Beeldbewerking in het gradiëntdomein a.d.h.v. de Poissonvergelijking

Thesis voorgedragen tot het behalen van de graad van licentiaat in de Informatica, afstudeervariant Informatica-Multimedia

Pascal Knapen
Promotor: Prof. dr. Philippe Bekaert
Begeleider: Tom Mertens

School voor Informatie Technologie
Transnationale Universiteit Limburg
6 juni 2005

Voorwoord

Ik zou graag enkele mensen willen bedanken die een onmisbare steun zijn geweest tijdens het schrijven van deze thesis. Mijn promoter prof. dr. Philippe Bekaert zonder wie deze thesis nooit tot stand zou zijn gekomen. Mijn begeleider Tom Mertens die mij vele nuttige opmerkingen heeft gegeven en bij wie ik altijd met al mijn vragen terecht kon. Ook zou ik graag de medestudenten die mij hebben geholpen tijdens het schrijven van deze thesis willen bedanken. Tot slot mag ook mijn familie en vrienden hier zeker niet ontbreken die mij altijd tot steun zijn geweest.

Abstract

We introduceren verschillende technieken om beelden aan te passen door te werken in het gradiëntdomein. De Poisson vergelijking is hierbij het mathematische hart. Dit is een partiële differentiaal vergelijking die ons toelaat om van het gradiëntdomein terug te keren naar het spatiale domein. We zullen zien hoe deze afgeleid wordt en hoe we deze in discrete gevallen kunnen oplossen. Vervolgens wordt deze methode aangepast aan verscheidene toepassingsdomeinen. Tot slot wordt mijn implementatie besproken.

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

1.1 Doel

Fotobewerking gebeurt overal rondom ons al staan we hier zelden bij stil. Er bestaan dan ook ontelbare tools, denk maar aan *AdobePhotoshop* of *Gimp*. Echter vragen deze tools bij het aanleren veel oefening en tijd. Maar ook bij experten is er nog steeds de nodige creativiteit nodig om tot goede resultaten te komen. Omwille hiervan wordt er onderzoek gedaan naar nieuwe technieken die krachtiger, gebruiksvriendelijker en intuïtiever zijn.

In de reclame-industrie is fotobewerking een belangrijk onderdeel. Zo worden zichtbare oneffenheden op de huid van modellen weggewerkt om tot een zo aantrekkelijk mogelijk beeld te komen. Ook in de filmindustrie zijn deze technieken veel gebruikt. Bij stunts worden de veiligheidskoorden op de beelden verwijderd zodat de actie er spannender, gevaarlijker en vooral realistischer uitziet.

Niet enkel in de entertainment business is beelbewerking belangrijk. Ook in de medische sector wordt hier veelvuldig gebruik van gemaakt. Meestal gaat het hier dan om beeldverbetering zodat bijvoorbeeld tumoren makkelijker opgespoord kunnen worden.

Omdat het onderzoeksgebied van beeldbewerking gigantisch groot is, zullen we ons in deze thesis toespitsen op enkele recente technieken met in de hoofdrol de gradiënt. Er is ook een implementatie gemaakt en een vergelijkende studie.

1.2 Overzicht

We geven een overzicht wat er in de volgende hoofdstukken beschreven wordt:

Hoofdstuk 2 Dit hoofdstuk geeft een inleiding in het gradiëntdomein en beschrijft een mathematisch model waarop de rest van de hoofdstukken gebaseerd is.

Hoofdstuk 3 Enkele intuïtieve tools worden afgeleid van het mathematisch model. Ook bekijken we de problemen die kunnen opduiken bij het gebruik van deze tools.

Hoofdstuk 4 Een overzicht wordt gegeven van technieken voor het creëren van mozaïekbeelden. Met behulp van Hoofdstuk 3 zetten we enkele methoden om naar het gradiëntdomein. We vergelijken de resultaten bekomen in het gradiëntdomein met deze in het spatiale domein.

Hoofdstuk 5 Een studie van textuursynthese geeft ons inzicht om deze techniek te gebruiken om onbekende beeldregio's in te vullen. Weerom zetten we deze methode om naar het gradiëntdomein en vergelijken de resultaten.

Hoofdstuk 6 Bespreken van mijn implementatie.

Tot slot maken we nog de opmerking dat alle foto's die niet van mijn hand zijn, aangeduid zijn met een (*). Veel plezier!

Hoofdstuk 2

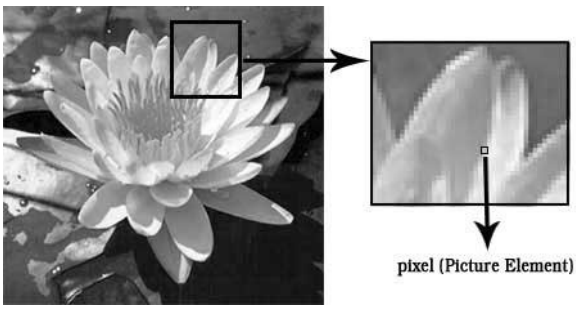
Het Gradientdomein

In dit hoofdstuk geven we een inleiding in det gradiëntdomein. We leggen uit wat de gradiënt van een beeld is en bekijken enkele toepassingen hiervan in het hedendaagse leven. Verder stellen we een mathematisch model op voor het reconstrueren van een beeld op basis van de zijn gradiënt en stellen we hoe we dit discreet kunnen oplossen.

2.1 Wat is de gradiënt van een beeld

2.1.1 Algemeen

Een digitaal beeld is opgebouwd uit pixels (figuur 2.1).



Figuur 2.1: Pixel

Hieruit volgt dat een beeld kan beschreven worden als een twee-dimensionale funtie, $f(x, y) = kleurwaarde$, waarbij x en y respectievelijk de rij en kolom voorstellen. De *kleurwaarde* is afhankelijk welk kleurmodel gebruikt wordt. Het meest voorkomende is het RGB-kleurmodel. Dit wil zeggen dat een kleur wordt voorgesteld door een combinatie van 3 kleurwaarden: Rood, Groen en Blauw. In het verloop van deze thesis zullen we steeds werken met het RGB-kleurmodel tenzij anders vermeld.

De gradiënt van een beeld is de eerste afgeleide van $f(x, y)$ zoals beschreven in [14]. De afgeleide van een digitale functie wordt gedefinieerd in termen van verschillen. Er zijn verschillende manieren om deze verschillen te definiëren, maar moeten steeds voldoen aan volgende voorwaarden:

1. moet 0 zijn in gebieden waar de kleur constant is
2. moet verschillend zijn van 0 bij aanvang van een kleurverandering
3. moet verschillend zijn van 0 in gebieden met een verloop

Voor de functie $f(x, y)$ is de gradiënt van f op coördinaten (x,y) gedefinieerd als een twee-dimensionale kolomvector.

$$\nabla f = \begin{bmatrix} Gx \\ Gy \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.1}$$

Hierbij stellen $\frac{\partial f}{\partial x}$ en $\frac{\partial f}{\partial y}$ de partiële afgeleiden voor in x en y. De grootte of norm van deze vector is gegeven door

$$mag(\nabla f) = \left[Gx^2 + Gy^2\right]^{1/2} = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \tag{2.2}$$

Hoewel het niet correct is, wordt de grootte van de gradiëntvector ook wel vaak gradiënt genoemd. Omdat vergelijking 2.2 relatief zwaar is te berekenen, wordt 2.2 vaak vervangen door:

$$mag(\nabla f) \approx |Gx| + |Gy| \tag{2.3}$$

Deze vergelijking is simpeler te berekenen. Hoewel dit slechts een benadering is van de gradiënt, blijven de eigenschappen van de gradiënt behouden. Om aan te tonen op welke manieren de gradiënt kan berekend worden zullen we gebruik maken van figuur 2.2.

z1	z2	z3
z4	z5	z6
z7	z8	z9

Figuur 2.2: 3 x 3 omgeving van een beeld

In 2.2 stellen **z1-z9** pixels van een beeld voor en we willen de gradiënt voor **z5** berekenen. De meest eenvoudige manier om de gradiënt van **z5** te berekenen is:

$$Gx = (z6 - z5) \;\; en \;\; Gy = (z8 - z5) \tag{2.4}$$

Een andere manier wordt voorgesteld door gebruik te maken van gekruisde verschillen:

$$Gx = (z9 - z5) \ \ en \ \ Gy = (z8 - z6) \tag{2.5}$$

De laatste methode die we hier bespreken, is door gebruik te maken van de zogenaamde *Sobel* operators:

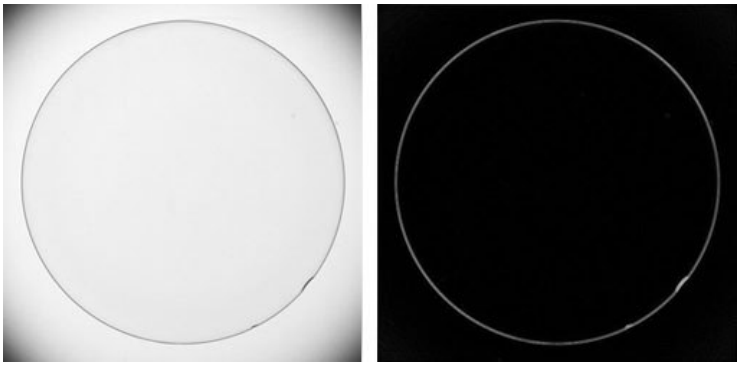
$$Gx = (z7+2z8+z9)-(z1+2z2+z3) \ \ en \ \ Gy = (z3+2z6+z9)-(z1+2z4+z7) \tag{2.6}$$

De bedoeling van de *Sobel* operators is om een zekere gladheid te bekomen. Merk op dat voor elk van de vergelijkingen 2.4-2.6, Gx en Gy zullen resulteren in 0 in omgevingen met constante kleurwaarde.
Indien we nu voor elke pixel, met eender welke definitie, de gradiënt berekenen, bekomen we een vectorveld dat we het gradiëntveld noemen. Nemen we vervolgens voor elke vector de norm dan bekomen we een gradiëntbeeld.

2.1.2 Algemene Toepassingen van de gradiënt

Industriële inspectie

De gradiënt wordt vaak gebruikt in industriële inspectie, ofwel om de mens te helpen in het opsporen van fouten of als een préprocessing stap in geautomatizeerde inspectie. Figuur 2.3 geeft een beeld weer van een lens en de gradiënt van dit beeld. De belichting is zo gekozen dat inperfecties makkelijk worden waargenomen, zoals in de hoek rechts onderaan. Het gradiëntbeeld werd bekomen met behulp van de Sobel operatoren.



Figuur 2.3: Beeld van een lens en het gradiëntbeeld

Detectie van randen

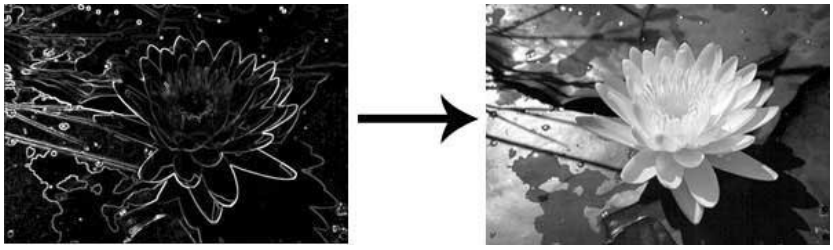
Zoals beschreven in [22, 13] werken de meeste rand-detectie algoritmen onder de veronderstelling dat een rand voorkomt waar er discontinuïteit is of waar de

Figuur 2.4: Detectie van randen van een beeld

norm van de gradiënt relatief groot is. Als de afgeleide genomen wordt van het beeld en we de punten vinden waar de gradient groot is, zullen we ook de randen gevonden hebben. In figuur 2.4 zien we hier een voorbeeld van.

2.2 Reconstructie met behulp van gradiënt

In deze sectie zien we hoe we een beeld kunnen reconstrueren op basis van het gradiëntveld. Deze techniek werd reeds besproken in [11]. Figuur 2.5 illustreert de omzetting.



Figuur 2.5: Omzetten van gradient naar orginele beeld

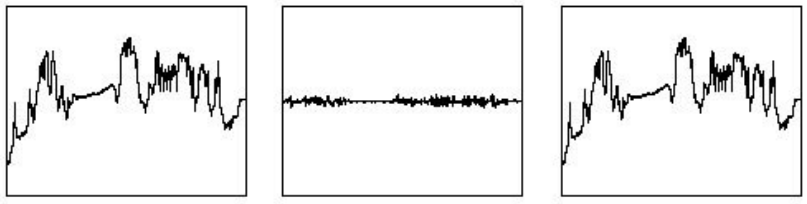
2.2.1 Achtergrond

We beginnen met het idee uit te leggen in 1D. Beschouw de 1D functie $H(x)$ en de eerste afgeleide van H noteren we als $G(x) = H'(x)$. We kunnen nu H terug reconstrueren (tot op een constante term) door te integreren:

$$H(x) \approx I(x) = C + \int_0^x G(t)dt \tag{2.7}$$

Dit proces wordt weergegeven in figuur 2.6. De eerste figuur geeft een 1D functie

weer $H(x)$, de tweede stelt de eerste afgeleide voor $G(x)$ en de laatste stelt de gereconstrueerde functie voor $I(x) \approx H(x)$.



Figuur 2.6: Omzetting eerste afgeleide naar orginele functie

Om deze methode nu om te zetten naar 2D, $H(x, y)$, maken we gebruik van de gradiënt, ∇H. Zoals in 1 dimensie, stellen we:

$$G(x, y) = \nabla H \tag{2.8}$$

Aangezien H de ombekende is die we willen berekenen, kunnen we G niet berekenen uit H. We veronderstellen dat G op een bepaalde manier verkregen is en correct is voor het beeld dat we willen reconstrueren. In 2D is het niet zo simpel om gewoon te integreren omdat G niet noodzakelijk integreerbaar is. In andere woorden, er bestaat niet noodzakelijk een beeld I zodat $G = \nabla I$. Dit is enkel het geval indien G een *conservatief* vectorveld is [15, 28] of met andere woorden de gradiënt $\nabla I = (\partial I/\partial x, \partial I/\partial y)$ aan volgende voorwaarde voldoet:

$$\frac{\partial^2 I}{\partial x \partial y} = \frac{\partial^2 I}{\partial y \partial x} \tag{2.9}$$

wat zelden het geval is voor onze G. De oplossing bestaat erin om te zoeken in de ruimte van all potentële 2D functies naar een functie I waarvan de gradiënt zo dicht mogelijk G benadert. In andere woorden, I moet de volgende integraal minimalizeren:

$$\int\int F(\nabla I, G) dx dy \tag{2.10}$$

waar $F(\nabla I, G) = \|\nabla I - G\|^2 = \left(\frac{\partial I}{\partial x} - Gx\right)^2 + \left(\frac{\partial I}{\partial y} - Gy\right)^2$. Volgens het *Variational Principle* [25] moet een functie I die de integraal 2.10 minimalizeerd, voldoen aan Euler-Lagrange vergelijking:

$$\frac{\partial F}{\partial I} - \frac{d}{dx}\frac{\partial F}{\partial Ix} - \frac{d}{dy}\frac{\partial F}{\partial Iy} = 0 \tag{2.11}$$

wat een partiële differentiaal vergelijking is in I. Als we F substitueren krijgen we volgende vergelijking:

$$2\left(\frac{\partial^2 I}{\partial x^2} - \frac{\partial Gx}{\partial x}\right) + 2\left(\frac{\partial^2 I}{\partial y^2} - \frac{\partial Gy}{\partial y}\right) = 0 \tag{2.12}$$

Als we delen door twee en de termen herschikken, verkrijgen we de zogenaamde Poisson vergelijking:

$$\nabla^2 I = divG \tag{2.13}$$

waarbij ∇^2 de Laplaciaan is (tweede afgeleide), $\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$ en $divG$ de divergentie is van het vectorveld G, gedefinieerd als $divG = \frac{\partial Gx}{\partial x} + \frac{\partial Gy}{\partial y}$. Deze lineaire partiële differentiaal vergelijking kunnen we in het discrete geval numeriek oplossen en wordt besproken in volgende sectie.

2.2.2 Oplossen Poisson vergelijking

Om een differentiaal vergelijking op te lossen zoals 2.13 moeten er eerst grensvoorwaarden gespecifieerd worden. De meest voorkomende grensvoorwaarden zijn de *Neumann* en de *Dirichlet* grensvoorwaarden. De Neumann grensvoorwaarde, $\nabla I.\mathbf{n} = 0$, zegt dat de afgeleide in de richting van de grens gelijk is aan nul. Met deze gensvoorwaarde is de oplossing nu volledig bepaald tot op een enkele additieve term. Deze term heeft echter geen echte betekenis aangezien de oplossing toch aangepast wordt aan de beperkingen van het display device. De *Dirichlet* grensvoorwaarde zal besproken worden in het volgende hoofdstuk samen met de toepassingen hiervan.
Aangezien zowel de Laplaciaan ∇^2 als de *div* lineaire operatoren zijn, kunnen we ze benaderen met eindige verschillen wat resulteert in een lineair stelsel van vergelijkingen. Meer specifiek benaderen we de Laplaciaan met:

$$\nabla^2 I(x,y) \approx I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y) \tag{2.14}$$

en benaderen we $divG$ met:

$$divG \approx Gx(x,y) - Gx(x-1,y) + Gy(x,y) - Gy(x,y-1) \tag{2.15}$$

Door deze benaderingen te gebruiken krijgen we een groot stelsel van lineaire vergelijkingen. Indien het beeld dat we willen reconstrueren bestaat uit n pixels, hebben we een lineair stelsel van n vergelijkingen en n onbekenden. Merk wel op dat elke onbekende enkel afhankelijk is van zijn vier buren, waardoor elke rij van de matrix slechts vijf niet nul elementen heeft. We kunnen dit stelsel oplossen met behulp van de Gauss-Seidel [4, 20] iteratieve methode.

2.2.3 Gauss-Seidel methode

In deze sectie geven we een inleiding in de Gauss-Seidel iteratieve methode. Vaak zal er in de praktijk echter een variant gebruikt worden van deze methode. We zullen de werking illustreren met een voorbeeld. Veronderstel volgend simpel stelsel:

$$\begin{aligned} 10x_1 - x_2 + 3x_3 &= 7 \\ x_1 + 11x_2 - 5x_3 &= 5 \\ 2x_1 - x_2 + 13x_3 &= 8 \end{aligned} \tag{2.16}$$

We willen dit nu gaan oplossen met behulp van de Gauss-Seidel iteratieve methode. Eerst herschrijven we het stelsel als volgt:

$$\begin{aligned} 10x_1 &= 7 + x_2 - 3x_3 \\ 11x_2 &= 5 - x_1 + 5x_3 \\ 13x_3 &= 8 - 2x_1 + x_2 \end{aligned} \tag{2.17}$$

Voor de eerste vergelijking kiezen we de eerste keer de waarden van x_2 en x_3. In dit geval stellen we $x_2 = 0$ en $x_3 = 0$. Hieruit kunnen we een nieuwe waarde voor x_1 berekenen en deze kunnen we gebruiken in de tweede vergelijking waarbij we x_3 terug gelijk stellen aan nul en bekomen zo een nieuwe waarde voor x_2. In de derde vergelijking kunnen we nu zowel een nieuwe waarden van x_1 als x_2 invullen en bekomen we een nieuwe waarde voor x_3. We gebruiken nu de nieuwe waarden van x_2 en x_3 in de eerste vergelijking en herhalen het hele proces totdat x_1, x_2 en x_3 niet meer veranderen:

$$\begin{aligned} 10x_1 &= 7 + x_2 - 3x_3 = 7 + 0 - 0 = 7 & x_1 &= 7/10 = 0.7 \\ 11x_2 &= 5 - x_1 + 5x_3 = 5 - 0.7 + 0 & x_2 &= 0.3903030310 \\ 13x_3 &= 8 - 2x_1 + x_2 = 8 - 2 * 0.7 + 0.3909090910 & x_3 &= 0.5377622378 \end{aligned} \tag{2.18}$$

De volgende iteratie produceert:

$$\begin{aligned} x_1 &= 0.5777622378 \\ x_2 &= 0.6464589956 \\ x_3 &= 0.5768857323 \end{aligned} \tag{2.19}$$

en na elf iteraties krijgen we:

$$\begin{aligned} x_1 &= 0.5937031485 \\ x_2 &= 0.6619190406 \\ x_3 &= 0.5749625188 \end{aligned} \tag{2.20}$$

Met elke volgende iteratie veranderen x_1, x_2 en x_3 niet meer en hebben we de oplossing van het stelsel gevonden.

Hoofdstuk 3

Beeldbewerking met Poisson

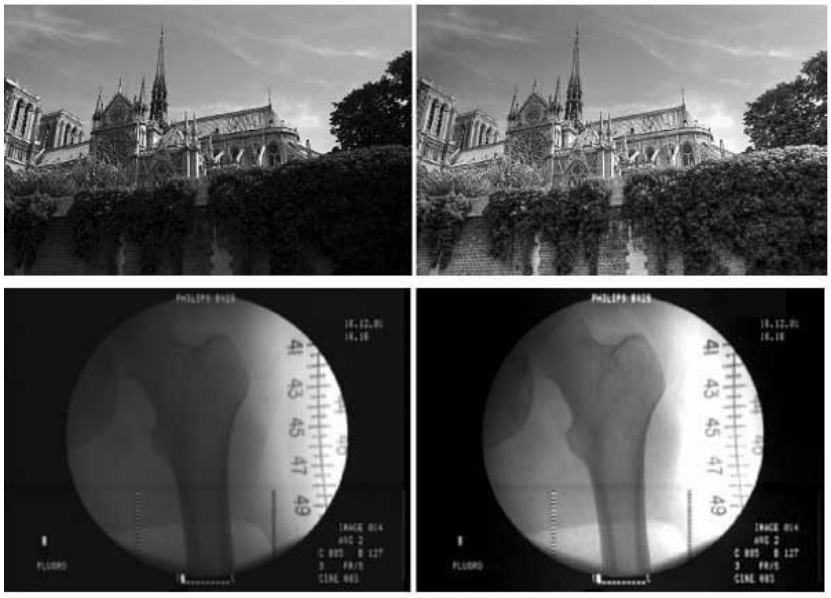
In [23] worden er een aantal tools gepresenteerd die gebaseerd zijn op de reconstructietechniek van het vorig hoofdstuk. De eerste set van tools zal toelaten om een bronbeeld naadloos in een ander beeld te voegen. De tweede set van tools laat toe om het uiterlijk van een beeld naadloos aan te passen. Vervolgens bespreken we enkele tekortkomingen van deze techniek.

3.1 Algemeen

Het berwerken van beelden komt neer op of globale veranderingen of lokale veranderingen. Wij zijn enkel geïnteresseerd in lokale veranderingen. Veranderingen die beperkt zijn tot een regio die op een eenvoudige, moeiteloze manier geselecteerd is. De veranderingen in deze regio kunnen gaan van kleine aanpassingen tot een totale vervanging van de inhoud. Klassieke tools zouden werken met filters en interactieve knip-en-plak tools, die bijna steeds zichtbare naden opleveren.
We stellen hier een methode voor waaruit we een aantal tools kunnen afleiden die naadloze resultaten opleveren. Het mathematische hart van deze methode bestaat uit een Poisson partiële differentaal vergelijking met *Dirichlet* grensvoorwaarden (zie ook 2.2.2). Met *Dirichlet* grensvoorwaarden bedoelen we dat de grenzen van de regio (waarover de Poisson vergelijking gedefinieerd is) aan bepaalde waarden moeten voldoen.
Zoals in 2.2.1 kan het oplossen van de Poisson vergelijking ook geïnterpreteerd worden als een minimalizatieprobleem: het berekent de functie wiens gradiënt het dichtst een gegeven vectorveld benadert onder gegeven grenscondities. Op deze manier zal de reconstructiefunctie de grenzen naar binnen interpoleren waarbij het de veranderingen van het vectorveld zo dicht mogelijk benadert. Dit zou moeten leiden tot een naadloos resultaat. Het is immers geweten dat zachte intensiteitsgradiënten bovenop een beeld kunnen geplaatst worden met

3.2.1 Comprimeren van het dynamisch bereik in het gradiëntdomein

In [11], wordt het gradiëntveld van een beeld met een groot dynamisch bereik (HDR - High Dynamic Range), aangepast op een niet lineaire manier door de norm van grote gradiënten te verzwakken. Een nieuw beeld met een lager dynamisch bereik wordt dan bekomen door de Poisson vergelijking met *Neumann* grensvoorwaarden op te lossen zoals beschreven in sectie 2.2. Deze methode kan ook gebruikt worden voor beeldverbetering, zie figuur 3.2.

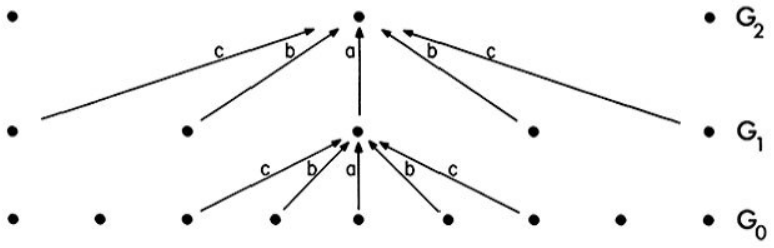


Figuur 3.2: Beeldverbetering met Gradient Domain High Dynamic Range Compression (*)

3.2.2 De multiresolutie curve

De techniek besproken in [5] dient om twee of meer beelden te combineren op een naadloze manier. Deze methode kan dus zowel gebruikt worden voor het creëren van mozaïekbeelden (zie Hoofdstuk 4) als voor naadloos klonen 3.3. In deze methode worden de beelden die gecombineerd moeten worden eerst opgesplitst in een set van componentbeelden. De componentbeelden van het ene beeld worden dan gecombineerd met deze van het andere beeld door gebruik te maken van een gewogen gemiddelde in de overgangszone. De nieuw bekomen componentbeelden worden vervolgens opgeteld om het naadloze beeld te bekomen. Omdat deze methode een enorm toepassingsdomein heeft en om een inleiding te geven op multiresolutie technieken gaan we hier in meer detail op

in.
De eerste stap is het opsplitsen van het de beelden in de zogenaamde componentbeelden. We gaan eerst uit het orginele beeld G_0 een set van beelden genereren $G_0, G_1, G_2, ..., G_N$. Figuur 3.3 geeft een voorbeeld hoe de beelden bekomen kunnen worden in 1 dimensie.



Figuur 3.3: 1D grafische representatie van de REDUCE operatie

Voor 2D beelden vertrekken we dus bij G_0. De waarde van elke pixel in de volgende rij, G_1, wordt berekend als het gewogen gemiddelde van een 5x5 subarray van G_0. G_2 kan op dezelfde manier bekomen worden uit G_1 en als dit proces iteratief wordt voortgezet krijgen we een set: $G_0, G_1, G_2, ..., G_N$. Met elke iteratie wordt dus de resolutie van het beeld gehalveerd. Als we ons voorstellen dat al deze beelden op elkaar liggen, krijgen we een piramide (*Gaussiaanse* piramide). Dit iteratief proces wordt gedefinieerd als een REDUCE operatie:

$$G_l = REDUCE[G_{l-1}], \tag{3.3}$$

waarbij

$$G_l(i,j) = \sum \sum_{m,n=1}^{5} w(m,n) G_{l-1}(2i+m, 2j+n) \tag{3.4}$$

waarbij $w(m,n)$ de gewicht functie voorstelt om het gewogen gemiddelde te bereken.
Vervolgens wordt een nieuwe set van beelden gecreëerd uit de bekomen piramide door elk level van de piramide van het volgende lagere level af te trekken. Omdat de verschillende beelden van resolutie verschillen introduceren we nu de EXPAND operatie. Laat $G_{l,k}$ het beeld zijn door G_l k keer uit te breiden.

$$G_{l,k} = EXPAND[G_{l,k-1}], \tag{3.5}$$

waarbij

$$G_{l,k}(i,j) = 4 \sum \sum_{m,n=2}^{2} G_{l,k-1}\left(\frac{2i+m}{2}, \frac{2j+n}{2}\right) \tag{3.6}$$

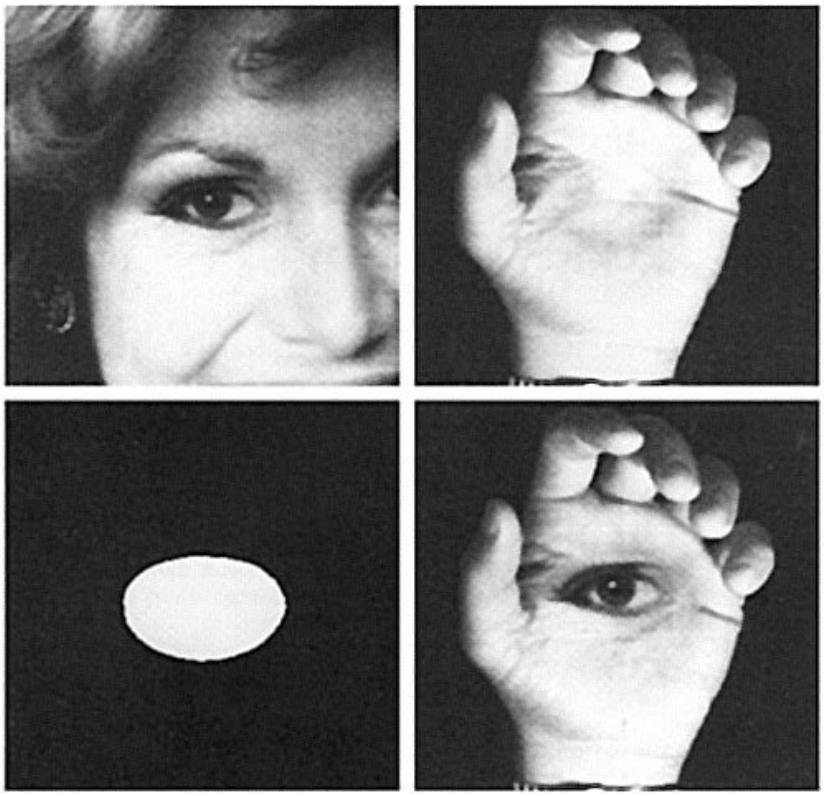
De componentbeelden $L_0, L_1, ..., L_N$ worden bekomen met volgende formule

$$L_l = Gl - EXPAND[G_{l+1}] = G_l - G_{l+1,1} \tag{3.7}$$

De set van componentbeelden $L_0, L_1, ..., L_N$ noemen we ook wel de *Laplaciaanse* piramide. De stappen om de *Laplaciaanse* piramide te bouwen kunnen worden omgekeerd om zo het orginele beeld te bekomen.

$$G_0 = \sum_{l=0}^{N} L_{l,l} \tag{3.8}$$

Nu kunnen we twee beelden gaan combineren zoals in figuur 3.4.



Figuur 3.4: Naadloos klonen met de multiresolutie curve methode (*)

Hierbij gaan we als volgt te werk:

1. Bouw de *Laplaciaanse* piramide voor beide figuren.
2. Bouw een *Gaussiaans* piramide voor het masker.
3. Bouw een nieuwe piramide op basis van de *Laplaciaanse* piramides waarbij we de knopen van de *Gaussiaanse* piramide gebruiken als gewichten.

4. Bekom het nieuwe beeld door vergelijking 3.8 toe te passen op de nieuwe piramide.

3.2.3 Verwijderen van schaduwen

Omdat belichting veel problemen kan veroorzaken, wordt er hier een algoritme voorgesteld om één van de meest voorkomende problemen, namelijk het voorkomen van ongewenste schaduwen, op te lossen. Schaduwen kunnen er binnen de beeldverwerking namelijk voor zorgen dat segmentatie- of herkenningsalgoritmen falen. De methode beschreven in [12] komt erop neer dat we een illuminatie-invariant schaduwvrij beeld afleiden. Dit beeld wordt vervolgens gebruikt om samen met het orginele beeld de schaduwranden te bepalen. Vervolgens wordt het gradiëntveld van het orginele beeld op die posities aangepast waarna we na reconstructie een schaduwvrij beeld bekomen. Het grootste probleem bij deze techniek is het bekomen van een illuminatie-invariant beeld. Hiervoor is er immers een gecalibreerde camera nodig. Het resultaat van deze techniek kan je zien op figuur 3.5. De eerste foto geeft het orginele beeld, de tweede het invariante beeld en het derde het gereconstrueerde schaduwvrije beeld.

Figuur 3.5: Verwijderen van schaduwen (*)

3.2.4 Detail halen uit donkere beelden

In [16] wordt een opsplitsing voorgesteld in detail en intensiteit. Deze splitsing kan dan gebruikt worden om bepaalde beeldaanpassingen te automatizeren. De motivatie voor de ontwikkeling van deze methode lag bij de film *102 Dalmatians* of de *102 Dalmatiërs*, waarin er één pup voorkwam zonder vlekken. Omdat er geen geschikte verf was, bleef er voor Disney maar één oplossing over: het wegwerken van de vlekken om de zoveel frames. Het verwijderen van deze vlekken was een veel groter werk dan verwacht (40 artiesten gedurende 8 maanden).
Eens de geselecteerd regio is opgesplitst in detail en intesiteit, vervangen we de intensiteitswaarde in deze regio door een interpolatie van de intensiteit van de randen. Dit komt neer op het oplossen van een Laplace vergelijking. Vervolgens brengen we detail en intensiteit terug samen om het aangepaste beeld te bekomen (zie figuur 3.6).

Figuur 3.6: Detail halen uit donkere beelden (*)

3.2.5 Textuursynthese

Waar het oplossen van de Poisson vergelijking (zie vergelijking 3.1) een geleide interpolatie voorstelt waarbij de gebruiker een vectorveld specifieerd, zijn er vele geautomatizeerde interpolatiemethoden die enkel gebruik maken van de kennis van de randen [3] of die gebruik maken van stalen om een regio te vullen. Een voorbeeld van een sample-gebaseerde manier is *Image Quilting*[9]. We gaan in Hoofdstukken 4 en 5 hier dieper op in.

3.3 Naadloos klonen

We komen nu terug op de essentie van dit hoofdstuk. De eerste tool die we bespreken is het naadloos klonen. Als we het beeld B of een deel hiervan in beeld D willen invoegen, zullen we naar B refereren als het bronbeeld en naar D als het doelbeeld. De regio van het naadloos klonen wordt nog steeds aageduid met Ω.

3.3.1 Invoegen gradiënten

De natuurlijke keuze voor het vectorveld v zoals in formule 3.1 en 3.2 is het gradiëntveld van het bronbeeld:

$$v = \nabla B \tag{3.9}$$

met andere woorden, als we dit discreet gaan toepassen stellen we:

$$voor\ elke\ \ < p, q >,\ v_{pq} = B_p - B_q \tag{3.10}$$

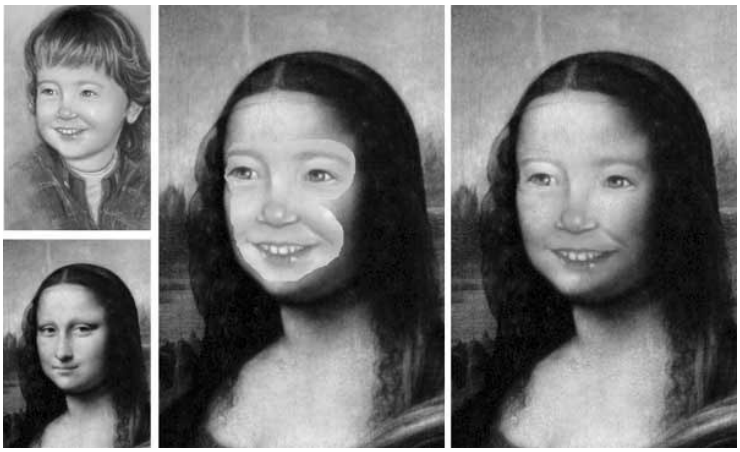
Een andere manier is, het gradiëntveld op voorhand te berekenen met volgende formule:

$$v(x, y) \approx (B(x + 1, y) - B(x, y), B(x, y + 1) - B(x, y)) \tag{3.11}$$

en vervolgens $v_{p,q}$ gelijk te stellen aan $-div\mathbf{v}$ met de divergentie zoals in formule 2.15. De bekomen tool kunnen we nu op verschillende manieren aanwenden.

In de eerste plaats kunnen we een aantal ongewenste features wegwerken en objecten naadloos invoegen. In figuur 3.7 verbergen we de rimpels door een niet gerimpeld deel huid over de rimpels naadloos te klonen en in figuur 3.8 geven we een voorbeeld van hoe een bronbeeld naadloos kan ingevoegd worden. In figuur 3.8 zien we ook duidelijk dat het bronbeeld op een moeiteloze manier geselecteerd is. De positionering moet in dit geval echter wel precies zijn.

Figuur 3.7: Het verbergen van ongewenste features



Figuur 3.8: Naadloos invoegen

Een andere mogelijkheid is het het uitwisselen van texturen. Hierbij kan het zijn dat we slechts een deel van de broninhoud willen overbrengen. Het meest voorkomende hierin is dat we wel het patroon van de textuur willen overbrengen maar niet de kleur. Dit kan opgelost worden door het bronbeeld monochroom te maken voordat we het klonen toepassen. In figuur 3.9 zien we in de bovenste rij de twee orginele beelden. In de tweede rij zien we in de eerste figuur gewoon

dit kan gediscretizeerd worden als:

$$voor\ alle\ x \in \Omega,\ v_{pq} = \begin{cases} D_p - D_q\ als\ |D_p - D_q| > |B_p - B_q|, \\ B_p - B_q\ anders \end{cases} \tag{3.13}$$

Het resultaat zien we in figuur 3.10. De bovenste tekst is ingevoegd met de gewone methode, de tweede door het lineair mengen van de gradiënten en de onderste door formule 3.13 toe te passen.

Figuur 3.10: Mengen van geadiënten

Het invoegen van semi-transparante objecten zien we in figuur 3.11 en een voorbeeld van het oplossen van het bleeding effect zien we in figuur 3.12.

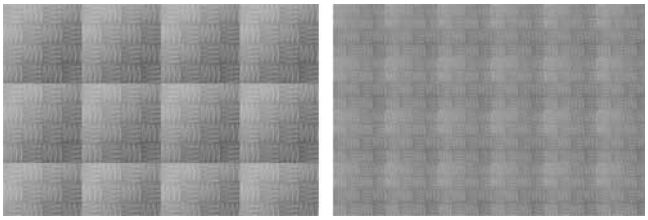
3.4 Naadloze beeldaanpassingen

In de vorige sectie was het vectorveld v volledig of gedeeltelijke afhankelijk van het gradiëntveld van het bronbeeld. Een hele set van tools kan nu afgeleid worden door het vectorveld afhankelijk te maken van het gradiëntveld van het orginele beeld. Gebaseerd op dit idee introduceren we nu: *Locale kleurverandering*, *Locale belichtingsveranderingen* en *Naadloze betegeling*.

3.4.1 Locale kleurverandering

De Poisson vergelijking is ook een krachtige tool voor het manipuleren van kleuren. Gegeven een orgineel kleurbeeld en een selectie Ω, twee verschillende gekleurde versies van dit beeld kunnen dan naadloos gecombineerd worden. Eén

van B, zodat tegenovergestelde zijden van het rechthoekig domein overeenkomen met identieke *Dirichlet* voorwaarden. We noteren het beeld D als het doelbeeld. Dan stellen we: $D_{noord} = D_{zuid} = 0.5(B_{noord} + B_{zuid})$ en hetzelfde voor oostelijke en westelijke randen. Zie figuur 3.15.



Figuur 3.15: Naadloze betegeling

3.5 Mogelijke problemen

Uit observatie blijkt dat deze techniek toch niet steeds de verwachte resultaten oplevert. We onderscheiden hier twee verschillende problemen. Beide problemen doen zich voor bij het naadloos klonen maar kunnen ook opduiken bij andere tools. Het eerste probleem kan zich voordoen wanneer het bronbeeld ongewenste gradiënten heeft rondom het in te voegen object. De tweede soort problemen hebben te maken met kleurveranderingen die optreden om het naadloze effect te bekomen. We bekijken deze problemen nu in detail en stellen enkele mogelijke oplossingen voor.

3.5.1 Gradiëntproblemen

Doordat we willen dat de gebruiker het in te voegen object moeiteloos kan selecteren, is met mogelijk dat de omgeving rondom ongewenste informatie bevat. Als dit het geval, kan dit ongeweste resultaten opleveren. In sommige gevallen kan het mengen van gradiënten zoals in sectie 3.3.2 een verbetering opleveren. Is het doelbeeld echter constant in de regio van het invoegen, helpt dit echter niet. We moeten dus sommige gradiënten gaan verwijderen uit het gradiëntveld. We stellen dat het in te voegen object duidelijk te onderscheiden is van de achtergrond. Dit wil zeggen dat de randen van het object aanleiding geven tot grote gradiënten. In de veronderstelling dat de achtergrond van het bronbeeld eveneens grote gradiënten kan bevatten die echter steeds kleiner dan deze van het object, kunnen we de gebruiker van een tool voorzien. We gaan in het bronbeeld gradiënten negeren waarvan de norm onder een door de gebruiker gedefinieerde waarde liggen. Deze methode heeft echter ook nadelen:

1. Het zoeken naar de juiste waarde kan tijdrovend zijn.
2. Door het negeren van gradiënten gaat er detail verloren.

waarover we controle hebben zonder de naadloosheid aan te tasten is het gradientveld van het bronbeeld. Enkele poging om de gradiënten aan te passen om zo de orginele kleur dichter te benaderen, bleken zonder succes.

Hoofdstuk 4

Mozaiekbeelden

Onder *Mozaïekbeelden* verstaan we het samenvoegen van beelden tot een groter beeld zoals bij panoramafoto's. Eerst wordt er een inleiding gegeven op reeds gekende technieken. Vervolgens zoeken we een methode waarop we de Poisson vergelijking van vorig hoofdstuk kunnen aanpassen om een zo goed mogelijk resultaat te krijgen. We vergelijken dan de methoden in het spatiale domein met deze in het gradiëntdomein.

4.1 Gerelateerde technieken

Er zijn twee verschillende aanpakken voor het creëren van mozaïeken in de literatuur. De eerste methode minimalizeerd artefacten door de overgang tussen de twee beelden zacht te laten verlopen. Hierin hebben we enkele methoden zoals beschreven in *Eliminating Ghosting and Exposure Artifacts in Image Mosaics*[26] en *A multiresolution Spline With Application to Image Mozaics*[5]. De tweede soort algoritmen zijn de optimale naad algoritmen die op zoek gaan naar de curve in de overlappende regio waar de verschillen tussen de twee beelden minimaal zijn. Deze techniek wordt besproken in: *Image Quilting for texture synthesis and transfer*[9] en *Mozaics of scenes with moving objects*[7].

4.1.1 Uitvloeien

In [26] wordt er een methode voorgesteld om af te rekenen met problemen die kunnen opduiken bij het creëren van mozaïeken. Twee soorten artefacten in het bijzonder: *ghosting* en *belichtings* artefacten. *Ghosting* zal in sectie 4.1.4 dieper besproken worden. Onder belichtingsartefacten veronderstellen we zichtbare naden door een verschil in helderheid en kleurtoon. Veronderstel dat we twee beelden hebben waarvan we een panorama beeld willen maken maar er is een belichtingsverschil. In eerste instantie lijkt het vermengen van de kleurwaarden in de overlappende zone op basis van afstand een goede methode. Dit noemen we uitvloeien. Als de belichtingsverschillen echter groot genoeg zijn, zal de

panorama nog steeds een zichtbare naad hebben (zie figuur 4.1).

Figuur 4.1: Vermengen van beelden met grote belichtingsverschillen (*)

In [26] wordt een overgangsfunctie gespecifieerd die we toepassen op een inputbeeld zodat het meer op de buurtbeelden gaat trekken. Het reslutaat van deze methode is te zien in figuur 4.2.

Figuur 4.2: Vermengen van beelden met verzachting van overgangsfunctie (*)

4.1.2 De multiresolutie curve

Deze methode werd uitvoerig besproken in hoofdstuk 3. Mits enkele kleine aanpassingen kan deze ook gebruikt worden voor het creëren van mozaïekbeelden.

4.1.3 Textuursynthese

In hoofdstuk 3 werd *Image Quilting* reeds vermeld. [9] introduceert een optimaal naad algoritme. We willen de scheiding maken tussen de twee overlappende beelden waar de beelden het best overeenkomen. Dit kan makkelijk bekomen door het probleem achterwaarts te bekijken (dynamic programming). We vragen ons dus eerst af wat de beste beslissing is in de laatste stap, dan in de voorlaatste enzovoort.
Voor het vinden van de beste scheidingscurve gaan we als volgt te werk. Veronderstel beeld B_1 en beeld B_2 de twee beelden die we moeten combineren en dat deze overlappen op de verticale rand met de overlappende regio's als B_1^{ov} en B_2^{ov} (zie figuur 4.3). De eerste illustratie geeft de twee beelden en de overlapping, de tweede de gevonden ideale curve.

Figuur 4.3: Twee beelden die overlappen op de verticale rand

We definiëren nu het foutenoppervlak e als volgt:

$$e = (B_1^{ov} - B_2^{ov})^2 \tag{4.1}$$

Om nu de minimale scheidingscurve te vinden moeten we e doorlopen en de cumulatieve minimumfout E voor alle paden berekenen:

$$E_{i,j} = e_{i,j} + min(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}) \tag{4.2}$$

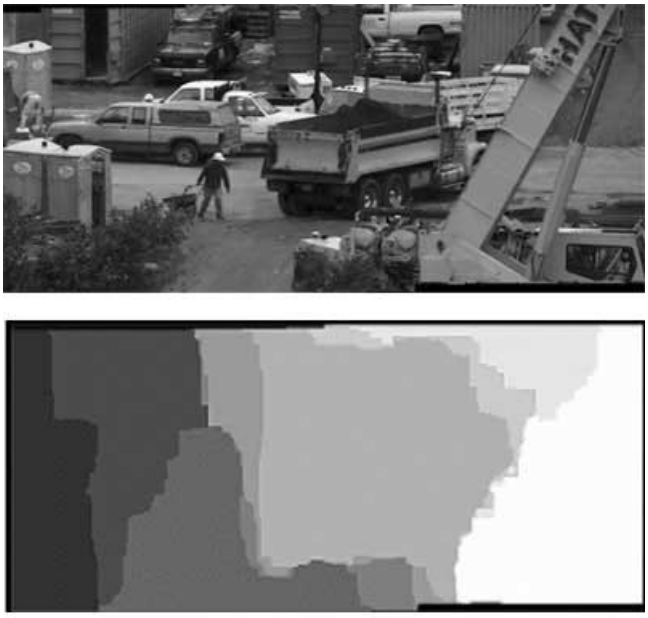
Op het einde zal het minimum van de laatste rij in E het einde aangeven van het kortste pad doorheen e. We kunnen nu terug komen op onze stappen en zo de beste scheidingscurve vinden. Dezelfde methode kan toegepast worden bij horizontale overlapping. Als er zowel horizontale als verticale overlapping is, kruisen de scheidingscurves en dan wordt het algemene minimum gekozen voor de scheiding.

4.1.4 Mozaïekbeelden met bewegende objecten

De techniek in [7] spits zich vooral toe op het combineren van beelden waarin bewegende voorwerpen voorkomen, welk probleem dit veroorzaakt en hoe dit opgelost kan worden. Het probleem wordt omschreven door de Engelse term *Ghosting* en is te zien in figuur 4.4. Wanneer de beelden die we willen samenstellen genomen worden, is er geen enkele garantie dat de objecten stationair blijven van één beeld naar het andere. Dit vormt een probleem in de overlappende regio's tussen beelden. Als de beelden gecombineerd worden door een samenstelling van de overlapende beelden dan zal de overlap zone van elk beeld dat bijdraagt tot de overlap zone net iets anders zijn. Hierdoor zal de uiteindelijke overlap een combinatie van pixels bevatten van verschillende delen van de scene.
Ook hier wordt een optimale naad algoritme geïntroduceerd: het Dijkstra algoritme. Dit is een kortste pad algoritme dat wordt toegepast op grafen. Gegeven een gerichte graaf G waarin de afstand tussen ieder tweetal verbonden punten van G tenminste 0 bedraagt, rekent het algoritme voor een bepaalde beginknoop $g_b \in G$ de kortste afstand uit tot alle punten van G. Met behulp van dit algoritme kunnen we ook het kortste pad tussen twee punten berekenen.
De gewichten voor het Dijkstra algoritme worden bepaald door het verschil tussen de twee beelden. Het gevonden minimale pad zal gebieden ontwijken waar de samensttellende beelden inconsistent zijn, dus ook de regio's waar objecten

Figuur 4.4: Voorbeeld van ghosting artefacten (*)

in beweging zijn (zie figuur 4.5). De bovenste figuur geeft het samengestelde beeld weer, terwijl de onderste figuur weergeeft hoe de figuur is samengesteld.



Figuur 4.5: Samenvoegen van beelden met behulp van Dijkstra (*)

4.2 Mozaïekbeelden in het gradiëntdomein

In [17] wordt een techniek uit de doeken gedaan die werkt op het gradiëntdomein. Het doel van een mozaïek algotitme kan als volgt beschreven worden: ten eerste moet het gecombineerde beeld zo gelijk mogelijk zijn aan de inputbeel-

den. Ten tweede, de naad waar de beelden aan elkaar genaaid worden, moet onzichtbaar zijn. De kwaliteit van de naad regio kan gemeten worden in het gradiëntdomein. Het gecombineerde beeld moet zo min mogelijk naad artefacten vertonen, met andere woorden, een naad mag geen nieuwe rand introduceren die niet voorkomt in de orginele beelden. De gradiënt van het gecombineerde beeld wordt dus vergeleken van deze van de input beelden. Dit vermindert het effect dat veroorzaakt wordt door inconsistenties tussen beelden.
[17] beschrijft twee algoritmes. Als eerste GIST1 (Gradient Domain Image Stitching), waar het mozaïek beeld onmiddelijk wordt afgeleid van de afgeleiden van de input beelden. Het tweede algoritme, GIST2, beschrijft een twee-stappen proces voor het creëren van mozaïekbeelden.

Het GIST1 algoritme

Deze eerste benadering berekent het mozaïekbeeld door een kostfunctie E_p te minimalizeren. E_p is een maatstaf voor de ongelijkheid tussen de afgeleide van het mozaïekbeeld en de afgeleiden van de inputbeelden. Meer specifiek, laat I_1, I_2 twee uitgelijnde input beelden zijn. Laat τ_1 (τ_2 *resp.*) de regio zijn die enkel kan gezien worden in beeld I_1 (I_2 *resp.*) en laat ω de overlappende regio zijn zodat $\tau_1 \cap \tau_2 = \tau_1 \cap \omega = \tau_2 \cap \omega = \emptyset$. Laat W een gewogen masker beeld zijn. Het resultaat I van GIST1 is gedefinieerd als het minimum van E_p met respect tot $\hat{I}$:

$$E_p(\hat{I}, I_1, I_2, W) = d_p(\nabla\hat{I}, \nabla I_1, \tau_1 \cup \omega, W) + d_p(\nabla\hat{I}, \nabla I_2, \tau_2 \cup \omega, U - W) \quad (4.3)$$

waar U een uniform beeld is en $d_p(J1, J2, \phi, W)$ de afstand is tussen J_1, J_2 op ϕ:

$$d_p(J_1, J_2, \phi, W) = \sum_{q \in \phi} W(q) \left\| J_1(q) - J_2(q) \right\|_p^p \quad (4.4)$$

waar $\|.\|_p$ duidt op de l_p-norm. De ongelijkheid E_p tussen de beelden is gedefinieerd door de afstand tussen hun afgeleiden. In de overlappende regio ω, straft de kostfunctie E_p afgeleiden waar inconsistenties optreden. Op locaties waar zowel I_1 als I_2 kleine gradiënten hebben, straft E_p grote gradiënt waarden in het mozaïekbeeld. Deze eigenschap is bijzonder nuttig tegen valse naden. De invloed van de keuze van p op het resultaat wordt later besproken.

Het GIST2 algoritme

Een simpelere aanpak is om de afgeleiden van de input beelden aaneen te naaien:

1. Bereken de afgeleide voor de input beelden $\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y}, \frac{\partial I_2}{\partial x}, \frac{\partial I_2}{\partial y}$.
2. Combineer nu de afgeleide beelden om een veld $F = (F_x, F_y)$ te vormen. F_x wordt bekomen door het combineren van $\frac{\partial I_1}{\partial x}$ en $\frac{\partial I_2}{\partial x}$ en F_y wordt bekomen door het combineren van $\frac{\partial I_1}{\partial y}$ en $\frac{\partial I_2}{\partial y}$

3. Vind het mozaïek beeld wiens gradiënt het dichtst bij F ligt. Dit is equivalent aan het minimalizeren van $d_p(\nabla I, F, \pi, U)$ waarbij π de totale beeld regio en U een uniform beeld is.

In de tweede stap mogen we eender welk algoritme voor het creëren van een mozaïek gebruiken.

Welke methode gebruiken?

In de vorige secties hebben we twee stitching methodes voorgesteld. Meestal worden de resultaten visueel getest en de gekozen methode kan afhankelijk zijn van persoon tot persoon. Gebaseerd op een formele analyse kan er echter geconcludeerd worden dat GIST1 onder l_1 de beste resultaten oplevert. Dit volgt uit volgende stelling:

Stelling 1 Laat I_1, I_2 twee input beelden zijn voor het algoritme en veronderstel dat er een curve $x = f(y)$ bestaat, zodat voor elke $q \in (f(y), y), I_1(q) = I_2(q)$. Laat U een uniform beeld zijn. Dan de optimale naad oplossing I, zoals hieronder gedefinieerd, is een globaal minimum van $E_p(I, I_1, I_2, U)$ zoals in vergelijking 4.3 voor elke $0 < p <= 1$.

$$I = \{ \begin{array}{ll} I_1(x,y) & x < f(y) \\ I_2(x,y) & x \geq f(y) \end{array} \tag{4.5}$$

De stelling impliceert dat GIST1 onder l_1 even goed is als de optimale naad methoden als er een perfecte naad bestaat. Hierdoor zal GIST1 onder l_1 ook goede resultaten behalen bij geometrische slechte uitlijning zoals de optimale naad methoden. Het voordeel van GIST1 ten opzichte van optimale naad methoden is wanneer er geen perfecte naad kan gevonden worden, zoals bijvoorbeeld bij belichtingsverschillen.
Met behulp van volgende stelling kunnen we ook aantonen dat GIST1 onder l_2 en het uitvloeien van afgeleiden (GIST2) onder l_2 equivalent zijn:

Stelling 2 Laat I_1, I_2 twee input beelden zijn voor het algoritme en laat W een masker zijn voor het uitvloeien. Laat ω, de overlappende regio van I_1, I_2, het volledige beeld zijn. Laat I_{Gist} het minimum zijn van $E_2(I, I_1, I_2, W)$ zoals in vergelijking 4.3. Laat F het volgende veld zijn:

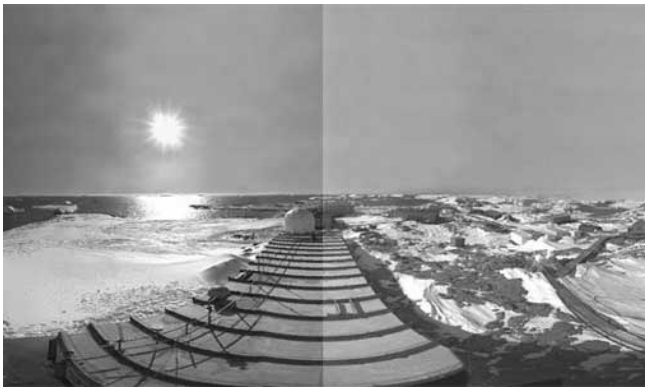
$$F = W(q)\nabla I_1(q) + (1 - W(q))\nabla I_2(q) \tag{4.6}$$

Dan is I_{Gist} het beeld met het gradiënt veld het dichtst bij F onder l_2.

Dit geeft ons inzicht in het verschil tussen GIST1 onder l_1 en onder l_2. Onder l_2 zal het algoritme de afgeleide mengen waardoor de textuur in de overlappende regio vager zal worden. Onder l_1 zal het algoritme zich gedragen als een optimaal naad algoritme waardoor inconsistenties verminderd worden. Het bewijs voor beide stellingen kan gevonden worden in [17].

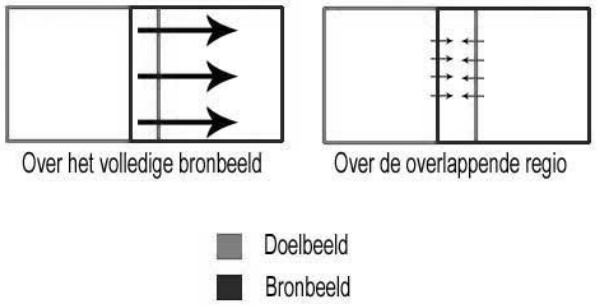
4.3 Mozaïekbeelden met behulp van Poisson

Voorgaande sectie vertoont veel overeenkomsten met de tools besproken in Hoofdstuk 3. Er zijn echter enkele fundamentele verschillen tussen GIST1 onder l_1 en de Poisson methode. In de eerste plaatst werkt GIST1 op gradiënten van beide beelden in de overlappende regio terwijl het naadloos klonen van vorig hoofdstuk enkel de gradiënt gebruikt van het bronbeeld en de intensiteiten van het doelbeeld. Een tweede verschil ligt in het gebruik van de norm. Beide verschillen leveren een zichtbaar resultaat op vooral als er sprake is van slechte uitlijning.
De aandachtige lezer zal opgemerkt hebben dat de methoden van hoofdstuk 3 makkelijk kunnen uitgebreid worden tot GIST2. Deze methode bepaalt immers eerst een vectorveld en gebruikt dit vervolgens voor de overgangszone te bepalen. We kunnen de gradiënten op verschillende manieren combineren.
In figuur 4.6 zien we een voorbeeld van een panoramafoto waarbij er een intensiteitsverschil is tussen de twee beelden.



Figuur 4.6: Intensiteitsverschillen

Om te beginnen zullen we de technieken van hoofdstuk 3 hierop toepassen. We kunnen dit echter al op twee manieren. Figuur 4.7 illustreert beide methoden. In de eerste illustratie interpoleren we over het ganse bronbeeld en gebruiken we enkel de grenzen van het doelbeeld. In het tweede geval interpoleren we enkel over de overlappende regio en gebruiken we de intensiteiten van zowel het bronbeeld als het doelbeeld. Deze aanpakken leveren ook andere resultaten. Wanneer de intensiteitsverschillen niet te groot zijn, zal de tweede methode een goed resultaat leveren en veel sneller. We berekenen immers enkel de pixels voor de overgangszone. Wanneer we echter te maken hebben met zeer grote intensiteitsverschillen of een zeer kleine overgangszone is het aan te raden te interpoleren over het volledige bronbeeld. Deze methoden werken zeer goed wanneer er enkele sprake is van intensiteitsverschillen tussen de beelden (zie figuur 4.8). Indien de beelden echter slecht uitgelijd zijn of er sprake is van ghosting artefacten leveren deze methoden minder goede resultaten.



Figuur 4.7: Illustratie Poisson methoden

Figuur 4.8: Poisson op Mozaïekbeelden

Eén van de nadelen van bovenstaande methoden is dat het enkel het gadientveld gebruikt van het bronbeeld. In een eerste poging om de gradiëntvelden te combineren, nemen we het gemiddelde van beide gradiëntvelden. Het resultaat hiervan is echter dat de textuur wordt uitgesmeerd bij kleine afwijkingen in de uitlijning (zie figuur 4.9).
Om tot betere resultaten te komen, gaan we twee technieken van het spatiale domein nu toepassen in het gradiëntdomein: het uitvloeien van de overlappende regio en een optimale naad algoritme. Onder uitvloeien verstaan we dat naargelang we in de overgangszone dichter bij het centrum van het bronbeeld komen, de gradiënten hiervan een groter gewicht krijgen dan deze van het doelbeeld. Het omgekeerde geldt natuurlijk ook. Het resultaat hiervan zien we in figuur 4.10.
De eerste foto geeft de orginele combinatie van twee beelden die slecht uitgelijnd zijn en een intensiteitsverschil vertonen. De tweede foto geeft de oplossing

Figuur 4.9: Gemiddelde van de gradiëntvelden

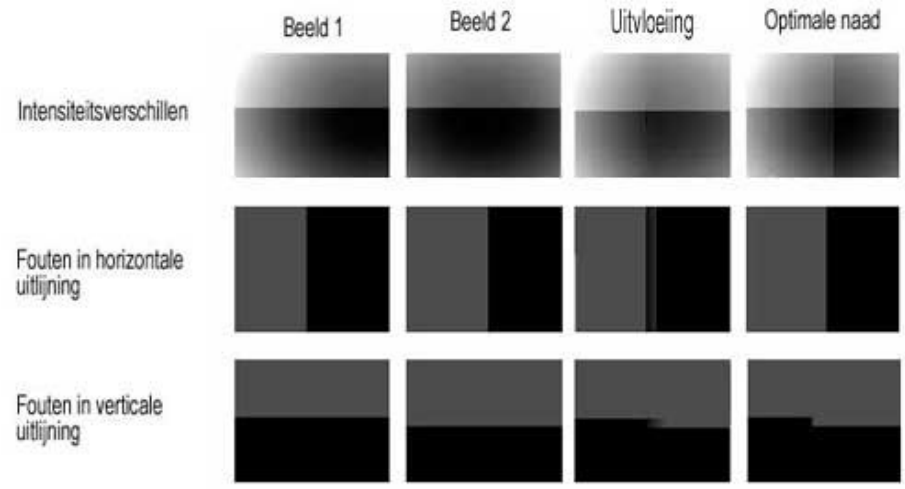
Figuur 4.10: Poisson met uitvloeiing

met Poisson en we zien dat de intensiteitsverschillen verdwenen zijn maar de slechte uitlijning is nog steeds zichtbaar. De derde foto geeft de oplossing weer door middel van uitvloeiing van de gradiëntvelden. We zien dat het resultaat reeds beter is maar de slechte uitlijning is nog steeds zichtbaar. Uit wat reeds besproken is weten we dat slechte uitlijning kan opgelost worden door toepassing van optimale naad algortimen. Deze hebben echter wel problemen met intensiteitsverschillen. We passen dus een optimaal naad algoritme toe op de gradiëntvelden van het doelbeeld en het bronbeeld en reconstrueren vervolgens de overlappende regio. Stelling 1 van sectie 4.2 laat ons reeds vermoeden dat dit betere resultaten zal opleveren. Figuur 4.11 laat ons het resultaat hiervan zien.

4.4 Vergelijkende studie

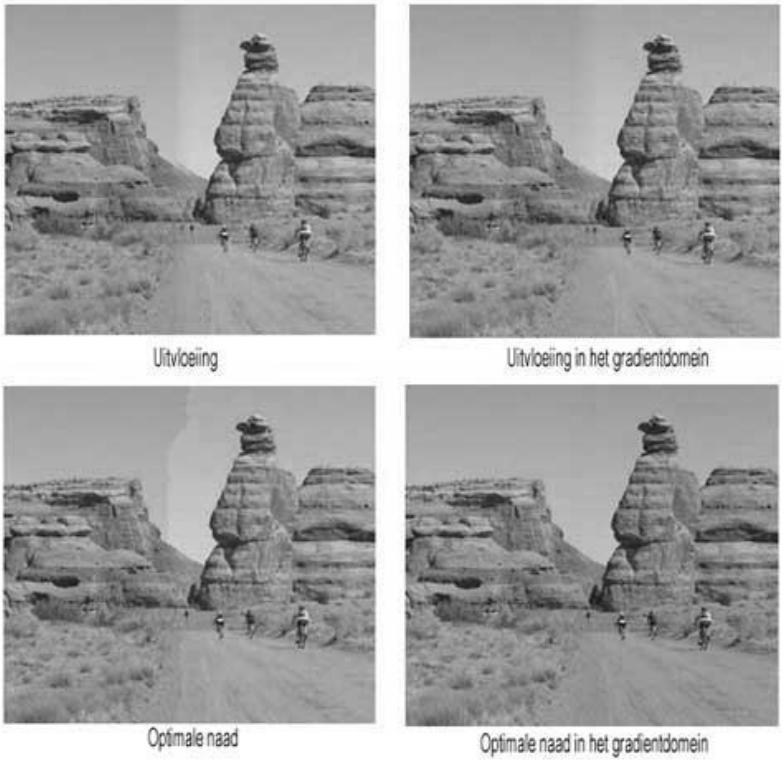
We zullen de Poisson technieken vergelijken met methoden die werken in het spatiale domein: uitvloeiings- en optimale naad algoritmen. In figuur 4.12 zien we de problemen die kunnen voorkomen bij methoden in het spatiale domein.

Figuur 4.11: Poisson met optimale naad



Figuur 4.12: Problemen in intensiteitsdomein

Bij optimale naad algoritmen zal er een naad zichtbaar blijven als er sprake is van intensiteitsverschillen. Uitvloeiing zal in tegenstelling tot optimale naad een dubbele rand creëren bij fouten in de horizontale uitlijning. Bij een slechte verticale uitlijning zien we dat uitvloeiing duidelijk beter resultaten oplevert dan de optimale naad algoritmes. We vergelijken nu deze methoden met de overeenkomstige methode in het gradiëntdomein (zie figuur 4.13).
Figuur 4.13 leert ons dat zowel overvloeiing als het optimale naad algoritme beter presteert in het gradiëntdomein. In de eerste rij is ook duidelijk dat zowel overvloeiing in het spatiale domein als in het gradiëntdomein artefacten vertoont bij slechte uitlijning. Onze voorkeur gaat dan ook uit naar een optimale naad algoritme in het gradiëntdomein. Zowel slechte uitlijning als intensiteitsverschillen als ghosting artefacten worden tot een goed einde gebracht.



Figuur 4.13: Vergelijking intensiteitsdomein en gradiëntdomein

Hoofdstuk 5

Inpainting en Textuursynthese

Voor we van start gaan, geven we eerst een definitie van inpainting en textuursynthese. Inpainting is de techniek om beelden aan te passen op een niet-detecteerbare manier. De doelen en applicaties zijn uiteenlopend, van het restoreren van beschadigde beelden tot het verwijderen van geselecteerde objecten. Onder textuursynthese verstaan we het genereren van een nieuwe textuur gebaseerd op een beperkte staal van een gegeven textuur. De technieken van textuursynthese kunnen ook toegepast worden als inpainting algoritmen. We zullen eerst een bespreking doen van enkele reeds bestaande inpainting technieken waarbij de onbekende zone volledig bepaald wordt door zijn randen. We doen een studie van de verschillende textuursynthese algoritmes en bekijken vervolgens hoe deze kunnen toegepast worden als inpainting algoritmes. Ook een combinatie van inpainting op basis van de randen en inpainting door middel van textuursynthese wordt kort aangehaald. Hierna gaan we opzoek naar een nieuwe techniek waarbij we met behulp van textuursynthese algoritmen proberen een vectorveld op te stellen voor de onbekende zone om deze met behulp van Poisson te reconstrueren.

5.1 Inpainting op basis van de randen

Het aanpassen van beelden op een niet-detecteerbare manier voor iemand die het orginele beeld niet kent is een beoefening die zo oud is als de kunst zelf. Hoe gaat dit in zijn werk:

1. Het globale beeld bepaalt hoe de onbekende regio gaat ingevuld worden met de bedoeling om de eenheid van het werk te restoreren.

2. De structuur van de regio die Ω omringd wordt voortgezet in de onbekende regio, de vormlijnen worden getekend door het verlengen van diegene die arriveren aan $\partial\Omega$.

3. De verschillende regio's binnen Ω, gedefinieerd door de vormlijnen, worden ingekleurd.

4. De kleine details worden getekend.

Dat inpainting zijn weg heeft gevonden naar de fotografie en de film is dan ook niet verwonderlijk. Digitale technieken worden meer en meer gebruikt, bijvoorbeeld voor het automatisch detecteren en verwijderen van krassen in films. Het algoritme besproken in [3] zal proberen onbekende regio's in te vullen met behulp van informatie rondom deze regio's.
Laat Ω de regio zijn die we moeten invullen en laat $\partial\Omega$ de grens van deze regio zijn. Het invullen wordt zo gedaan dat lijnen die arriveren op de randen van de regio's verlengd worden, maar toch de hoek die ze met de rand maken behouden. We tekenen dus van buiten naar binnen waarbij we de verlengende lijnen curven om te voorkomen dat ze met elkaar kruisen. Resultaten van deze techniek zijn te zien in figuur 5.1 en 5.2.

Figuur 5.1: Reconstructie (*)

Figuur 5.2: Verwijderen van objecten (*)

5.2 Textuursynthese

We zullen nu verscheidene textuursynthese algoritmen bekijken. Hierbij maken we een onderscheid tussen pixel-gebaseerde en fragment-gebaseerde algoritmen. Bij de eerste soort wordt pixel per pixel ingevuld terwijl bij de tweede soort we meerder pixels, gebundeld in een fragment, tegelijk gaan inkleuren. Fragmenten hebben meestal een rechthoekige (vensters) of een ronde structuur. Een pixel-gebaseerde methode vinden we terug in *Texture Synthesis by Non-parametric Sampling* [10], fragment-gebaseerde algoritmen in *Image Quilting*[9], *Fast and High Quality Overlap Repair for Patch-Based Texture Synthesis*[21] en *Synthesizing Natural Textures*[2].

5.2.1 Pixel-gebaseerd algoritme

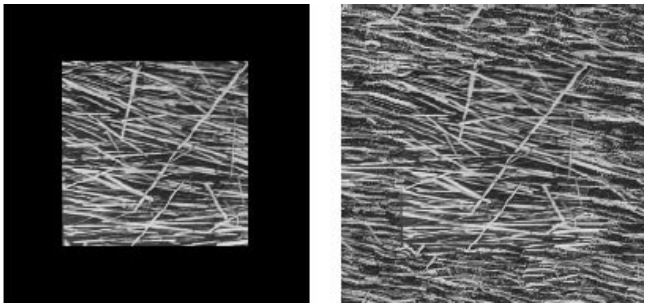
Het textuursynthese algoritme in [10] laat een nieuw beeld naar buiten groeien vertrekkende van gegeven staal en dit pixel per pixel. De methode is gebaseerd op een artikel uit 1948, *A Mathematical Theory of Communication*, waarbij een interessante manier besproken werd om geschreven Engelse tekst om te zetten in spraak door gebruik te maken van n-grams. Het idee is om een taal te modelleren als een algemene Markov ketting: een set van n opeenvolgende letters (of woorden) maken samen een n-gram en bepalen volledig de waarschijnlijkheidsdistributie van de volgende letter (of woord). We passen dit nu toe op texturen: de gekende omgeving van een onbekende pixel zal volledig de kleurwaarde van deze pixel bepalen. We zien een texture als een Markov Random Field (MRF) wat wil zeggen dat we veronderstellen dat de waarschijnlijkheidsdistributie van de kleurwaard van een pixel, gegeven de kleurwaarden van zijn omgeving, onafhankelijk is van de rest van het beeld. We bechrijven nu het algoritme in meer detail. Deze methode legt immers de basis voor wat volgt.

Synthetizeren van één pixel

Laat I een beeld zijn dat gesynthetizeerd wordt van een textuurmonster I_{sample}. Laat $p \in I$ een pixel zijn en laat $\omega(p) \subset I$ een rechthoekig regio, een venster, zijn van grootte w gecentreerd rond p. Laat $d(w_1, w_2)$ de afstand aanduiden tussen twee vensters. Laten we een moment veronderstellen dat alle pixels in I gekend zijn buiten p. Om de waarde van p te bepalen construeren we eerst een benadering van de voorwaardelijk waarschijnlijkheidsdistributie en nemen hiervan een staal. Steunend op het MRF model veronderstellen we dat p onafhankelijk is van $I \backslash \omega(p)$ gegeven $\omega(p)$. In deze implementatie zullen we nu vervolgens $\omega_{best} = min_{\omega}(d(\omega(p), \omega)) \subset I_{sample}$ berekenen om vervolgens alle ω_{best} en alle ω met $d(\omega(p), \omega) < (1 + \epsilon)d(\omega(p), \omega_{best})$ toe te voegen aan $\Omega(p)$ waarbij $\epsilon = 0.1$ in dit algoritme. De centrum pixels van alle vensters in $\omega(p)$ geeft ons een histogram voor p waaruit dan een staal kan genomen worden, ofwel uniform ofwel gewogen door d. Nu moeten we enkel nog d definiëren. Hiervoor nemen we de standaard definitie voor afstand, enkel afhankelijk van hoe de kleurwaarde gedefinieerd is, verminigvuldigd met een twee-dimensionale Gaussiaanse kernel.

Het algoritme

In vorige sectie hebben we verondersteld dan alle pixels buiten p gekend zijn, dit is echter zelden het geval. Dus om een pixel te synthetizeren moeten we in de praktijk enkel rekening houden met de pixels van $\omega(p)$ die gekend zijn en vervolgens de afstand tussen de twee vensters te normalizeren door te delen door het aantal gekende pixels. In figuur 5.3 zien we het resultaat van deze aanpak.



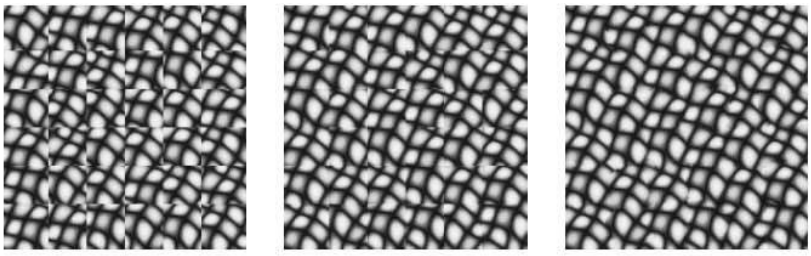
Figuur 5.3: Textuursynthese door middel van pixel-gebaseerde methode (*)

5.2.2 Fragment-gebaseerde algoritmes

Image Quilting

Deze methode of delen ervan werden reeds aanghaald in hoofdstukken 3 en 4. We bekijken deze techniek nu in zijn orginele context en beschrijven het algemene algoritme. In tegenstelling tot het vorige algoritme wordt een fragment-gebaseerde aanpak voorgesteld.
Laten we de eenheid van het synthetizeren definiëren als B_i. Dit is een vierkant blok, een fragment, van een door de gebruiker bepaalde grootte, dat afkomstig is uit de set S_b van alle overlappende blokken van de inputtextuur. Om nu een nieuwe textuurbeeld te synthetizeren, laten we in de eerste plaats het beeld betegelen met blokken willekeurig genomen uit S_b. Het resultaat hiervan is zichtbaar in figuur 5.4a. Hoeveel we de randen ook gaan verzachten, het zal altijd duidelijk blijven dat de blokken niet in elkaar passen. In de volgende stap zullen we een overlap introduceren bij het plaatsen van de blokken. In plaats van de blokken willekeurig te kiezen gaan we nu op zoek naar blokken in S_b die overeenkomen met zijn buren in de overlap regio. Figuur 5.4b geeft hier een voorbeeld van en is een duidelijk verbetering ten opzichte van het vorige resultaat maar nog steeds niet voldoende. Vervolgens wordt een optimale naad algoritme toegepast op de overlappende regio's (zie hoofdstuk 4).
Het complete quilting algoritme kan nu als volgt worden samengevat:

- Ga doorheen het beeld dat gesynthetizeerd moet worden in raster scan volgorde in stappen van één fragment (min de overlap).



Figuur 5.4: Textuursynthese door middel van Image Quilting

- Voor elke locatie, zoek in de inputtextuur naar fragmenten die voldoen aan de overlap voorwaarde binnen een bepaalde tolerantie. Kies willekeurig één van deze fragmentn.
- Bereken het foutenoppervlak in de overlappende regio. Vind het minimum pad en maak zo de grens van het nieuwe fragment. Kopieer het fragment naar de nieuwe textuur.

De enige parameter die de gebruiker kan istellen is de grootte van de fragmenten. De fragmenten moeten groot genoeg zij om relevante structuren te kunnen detecteren maar klein genoeg zodat de intersectie tussen deze structuren overgelaten wordt aan het algoritme. Een soort gelijke techniek wordt ook besproken in [18].

Verbetering van de overlappende regio bij fragment-gebaseerde methoden

Het algoritme besproken in [21] richt zich vooral op het verbeteren van de overlappende regio tussen twee fragmenten. Deze methode kan dan ook gezien worden als een uitbreiding van de hierboven vermelde fragment-gebaseerde algoritmen. De fragment-gebaseerde methoden hierboven zullen in sommige gevallen artefacten vertonen. Een oplossing is om slechte pixels te gaan repareren in de overlapende regio. Gegeven een inputtextuur en het doel om een outputtextuur te genereren van willekeurig grootte. Het algoritme kan dan informeel als volgt beschreven worden:

- Voor elke outputtextuur fragment
  - Vind het beste fragment in de inputtextuur door te vergelijken op de overlappende zone.
  - Bereken de fout in de overlappende zone. Duid elke pixel aan waarvan de fout een bepaalde tolerantiegrens overschreidt.
  - Bereken een volgorde voor deze ongeldige pixels.
  - Hersynthetizeer de ongeldige pixels.

Het hersynthetizeren kan gebeuren met behulp van een pixel-gebaseerde methode.

Synthetizeren van natuurlijke texturen

In [2] wordt er een textuursynthese methode voorgesteld die zich vooral toespits op een bepaalde klasse van natuurlijke texturen. In deze klasse zitten onder meer patronen bestaande uit kleine objecten met bekende doch ongelijke vorm zoals bijvoorbeeld een bloemenveld. Het algoritme is gebaseerd op dit van Wei en Levoy [27]. Hoewel het Wei en Levoy algoritme goed werkt voor zeer vele textures levert het geen goede resultaten voor patronen bestaande uit kleine objecten met bekende doch ongelijke vorm. Er wordt een aanpassing van het Wei en Levoy algoritme voorgesteld die naast betere resultaten voor deze patronen ook veel sneller werkt.

5.3 Inpainting door middel van textuursynthese technieken

Er bestaan reeds wat inpainting algoritmes die gebaseerd zijn op textuursynthese. De algoritmen in [8] en [1] zijn hier voorbeelden van. Deze stellen een iteratief proces voor dat fragmenten gebruikt uit de zichtbare delen om het beeld te reconstrueren. Om het vervolledigen goed te laten verlopen is er relevante informatie nodig binnen het beeld. Dit wil zeggen dat er overeenkomsten binnen het beeld moeten voorkomen. Beide methoden houden geen rekening met de globale context van het beeld en kunnen hierdoor onnatuurlijke resultaten creëren. We zullen beide technieken bespreken maar ons vooral toespitsen op de laatste.

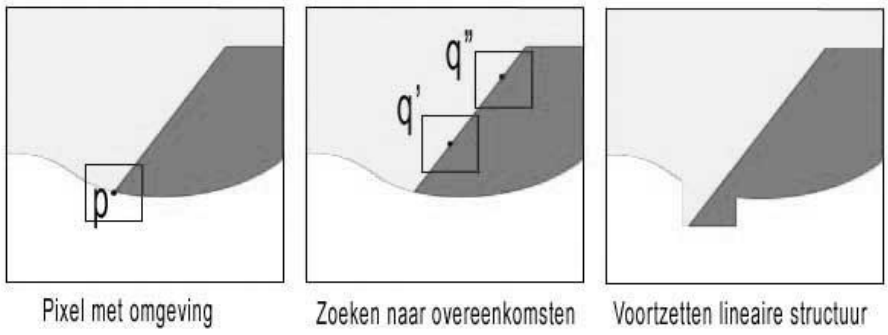
5.3.1 Interpolatie met toevoeging van details

In [8] wordt een algoritme beschreven dat een fragment-gebaseerde aanpak gebruikt voor inpainting. Er wordt gesteld dat de zone die ingevuld moet worden, is aangeduid door de gebruiker of een α kanaal is bekomen met een matting tool zoals beschreven in [24]. Hiervan berekenen we de inverse *matte* $\bar{\alpha}$ die het beeld in drie regio's opsplitst: de gekende regio waar $\bar{\alpha}_i = 1$, de ongekende regio waar $\bar{\alpha}_i = 0$ en mogelijk een grijze regio waar $0 < \bar{\alpha}_i < 1$. Deze inverse *matte* definieert het vertrouwen dat we hebben in elke pixel. In het begin is het vertrouwen in de onbekende zone laag, maar zal toenemen naarmate het invullen vordert. Een benadering is gegenereerd door een simpel verzachtingsproces in de gebieden waar we weinig vertrouwen hebben. Vervolgens wordt deze benadering uitgebreid door details te nemen uit gebieden met een groter vertrouwen. Al deze processen worden gerealizeerd op fragment niveau waarbij een fragment gedefinieerd is als een ronde omgeving rond een pixel. Het resultaat zien we in figuur 5.5

Figuur 5.5: Fragment-based Image Completion (*)

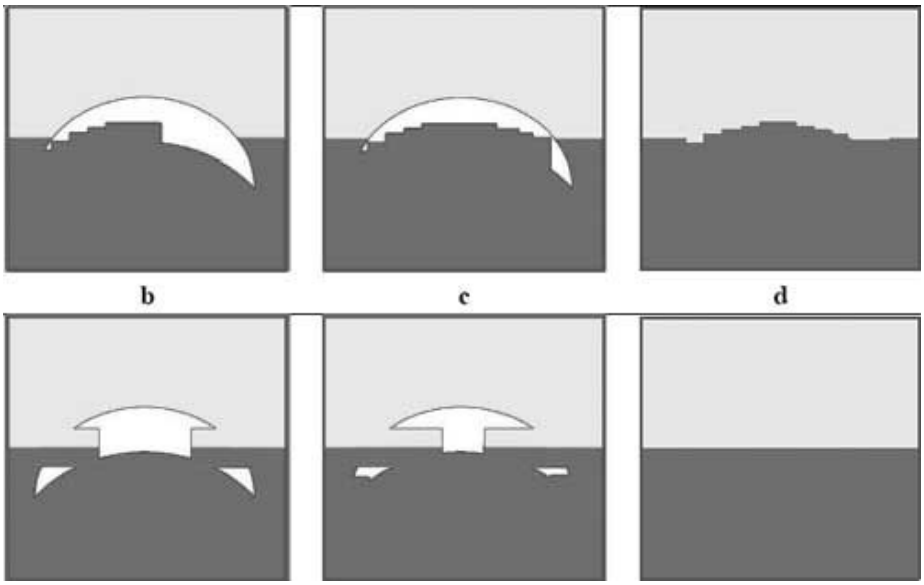
5.3.2 Fragment-gebaseerde inpainting met voortzetting van lineaire structuren

In [1] hechten we net als in sectie 5.1 meer waarde aan lineaire structuren binnen het beeld. Maar de lineaire structuren rondom de doelregio hebben enkel een invloed op de volgorde van het opvullen. Hierdoor wordt er een algoritme voorgesteld dat de efficiëntie en kwaliteit heeft van textuursynthese maar ook rekening houdt met de omliggende lineaire structuren. Het algoritme is gebaseerd op twee observaties. In de eerste plaats dat de textuursynthese methoden voldoen om lineaire structuren voort te zetten. Dit zien we in figuur 5.6. Hieruit volgt dat we dus geen speciale methode nodig hebben om de lineaire structuren af te handelen. De tweede observatie laat zien dat de volgorde van het invullen



Figuur 5.6: Voortzetting lineaire structuren

een belangrijk punt is. Dit is goed zichtbaar in figuur 5.7.
Uit deze observaties volgt het volgende algoritme. Er wordt verondersteld dat de in te vullen regio, Ω, door de gebruiker is aageduid. Het beeld min Ω wordt als Φ genoteerd en de rand van het doelgebied als $\delta\Omega$. Net als bij de textuursynthese algoritmen hierboven wordt er een venster gedefinieerd. Elke pixel houdt een kleurwaarde en een vertrouwensniveau bij welke weergeeft hoeveel vertrouwen we hebben in de juistheid van deze pixel. In het begin zullen gekende pixels een vertrouwenswaarde hebben van 1 en onbekende een vertrouwenswaarde van 0. Gedurende het algoritme krijgen vensters langs de rand een prioriteitswaarde toegewezen welke de volgorde van invullen zal bepalen.



Figuur 5.7: Volgorde van kritisch belang (*)

Berekenen van de prioriteiten

De prioriteit van een venster Ψ_p met middelpunt p kan gedefinieerd worden als $P(p) = C(p)D(p)$ waarbij $C(p)$ de vertrouwensterm is en $D(p)$ de data term is. Beiden zijn als volgt gedefinieerd:

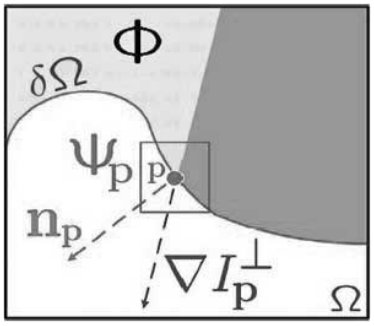
$$C(p) = \frac{\sum_{q \in \Psi_p} C(q)}{|\Psi_p|} \tag{5.1}$$

$$D(p) = \frac{\nabla I_p^{\perp} . n_p}{\alpha} \tag{5.2}$$

waarbij $|\Psi_p|$ het gebied is van Ψ_p en α een normalizatiefactor is ($\alpha = 255$ voor een grijswaardenbeeld), n_p is de eenheidsvector loodrecht op $\delta\Omega$ in het punt p en $\nabla I_p^{\perp}$ is de vector orthogonaal ten opzichte van de linaire structuur in het venster. Dit alles wordt geïllustreerd in figuur 5.8.
De vertrouwensterm zal zorgen voor een concentrisch volgorde. Als het vullen vordert zullen pixels in de buitenste lagen een hogere vertrouwenswaarde hebben en daardoor eerder worden ingevuld. De dataterm $D(p)$ is een functie die de sterkte van de lineaire structuur die $\delta\Omega$ raakt aangeeft. Deze term verhoogt de prioriteit naargelang de sterkte van de lineaire structuur.

Invullen van het venster

Eenmaal de prioriteit van alle vensters langs $\delta\Omega$ zijn berekent, zoeken we het venster Ψ_p met de hoogste prioriteit. Net als in de textuursynthese algoritmen



Figuur 5.8: Illustratie van de gebruikte notatie

wordt het beeld doorzocht naar het venster dat het best overeenkomt met Ψ_p. Vervolgens worden enkel de onbekende pixels ingevuld.

Aanpassen van het vertrouwensniveau

Het enige wat nu nog moet gebeuren is het updaten van het vertrouwensniveau van de pixels in Ψ_p. Dit gebeurd als volgt:

$$C(q) = C(p) \; \forall q \in \Psi_p \cap \Omega \tag{5.3}$$

Hierdoor zal naarmate het invullen vordert het vertrouwensniveau dalen om aan te duiden we minder zeker zijn van de kleurwaarden van de pixels in het midden van de in te vullen regio. Figuur 5.9 geeft het resultaat weer van deze methode.

Figuur 5.9: Verwijderen van een object (*)

5.4 Gecombineerde technieken

Er bestaan ook combinaties van technieken. Een voorbeeld hiervan zien we in *Simultaneous Structure and Texture Image Inpainting* [19]. De techniek beschreven combineert inpainting op basis van randen en texuursynthese. Het idee is van het beeld op de splitsen in de som van twee functies met verschillende

eigenschappen en dan elk van deze functies apart te reconstrueren. De eerste functie stelt de onderliggende structuur voor en de tweede functie de textuur en ruis. De eerste functie wordt gereconstrueerd met een inpainting algoritme op bais van randen, terwijl de tweede gereconstrueerd wordt met een textuursynthese algoritme. Het orginele beel wordt dan gereconstrueerd door deze twee subbeelden terug op te tellen. Dit principe wordt geïllustreerd in figuur 5.10. In deze figuur geeft het eerste figuur het orginele beeld voor en rechts daarvan zien we de oplossing door middel van het hier besproken algoritme. In de tweede rij zien we de opsplitsing tussen structuur en textuur. De derde rij laat de aparte oplossingen zien.

Figuur 5.10: Resultaat en werkwijze van gecombineerd algoritme

Aangezien zowel inpainting als textuursynthese algortimen hierboven al besproken zijn, gaan we hier niet verder op in.

5.5 Met behulp van Poisson

We passen nu bovenstaande technieken toe in het gradiëntdomein. We gaan ervan uit dat de gebruiker de in te vullen regio heeft aangeduid. Waar we in het spatiale domein steeds de kleurwaarde van een onbekende pixel gingen benaderen op basis van de kleurwaarde van zijn omgeving, gaan we nu op zoek naar het gradiëntveld in de onbekende regio. Eénmaal deze gevonden passen we de technieken van hoofdstuk 3 toe om de zone naadloos in te vullen. Omdat de methode in 5.2.1 de basis vormt voor vele textuursynthese algoritmen zullen we dit algoritme als eerste omzetten naar het gradiëntdomein. Vervolgens zullen we toewerken naar de methode besproken in sectie 5.3.2. Tenslotte zullen we nog enkele mogelijke verbeteringen voorstellen. Net als bij textuursynthese waar we een onderscheid maakten tussen pixel-gebaseerde en fragment-gebaseerde methoden, kunnen we hier spreken van gradiënt-gebaseerde en gradiëntveld-gebaseerde algoritmen.

5.5.1 Gradiënt-gebaseerd algoritme

Het textuursynthese algorimte van sectie 5.2.1 kan makkelijk aangepast worden om een ombekende regio in te vullen. Stel de notatie zoals in sectie 5.3.2 waarbij Ω de in te vullen regio is en $\delta\Omega$ de rand hiervan. Het volledige beeld definiëren we als I en enkel de regio met de reeds gekende pixels met Φ. Het algoritme kan als volgt beschreven worden:

- Herhaal tot er geen onbekende pixels meer zijn
  - Zoek alle pixels van $\delta\Omega$ die een buur hebben waarvan de kleurwaarde bekend is
  - Voor elke gevonden pixel p:
    - Zoek in het beeld het venster dat het best overeenkomt met het venster rond p, Ψ_p
    - Vul de kleurwaard van p in

We vinden het venster dat het best overeenkomt Ψ_p door het verschil tussen Ψ_p en elk venster van Φ te berekenen en vervolgens het venster te nemen met het kleinste verschil. We werken met het RGB-kleurenmodel, wat er op neer komt dat we het verschil, E, berekenen in de drie kanalen en deze vervolgens optellen.

$$E = E_r + E_g + E_b \tag{5.4}$$

waarbij E_r het verschil is in het rode kanaal, E_g in het groene en E_b in het blauwe. We schakelen nu over van het spatiale naar het gradiëntdomein. De eerste stap van het algoritme in het gradiëntdomein is het berekenen van het gradiëntveld van het beeld. We hoeven nu enkel nog het verschil tussen twee vensters opnieuw te definiëren. Omdat het orginele beeld gedefinieerd is in het RGB-kleurenmodel, zal het gradiëntveld bestaan uit drie vectoren: $\vec{v}_r$, $\vec{v}_g$ en $\vec{v}_b$. Het verschil tussen twee vectoren, A en B, is gedefinieerd als de absolute

Hoofdstuk 6

Implementatie

In dit hoofdstuk zal mijn implementatie worden besproken. De doelstelling was om de verschillende technieken besproken in deze thesis eerst te implementeren in het spatiale domein en vervolgens om te zetten naar het gradiëntdomein waardoor we een vergelijkende studie konden doen. Het framework is gemaakt in C++ met behulp van Microsoft Visual Studio 6.0. Qt wordt gebruikt voor de user interface. We zullen nu de implementaties, hoofdstuk per hoofdstuk bespreken.

6.1 Hoofdstuk 1

- Verschillende methoden om gradiënt van een beeld te berekenen.
- Reconstrueren van een beeld op basis van het gradiëntveld.

Om het stelsel van linaire vergelijkingen op te lossen wordt gebruik gemaakt van de Gauss-Seidel methode met overrelaxatie waarbij we gebruik maken van de *VL: Vector Library* van *Andrew Willmott*[29]. De Gauss-Seidel iteratie methode is zelf ook geïmplementeerd maar omdat deze geen rekening hield met de sparseheid van de matrix, was het geheugenverbruik te hoog voor het oplossen van de Poisson vergelijking.

6.2 Hoofdstuk 2

- Naadloos klonen.
- Mengen van gradiënten.
- Locale kleurverandering, locale belichtingsveranderingen en naadloze betegeling

Het framework laat toe om met verschillende lagen te werken waarbij we dus twee lagen naadloos in elkaar kunnen klonen. Indien nodig kan de gebruiker ook een masker definiëren om de selectie te beperken.

6.3 Hoofdstuk 3

- Interpolatie over het ganse beeld.
- Interpolatie over de overlappende regio.
- Combineren van gradiëntvelden door nemen van het gemiddelde.
- Overvloeiing in het spatiale domein.
- Optimale naad algoritme in het spatiale domein.
- Overvloeiing in het gradiënt domein.
- Optimale naad algoritme in het gradiënt domein.

Een simpel masker werd gecreëerd om twee beelden te laten overvloeiien. Het geïmplementeerde optimale naad algoritme is dit beschreven in Image Quilting [9].

6.4 Hoofdstuk 4

- Pixel-gebaseerd textuursynthese algoritme.
- Fragment-gebaseerd textuursynthese algoritme.
- Aanpassen van pixel-gebaseerde textuursynthese voor inpainting.
- Fragment-gebaseerde inpainting.
- Gradiënt-gebaseerde inpainting.
- Gradiëntveld-gebaseerde inpainting.

Het pixel-gebaseerd textuursynthese algoritme is dit van *Efros en Leung*[10]. De implementatie van het fragment-gebaseerd textuursynthese algoritme is zoals beschreven in [18]. De fragment-gebaseerde inpainting methode wordt beschreven in [1] en sectie 5.3.2.

Bibliografie

[1] Kentaro Toyama Antonio Criminisi, Patrick Perez. Region filling and object removal by exemplar-based image inpainting. In *Microsoft Research*, 2003.

[2] Michael Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[3] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[4] Noel Black and Shirley Moore. Gauss-seidel method. `http://mathworld.wolfram.com/`.

[5] Peter J. Burt and Edward H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2(4):217–236, 1983.

[6] Henry Chu. Luminance of images. `http://www.cacs.louisiana.edu/~cice/lal/`, 2003.

[7] James Davis. Mosaics of scenes with moving objects. In *CVPR*, pages 354–360, 1998.

[8] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, 2003.

[9] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[10] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033, Washington, DC, USA, 1999. IEEE Computer Society.

[11] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 249–256, New York, NY, USA, 2002. ACM Press.

[12] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, pages 823–836, London, UK, 2002. Springer-Verlag.

[13] Rafael C. Gonzalez and Richard E. Woods. Image segmentation. In *Digital Image Processing*, pages 125–137. Prentice-Hall, 2001.

[14] Rafael C. Gonzalez and Richard E. Woods. Sharpening spatial filters. In *Digital Image Processing*, pages 125–137. Prentice-Hall, 2001.

[15] Larry Green. Vector calculus. `http://ltcconline.net/greenl/courses/202/202.htm`, 2001.

[16] J.P.Lewis. Lifting detail from darkness. In *ACM Siggraph 2001 Technical Sketch*, 2001.

[17] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *European Conference on Computer Vision (ECCV 04)*, 2003.

[18] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[19] Guillermo Sapiro Marcelo Bertalmio, Luminita Vese. Simultaneous structure and texture image inpainting. *Transactions on Image processing*, 12(8):882–889, 2003.

[20] John H. Mathews. Gauss-seidel method. `http://math.fullerton.edu/mathews/n2003/GaussSeidelMod.html`, 2004.

[21] Andrew Nealen and Marc Alexa. Fast and high quality overlap repair for patch-based texture synthesis. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 582–585, Washington, DC, USA, 2004. IEEE Computer Society.

[22] Robyn Owens. Classical feature detection. `http://www.mathworks.com/products/demos/image/detect_cell/morph2.html`, 1997.

[23] Patrick Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.

[24] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Trans. Graph.*, 23(3):315–321, 2004.

[25] Adrian Taga. The variational principle. `http://www.fysik4.fysik.uu.se/documents/LicBettan/node8.html`, 1999.

[26] Matthew Uyttendaele, Ashley Eden, and Richard Szeliski. Eliminating ghosting and exposure artifacts in image mosaics. In *CVPR*, pages 509–516, 2001.

[27] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[28] Eric W. Weisstein. Conservative field. `http://mathworld.wolfram.com/`.

[29] Andrew Willmott. Vl: Vector library. `http://www-2.cs.cmu.edu/~ajw/doc/vl.html`.