–1451.

Nishihara, H. (1987), 'Practical real-time imaging stereo matcher', *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms* **23**(51), 536–545.

Noriega, P. & Bernier, O. (2006), 'Real Time Illumination Invariant Background Subtraction Using Local Kernel Histograms', *British Machine Vision Association (BMVC)*.

Oh, B., Durand, F. & Chen, M. (2007), 'Image-based modeling and photo editing'. US Patent 7,199,793.

Okabe, M., Zeng, G., Matsushita, Y., Igarashi, T., Quan, L. & Shum, H. (2006), 'Single-view relighting with normal map painting', *Proceedings of Pacific Graphics* pp. 27–34.

Okutomi, M. & Kanade, T. (1992), 'A locally adaptive window for signal matching', *International journal of computer vision* **7**(2), 143–162.

Photoshop CS3 editions (2008). `http://www.adobe.com/products/photoshop/index.html`, laatst bekeken op 7 mei 2008.

Pons, J., Keriven, R., Faugeras, O. & Hermosillo, G. (2003), 'Variational stereovision and 3D scene flow estimation with statistical similarity measures', *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on* pp. 597–602.

Potmesil, M. (1987), 'Generating octree models of 3D objects from their silhouettes in a sequence of images', *Computer Vision, Graphics, and Image Processing* **40**(1), 1–29.

Press, W., Teukolsky, S., Vetterling, W. & Flannery, B. (2002), *Numerical recipes in C: the art of scientific computing*, Cambridge University Press.

Renishaw - CMM (2008*a*). `http://www.renishaw.com/en/6329.aspx`, laatst bekeken op 18 mei 2008.

Renishaw - CMM (2008*b*). `http://www.ace-cmm.com/`, laatst bekeken op 18 mei 2008.

Rother, C., Kolmogorov, V. & Blake, A. (2004), '"GrabCut": interactive foreground extraction using iterated graph cuts', *ACM Transactions on Graphics (TOG)* **23**(3), 309–314.

Roy, S. (1999), 'Stereo Without Epipolar Lines: A Maximum-Flow Formulation', *International Journal of Computer Vision* **34**(2/3), 147–161.

Scanning Monticello (2002). `http://www.cs.virginia.edu/Monticello/`, laatst bekeken op 18 mei 2008.

Scharstein, D. & Szeliski, R. (2002), 'A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms', *International Journal of Computer Vision* **47**(1), 7–42.

Scharstein, D., Szeliski, R. & Coll, M. (2003), 'High-accuracy stereo depth maps using structured light', *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on.*

Seitz, S. & Dyer, C. (1999), 'Photorealistic Scene Reconstruction by Voxel Coloring', *International Journal of Computer Vision* **35**(2), 151–173.

Seitz, S., Curless, B., Diebel, J., Scharstein, D. & Szeliski, R. (2006), 'A comparison and evaluation of multi-view stereo reconstruction algorithms', *Int. Conf. on Computer Vision and Pattern Recognition* pp. 519–528.

Shapiro, L. & Stockman, G. (2001), *Computer Vision. 2001*, Prentice Hall.

Shum, H., Han, M. & Szeliski, R. (1998), 'Interactive construction of 3D models from panoramic mosaics', *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR).*

Smith, A. & Blinn, J. (1996), 'Blue screen matting', *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* pp. 259–268.

Soatto, S. & Jin, A. (2003), 'Tales of shape and radiance in multiview stereo', *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on* pp. 974–981.

Szeliski, R. & Golland, P. (1999), 'Stereo Matching with Transparency and Matting', *International Journal of Computer Vision* **32**(1), 45–61.

TechEBlog - How to Colorize Black ans White Images in Photoshop (2008). http://www.techeblog.com/index.php/tech-gadget/how-to-colorize-bw-images-in-photoshop, laatst bekeken op 7 mei 2008.

Thomas Jefferson's monticello (2008). http://www.monticello.org/, laatst bekeken op 18 mei 2008.

Valente, C. (2008), 'Hessian matrix'. http://planetmath.org/encyclopedia/HessianMatrix.html, laatst bekeken op 7 augustus 2008.

Wang, J. & Cohen, M. (2005), 'An iterative optimization approach for unified image segmentation and matting', *Proc. IEEE Intl. Conf. on Computer Vision* **2**, 930–935.

Weinman, J., Hanson, A. & McCallum, A. (2004), 'Sign detection in natural images with conditional random fields', *Proc. of IEEE International Workshop on Machine Learning for Signal Processing*.

Woodham, R. (1989), 'Photometric method for determining surface orientation from multiple images', *Mit Press Series Of Artificial Intelligence Series* pp. 513–531.

Yang, R. & Pollefeys, M. (2003), 'Multi-resolution real-time stereo on commodity graphics hardware', *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*.

Zhang, L., Dugas-Phocion, G., Samson, J. & Seitz, S. (2002), 'Single-view modelling of free-form scenes', *Journal of Visualization and Computer Animation* **13**(4), 225–235.