

Intuïtieve Manipulatie van Textuur

Johan Nulens

Promotor: Prof. dr. Philippe Bekaert

Begeleider: dr. Tom Mertens

Academiejaar 2004-2005

Eindverhandeling voorgedragen tot het behalen van de graad van
Licentiaat in de Informatica afstudeervariant Multimedia

School voor Informatie Technologie
Transnationale Universiteit Limburg
Diepenbeek, 23 augustus 2005

Dankwoord

Ik zou graag enkele mensen willen bedanken die een grote hulp zijn geweest tijdens het schrijven van deze thesis.

Eerst en vooral, mijn begeleider dr. Tom Mertens die me geduldig door het hele proces heeft geleid met vele suggesties, feedback en soms ook een waarschuwing over het tijdsbestek die ik ter harte genomen heb.

Verder wil ik mijn dank overbrengen aan Cedric Vanaken, talloze keren heeft hij me geholpen door mijn vragen te beantwoorden en de tijd genomen om dieper in het onderwerp te graven. Uiteraard wil ik ook mijn promotor, Prof. dr. Philippe Bekaert, bedanken.

Tenslotte wil ik nog mijn ouders en in het bijzonder Jo Vermeulen bedanken voor het nalezen van mijn thesis, zijn vele tips en kritiek zijn een grote hulp geweest.

Samenvatting

Textuur manipulatie is een wijd verspreide toepassing. Er bestaan vele tools die het toelaten om texturen te manipuleren maar deze zijn vaak te complex. Met deze thesis onderzoek ik enkele nieuwe veelbelovende methodes om textuur op een intuïtieve manier te implementeren. Om te beginnen maak ik een implementatie over textuurbewerkingen door zelf-similariteit. Met deze methode zal de gebruiker in staat zijn met één selectie het uitzicht van een textuur op een verscheidenheid van manieren kunnen veranderen. Nadien onderzoek ik ook hoe texturen fundamenteeler kunnen worden gemanipuleerd. Hierbij wordt gebruik gemaakt van een textuurklasse pseudo-reguliere texturen. We zien dat we op een eenvoudige manier een textuur kunnen modelleren aan de hand van de regulariteit van zijn geometrie en kleur. Zo zullen we in staat zijn nieuwe texturen te genereren waarbij de gebruiker een sterke controle heeft over het uitzicht van de nieuwe textuur. Eveneens zal deze techniek worden toegepast om texturen te synthetiseren naar model van de textuur. Verder worden enkele toepassingen bestudeerd van deze methodes zoals textuur vervanging en beeld analogieën.

Inhoudsopgave

Dankwoord	i
Samenvatting	ii
I Inleiding	5
1 Overzicht	6
1.1 Probleemstelling	6
1.2 Doel	6
1.3 Overzicht van deze thesis	6
2 Inleidende begrippen	7
2.1 Wat is Textuur?	7
2.2 Textuur manipulatie	7
2.3 Textuur synthese	7
II Literatuurstudie	8
3 Textuur synthese	9
4 Textuur bewerken door zelf-similariteit	14
4.1 Textuur inkleuren	14
4.2 Textuur klonen	16
4.3 Textuur warpen	17
5 Textuur analyse	19
5.1 Regulariteit van een textuur	19
5.2 Verband tussen irregulier en reguliere texturen	20
5.2.1 Transformatie van irreguliere naar reguliere structuur	20
5.2.2 Berekenen van het geometrisch deformatieveld	21
5.2.3 Dualiteit van het geometrisch deformatieveld	21
5.3 Kleur deformatie veld	22

5.4	Belichting deformatie veld	23
6	Pseudo-reguliere textuur manipulatie	26
6.1	Synthese van het geometrisch deformatieveld	26
6.2	Synthese van pseudo-reguliere texturen	27
6.3	Manipulatie	28
7	Textuur vervanging	30
7.1	Textuur vervanging m.b.v. deformatievelden	30
7.2	Textuur vervanging door dieptekaarten	30
7.2.1	Bepalen van normalen.	31
7.2.2	Patchen van het beeld	31
7.2.3	Patch oriëntatie	32
8	Deformatieveld analogieën	33
8.1	Beeld analogieën	33
8.2	Deformatieveld analogieën	34
III	Implementatie	35
9	Zelf Similariteit	36
9.1	Textuur bewerken	36
9.2	Textuur klonen	37
9.3	Textuur warpen	38
10	Irreguliere textuur manipulatie	40
10.1	Bepaling van het geometrische deformatieveld	40
10.2	Synthese van het geometrische deformatieveld	42
10.3	Synthese van de pseudo-reguliere textuur	43
10.4	Manipulatie van de pseudo-reguliere textuur	45
11	Uitbreidingen	48
11.1	Textuur vervanging	48
11.1.1	Het belichtingsdeformatieveld	48
11.1.2	Textuur vervangen	49
11.2	Deformatieveld analogieën	50
IV	Conclusie	51

Lijst van figuren

3.1	Efros en Lung textuur synthese	9
3.2	Matchen van omgevingen	10
3.3	Vergelijking Wei en Levoy met Efros en Lung	11
3.4	Ashikhmin textuur synthese: shifting	12
3.5	Ashikhmin: gecontroleerde synthese	13
4.1	Zelf-similariteit gebaseerde textuur editing	15
4.2	zelf similariteit: wijzigen globale sterkte en afstand	15
4.3	Multi-schaal omgevingen	16
4.4	Textuur klonen	16
4.5	Niet-lineaire magnificatieveld	17
4.6	Similariteit-gebaseerde warping	18
5.1	Regulariteit van een textuur	19
5.2	Reguliere vs. Irreguliere lattice	21
5.3	Geometrisch deformatieveld in HSV	22
5.4	pca decompositie	24
6.1	Deformatieveld synthese	26
6.2	Pseudo-reguliere textuur synthese	27
6.3	Textuur deformaties	28
6.4	Overzicht pseudo-reguliere textuur manipulatie	29
7.1	Voorbeeld: textuur vervanging	30
7.2	Textuur vervanging door dieptekaarten	31
7.3	Patch oriëntatie	32
8.1	Beeld analogie voorbeeld	33
8.2	Beeld analogie matching	34
9.1	Texture editeren voorbeeld 1	36
9.2	Texture editeren voorbeeld 2	37
9.3	Texture editeren voorbeeld 3	38
9.4	Texture editeren voorbeeld 4	38
9.5	Texture editing	39

9.6	Zelf-similariteit warping	39
10.1	Relateren van een textuur met zijn pseudo-reguliere oorsprong	41
10.2	Per-pixel deformatievelden	42
10.3	voorbeeld: G-scores	43
10.4	Toepassing geometrisch deformatieveld	44
10.5	PCA	45
10.6	PCA gesynthetiseerde textuur 1	46
10.7	PCA gesynthetiseerde textuur 2	46
10.8	Geometrische deformatie manipulatie	47
10.9	Kleurdeformatie manipulatie	47
10.10	Geometriedeformatie op een andere textuur	47
11.1	Belichtingsmap	49

Deel I

Inleiding

Hoofdstuk 1

Overzicht

1.1 Probleemstelling

Textuurmanipulatie is voornamelijk een artistieke bezigheid. Huidige grafische tools laten toe een grote verscheidenheid aan effecten aan te brengen maar dit kost veel tijd en moeite. Er is dus nood aan tools die de gebruiker toelaten om op een eenvoudige en efficiënte manier texturen te manipuleren.

1.2 Doel

Het doel van deze thesis is nieuwe methoden te onderzoeken die toelaten een textuur op een intelligente en intuïtieve manier te manipuleren. We vergelijken deze met bestaande methodes en maken een implementatie.

1.3 Overzicht van deze thesis

Eerst leiden we enkele begrippen in die gerelateerd zijn aan het probleem. Daarna geven we een overzicht van de verrichte literatuurstudie. Vervolgens bespreken we de implementatie van de verschillende onderdelen van de onderzochte methodes en enkele applicaties. Tenslotte eindigen we met een conclusie.

Hoofdstuk 2

Inleidende begrippen

2.1 Wat is Textuur?

Textuur is een veelzijdig begrip. Het kan repetitieve elementen in bv. een geluid(een ronkende motor), de voeling met een stof of het uiterlijk van een object beschrijven. Wij beschouwen een textuur als een 2D afbeelding van een object.

2.2 Textuur manipulatie

Praktisch gezien verstaan we hieronder het aanpassen van karakteriserende kenmerken van de textuur. Bv. het accentueren, verkleinen, vergroten, vervangen, toevoegen, verwijderen en de kleurinformatie aanpassen vallen hieronder.

2.3 Textuur synthese

Het proces om een nieuwe textuur aan te maken aan de hand van informatie uit een bestaande voorbeeld textuur noemt men textuur synthese. De nieuwe textuur is niet exact hetzelfde als de originele textuur maar voor het menselijke oog wel perceptueel dezelfde textuur.

Deel II

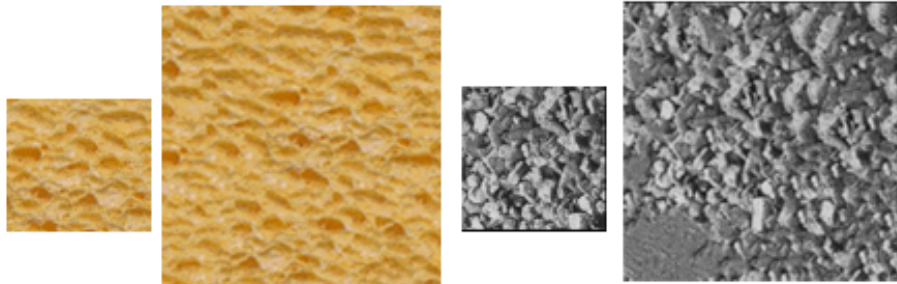
Literatuurstudie

Hoofdstuk 3

Textuur synthese

Textuur synthese is een veel gebruikte techniek in texture tools. In de laatste jaren zijn vele methoden ontwikkeld en vaak zijn deze methoden gebaseerd op hun voorgangers. Het basisidee achter deze algoritme is een nieuwe textuur te genereren die niet exact hetzelfde is maar wel alle kenmerken heeft van het voorbeeld image. De belangrijkste algoritmen worden kort voorgesteld.

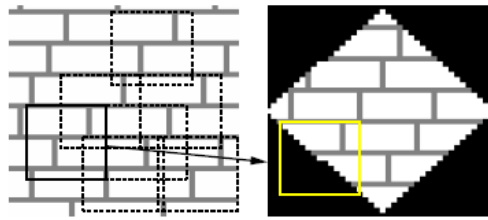
Een begrip dat vaak terugkomt is MRF of *Markov Random Field*. Wanneer een MRF wordt verondersteld voor een input beeld in een textuur synthese algoritme neemt men aan dat de probabilliteit van een pixel willekeurig is bepaald.



Figuur 3.1: (links) Een geslaagde textuur synthese. (rechts) Het algoritme verloopt verkeerd en produceert dan een verkeerd resultaat. (bron: [EL99])

Efros en Lung Een fundamenteel algoritme in textuur synthese is dat van Efros en Lung [EL99]. Het synthese proces start vanaf een initieël startpunt en groeit vandaar uit een nieuw beeld, één pixel tegelijk. De waarde voor een pixel wordt geschat door de reeds gesynthetiseerde bure

van de pixel te evalueren en te zoeken naar gelijkaardige omgevingen in het beeld. Dit wordt gedaan door een SSD (Squared Sum Difference) te nemen van de gesynthetiseerde burens. Dit wordt ook gewogen door een gaussiaanse kernel zodat verdergelegen burens minder invloed hebben op de keuze van de inkleuring van de pixel. De methode tracht zoveel mogelijk de lokale structuur te behouden maar kan niet echter altijd de globale structuur behouden.

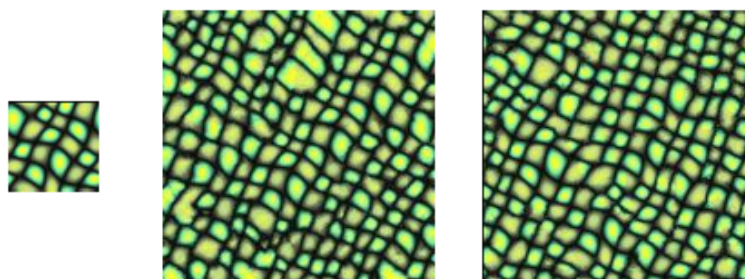


Figuur 3.2: Voor elke pixel met gevulde burens zoeken we de best gelijkende omgeving in het input sample. (bron: [EL99])

Wei en Levoy In [WL00] presenteren Wei en Levoy een efficient algoritme om textuur te synthetiseren. De basis van het algoritme, enkelvoudige resolutie synthese, lijkt veel op dat van Efros en Lung.¹ In scanline orde vult men het gesynthetiseerde beeld door de meest overeenkomende pixel uit het voorbeeld beeld te zoeken. Men doet dit door de SSD te nemen van de L-vormige omgeving van de nieuwe pixel en te vergelijken met alle L-vormige omgevingen uit het voorbeeld beeld.

Men probeert in deze paper twee problemen op te lossen. Het eerste probleem met textuur synthese algoritmen tot hier toe was dat structuren op grote schaal niet behouden bleven. Hiervoor introduceert men multiresolutie synthese. Multiresolutie synthese doorzoekt de structuur van het beeld op verschillende schalen. Een gaussiaanse pyramide wordt opgebouwd van het doel-en bronbeeld met twee of meer niveau's. Overeenkomstige omgevingen worden nu (exhaustief) gezocht in alle lagen van de pyramide. Ten tweede, exhaustieve matching is traag. Het zou goed zijn mochten we een datastructuur hebben die ons snel toelaat, gegeven een set omgeving pixels S en een nieuwe lege pixel Q , een waarde te vinden waarvoor de afstand tot Q minimaal is. Zo een datastructuur is TSVQ of *Tree-Structured Vector Quantization*. In essentie is dit een binaire datastructuur die vectors als input neemt, een spilvector berekent en vervolgens alle vectoren verdeelt over de twee takken volgens dit spilpunt. Wanneer de boom is opgebouwd kan

¹Het is zelfs zo dat velen denken dat het werk van Wei en Levoy gebaseerd is op dat van Efros en Lung, dit is echter niet zo en ze lichten dit ook toe op hun website.

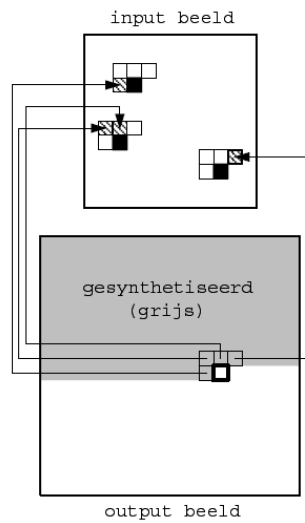


Figuur 3.3: (Links)input beeld, (midden)synthese met Efron en Lung (rechts) synthese met Wei en Levoy Alhoewel het verschil niet schokkend is, kan er wel opgemerkt worden dat het rechterbeeld een beter gereproduceerde structuur bevat. Dit is te danken aan de multiresolutie aanpak van Wei en Levoy.(bron: [WL00])

een query worden opgelost in $O(\log(S))$ wat natuurlijk veel sneller is als een exhaustive query met een complexiteit van $O(S)$.

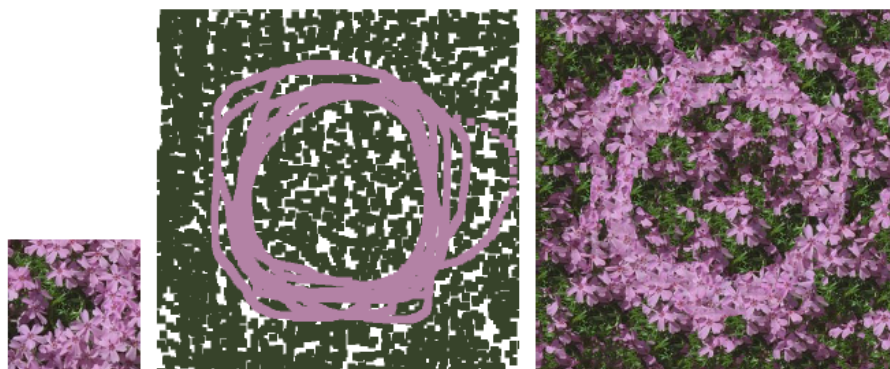
Ashikhmin Ashikhmin bemerkt in [Ash01] dat de methode voorgesteld door Wei en Levoy enkele gebreken heeft. Hij stelt dat onze perceptie gevoelig is voor randen, hoeken en andere speciale kenmerken en dat de simpele L_2 norm niet voldoet om perceptuele gelijkenissen tussen twee patches te garanderen. Bij beelden met hogere complexiteit zoals natuurlijke beelden faalt het algoritme van Wei en Levoy. De voornaamste problemen daarbij zijn wazige beelden en vooral randen die te zacht zijn. Meer nog, het TSVQ produceert beelden die nog waziger zijn (TSVQ zoekt een benaderende oplossing voor een pixel). Ashikhmin maakt de opmerking dat we in het algoritme van Wei en Levoy al pixels hebben gevonden in het voorbeeld beeld met omgevingen gelijkaardig aan verschoven huidige omgevingen in het output beeld. Hoe maken we gebruik van deze informatie? We slaan voor elke reeds gesynthetiseerde pixel een lokatie op uit het input beeld. Gebruik makend van deze lijst maken we een nieuwe lijst kandidaten uit de referenties van pixels uit de L-vormige omgeving van de te synthetiseren pixel. Deze kandidaatlijsten hebben gewoonlijk een lengte gelijk aan de grootte van de L-vormige omgevingen. Dit is voldoende als de grootte van de omgevingen overeenkomt met de grootte van de features. In de paper wordt ook een tool voorgesteld die meer controle geeft over de synthese. Het algoritme neemt een handgetekend beeld als tweede input en synthetiseert het beeld naargelang.

Patch-gebaseerde synthese Patch-gebaseerde methodes vereisen vaak een heel andere aanpak dan we tot nu toe gewoon zijn. Over deze methodes



Figuur 3.4: Ashikhmin: De kandidaatpixels bestaan uit verschoven referenties uit de L-vormige omgeving.(bron: [Ash01])

weiden we echter niet uit, er wordt enkel een korte omschrijving gegeven voor de volledigheid van dit hoofdstuk. Enkele varianten zijn chaos mozaïek [XGS00] en Image quilting [EF01]. Image quilting neemt het input beeld, verdeelt dit in blokken en plaats deze blokken dan random in het synthese beeld. Het komt er nu op neer de blokken aaneen te sluiten zodat buurblokken een naadloze rand hebben. Men vindt de best aansluitende blokken door een minimum kost pad te berekenen van de error op de overlappende gedeelten. Chaos mozaïek verdeelt het input beeld in tegels(patchen) alvorens aan de synthese te beginnen. Het doel is een beeld te creëren uit deze tegels met een equivalente distributie van de lokale features uit het input beeld. Globale features worden hier genegeerd omdat deze de visuele gelijkenis niet al te zeer aantasten en het algoritme hierdoor zeer snel kan worden. Tegels worden geplaatst door een iteratief algoritme uit het veld van deterministische chaos. Een gedetailleerde omschrijving van het algoritme zou ons te ver leiden en behoort niet tot het bereik van dit hoofdstuk.



Figuur 3.5: Ashikhmin: gecontroleerde textuur synthese.(bron: [Ash01])

Hoofdstuk 4

Textuur bewerken door zelf-similariteit

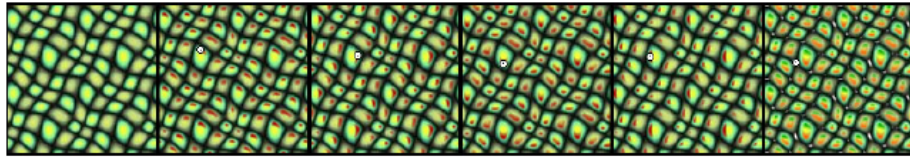
Voor het modelleren van scenes met een hoge complexiteit wordt de ontwerper vaak ondersteund door een verscheidenheid aan complexe tools die hem een grote vrijheid geven in het ontwikkelingsproces. Alhoewel deze methode werkt, wordt van de ontwerper een grote vakkundigheid verwacht en vaak is een lange opleiding en computerkennis vereist. Er is dus een nood aan tools die gemakkelijk en intuïtief te gebruiken zijn en toch een grote flexibiliteit hebben.

Er zijn een aantal tools ontwikkeld zoals textuur synthese tools die een nieuwe instantie van de textuur afleveren. Er bestaan textuurbewerkingstools en hybride varianten van textuur synthese en bewerking zoals bijvoorbeeld [Ash01] met zijn gebruikersgestuurde synthese (zie hoofdstuk 3).

We volgen dezelfde lijn als in de paper [BD02]. Deze nieuwe eenvoudige methode laat het toe om texturen interactief te manipuleren door een wijziging aan te brengen in één enkele pixel. Diezelfde wijziging wordt globaal toegepast op de hele textuur. De techniek gebruikt multi-schaal omgevingen om een overeenkomst te vinden. De wijziging van de gebruiker wordt dan gereproduceerd over pixels met een gelijkaardige omgeving. Dat is het hele concept van **zelf-similariteit**. De drie varianten van deze techniek zijn samengevat in de volgende secties.

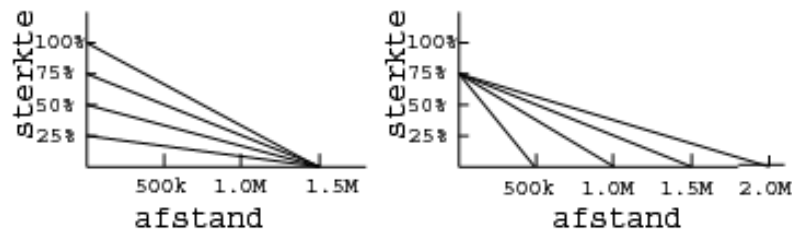
4.1 Textuur inkleuren

Zelf-similariteit wordt hier aangewend om de kleur van pixels over het hele beeld te wijzigen. De gebruiker selecteert een pixel en kent een bepaalde kleur toe aan een pixel. Niet alleen de pixel wordt gekleurd maar ook een subset van het beeld. Dit zijn de pixels die een omgeving hebben met een afstand tot de omgeving van de geselecteerde pixel binnen een zekere threshold.



Figuur 4.1: Zelf-similariteit gebaseerde textuur editing: (links)origineel, (tweede links)een editing toegepast, (rechts)meerdere editings toegepast.(bron: [BD02])

Voor de omgeving stelt men een cirkelvormig masker voor met een diameter d . De afstand tussen omgevingen wordt gemeten met de L_2 norm, zoals eerder gezien is dit gewoon de som van de kwadraten van verschillen tussen de pixels. Ter illustratie, voor een RGB beeld met een diameter van 9 kan dit verschil oplopen tot $256 \cdot 3 \cdot 69 = 13.5$ miljoen. De afstand drempelwaarde dient een maximum afstand aan te geven voor welke de transparantie 0 is. Voor een pixel met een afstand 0 zal de transparantie 100%(volledige toekenning) zijn en voor een pixel met maximum afstand 0%(pixel wordt niet gewijzigd). Ertussen wordt transparantie lineair bepaald door de afstand.

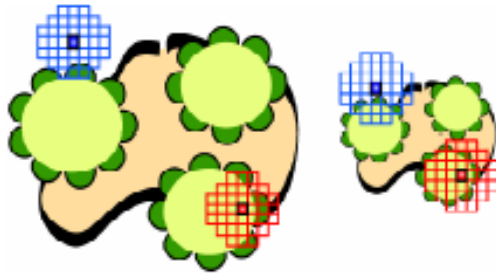


Figuur 4.2: Het wijzigen van globale sterkte (links) of afstand threshold (rechts).

In figuur 4.2 kan de gebruiker het algoritme op 2 manieren beïnvloeden. Door de globale sterkte te veranderen (met een vaste afstand) wordt de sterkte van de transparantie aangepast. Door de afstand aan te passen (een vaste sterkte) wordt het bereik van pixels die worden beïnvloed groter of kleiner.

Om het algoritme interactief te houden voor grotere texturen versnellen we het door multi-schaal omgevingen te onderzoeken. Er wordt een gausiaanse pyramide gebouwd van de basistextuur. Zo kunnen we een grotere textuur onderzoeken tegen een lage kost. Voor de pixels met een kleine kost kunnen we nog steeds in het originele beeld zoeken.

Gaussiaanse pyramide Is een verzameling herbemonsterde beelden met steeds een kleinere resolutie. De bemonstering gebeurt met een Gaussiaans masker. Gewoonlijk heeft elk opeenvolgend beeld de helft van de resolutie van het vorige beeld.



Figuur 4.3: Multi schaal omgevingen onderzoeken.(bron: [BD02])

4.2 Textuur klonen

Het klonen van een textuur werkt nagenoeg hetzelfde als het editeren van een textuur. De gebruiker specificeert een punt op de textuur met volledige transparantie. Maar ipv een geselecteerde kleur toe te passen zal de pixel met overeenkomstige coördinaten uit de kloontextuur gebruikt worden. De transparantie wordt weer bepaald door de afstand van de omgevingen van de geselecteerde pixel tot de omgeving van de andere pixels.

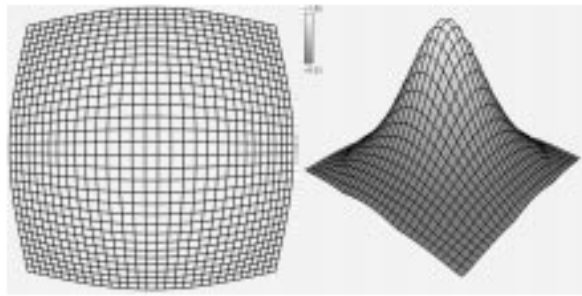


Figuur 4.4: Textuur klonen:(links)grondtextuur, (rechts)kloontextuur, (onder)klonen toegepast. (bron: [BD02])

4.3 Textuur warpen

Weer geldt hetzelfde principe van omgevingen vergelijken om afstand te bepalen met de geselecteerde pixel. Maar dit keer gaan we de textuur warpen i.p.v inkleuren. Waar we eerst een bepaalde transparantie toekenden aan een pixel kennen we nu een expansie of inkrumping toe. De techniek die we hiervoor gebruiken noemt **niet-lineaire magnificatie velden** uit [KR97].

Niet-lineaire magnificatie velden zijn abstracte representaties van niet-lineaire magnificatie die een methode voorzien om een omzetting teweeg te brengen tussen een magnificatieveld M en een transformatieveld T . Een magnificatie functie bepaalt een transformatie t die een punt uit een vierhoekig domein D verplaatst. Om de berekeningen te kunnen uitvoeren werken we op een discretisatie van D , nl. een $p \times q$ grid G . Concreet is een magnificatie veld M een scalair veld van de vorm $z = m(x, y)$ welke een regionale expansie geeft rond elk punt. De paper beschrijft gedetailleerd conversies tussen



Figuur 4.5: (links)Transformatieveld met bijhorend magnificatieveld(rechts) voorgesteld als een hoogteveld.(bron: [BD02])

M en T . Een conversie van T naar M komt er op neer dat we de oppervlakte berekenen van de getransformeerde knopen. Het is echter de conversie van M naar T waarin we geïnteresseerd zijn. We berekenen immers een magnificatie veld op dezelfde manier als we de transparantie van een pixel uit een textuur toekenden. Men gebruikt een iteratieve methode om het integratie probleem op te lossen:

1. Stel we hebben een magnificatiemesh M_s als input, initialiseer een transformatiegrid T_c uniform.
2. Bereken de huidige magnificatie M_c uit T_c door voor elke node de oppervlakte te berekenen.
3. Bereken de magnificatie error $M_e = M_s - M_c$. Als $M_e(i,j) > 0$ dan duwen we de burens van $T_c(i,j)$ weg en omgekeerd als $M_e(i,j) < 0$

dan trekken we de burens van $T_c(i,j)$ iets dichtert. In deze stap is het belangrijk de ordening van knopen te behouden.

4. Als de totale fout kleiner is als een vooraf bepaalde waarde stop, anders ga terug naar stap 2. De foutmetriek die toegepast wordt is $RMSE = \sqrt{\sum_{i=1}^p \sum_{j=1}^q \text{Max}(0, Me(i,j) - Ce)}$ of *root mean square error*.

In similariteit-gebaseerde warping worden de similariteitswaarden geïnterpreteerd als de magnificatie functie M . Als een pixel een transparantie heeft van 75% dan zal de lokale omgeving van die pixel ook een toename kennen van 75%. Voor het warpen zelf gebruikt men hardware om de mesh te bekleden met textuur. Een textuur van 256×256 zal dus een mesh van 256×256 nodes nodig hebben maar een mesh met een iets lagere resolutie (bv 128×128 zal ook nog goede resultaten reproduceren). Een opmerking: In regio's waar een sterke magnificatie heeft plaatsgevonden kunnen details verloren gaan, dit kan opgevangen worden door verloren gegane pixels op te vullen met een synthese algoritme als bv Efros en Lung.



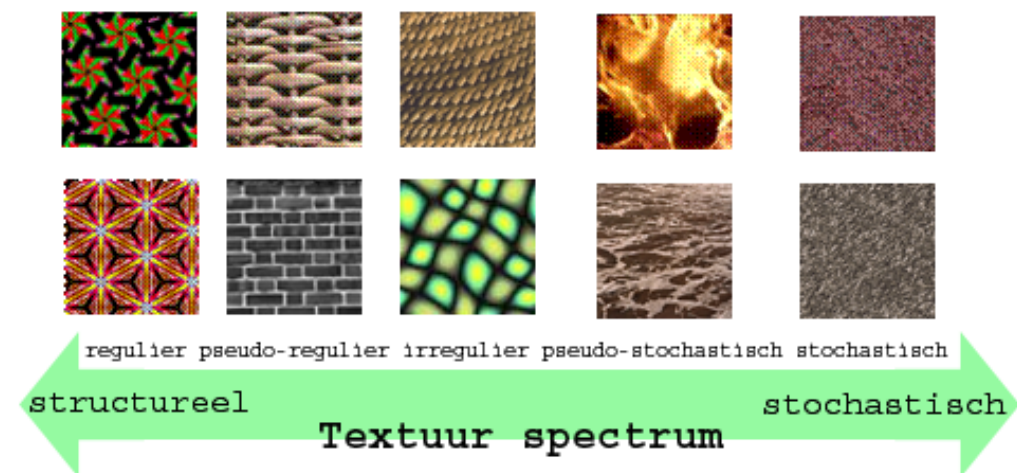
Figuur 4.6: Similariteit-gebaseerde warping: (links)Verkleinde bloemen, (midden)originele textuur, (rechts)vergrootte bloemen.(bron: [BD02])

Hoofdstuk 5

Textuur analyse

5.1 Regulariteit van een textuur

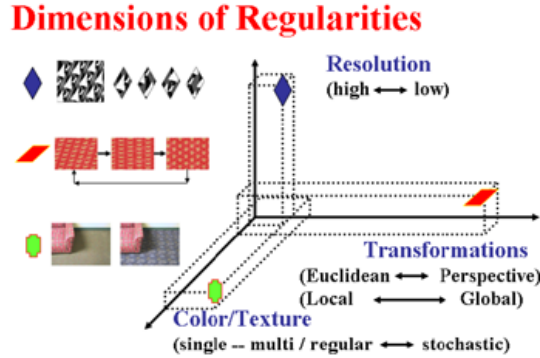
Pseudo-reguliere texturen zijn overal terug te vinden. Gebouwen, behangpapier, vloeren, stoffen bevatten vaak een patroon dat regulier is ofwel een sterke hang heeft naar symmetrie. We kunnen een textuur classificeren aan de hand van zijn afwijking ten op zichte van een regulier textuur patroon.



Figuur 5.1: De verschillende maten van regulariteit van een textuur. (bron: [LL03])

Een pseudo-reguliere textuur kan gezien worden als een statistische misvorming van een regulier getegeld patroon met verschillende variaties in uitzicht tussen de tegels.

Een tegel kan dus verschillen van zijn burens in vorm, grootte, kleur en belichting. Bij het classificeren van pseudo-reguliere texturen maken we een



onderscheid in enkele categoriën zoals in [LLH04].

5.2 Verband tussen irregulier en reguliere texturen

Zoals we daarnet gezien hebben heeft een irreguliere textuur een sterke structurele en kleurdeformatie ten opzichte van zijn reguliere oorsprong. De sleutel tot de analyse en manipulatie van irreguliere texturen is de deformatie tussen deze twee textuuruimten. In deze sectie onderzoeken we hoe een textuur van zijn irreguliere staat kan worden gevormd naar zijn reguliere oorsprong. We volgen hier de vooral [LL03] en [LLH04].

5.2.1 Transformatie van irreguliere naar reguliere structuur

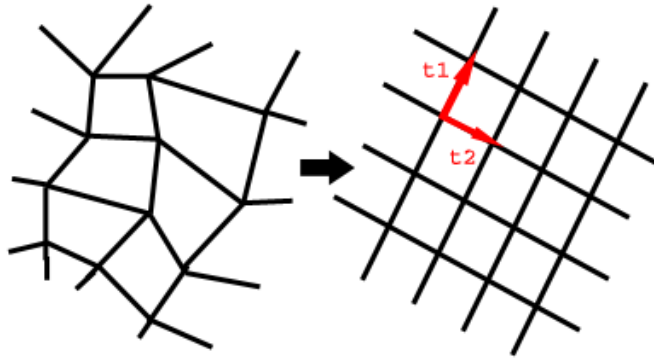
We gaan er uit dat elke regulier textuur een onderliggende 2D lattice heeft gegenereerd door twee lineair onafhankelijke vectoren t_1 en t_2 . We noemen deze lattice L_r . Een irreguliere textuur heeft een onderliggende lattice L_{ir} die een distortie is van deze L_r . Er zijn echter veel mogelijke lattices L_r waar L_{ir} naar kan warpen. We proberen echter een lattice te genereren samengesteld door twee vectoren t_1 en t_2 zo dat de deformatie tussen L_r en L_{ir} minimaal is. De totale deformatie wordt gemeten door de som van alle lengtes van verschillen tussen overeenkomstige punten van L_r en L_{ir} . Zo wordt het berekenen van L_r een **minimalisatieprobleem**. Hoe we dit oplossen zullen we later zien.

$$\min_{\|\vec{t}_1\|, \|\vec{t}_2\|, \theta} E = \sum_{i=1}^{N_i} (l_i - \|\vec{t}_1\|)^2 + \sum_{j=1}^{N_j} (l_j - \|\vec{t}_2\|)^2 + \sum_{k=1}^{N_k} (l_k - \|\vec{t}_1 + \vec{t}_2\|)^2 + \sum_{m=1}^{N_m} (l_m - \|\vec{t}_1 - \vec{t}_2\|)^2$$

5.2.2 Berekenen van het geometrisch deformatieveld

Nu we L_r hebben bepaald kunnen we een *Geometrisch deformatieveld* d_{geo} berekenen van L_r naar L_r . In [LL03] wordt voorgesteld het MFFD algoritme te gebruiken door corresponderende lattice punten van L_r en L_r te gebruiken als controlepunten. MFFD of *multilevel free-form deformations* wordt beschreven in [LCS95] is een interpolatietechniek die een bijectieve warping berekend tussen features van twee beelden. De techniek is gebaseerd op B-spline benadering en de warping is C continu.

De resultaten van uit de paper zien er zeer goed uit maar omdat tijd ook een bepalende factor is en omdat de warping van L_r naar L_r de enige echte belangrijke is zal ik een benadering zoeken door L_r te discretiseren en door parametrizatie de warping vectoren te berekenen. Hoe dit gedaan wordt zal gedetailleerder besproken worden in hoofdstuk 6.



Figuur 5.2: Irreguliere lattice(links) en reguliere lattice(rechts).

5.2.3 Dualiteit van het geometrisch deformatieveld

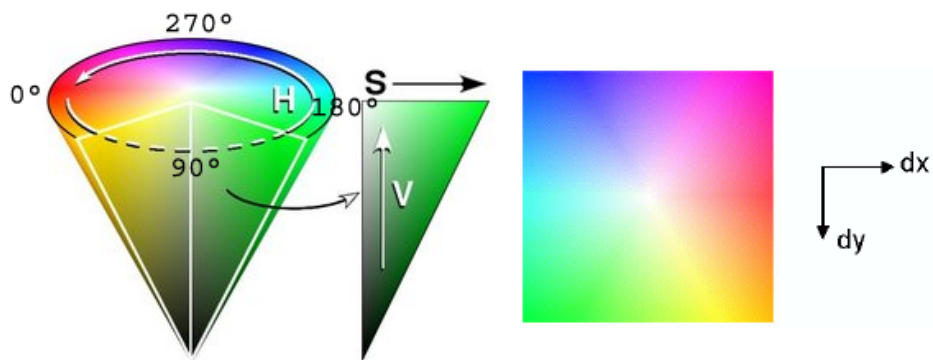
Het deformatieveld kan worden voorgesteld als een 2D vectorveld en als een textuur die kan gesynthetiseerd worden. We komen hier bij een kernbegrip van deze thesis, nl. dualiteit van het deformatieveld. Dit is zo belangrijk omdat het centraal zal staan voor een getrouwe synthese van het deformatieveld voor nieuwe irreguliere texturen.

Een deformatieveld kan op 2 manieren worden voorgesteld.

- Ten eerste is het deformatieveld een vector veld dat een beeld transformeert naar zijn vervormde versie.
- Ten tweede kan het deformatieveld zelf worden gezien als een textuur.

Om een vectorveld voor te stellen als een textuur kunnen we het coderen in de HSV kleurruimte. De HSV kleurruimte is zeer geschikt omdat we de

vector kunnen voorstellen als een hoek en een lengte. De informatie over de hoek zetten we in Hue (kleur) met een waarde tussen 0 en 360. De lengte van de vector kunnen we toekennen aan Saturatie (verzadiging) met een waarde tussen 0 en 1. De waarde van het V(brightness) kanaal zetten we op 1. Het deformatieveld is nu klaar voor synthese. Dit geeft ons interessante mogelijkheden voor manipulatie van het deformatieveld. In het volgende hoofdstuk zien we hoe het deformatieveld zal gesynthetiseerd worden.



Figuur 5.3: Voorstelling van het geometrisch deformatieveld in HSV kleur-ruimte. (bron:http://en.wikipedia.org/wiki/HSV_color_space)

5.3 Kleur deformatie veld

Naast de geometrische deformatie kunnen we ook het kleurdeformatieveld modelleren. Dit heeft als doel de kleurinformatie van tegels te manipuleren net zoals we bij geometrische deformatie doen. Hoe doen we dit? Allereerst hebben we een pseudo-reguliere textuur nodig zoals bekomen door rectificatie. Hieruit halen we enkele rechthoekige tegels die representatief zijn voor de mogelijke kleurvariëaties van de tiles over de gehele textuur. Een tile met een hoogte en breedte van 100 pixels en RGB waarden per pixel heeft 30000 dimensies. In realiteit zijn er slechts zeer weinig onafhankelijke dimensies waarin een tegel's kleuren variëren en dus zou het zinloos zijn het kleurdeformatieveld zo voor te stellen. We gebruiken PCA om een schatting te maken van de gereduceerde basis.

PCA of *Principal Component Analysis* is een klassieke statische methode om de variantie van een dataset te berekenen. PCA berekent een orthogonale basis voor de ingevoerde data, deze stelt de dimensies voor waarin de data kan variëren. De output van het algoritme is deze basis en een set lineaire combinaties, de 'PCA componenten' waaruit de dataset d.m.v de basis is opgebouwd.

Om het kleur deformatieveld te berekenen nemen we een aantal tegels uit de textuur en stoppen we deze in een kolom van een twee-dimensionale dataset. Om PCA uit te voeren trekken we de gemiddelde tegel van elke tegel af en vermenigvuldigen we de dataset met zijn getransponeerde. Voor ons type dataset zou dit een te grote matrix opleveren om praktisch uitvoerbaar te zijn. Daarom laten we deze methode vallen en onderzoeken we een andere methode om PCA te berekenen, nl. de methode volgens Matusik uit [MPBM03].

PCA volgens Matusik is een zeer efficiënte variant van het algoritme. Het basisprincipe van het algoritme blijft hetzelfde maar we berekenen de covariantie matrix andersom. Hier volgt een overzicht van de methode. Eerst trekken we het gemiddelde af van de dataset D en dit levert D' op. We berekenen de covariante matrix C en geven deze als input aan het SVD of *singuliere waarde decompositie* algoritme. Deze berekent een diagonaal matrix S die de eigenwaarden bevat en een matrix V met de eigenvectoren.

$$C = D'(t)xD'$$

$$C = V.S.V(t)$$

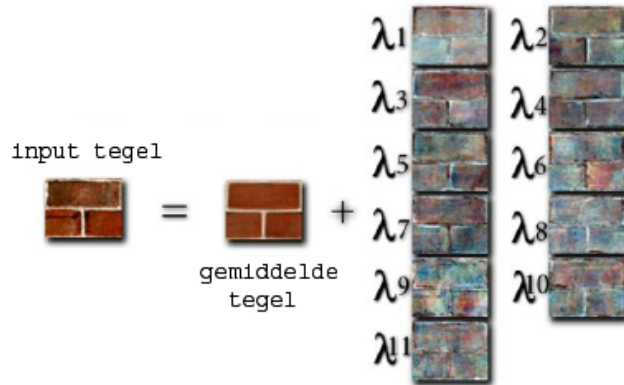
$$U = D'.V.S$$

Nu is $U(t)$ de nieuwe basis. We berekenen de PCA componenten X . De PCA componenten zijn de lineaire combinaties van de basis waaruit de originele dataset bestaat.

$$X = U(t).D'$$

5.4 Belichting deformatie veld

Als laatste deformatieveld om de textuur analyse te voltooien willen we ook in staat zijn de textuur op te splitsen in reflectantie en belichting. In de paper [LLH04] waar veel van de literatuurstudie naar gelinkt is gebruikt men de methode beschreven in [TLR01]. Deze bespreekt uitvoerig een methode om complexe belichting uit een beeld te berekenen. Maar door enkele vereenvoudigende aannames vonden we een andere intuïtievere oplossing, welke ook gebruikt wordt in [OCDD01] voldoende. Een eerste aanname die we maken is dat de textuur waar we een belichtingsmap uit berekenen pseudo-regulier is(1). Omdat we enkel op de pseudo-reguliere versies van texturen gaan werken is dit geen probleem. Een tweede aanname die we maken is dat de belichting in de textuur wordt veroorzaakt door laag-frequente veranderingen in het beeld en dat de reflectantie van de textuur wordt veroorzaakt door hoogfrequente veranderingen in het beeld(2). Deze



Figuur 5.4: Een dataset(tegel) is een lineaire combinatie van een beperkt aantal orthogonale basissen.(bron: [LLH04])

tweede aanname hebben we nodig omdat we het beeld gaan wazig gaan maken met een Gaussiaanse filter met een grootte gelijk aan de feature grootte. Hier volgt een beschrijving.

Stel I een textuur die wordt opgesplitst in een shadingmap S en een reflectantiemap R .

$$I = S \times R$$

$$\log(I) = \log(S) + \log(R)$$

We maken een schatting van $\log(S)$ door $\log(I)$ te blurren en het gemiddelde ervan af te trekken. Dit is gebaseerd op onze aanname dat de shading van de textuur overeenkomt met de laagfrequente veranderingen in het beeld(2). De gemiddelde kleur van dit beeld wordt hiervan afgetrokken omdat we stellen dat de gemiddelde kleur afkomstig is van de reflectantie van het beeld. Dit klopt bij deze textuur omwille van (1). Noem de bekomen shading van het beeld S' . We kunnen nu eveneens de reflectantie van het beeld berekenen.

$$\log(R) = \log(I) - \log(S) + \text{gemiddelde}(\text{blur}(\log(I)))$$

$$R = \exp(\log(R))$$

Een mogelijke verfijning van deze methode is het gebruik van bilaterale filters. Bij het zoeken van de laagfrequente veranderingen in het beeld, 'het blurren', gaan details van de shading verloren. Vooral shaduwranden hebben hieronder te lijden. Een bilaterale filter is in staat net dit probleem op te lossen, de methode wordt voorgesteld in [OCDD01].

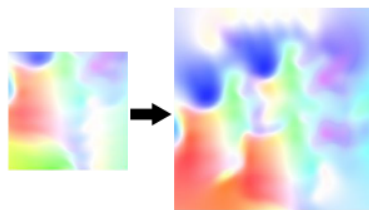
bilaterale filters Een bilaterale filter is een twee-stage filter met als doel het beeld wazig te maken en tegelijk de scherpheid van randen te behouden. Dit wordt gerealiseerd door enkel de pixels wazig te maken met een kleine afstand tot de center pixel. Met afstand bedoelen we hier het SSD of *squared sum difference*. In de eerste pass wordt een gewichtsmasker berekend aan de hand van de afstand. Pixels met een grote afstand krijgen een kleiner gewicht. Na een normalisatie wordt dit masker pixelgewijs vermenigvuldigd met een Gaussiaansmasker.

Hoofdstuk 6

Pseudo-reguliere textuur manipulatie

6.1 Synthese van het geometrisch deformatieveld

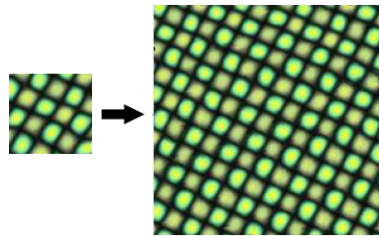
Deformatievelden zijn gewoonlijk niet-reguliere, stochastische texturen. Elk bestaand textuur synthese algoritme zou dus in principe gebruikt kunnen worden. Het deformatieveld kan echter op twee manieren gesynthetiseerd worden. Men stelt voor een patch-gebaseerd synthese algoritme te gebruiken zoals in [LTL05] en [EF01] op het gewone deformatieveld. Patch gebaseerde methodes werden besproken in hoofdstuk 3. Een andere methode is een per pixel synthese van de controlepunten van het deformatieveld. De verplaatsingsvectoren van het deformatieveld worden dan zoals voordien gecodeerd als een kleur. Voor deze synthese kunnen we eender welke pixel-gebaseerde methode gebruiken zoals bv. in [EL99]. Ook in [LLH04] wordt deze methode voor dit doel gebruikt.



Figuur 6.1: Een gesynthetiseerd deformatieveld.(bron: [LLH04])

6.2 Synthese van pseudo-reguliere texturen

Meer precies bedoelen we hier de synthese van de reguliere oorsprong van een irreguliere textuur. We willen een nieuwe textuur genereren met een exacte reproductie van de structuur en een voldoende variatie aan kleur van de tegels zoals in de originele reguliere textuur. We kunnen geen pixel gebaseerde synthese technieken gebruiken. Deze doen het goed voor stochastische en irreguliere texturen maar schieten tekort bij het exact reproduceren van de structuur die eigen is aan reguliere texturen. Bij patch-gebaseerde methodes zal dit probleem veel minder erg zijn maar dan moet de grootte van de patch overeenkomen met de grootte van de feature. In zekere zin is de methode voorgesteld in [LTL05] een patch-gebaseerde methode. Deze gaat echter veel verder in het behoud van structuur.



Figuur 6.2: Synthese van een pseudo-reguliere textuur.(bron: [LLH04])

De methode tegelt een nieuwe textuur aan de hand van de onderliggende reguliere lattice gevormd door t_1 en t_2 . Een tegel kan verschillende vormen hebben gaande van vierhoekig, vierkant, ruitvormig tot parallellografisch.

Om het algoritme te beschrijven worden enkele nieuwe begrippen worden geïntroduceerd. We creëren een verzameling **minimum tegels** t_i zijnde de tiles bepaald door de onderliggende lattice. Dit zijn de kleinste onderdelen die de structuur van de textuur vatten en vormen dus de basis voor de synthese. We definieëren ook een verzameling **maximum tegels** T_i zijnde de kleinste rechthoekige omhulsels van tegels uit t_i . De minimum verzamelingen en dus ook bijgevolg de maximum verzamelingen worden ook uitgebreid met een groep tegels verschoven over de lattice met een offset $((n + 1/2)t_1, (m + 1/2)t_2)$ waar m en n de nummering van de controle punten zijn van de lattice. Hier volgt een overzicht van de werking van het algoritme, voor meer details zie [LTL05].

1. analyse

- Bepaal de vectoren t_1 en t_2 van de onderliggende lattice.
- Bepaal de minimum verzameling t_i .

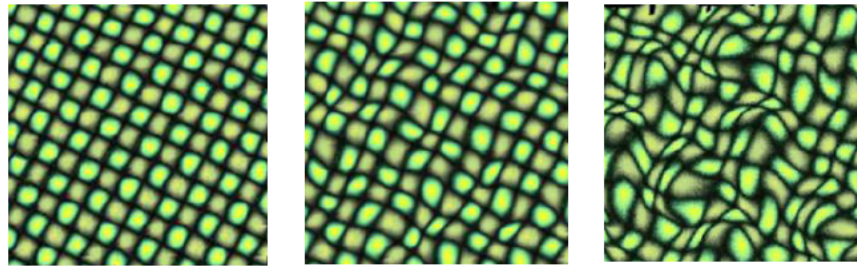
- Bepaal de maximum verzameling T en Th , T is de verzameling omhullenden van de oorspronkelijke $ti's$ en Th is de verzameling omhullenden van de verschoven $ti's$.

2. synthese

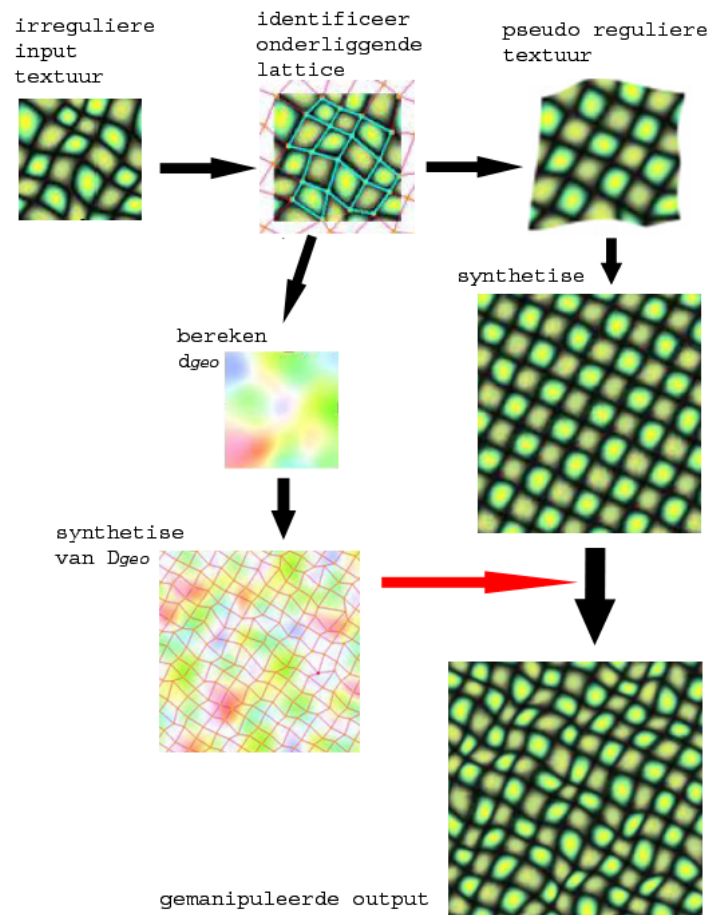
- Start vanaf de linkerbovenhoek met een willekeurige tegel uit T .
- Voeg één per één een tile toe aan de gesynthetiseerde langs de vector $t1 + t2$ met een stapgrootte van $(t1 + t2)/2$.

6.3 Manipulatie

Gebruik makend van de dualiteit van het deformatieveld kunnen we nu de textuur manipuleren. We stellen het deformatieveld voor als een textuur en synthetiseren het. Het deformatieveld passen we toe op de gesynthetiseerde pseudo-reguliere textuur. Het deformatieveld kan worden gemanipuleerd d.m.v een deformatie factor, wanneer deze factor 1 is bootsen we een natuurgetrouwe deformatie na als in de originele irreguliere textuur. Met een deformatie van 0 behouden we een reguliere structuur en bij een deformatie groter als 1 versterken we de deformatie.



Figuur 6.3: deformatie factor 0(links), deformatie factor 1(midden), deformatiefactor 2(rechts). (bron: [LLH04])



Figuur 6.4: Overzicht pseudo-reguliere textuur manipulatie: allereerst wordt de onderliggende lattice L_{ir} geïdentificeerd en getransformeerd naar L_r . De hieruit voortkomende pseudo-reguliere textuur en geometrische deformatieveld worden beide gesynthetiseerd. Tenslotte wordt de distortie op de pseudo-reguliere textuur berekend door D_{geo} .

Hoofdstuk 7

Textuur vervanging

7.1 Textuur vervanging m.b.v. deformatievelden

Eén van de mogelijke applicaties van pseudo-reguliere textuur analyse beschreven in [LLH04] is textuur vervanging. Wanneer we beide het geometrische en belichtingsdeformatieveld hebben gedefinieerd voor een textuur kunnen we deze toepassen op een andere textuur of die nu regulier is of niet. De



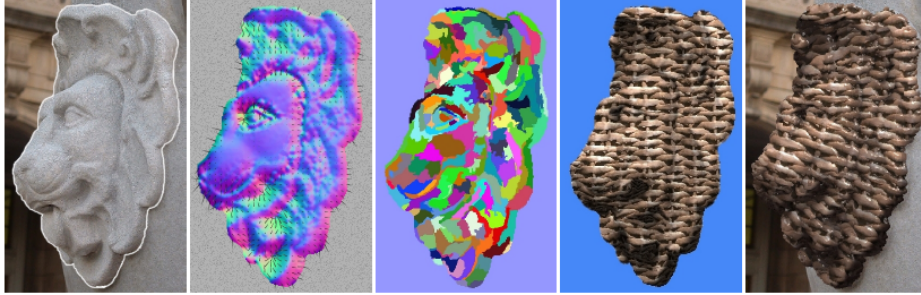
Figuur 7.1: Enkele texturen met hetzelfde geometrisch-en belichtingsdeformatieveld toegepast.(bron: [LLH04])

methode werkt als volgt. Eerst berekenen we de reguliere oorsprong van het geometrisch deformatieveld van de textuur. Vervolgens passen we methode uit 5.4 toe om het belichtingsdeformatieveld te berekenen uit de gectificeerde textuur. We kunnen nu elke textuur nemen die we willen om er het belichtings deformatie veld op toe te passen door een pixel gewijze vermenigvuldiging. Als laatste stap passen we het inverse geometrische deformatieveld toe op de textuur.

7.2 Textuur vervanging door dieptekaarten

Een alternatief voor textuur vervanging wordt beschreven in [FH04]. De techniek combineert Shape from shading en texture synthese om een textuur toe te passen op objecten in een beeld. Een object in het beeld wordt aangeduid door de gebruiker. Shape from shading wordt toegepast om normalen

te schatten van het object. Deze normalen worden geclustered in patches en deze worden gevuld met textuur synthese. Tenslotte wordt een distorsie toepast op elke patch aan de hand van de normalen van de patch. We lichten elke stap van het algoritme gedetailleerder toe maar het gros van de technische details is weggelaten.



Figuur 7.2: Textuur vervanging door dieptekaarten.(bron: [FH04])

7.2.1 Bepalen van normalen.

Shape from shading algoritmen hebben als bedoeling een drie-dimensionale oppervlakte te construeren uit een beeld door als output een gradient en een hoogtemap te geven. Men volgt hier echter een andere aanpak die volstaat voor hun doeleinden. Men gebruikt het Lambertiaans reflectie model om de belichting te schatten. De lichtste pixel wordt direct belicht door een imaginaire lichtbron ver genoeg van het beeld. De donkerste pixel is beschaduwde en zijn waarde stelt het ambient licht voor in het beeld. De cosinus van de hoek tot de lichtbron wordt geschat door $c(x, y) = (I(x, y) - I_{min}) / (I_{max} - I_{min})$ en de sinus van de hoek door $s(x, y) = \sqrt{1 - c(x, y)^2}$. De normaal kan dan als volgt berekend worden.

$$G(x, y) = \nabla I(x, y) - (\nabla I(x, y) \cdot S)S$$

$$N(x, y) = c(x, y)S + s(x, y)G(x, y) / ||G(x, y)||$$

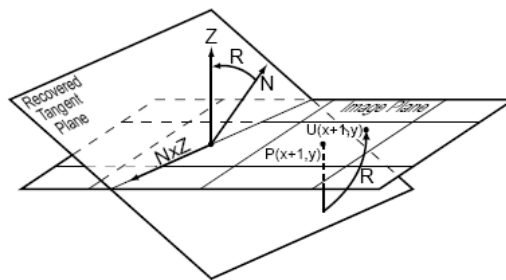
met $\nabla I(x, y) = (\delta I / \delta x, \delta I / \delta y, 0)$ de gradiënt van het beeld.

7.2.2 Patchen van het beeld

Wanneer de normalen berekend zijn verdelen we het beeld op in patches met gelijkaardige normalen. Het algoritme gebruikt een bottom-up schema voor het bepalen van de patches. We initialiseren iedere pixel door zijn eigen patch toe te kennen waarna we patches gaan samenvoegen en een foutmetriek wordt gebruikt om een grens te bepalen voor de patches.

7.2.3 Patch oriëntatie

Om de illusie van een vervormd oppervlak te creëren moeten we aan de pixels uit de patches nieuwe textuurcoördinaten toekennen aan de hand van de normalen. De pixel in het centrum van de patch $P(0,0)$ kennen we de textuurcoördinaten $U(0,0) = (0,0)$ toe. Voor het punt $P(x+1,y)$ kennen we de textuurcoördinaten $U(x+1,y)$ toe dewelke een projectie van $P(x+1,y)$ is op het vlak bepaald door de normaal van $P(x,y)$ door dat punt. Zie figuur 7.3 ter verduidelijking.



Figuur 7.3: Berekening van de textuurcoördinaat U van een pixel P .(bron: [FH04])

We kunnen nu de patches synthetiseren en uitlijnen met elkaar. Zichtbare randen tussen patches worden gecamoufleerd door patches te laten overlappen door het **graphcut** algoritme [KSE⁺03]. Door nadien de patch-distortie toe te passen kunnen we een natuurgetrouwe textuur vervanging realiseren.

Hoofdstuk 8

Deformatieveld analogieën

Een laatste toepassing van pseudo-reguliere texturen zijn deformatieveld analogieën. Dit werd eveneens onderzocht in [LLH04]. We geven eerst een introductie over beeldanalogieën alvorens het toe te passen op deformatievelden.

8.1 Beeld analogieën

Deze methode wordt voorgesteld in [HJO⁺01]. Het doel is een **analogie** te zoeken tussen twee verschillende beelden en dan deze informatie te gebruiken om een ander beeld te transformeren. Meer concreet, gegeven een paar beelden A en A' met A het ongefilterde en A' het gefilterde beeld en een extra ongefilterd beeld B als input. Synthetiseer een nieuw image B' zodat B en B' analoog zijn aan elkaar zoals A en A' analoog aan elkaar zijn.

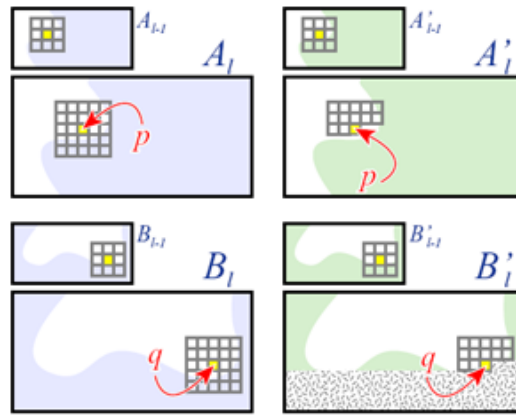
Deze techniek kan gebruikt worden voor nabootsing van een verscheidenheid aan methoden waaronder traditionele filters zoals blurring, textuur synthese, superresolutie(waarin een laag resolutie beeld wordt omgezet in een beeld met hogere resolutie), textuur transfer, artistieke filters enz...



Figuur 8.1: Een beeld analogie voorbeeld.(bron: [HJO⁺01])

Een belangrijk begrip zijn **feature vectoren**. Men specificeert feature vectoren voor elke pixel. Deze vectoren zullen het matchen van pixels

uit verschillende beelden vergemakkelijken. Een feature vector is een reeks waarden geassocieerd aan de pixel van een beeld. De meest voor de hand liggende featurevector voor een pixel is natuurlijk zijn RGB waarde, dit is echter niet altijd voldoende en soms vervangt of vult men dit aan met de luminantie van de pixel(luminantie kan bekomen worden uit het Y kanaal van YIQ kleur ruimte), wanneer we de kleur terug willen kopiëren we het I en Q kanaal uit B naar B', en converteren we de kleurruimte terug naar RGB. We zullen geen volledige uitwerking van het algoritme presenteren omdat we het framework zullen gebruiken dat vrij kan gedownload worden op de website van de paper.



Figuur 8.2: Illustratie van het matching principe(bron: [HJO⁺01]). Om de waarde van de pixel q in het gefilterde beeld B' te synthetiseren, evalueren we de omgeving van de pixels uit B, B', B_{l-1}, B'_{l-1} . We zoeken naar de pixel p in de A beelden voor de grootste gelijkheid. The synthese gebeurt in scan-line orde in B' .

8.2 Deformatieveld analogieën

Men beweert dat er een sterke associatie is tussen het geometrische deformatieveld van een textuur en het belichtingsdeformatie veld. Het algoritme voor beeld analogieën uit 8.1 is een uitstekende manier om dit idee uit te buiten. Gegeven een pseudo-reguliere textuur, bereken het geometrisch en belichtingsdeformatieveld. Deze kunnen worden behandeld als een paar A en A'. We synthetiseren nu A naar een kunstmatig geometrisch deformatieveld B. We kunnen nu de belichtingsmap B' vinden door het algoritme toe te passen.

Deel III

Implementatie

Hoofdstuk 9

Zelf Similariteit

Dit is de eerste implementatie die ik heb gemaakt en is volledig gebaseerd op [BD02]. Deze paper bespreekt drie toepassingen over zelf-similariteit, nl. textuur bewerken, textuur klonen en textuur warpen. Ik bespreek de implementatie van deze 3 delen afzonderlijk.

9.1 Textuur bewerken

Het doel is een tool te maken die op een interactieve manier toelaat een pixel in te kleuren en deze actie reproduceert over de hele image. Om de selectie van de gebruiker enige betekenis te laten hebben moeten we de omgeving bekijken van de pixel. We berekenen de omgeving van de geselecteerde pixel en vergelijken die met de omgeving van elke andere pixel in het beeld. In de paper werd een cirkelvormig masker voorgesteld, men zou ook een vierkant masker kunnen gebruiken maar zeker voor grotere maskers geeft een cirkelvormig masker een nauwkeurigere representatie van de omgeving. Ik zal dus eveneens een cirkelvormig masker gebruiken.

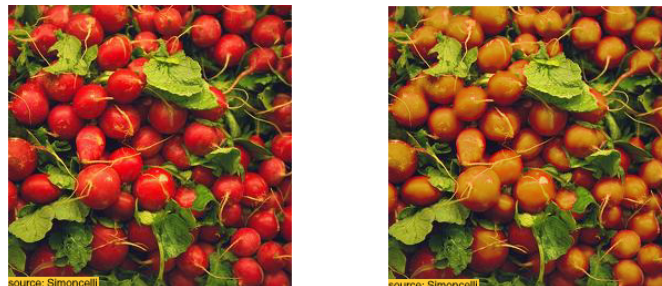


Figuur 9.1: Een eerste resultaat van textuur editing. De bladeren van de stengel van de bloemen hebben een andere kleur gekregen.

afstandsmetrick Als afstandsmetrick gebruiken we de L_2 -norm SSD of *Squared Sum Difference* van de RGB waarden van de pixels. (bijvoorbeeld: een RGB triplet (34,124,56) en een ander (198,120,100) hebben een SSD van $(34 - 198)^2 + (124 - 120)^2 + (56 - 100)^2$).

transparantie Pixels met een omgeving identiek aan de geselecteerde pixel zijn 100% transparent en worden dus volledig vervangen. Voor omgevingen met een afstand binnen een zekere drempelwaarde wordt ook een gedeeltelijke transparantie toegekend. Tussen een afstand van 0 en de maximale afstand wordt de transparantie geschaald tussen de maximale transparantie en 0 transparantie. De transparantie wordt ook nog beïnvloed door de globale sterkte. Bijvoorbeeld: bij een globale sterkte van 75% is de maximum transparantie 75%.

Hier zijn nog enkele resultaten uit mijn implementatie. De methode komt het best tot zijn recht bij complexe natuurlijke texturen. Bij elke figuur is aangegeven wat de radius en maximale afstand van de omgeving was en ook de globale sterkte van de transparantie.



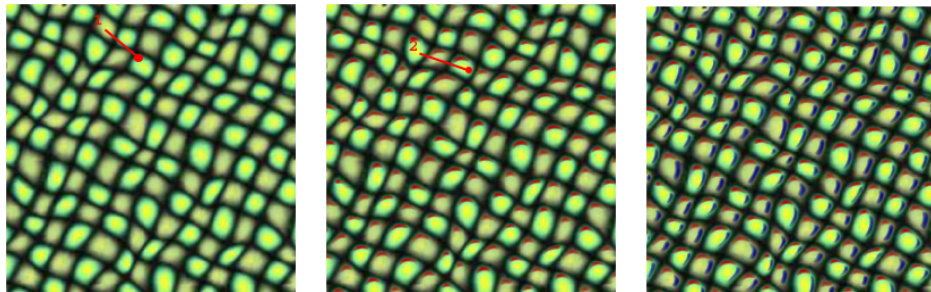
Figuur 9.2: (links) originele texture (bron: <http://www.cns.nyu.edu/eero/texture>) (rechts) afstand 500k, radius 4, transparantie 50%.

9.2 Textuur klonen

De implementatie van deze methode was niet zo verschillend als die van textuur bewrken. We hebben enkel nog een extra input nodig, nl. een kloon textuur. We selecteren net zoals tevoren een punt in de textuur maar in plaats van een pixel met transparantie een geselecteerde kleur toe te kennen, nemen we een pixel met overeenkomstige coördinaten uit de kloon textuur. De methode geeft op een eenvoudige manier zeer overtuigende resultaten.



Figuur 9.3: (links)originele textuur, (rechts)afstand 500k, radius 3, transparantie 100%.



Figuur 9.4: (Op de eerste en tweede figuur is aangegeven waar we een pixel inkleuren.)(links)originele textuur, (midden en rechts)afstand 500k, radius 3

9.3 Textuur warpen

Als laatste onderdeel van deze applicatie heb ik textuur warpen geïmplementeerd. Voor de textuur warping heb ik het algoritme van niet-lineaire magnificatie gebruikt zoals beschreven in de literatuurstudie. Voor de visuele representatie van de vervormde textuur heb ik een getextureerde mesh getekend m.b.v OpenGL.

Mijn implementatie werkt goed tot op een zekere hoogte, bij sterke deformaties convergeert het algoritme niet meer stabiel en wijkt het resultaat steeds meer af van de mesh waar naar we willen convergeren. Een stabielere implementatie van het magnificatie algoritme zou het resultaat moeten leveren zoals getoond in de paper.

De gebruiker selecteert een pixel die een expansie (of inkrimping) moet ondergaan en deze actie wordt gereproduceerd over het hele beeld analoog aan de vorige methodes. We doen dit door een magnificatie mesh te berekenen die als input dient voor het magnificatie algoritme. Dit berekent iteratief een transformatie mesh door omgevingen van knopen in te krim-



Figuur 9.5: Texture editing



Figuur 9.6: Zelf-similariteit warping: (links)origineel, (midden)resultaat na enkele iteraties, er is gepoogd de features te vergroten (rechts)warping mesh

pen of uit elkaar te duwen (Zie hoofdstuk 4.3 voor een uitgebreide werking van het algoritme). De correctheid van de transformatie mesh bepaald de kwaliteit van het beeld. In [KR97] gebruikt men de eenvoudige omgevingen van simpelweg de linker, rechter, onder en bovenbuur van de node. Dit hebben we geprobeerd maar bleek weinig stabiel te zijn. Het algoritme is ook getest met een cirkelvormig masker met een grotere straal. Dit bleek stabiel te zijn maar ook verschillende malen trager en na enkele iteraties meer divergeerde het resultaat toch. Knopen moeten ook altijd dezelfde volgorde behouden, daarom moeten extra beperkingen gelegd worden op de verplaatsing van de knopen. Wanneer een knoop met een sterke magnificatie in een positie geraakt waar hij een zeer kleine oppervlakte heeft zal hij slechts zeer traag (of helemaal niet) convergeren naar het correcte resultaat. Als aan deze moeilijkheden extra aandacht wordt besteed kan het algoritme stabiel worden maar dit valt echter buiten het bereik van deze thesis.

Hoofdstuk 10

Irreguliere textuur manipulatie

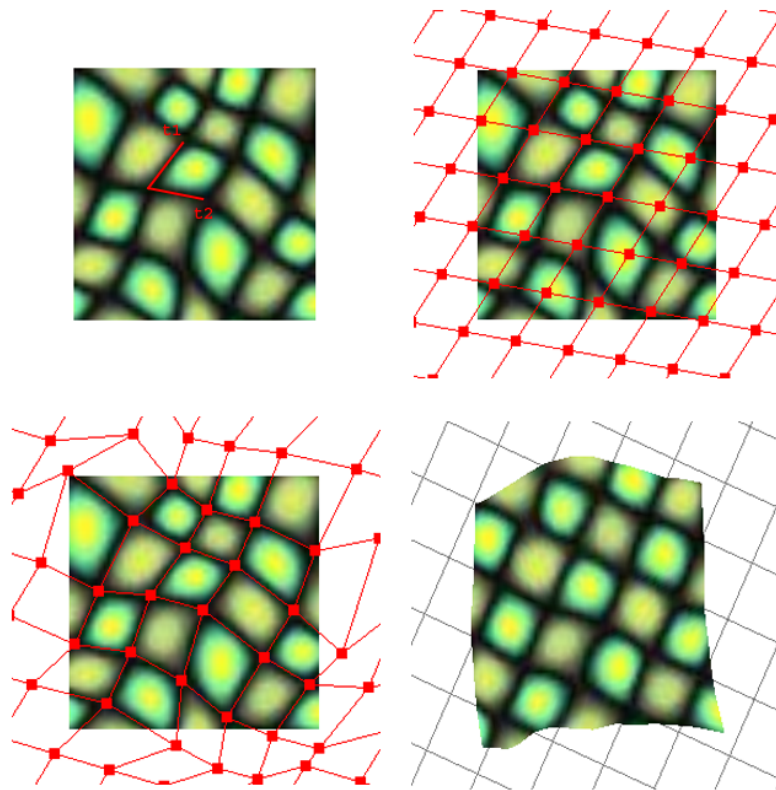
10.1 Bepaling van het geometrische deformatieveld

Om het geometrische deformatieveld te kunnen berekenen moeten we eerst de textuur relateren met zijn pseudo-reguliere oorsprong. In de paper [LL03] wordt de lattice numeriek berekend door het probleem te stellen als een **minimalisatieprobleem**(zie 5.2.1). Wij gaan echter de reguliere lattice bepalen door een mass-spring deformatie systeem. Eerst specificeren we handmatig een lattice met punten op de features van de textuur zodat de tegels van de lattice overeenkomen met de tegels van de textuur. Voor het gebruiksgemak worden eerst twee vectoren geplaatst die een lattice schatten over de hele textuur. Sommige controle punten zullen niet exact overeenkomen met de features van de textuur en dus moet de gebruiker die handmatig kunnen verplaatsen. Wanneer de gebruiker klaar is wordt de lattice vervormd naar zijn reguliere oorsprong d.m.v. een mass-spring systeem. Zie fig. 10.1.

Vervolgens berekenen we het deformatieveld. Zoals eerder vermeld kan het geometrisch deformatieveld worden voorgesteld als een 2D vectorveld waarbij elke vector de verplaatsing is tussen overeenkomstige controlepunten uit de irreguliere en reguliere lattice. Deze representatie zullen we later ook gebruiken voor de synthese. Een andere representatie van de deformatie is een per-pixel deformatie van de mesh. Beide zijn geïmplementeerd(zie fig 10.3 voor een voorbeeld van een per-pixel deformatieveld).

We stellen de textuur voor als een rechthoekige grid met een resolutie evenredig aan de grootte van de textuur en parametrizeren elk punt van de grid aan de hand van de controle lattice. Wanneer de lattice getransformeerd wordt naar zijn reguliere oorsprong herberekenen we de posities van de mesh aan de hand van deze parametrizatie. De verplaatsingen van deze punten zijn dan ons vectorveld. We moeten wel opmerken dat in tegenstelling tot

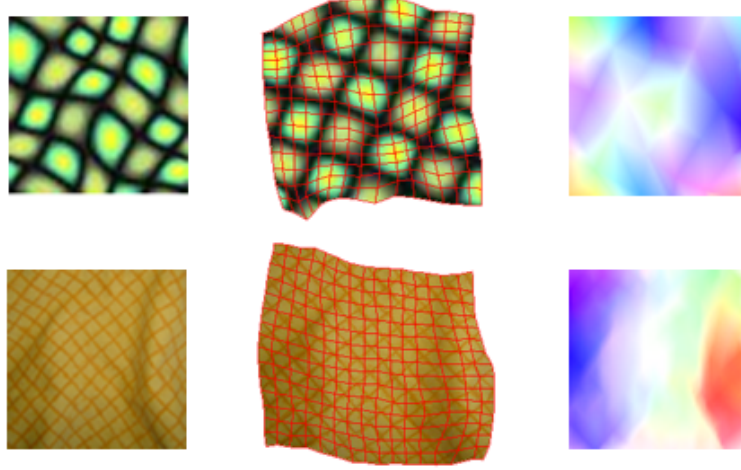
de methode uit [LCS95] waarbij de resulterende functie bijtief is, ons vectorveld injectief is. We berekenen de transformatie van irregulier naar regulier of vice versa. Voor onze doeleinden zal het volstaan de transformatie van regulier naar irregulier te berekenen.



Figuur 10.1: Relateren van een textuur met zijn pseudo-reguliere oorsprong: Eerst specificeert men twee vectoren die de basis leggen voor een lattice(linksboven). Dit creëert een lattice waarvan de punten niet exact overeenkomen met de features van een textuur(rechtsboven). We corrigeren de misplaatste controle punten(linksonder). Het resultaat van de deformatie is pseudo-regulier(rechtsonder).

We kunnen het bekomen deformatieveld visueel voorstellen als een textuur door de deformatie vectoren te coderen als een kleur in HSV ruimte. Hue stelt de hoek van de vector voor (0-360), Hue is de kleur van de pixel bv. een verplaatsing in de x-richting geeft een rode kleur. Saturatie zal de lengte van de vector voor stellen(0-1), Saturatie is de verzadiging van de kleur bv. een hoge waarde zal een diepe kleur geven. Value stelt de intensiteit van de pixel voor en wordt niet gebruikt voor onze vector, in plaats daarvan zetten we de waarde altijd op 1. De waarde van de saturatie wordt

genormaliseerd om een zo breed mogelijk spectrum aan kleuren te bevatten zelfs als de lengtes van de vectoren dicht bij elkaar liggen. Hieronder zijn enkele visualisaties van per-pixel deformatievelden.



Figuur 10.2: Enkele per-pixel deformatievelden

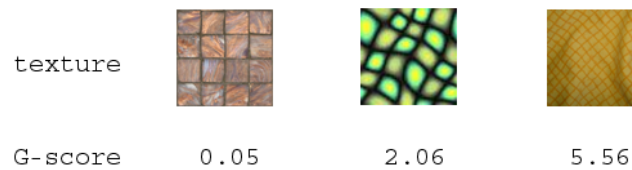
Maat voor geometrische regulariteit Bij een analyse is het interessant de regulariteit van een irreguliere textuur te berekenen ten opzichte van andere irreguliere texturen. Dit doen we door een textuur te relateren met zijn reguliere oorsprong en dan volgende formule toe te passen. We hebben

$$G = \sum_{i=1}^{N_i} \frac{(l_i - \|\vec{l}_1\|)^2}{\|\vec{l}_1\|^2} + \sum_{j=1}^{N_j} \frac{(l_j - \|\vec{l}_2\|)^2}{\|\vec{l}_2\|^2} + \sum_{k=1}^{N_k} \frac{(l_k - \|\vec{l}_1 + \vec{l}_2\|)^2}{\|\vec{l}_1 + \vec{l}_2\|^2} + \sum_{m=1}^{N_m} \frac{(l_m - \|\vec{l}_1 - \vec{l}_2\|)^2}{\|\vec{l}_1 - \vec{l}_2\|^2}$$

dit toegepast op de deformatievelden voor enkele texturen en de resultaten zijn voorspelbaar. Texturen met hoge regulariteit hebben een lage G-score en texturen met een irreguliere structuur hebben een hoge G-score. Een G-score van 0 zou een perfecte geometrische regulariteit aantonen.

10.2 Synthese van het geometrische deformatieveld

Synthese van het geometrische deformatieveld kan op twee manieren. De eerste methode is een per-pixel deformatie te modelleren van de textuurmesh en deze te laten synthetiseren door een algoritme als Efros en Lung. Dit



Figuur 10.3: Enkele voorbeelden van G-scores van texturen.

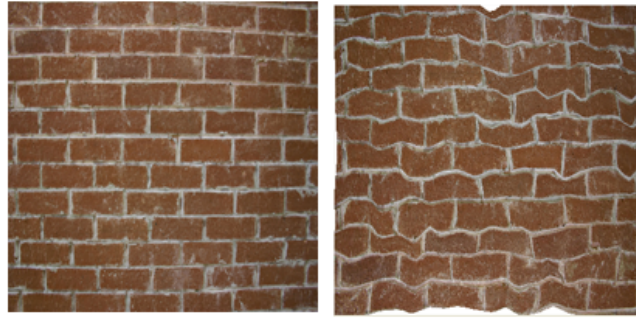
kan een volgend resultaat geven. Dit resultaat is een poging om het deformatieveld te reproduceren met het algoritme van Efros en Lung. Zoals je kan zien zijn de verschillende deformaties opgenomen in de gesynthetiseerde textuur zij het wat vervormd. Het is zeer moeilijk een synthese algoritme te produceren dat alle eigenschappen van de originele textuur overneemt. Een probleem bij de synthese is de keuze van de grootte van het masker. Nemen we het masker te klein, dan lopen we het risico dat de deformaties van pixels t.o.v hun burens weinig coherentie vertonen. Nemen we het masker te groot dan loopt het algoritme een kans om te blijven voortborduren op één groep coherente pixels waardoor de rest van het gesynthetiseerde beeld een vlak uitgestreken kleur krijgt.

De kwaliteit van textuur manipulatie is sterk afhankelijk van de kwaliteit van de deformatieveld synthese en kan nog sterk verbeterd worden.

De tweede methode om het geometrische deformatieveld te synthetiseren, en diegene die wij gaan toepassen bij onze textuur manipulatie, is synthese van de deformatie lattice. We relateren onze reguliere lattice L_r met de oorspronkelijke irreguliere lattice L_{ir} door de verplaatsing van elke knoop van de lattice als een vector voor te stellen. Vervolgens coderen we de vectoren als een textuur. We maken hier weer gebruik van de **dualiteit van het deformatieveld**. Voor de synthese van het algoritme gebruiken we eveneens Efros en Lung.

10.3 Synthese van de pseudo-reguliere textuur

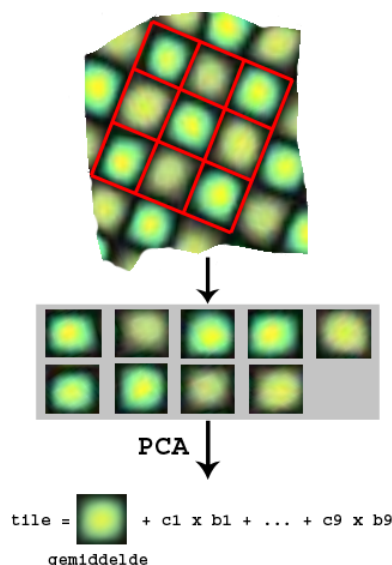
In de literatuur studie hebben we gezien dat er verschillende methodes zijn om de pseudo reguliere textuur te synthetiseren maar wij gaan de pseudo reguliere textuur synthetiseren door tegels te reconstrueren m.b.v. PCA volgens Matusik en deze te plaatsen op de reguliere lattice. Het PCA algoritme berekent een orthogonale basis voor de groep tegels. De output zijn deze basis tegels en een matrix met PCA componenten die ons informatie geven over de samenstelling van de tegels. Met deze informatie kunnen we een bereik van bemonsterwaarden voor elke basis onderzoeken en op die manier een onbeperkt aantal verschillende tegels samenstellen door lineaire combinaties.



Figuur 10.4: Toepassing van een geometrisch deformatieveld. De originele textuur(links) en dezelfde textuur met een geometrisch deformatieveld toegepast(rechts).(bron originele textuur: <http://www.mayang.com/textures/>)

Na de rectificatie van de irreguliere textuur nemen we de tegels uit de reguliere lattice en geven we ze als input aan ons PCA algoritme. De tegels moesten wel nog eerst met de x-as en y-as uitgelijnd worden. Elke tegel kan gezien worden als een één-dimensionale array waarden tussen 0 en 255. Een tegel van 50x50 pixels zal dan een array met een lengte van $50 \times 50 \times 3 = 7500$ opleveren. De array van elke tegel stoppen we samen in een tweedimensionale matrix D . Bij een input van 10 tegels zal D dan een dimensie hebben van $[7500 \times 10]$. Het PCA algoritme werkt echter niet rechtstreeks met deze waarden, we moeten eerst de gemiddelde tegel berekenen en deze aftrekken van elke tegel. Vervolgens berekenen we de covariantie matrix $C = D(t)' * D'$ en voeren we SVD (singulaire waarde decompositie) uit op C . Daar $C = V.S.V(t)$ zal S een diagonaalmatrix zijn met de eigenwaarden en V de eigenvectoren van C . De basis wordt dan gevonden door $U = D'.V.S$, in ons voorbeeld een matrix met dimensie $[7500 \times 10][10 \times 10][10 \times 10] = [7500 \times 10]$. Nu is $X = U(t).D'$ met X de matrix die de PCA componenten bevat, een kolom bevat de coëfficiënten van de basissen om één tegel uit de input samen te stellen.

Performantie PCA volgens Matusik Een eerste implementatie van het PCA algoritme gebruikte de traditionele methode. Dwz. de covariante matrix wordt berekend door de dataset(min gemiddelde) te vermenigvuldigen met zijn getransponeerde. Voor een dataset als de onze zou dit een covariante matrix opleveren van $[7500 \times 10][10 \times 7500] = [7500 \times 7500]$ wat een buitensporig geheugengebruik zou veroorzaken, en een al even trage uitvoering van het SVD algoritme. Dankzij Matusik kan het algoritme worden uitgevoerd op een covariante matrix van $[10 \times 7500][7500 \times 10] = [10 \times 10]$, een dus laag geheugen gebruik en een ogenblikkelijke beëindiging van het algoritme.



Figuur 10.5: PCA neemt enkele tegels uit de pseudo-reguliere textuur en zoekt een geschikte orthogonale basis en slaat de PCA componenten van de originele tegels op.

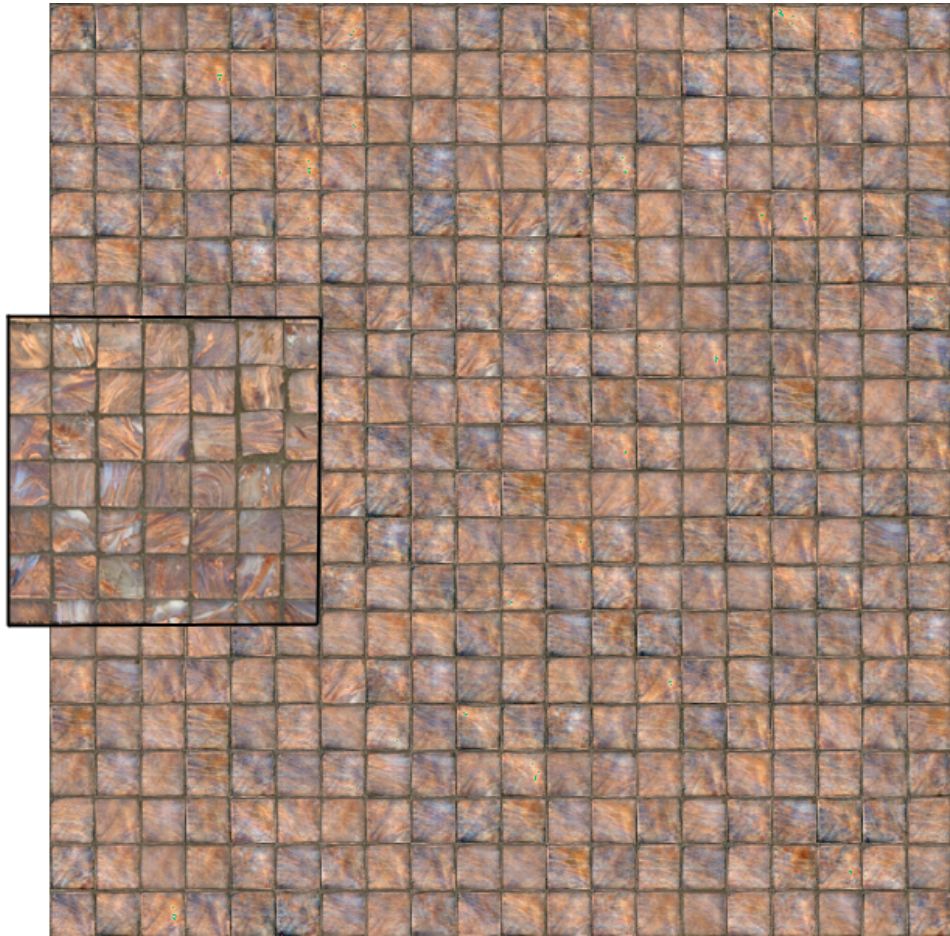
10.4 Manipulatie van de pseudo-reguliere textuur

In de vorige secties hebben we het geometrisch deformatieveld en de pseudo-reguliere textuur gesynthetiseerd. Dat is alles wat we nodig hebben om een irreguliere textuur te manipuleren.

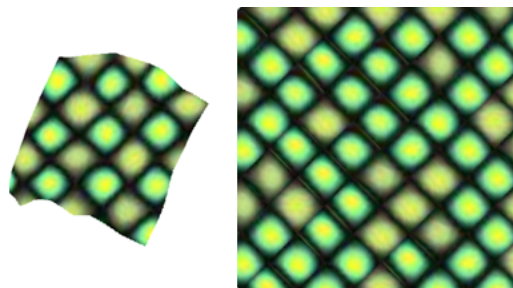
De geometrische regulariteit kunnen we naar wens regelen door te verrekken vanaf de reguliere mesh en daar het deformatieveld op toe te passen. Als maat voor de regulariteit kunnen de warping vectoren van de controlepunten eenvoudigweg geschaald worden met een factor. Wanneer deze factor 1 is zal de regulariteit een geometrische deformatie gelijk aan die van de originele textuur. Een factor 0 zal de lattice niet vervormen en de textuur zal geometrisch regulair zijn. Zie fig. 10.8.

De kleur regulariteit kan ook interactief worden aangepast dankzij de PCA analyse van de tegels. Voor het gesynthetiseerde beeld wil dit zeggen dat voor elke tile de bemonsterde PCA waarden worden vermenigvuldigd met een factor tussen 0 en 1 of zelfs hoger. Een factor 0 wil zeggen dat elke tegel van de textuur de gemiddelde tegels is en dus een kleurdeformatie van 0. Een factor 1 zal voor de tiles een kleurdeformatie betekenen getrouw aan het origineel. Zie fig. 10.9.

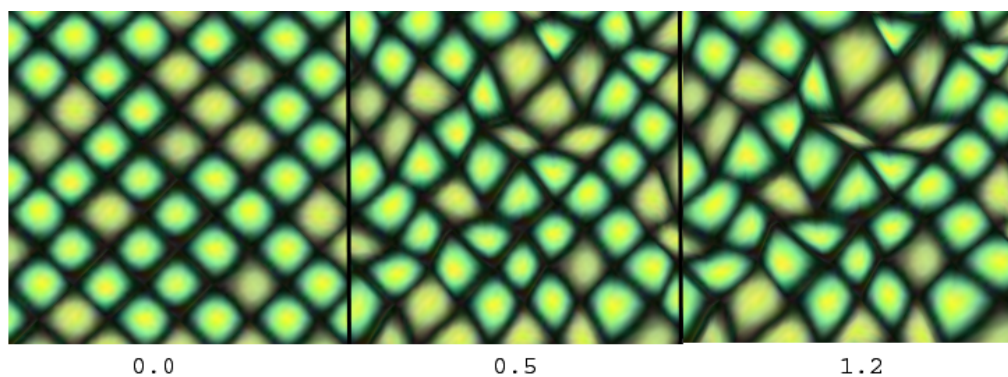
We kunnen interessante texturen creëren door geometrische deformatievelden toe te passen op de reguliere lattices van andere texturen.



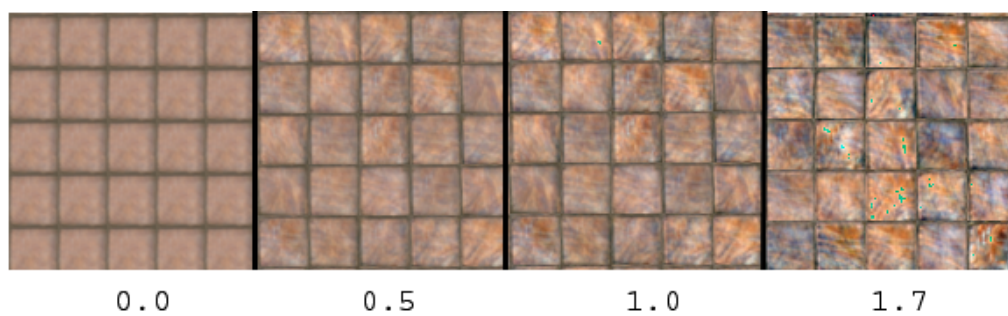
Figuur 10.6: Synthese: een textuur gesynthetiseerd met 400 unieke tegels. De coëfficiënten zijn gesampled binnen bereik van de pca componenten en hebben dus een kleurdeformatie gelijk aan die van de originele textuur. (bron originele textuur: <http://www.mayang.com/textures/>)



Figuur 10.7: Synthese: nog een synthese van een pseudo-reguliere textuur



Figuur 10.8: Geometrische deformatie manipulatie.



Figuur 10.9: Kleurdeformatie manipulatie.



Figuur 10.10: Een gesynthetiseerde textuur werd geplaatst op de reguliere lattice van de textuur uit fig. 10.9 en het bijhorende deformatieveld werd toegepast.

Hoofdstuk 11

Uitbreidingen

11.1 Textuur vervanging

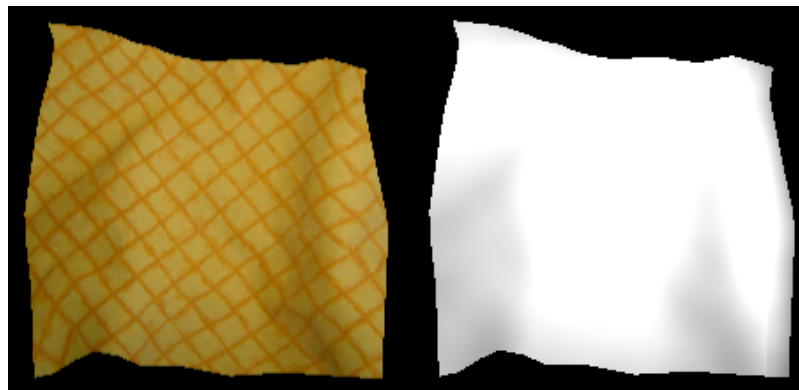
Een toepassing die ook van pseudo-reguliere texturen gebruik kan maken is textuur vervanging. Dit werd niet geïmplementeerd in deze thesis maar is wel een logische extensie aangezien de methode gebaseerd is op de analyse van het geometrisch en het belichtingsdeformatieveld van de te vervangen textuur. Deze toepassing is dus een mogelijke uitbreiding en we bespreken elke stap van de methode.

We relaten de textuur met zijn reguliere oorsprong zodat we het geometrische deformatieveld kunnen modelleren(pseudo-regulier naar irregulier). We gebruiken de per-pixel modellering, later zullen we het deformatieveld gewoon toepassen op elke vertex van de getextureerde mesh. Wanneer we nu een nieuwe textuur zouden plaatsen op de reguliere textuur en terug de inverse deformatie toepassen zouden de geometrische distorties in de textuur voorkomen maar het resultaat zou niet overtuigend zijn. We missen namelijk de belichtingseffecten en die zullen we nu onderzoeken.

11.1.1 Het belichtingsdeformatieveld

Vooraleer we de belichtingsmap kunnen bepalen van de textuur moeten we ze eerst rectificeren. We hebben dan een regulier patroon van tegels en dit is belangrijk voor het blurren van het beeld. We maken namelijk de aanname dat de gemiddelde kleur van het geblurrede beeld afkomstig is van de textuur en dit kan enkel waar zijn wanneer de filter voor het beeld groot genoeg was én de features van de textuur een regulier patroon hadden.

Een andere aanname die we maken is dat de belichting verantwoordelijk is voor de laagfrequente veranderingen in de textuur. We splitsen de textuur T op in een reflectie component R en een shading component S . Dus $T = R \times S$ waarbij $\times$ een pixel-gewijze vermenigvuldiging is van de twee resulterende componenten. We gaan als volgt te werk om S te zoeken.



Figuur 11.1: (links) Een pseudo-reguliere textuur. (rechts) De belichtingsmap uit deze textuur.

- $I = S \times R$ en dus ook $\log(I) = \log(S) + \log(R)$
- We blurren $\log(I)$, hiervoor passen we een Gaussiaanse filter met een grootte afhankelijk van de feature size van de textuur op. De filter mag ook niet te scherp zijn en we passen dus een standaardafwijking toe van circa $1/4$ de grootte van het masker.
- We nemen aan dat de gemiddelde kleur afkomstig is van de reflectie van de textuur dus we maken een schatting van $\log(S)$ door de gemiddelde kleur af te trekken van het geblurrede $\log(I)$. $\log(S) = \text{blur}(\log(I)) - \text{gemiddelde}(\text{blur}(\log(I)))$
- De reflectantie wordt geschat door $\log(R) = \log(I) - \log(S) + \text{gemiddelde}(\text{blur}(\log(I)))$ en bijgevolg is $R = \exp(\log(R))$.

Omdat we een Gaussiaanse filter gebruiken op het beeld om de laagfrequente veranderingen in het beeld op te sporen, gaan de hoogfrequente veranderingen verloren. Dit is meestal geen probleem omdat de hoogfrequente veranderingen in de textuur meestal worden veroorzaakt door het patroon van het textuur en niet door belichting. Het kan wel een probleem opleveren bij harde schaduwranden, het algoritme denkt dan immers dat deze horen bij de reflectie van de textuur. Een mogelijke verbetering hierbij zijn bilaterale filters; een extra gewichtsmasker wordt dan vermenigvuldigd met het Gaussiaanse masker zodat detail van randen behouden blijft.

11.1.2 Textuur vervangen

Alles wat we nu nog hoeven te doen is een andere textuur plaatsen op de reguliere grid en pixel gewijs vermenigvuldigen met de belichtingsmap. We warpen de textuur terug met het inverse geometrische deformatieveld en het

resultaat is een textuurvervanging met dezelfde deformatie onder dezelfde belichting.

11.2 Deformatieveld analogieën

Een tweede mogelijke uitbreiding zijn deformatieveld analogieën. De methode steunt op de veronderstelling dat er een sterke associatie is tussen het geometrische deformatieveld en het belichtingsdeformatieveld. De uitgelezen techniek om deze associatie uit te buiten is beeld analogieën uit [LLH04]. Men stelt dat gegeven een geometrisch deformatieveld (geïnterpreteerd als textuur), zijn overeenkomstige belichtingsmap en een derde deformatieveld (gewoonlijk gesynthetiseerd) er een analogie kan gevonden worden voor de belichtingsmap. Op deze manier kunnen we beeldsynthese doen op een manier die voor andere synthese methoden onmogelijk is. De basiscomponenten werden reeds geïmplementeerd, d.w.z het deformatieveld kunnen we berekenen en zijn bijhorende belichtingsmap eveneens. Een stabielere deformatieveld synthese is wel nodig om het algoritme succesvol te laten zijn. Voor de beeld analogie zelf kunnen we het algoritme gebruiken dat ter beschikking is gesteld op de website van de paper.

Deel IV

Conclusie

We hebben in deze thesis een studie gemaakt van enkele veelbelovende methoden om texturen te manipuleren. We hebben ondervonden dat deze tools toelaten de textuur op een veel diepgaander niveau te bewerken. Sommige tools zijn relatief eenvoudig te implementeren zoals het concept van zelf-similariteit uit [BD02]. Alhoewel de werking van deze algoritmen afhankelijk was van verschillende factoren en dus wel wat trial en error vereisen van de programmeur (b.v. het warpen van de textuur) werken deze algoritmen goed en met een minimale input van de gebruiker.

We hebben ook meer geavanceerde methodes van textuur manipulatie bestudeerd uit [LLH04]. Deze tools laten toe de textuur op een multi-modale wijze te wijzigen, dwz. het modelleren en manipuleren van de geometrische, kleur en belichtingsdeformatievelden. Deze methoden vereisen meer input van de gebruiker maar kunnen op een betrouwbare manier de textuur fundamenteel veranderen. Bijvoorbeeld de identificatie van de irreguliere lattice door de gebruiker is een moeizaam proces en laat nog ruimte over voor verbeteringen. We hebben enkele aanpassingen gedaan aan de originele implementatie van deze algoritmen. Zo hebben we de numerieke berekening van de reguliere lattice opgelost door een mass-spring systeem. Het geometrische deformatieveld wordt ook niet exact hetzelfde berekend, i.p.v. een bijectieve relatie tussen de reguliere en irreguliere textuur vinden we dat een relatie van reguliere naar irreguliere textuur voldoende is voor de textuur te manipuleren. Als laatste is het belichtingsdeformatieveld berekend door een intuïtievere methode gebaseerd op vereenvoudigende aannames. Hierdoor boet het weinig in aan nauwkeurigheid maar wordt de implementatie sterk vereenvoudigd.

Ook hebben we een textuur synthese algoritme geïmplementeerd dat voor de irreguliere en reguliere textuur klassen zeer goede, respectievelijk betere resultaten geeft als voorgaande methodes zoals Efros en Lung, Wei en Levoy, Ashikhmin, en anderen.

Bibliografie

- [Ash01] Michael Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.
- [BD02] Stephen Brooks and Neil Dodgson. Self-similarity based texture editing. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 653–656, New York, NY, USA, 2002. ACM Press.
- [EF01] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.
- [EL99] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.
- [FH04] Hui Fang and John C. Hart. Textureshop: texture synthesis as a photograph editing tool. *ACM Trans. Graph.*, 23(3):354–359, 2004.
- [HJO⁺01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340, New York, NY, USA, 2001. ACM Press.
- [KR97] T. Alan Keahey and Edward L. Robertson. Nonlinear magnification fields. In *INFOVIS '97: Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, page 51, Washington, DC, USA, 1997. IEEE Computer Society.
- [KSE⁺03] Vivek Kwatra, Arno Schdl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003.

- [LCS95] Seung-Yong Lee, Kyung-Yong Chwa, and Sung Yong Shin. Image metamorphosis using snakes and free-form deformations. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 439–448, New York, NY, USA, 1995. ACM Press.
- [LL03] Yanxi Liu and Wen-Chieh Lin. Deformable texture: the irregular-regular-irregular cycle. Technical Report CMU-RI-TR-03-26, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2003.
- [LLH04] Yanxi Liu, Wen-Chieh Lin, and James H. Hays. Near regular texture analysis and manipulation. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3):368 – 376, August 2004. The following movies accompany the paper: * NRTAM.avi [110M5, divx 5.11 encoded] * NRTAM.mp4 [68M9, QuickTime].
- [LTL05] Yanxi Liu, Yanghai Tsin, and Wen-Chieh Lin. The promise and perils of near-regular texture. *Int. J. Comput. Vision*, 62(1-2):145–159, 2005.
- [MPBM03] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Trans. Graph.*, 22(3):759–769, 2003.
- [OCDD01] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 433–442, New York, NY, USA, 2001. ACM Press.
- [TLR01] Yanghai Tsin, Yanxi Liu, and Visvanathan Ramesh. Texture replacement in real images. In *IEEE Conference on Computer Vision and Pattern Recognition*, December 2001.
- [WL00] 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.
- [XGS00] Ying-Qing XU, Baining GUO, and Harry SHUM. Chaos mosaic: Fast and memory efficient texture synthesis. Technical report, 2000.