Video-gebaseerde Computeranimatie

Stijn STEEGEN

promotor :
Prof. dr. Frank VAN REETH

Universiteit Maastricht

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica afstudeerrichting multimedia

universiteit hasselt

Animatie van video sprites

Thesis voorgedragen tot het behalen van de graad van
master in de informatica/ICT/kennistechnologie

Stijn Steegen
Promotor: Frank van Reeth
Begeleider: William van Haevre
2005 - 2006

Samenvatting

Computeranimatie probeert door objecten te animeren deze objecten realistischer weer te geven. Geanimeerde objecten zijn meestal interessanter dan statische objecten, omdat ze er levendiger uitzien. Het creëeren van animaties van complexe objecten, zoals dieren, is niet zo evident. Dit komt door hun complexe structuur en door hun niet egaal oppervlak, vaak bezet met vele kleine haren. Het modelleren van zulke objecten is complex en erg tijdrovend. Verder is het animeren van 3D-objecten voor de animator een trail-and-error proces, omdat dit vaak handmatig gebeurt.

Video-based rendering en video-based animation proberen het werk van de animator te verlichten. Vertrekkende van een hoeveelheid trainingsdata van een object kunnen nieuwe animaties met deze data gemaakt worden. Doordat het object in zijn natuurlijke omgevingen gefilmd werd, is het realisme gegarandeerd. Deze videosequenties kunnen vervolgens als video textures opgeslagen worden. Door de volgorde van de frames van de video texture te ver-anderen, kunnen dan nieuwe video sequenties bekomen worden. Dit geeft oneindige loops die globaal het object dezelfde bewegingen laten uitvoeren als in de originel videosequentie. Hierbij moet opgepast worden dat er geen zichtbare herhalingen zijn. Verschillende algoritmen om goede animaties met video textures te maken komen in deze thesis aan bod.

Hoewel video textures de animator in staat stelt oneindige animaties te maken van objecten, is deze techniek toch beperkt. Enkel de volgorden van afspelen van de frames van de video texture kunnen immers veranderd worden. Video sprites gaan een stap verder. Uit elke frame wordt enkel het gefilmde object gehaald. De frames van deze videosequentie kunnen dan gezien worden als sprites. Deze sprites kunnen vrij over een oppervlak bewogen worden, en hebben een positie en een velocity. Door het toevoegen van kostfuncties kan een nieuwe animatie van dit object bekomen worden. Deze kostfuncties bepalen hoe de animatie er zal uitzien door de positie en de velocity van de sprites te beïnvloeden. Zo is het mogelijk het object een bepaald pad te laten volgen, terwijl het allerlei andere objecten moet vermijden. Hoewel deze techniek specifiek ontworpen is voor kleine dieren, kan dit ook toegepast worden op andere objecten, zoals bijvoorbeeld een vliegtuig in volle vlucht. In deze thesis komen enkele algoritmen aan bod om zulke animaties met video sprites te maken, alsook andere aspecten die van belang zijn bij het maken van deze animaties.

Inhoudsopgave

1 Introductie **5**

2 Situering binnen computer graphics **7**
2.1 Render-methodes 7
2.1.1 Geometrie-based rendering 7
2.1.2 Image-based rendering 11
2.1.3 Video-based rendering 12
2.2 Animatie methodes 12
2.2.1 Sprite animaties 13
2.2.2 Mesh animaties 14
2.2.3 Key-framing 14
2.2.4 Kostfuncties 15
2.3 Vergelijking van render- en animatiemethoden 15

3 Video-based rendering **16**
3.1 Motivatie 16
3.2 Rendering 16
3.2.1 Acquisition 16
3.2.2 Background subtraction 17
3.2.3 Graph cut 20
3.2.4 Visual hull 22
3.2.5 View-dependent rendering 22
3.2.6 3D-TV 23
3.2.7 Compressie 25
3.3 Animatie 25
3.3.1 3D-modellen 25
3.3.2 2D-modellen 26
3.4 Slot 26

4 Video textures **27**
4.1 Textures 27
4.1.1 3D textures 28
4.1.2 Video textures 30
4.2 Transities voor video textures 30
4.2.1 Kleureigenschappen 30
4.2.2 Transities verbeteren 34
4.3 Conclusie 34

5 Animaties met video textures **35**
5.1 Voorloper: video rewrite . . . 35
5.2 Video textures . . . 36
5.2.1 Analyse . . . 36
5.2.2 Synthese . . . 38
5.2.3 Uitbreidingen . . . 41
5.3 Symmetrie uitbuiten . . . 42
5.3.1 Analyse . . . 42
5.3.2 Synthese . . . 43
5.3.3 Hybride methode . . . 44
5.4 Cartoon textures . . . 44
5.5 Toepassingen . . . 45
5.5.1 Panoramic video textures . . . 45
5.5.2 Graphcut textures . . . 47
5.5.3 Flow-based video synthesis and editing . . . 48
5.5.4 Auto-regressive process . . . 49
5.5.5 Motion graph . . . 49
5.6 Slot . . . 49

6 Video sprites **50**
6.1 Definitie . . . 50
6.1.1 3D video sprites . . . 50
6.2 Segmentatie van video sprites . . . 50
6.2.1 Vergelijking tussen background subtraction/graph cut . . . 51
6.3 Perspectieve correctie . . . 51
6.4 Transities voor video sprites . . . 51
6.4.1 Kleureigenschappen . . . 52
6.4.2 Oppervlakte . . . 52
6.4.3 Velocity . . . 52
6.5 Conclusie . . . 52

7 Animaties met video sprites **53**
7.1 Transities bepalen . . . 53
7.2 Controle over video sprites . . . 54
7.2.1 Beam search . . . 54
7.2.2 Hill-climbing optimalisatie . . . 54
7.3 Animatie toevoegen . . . 56
7.3.1 Locatie constraint . . . 56
7.3.2 Pad constraint . . . 56
7.3.3 Anti-collision constraint . . . 57
7.3.4 Frame constraint . . . 57
7.4 Uitbreidingen . . . 59
7.4.1 Animatie constraints . . . 59
7.4.2 Toepassingen . . . 63
7.5 Slot . . . 64

8 Implementatie **65**
8.1 Analyse 65
8.1.1 Vergelijking van verschillende metrieken 65
8.1.2 Andere kleurenruimte 67
8.1.3 Post processing 68
8.1.4 Conclusie 68
8.2 Video textures 68
8.2.1 Origineel video texture algoritme 68
8.2.2 Algoritme met symmetrische aanpak 70
8.2.3 Vergelijking video textures met symmetrische aanpak 71
8.3 Video sprites 72
8.3.1 Analyse 72
8.3.2 Synthese 73
8.3.3 Resultaten 75
8.3.4 Discussie 76

9 Conclusie **77**

Hoofdstuk 1

Introductie

Computer graphics is het domein waar visuele afbeeldingen gegenereerd worden m.b.v. computers. Hierbinnen is computeranimatie een belangrijk onderdeel. Zeker in de film- en game-industrie spelen animaties een cruciale rol om objecten realistisch weer te geven. Het maken van realistische animaties met computers is vaak een ingewikkeld proces. Zeker voor objecten met een complexe architectuur is het creëeren van een realistisch computermodel een heel karwei. Dit vraagt immers veel inspanningen van de animator. Ook het animeren van zulke objecten is erg tijdrovend, omdat de animator dit vaak handmatig moet doen.

Video-based animation probeert enigszins het werk van de animator te verlichten door gebruik te maken van eerder opgenomen videosequenties. Omdat het object in zijn natuurlijke omgeving gefilmd wordt, is dit per definitie realistisch. Wanneer dan deze data gebruikt wordt om animaties te maken, zullen ze er realistischer uitzien dan wanneer andere animatietechnieken gebruikt werden.
Eerst volgende hoofdstuk geeft een situering van video-based rendering, en meer bepaald video-based animation, binnen het domein van de computer graphics. Verschillende render- en animatiemethoden er worden besproken en met elkaar vergeleken. Ook wordt vermeld waarom deze technieken in sommige situaties onvoldoende goede resultaten leveren, en waarom video-based animation in deze situaties wel realistische resultaten zal geven.

Daarna volgt een meer gedetailleerde bespreking van video-based rendering, met enkele toepassingen, en de plaats van video sprite animaties binnen dit domein. Zo komen ook verschillende basisbegrippen van video-based rendering aan bod, zoals segmentatie technieken om bepaalde objecten uit hun achtergrond te halen. Ook worden enkele methoden van video-based animation aangehaald.

Het vierde hoofdstuk biedt een kennismaking met video textures, en waarom deze gebruikt worden. Er worden ook metrieken aangehaald die toelaten deze video textures met elkaar te vergelijken. Deze metrieken bepalen of 2 videoframes na elkaar kunnen worden afgespeeld zonder visuele artifacten. Verder worden ook enkele technieken besproken om de transities tussen de videoframes te verbeteren.

Vervolgens komt een hoofdstuk aan bod over animaties die mogelijk zijn m.b.v. video textures. Er worden algoritmen uitgelegd om animaties te maken door de eerder besproken

metrieken te gebruiken. Verschillende aspecten die de kwaliteit van de animatie bepalen worden aangehaald. Verder worden ook verschillende toepassingen van video textures kort besproken.

Hoofdstuk 6 behandelt video sprites, en geeft een definitie en een motivatie om video sprites te gebuiken. Ook belangrijke aspecten om deze video sprites te bekomen worden besproken. Dan volgt een bespreking van handige metrieken om goede sprite paren te bepalen. Door video sprites te gebruiken zijn er immers nieuwe eigenschappen. Een belangrijke hiervan is de velocity van de sprite. Deze laat de sprite toe naar bepaalde positie te bewegen. Maar ook andere eigenschappen van video sprites komen aan bod.

Daarna komen de animaties gebaseerd op video sprites aan bod. Er zal aangetoond worden dat video sprites een grotere waaier aan animaties toelaten dan video textures. Verder worden ook algoritmen besproken om controle over deze animaties te krijgen, en welke animaties mogelijk zijn. Verschillende constraint om de animaties te maken worden besproken, alsook mogelijke toepassingen ervan. Ook worden enkele eigen uitbreidingen besproken.

Ten slotte volgt een bespreking van mijn implementatie, met eigen bevindingen. Zo worden de verschillende metrieken die mogelijk zijn voor video textures en video sprites experimenteel met elkaar vergeleken. Dan komen de belangrijke aspecten van de eigenlijke implementatie aan bod voor zowel video textures als video sprites, samen met de resultaten. Verder worden mijn resultaten ook vergeleken met resultaten van anderen, om zo een beter beeld te bieden. Deze thesis wordt afgerond met een conclusie.

Hoofdstuk 2

Situering binnen computer graphics

Computer graphics probeert afbeeldingen van objecten te genereren met behulp van computers. Hierbij wordt vaak getracht het realisme van deze objecten zo hoog mogelijk te houden. In dit hoofdstuk worden eerst enkele manieren om objecten af te beelden aangehaald, en daarna volgen dan methoden om deze objecten te animeren. Er worden tevens beperkingen van deze methoden aangehaald, zodat het gebruik van video-based rendering en video-based animation duidelijk wordt voor de lezer.

2.1 Render-methodes

Rendering is het proces waarbij een afbeelding van een bepaald model vanuit een zeker standpunt wordt berekend. Het model is vaak een 3 dimensionaal object, met een zekere geometrie, textuur en materiaaleigenschappen die de belichting beïnvloeden. Tijdens het renderen worden verschillende eigenschappen, zoals schaduwen, belichting, texture mapping,... in rekening gebracht. Er bestaan vele methoden om een model te renderen, elk met typerende eigenschappen. In deze sectie worden de belangrijkste render-methoden kort besproken.

2.1.1 Geometrie-based rendering

Geometrie-based renderalgoritmen vertrekken van een expliciete voorstelling van het model om vervolgens een 2 dimensionale projectie ervan in schermcoördinaten te berekenen. Er zijn grofweg 4 grote categorieen van geometrie-based rendermethoden [wik]. Deze zijn:

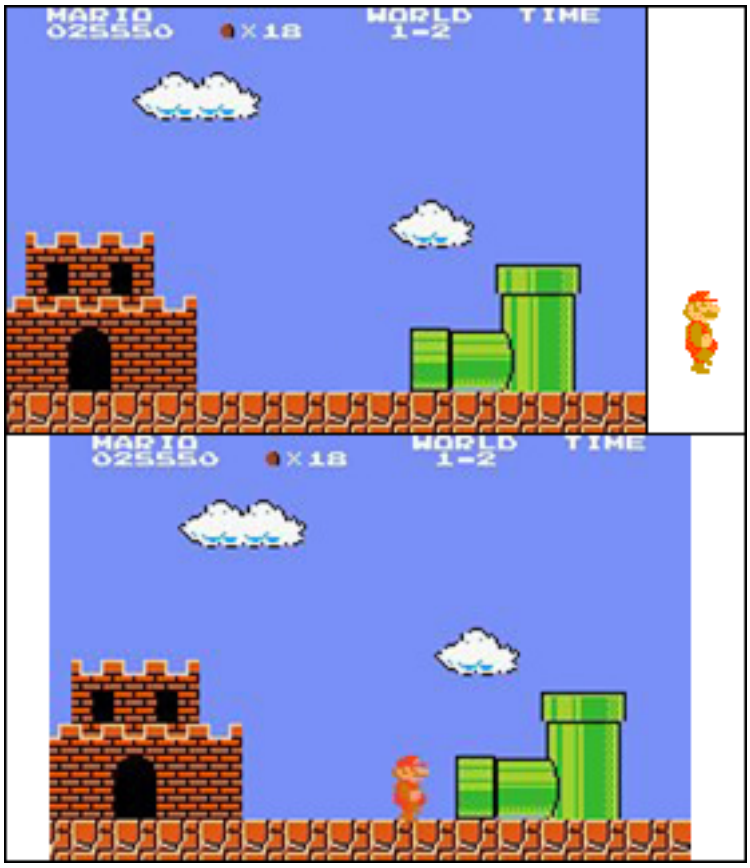
- Rasterization
- Ray casting
- Ray tracing
- Radiosity

Rasterization

Rasterization [Bou70] is waarschijnlijk de meest gebruikte methode. Hierbij worden alle objecten om de beurt vanuit een zeker standpunt geprojecteerd op het scherm. Dit is mogelijk in 2D, en werd uitgebreid naar 3D.

2D rendering:

Er zijn verschillende manieren om virtuele scènes te creëren. Een eerste manier om een virtuele scène te creëren d.m.v rasterizatie is door gebruik te maken van 2-dimensionale afbeeldingen. Deze techniek ontstond in de jaren 50, en is gebaseerd op vector graphics. Vaak wordt hierbij gebruik gemaakt van sprites. Dit zijn 2 dimensionale afbeeldingen of animaties van een bepaald object die deel uitmaken van een scène. Deze worden dan bovenop de scène "geplakt" zodat het lijkt of het object zich in de scène bevindt. Dit wordt geïllustreerd in onderstaande tekening.



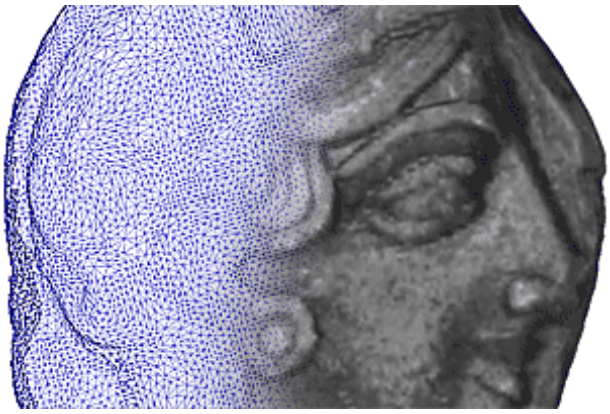
Figuur 2.1: Opbouw van sprite rendering voor het spel "Mario" van Nintendo. Linksboven is de achtergrond, bestaande uit verschillende sprites. Rechtsboven is de Mario sprite. Onder is het resultaat bekomen door de Mario sprite op de achtergrond te projecteren.

Deze techniek is erg beperkt. Ten eerste zijn er slechts bewegingen in 2 richtingen mogelijk, namelijk horizontaal en verticaal. Het is bovendien bijzonder moeilijk diepte op te wekken. Ook lijken de sprites niet realistisch door het ontbreken van diepte. Vaak wordt een object door 1 of enkele sprites voorgesteld, zodat voor kleine veranderingen van

viewpoint vaak geen nieuwe sprite voorhanden is. Dit levert slechte animaties, en daarom wordt deze techniek steeds minder gebruikt. Echter in de beginperiode van graphics kende deze techniek een groot succes. Een voordeel is dat deze techniek erg eenvoudig en snel is.

3D rendering:

Door de derde dimensie toe te voegen, wordt diepte gecreëerd. Door de diepte lijken de objecten, alsook de hele scène, realistischer. Maar door toevoeging van de derde dimensie treden er nieuwe moeilijkheden op. Eerst en vooral is het voorstellen van objecten ingewikkelder. Deze worden nu door middel van een mesh voorgesteld. Dit is een lijst van polygonen, en een polygon is een veelhoek in de ruimte. Deze lijst kan voor grote, oneffen objecten erg groot worden. Het modelleren van de mesh is dus een ingewikkeld proces, en vergt grote inspanningen van de animator. Ook het modelleren van mensen en dieren is erg moeilijk, en daarom lijken deze models vaak onvoldoende realistisch en zijn ze onbruikbaar voor sommige toepassingen zoals films.



Figuur 2.2: Voorbeeld van een polygonmesh. Links zijn de verschillende polygonen zichtbaar, en rechts is het getexturede resultaat. Duidelijk is te zien dat de complexiteit veel vraagt van de animator.

Het 3-dimensionaal werken compliceert het berekenen van vele eigenschappen van de scène, zoals clipping, schaduwen, belichting... Daarom is 3-dimensionale rasterization niet alleen moeilijker dan diens 2-dimensionale variant, maar ook veel trager. Hierdoor worden vaak beperkingen op de complexiteit van de scène en op de kwaliteit van de rendering gelegd. Maar dit probleem wordt geleidelijk aan opgelost door snellere computers en goedkopere hardware.

Ray casting

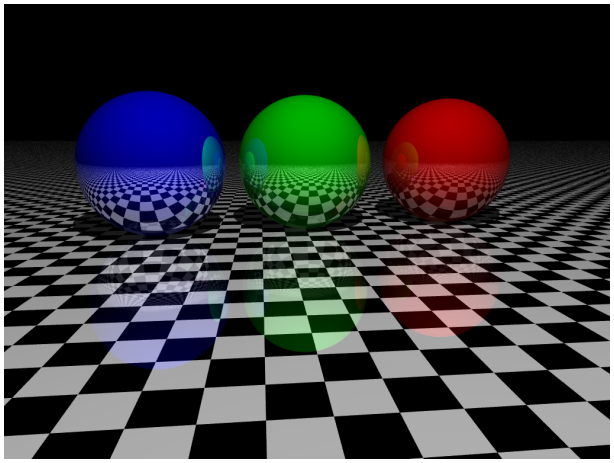
Een heel ander methode van geometrie-based rendering is ray casting. Ray casting werd voor het eerst voorgesteld door A. Appel in 1968. De scène wordt gezien vanuit een welbepaald viewpoint, en berekend de objecten enkel op basis van hun geometrie en heel eenvoudige belichting. Vanuit het viewpoint wordt dan lichtstralen naar de lichtbronnen gestuurd, maar deze stoppen bij een intersectie. Hierdoor zijn accurate belichtingsmodellen uitgesloten.

Door zijn hoge snelheid werd ray casting met succes toegepast in de beginjaren van de real-time 3D videogames, zoals Wolfenstein 3d[1], Duke Nukem 3d[2] en Doom[3].

Figuur 2.3: Screenshot van Doom, dat volledig is gemaakt m.b.v. ray casting.

Ray tracing

Ray tracing [Whi80] is verwant met ray casting, maar gebruikt geavanceerde belichtingsmodellen, zoals Monte Carlo technieken. Net zoals bij ray casting worden vanuit het oogpunt lichtstralen door de hele scène afgeschoten. Wanneer deze lichtstralen een oppervlak raken, worden ze gereflecteerd, gerefracteerd en geabsorbeerd. Hierdoor zijn meer geavanceerde belichtingen mogelijk, en is ray tracing veel realistischer dan ray casting.
Een nadeel is dat ray tracing veel trager is, omdat per intersectie vele nieuwe lichtstralen worden afgeschoten. Hoewel ray tracing al erg oud is (1980), is real time ray tracing een huidig topic van onderzoek [PBMH02].



Figuur 2.4: Afbeelding bekomen door ray tracing. Vooral de reflecties en refracties geven ray tracing een realistische look.

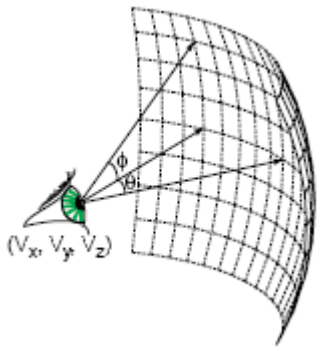
[1]http://www.idsoftware.com/games/wolfenstein/wolf3d/
[2]http://www.3drealms.com/duke3d/
[3]http://www.idsoftware.com/games/doom

Radiosity

Radiosity [GTGB84] is een oudere rendermethode om goede belichting in een scène te berekenen. Radiosity probeert de manier waarop gereflecteerd licht een gebied belicht te simuleren, en niet zozeer de reflectie in een punt naar een ander oppervlak zoals bij ray tracing. De belichting wordt niet lokaal, maar globaal berekend. Als gevolg hiervan zijn speculaire reflecties en refracties onmogelijk. Om deze reden wordt ray tracing vaak boven radiosity verkozen, en wordt radiosity nu nog zelden gebruikt.

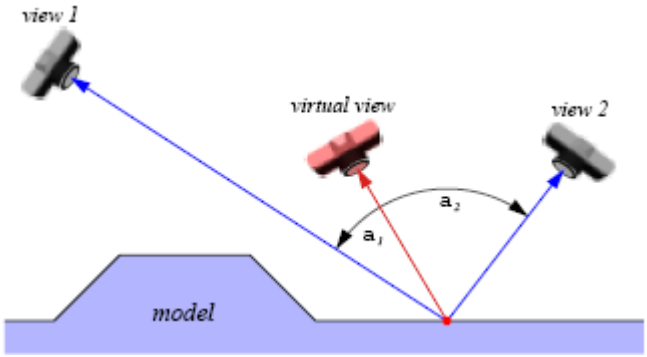
2.1.2 Image-based rendering

Vaak is het visualiseren van foto realistische beelden erg moeilijk. Bij geometrie-based algoritmen zijn vele polygonen nodig, en elk heeft zijn eigen textures en belichting. Goede schaduwen en belichting zijn bijzonder duur, omdat dit per pixel per polygon berekend moet worden.
Typisch wordt in computer graphics geprobeerd een 3-dimensionale wereld te construeren en deze dan op een 2-dimensionaal vlak te projecteren. Computer vision probeert net afbeeldingen te interpreteren op zoek naar 3-dimensionale informatie. Image-based rendering [MB95] [Che95] vertrekt van een reeks foto's, die dan gebruikt worden bij het opstellen van een 3-dimensionaal beeld van de scène [Da99]. Door het gebruik van echte foto's kunnen zeer realistische scènes gerenderd worden. Vooral de belichting van de scènes speelt hier een belangrijke rol.
Het fundamenteel concept achter image-based rendering is de plenoptische illuminatie functie. Deze functie beschrijft de hoeveelheid licht op een bepaalde plaats. De functie is 7-dimensionaal: een ray wordt gedefiniëerd door een positie(V_x, V_y, V_z), een richting(θ,φ), de golflengte(γ) en de tijd(t). Image-based modelling en rendering proberen met de plenoptische functie nieuwe afbeeldingen te creëren uit andere afbeeldingen.



Figuur 2.5: De plenoptische functie bij image-based rendering, hier weergegeven met slechts 5 dimensies.

Een eerste methode maakt gebruik van vele viewpoints om zo voldoende beelden te hebben, en deze te combineren tot een environment map [Gre86]. Deze environment map kan vervolgens gebruikt worden als een texture van de omgeving. Nieuwe viewpoints kunnen dan gecreëerd worden door de texture terug op een vlak in de kijkrichting van de gebruiker te projecteren.



Figuur 2.6: creëren van een nieuwe view uit een reeks afbeeldingen. Door de afbeeldingen bekomen vanuit view 1 en 2 te combineren, wordt een nieuwe afbeelding voor de virtuele view bekomen.

Een tweede methode voegt diepte toe door een depth map te berekenen vanuit de verschillende viewpoints. Met standard texture mapping kunnen dan nieuwe viewpoints berekend worden.

Het nadeel van image-based rendering is de rendertijd. Door de complexiteit van de plenoptische functie is deze namelijk niet real-time, hetgeen deze techniek onbruikbaar maakt voor vele toepassingen.
Vaak wordt image-based rendering gecombineerd met geometrie-based rendering. Dit zijn dan de hybride modellen. Hier worden de voordelen van beide methoden gebruikt: door gebruik te maken van foto's wordt een hoge kwaliteit bereikt, en er worden geometrie-technieken gebruikt om het probleem van diepte op te lossen. Deze algoritmen zijn echter vaak complexer en trager.

2.1.3 Video-based rendering

Video-based rendering is een subdomein binnen image-based rendering, met toevoeging van temporele informatie. I.p.v. stilstaande foto's, zoals bij image-based rendering, wordt er gebruik gemaakt van bewegende beelden. In het volgende hoofdstuk zal dieper ingaan worden op de manieren van analyseren en verwerken van deze video sequenties.

2.2 Animatie methodes

Animatie is niets meer dan het creëren van een illusie: vele stilstaande afbeeldingen worden snel na elkaar afgespeeld zodat deze afbeeldingen lijken te bewegen. De snelheid waarmee de afbeeldingen elkaar opvolgen varieert van toepassing tot toepassing: cartoo-nanimaties gebruiken een snelheid van 12 frames per seconden, maar voor videogames is een 60 fps wenselijk. De illusie is analoog met de werking van bijvoorbeeld een tv, waar 24 beelden per seconde getoond worden.
Computeranimatie berekent de animaties met een computer. De animaties kunnen ook buiten het computer domein gebruikt worden, denk maar aan films waar de laatste jaren meer en meer CGI (computer generated imagery) gebruikt wordt.

2.2.1 Sprite animaties

2D-raserization gebruikt sprites om de scène voor te stellen. Deze sprites kunnen gebruikt worden om animaties te maken door per frame de sprite door een andere sprite te vervangen. Voor een bepaalde animatie moet dan voor elke frame een sprite getekend worden door de animator. Hieronder is een voorbeeld van een zwemanimatie van 6 frames. De animator moet voor elke frame de sprite zelf helemaal tekenen, hetgeen voor langere sequenties een heel karwei is. Wanneer onvoldoende sprites gebruikt worden, zal de animatie niet vloeiend zijn.

Figuur 2.7: 6 sprites voor een zwemanimatie van het spel “Mario”, handgetekend door de animator. Door per frame een volgende sprite uit deze reeks te nemen, wordt de zwemanimatie bekomen.

Bill-boarding

Sprite animaties kunnen ook binnen 3D-rendering gebruikt worden. Dit gebeurt d.m.v. bill-boards. Dit zijn rechthoekige vlakken die loodrecht op de richting van de camera staan. Hierdoor zijn ze vanuit elk standpunt zichtbaar. Vaak worden deze bill-boards geblend met de achtergrond, om zo de illusie op te wekken dat de sprite daadwerkelijk in de scène aanwezig is[4].

Figuur 2.8: Een vuuranimatie gemaakt m.b.v. een particle system. Elk particle bevat een sprite die m.b.v. een bill-board gerenderd wordt. Deze bill-boards worden dan met elkaar geblend.

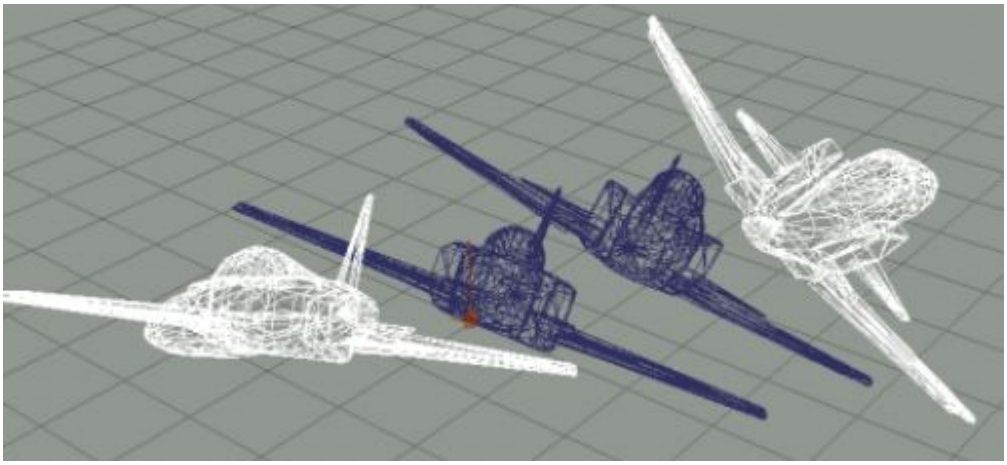
[4]Eigen animatie gemaakt voor het vak architectuur en algoritmen van computer games.

2.2.2 Mesh animaties

Door de geometrie van een mesh van een object tijdelijk te veranderen, zijn animaties binnen 3D mogelijk. Morph target animation en skeleton animation zijn hiervan voorbeelden. Morph target animation laat de animator toe de vertices te manipuleren om zo de gewenste animatie te bekomen. Hierdoor heeft de animator echter veel werk. Hij moet immers alle vertices handmatig aanpassen elk frame van de animatie. Daarom wordt vaak gebruik gemaakt van key-framing, besproken in volgende sectie.
Bij skeleton animation bestaat het object uit 2 (of meer) delen: het skelet en de skin (= uiterlijk). De vertices van het skelet kunnen dan handmatig aangepast worden. Dit heeft als resultaat dat de bovenliggende skin, die over het skelet zit, mee geanimeerd wordt. Door aanpassing van het skelet moet de animator niet meer alle vertices zelf aanpassen, enkel de joints van het skelet dienen aangepast te worden. Daardoor verliest de animator een deel van de controle over het object. Dit kan deels opgelost worden door toevoeging van key frames.

2.2.3 Key-framing

Vele van de vroegere computeranimatie systemen waren key-frame systemen [BW71] [Par05]. Key-framing verplicht de animator tot het tekenen van slechts enkele frames. Dit zijn de key frames. De andere frames van de animatie kunnen dan bekomen worden door interpolatie van de 2 omliggende key frames. Dit wordt in volgende afbeelding geïllustreerd. Hierdoor moet de animator niet meet alle frames zelf tekenen. De animator kan tevens elk bekomen frame nadien nog aanpassen, om ongewenste artifacten te vermijden.



Figuur 2.9: Vanbuiten zijn 2 key frames (wit) getekend door de animator. Ertussen zijn de nieuwe frames (blauw) bekomen door interpolatie van de key frames.

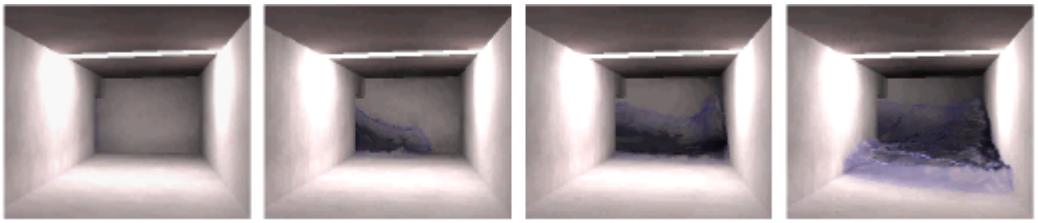
Hoewel key-framing het werk van de animator enigzins verlicht, moet hij toch nog altijd een reeks key-frames handmatig tekenen. Zeker voor complexere animaties moeten voldoende key frames aanwezig zijn, anders zullen de geïnterpoleerde frames niet aan de wensen van de animator voldoen, en moet hij opnieuw beginnen.

2.2.4 Kostfuncties

Een andere methode dan key-framing om bewegingen weer te geven, zijn kostfuncties [WK88] [Coh92]. Deze functies optimaliseren de interne parameters automatisch, zodat de globale kost geminimaliseerd wordt. Dit levert dan de gewenste bewegingen.

Fysische simulaties

Kostfunctie kunnen gecombineerd worden met fysische simulaties. Deze methoden creëren een animatie door een model te onderwerpen aan bepaalde fysische eigenschappen, zoals bijvoorbeeld zwaartekracht of veerkracht. Hierdoor worden de bewegingen fysisch correct. Een voorbeeld hiervan zijn fluid dynamics[5] [Dec04].



Figuur 2.10: Een animatie met fluid dynamics waar water door een gang stroomt. Nadat de animator de simulatie gestart heeft, kan hij deze niet meer beïnvloeden.

Hoewel deze simulaties er vaak erg realistisch uitzien en toch tamelijk snel zijn, heeft de animator heel weinig controle over de animatie. Wanneer de kostfuncties vastgelegd zijn, kan de animator de animaties haast nauwelijks corrigeren. Daarom moeten de parameters van de kostfuncties goed bepaald worden. Meestal is dit een proces van trail-and-error.

2.3 Vergelijking van render- en animatiemethoden

Het animeren van complexe modellen, zoals mensen en dieren, is in computer graphics erg moeilijk. Het handmatig bepalen van de geometrie is erg complex en tijdrovend, en biedt meestal geen visuele voldoening. In een oogopslag kan het computermodel van het echte onderscheiden worden.
Verder is het bepalen van key frames voor de animatie er tijdrovend. Fysische simulaties bieden de animator dan weer te weining controle.
Voor deze problemen kan video-based animation een oplossing zijn.

[5] Afbeelding uit masterthesis van Bert De Decker

Hoofdstuk 3

Video-based rendering

In het vorige hoofdstuk werden enkele veel gebruikte render- en animatietechnieken besproken, met elk met hun voor- en nadelen. Hierdoor zijn ze niet altijd bruikbaar, vooral wanneer realisme primeert. In dit hoofdstuk wordt video-based rendering meer in detail besproken. Ook komen animatietechnieken voor video-based rendering aan bod.

3.1 Motivatie

Video-based rendering is een subdomein binnen image-based rendering, met toevoeging van temporele informatie. I.p.v. stilstaande foto's, zoals bij image-based rendering, worden bij video-based rendering bewegende beelden gebruikt.

Zoals in vorig hoofdstuk werd aangetoond, hebben de verschillende render- en animatiemethoden verschillende nadelen.
Voor geometrie-based rendermethoden is het creëren van de gewenste modellen vaak erg complex. Daarom zal de animator om echt realistische modellen te maken andere technieken moeten gebruiken. Een mogelijke oplossing ligt binnen de image-based rendering. Nadelig hieraan is dat er geen temporele informatie aanwezig is. Deze kan toegevoegd worden, zodat video-based rendering ontstaat. Door echt filmmateriaal te gebruiken, zal het model, alsook de animaties, per definitie al realistisch zijn. In dit hoofdstuk worden enkele belangrijke concepten bij video-based rendering besproken.

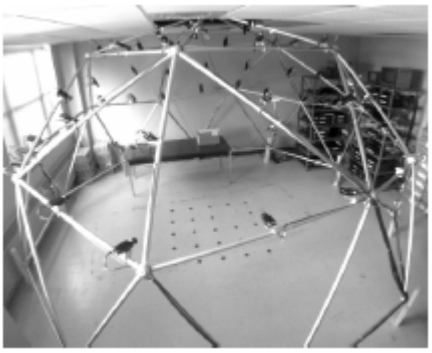
3.2 Rendering

In deze sectie wordt uitgelegd hoe video-based rendering werkt. De verschillende stappen komen aan bod, alsook toepassingen worden aangehaald.

3.2.1 Acquisition

Een eerste stap in video-based rendering is het opnemen van de video. Typisch wordt dit gedaan met behulp van speciaal ontworpen opstellingen die verschillende camera's bevatten. Vaak wordt ook gefild voor een blue screen, om dan gemakkelijk de objecten van de achtergrond te scheiden. Elke camera registreert dan zijn eigen videosequentie van de

scène. Meer technische uitleg over de verschillende mogelijke opstellingen is te vinden op verschillende plaatsen[1] [MPC+05].



Figuur 3.1: Voorbeeld van een video-opstelling. De opstelling bevat verschillende camera's die elk hun eigen videosequentie capturen van het gefilmde object.

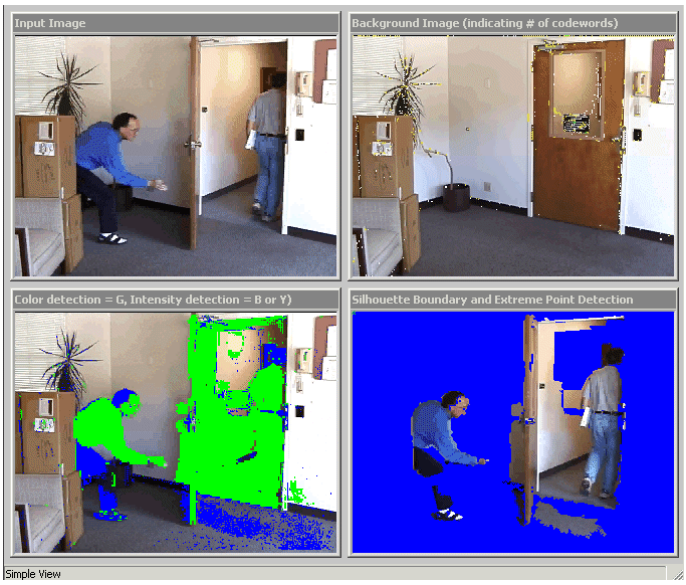
Een volgend belangrijk onderdeel van video-based rendering is het scheiden van voor- en achtergrond. Meestal is men immers niet geïnteresseert in de hele scène, maar enkel in enkele objecten binnen die scène. Daarom zijn algoritmen die voor- en achtergrond scheiden, nodig. Mogelijke algoritmen zijn background subtraction en graph cuts. Deze worden nu kort beschreven.

3.2.2 Background subtraction

Background subtraction is een techniek die veel gebruikt wordt in computer vision om voorgrond en achtergrond in een video van elkaar te scheiden. Background subtraction is een 2-pass algoritme: in de eerste stap wordt voor een frame zijn achtergrond berekend, om dan in de tweede stap het frame van zijn achtergrond af te trekken. Zo worden de objecten in de voorgrond bekend. Toepassingen van background subtraction zijn object tracking en object recognition.

Om voorgrond objecten te isoleren, moet voor een bepaald frame diens achtergrond berekend worden. Wanneer de achtergrond bekend is, worden voor een gegeven frame alle pixels afgegaan om te bepalen of deze bij de voorgrond of bij de achtergrond behoren. Een pixel in een bepaald frame i behoort tot de voorgrond wanneer de waarde van die pixel genoeg verschilt met de overeenkomstige pixel in het achtergrond model. Wanneer dit voor elke pixel in een frame gedaan is, is de voorgrond van dat frame bekend, namelijk die pixels die voldoende verschillend waren.
De moeilijkheid hierbij ligt hem echter in het bepalen van de achtergrondframe. De achtergrond is niets meer dan de lege scène. Vaak is deze niet gekend en moet dus berekend worden. Hiervoor bestaan vele manieren, waarvan enkelen hieronder kort beschreven worden.

[1] Afbeelding van CMU Virtualized Reality onderzoekscentrum.



Figuur 3.2: Werking van background subtraction: linksboven is een inputframe, rechtsboven de overeenkomstige achtergrond. Linksonder het verschil van beide te zien, en rechts-onder is de bekomen voorgrond.

Statische modellen

De eerste groep zijn de statische modellen. Bij deze modellen is de achtergrond voor de hele video gelijk. Een pixel in frame i behoort tot de voorgrond wanneer:

$$|f_i - b| \geq t,$$

waar f_i het i-de frame is, b de globale achtergrond en t de threshold die de gevoeligheid van de toegestane afwijking aangeeft. De threshold vermijdt dat kleine schommelingen in kleurwaarde een ongewenste invloed zouden hebben in de voorgrond. 2 kleurwaarden zijn zelden exact gelijk gedurende een videofragment om wille van veranderingen in lichtsterkte. Een voorbeeld van een statisch model is het gemiddelde van de hele video.

De nadelen van deze groep modellen zijn dat veranderingen in lichtsterkte gedurende de video niet in rekening worden gebracht. Ook worden objecten die gedurende een lange tijd aanwezig blijven, niet opgenomen in de achtergrond hoewel het een statisch object geworden is. Neem bijvoorbeeld een auto die een parkeerplaats komt opgereden. Op dit moment behoort de auto tot de voorgrond. Maar wanneer de auto nu voor een lange tijd blijft staan, moet hij echter in de achtergrond worden opgenomen. Dit probleem geeft aanleiding tot een andere groep van algoritmen.

Dynamische modellen

Dynamische modellen bieden een oplossing voor de problemen van de statische modellen. Voor elk frame wordt nu een eigen achtergrond berekend:

$$|f_i - b_i| \geq t.$$

Merk op dat enkel de index i voor de background in deze formule is toegevoegd. Dit geeft aan dat iedere frame zijn eigen achtergrond heeft.

Een eerste mogelijke achtergrond is de vorige frame. Maar deze aanpak kent een groot nadeel. Wanneer objecten traag bewegen, treden er overlappingen op tussen beide frames. Hierdoor zullen de overlappende delen van bewegende objecten onterecht in het achtergrondmodel opgenomen worden.

Beter is dus meerdere frames te gebruiken bij het bepalen van de achtergrond. Hiervoor worden begrippen uit de statistiek gebruikt. Goede maten voor de achtergrond zijn het gemiddelde [LV00], de mediaan [CGPP03] of de modus van de voorgaande frames. Belangrijk bij deze algoritmen is dat er bij het updaten van het achtergrondmodel na de verwerking van elk frame telkens de recente geschiedenis gebruikt wordt bij het berekenen van de achtergrond voor het volgende frame.

Er bestaan ook meer geavanceerdere technieken die gebruik maken van geavanceerde methoden uit de statistiek, zoals frequentieverdelingen, kansdichtheidfuncties en eigenbackgrounds. Frequentieverdelingen [SG00] [WADP97] houden in een histogram bij hoe vaak een bepaalde kleurwaarde aanwezig was in een frame. Deze histogrammen worden vervolgens gebruikt bij het bepalen van de achtergrondpixels. Bovenop de frequentieverdelingen kan dan een kansverdeling opgesteld worden, die de kans aangeeft dat die pixels achtergrondpixels zijn.
Eigenbackgrounds [ORP00] is een heel verschillende techniek, die gebruik maakt van Principal Component Analysis (PCA). PCA wordt gebruikt om, door eigenvectoren decompositie, de dimensies van de ruimte te verminderen. Indien PCA wordt gebruikt over een sequentie van n frames, geeft dit de eigenbackgrounds. Voor meer uitleg wordt verwezen naar de literatuur [Bun05] [Pic04].

Verwijderen van ruis

Vaak is het bekomen masker van background subtraction bevuild met ruis. Dit is het gevolg van kleine schommelingen in lichtsterkte in de videosequentie. Om mooie silhouetten te bekomen, moet deze ruis verwijderd worden. Hiervoor worden morphologische operaties gebruikt [GW01].

Dilatie:
Dilatie is een operatie die een verzameling A laat 'groeien' door een masker B erover te bewegen. Elk element van B die verzameling A niet leeg maakt, is een element van de dilatie:

$$A \oplus B = \left\{ z \mid (\hat{B})_z \cap A \neq 0 \right\}.$$

Intuïtief wil dit zeggen dat elk element van B bij verzameling A wordt toegevoegd wanneer, bij translatie van B over A, de doorsnede van A met B niet leeg wordt.
Wanneer B een rechthoek van 3 op 3 is, zal de verzameling A groter worden met 1 pixel aan de randen. Wanneer dit toegepast wordt op een masker van background subtraction, zal dit masker dus groter worden.

Erosie:

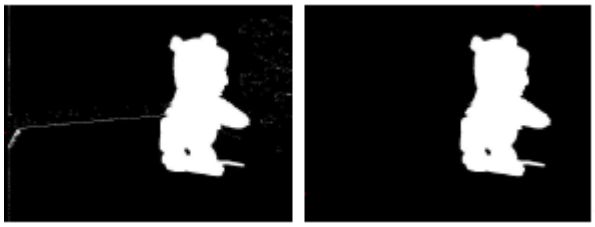
Erosie is de operatie die de verzameling A laat 'krimper' door een masker B erover te bewegen. Elk element van B dat ook een element is van verzameling A, is een element van de erosie:

$$A \ominus B = \{z \mid (B)_z \subset A\}.$$

Wanneer B een rechthoek van 3 op 3 is, zal de verzameling A kleiner worden met 1 pixel aan de randen. Wanneer dit toegepast wordt op een masker van background subtraction, zal dit masker dus kleiner worden.
Erosie is het tegenovergestelde van dilatie. Toch zijn ze niet elkaar inverse: erosie na dilatie of dilatie na erosie zal niet terug het oorspronkelijke resultaat geven, zoals hieronder wordt geïllustreerd.

Opening en closing:

Dilatie en erosie kunnen vervolgens gecombineerd worden om het masker van background subtraction ruisvrij te maken. Wanneer dilatie na erosie wordt toegepast, zal dit geïsoleerde punten verwijderen. Ook zullen scherpe randen afgevlakt worden. Deze operatie wordt de opening genoemd: $A \circ B$. Het oorspronkelijke masker is nu properder.
Erosie kan ook na dilatie worden toegepast. Hierdoor zullen randen en geïsoleerde punten feller geaccentueerd worden. Deze operatie wordt de closing genoemd: $A \bullet B$. Hierdoor zal het masker net minder proper worden.

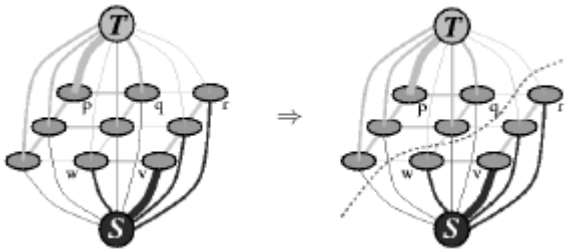


Figuur 3.3: Links is het masker bekomen na background subtraction. Rechts is hetzelfde masker na toepassing van ersoie en dilatie. Een grote hoeveelheid van de ruis is verwijderd. Toch is het silhouet nog niet ideaal, en blijft een kleine hoeveelheid ruis aanwezig.

3.2.3 Graph cut

Graph cut is een andere manier om objecten uit een frame te halen. In tegenstelling met background subtraction gebeurt dit interactief; de gebruiker geeft eerst een aantal voor- en achtergrondpixels aan en vervolgens wordt m.b.v. de max-flow min-cut theorie [EFS56] die delen uit de frame te halen waar deze energy flow maximaal is. Dit algoritme werd voor het eerst voorgesteld door Yuri Boykov en Vladimir Kolmogorov [BJ01].

Zoals de naam graph cut al laat vermoeden, wordt er een graaf $G(V, E)$ gebruikt die alle pixels voorstelt. Aan elke pixel in een frame wordt een node toegekend. Net zoals de pixels, worden de nodes verbonden met naburige nodes. Er is ook 1-source en 1 sink-punt. Het source-punt stelt de voorgrond voor, en het sink punt de achtergrond.



Figuur 3.4: Werking van graph cut: links is de graaf enkel met het source-punt S en sink-punt T, rechts met de cut. De stippellijn geeft de scheiding van voor- en achtergrond aan.

Deze techniek is echter niet beperkt tot 1 afbeelding per graaf. Meerdere afbeeldingen kunnen samen in 1 graaf opgenomen worden. Dit geeft dan een 3-dimensionale graaf, met het source-punt aan de ene kant en het sink-punt aan de andere.
Vervolgens moeten een aantal voor- en achtergrondpixels worden aangeduid. Dit gebeurt meestal met behulp van een brush. Elke voorgrond pixel wordt verbonden met het source-punt, en elke achtergrond pixel met het sink-punt. In beide gevallen krijgen deze links een maximale waarde. Het bepalen van de waarden van de andere links, die noch bij de voorgrond noch bij de achtergrond horen, is een moeilijk proces en kan afhangen van de gekozen afbeelding. Voor formules om deze waarden te bepalen wordt verwezen naar de literatuur [BJ01] [RKB04].

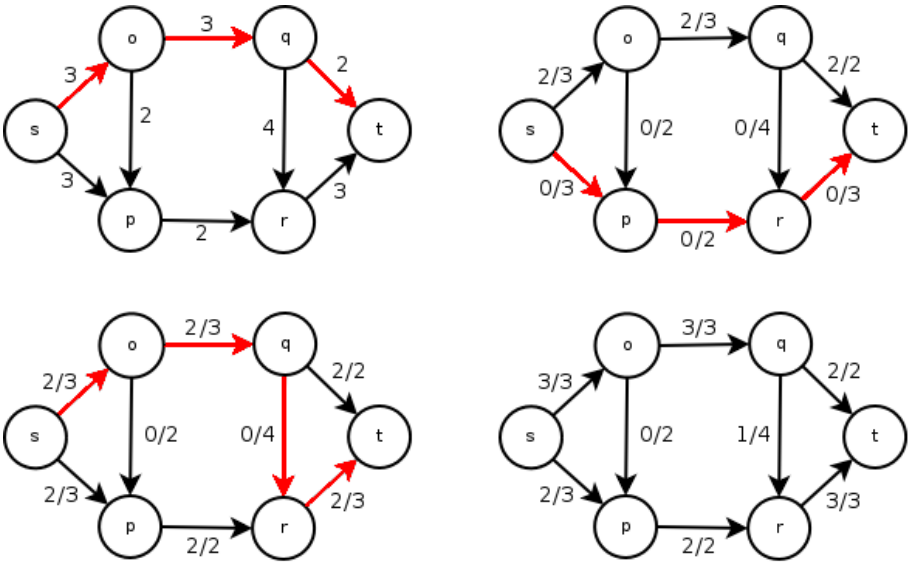
Na het opstellen van de graaf, kan de flow berekend worden. Vertrekkende van een source-punt, wordt er een pad gezocht via de links naar 1 van de sink punten. De laagste waarde van de links op dat pad wordt bijgehouden, en elke link op dit pad wordt verminderd met deze waarde. De bekomen waarde geeft de huidige flow aan. Dit wordt herhaald tot er geen pad meer bestaat van een source naar een sink-punt zonder een link met label nul.

Na het berekenen van de flow, kan de graph cut bepaald worden. Vertrekkende van de source elementen wordt elk element toegevoegd aan het resultaat waar een link met een positief label naartoe bestaat. Dit wordt iteratief herhaald tot het resultaat niet meer verandert. Dit is dan de voorgrond. De capaciteit van de cut is gedefiniëerd door

$$c\left(S,T\right)=\sum_{u\in S,v\in T}c\left(u,v\right),$$

met S de source en T de sink. De capaciteit is dus de som van de waarden van de links die de cut definiëert.

Een uitgewerkt voorbeeld:
Vertrekkend van het source-punt wordt een eerste pad gevonden naar een sink-punt. De laagste waarde op dit pad is 2. Dit is de huidige flow en wordt links bij elk label op dit pad gezet. Het linker getal van een label mag nooit groter worden dan het rechter, anders worden negatieve labels bekomen. Dan wordt een tweede pad gevonden, ook weer met waarde 2 als laagste. Een derde pad wordt nog gevonden, met waarde 1. Hierna zijn geen paden meer mogelijk van het source-punt naar het sink-punt zonder negatieve labels. Nu kan de cut genomen worden.



Figuur 3.5: Voorbeeld van graph cut.

Er zijn 3 minimale-cuts voor dit voorbeeld: $S = \{s, p\}, T = \{o, q, r, t\}$ met capaciteit $c(s, o) + c(p, r)$ = 3 + 2 = 5, $S = \{s, o, p\}, T = \{q, r, t\}$ met capaciteit $c(o, q) + c(p, r)$ = 3 + 2 = 5 en $S = \{s, o, p, q, r\}, T = \{t\}$ met capaciteit $c(q, t) + c(r, t)$ = 2 + 3 = 5.

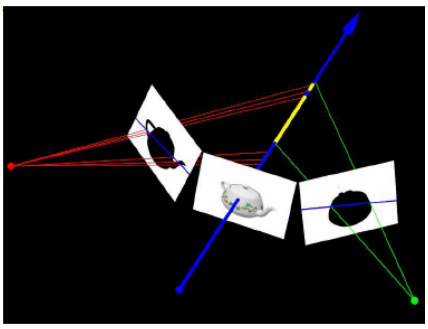
Ter vermelding: de max-flow min-cut theorie wordt ook toegepast om de capaciteit van netwerken te berekenen.

3.2.4 Visual hull

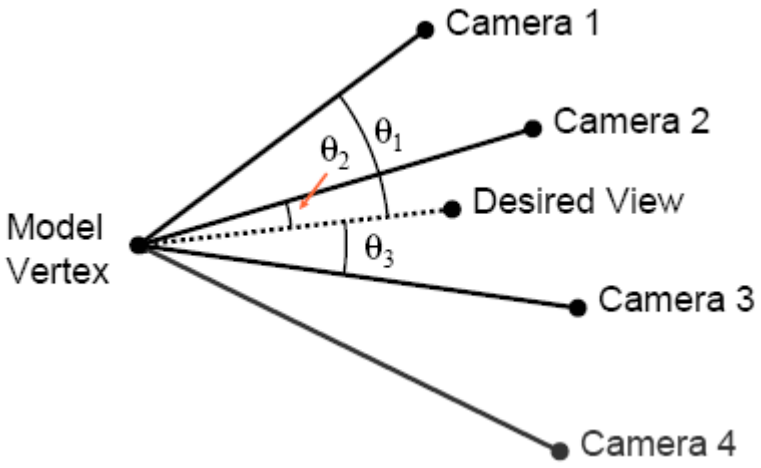
Het bepalen in welke regio's een gefilmd object zich bevindt, is al door vele onderzoekers bestudeerd. De visual hull [Lau94] van een object is een vorm van het object dat deze indeling aangeeft. Een veel gebruikte techniek om deze indeling te berekenen, is volume carving [Pot87]. Deze techniek verwijdert lege ruimtes zodat het silhouet van het object overblijft. De visual hull is dan het maximale volume dat geconstrueerd wordt vanuit alle mogelijke silhouetten. Meestal wordt echter een beperking opgelegd op het aantal silhouetten. Een silhouet, gezien vanuit een pinhole-camera, bevat een 3-dimensionaal volume, vertrekkende vanuit de oorsprong van de projectie tot oneindig, terwijl het object gepasseerd wordt. Dit werd in onderstaande tekening in het rood aangeduid. Zo kan men te weten komen waar een object zich bevindt in de ruimte.

3.2.5 View-dependent rendering

Met eerder berekend visual hull kan de silhouet ook getextured worden [TLMS03]. Hierdoor zal het resultaat veel realistischer zijn.
Voor elke camera wordt diens geprojecteerd silhouet getextured. Voor elke nieuwe view worden deze textures geblend, zodat de nieuwe silhouet ook ingekleed wordt. Het bepalen van de gewichten voor elke camera tijdens het texture-mappen wordt gedaan door de afstand tussen de hoeken van de nieuwe view en de camera's te berekenen. Dit is geïllustreerd in onderstaande afbeelding.



Figuur 3.6: Visual hull van een object in de ruimte. Voor de gewenste ray (blauwe) worden de projecties op elke reference view (rood) berekend. Het resultaat kan dan gemapt worden naar 3D. Wanneer dit voor elke ray gedaan wordt, is de visual hull van het silhouet bekomen.



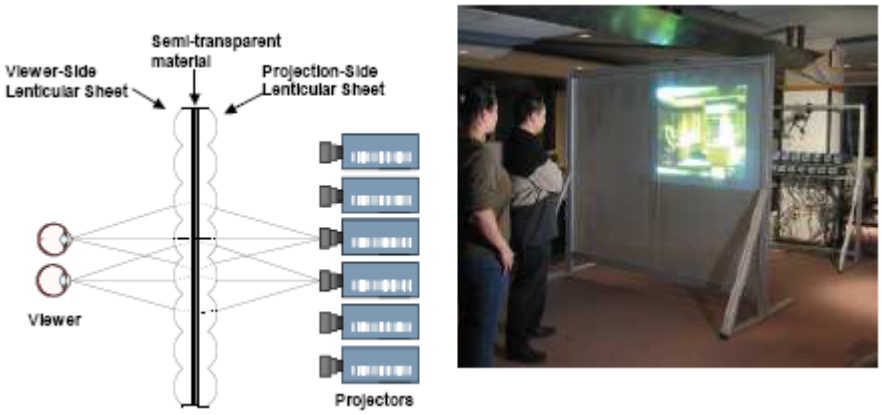
Figuur 3.7: Gebruik makend van verschillende views kunnen dan nieuwe views geconstrueerd worden. Enkel de 3 dichtstbijzijnde camera's worden gebruikt. De gewichten van de camera's worden bepaald door de hoek t.o.v. de nieuwe view.

3.2.6 3D-TV

Het 3 dimensionaal weergeven van scènes op een gewoon scherm is erg moeilijk. Het doel van 3D-TV [VMPX04] is door gebruik van verschillende camera's de scène te filmen, en deze dan op een scherm zo te projecteren, dat, wanneer vanuit een ander standpunt naar het scherm gekeken wordt, ook een ander deel van de scène zichtbaar is. Hierbij zijn 3 onderdelen van belang:

Data capturing

Eerst en vooral moeten de objecten door een 3D-systeem gefilmd worden. Hiervoor zijn verschillende camera's nodig, die vaak op 1 lijn staan. Elke camera creëert dan zijn eigen videosequentie.



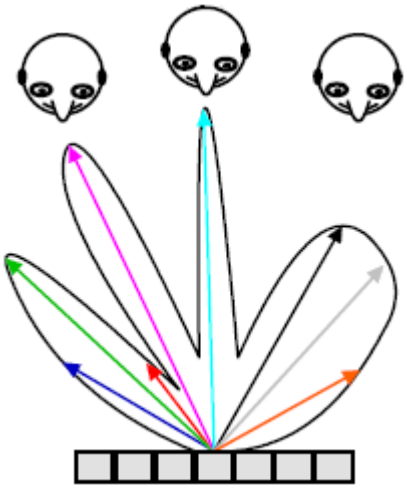
Figuur 3.8: Voorbeeld van een 3D TV.

Transmission

De volgende stap is de transmissie van elke videosequentie naar de display. Dit moet real-time gebeuren, anders zal de gebruiker niet ervaren deel te zijn van de scène.

Display

Ten slotte moet elke videosequentie verwerkt worden om deze op een scherm te kunnen projecteren. De video sequenties kunnen gecombineerd worden tot 1 sequentie, rekening houdend met de positie van de gebruiker. Wanneer deze zijn positie verandert, moet de hele sequentie weer aangepast worden.
Als alternatief zijn er de 3D-display's. Deze bevatten view-dependent pixels. Deze pixels zenden verschillende hoeveelheden licht uit in verschillende richtingen, en zijn hierdoor in staat voor verschillende gebruikers tegelijk beeld te projecteren.



Figuur 3.9: Werking van een 3D-display, de pixels zenden verschillende hoeveelheden licht uit in de verschillende richtingen. Wanneer een gebruiker van positie verandert, zal hij andere beelden zien op het scherm.

3.2.7 Compressie

Een videobestand is opgebouwd uit frames. Een frame is enkel een 2-dimensionale tabel van pixels. Elke pixel stelt een kleur voor. Meestal worden voor 1 pixel 4 waarden bijgehouden: 1 voor elk van de 3 hoofdkleuren (rood, groen en blauw) en een aplha-waarde. Deze alpha wordt gebruikt bij het blenden van 2 verschillende kleuren.
Vaak worden compressietechnieken gebruikt om de omvang van de videofile te verkleinen. Dit is nodig omdat het opslaan van 3 kleurwaarden per pixel per frame veel geheugen vraagt. Voor een resolutie van 800 op 600 en met 25 frames per seconde levert dit al 3*800*600*25=3600000 bytes, of zo een 34.5 MB per seconde. Voor een film van 2 uur zou dit 235 GB betekenen, hetgeen het belang van compressie onderlijnt.

3.3 Animatie

Net zoals bij de andere render-methode kunnen ook met video-based rendering animaties gemaakt worden. Video-based animation heeft een heel interessant voordeel. Omdat video sequenties gebruikt worden, is het realisme immers gegarandeerd.
Maar video-based animation heeft ook nadelen. Zo is er geen directe informatie over de diepte, en moet deze eerst berekend worden. Verder moet er voldoende motion capturing data zijn, anders kan geen accuraat model van de input scène berekend worden.

3.3.1 3D-modellen

Gebruik makend van verschillende views, kunnen reeds eenvoudige animaties gemaakt worden. Deze methoden creëren dan een 3-dimensionaal beeld van het gefilmde object. Hierdoor zijn zowel animaties van het gezicht, als het hele lichaam mogelijk.
Het gecaptured model kan gebruikt worden om een skeletstructuur van dit model op te stellen [CBK05]. Met dit skelet kunnen dan nieuwe poses vastgelegd worden m.b.v. kinematics. Zo worden nieuwe animaties mogelijk [Gle98]. Ook kan een bepaalde animatie van 1 persoon door een andere persoon uitgevoerd worden door de poses van het skelet van persoon 1 te kopiëren op dat van persoon 2 [CBHK04]. Andere toepassingen zijn natuurlijk ook mogelijk.

Om nieuwe animaties te maken wordt motion capturing data vaak gecombineerd met fysische [PW99] en/of kinematische modellen. Dit is nodig om voldoende controle te behouden over het model. Fysische modellen steunen op verdeling van krachten, kinematische modellen op energieverdeling. Beide modellen kunnen ook gecombineerd worden [SHP04], om een nog grotere controle te krijgen.

Deze algoritmen leveren heel realistische animaties. Maar het grootste nadeel is dat deze algoritmen veel inspanningen van de animator vragen. Deze moet zowel de fysische als de kinematische modellen zelf specifiëren. Bovendien is de rendertijd van deze algoritmen erg hoog, zodat dimensievermindering op de bewegingen nodig is [SHP04].

3.3.2 2D-modellen

Het opstellen van een goed 3D-model voor motion capturing data is erg moeilijk, zeker voor complexe geometrie. Hierdoor is het soms aangewezen het opstellen van dit model over te slaan. Zo kunnen image-based render-technieken gebruikt worden om nieuwe views te creëren [Che95]. Door een camera over een bepaald pad te laten bewegen en voor elke tijdstap image-based rendering te gebruiken om de nieuwe views te maken, kan een animatie bekomen worden die een camera in een scène laat bewegen. Het nadeel van deze methode is dat er veel tijd nodig is om de animatie te maken. Door per tijdseenheid dure image-based render-technieken te gebruiken, zullen deze algoritmen erg traag zijn.

Motion without movement [FAH91] laat objecten de indruk wekken te bewegen. Deze bewegingen veranderen de positie niet. De methode maakt gebruikt van 2 indentieke filters die enkel een 90-graden fase verschil hebben. Een beweging kan gesimuleerd worden door een serie van fase-shift filters te berekenen met de 2 opgegeven filters. Dit wordt gedaan door een lineaire combinatie van de 2 filters te gebruiken:

$$F(t) = \cos(\omega t)G + \sin(\omega t)H,$$

met G en H de input filters en F de nieuwe fase-shift filter.
Hoewel met deze methode visuele illusies van objectbewegingen gecreëerd worden, zijn echte bewegingen niet mogelijk. Daarom is deze methode te beperkt.

Een heel andere video-based animation methode zijn video textures [SSSE00]. Door de volgorde van de video frames aan te passen, zijn nieuwe animaties mogelijk. Natuurlijk kan de volgorde niet zomaar veranderd worden. Er moet immers nagegaan worden of 2 videoframes wel na elkaar afgespeeld kunnen worden zonder zichtbare artifacten. In volgend hoofdstuk staan verschillende mogelijke metrieken vermeld die helpen bij het bepalen van goede transities tussen video textures. Hoofdstuk 5 vermeldt enkele belangrijke algoritmen om animaties met deze video textures te maken zonder hinderende visuele artifacten.

Een stap verder dan video textures zijn video sprites [SE02]. In tegenstelling tot video textures worden in video sprites enkel die objecten opgeslagen die van belang zijn, en dus geen achtergrondinformatie. Video sprites hebben daarom extra eigenschappen t.o.v. video textures, en hierdoor zijn meer animaties mogelijkheden. In hoofdstukken 6 en 7 wordt hier verder op ingegaan.

3.4 Slot

Wanneer realistische animaties nodig zijn, is het aangeraden data van videosequenties te gebruiken. Door objecten in hun natuurlijke omgeving te filmen, is realisme immers gegarandeerd.
In dit hoofdstuk werden eerst enkele belangrijke aspecten tijdens de renderfase van de videosequenties besproken, zoals acquisition, background subtraction en view-dependent renderen.
Daarnaast werden ook enkele video-based animation technieken besproken, en werden video textures en video sprites ingeleid.

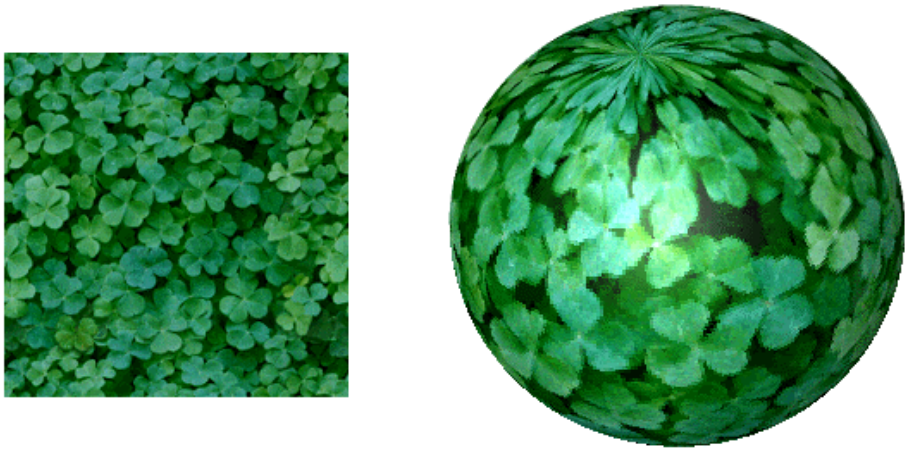
Hoofdstuk 4

Video textures

In vorig hoofdstukken werd video-based rendering gesitueerd binnen de graphics, en werden enkele belangrijke aspecten ervan besproken. Er werden tevens enkele animatiemogelijkheden van video-based rendering aangehaald. Hierbinnen werden video textures voorgesteld als mogelijke animatie vorm.
Dit hoofdstuk zal de basisbegrippen van video textures behandelen, alsook metrieken nodig voor animaties met deze video textures.

4.1 Textures

Textures vormen een belangrijk onderdeel in 3D*rendering [SWND02]. Ze laten immers toe een bepaald object te bekleden volgens een bepaald patroon [Cat74], waardoor het object er levendiger zal uitzien. Een texture is niet meer dan een eenvoudige rechthoekige tabel met data voor bijvoorbeeld kleurwaarden, lichtwaarden, schaduwen,... Tijdens het renderen worden dan de waarden van de texture gemapt op het object. Dit proces wordt texture mapping genoemd. Hoewel de texture rechthoekig is, hoeft het object dat helemaal niet te zijn. Dit is vaak één van de moeilijkheden van texture mapping.



Figuur 4.1: Principe van texture mapping: de bol wordt bekleed met een rechthoekige texture.

Texture synthese

Texture synthese is het algoritmisch creëren van grote digitale afbeeldingen uit kleinere digitale afbeeldingen. Hierbij is het behouden van de globale structuur van de texture van belang. Deze structuur kan gaan van stochastic textures ("noisy textures") tot structured textures (textures met een uitgesproken structuur, zoals een dambord patroon). De synthesemethode hangt hier rechtstreeks van af. Deze methoden kunnen onderverdeeld worden in enkele categoriën [wik]:

Tiling: Dit is de meeste eenvoudige methode. Deze kopieert de input texture enkele malen langs elkaar. Meestal levert dit zwakke resultaten.

Stochastic texture synthesis: Bij deze methode wordt de pixelwaarde bepaald door een random keuze, enkel beïnvloed door simpele parameters zoals helderheid, gemiddelde kleur, maximaal contrast,... Hierdoor wordt enkel voor stochastic textures goede resultaten bekomen.

Single purpose structured texture synthesis: Deze categorie van algoritmen behoudt de globale structuur van de texture. Maar deze algoritmen zijn vaak specifiek voor een welbepaalde soort structuur, en kunnen hierdoor enkel toegepast worden op structured textures met een gelijkaardige structuur als de input texture.

Chaos mosaic: Dit is een verbeterde versie van het tiling algoritme, en bestaat uit 3 stappen. Eerst wordt gewone tiling toegepast, waardoor zichtbare randen ontstaan. Vervolgens worden random gebieden van de output texture vervangen. Dit geeft een niet-repetitieve texture met nog steeds zichtbare randen. Ten slotte wordt dit resultaat geblurd. Hoewel dit in sommige gevallen goede resultaten geeft, is het algoritme niet ideaal. Het ziet er immers geblurd uit door de filter van stap 3.

Pixel-based texture synthesis: Zoals de naam al laat vermoeden worden de pixels van de output texture bepaald door de naburige pixels van de input texture waar de pixel gelijkaardig aan is [HB95] [EL99] [WL00] [HJO+01]. Deze algoritmen zijn erg eenvoudig en succesvol, en zeer geschikt voor image completion.

Patch-based texture synthesis: Deze algoritmen kopiëren niet de pixels individueel, maar kleine groepen pixels (patches) [Bon97] [EF01] [KSEB03]. Hierdoor zijn ze sneller en effectiever dan pixel-based texture synthesis-methoden.

4.1.1 3D textures

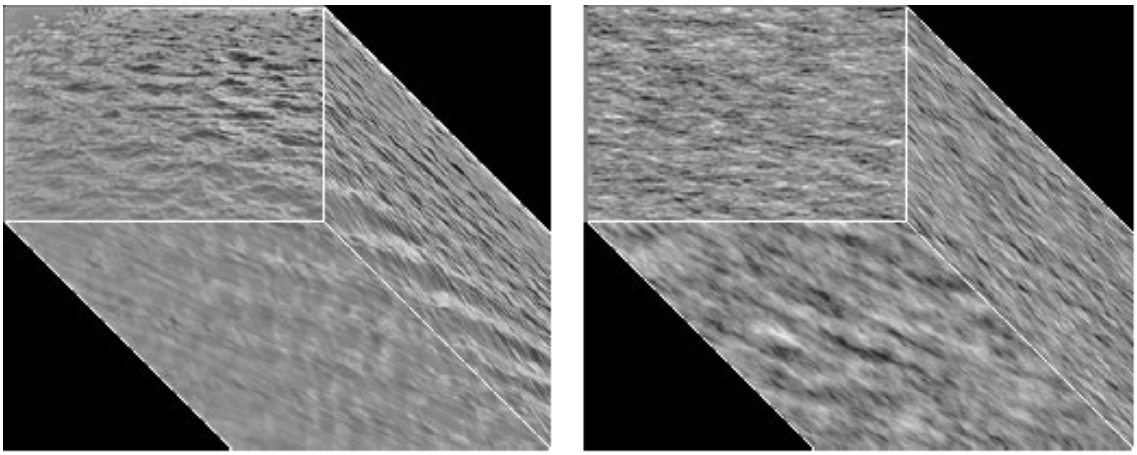
3D textures zijn een uitbreiding op de 2D textures, en worden vaak gebruikt in de medische en geografische wereld. Meestal wordt de derde dimensie gebruikt om diepte voor te stellen, maar deze dimensie kan ook gebruikt worden om de tijd voor te stellen. Hierdoor worden de 3 dimensionale textures interessant voor videoverwerkingen. Zo kunnen bijvoorbeeld voor een videofragment alle frames in een 3D texture gestopt worden. Dan kan deze texture gemapt worden op een object, zodat het lijkt of het videofragment op dit object afgespeeld wordt.

Temporal textures

Temporal textures [SP96] zijn textures met een beweging. Voorbeelden zijn golvend water, rook en vuur. Deze bewegingen zijn niet beperkt in ruimte en tijd, en is dus een subdomein binnen de 3D textures. De serie afbeeldingen van een temporal texture worden gemodelleerd m.b.v. spatio-temporal autoregressive model (STAR) [PD80]. Dit is een 3-dimensionele uitbreiding van de autoregressive model (AR). Dit model stelt de pixels voor als lineaire combinaties van de naburige pixels zowel in ruimte als tijd, en heeft de vorm

$$s(x, y, t) = \sum_{i=1}^{p} \phi_i s(x + \Delta x_i, y + \Delta y_i, t + \Delta z_i) + a(x, y, t).$$

Het signaal $s(x, y, t)$ wordt gemodelleerd als een lineaire combinatie van een omliggende waarden van zichzelf, plus Gaussian white noise $a(x, y, t)$.
Dit model kan zowel gebruikt worden om herkenning van de temporal textures als de synthese ervan. De least squares methode kan dan gebruikt worden om nauwkeurig de parameters van grote lineaire combinaties af te schatten. Hierdoor worden zowel de ruimtelijke als temporele karakteristieken goed waargenomen.



Figuur 4.2: Werking van temporal textures. Links is een serie input-frames, en rechts is de serie frames bekomen door het spatio-temporal autoregressive model.

Dynamic textures

Een dynamic texture [SDW01] is eveneens een speciaal geval van een 3D textures. Dynamische textures zijn series van frames van een bewegende scène die stationaire fenomenen weergeeft. Zulke fenomenen zijn o.a. rook, zeegolven,..., maar ook een sprekend gezicht of een verkeerssituatie. Deze scènes worden dan geïnterpreteerd door een geschikt stochastisch model te zoeken. Dit model moet in staat zijn om toekomstige gebeurtenissen goed te voorspellen.
Temporal textures gebruiken de naburige pixels om zowel spatiale als temporele informaties te leren. Hierdoor kunnen geen draaiende bewegingen of versnellingen worden voorgesteld. Omdat dynamic textures eerst voor de input texture een geschikt stochastisch model zoekt, kunnen deze fenomenen wel worden voorgesteld.

4.1.2 Video textures

Een video texture [SSSE00] is eveneens een speciaal geval van een 3D textures. Een video textures heeft buiten de spatiale informatie ook weer temporele informatie. De texture heeft per frame ook een absoluut tijdstip. Een video texture heeft een continue, oneindige sequentie van videoframes. Hoewel delen van een video textures herhaald kunnen worden, bevat het geheel nooit herhalingen. De video textures worden gesynthetiseerd door de volgorde van de frames van de originele texture te veranderen.

4.2 Transities voor video textures

Een video texture is een oneindige sequentie van textures. Het is dus een reeks afbeeldingen van een bepaald object. Om animaties voor video textures te maken op basis van bestaande videosequenties, moet nagegaan kunnen worden welke frames van de video textures op elkaar kunnen volgen. Zo een overgang wordt een transitie genoemd. Deze transities helpen vervolgens om animaties met de video textures te maken (volgend hoofdstuk).

4.2.1 Kleureigenschappen

Een eerste maatstaf bij het bepalen van een mogelijke transities tussen 2 video frames, is hun kleur. Kleur is erg belangrijk: het menselijk oog kan al snel kleine kleurschommelingen waarnemen, en te grote schommelingen zullen als storende artifacten ervaren worden. Een keuze voor een goede methode om kleurverschillen te berekenen is noodzakelijk.

Gemiddelde en standaardafwijking

Om een algemeen beeld te vormen van de kleur in een afbeelding, kan het gemiddelde genomen worden. Maar het gemiddelde geeft geen informatie over de kleurverdeling. Het is goed mogelijk dat 2 afbeeldingen met hetzelfde gemiddelde toch helemaal niet gelijkaardig zijn. Een ander begrip uit de statistiek vaak gerelateerd met het gemiddelde, is de standaardafwijking. De standaardafwijking geeft de spreiding van de kleurwaarden weer. Door ook de standaardafwijking te gebruiken wordt deze situatie iets verbeterd, maar dit geeft geen oplossing. Om echt te kunnen bepalen of 2 frames van een video texture op elkaar kunnen volgen zijn pixel-operaties nodig.

Pixel-operaties

Zoals net is aangetoond zijn pixel per pixel vergelijkingen noodzakelijk bij het bepalen van mogelijke transities. Wanneer de overeenkomstige pixels van 2 videoframes immers weinig van elkaar verschillen, zal een vloeiende overgang mogelijk zijn. Er zijn verschillende metrieken voorhanden.

L_1 **norm:** Een voor de hand liggende methode is om de som te nemen van het verschil in kleurwaarde:

$$L_1(p1, p2) = |p1(r) - p2(r)| + |p1(g) - p2(g)| + |p1(b) - p2(b)|$$

De L_1 norm neemt het gemiddelde van de puntsgewijze verschillen in kleurwaarden. Wanneer 2 van de 3 waarden nagenoeg gelijk zijn en slechts 1 fel verschillende, zal de L_1 norm toch redelijk weinig zijn. Grote verschillen wegen dus niet zwaar genoeg.

L_2 norm: De euclidische afstand tussen 2 punten in de ruimte is als volgt gedefiniëerd:

$$L_2(p1, p2) = \sqrt{(p1(x) - p2(x))^2 + (p1(y) - p2(y))^2 + (p1(z) - p2(z))^2}$$

Wanneer nu de punten $p1$ en $p2$ worden vervangen door 2 pixels en wanneer de x, y en z van de beide punten $p1$ en $p2$ vervangen wordt door de 3 kleurcomponenten rood, groen en blauw, wordt de euclidische afstand tussen de 2 overeenkomstige pixels berekend. Dit is de L_2 norm.
In tegenstelling met de L_1 norm zullen grote verschillen van 1 (of meer) componenten zwaarder meetellen, zodat betere resultaten bekomen worden. De L_2 norm is simpel en geschikt voor grote hoeveelheden data van incrementele aard, waaronder video textures en video sprites, zoals later zal blijken. Hierdoor is deze metriek ook gebruikt voor animaties van video textures [SSSE00] en video sprites [SE01] en [SE02].

L_p norm en L_∞ norm: Zowel de L_1 norm als de L_2 norm zijn speciale gevallen van de p-norm afstand:

$$L_p(p1, p2) = \left(\sum_n^{i=1} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Wanneer p naar oneindig gaat, geeft dit

$$L_\infty(p1, p2) = lim_{p \to \infty} \left(\sum_n^{i=1} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Dit is de L_∞ norm, het maximum van de puntsgewijze verschillen. Andere benamingen voor deze norm zijn de Chebyshev-afstand en de schaakbord-afstand [wik].

Cross-correlation: Correlatie geeft een lineair verband weer tussen 2 sets variabelen, en kan dus ook gebruikt worden voor 2 afbeeldingen. Hierbij worden de waarden uit de 2 sets vergeleken met hun gemiddelde, om zo te kunnen nagaan of er een lineair verband is.
De cross-correlation is gedefiniëerd door

$$C_{i,j} = \frac{\sum_m \sum_n (I_{i_{mn}} - I_i^-)(I_{j_{mn}} - I_j^-)}{\sqrt{(\sum (I_{i_{mn}} - I_i^-)^2)(\sum (I_{j_{mn}} - I_j^-)^2)}}$$

met $\overline{I_i}$ en $\overline{I_j}$ als gemiddelde voor I_i en I_j. Deze functie geeft waarden tussen -1 en 1. Door deze te mappen met (1.0 - $c_{i,j}$/2.0) worden waarden tussen 0 en 1 bekomen, met 0 voor goede correlatie.

Haussdorf afstand: Deze metriek gebruikt een edge-map en een distance-map voor elke afbeelding. De edge-map E wordt berekend met standaard edge detection [Can86]. De distance-map X is de getransformeerde afstand berekend met de edge-map E, en stelt de afstand tot de dichtstbijzijnde edge voor.

$$H_{i,j} = \frac{\sum_{(x,y)\in E_{i\equiv 1}} X_j(x,y)}{\sum_{(x,y)\in E_{i\equiv 1}} E_i(x,y)}$$

Figuur 4.3: De edge-map (midden) en de distance-map (rechts) van een cartoon texture (links).

Een andere manier om de Haussdorf afstand te interpreteren, is als het maximum van de minimale afstanden tussen de punten in de 2 verzamelingen:

$$H(A,B) = max(h(A,B), h(B,A))$$

met

$$h(A,B) = max_{a\in A} min_{b\in B} \|a-b\| .$$

$h(A,B)$ neemt voor ieder punt in de eerste frame de kleinste afstand tot een punt in de tweede frame, en neemt hiervan het maximum. Merk op dat $h(A,B) \neq h(B,A)$. Deze metriek wordt met succes gebruikt in cartoon textures [dJB04].

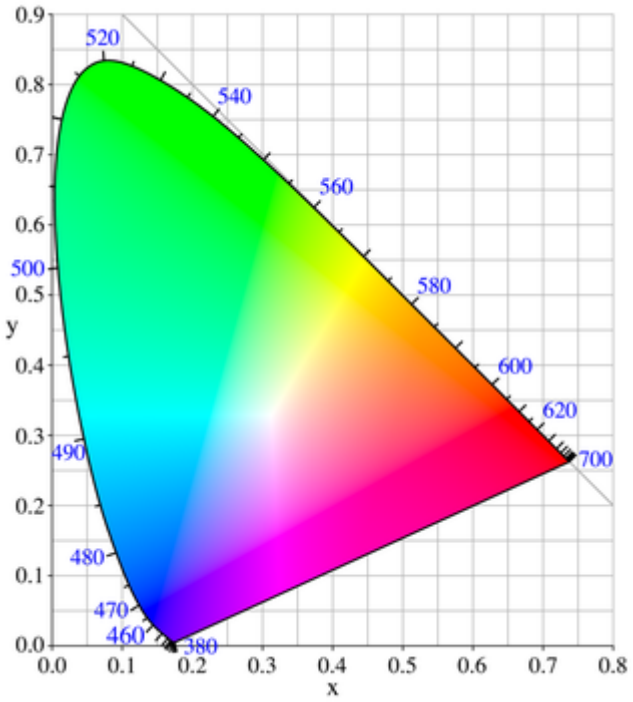
Alpha-waarde: Nog een metriek die per pixel werkt, is het verschil in alpha-waarde van 2 afbeeldingen. Hoewel dit een handige metriek kan zijn in combinatie met eerder vermelden metrieken, is deze op zichzelf weinig zinvol.

Andere kleurruimten

Hoewel de RGB kleurenruimte veel gebruikt wordt in computer graphics, is het echter niet intuïtief. Het is immers verschillend van de manier waarop het menselijk oog kleur waarneemt. Hiervoor werd de CIE-XYZ kleurenruimte ingevoerd.
Het menselijk oog heeft 3 types receptoren voor korte, middellange en lange golflengte. De CIE-XYZ kleurenruimte is net ontworpen om deze 3 receptoren best te evenaren, namelijk X, Y en Z. Kleur kan immers onderverdeeld worden in 2 delen, namelijk de helderheid en de chromaciteit. Zo hebben wit en grijs dezelfde chromaciteit, enkel de helderheid is verschillend. In de CIE-XYZ kleurenruimte stelt Y de helderheid voor, en 2 afgeleide parameters x en y de chromaciteit (zie tekening), met

$$x = \frac{X}{X+Y+Z}$$

$$y = \frac{Y}{X+Y+Z}$$



Figuur 4.4: Het CIE kleur chromaciteitsdiagram, bepaald door x en y.

De CIE-$L^*u^*v^*$ kleurenruimte is rechtstreeks gebaseerd op de CIE-XYZ kleurenruimte, en linearizeert het waarnemen van de kleurverschillen. Deze relatie is gegeven door:

$$L^* = 116(\frac{Y}{Y_n})^{\frac{1}{3}} - 16,$$

$$u^* = 13L^*(u' - u'_n),$$

$$v^* = 13L^*(v' - v'_n).$$

Hier refereren u'_n en v'_n naar wit licht, en u' en v' zijn gegeven door

$$u' = \frac{4X}{(X + 15Y + 3Z)}$$

$$v' = \frac{9Y}{(X + 15Y + 3Z)}$$

Door nu de CIE-$L^*u^*v^*$ kleurenruimte te gebruiken, zullen resultaten bekomen worden die dichter aanleunen bij de perceptie van het menselijk oog. Verder onderzoek hiernaar is nodig.

4.2.2 Transities verbeteren

Soms kunnen transities waar de frames te veel verschillen slechte resultaten geven. Om dit op te lossen zijn enkele rendering algoritmen mogelijk.

Blenden

2 frames kunnen in elkaar geblend worden door voor deze 2 frames tussenliggende frames te berekenen. Dit kan gedaan worden door de kleurwaarden per pixel te interpoleren. Hoewel dit mooiere transities geeft, worden de frames tijdelijk geblurd.

Cross-fading

Cross-fading gaat een stap verder dan blenden. Hier kunnen 2 sequenties van frames met elkaar geblend worden, zodat er een mooiere overgang is van de ene sequentie naar de andere. Ook hier zijn blurs een probleem.

Morphing

Morphing zoekt overeenkomstige punten in 2 frames en gaat deze overeenkomsten vloeiender naar elkaar laten overgaan door tussen deze punten te interpoleren. Door niet tussen de kleurwaarden per pixel te interpoleren, worden blurs bestreden. Daar tegenover staat wel een extra aan rekentijd, en is het zoeken naar overeenkomsten vaak niet eenvoudig.

4.3 Conclusie

Om animaties te maken met video textures, is een goede metriek om transities te bepalen cruciaal. Eenvoudige metrieken, zoals het gemiddelde en de standaardafwijking, leveren slechte resultaten (voor een vergelijking zie sectie 8.1.1). De metriek die de beste resultaten geeft, is de L_2 norm. Omdat deze metriek pixelvergelijkingen uitvoert, is ze erg traag voor grote hoeveelheden data. Daarom kan de gemiddelde kleur gebruikt worden om al een grote groep transities te verwijderen. Enkel wanneer de gemiddelde kleur van 2 frames nagenoeg gelijk is, is een goede transitie mogelijk en enkel in dit geval moet nagegaan worden of de L_2 norm de transitie behoudt.
Hoewel de L_2 norm goede resultaten geeft, is ze niet ideaal. Kleine schommelingen in globale lichtsterkte hebben geen negatief effect op de L_2 norm. Toch zijn deze overgangen storend voor het menselijk oog. Door andere kleurruimten te gebruiken die dichter bij de waarnemingen van het menselijk oog staan, kan dit probleem opgelost worden. Dit wordt verder besproken in sectie 8.1.2.
Een andere manier om minder goede transities te verbeteren, is om blending of morphing te gebruiken. Het nadeel van blending is het tijdelijk onscherp maken van de frames. Morphing is dan weer complex in minder voor de hand liggend om toe te passen.

Hoofdstuk 5

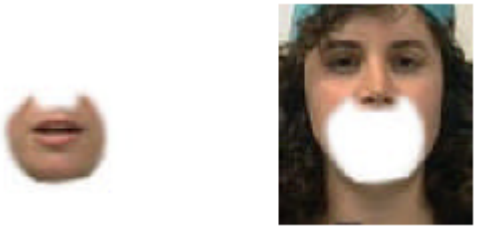
Animaties met video textures

In vorig hoofdstuk werden metrieken aangegeven om goede transities tussen video textures te vinden. Deze metrieken kunnen gebruikt worden om er animaties mee te maken. Enkele mogelijke algoritme komen in dit hoofdstuk aan bod.

5.1 Voorloper: video rewrite

Het eerste algoritme dat nieuwe animaties maakt van bestaande videosequenties is video rewrite [BCM97]. Dit werd gedaan op basis van de klonken die de persoon uitsprak. Zo kan men de gefilmde persoon woorden laat uitspreken die niet in de originele videosequenties zaten. Deze techniek kan gebruikt worden voor o.a. film dubbing, om lip synchronisatie te bekomen.

Video rewrite labelt de audio phonemes automatisch, en houdt de bijbehorende video afbeeldingen bij. Wanneer dan een nieuwe audio sequentie wordt ingegeven, wordt de volgorde van de afbeeldingen zo aangepast dat de afbeeldingen in overeenstemming zijn met de audio. Wanneer geen goede afbeelding beschikbaar is, wordt de meest gelijkaardige gekozen. Vervolgens wordt de bekomen sequentie terug op de achtergrond gemapt, met de nodige correcties aan de positie en de orientatie van het hoofd.



Figuur 5.1: Het masker om de mond uit een videosequentie te halen. Dit masker werd dan gebruikt om nieuwe animaties terug op de achtergrond te zetten.

Hoewel video rewrite werkt met audio samples, is het toch het eerste algoritme dat animaties van video textures voorziet. Er werd een masker voor de mond gebruikt om zo de mond van de gefilmde persoon uit de video te halen, en de bekomen afbeeldingen werden

dan herordert zodat ze de nieuwe audio sequentie matcht. Als metriek om transities te bepalen werden audio samples gebruikt, omdat audio samples en mondbewegingen sterk met elkaar gerelateerd zijn. Daardoor zijn discontinuïteiten haast uitgesloten. Toch werden morphing technieken gebruikt om de transities te verbeteren.

Figuur 5.2: Resultaat na het herordenen van de mond textures.

5.2 Video textures

Een manier om video-based animations te maken, is om de volgorde van afspelen van de video frames te veranderen. De volgorde van de video frames kan enkel zo aangepast worden, dat de nieuwe transities gelijkaardige zijn aan de originele transities. Dit is nodig om een goede animatie te maken. Een eerste methode om willekeurige videosequenties te bekomen, is door voor elke frame de transities met alle andere video frames te berekenen en 1 of enkele goede transities bij te houden. Zo kan dan voor de hele videosequentie 1 graaf opgesteld worden, die dan tijdens de synthese fase doorlopen kan worden om oneindige animaties te krijgen. De animatie mogelijkheden zijn eerder beperkt, maar in sommige gevallen toch voldoende.

5.2.1 Analyse

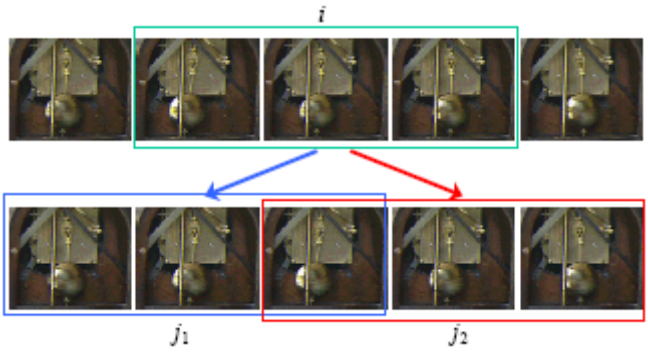
Eerst moeten alle gelijkaardige paren frames gevonden worden. Dit gebeurt door voor elke 2 frames hun verschil $D_{i,j}$ te berekenen, met bijvoorbeeld de L_2 norm (goede metrieken voor D werden in vorig hoofdstuk beschreven). Frame j is een mogelijke volgende frame van frame i wanneer $D_{i+1,j}$ laag is, dus wanneer ze weinig in kleur verschillen. Dit wil zeggen wanneer het normale volgende frame van frame i en het te testen volgende frame j sterk gelijk zijn.

Maar dit is echter niet voldoende. Wanneer bijvoorbeeld een object een bepaalde beweging volgt, moet deze beweging behouden blijven. Anders kan dit object immers voortdurend van beweging veranderen, wat meestal niet gewenst is. Een eerste mogelijkheid is om een richtingsvector bij te houden. Helaas is het berekenen van zo een richtingsvector voor video textures niet zo eenvoudig. Gelukkig is er een simpelere oplossing, die toch gelijkaardig resultaat geeft.
Om een beweging van een object te detecteren, is er meer dan 1 frame nodig tijdens het testen voor een goede transitie. Daarom is het beter meerdere frames in de test op te nemen.

Zo wordt de richting van de beweging behouden. Wanneer meerdere frames gebruikt worden, moeten de dichterbij gelegen frames feller worden meergeteld dan verder af gelegen frames. Dit kan gedaan worden door binomiale gewichten te gebruiken. Dit geeft dan

$$D'_{i,j} = \sum_{k=-m}^{m-1} w_k D_{i+k,j+k}.$$

Meestal wordt gebruikt gemaakt van een filter met even lengte. Voor een filter met lengte 2 wil dit zeggen dat frame i en frame $j-1$ gelijkaardig moeten zijn, alsook frame $i+1$ en frame j. Dit vermijdt asymmetrie, omdat beide transities even fel meetellen. Wanneer de transitie van frame $i+1$ naar frame j meer zou meetellen als de andere transitie, zou dit asymmetrie veroorzaken.



Figuur 5.3: Een voorbeeld van mogelijke transities. Frame i lijkt sterk op frame j_1 en frame j_2. Maar wanneer de naburige frames meegerekend worden, is enkel frame j_2 nog gelijk. Frame j_1 zit immers in een andere beweging.

Een ander probleem zijn de doodlopende eindes: frames waar onvoldoende goede volgende frames voor bestaan. Deze kunnen de animatie laten vastlopen, zoals in onderstaande tekening wordt geïllustreerd. Dit kan opgelost worden door voor elke mogelijke transitie na te gaan of er voldoende goede volgende frames zijn. Maar dit verschuift het probleem enkel. Wat als die volgende (of nog verder afgelegen) frames ook onvoldoende opvolgers hebben?

Figuur 5.4: Een videosequentie met een doodlopend einde: wanneer de hand zichtbaar wordt, zijn geen verdere animaties mogelijk. Er moet vermeden worden dat de animatie in deze serie frames geraakt.

Om dit probleem tijdig in rekening te brengen, moet voor elke transitie al diens mogelijke toekomstige kosten bij opgeteld worden,

$$D''_{i,j} = \left(D'_{i,j}\right)^p + \alpha \sum_k P''_{j,k} D''_{j,k}.$$

Hier is p een constante dit aangeeft hoe fel een minder goede transitie weegt t.o.v. verschillende goede transities. Hierdoor kan dus een minder goede transitie worden toegestaan om een betere reeks te bekomen. α bepaalt hoe fel de toekomstige transities mogen meetellen. Om convergentie te behouden moet $0 < \alpha < 1$.
Voor elke transitie moet dan diens kans bepaald worden. $P''_{i,j}$ geeft de kans aan dat frame j de opvolger wordt van frame i, en kan bekomen worden door

$$P''_{i,j} = exp\left(-D''_{i+1,j}/\sigma\right),$$

met σ typisch een klein veelvoud van het gemiddelde van de niet-nul $D''_{i,j}$ waarden.
Hoewel dit met een eenvoudig iteratief algoritme opgelost kan worden, is de convergentie traag. Hiervoor bestaat een sneller algoritme.

Wanneer $\sigma \to 0$, zal $P''_{j,k}$ naar 1 gaan voor de beste transities, anders naar 0. Daarom geldt

$$D''_{i,j} = \left(D'_{i,j}\right)^p + \alpha\, min_k\, D''_{j,k}.$$

Nu kan een lage transitiekost gezien worden als een transitie aan het einde van een reeks. Door deze laagste kost van een reeks bij te houden, kan het feit dat de reeks een doodlopend eind is beter vooraf meegeteld worden. Wanneer dan $D''_{i,j} = (D'_{i,j})^p$ geïnitialiseerd wordt en

$$m_j = min_k D''_{j,k},$$

kunnen toekomstige transities meegeteld worden met

$$D''_{i,j} = \left(D'_{i,j}\right)^p + \alpha\, m_j$$

en vervolgens m_j up te daten. Dit moet herhaald worden tot de transitiekosten stabiliseren.

Hoewel bovenstaand algoritme goede transities berekend, is het meestal aangeraden het aantal transities per frame te verminderen, niet alleen om geheugen te sparen, maar ook om de kwaliteit te verhogen.
Thresholding zal de minder goede transities te verwijderen, maar thresholding is meestal niet voldoende. Wanneer nu voor een frame j verschillende opeenvolgende frames geschikt zijn als mogelijke opvolger van frame i, is het voldoende om slechts 1 (of enkel) bij te houden. Dit kan gedaan worden door het lokaal maximum te berekenen. Dit maximum is per definitie de beste transitie binnen het gebruikte interval. Best worden beide strategieën gecombineerd.

5.2.2 Synthese

Door bovenstaande graaf te doorlopen, worden dan de oneindige animaties gemaakt. Dit kan op 2 manieren:

- volledig random
- m.b.v. looping

Random play

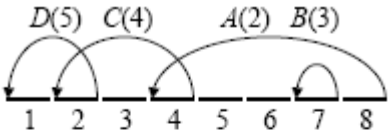
Een eerste algoritme om video loops te maken is door random play. Dit is een zeer eenvoudig algoritme dat voor elke frame i een volgende frame j bepaald aan de hand van de kans $P_{i,j}$. Door een simpele Monte Carlo techniek kan dan elk volgende frame gekozen worden. Monte Carlo technieken bieden eenvoudige oplossingen voor moeilijke wiskundige pro-blemen. In dit geval worden de mogelijke transities voor een video frame genormaliseerd, zodat de som 1 is. Vervolgens wordt een random waarde gekozen (= Monte Carlo variabele), en deze wordt gebruikt om de volgende transitie te bepalen. Voor elke volgende transitie wordt diens kost opgeteld bij een teller, totdat deze teller groter wordt dan de Monte Carlo variabele. Deze transitie wordt dan gekozen.
Merk op dat voor frame i meestal frame $i + 1$ de beste kandidaat is, maar niet altijd. Door het vermijden van doodlopende eindes kan dit namelijk anders zijn.

Looping

Een andere manier om video loops te maken, is door een video speler te gebruiken die dan de video in loop mode afspeeld. Hierdoor is het noodzakelijk dat de video zelf een terugkerende loop is. Het einde en begin moet immers op elkaar volgen om geen merkbare sprong in de loop te hebben. Daarom is het noodzakelijk voor een sequentie van frame i naar frame j een algoritme te hebben dat van frame j een sequentie van frames zal vinden dat terug naar frame i keert.

Een enkelvoudige transitie van frame i naar frame j wordt ook wel een primaire loop genoemd. Een enkelvoudige transitie kan enkel een cyclus creëren wanneer $i >= j$. De kost van deze transitie is $D'_{i,j}$.
Primaire loops kunnen gecombineerd worden. Dit worden dan samengestelde loops. Om 2 loops te kunnen combineren, moeten ze elkaar overlappen. Anders is het onmogelijk de eerste loop af te spelen na de tweede. De samengestelde loop heeft als bereik de unie van de 2 loops, en de kost is de som van beide.
Voorwaartse transities zijn ook mogelijk. Dit zijn transities van frame i naar frame j waar $i + 1 < j$.

De simpelste manier om de beste samengestelde loop met lengte L te hebben, is alle mogelijke loops van lengte L te creëren en hieruit de beste te nemen. Maar deze methode is exponentieel is dus niet bruikbaar. Betere methoden zijn nodig.
Een betere methode is een dynamische aanpak, door iteratief te werken met de lengte van de loops. Om te beginnen is een tabel van L op N nodig, waar L de maximale toegestane loop lengte is en N is het aantal primaire loops. Het idee is nu telkens de beste samengestelde loop te creëren door gebruikt te maken van de primaire en eerder berekende samengestelde loops. Elke cel van de tabel houdt dan de gebruikte samengestelde loops bij, alsook de totale kost van deze loop. De tabel wordt nu ingevuld, rij per rij en cel per cel. Voor elke cel worden alle kortere loops afgegaan binnen dezelfde kolom. Alle loops worden gecombineerd om een loop te vinden met overeenkomstige lengte dan de rij nummer. De loop met de laagste kost wordt dan bijgehouden.



length	A(2)	B(3)	C(4)	D(5)
1		$B^1(3)$		
2		$B^2(6)$		$D^1(5)$
3		$B^3(9)$	$C^1(4)$	
4		$B^4(12)$		$D^2(10)$
5	$A^1(2)$	$B^5(15)$	$CD(9)$	$CD(9)$
6	$AB(5)$	$AB(5)$	$C^2(8)$	$D^3(15)$
		⋮		

Figuur 5.5: De tabel voor het bepalen van de loops. Per rij is de beste loop met de gegeven lengte die de primaire loop van de kolom waar hij inzit bevat. De kost staat tussen haakjes.

Voor een tabel van L op N, moet er voor elke cel maximaal $L-1$ loops en $N-1$ rijen afgegaan worden. Dit wil zeggen dat het algoritme per cel $O(LN)$ is, oftewel het hele algorime $O(L^2N^2)$. Dit is uiteraart veel beter dan de naïeve exponentiële aanpak. De geheugen complexiteit is $O(LN)$.

Nadat de beste samengestelde loops gekend zijn, moet nog de volgorde waarin deze loops worden afgegaan bepaald worden zodat ze een geldig geheel vormen. Het algoritme steunt op 5 regels:

- Zet de transitie die begint bij het uiteinde van de sequentie als eerste, en verwijder deze transitie uit de lijst. In bovenstaand voorbeeld is dit transitie A.
- Het verwijderen van een transitie van frame i naar frame j splitst de overige transitie in 1 of meer groepen. Door transitie A te verwijderen, zijn er 2 sets: $\{C, D\}$ en $\{B\}$. Frame j is altijd in de eerste groep en een volgende transitie is een transitie die begint na frame j. Dit is hier transitie C.
- Herhaal vorige stap; verwijder alle transitie van frame i naar frame j totdat er geen primaire loops meer zijn in de eerste groep. Dit is transitie D.
- Kies een volgende transitie uit een andere groep, met het algoritme uit stap 2. In dit voorbeeld blijft enkel nog transitie D over.
- Keer terug naar stap 2 tot alle primaire loops verwijderd zijn.

De volgorde waarin de transitie worden afgespeeld in bovenstaande voorbeeld is dus A, C, D, B.
De complexitiet van dit algortime is kwadratisch volgens het aantal samengestelde loops. De keuze van een transitie in stap 2 kan gebeuren volgens een deterministische manier, of via een random waarde. Deze laatste leunt aan bij de Monte Carlo methode eerder beschreven.

5.2.3 Uitbreidingen

In deze sectie worden enkele uitbreidingen die mogelijk zijn met video textures besproken. Deze zijn toevoeging van audio samples, 3D video textures en een partiële aanpak.

Audio

Geluid kan eenvoudig worden toegevoegd bij video textures. Telkens er een nieuwe frame afgespeeld wordt, wordt de oveneenkomstige audio sample mee afgepeeld. Wanneer voor een nieuwe transities de audio samples te fel verschillen, kunnen cross-fading technieken gekozen worden.
Hoewel dit goed werkt voor fenomenen als water en vuur, zal voor complexere audio fragmenten dit falen. Wanneer immers de frequenties per frames te fel verschillen, zullen alle audio samples telkens opnieuw met elkaar gefade moeten worden, waardoor de kwaliteit van de audio samples drastisch achteruit gaat. Zo is het onmogelijk spraak te behouden. Hiervoor moet tijdens het kiezen van de transities al met de audio sample mee rekening gehouden worden.

3D video textures

Verschillende video textures kunnen gecombineerd worden tot een 3D video textures. Dit kan bekomen worden door view interpolatie technieken uit de image-based rendering (zie sectie 2.1.2). Het enige verschil met andere video-based rendertechnieken is dat de videosequenties hier gesynthetiseerd zijn.

Partiële aanpak

Een andere uitbreiding is de partiële aanpak. Hier worden de video textures opgesplitst in een aantal gebieden. Voor elk gebied worden dan de textures apart berekend. Nadien worden deze dan weer bij elkaar gezet.

Figuur 5.6: Links zijn 3 gebieden die elk hun eigen video texture definiëren. Rechts worden de video textures voor beide bewegingen apart berekend.

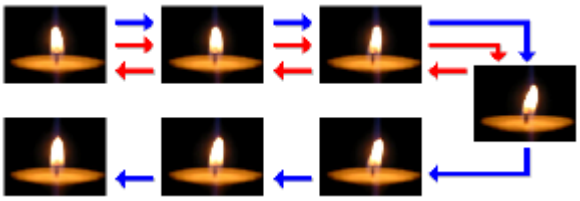
Hoewel dit in sommige gevallen goede resultaten geeft (voorbeeld met 2 kinderen op een schommel), is de toepasbaarheid eerder beperkt. Het terug bij elkaar zetten van de video textures moet immers onafhankelijk van elkaar kunnen gebeuren. Wanneer de ene video texture invloed heeft op de andere, wordt er met de partiële aanpak geen rekening mee gehouden. Dit levert dan slechte resultaten. Wanneer bijvoorbeeld dit toegepast zou worden op een hond, waar dan het hoofd en (delen van) het lichaam elk een eigen video

texture vormen, moet ervoor gezorgd worden dat tijdens de synthese geen onrealistische bewegingen toegelaten worden. Hierdoor is interactie tussen de partiële video textures nodig, hetgeen in strijd is met het idee van de partiële aanpak.
Een mogelijke oplossing hiervoor is om een soort skeletstructuur voor de bewegingen bij te houden en beperkingen op mogelijke transities toe te voegen die het skelet in onnatuurlijke posen zouden brengen. Zowel het bepalen van een skeletstructuur als aangeven van de beperkingen is erg kostelijk en tijdrovend, en wordt een groot voordeel van video textures t.o.v. andere animatie technieken verloren, namelijk een beperkte inbreng van de animator.

5.3 Symmetrie uitbuiten

Het berekenen van goede transities tussen video texture paren door alle mogelijken combinaties af te gaan, zal voor grotere videosequenties enige tijd in beslag nemen. Wanneer het gefilmde object dan nog eens een complexe structuur heeft die voortdurend, zei het weinig, verandert, zullen vele van de berekende transities slecht zijn. Een andere mogelijkheid om oneindige animaties te krijgen, is door tijdens het afspelen van een video fragment de richting van het afspelen om te draaien. Dit kan enkel op symmetrische punten binnen het video fragment [HR05].

5.3.1 Analyse



Figuur 5.7: Een voorbeeld van een symmetrisch punt.

Het doel is dus het vinden van die frames, waar de symmetrie hoog is. Dit wil dus zeggen dat voor een frame i de n vorige en n volgende frames sterk op elkaar gelijken. n bepaald hier de filtergrootte. Hoe groter n, des te meer frames symmetrisch moeten zijn. Meestal is het voldoende slechts enkele frames mee te laten tellen, omdat het bepalen van de symmetrie lokaal is. Wanneer te veel frames in de test worden gebruikt, zullen er minder symmetrie punten worden gevonden en hierdoor zal de animatie sneller in herhaling vallen. Een hoge filtergrootte vermijdt wel snelle veranderingen van richting.
Een andere belangrijke opmerking hier is dat de nabije frames vaak belangrijker zijn dan verder afgelegen frames. Hierdoor worden binomiale coëfficienten gebruikt:

$$Sym_i = \sum_{r=1}^{n} \binom{2(n-s)}{n-r} \cdot D_{i-r,i+r} \qquad met \;\; s < \frac{n}{2}.$$

De $D_{i,j}$ geeft het verschil aan tussen frame i en frame j. Meestal wordt de L_2 afstand gebruikt (zie sectie 4.2). s bepaalt hoe fel de meer afgelegen frames meetellen. Een hogere

waarde laat de dichtbije frames feller meetellen en de verder afgelegen minder.
Voor het gemak worden de resultaten in het interval tussen 0 en 1 gemapt. Hier stelt 0 de afwezigheid van symmetrie voor en 1 de maximale symmetrie. Deze mapping gebeurt door de hoogste symmetrie waarde te zoeken en vervolgens alle waarden in te vullen in

$$Sym_i = 1 - \frac{Sym_i}{Sym_{max}}.$$

Wanneer de symmetrie waarde voor elke frame berekend is, kunnen dan die frames gekozen worden met de beste symmetrie waarden. Een eerste manier is thresholding. Hier wordt een ondergrens gekozen en alle symmetrie waarden kleiner dan deze threshold worden op 0 gezet. Zo worden enkel die transities overgehouden waar er voldoende symmetrie is.
Hoewel dit vele eerder slechte kandidaten verwijdert, is deze methode niet ideaal. Wanneer nu verschillende ongeveer even symmetrische frames na elkaar zijn, is het aan te raden slechts 1 (of enkele) te behouden. Door verschillende symmetrie frames achter elkaar te houden, is het mogelijk dat dezelfde reeks frames vaker herhaald worden, hetgeen al snel merkbare herhalingen geeft. Door het aantal symmetrie frames binnen een gebied te beperken, wordt dit vermeden. Zo wordt in een bepaald interval het maximum gezocht en de overige waarden worden ook weer op 0 te zetten. Op deze manier zal er in een bepaald gebied slechts 1 symmetrie frame bijhouden worden, namelijk het frame waar deze symmetrie maximaal is.

Nadat de frames die geschikt zijn voor symmetrie bepaald zijn, moet nog de kans om deze symmetrie te gebruiken berekend worden. Voor een volledig symmetrisch frame (symmetrie waarde gelijk aan 1) moet de kans op omkeren van het afspelen van de video 50% zijn. Zo is voor het meest symmetrische frame de kans van het aanhouden van de huidige speelvolgorde en het omkeren van deze speelvolgorde gelijk. Dus de kans op omkeren voor een frame i is

$$P_{max_i} = \frac{Sym_i}{2}.$$

Een lage kanswaarde wil dus zeggen dat het minder waarschijnlijk is de volgorde van het afspelen om te keren. Om doodlopende eindes te vermijden, moeten de eerste en laatste kanswaarde op 1 gezet worden. Anders zou de video vastlopen en stoppen.

5.3.2 Synthese

Wanneer de symmetrische frames en bijbehorende kanswaarden berekend zijn, kunnen deze gebruikt worden om een oneindige animatie te maken zonder dat herhalingen opvallen. Vooraf moet wel nog voor elke symmetrische frame een startwaarde gekozen worden. Deze gaat telkens gebruikt worden om na te gaan of de volgorde van afspelen omgekeerd wordt of niet. Een mogelijkheid is de helft van de maximale waarde, zoals hierboven is vermeld.

Vervolgens kan het afspelen van de animatie beginnen. Wanneer er een symmetrisch frame wordt bereikt, worden diens waarde geupdated, afhankelijk of de volgorde van afspelen al dan niet is omgekeerd. Indien dit het geval was, wordt de kans verlaagd. Anders

wordt ze verhoogd zodat de kans op omkeren de volgende keer groter is. Dit geeft dus

$$P_i = max\,(P_i - incr, 0) \qquad bij\ omekeer$$
$$P_i = min\,(P_i + incr, P_{max_i}) \qquad anders$$

De kansen van de 2 uitersten moet wel ongewijzigd blijven om de animatie niet te laten stoppen. Het genereren van de videosequentie is volledig random en kan real-time gebeuren.

5.3.3 Hybride methode

Het is mogelijk de originele video texture methode te combineren met de symmetrische aanpak. Zo zullen de voordelen van beide methode bekomen worden. Door voldoende random transities toe te laten, zal de animaties niet snel in herhaling vallen. En de symmetrische punten garanderen transities voor complexe objecten en vermijden visuele artifacten.

Het probleem is een goede integratie van beide methoden te vinden. Een mogelijkheid is voor elk symmetrisch punt de originele video texture methode te gebruiken om extra transities te bekomen. Dit gaat herhalingen tegen. In het voorbeeld met de klok waren de symmetrische punten ook vaak de frames waar het origineel algoritme nieuwe transities nam. Door deze integratie zal de animatie verbeteren.
Voor sommige andere voorbeelden is deze integratie minder voor de hand liggend. Zeker videosequenties waar slecht 1 van beide algoritmen voldoende goede resultaten levert, zal ook de hybride aanpak de kwaliteit niet kunnen verbeteren. Een voorbeeld is een plant onderworpen aan wind. De symmetrische aanpak werkt hierbij goed, maar de originele aanpak faalde. Door de hybride aanpak toe te passen op dit voorbeeld, zal de algemene kwaliteit niet hoger liggen dan met de symmetrische aanpak, aangezien enkel deze aanpak goede transities vond.

5.4 Cartoon textures

Video textures kunnen op verschillende types van invoer data werken. Zo kunnen o.a. cartoon sequenties gebruikt worden om nieuwe animaties te maken. Dit zijn dan cartoon textures [dJB04], en zijn een uitbreiding op video textures. De animator geeft enkel een begin en eind cartoon aan, en dan wordt automatisch een cartoon animatie gegenereerd. Verder biedt het systeem voorgesteld door de Juan and Bodenheimer een mechanisme om automatisch discontinuiteïten op te sporen.

In vergelijking met video textures worden bij cartoon textures ook alle texture paren met elkaar vergeleken op zoek naar goede transities. Een eerste verschil is de gebruikte metriek. De L_2 afstand is minder geschikt voor cartoons. Hierdoor werd een andere metriek gebruikt, namelijk de Haussdorf afstand, uitgelegd in vorig hoofdstuk. Deze metriek werkt heel goed voor cartoons, aangezien cartoons vaak egale vlakken hebben en hierdoor zijn de randen van de cartoon het belangrijkste.
Een ander verschil is dat voor cartoon textures er gebruik gemaakt wordt van Isomaps. De Isomaps zorgen voor niet-lineaire dimensie vermindering. In deze maps wordt bijgehouden

welke frames bij elkaar passen en welke niet. Deze maps kunnen dan gebruikt worden om nieuwe animaties te maken. De gebruiker geeft een start en eind frame, en dan wordt in de Isomap het kortste pad tussen beide frames gezocht. Dit levert dan de gezochte animatie op.

Figuur 5.8: Een cartoon texture, bekomen door frame 326 en 98 respectievelijk als begin en eind frame aan te duiden. De andere frames werden bepaald door de Isomap.

Cartoon textures zijn een uitbreiding van video textures, toegepast op cartoons. Van een gegeven videosequentie kunnen door video texture methoden nieuwe videosequenties gegenereerd worden. Er is geen informatie over het model nodig. Hierdoor hoeft een animator niet meer alles zelf te doen; enkele door nieuwe key frames te specifiëren en eerder gemaakte cartoons kunnen dan de nieuwe animaties gemaakt worden. Wel is het mogelijk dat tussen sommige key-frames onvoldoende informatie aanwezig is, zodat de animator deze nog moet toevoegen.

Cartoon textures kunnen gecombineerd worden met key framing technieken en motion graphs [HFR05]. De animator geeft eerst een aantal key frames op. Vervolgens wordt tussen de key frames interpolatie frames berekend, zodat een visueel goede sequentie van frames bekomen is. Daarna wordt met de nieuwe frames een graaf opgesteld die de beste serie frames aangeeft om van de ene key frame naar de andere te gaan. Dan kunnen door het veranderen van de volgorde van de nieuwe frames nieuwe animaties gemaakt worden. Deze methode levert vloeiende, lage animaties die geen postprocessing vereist.

5.5 Toepassingen

Nu volgen enkele toepassingen waar video texture methoden gebruikt worden om nieuwe videosequenties te genereren.

5.5.1 Panoramic video textures

Video textures lieten toe om van een scène, opgenomen met een vaste camera, een oneindige animatie te maken. Panoramic video textures [AZP+05] gaat verschillende video textures van eenzelfde scène, bekomen door de camera rond een welbepaalde as te roteren, combineren om een breder zicht op de scène te krijgen.

Het probleem is dat niet voor elk deel van de scène op ieder tijdstip voldoende informatie gegeven is. Deze informatie moet berekend worden uit overeenkomstige informatie van hetzelfde deel van de scène maar van een ander tijdstip. Deze informatie wordt bekomen

door pixels uit het invoerfragment te kopiëren naar de output. Het invoerfragment bevat informatie op een ander tijdstip over een deel van de scène waar in het outputfragment geen informatie over beschikbaar is.
Dus voor een gegeven uitvoer pixel $p = (x, y, t)$, waar x en y de positie en t de tijd voorstellen, moet dus een mapping $\Delta(p)$ van de vorm $(0, 0, \delta(p))$ gezocht worden zodat (x, y, t) gemapt wordt op $(x, y, t + \delta(x, y, t))$. Dit wil zeggen concreet zeggen dat voor elke pixel (x, y) informatie gezocht moet worden op andere tijdstippen zodat deze pixel op tijdstip t informatie krijgt.

Een eerste eenvoudige methode zoekt voor een bepaald deel van de scène overeenkomstige delen in de invoerdata en deze informatie wordt dan sequentiëel gekopiëerd. De reeds gekende delen worden niet op dit nieuwe tijdstip mee opgenomen. Helaas kan dit de structuur van de beweging van de scène veranderen, en de bekomen animatie lijkt ook niet vloeiend.

Door nu de scène als een 3 dimensionale omgeving te benaderen, kan bovenstaand probleem opgelost worden. Hierdoor moet voor iedere uitvoer pixel p een mapping $\Delta(p)$ gezocht worden zodat $V(p + \Delta(p)) \neq 0$, en hiervoor moet de meest vloeiende mapping gekozen worden, indien er meerdere mappingen mogelijk zijn. Voor het berekenen van deze mapping wordt naar de literatuur verwezen [AZP+05].

Figuur 5.9: Panoramic video textures: links is de input video, in het midden is de simpele aanpak en rechts de geavanceerde methode. Hoewel de simpele aanpak realistische resultaten levert, is de geavanceerde aanpak toch beter omdat deze de globale structuur behoudt.

Panoramic video textures laten toe om ontbrekende informatie van een scène op bepaalde tijdstippen op te lossen, door overeenkomstige informatie te zoeken op andere tijdstippen. Dit geeft voor stationaire scène zeer goede resultaten. Ook periodische fenomenen zoals water en vuur komen realistischer over in vergelijking met eenvoudige mappingen (zie tekening).
Maar panoramic video textures zijn niet altijd geschikt om bepaalde scènes te visualiseren. Het grootste nadeel is dat dit algoritme slechts een beperkte toepasbaarheid heeft. Enkel stationaire scènes kunnen gebruikt worden. Dynamische scène zijn niet geschikt, omdat deze door de tijd voortdurend veranderen. Hierdoor kan geen goede mapping gevonden worden.
Verder zijn ook niet-periodische fenomenem, alsook periodische fenomenen zonder een volledige cyclus, onmogelijk te visualiseren. Dit komt omdat er onvoldoende informatie is over het gefilmde fenomeen, en hierdoor kan niet op ieder tijdstip een goede mapping

gevonden worden. Hierdoor is het vrij bewegen van een camera door eender welke scène (nog) onmogelijk.

5.5.2 Graphcut textures

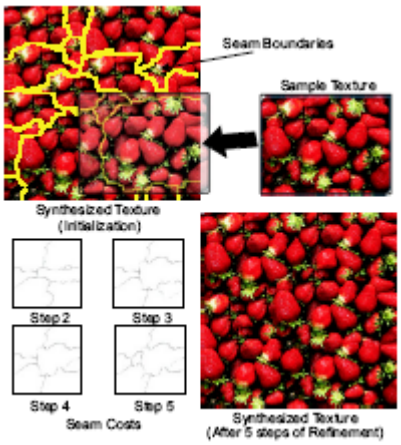
Texture mapping laat toe om een patroon van beperkte grootte op een veel groter object te mappen. Hiervoor wordt vaak van een kleine texture een oneindig grootte texture gegenereerd. Belangrijk hierbij is om het patroon van de texture te behouden. Daarom moeten bepaalde "patches" van de texture op de juiste manier gekopiëerd worden om hetzelfde patroon te behouden. Graphcut textures [KSEB03] gebruiken graphcut methoden om de optimale randen van de patches te berekenen.

2D synthese

Texture synthese is erg moeilijk. Het veranderen van de grootte van een texture heeft direct impact op diens structuur. Toch moet hetzelfde patroon behouden blijven. Texture synthese algoritmen moeten daarom met 2 problemen rekening houden:

- waar moet de invoer texture gepositioneerd worden t.o.v. de uitvoer textures.
- en welk deel van de invoer texture moet getransformeerd worden naar de uitvoer texture.

Door dit probleem nu te formuleren als een minimal-cost graph cut probleem, kan door middel van graph cut technieken nieuwe textures gecreëerd worden. Door elke pixel voor te stellen als een node, kan een graaf van de invoer texture berekend worden. Door dan een zoekalgoritme te gebruiken dat op zoek gaat naar de beste invoer patches, kan de uitvoer texture uitgebeid worden. Belangrijk is de globale structuur van de invoer texture ook in de uitvoer texture te bewaren.

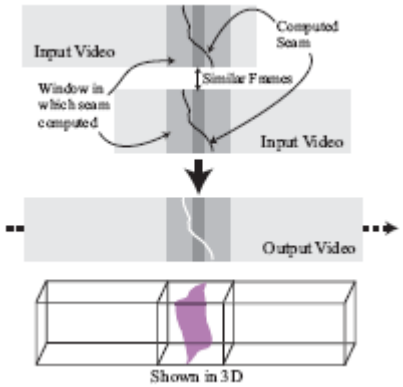


Figuur 5.10: Texture synthese door middel van graph cut technieken.

Graph cut technieken zoeken bij elkaar horende pixels in een texture. Deze pixels vormen een patch; een onregelmatig vlak dat een bepaalde structuur heeft. Deze patches moeten dan naast elkaar gekopiëerd worden op een manier dat de globale structuur van de patches behouden blijft. Opnieuw worden graph cut technieken gebruikt om overeenkomstige delen van verschillende patches te bepalen, zodat de juiste delen aan elkaar gekopiëerd worden.

Video synthese

Een belangrijk voordeel van graph cuts is dat deze algoritmen uitbreidbaar zijn voor 3D toepassingen. Hierdoor kunnen ook graphcut textures gebruikt worden voor video synthese. Patches worden dan volledig in 3 dimensionale ruimte gedefiniëerd. Door vervolgens het zoekalgoritme uit te breiden voor 3D, kunnen dan dezelfde technieken als de 2D synthese gebruikt worden. Het enige verschil is dat de textures niet alleen over het spatiaal domein veranderen, maar ook over de tijd. Door sterk overeenkomstige frames elkaar te laten vervangen, kunnen de textures over de tijd veranderen. Deze werkwijze is dezelfde als de originele video textures [SSSE00].

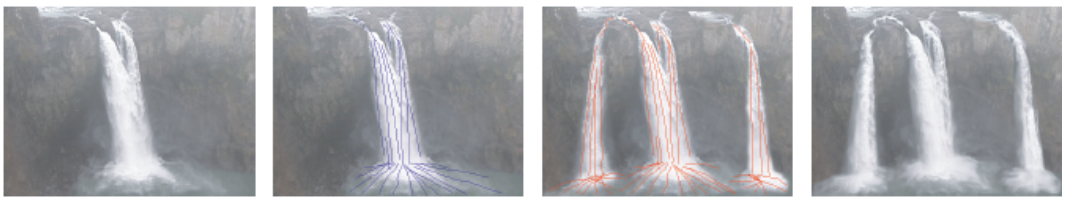


Figuur 5.11: Graph cut textures gebruikt voor video. Wanneer 2 gelijkaardige frames gevonden worden, kunnen graph cut technieken gebruikt worden om de videosequenties aan elkaar te koppelen.

5.5.3 Flow-based video synthesis and editing

Video textures konden slechts moeilijk periodische fenomenen zoals een waterval of een rookwolk uit een schouw simuleren. Panoramic video textures boden hiervoor een oplossing. Een andere aanpak is de flow-based video synthesis [BSHK03].

De gebruiker moet eerst een aantal flow-lines definiëren. Dit zijn lijnen die over de animatie getrokken worden. Vervolgens worden particles over deze lijnen bewogen. Deze particles definiëren video texture: een groep van textures die veranderen over de tijd.
Daarna kan de gebruiker nieuwe flow-lines specifiëren in de output video. Over deze lijnen worden dan gelijkaardige streams gegenereerd als de flow-lines in de input video. Hierdoor zijn nieuwe animaties mogelijk.



Figuur 5.12: Flow-based video synthesis and editing. Links is een input frame. Daarnaast de flow-lines van de input frame aangeduid door de gebruiker. Vervolgens duidt de gebruiker nieuwe flow-lines aan in de output frame. Rechts is het bekomen resultaat.

5.5.4 Auto-regressive process

Video textures liet toe nieuwe videosequenties te genereren door de volgorde van de frames te veranderen. Wel moet opgelet worden dat de nieuwe videosequenties dezelfde impressies leveren als de originele videosequentie. Een mogelijke manier om zulke videosequenties te genereren, is door elk frame te transformeren naar de eigenruimte m.b.v. Principal Components Analysis. Zo kan de originele videosequentie gezien worden als een kenteken van de lage dimensionele ruimte.
Vervolgens kan dan de lage dimensionele ruimte doorlopen worden. Door nieuwe gelijkaardige kentekens te maken in deze ruimte, kunnen nieuwe video textures gemaakt worden. Deze kentekens kunnen bekomen worden door een auto-regressive process [CDG+04].

5.5.5 Motion graph

Video texture algoritmen kunnen toegepast worden op motion capturing bewegingen. Tussen deze bewegingen kan overgegaan worden zoals een transitie binnen video textures. Typisch worden deze bewegingen opgeslagen in een graaf. Dit geeft dan een motion graph [KGP02] [LCR+02] [AF02].
Het bepalen van transities in de graaf gebreurt analoog met het bepalen van een transitie van een video texture. Verschillende frames worden gebruikt en wanneer 2 series van zulke frames gelijkaardige bewegingen weergeven, is ertussen een transitie mogelijk.
Om nieuwe bewegingen te krijgen, kan de graaf doorlopen worden en kunnen transities genomen worden die de gewenste beweging genereren.

5.6 Slot

Video textures kunnen gebruikt worden om realistische animaties te maken. Het originele video texture algoritme vergeleek alle frame paren, en hield de beste transities bij. Door per frame tussen deze transities te kiezen, worden nieuwe animaties bekomen. De symmetrische aanpak zocht naar punten in de videosequentie waar de speelvolgorde kon omgedraaid worden. Beide methoden hebben voor- en nadelen, en worden verder vergeleken in sectie 8.2.3. Ook werd aangetoond dat video texture algoritmen op verschillende types data kan werken, waaronder cartoons. Vervolgens werden verschillende toepassingen van video textures vermeld. Dit illustreert het belang van video textures. Toch zijn video textures beperkt. Enkel de volgorde van de frames kan veranderd worden, en er is weinig controle over de animatie. Hiervoor zijn andere technieken nodig, zoals video sprites.

Hoofdstuk 6

Video sprites

Vorige hoofdstukken werd aangegeven hoe m.b.v. video textures animaties konden gemaakt worden. In dit hoofdstuk zullen video sprites [PPHL98] worden voorgesteld, alsook methoden om deze sprites uit videosequenties te halen. Daarnaast zullen metrieken beschreven worden die gebruikt kunnen worden om video sprites met elkaar te vergelijken. In het volgende hoofdstuk zullen dan algoritmen om animaties met deze video sprites te maken beschreven worden.

6.1 Definitie

Het verschil tussen video textures en video sprites is dat video textures werken op hele frames, waar video sprites enkel op bepaalde delen ervan werken. Hierdoor zijn er andere mogelijkheden. Zo kan de video sprite op elke positie op het scherm gerenderd worden, en is met toevoeging van zijn velocity path-following mogelijk. Dit laat een grotere waaier aan animaties toe, zoals in volgend hoofdstuk zal blijken.
Om bepaalde objecten uit een videofragment te animeren, moeten deze objecten eerst van de videoframes gescheiden worden. Deze objecten worden vervolgens als sprites voorgesteld. Dit hoofdstuk zal hier verder op ingaan.

6.1.1 3D video sprites

Net zoals video textures kunnen video sprites eveneens uitgebreid worden tot 3D video sprites [PH98] [PPHL98]. Hierbij worden verschillende camera's gebruikt. Elke camera staat op een verschillende positie en registreert zijn eigen videosequentie. Vervolgens worden alle videosequenties gemorpht tot 1 videosequentie. Dit wordt gedaan door interpolatie tussen de edges. De nu bekomen videosequentie is geblurd, en kan verbeterd worden door frequency controlled blending. Tot slot kunnen nieuwe virtuele viewpoint gebruikt worden om nieuwe videosequenties te bekomen.

6.2 Segmentatie van video sprites

Bij het opnemen van een model in een scène is meestal niet elk deel van de videosequentie even belangrijk. Typisch bevat een videosequentie veel overbodige informatie, zoals de

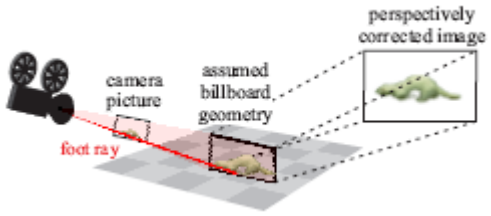
achtergrond. Het scheiden van voor- en achtergrond is bij video-based rendering erg belangrijk. Fouten hierin kunnen verdere rendering foutief of onmogelijk maken. Het is dus van belang een goed algoritme te hebben dat objecten uit een scène kan halen. Background subtraction is hiervoor waarschijnlijk het bekendste. Daarnaast wordt ook nog graph cut beschouwd. Beiden werden in detail besproken in hoofdstuk 3.

6.2.1 Vergelijking tussen background subtraction/graph cut

Het is moeilijk te zeggen welke van deze methode nu het beste is, omdat voor beide methoden allerlei varianten bestaan die vaak op specifieke data werken. Bij video-based animation, vertrekkende van een model met een vaste scène, zijn statische background subtraction modellen al voldoende. De achtergrond is constant en in vele gevallen een blue screen, ofwel is deze gekend. Meer geavanceerde modellen zijn niet nodig.
Graph cuts zijn eigenlijk al te geavanceerd voor deze scènes, omdat de achtergrond niet verandert en het model gekend is. Het telkens interactief aanduiden van enkele voor- en achtergrondpixels is een tijdsintensief proces, dat voor deze scènes overbodig lijkt (in vergelijking met simpele background subtraction).

6.3 Perspectieve correctie

Doordat de sprite op een vlak beweegt, is de kans groot dat de sprite tijdens het filmen naar de camera toe of ervan weg beweegt. Hierdoor zal de grootte van de sprites voortdurend veranderen. Dit kan het aantal transities verminderen.
Door de sprite op een bill-board te projecteren, krijgen alle sprites een gelijke grootte. Hierdoor zijn transities mogelijk tussen sprites die origineel op verschillende diepte werden gefilmd.



Figuur 6.1: De gefilmde sprite wordt op een bill-board geprojecteerd om perspectieve correctie toe te passen.

6.4 Transities voor video sprites

Door video textures uit te breiden naar video sprites, zijn enkele nieuwe eigenschappen aanwezig. Hierdoor zijn nieuwe metrieken mogelijk tijdens het bepalen van transities. Zo is kleur alleen niet meer voldoende om goede transities te bepalen. Verder bieden deze nieuwe eigenschappen nieuwe animatie mogelijkheden (zie volgend hoofdstuk). Ook bij video sprites kunnen blending en morphing technieken gebruikt worden om de kwaliteit van de transities te verbeteren.

6.4.1 Kleureigenschappen

Net zoals bij video textures is de kleur van de video sprites van cruciaal belang bij het bepalen van goede transities. Wanneer de video sprites op dezelfde positie gemapt worden, kunnen dezelfde kleurmetrieken als video textures gebruikt worden.
Bij de per pixel vergelijkingen treedt wel een nieuw voorval op. Wat als nu de pixel van de ene video sprite bij de voorgrond is en de bij de andere sprite niet? Een mogelijkheid hiervoor is de grootste penalty te tellen. Maar dit fenomeen is niet zo opvallend voor het oog, en kan zelfs in sommige gevallen genegeerd worden. Grote kleurverschillen tussen 2 pixels is erger en zal sneller storende artifacten opleveren.

6.4.2 Oppervlakte

2 sprites kunnen op elkaar volgen als hun grootte nagenoeg gelijk is. De oppervlakte van een sprite kan bekomen worden door diens pixels te tellen. Maar oppervlakte op zich vertelt weinig. Het bijhouden en vergelijken van een algemene vorm van een sprite kan erg complex worden, waardoor oppervlakte vaak gecombineerd wordt met andere metrieken zoals de L_2 afstand. De oppervlakte van de sprites wordt dan eerder gebruikt om al snel de slechtste transities te verwijderen.

6.4.3 Velocity

Een andere eigenschap die erg belangrijk is om te bepalen of 2 video sprite op elkaar kunnen volgen, is hun bewegingsvector (velocity). Voor beide sprites mag deze vector weinig afwijken, anders worden schokkende en onnatuurlijke animaties gecreëerd.
De bewegingsvector bestaat eigenlijk uit 2 delen: de richting en de grote. Beide zijn van belang tijdens het zoeken naar goede transities. Een te groot verschil in richting levert een animatie waar de sprite zigzag loopt. Wanneer de sprite naar een bepaalde positie moet bewegen, is meestal het korste pad wenselijk. Hiervoor mag de richting dus niet te fel veranderen. Ook de grootte van de bewegingsvector is van belang bij bovenstaand voorbeeld. Wanneer deze voortdurend verschilt, zal de sprite met schokken naar zijn doelpositie bewegen. Ook dit is niet realistisch en moet dus vermeden worden.

6.5 Conclusie

In dit hoofdstuk werden video sprites geïntroduceerd. Na een definitie van video sprites, werden enkele belangrijke aspecten ervan besproken, zoals sprite segmentatie en perspectieve correctie. Video sprites hebben ook nieuwe eigenschappen t.o.v. video textures. De belangrijkste zijn de velocity en de oppervlakte.
Door deze nieuwe eigenschappen is ook een nieuwe metriek nodig om goede transities te bepalen. De L_2 norm alleen is onvoldoende. Om realistische animaties te verkrijgen, moet ook de sprite velocities vergeleken worden, aangezien deze de oriëntatie van de sprite definiëert. Meestal is ook het gebruik van de sprite oppervlakte aangewezen om enkel transities toe te laten tussen sprites met een gelijke grootte (hoewel dit met L_2 norm al deels verholpen wordt). Vaak worden ook de alpha-waarden van de sprites mee gebruikt tijdens het bepalen van de transities.

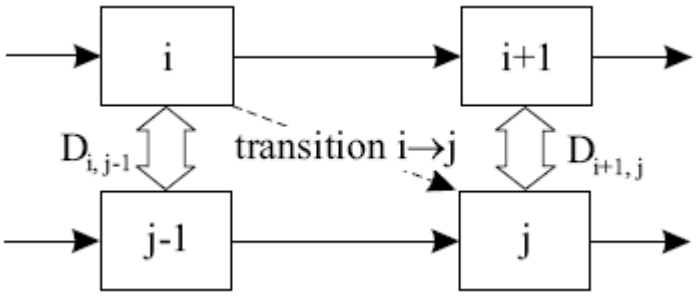
Hoofdstuk 7

Animaties met video sprites

Zoals eerder aangehaald biedt het gebruik van video sprites nieuwe animatiemogelijkheden. Eerst en vooral is de animator niet meer beperkt tot 1 texture. Het is nu immers mogelijk verschillende sprites binnen 1 frame te tonen, en deze met elkaar te laten interageren.
Ook bieden de eigenschappen, verbonden aan de sprites nieuwe, mogelijkheden. Vooral het gebruik van de sprite's velocity laat veel interessantere animaties toe. Zo is path-following mogelijk.

7.1 Transities bepalen

Voor video sprites zal de gewone L_2 afstand alleen falen, aangezien deze metriek geen rekening houdt met de positie en oriëntatie van de video sprites. Daarom moeten de sprites gemapt worden naar een centrale positie. Zo kunnen dan overeenkomstige paren gezocht worden.
Bij video sprites is het gebruik van de velocity cruciaal. Zowel de richting als de lengte van de velocity zijn van belang. Wanneer 2 sprites met verschillende richting op elkaar zouden volgen, geeft dit discontinuïteiten. Wanneer de lengte van de velocities te veel zouden verschillen, geeft dit schokkende animaties.



Figuur 7.1: Berekenen van een transitie tussen 2 sprites. Frame i en frame $j-1$ moeten gelijkaardig zijn, net als frame $i+1$ en frame j.

Om nu 2 sprites i en j met elkaar te vergelijken, wordt volgende formule gebruikt:

$$C_{i \to j} = \frac{1}{2} D_{i,j-1} + \frac{1}{2} D_{i+1,j},$$

met $D_{i,j}$ de kost tussen sprite i en sprite j. Een goede kostfunctie houdt rekening met kleur, oppervlakte en velocity.

7.2 Controle over video sprites

Nadat de mogelijke transities tussen sprite-paren bepaald is, kan hieruit een graaf worden opgesteld. De knopen stellen de sprites voor, en de edges zijn de transities. Een sequentie van sprites $S = \{s_1, ..., s_n\}$ stelt het pad van sprite s_1 tot s_n in de graaf voor. De kost van dit pad wordt bepaald door

$$C_s(S) = \sum_{i=1}^{n-1} C_{s_i \to s_{i+1}}.$$

Om nu een animatie toe te voegen, wordt een tweede kostfunctie gebruikt. Deze kostfunctie C_c bepaald de animatie (zie volgende sectie). De totale kostfunctie is dan

$$C(S) = C_s(S) + C_c(S).$$

Om nu de gewenste animatie te bekomen, moet een sequentie S gevonden worden die deze kostfunctie minimaliseert:

$$S = arg\, min_S\, C(S).$$

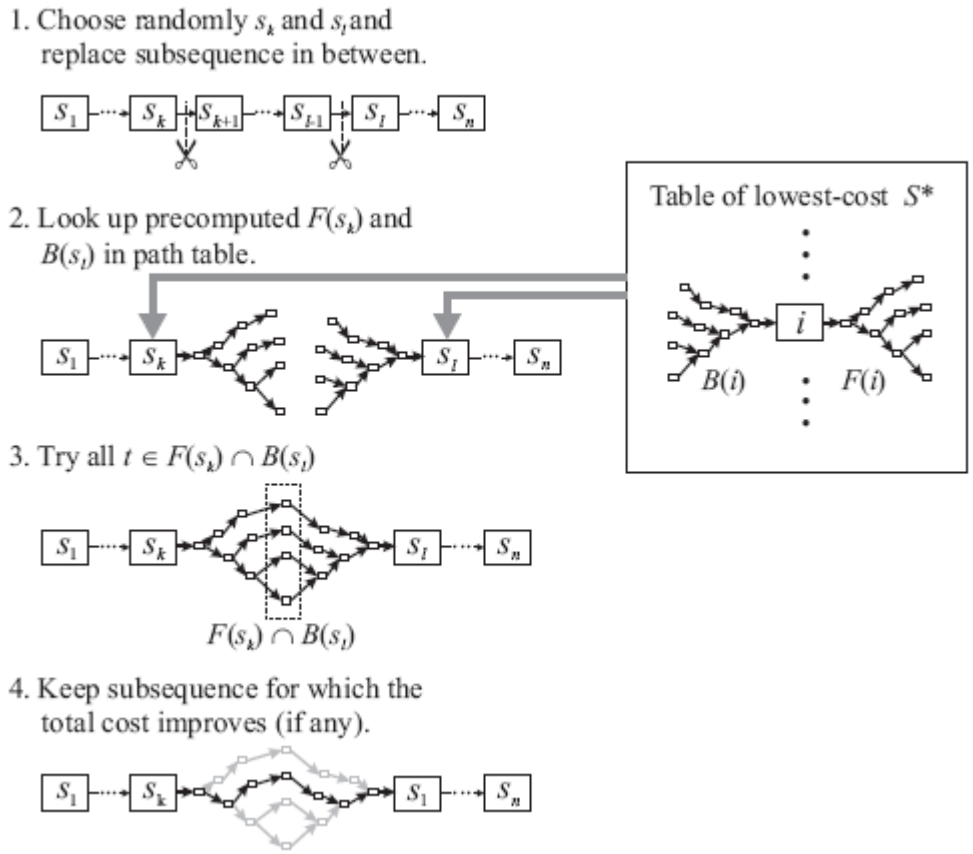
7.2.1 Beam search

Om nu de beste sequentie S te zoeken die de kostfunctie minimaliseert, kan beam search gebruikt worden. Vertrekkend van een gegeven sprite, worden een vast aantal minimale sequenties bijgehouden. Voor elke sequentie wordt de reeks sprites en tevens diens kost bijgehouden, alsook de positie van de huidige sprite. Per iteratie wordt een nieuwe set van sequenties S' bepaald, die voor de sequentie $S = \{s_1, ..., s_n\} \in S$ alle mogelijke volgende transities meetelt. Het is onmogelijk de hele boom bij te houden, vandaar dat deze gesnoeid moet worden.
Het grootste nadeel van beam search is dat beam search niet in de toekomst kijkt. Hierdoor kunnen mogelijke obstakels niet vermeden worden. Een mindere transitie, die mogelijk een globaal minimum voor $C(S)$ oplevert, zal genegeerd worden omdat de transitie geen lokaal minimum is.

7.2.2 Hill-climbing optimalisatie

Het probleem van beam search is dat deze methode sequentiëel werkt. Daardoor is het onmogelijk om terug te keren en fouten te verbeteren. Hiervoor biedt de hill-climbing optimalisatie een oplossing. Hill-climbing probeert de sequentie in zijn geheel te minimaliseren. Het begint met een random sequentie, en deze wordt iteratief vervangen door een andere sequentie met een lagere kost.

De meest voor de hand liggende oplossing is om 1 frame te vervangen. Voor een sequentie wordt dan frame s_k vervangen door frame t, zodat de nieuwe sequentie $s_1, ..., s_{k-1}, t, s_{k+1}, ..., s_n$ een nieuw minimum oplevert. Dit levert 2 nieuwe transities, $s_{k-1} \rightarrow t$ en $t \rightarrow s_{k-1}$. Helaas geven deze transities zelden een lagere kost.
Beter is om een deelsequentie te vervangen. Om nu een sequentie van s_k tot s_l te vervangen, moet eerst een heuristieke kost $\hat{C}$ bepaald worden. Een mogelijkheid is $\hat{C}$ gelijk te stellen aan C_s. Dan kan Dijkstra's algoritme gebruikt worden om een pad de vinden van s_k tot s_l, $S^*(s_k \rightarrow s_l)$. Helaas zal deze sequentie $\hat{C}$ niet veranderen. Dit komt omdat beam search zijn sequenties berekend m.b.v. het algoritme van Dijkstra.



Figuur 7.2: Hill-climbing optimalizatie voor beam search.

Beter is een frame t te zoeken, die $S^*(s_k \rightarrow t)$ en $S^*(t \rightarrow s_l)$ minimaliseert. Maar welke frames t zijn geschikt? Dijkstra's algoritme kan gebruikt worden om een reeks van N frames t te zoeken zodat de kost van $S^*(i \rightarrow t)$ zo laag mogelijk is. Deze t's vormen de set $F(i)$. Gelijkaardig kan een set $B(i)$ gezocht worden, die voor $S^*(t \rightarrow s_i)$ de laagste kosten aangeeft. Dan kunnen alle t's getest worden of ze in de intersectie $F(s_k) \cap B(s_l)$ zitten. Wanneer nu een frame t gevonden kan worden die de globale kost vermindert, wordt de oude sequentie vervangen door de nieuwe. Dit kan dan herhaald worden voor alle mogelijke frames s_k en s_l.
Het vervangen van de begin- of eindsequentie is zelfs nog eenvoudiger. Voor het begin kan sequentie $s_1, ..., s_l$ vervangen worden door $S^*(t \rightarrow s_l)$, en kan elke $t \in B(s_l)$ getest worden. Het vervangen van het einde gebeurt analoog.

Nu kan bijvoorbeeld eerst een random sequentie bekomen worden door beam search, en vervolgens kunnen dan een reeks random k's en l'en gekozen worden totdat er geen vervangingen meer mogelijk zijn die C minimaliseert.

7.3 Animatie toevoegen

In deze sectie wordt de controle kostfunctie opgebouwd. Deze kostfunctie bepaalt de animatie, en kan uit verschillende componenten bestaan. Elke component heeft een specifieke eigenschap, en de animator kan tussen de verschillende componenten kiezen om de gewenste animatie te maken. De kostfunctie wordt per tijdstip berekend, en bestaat uit een drietal (p, v, f), met p de locatie van de sprite, v de velocity van de sprite en f de input frame vanwaar de sprite gekopiëerd is.

De kostfunctie c op een bepaald tijdstip i, c_i, hangt af van de toestand van een of meer sprites in de animatie:

$$c_i = f\left(\left(p_1, v_1, f_1\right)_i, \left(p_2, v_2, f_2\right)_i, ...\right).$$

De totale kostfunctie C_c is de som van alle constraints en tijdstippen. Nu volgt een opsomming van enkele constraints.

7.3.1 Locatie constraint

Deze constraint houdt de sprite op een bepaalde positie:

$$c_i\left(p\right) = \gamma\left(p - p_{target}\right)^2,$$

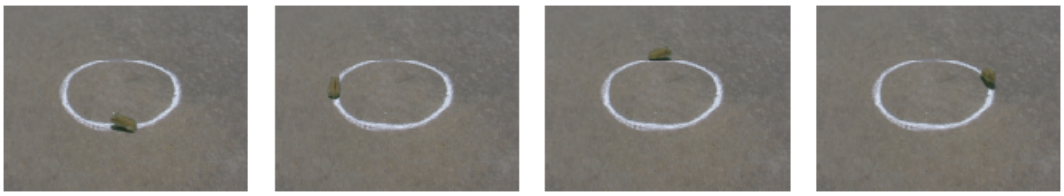
met p_{target} de positie waar de sprite gehouden wordt, en γ is de coëfficiënt die aangeeft hoe fel deze constraint de animatie beïnvloedt. Door p_{target} gelijk te stellen aan de positie van een andere sprites, kunnen formaties gevormd worden. Zo kan bijvoorbeeld een sprite een andere sprite achtervolgen.

7.3.2 Pad constraint

Hoewel positie constraints gebruikt kunnen worden om een sprite van de ene naar de andere positie te laten gaan, is het volgen van een pad moeilijk ermee te realiseren. Hiervoor moet immers de velocity van de sprite gebruikt worden, anders zal dit geen vloeiende animaties geven. De oriëntatie van de sprite mag tussen 2 opeenvolgende frames immers niet te fel veranderen, anders is het bekomen resultaat onrealistisch. Verder zal een te lage γ de sprite niet in een rechte lijn laten gaan (een pad wordt bepaald door lijnstukken), en een hoge γ zal de sprite er te snel naar toe forceren.
De pad constraint bestaat uit 2 delen: een eerste deel houdt rekening met de euclidische afstand van de sprite tot het huidige lijnsegment, en het tweede deel geeft de hoek weer tussen de velocity van de sprite en de richting van het lijnsegment:

$$c_i\left(p, v\right) = \delta\left|\angle v - \angle v_{target}\right| + \phi\left(\left(p - p_{target}\right) \cdot \perp v_{target}\right)^2,$$

met p_{target} en v_{target} respectievelijk de locatie en richting van het huidige lijnsegment. $\angle$ berekend de hoek van een vector en δ en ϕ zijn gewichten voor de constraint.
Deze constraint houdt geen rekening met de tijd. De sequenties van de sprites die de kostfunctie minimaliseren, wordt gekozen, onafhankelijk van de lengte van de animatie. Dit kan wel bereikt worden door een constraint toe te voegen op het aantal gebruikte sprites, zodat dit naar een opgegeven waarde streeft. Dit kan wel ten koste van de kwaliteit gaan.
Ook wordt enkel de richting van de velocity gebruikt, en niet de grootte. Wanneer ook de grootte van de velocity gebruikt wordt, worden egalere bewegingen bekomen. Hiervoor moeten wel voldoende mogelijke transities zijn, en daarom is het gebruik van de velocity grootte niet altijd mogelijk [SE02].



Figuur 7.3: Animatie met pad constraint. De hamster moest een cirkelvormig pad volgen.

7.3.3 Anti-collision constraint

Voor een animatie met verschillende sprites is het meestal wenselijk dat deze sprites niet te dicht bij elkaar komen. Zo kunnen botsingen vermeden worden, door de afstanden tussen elk paar sprites te berekenen en een extra penalty toe te kennen wanneer deze afstand te klein wordt:

$$c_i\left(p_1, p_2\right) = \mu\ max\left(d_{min} - \left\|p_1 - p_2\right\|_2, 0\right),$$

met d_{min} de minimale afstand die tussen 2 sprites moet gehouden worden. Een mogelijke uitbreiding is de error sneller te laten verhogen afhankelijk van de afstand, zoals hyperbolics, maar door de constante μ een hogere waarde te geven, kan hetzelfde effect bekomen worden. Deze constraint is eigenlijk een speciaal geval van een locatie constraint, waar p_{target} de positie is van de andere sprite, en de constaint geïnverteerd werd.

Figuur 7.4: Animatie met locatie en anti-collision constraints. De vliegen moesten naar een specifieke locatie gaan zonder tegen elkaar de botsen. Deze animatie duurde een dag om te berekenen [SE02].

7.3.4 Frame constraint

Soms wil de animator de sprite in een bepaalde pose houden. Dit is mogelijk door een groep G van sprites te bepalen, die alle mogelijke sprite bevat die aan die gewenste pose

voldoen. Zo wordt een kost λ toegevoegd wanneer een sprite niet in deze groep zit:

$$c_i(f) = 0 \;\; als \; f \in G,$$
$$c_i(f) = \lambda \qquad anders$$

Schödl en Essa [SE02] gebruikten deze constraint om een hamster voor enkele seconden op 2 poten te laten staan.
Het nadeel is wel dat de animator handmatig alle toegestane frames moet aanduiden. Een automatisch selectiealgoritme hiervoor is op dit nog niet beschikbaar.

Figuur 7.5: Animatie met frame constraint. De animator duidt een serie sprites aan die de pose van de gewenste animatie bevatten.

Interactive toepassing

Om video sprites in computergames te kunnen gebruiken, moeten de transities real-time bepaald kunnen worden. Hierdoor is het onmogelijk per transitie toekomstige kosten mee te rekenen. D.w.z. dat voor alle bewegingen deze reeds gekend moeten zijn.

Wanneer de pad constraint gebruikt wordt en enkel de velocity hierbinnen genomen wordt, zal de sprite enkel in een algemene richting bewegen. Dit komt overeen met een pad naar oneindig in deze richting. Vervolgens kan Q-Learning [KLM96] gebruikt worden om iteratief de kosten van alle richtingen te berekenen:

$$F_{i,j}(v) = \delta \left| \angle v - \angle v_{target} \right| + \alpha \; min_k \; F_{j,k}.$$

Initiëel wordt $F_{i,j}$ gelijk gesteld aan $C_{i \rightarrow j}$, dit is de smoothness component. Wel moet $0 < \alpha < 1$, anders zal de vergelijking niet convergeren.

Bovenstaande formule berekent bewegingen in algemene richtingen. Door dit te doen voor bijvoorbeeld de 8 kompas richtingen, kan de sprite de muispointer volgen. Er treedt echter een probleem op wanneer er van richting moet veranderd worden. Daarom moeten er voor elke richting mogelijke overgangen bestaan naar alle andere richtingen. Wanneer er een verandering in richting is, moet de velocity van de sprite voor enkele frames niet worden meegeteld, om daarna de andere richting te gebruiken. Hierdoor zal de sprite voor enkele frames N vrij kunnen bewegen, zonder een beperking van richting. Dit is mogelijk door

$$F^n_{i,j} = T_{i,j} + \alpha \; min_k \; F^{n-1}_{j,k},$$

waar $n = 1 \ldots N$ en $F^0 = F$. Tijdens het veranderen van richting wordt F^N gebruikt voor de eerste transitie, F^{N-1} voor de tweede,... Wel mag N niet te groot worden, anders zijn

interactive resultaten niet meer mogelijk. Maar door N te laag te kiezen, zal de sprite niet genoeg tijd hebben om van richting te kunnen veranderen zonder visuele artifacten.

Figuur 7.6: Een animatie waar de gebruiker de bewegingen van de vis interactief kan veranderen. De muis beweegt voortdurend in de richting van de muispointer (rode stip). Er werden voor de 8 kompas richtingen toekomstige kosten berekend, en tijdens het veranderen van richting werden 5 frames gebruikt waar de velocity niet meetelde.

7.4 Uitbreidingen

In deze sectie worden enkele mogelijke uitbreidingen op video sprites besproken. De meeste zijn een eigen idee.

7.4.1 Animatie constraints

Hoewel de eerder besproken animatie constraints al veel animaties toelieten, zijn er enkele specifieke gevallen waar deze niet voldoende zijn. Daarom worden enkele uitbreidingen toegevoegd.

Area constraint

De animator duidt een gebied aan als plaats waar de sprite binnen moet blijven. Vervolgens wordt voor elke transitie getest of de nieuwe positie binnen de area A valt. Indien dit niet het geval is, wordt er een extra penalty bij gerekend:

$$c_i(p) = 0 \quad als\ p \in A,$$
$$c_i(p) = \lambda \qquad anders.$$

Hoewel deze constraint ook gesimuleerd kan worden met locatie constraints, zijn er echter veel locatie constraints nodig. Daarboven is de locatie constraint te streng genoeg voor verschillen in afstanden. De sprites probeert altijd het middenpunt van de locatie constraint te bereiken. Met de area constraint hoeft de animator slechts met een brush een bepaald gebied aan te duiden waar de sprite binnen moet blijven. Dit is eenvoudiger en gemakkelijker.

Velocity constraint

Soms kan het interessant zijn de sprite in een algemene richting te laten bewegen, zonder echte posities te moeten definiëren. Dit werd gedaan bij de interactieve toepassing. Een velocity constraint berekent de hoek tussen de velocity van de sprite en een opgegeven vector:

$$c_i(v) = \delta \left| \angle v - \angle v_{target} \right|.$$

Deze formule is een deel van de pad constraint, namelijk het deel waar de sprite velocity werd getest. Wanneer in die formule ϕ op nul wordt gezet, geeft dit de velocity constraint.

Tijd toevoegen

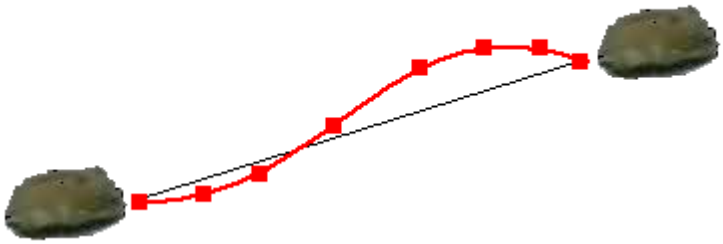
Alle eerder gegeven animatie constraint hielden geen rekening met de tijd. Dit werd gedaan om meer goede transities over te houden. Daarom waren korte input-videosequenties al voldoende om deze animaties te maken. Maar voor sommige toepassingen is tijd cruciaal. Voorbeelden zijn films en games, waar de sprite op vaste tijdstippen op bepaalde plaatsen moet zijn.

Een eerste mogelijkheid is om de lengte van de animatie te gebruiken. De kostfuncties worden dan uitgebreid tot een kwartet: (p, v, t, f) met t de tijd. De animator geeft hierbij het aantal frames aan hoelang de animatie moet lopen. Voor bijvoorbeeld een animatie waar een hamster een lijn moet afleggen binnen 3 seconden, kan de animator opgeven dat er 75 frames gebruikt moeten worden voor de animatie (25 fps). Maar het probleem is hiermee niet opgelost. De hamster kan immers het traject afleggen in 50 frames, en voor 25 frames ronddraaien. Dit is natuurlijk niet gewenst.

Daarom is een meer dynamische aanpak nodig. Het lijnsegment moet onderverdeeld worden in het aantal stappen. In bovenstaand voorbeeld wil dit zeggen dat er 75 intervallen zijn, 1 per frame. Per iteratie wordt het volgende interval genomen. Wanneer de intervallen gelijk verdeeld zijn, zal de sprite met ongeveer constante snelheid bewegen omdat de velocity grootte constant is. De locatie wordt per interval geüpdate. Hierdoor zal de sprite ook niet geforceerd worden het traject zo snel mogelijk af te leggen. Dit zal nu egaler gebeuren. Ook is de animator niet meer beperkt tot het gebruiken van lijnstukken voor het pad. Bezier curves en Catmull-Rom curves [Par05] kunnen ook gevolgd worden.

Diepte toevoegen

Om sprite in 3D-computer games te kunnen gebruiken, is diepte nodig. Dit kan toegevoegd worden door de positie in elke constraint uit te breiden met een z-waarde. Maar het probleem is echter het bepalen van de diepte-waardes van de input-video. Deze is niet gekend, maar kan afgeschat worden door de oppervlakte van de sprite te gebruiken voor er een perspectieve correctie werd uitgevoerd. Wanneer de diepte van de sprites gekend is, kan deze gerenderd worden in 3D-scènes m.b.v. billboarding. Dit kan uitermate interessant zijn in computergames, waar het modelleren en animeren van dieren en andere complexe objecten erg moeilijk is.



Figuur 7.7: I.p.v. een rechte weg kan de hamster een Catmull-Rom curve volgen die per tijdsinterval een nieuwe locatie en velocity heeft.

Input data vergroten

Vaak moet een object gedurende enkele uren gefilmd worden vooraleer er voldoende informatie is om genoeg goede transities te maken. Anders zijn bepaalde bewegingen niet mogelijk te reconstrueren, omdat deze niet in de input-data zaten. Toch is het filmen van een object gedurende zo een lange tijd niet evident, en wordt veel tijd verloren door het verwijderen van slechte data. Hoe meer data aanwezig is, hoe meer data verwerkt moet worden.

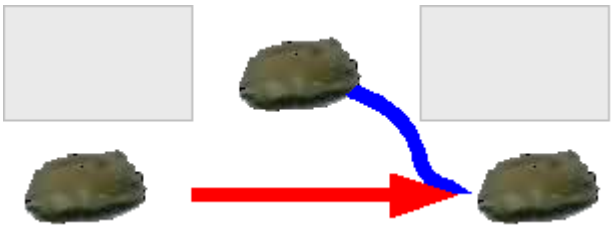
Vooraleerst kunnen de sprites gespiegeld worden [SE02]. Dit is mogelijk wanneer de sprite symmetrisch zijn. In het voorbeeld van de hamster werden alle sprites horizontaal gespiegeld.

Een andere manier om de hoeveelheid data te verminderen, is om warping-algoritmen te gebruiken [SE02]. Wanneer het object van oriëntatie verandert, kan dan nagegaan worden of een via warping voldoende transities mogelijk worden.
Het nadeel is dat warping algoritmen erg traag zijn, en dus moet op een snelle manier beslist kunnen worden of warping de moeite is of niet.

Door verschillende camera's te gebruiken tijdens het filmen van een object, wordt meer data bekomen vanuit verschillende standpunten. Hierdoor is sneller voldoende informatie over het object beschikbaar. Deze werkwijze bemoeilijkt wel de perspectieve correctie, die nu voor elke camera gedaan moet worden. Hierbij moet gezorgd worden dat voor elke camera een gelijkaardige projectie gedaan wordt, anders kan tijdens het bepalen van de transities niet van camera geswitcht worden.

Backward transities

Een andere leuke uitbreiding zijn het toevoegen van achterwaartse transities. Hoewel voor vele objecten de transities niet in de natuur voorkomen, kunnen deze toch interessant zijn. Zo kan men bijvoorbeeld een muis achterwaarts laten lopen. Verder kunnen voorwaartse en achterwaarts transities gecombineerd worden, zodat de muis een parkeerbeweging van een auto kan nadoen.



Figuur 7.8: Een animatie met een parkeerbeweging van een hamster. De rode pijl duidt op voorwaartse transities, en voor de blauwe beweging worden achterwaartse transities gebruikt.

Motion graphs

Tijdens het maken van verschillende animaties viel op dat, wanneer een sprite een complexe beweging maakt (vb. draaien), haast altijd de originele volgende frames gebruikt werden. Er waren geen nieuwe overgangen binnen deze bewegingen. Dit geeft de aanleiding tot motion graphs [KGP02]. Deze bewegingen kunnen dan als 1 geheel gezien worden. Hierdoor moet er niet meer per frame op volgende frames getest worden, maar worden enkele frames in 1 keer gekozen. Dit is niet noodzakelijk een beperking op de mogelijkheden, aangezien deze frames zowat altijd gekozen werden.

Dit kan dan uitgebreid worden naar een hoger niveau. Zo kunnen de grafen acties voorstellen, zoals bijvoorbeeld een stap van een mens. Hierdoor is het misschien mogelijk sprite animation uit te breiden naar grotere dieren en mensen. Met de huidige methoden worden voor grotere dieren onvoldoende goede transitie gevonden [SE02]. Toch zijn motion graph al met succes toegepast op mensen door de motion capturing data te herschikken m.b.v. video texture-achtige algoritmen (zie sectie 5.5.5.). Er kan dus afgevraagd worden of motion graphs ook kunnen toegepast worden op video sprites. Verder onderzoek hiernaar is nodig.

Partiële aanpak

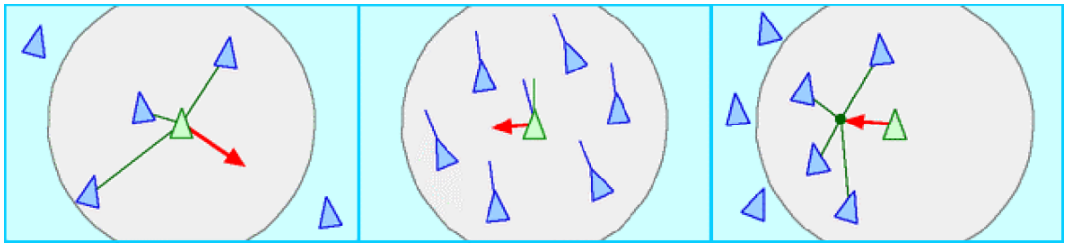
Net zoals video texture kunnen video sprites ook uitgebreid worden door een partiële aanpak. Maar hier moet extra aandacht aan geschonken worden. De sprite moet immers opgesplitst worden in enkele delen, en vervolgens moeten deze delen ook weer bij elkaar gezet worden. Hierdoor moeten er overeenkomsen tussen deze delen zijn. Ook hier kan een skeletstructuur gebruikt worden. Zo zouden de armen, benen en romp van een mens apart berekend kunnen worden. Maar het samenstellen van de animatie biedt ook weer nieuwe problemen. Zo moeten ook beperkingen op de transities gelegd worden om onnatuurlijke bewegingen te vermijden. Dit levert de animator extra werk, en kan het maar erg de vraag zijn of deze aanpak kan werken.

7.4.2 Toepassingen

Met de net aangegeven uitbreidingen zijn heel wat nieuwe toepassingen mogelijk. Enkele hiervan worden nu weergegeven.

Flocking

De velocity constraint is uitermate geschikt om flocking [Rey86] te bekomen. Flocking-gedrag wordt bekomen door 3 eenvoudige basisregels: seperation, alignment en cohesion. Seperation zorgt ervoor dat de sprites niet tegen elkaar komen, door de sprite weg te laten gaan van de dichtstbijzijnde andere sprites. Alignment laat alle sprite dezelfde richting uitgaan. Zo lijkt het alsof de groep in eenzelfde richting beweegt. Cohesion laat een sprite bewegen in de richting van de naburige sprites. Zo blijft de groep bij elkaar.



Figuur 7.9: De 3 basisregels voor flocking: seperation links, aligment rechts en cohesion rechts. Door voor elke behaviour de richtingsvector te berekenen, en daarna deze bij elkaar op te tellen, wordt de huidige flocking richting bekomen.

Elke behaviour heeft zijn eigen bewegingsvector. Deze vector bepaalt de richting die de sprite moet volgen om de behaviours te simuleren. Door de 3 bewegingsvectoren bij elkaar op te tellen, wordt een globale bewegingsvector bekomen. Deze vector geeft aan hoe de sprite op het huidige tijdstip moet bewegen. Elk tijdstip moet dit proces herhaald worden.

Flocking kan uitgebreid worden door avoidance. Dit zorgt ervoor dat de sprites niet tegen obstakels botsen. Anti-collision constraints zorgen voor dit gedrag.
Nog een andere uitbreiding is survival. Sprites bewegen naar voedsel bronnen, en lopen weg van andere sprites die hen kunnen opeten. Hiervoor moeten de sprites wel een honger-waarde hebben.

Keyboard controle

Zoals bij de interactieve toepassing kan de sprite ook bestuurd worden door een keyboard. Hierdoor kan de sprite de 8 kompas richtingen volgen. De werkwijze is volledig analoog met de eerder besproken interactieve toepassing.
Door de keyboard controle kan de sprite gebruikt worden in computergames, waar hij door de gebruiker bestuurd kan worden. Zo kan bijvoorbeeld een hamster bestuurd worden over een vlak. Omdat video-based animation werd gebruikt, is het realisme weer gegarandeerd. Dit realisme kan op dit moment door geen enkele game geëvenaard worden.

7.5 Slot

Nadat een goede manier om sprite transities te bepalen werd beschreven, werden enkele algoritmen die de controle van de transities bepalen besproken. Hieruit bleek dat beam search alleen erg beperkt is, omdat foute transities niet ongedaan gemaakt konden worden. Dit kon opgelost worden door de hill-climbing optimalisatie.
Vervolgens werd een methode gegeven om animatie toe te voegen. Verschillende animatie constraints werden besproken om zo de gewenste animatie te bekomen.
Verder werden ook enkele uitbreidingen aangehaald. Vooral de controle over tijd is erg belangrijk om de animatiemogelijkheden van video sprites te vergroten. Tot slot werden ook enkele toepassingen aangehaald, zoals flocking en interactieve besturing van de sprites.

Hoofdstuk 8

Implementatie

In dit hoofdstuk zal mijn implementatie besproken worden, alsmede bekomen resultaten. Deze zullen tevens vergeleken worden met de resultaten uit de desbetreffende papers om een beter beeld te geven. Ook worden mogelijke uitbreidingen en verbeteringen aangehaald.

Ik heb een stage ("Opstellen van een gebruiksvriendelijke video library") kunnen volgen als voorbereidend werk op deze thesis. Hier maakte ik kennis met verschillende principes die belangrijk zijn bij het bewerken van videosequenties, zoals het extracten van frames uit een videofile, background subtraction, graph cut, enz... Dit gaf mij een goede basis om aan deze thesis te beginnen.

8.1 Analyse

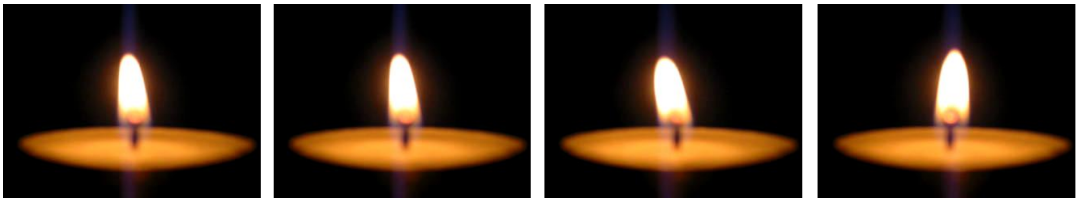
Vooraleerst moeten de frames uit het videofragment gehaald worden. Dit gebeurt m.b.v. MPlayer[1]. Vervolgens kunnen de goede transities tussen de frames gezocht worden.

8.1.1 Vergelijking van verschillende metrieken

Zoals aangegeven in eerdere hoofdstukken is het kiezen van een goede metriek cruciaal, anders worden geen goede transities gevonden. Dit gebeurt met de L_2 norm, omdat deze met succes werd gebruikt door anderen [SSSE00], [SE01] en [SE02], alsook omdat deze goede resultaten geeft. Ik heb de L_2 norm experimenteel vergeleken met de L_1 norm, de L_3 norm (als vb. voor de L_p norm), de cross-correlation en de Haussdorf afstand.
Ik heb deze vergelijkingen zowel voor video textures als voor video sprites gedaan. In beide tabellen is de tweede kolom telkens een oorspronkelijke transitie, de derde kolom een goede transitie en die laatste kolom een slechte transitie. Verder zijn de resultaten genormaliseerd: 0 wil zeggen dat beide frames identiek zijn, 1 totaal verschillend.

[1] http://www.mplayerhq.hu/design7/news.html

Vergelijking voor video textures



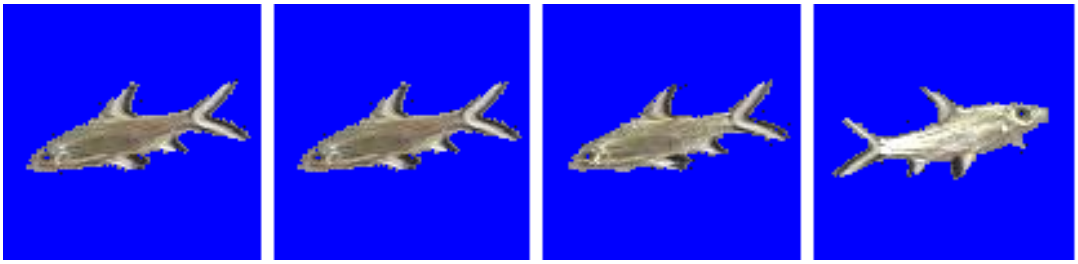
Figuur 8.1: 4 frames uit een video texture: links is de texture waarmee de andere textures vergeleken werden, linksmidden is de texture die origineel op de linker texture volgde, rechtsmidden is een texture die op de linker texture kan volgen, en rechts een texture die niet na de linker texture mag gespeeld worden.

Metriek	**frame 1 en 2**	**frame 1 en 3**	**frame 1 en 4**
gemiddelde	0	0,005	0,0055
l1	0,0022	0,0167	0,023
l2	0,0075	0,0699	0,0939
l3	0,0126	0,0298	0,0833
cross	0,0001	0,021	0,039
hauss	0	0,029	0,0132

Tabel 8.1: Vergelijking van de verschillende metrieken voor video textures, met de textures uit bovenstaande tekening.

Eerst en vooral is duidelijk te zien dat het verschil van de gemiddelde van de 2 frames een slecht resultaat geeft, aangezien deze metriek het verschil tussen een goede en een slechte transitie niet kan bepalen.
De L_1 norm levert betere resultaten, maar het verschil tussen een goede en een slechte transitie is te klein, zodat het bepalen van een goede threshold erg moeilijk is. Daartegenover staat dat de L_2 norm hiervan minder last heeft, hoewel hier het verschil eerder klein is. Verder is ook te zien dat de L_3 norm ook bruikbaar is. Maar wanneer ik de L_3 norm gebruikte, gaf dit toch een slecht resultaat. Te veel transities werden toegestaan waardoor de kwaliteit achteruit liep. De L_3 norm verwijderde dus onvoldoende minder goede transities.
De cross-correlation heeft enigszins hetzelfde probleem als de L_1 norm. Tot slot is duidelijk te zien dat de Haussdorf afstand zeer slechte resultaten geeft. Dit komt omdat kleine pixelverschillen te fel doorwegen.

Vergelijking voor video sprites



Figuur 8.2: 4 video sprite: links is de sprite waarmee de andere sprites vergeleken werden, linksmidden is de sprite die origineel op de linker sprite volgde, rechtsmidden is een sprite die op de linker sprite kan volgen, en rechts is een sprite die niet na de linker sprite mag gespeeld worden.

Metriek	**frame 1 en 2**	**frame 1 en 3**	**frame 1 en 4**
gemiddelde	0	0,0036	0,0181
l1	0,0105	0,163	0,4299
l2	0,00955	0,0867	0,1827
l3	0,0369	0,1397	0,2501
cross	0,0016	0,0796	0,2178
hauss	0	0	0

Tabel 8.2: Vergelijking van de verschillende metrieken voor video sprites, met de sprites uit bovenstaande tekening.

In deze tabel is te zien dat de verschillende metrieken voor video textures en video sprites gelijke resultaten geeft. Wel merkbaar is dat in deze tabel alle metrieken, op de Haussdorf afstand na, betere resultaten geven. Toch is ook hier de L_2 norm het meest geschikt.
Een andere belangrijke metriek die bij video sprites zeker moet meegerekend worden, is het verschil tussen de hoeken van de velocity's van de sprites. Deze geeft immers weer hoe fel de sprites van richting veranderen tijdens de animatie. Wanneer dit verschil te groot is, zal de sprite geen realistische voorwaartse bewegingen kunnen maken.

8.1.2 Andere kleurenruimte

De RGB kleurenruimte is erg verschillend met de manier waarop het menselijk oog kleur waarneemt. Andere kleurenruimten werden daarom ontworpen. Ik heb als voorbeeld van zo een kleurruimte de CIE-XYZ kleurenruimte bestudeerd. Toch waren de resultaten niet zoals verwacht.

Het voorbeeld van de klok faalde volledig. Er werden geen goede transities gevonden. Dit is echter niet verwonderlijk, omdat gedurende de videosequentie de belichting voortdurend veranderde. De videosequentie was onvoldoende lang om goede transities te vinden.

Maar wanneer het voorbeeld van de vlam gebruikt werd, werden nog steeds onvoldoende goede transities gevonden, hoewel deze er wel waren. Daarom werden vaak dezelfde stukken herhaald. Een mogelijke verklaring is de gebruikte metriek, de L_2 norm die minder geschikt lijkt om transities te bepalen in de CIE-XYZ kleurenruimte. Toch is verder onderzoek nodig om definitieve conclusies te trekken.

8.1.3 Post processing

Soms viel op dat resultaten bekomen met video sprite animaties niet visueel voldoende waren. Vaak bewoog de sprite licht schokkend. Dit kwam door de manier waarop de sprite velocity werd berekend. Hier werden soms foute resultaten bekomen, waardoor de kwaliteit van de hele animatie achteruit liep.
Dit probleem kan opgelost worden door voor de mindere transities post processing technieken te gebruiken. Een voorbeeld is blending. Hierbij worden de volgende en de vorige sprite samen met de huidige sprite geblend, zodat een mooie egale beweging wordt bekomen. Het nadeel van blending zijn de blurs, waardoor blending op de hele videosequentie moet toegepast worden om visueel mooie resultaten te bekomen.

8.1.4 Conclusie

Zoals uit de vergelijkingen is gebleken, is de L_2 norm het meest geschikt als metriek om goede transities te bepalen. Voor video sprites is het wel aangeraden ook het verschil in sprite velocity mee te tellen.
Verder werd aangetoond dat de kwaliteit van video sprites kan verbeterd worden door blenden te gebruiken. Hierdoor worden schokkende bewegingen bestreden. Blending kan ook bij video textures gebruikt worden, maar dit is meestal niet nodig.

8.2 Video textures

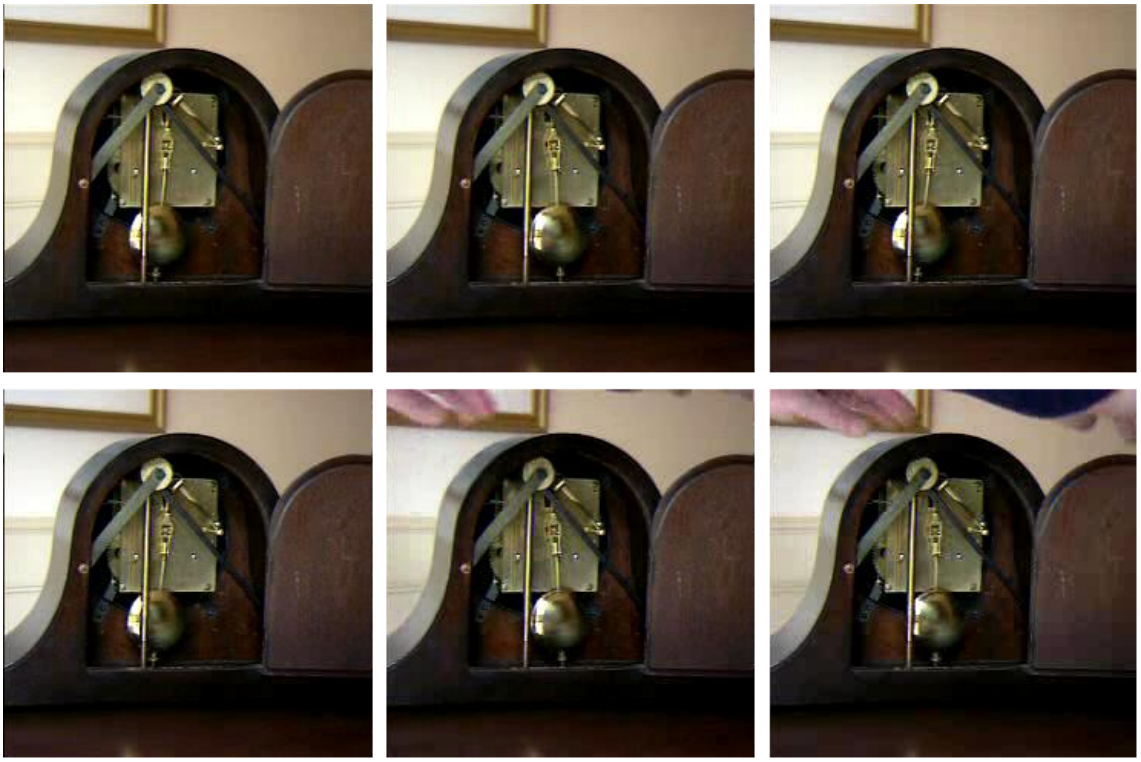
Uit bovenstaande vergelijking bleek dat de L_2 norm meest geschikt is om de transities voor video textures te bepalen. In deze sectie worden vervolgens synthese-algoritmen besproken die deze transities gebruiken.

8.2.1 Origineel video texture algoritme

Analyse

Nadat een goede metriek gekozen is om de transities te bepalen, kan deze metriek nu gebruikt worden. Voor elk paar frames wordt de kost berekend. Zoals aangetoond in sectie 5.2, is het beter meerdere frames in deze berekening te betrekken. Anders wordt de beweging waar de video texture op dat moment is zit, verstoord. Ik heb verschillende venstergroottes getest, telkens met binomiale gewichten. Een groter venster geeft betere transities, maar zal het aantal transities drastisch verminderen. Voor korte videofiles is een grootte van 1 al voldoende. Voor langere files met veel gelijkaardige frames mag de venstergrootte hoger gekozen worden. Anders zijn er te veel mogelijke transities en kan de globale beweging van de video texture verstoord worden.

Ook berekende ik hierna voor elke transitie diens minimale volgende kost, en liet ik deze meetellen. Hierdoor werden dead-ends enigzins vermeden. In een voorbeeld met een klok waar de slinger in het origineel videofragment tot stilstand kwam, wordt nu telkens teruggesprongen in de tijd, zodat het lijkt alsof de klok oneindig blijft lopen.



Figuur 8.3: Boven zijn 3 frames uit een animatie afgebeeld waar voor elke transitie telkens de minimale volgende kost is bijgeteld. Onder is een animatie gegeven waar dit niet is meegeteld. Deze animatie loopt vast wanneer de handen van de persoon in beeld komen, en er zijn dan geen goede transities meer die de animatie terug laat lopen.

Na het bepalen van de kost voor elke transitie, worden de overeenkomstige kansen berekend. Deze kansen gaan immers bepalen of de gevonden transitie goed is, dan wel verworpen moet worden. Voor formules wordt terug verwezen sectie 5.2. Daarna moet het aantal transities gesnoeid worden; de slechte transities moeten verworpen worden. Hiervoor gebruikte ik zowel een threshold van (0.75) van de originele volgende frame, alsook werd voor een interval van 5 frames (5 voor de huidige doel frame + 5 erna) enkel die transitie als deze in dit interval minimaal was.
Tot slot werden alle kosten genormaliseerd, en werden alle frames met hun transities opgeslagen in een graaf. Deze graaf kan dan weggeschreven worden in een xml-file, zodat alle transities niet opnieuw berekend moeten worden wanneer hetzelfde videofragment opnieuw gebruikt zou worden.

Synthese

Met bovenstaand berekende graaf kunnen dan oneindige videosequenties geproduceerd worden. Ik heb de frames laten afspelen via een random getal. Dit getal werd vergeleken

met de genormaliseerde som van alle mogelijke transities van de huidige frame. Hierdoor werd dan de volgende frame bepaald. Enkele resultaten zijn te zien in onderstaande tekeningen.

Figuur 8.4: 3 opeenvolgende frames van een video texture. De eerste transitie is een originele, de tweede is een nieuwe. De kwaliteit van beide transities is gelijk. Resultaat bekomen met venstergrootte 7 en voorkomen van dead ends.

8.2.2 Algoritme met symmetrische aanpak

Analyse

Ook weer vertrekkend van de videoframes, moeten nu die frames gevonden worden waar de symmetrie maximaal is. Voor elk frame wordt in zijn venster van grootte 2 de kost berekend d.m.v. de L_2 norm. In vergelijking met video textures moet de venstergrootte hier wel voldoende zijn, anders worden minder goede symmetriepunten toch aanvaard. Ook nu worden weer binomiale gewichten gebruikt. Vervolgens wordt getest of het frame binnen het interval met grootte 5 de minimale kost heeft. Indien dit het geval is, wordt dit frame een symmetriepunt. Nu moet wel nog een goede initiële kans voor dit symmetriepunt berekend worden. Dit wordt berekend met de formules uit sectie 5.3.

Figuur 8.5: In het midden is een symmetrisch punt. Links is het frame 5 posities terug in de tijd en rechts het frame 5 posities verder in de tijd. Beide frames zijn identiek.

Synthese

Het afspelen van de videosequentie gebeurt via random waarden. Telkens wanneer een symmetriepunt bereikt wordt, wordt diens kans vergeleken met een random waarde. Wan-

neer die kans groter is dan de random waarde, wordt de speelvolgorde omgekeerd. Telkens een symmetriepunt bereikt wordt, wordt diens kans geüpdated, zoals eerder weergegeven werd. Op deze manier kan dan een oneindige videosequentie gegenereerd worden.

8.2.3 Vergelijking video textures met symmetrische aanpak

Beide algoritmen werken tamelijk gelijkaardig, zodat ze voor vele voorbeelden kwalitatief haast evenwaardig zijn. Een eerste verschil, en tevens de motivatie van de symmetrische aanpak, is duidelijk te merken in scènes van een object met complexe geometrie, dat voortdurend, zij het traag, beweegt. Een goed voorbeeld hiervan is een plant. De stengels met hun bladeren zijn erg complex, en bewegen voortdurend door de wind. Video textures zullen voor dit voorbeeld falen: weinig goede transities kunnen gevonden worden. In mijn voorbeeld waren er slechts enkele nieuwe transities voor de hele videosequentie. Dit komt voornamelijk door de gebruikte metriek, de L_2 norm. Deze geeft slechte resultaten voor kleine bewegingen, omdat de pixelverschillen toch voldoende groot zijn door de complexe geometrie. Helaas is het gebuiken van andere metrieken, die rekening houden met zulke kleine bewegingen, niet zo voor de hand liggend.
Hieronder wordt dit geïllustreerd door een voorbeeld met een wesp die op een bloem beweegt. De gewone aanpak faalde, en vond enkel slechte transities waardoor de wesp begint te springen. De symmetrische aanpak vond verschillende symmetriepunten, waartussen verscheidene slechte waren. Toch kwam het geheel realistisch over, omdat er geen discontinuïteiten waren. Wel viel op dat de wesp soms achterwaarts bewoog, hetgeen niet in de inputsequentie was.

Figuur 8.6: 3 opeenvolgende frames van een video texture. Duidelijk is te zien dat de transitie tussen het tweede en het derde frame slecht is. De wesp maakt als het ware een sprong. De symmetrische aanpak onder heeft geen last van deze discontinuïteiten.

Video textures hebben ook last van kleine visuele artifacten, vooral door belichtingsverschillen tijdens de videosequentie. Dit is duidelijk merkbaar in het voorbeeld met de klok.

De nieuwe transities van de video textures hebben een klein verschil in belichting van de omgeving. De symmetrische aanpak heeft hier geen last van, aangezien deze geen nieuwe transities toegevoegd. Enkel de originele transities worden gebruikt, en deze transities hebben geen verschil in belichting van de omgeving.
Een ander feit is dat video textures voor lange animaties waar slechts een korte input-file beschikbaar voor was, betere resultaten zal geven. Dit komt omdat er veel meer transitiemogelijkheden zijn i.v.m. het aantal symmetrische punten. Voor het voorbeeld met de kaars, werden er 15 symmetriepunten gevonden. Met de video texture aanpak werd er per frame 2 tot 5 transities gevonden. Dit is gevoelig meer, en hierdoor waren herhalingen minder snel merkbaar. Een ander nadeel van video textures is wel de berekentijd tijdens de analyse fase, die vele malen hoger ligt dan de symmetrische aanpak. Dit is niet verwonderlijk, omdat alle frame paren vergeleken worden. De synthesefase gebeurt bij beiden vele malen sneller dan de afspeelsnelheid, zodat dit geen probleem is.

8.3 Video sprites

Door video textures uit te breiden naar video sprites, zijn verschillende nieuwe animaties mogelijk. Video textures waren beperkt tot de input frames, en konden enkel de volgorde van de frames veranderen. Video sprites bieden de mogelijkheid 1 of meerdere sprites over een achtergrond te laten bewegen, en zelfs ermee en met elkaar te interageren.

8.3.1 Analyse

Video sprites zorgen echter voor nieuwe problemen, zoals sprite extraction en de nood aan complexere zoekalgoritmen om de volgorde en positie van de sprites te bepalen. Dit zal nu verder toegelicht worden.

Sprite extraction

Vooraleer er met de sprites gewerkt kan worden, moeten zij eerst uit het videofragment gehaald worden. Veel gebruikte methoden om objecten uit een scène te halen zijn background subtraction en graph cuts. Ik heb beide geïmplementeerd, en zal deze ook kort met elkaar vergelijken.

Background subtraction: Aangezien video sprites meestal voor een blauw scherm (of voor een vaste, gekende achtergrond) opgenomen worden, zijn de meest eenvoudige background subtraction algoritmen al voldoende. Dynamische modellen zijn daarom niet nodig. Ik heb een methode geïmplementeerd waar op voorhand al de achtergrond aan wordt meegegeven, aangezien deze in mijn voorbeelden constant was. De sprites werden dan uit de frame gehaald, en bijgehouden voor verdere verwerking. Verder werden nog morfologische operaties toegepast om de kwaliteit van de sprites te verhogen.

Graph cut: Naast background subtraction heb ik ook graph cuts geïmplementeerd. Deze splitsten de videosequentie op een groepen van 6 frames, waar dan een 3-dimensionale cut werd op berekend. Hoewel de controle over voor- en achtergrond hier groot is, is background subtraction toch handiger, omdat het vele malen sneller is. Voor

elke reeks 6 frames moeten telkens voldoende voor- en achtergrondpixels aangeduid worden. Hiervoor moet zelfs opgepast worden dat discontinuïteiten tussen verschillende reeksen vermeden werden. Vergelijk de tijd van zo een cut (voor een reeks van 6 frames was dit al snel enkele seconden) met background subtraction, dat real-time kan uitgevoerd worden, was de graph cut methode toch eerder nadelig. Daarenboven waren de achtergronden van mijn voorbeeld files zeer eenvoudig, waardoor de kracht van graph cuts overbodig was.

Transities bepalen

Het berekenen van goede transities tussen video sprites is enigzins complexer dan tussen video textures. Eerst worden 2 op elkaar volgende sprites met elkaar vergeleken, om de velocity van 1 sprite te bepalen. Tot nu deed ik dit door het centrum van de ene sprite af te trekken met het centrum van diens opvolger. Dit geeft slechts een relatieve afschatting van de echte waarde, toch was deze goed genoeg voor gebruik bij de synthese.
Vervolgens werden alle sprites (eigenlijk waren ze tot dit moment enkel textures) gemapt naar een opgegeven centrum. Hierna kon dan de L_2 norm tussen sprite paren berekend worden. Ook hier werden alle mogelijke paren getest, en gebruikte ik dezelfde regels om dit aantal te snoeien als bij video textures. Na het snoeien werden de sprites in een graaf geplaatst, die ook weer als xml-file kon worden opgeslagen.

8.3.2 Synthese

In deze sectie zullen algoritmen besproken worden om met de net berekende graaf animaties te maken. Vertrekkende van random keuzes, over het toevoegen van controle m.b.v. beam search, tot het animeren door de verschillende constraints.

Random play

Met behulp van Morte Carlo technieken kan al een eerste vorm van animatie gemaakt worden. De graaf kan immers ook weer doorlopen worden door telkens random een transitie te kiezen, net zoals bij de video textures. Gebruik makend van de sprite velocity, kan dan de sprite al over een achtergrond bewegen. Tevens kunnen hierbij nog voorgrondobjecten worden toegevoegd, zodat het lijkt alsof de sprites al binnen hun natuurlijke omgeving bewegen. Natuurlijks is deze methode beperkt, maar toch interessant om aan te halen.

Figuur 8.7: Vis animaties door transities via Monte Carlo-integratie.

Controle toevoegen

Het random kiezen van een transitie levert geen controle over de animatie, en is dus onbruikbaar in combinatie met constraint kostfuncties. Er moet een beslissingsalgoritme zijn

die een keuze maakt voor elke mogelijke transitie, zodat de beste transitie (niet noodzakelijk lokaal, maar eerder globaal) genomen wordt.
Ik heb daarom beam search geïmplementeerd. Beam search is een path-finding algoritme, gebaseerd op dat van Dijkstra. Het grote verschil met het algoritme van Dijkstra is dat beam search niet de hele graaf doorloopt, maar slechts een deel ervan. De hele graaf per iteratie doorlopen zou te veel tijd kosten, vandaar dat er een manier aanwezig moest zijn om deze graaf te snoeien. Beam search houdt namelijk een queue bij van de beste N resultaten per iteratie, en update enkel deze queue. De grote van de queue heeft direct effect op de bekomen resultaten. Een kleine queue zal moeilijker foute keuzes ongedaan kunnen maken, omdat er weinig paden in deze queue zullen zijn die deze fouten niet hebben gemaakt. Wanneer bijvoorbeeld een huidige transitie een hoge waarde heeft, is de kans groot dat deze in de queue komt, ook al zal deze later slechte resultaten geven. Een huidige lage transitie zal minder snel opgenomen worden, ook al wordt dit later een globaal minimum.
Verder heb ik ook het algoritme van Dijkstra geïmplementeerd. Hierdoor zijn verdere optimalisaties mogelijk. Zo bestaat de mogelijkheid tot key framing: gegeven 2 sprites zal het algoritme van Dijkstra het kortste pad vinden, en dus visueel het best resultaat.

Animatie toevoegen

Om animaties toe te voegen wordt een tweede kostfunctie berekend. Deze wordt bepaald door de animatie constraints, en wordt samen bij de smoothness functie geteld tijdens het bepalen van een volgende transitie. Afhankelijk van de gewichten kan de ene kostfunctie meer meetellen dan de andere.

Locatie constraint: Voor deze constraint werd de Euclidische afstand van de sprite tot de opgegeven locatie berekend. Wanneer deze waarde groter was dan een threshold, werd deze gelimiteerd tot de threshold. Hierdoor werden sprites minder fel naar de locatie geforceerd.

Path constraint: Een pad bestond uit een reeks lijnsegmenten. Om een sprite over zo een lijnsegment te laten bewegen, werden 2 eigenschappen berekend: de afstand tot het einde van het huidige lijnsegment en de hoek tussen de velocity van de sprite en de richting van het lijnsegment. Telkens wanneer er een einde van een lijnsegment bereikt werd, werd het volgende lijnsegment genomen. Wel viel eerst op dat de sprite na 1 lijnsegment niet meer het pad wou volgen, maar op de locatie wou blijven. Dit kwam omdat de nieuwe animatiekost immers maximaal werd: de sprite was ver van de nieuwe locatie af en de velocities verschilden fel. Daarom liet ik per lijnsegment diens index meetellen, zodat de sprite het volgende lijnsegment volgde.

Anti-collision constraint: De anti-collision constraint implementeerde ik als een inverse van een locatie constraint. De sprite moest immers de locatie vermijden. Wanneer ik zowel een locatie als een anti-collision constraint in elkaars buurt plaatste, werd de constraint met het laagste gewicht genegeerd. Dit is logisch, omdat de constraints elkaar inverse zijn.

Area constraint: De area constraint is soms een handig alternatief voor de locatie constraint, wanneer de exacte positie niet zo belangrijk is. Ook voor grotere gebieden

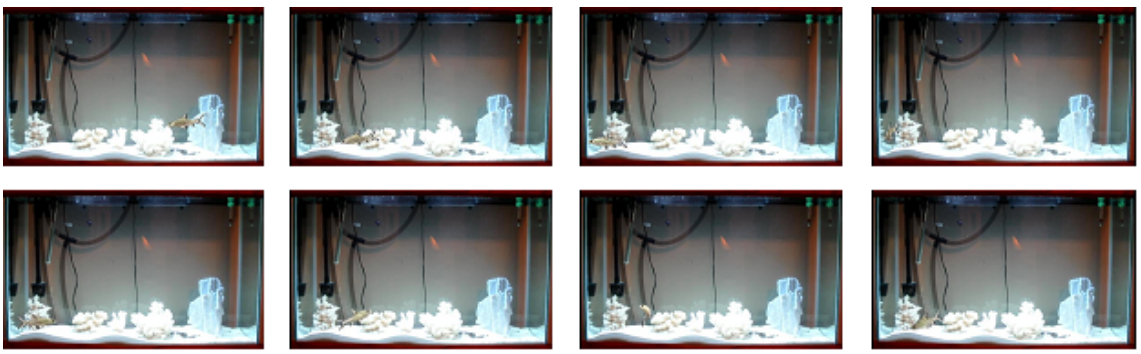
is de area constraint gemakkelijker, aangezien de animator dit gebied met bijvoorbeeld een brush kan aanduiden. Wanneer locatie constraints gebruikt worden, moet de animator er immers vele aangeven.

Tijd: Wanneer bijvoorbeeld een sprite een pad moet afleggen, is het soms handig ook tijstippen bij de positie te voegen. Tijd kan toegevoegd worden door voor elk eindpunt van het pad een tijdstip mee te geven waarop de sprite op dit punt moet zijn. Wanneer de sprite er te vroeg of te laat is, kan een penalty worden meegeteld afhankelijk van hoeveel de sprite te vroeg of te laat is. Dit zorgt er voor dat transities die de sprite op een bepaald tijdstip op een eindpunt brengen, meer zullen meetellen.
Toch is deze manier van werken niet ideaal. Zo wordt enkel bij het eindpunt een tijdstip geassocieerd. Hierdoor is er tijdens de beweging geen beperking op de tijd. Dit kan als resultaat geven dat de sprite nog altijd te snel naar zijn eindpunt gaat, en er dan gedurende een bepaalde tijd blijft eer terug te keren. Om meer egale bewegingen te bekomen, moet per tijdstip een nieuwe locatie en velocity berekend worden, zoals de dynamische aanpak in sectie 7.4.1 demonstreerde.

8.3.3 Resultaten

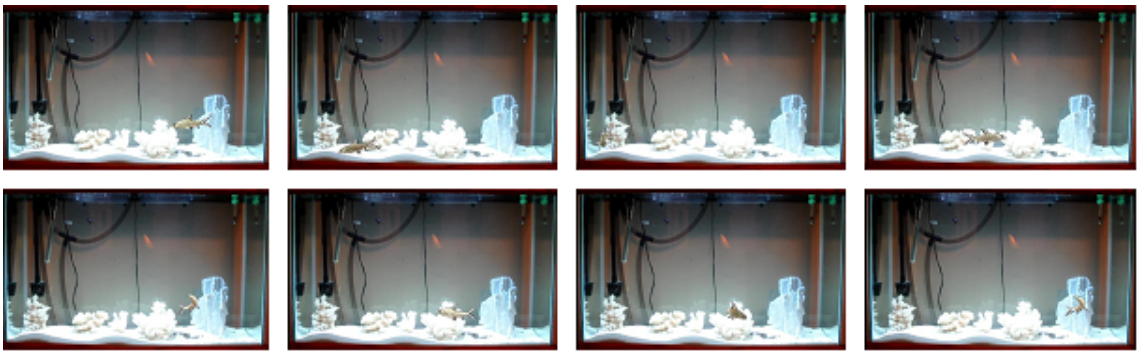
Deze sectie bevat enkele van mijn resultaten. Ik zal deze resultaten ook vergelijken met resultaten van anderen, om een beter beeld te schetsen.

In een eerste voorbeeld heb ik een locatie constraint toegevoegd. Deze locatie was aan de andere kant van het aquarium, zodat de vis eerst een weg tot de locatie moest vinden. Omdat de vis initiëel ver van deze locatie was, werd hij ernaar geforceerd. Dit kwam doordat de velocity van de sprite niet werd gebruikt tijdens het bepalen van de transities.



Figuur 8.8: De vis werd initiëel rechts geplaatst, en de location constraint bevond zich links. Hierdoor zwom de vis eerst zo snel mogelijk naar de locatie, en bleef er vervolgens rondzwemmen.

Een tweede voorbeeld had een path constraint. De vis moest 1 keer heen en weer zwemmen, en dan op de laatste positie blijven zwemmen.



Figuur 8.9: De vis werd initiëel rechts geplaatst, en een pad constraint bestaande uit 2 segmenten werd toegevoegd. Het pad liep eerst van rechts naar links, en vervolgens terug. Daarna moest de vis op de positie aan het eind van het pad blijven voor enkele seconden.

8.3.4 Discussie

Duidelijk was te zien dat er met video sprites veel meer animatiemogelijkheden zijn dan video textures. Er werden verschillende animatie constraints gedemonstreerd, waaronder path-following.

Veel hangt af van de input-video. Deze moet voldoende lang zijn, anders zijn niet alle bewegingen aanwezig. Bewegingen die niet in de input-video waren, kunnen ook niet gecreëerd worden. Zo heb ik met een voorbeeld waar een vis enkel horizantale bewegingen maakte geprobeert een verticale beweging te simuleren. Dit mislukte volledig.
Ook de kwaliteit van de video speelt een belangrijke rol. Ik heb een voorbeeld van een hamster gebruikt maar faalde, omdat de video te veel compressie artifacten had. Daardoor werden geen mooie sprites uit de video gehaald, en dus werden ook geen goede transities gevonden.

Maar het voorbeeld van de vis leverde wel mooie resultaten, vergelijkbaar met resultaten uit de originele paper. Toch waren met mijn voorbeeld minder animaties mogelijk, omdat ik de hill-climbing optimalisatie niet gebruikte. Vooral wanneer meerdere sprites gebruikt werden, was beam search alleen niet voldoende om mooie resultaten te geven. Voor gewone locatie of pad constraints, gaf beam search visueel goede resultaten.

Toch waren er enkele beperkingen. Zoals eerder bewezen is beam search niet zo geschikt om transities te controleren. Andere zoekalgoritmen die verder in de tijd kunnen kijken zijn nodig. Ook is een betere manier om de sprite velocities automatisch te berekenen nodig. Dit zou de globale kwaliteit van de animaties al erg verbeteren. Hierdoor zou blending niet meer nodig worden, waardoor scherpere beelden bekomen worden.

Hoofdstuk 9

Conclusie

Animatie is één van de belangrijkste takken binnen de computer graphics. Het zorgt er immers voor dat objecten levendiger weergegeven kunnen worden. Nu is het creëren van computermodellen en -animaties van complexe objecten een moeilijk karwei. Meestal wordt er veel van de animator verwacht. Video-based animation probeert het werk van de animator te verlichten, door gecapturede data te gebruiken van de objecten. Hierdoor is het realisme immers gegarandeerd.

In deze thesis werd video-based rendering besproken. Eerst werd het belang ervan aangeduid, en waarom andere render- en animatiemethoden in sommige gevallen falen. Ook werd video-based animation gesitueerd binnen video-based rendering.

Video textures liet de animator toe oneindige animaties te maken door de volgorde waarin de video frames afgespeeld werden voortdurend te veranderen. Dit was mogelijk door alle frames met elkaar te vergelijken en goede transities te gebruiken. Er werd aangetoond dat de L_2 norm de meest geschikte metriek is voor het bepalen van deze transities.
Toch waren de bekomen animaties in sommige gevallen visueel matig. Voor deze problemen gaf de symmetrische aanpak een oplossing, door de volgorde van afspelen van de video frames om te draaien.
Verder werd ook gedemostreerd dat video textures ook werken op verschillende types invoer data, zoals cartoon textures. Er kwamen eveneens verschillende toepassingen van video textures aan bod, die het belang van video textures aanduide.

Vervolgens werd aangetoond hoe video textures uitgebreid konden worden naar video sprites. Deze video sprites boden veel meer animatiemogelijkheden. De animator had nu controle over de animatie door te kiezen tussen verschillende constraints. Hierdoor werd path-following mogelijk, en konden andere toepassingen zoals flocking geïmplementeerd worden.

Lijst van figuren

2.1 Het spel “Mario” als voorbeeld van 2D rendering 8
2.2 Een polygonmesh voor 3D rendering 9
2.3 Doom als voorbeeld van ray casting 10
2.4 Ray tracting reflecties 10
2.5 De plenoptische functie bij image-based rendering 11
2.6 creëren van een nieuwe view uit een reeks afbeeldingen 12
2.7 Een serie sprites voor een zwemanimatie van het spel “Mario”, handgetekend door de animator 13
2.8 Sprite animatie met bill-boarding 13
2.9 Key-framing als animatiemethode 14
2.10 Fluid dynamics als voorbeeld van een fysische simulatie 15

3.1 Voorbeeld van een video-opstelling om data te capturen 17
3.2 Background subtraction als segmentatie algoritme 18
3.3 Verwijderen van ruis bij background subtraction 20
3.4 Graph cut als segmentatie algoritme 21
3.5 Uitgewerkt voorbeeld van een graph cut 22
3.6 Visual hull van een object in de ruimte 23
3.7 View-dependent renderen binnen video-based rendering 23
3.8 Voorbeeld van een 3D TV 24
3.9 Werking van een 3D display 24

4.1 Principe van texture mapping 27
4.2 Temporal textures 29
4.3 De Haussdorf afstand 32
4.4 Het CIE kleur chromaciteitsdiagram 33

5.1 Preprocessing voor video rewrite 35
5.2 Resultaten van video rewrite 36
5.3 Transities voor video textures 37
5.4 Een videosequentie met een doodlopend einde 37
5.5 Looping bij video textures 40
5.6 Partiële aanpak voor video textures 41
5.7 Een symmetrisch punt binnen een video texture 42
5.8 Een animatie m.b.v. cartoon textures 45
5.9 Een animatie m.b.v. panoramic video textures 46
5.10 Texture synthese door middel van graph cut technieken 47
5.11 Graph cut textures gebruikt voor video 48

5.12 Een animatie m.b.v. flow-based video synthese en editing 49

6.1 Perspectieve correctie voor video sprites 51

7.1 Berekenen van een transitie tussen 2 sprites 53
7.2 Hill-climbing optimalizatie voor beam search 55
7.3 Animatie met pad constraint 57
7.4 Animatie met anti-collision constraint 57
7.5 Animatie met frame constraint 58
7.6 Interactieve animatie 59
7.7 Tijd toevoegen bij pad animatie 61
7.8 Backward transities 62
7.9 Flocking 63

8.1 Vergelijking van de verschillende metrieken voor video textures 66
8.2 Vergelijking van de verschillende metrieken voor video sprites 67
8.3 Voorkomen van dead ends bij video textures 69
8.4 Resultaten van video textures 70
8.5 Symmetrisch punt in video textures 70
8.6 Vergelijking video textures met symmetrische aanpak 71
8.7 Sprite animatie via random play 73
8.8 Sprite animatie via location constraint 75
8.9 Sprite animatie via path constraint 76

Bibliografie

[AF02] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *In Proceedings of SIGGRAPH 2002, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series*, 2002.

[AZP+05] A. Agarwala, K.C. Zheng, P. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski. Panoramic video textures. *ACM SIGGRAPH 2005 conference proceedings*, 2005.

[BCM97] C. Bregler, M. Covell, and Slaney M. Video rewrite: Driving visual speech with audio. *Computer Graphics (SIGGRAPH97)*, page p.353 tot 360, 1997.

[BJ01] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. *Proceedings of International Conference on computer vision*, Vol.1:p.105, 2001.

[Bon97] J. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics (SIGGRAPH97)*, page 361 tot 368, 1997.

[Bou70] W. J. Bouknight. A procedure for generation of three-dimensional half-tone computer graphics presentations. *Communications of the ACM*, 1970.

[BSHK03] K. Bhat, S. Seitz, J. Hodgins, and P. Khosla. Flow-based video synthesis and editing. *In Siggraph 2003, Computer Graphics Proceedings*, 2003.

[Bun05] J. Buntinx. Background subtraction. Master's thesis, Universiteit Hasselt, 2005.

[BW71] N. Burtnyk and M. Wein. Computer generated key frame animations. *Journal of the society of motion picture and television engineers*, Vol.8:p.149 tot 153, 1971.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, Vol.6:p.679 tot 698, 1986.

[Cat74] E. Catmull. *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah, 1974.

[CBHK04] K.M. Cheung, S. Baker, J. Hodgins, and T. Kanade. Markerless human motion transfer. *Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission*, 2004.

[CBK05] K.M. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time: part ii: applications to human modeling and markerless motion tracking. *International Journal of Computer Vision*, Vol.63(3):p.225 tot 245, 2005.

[CDG+04] N. Campbell, C. Dalton, D. Gibson, D. Oziem, and B. Thomas. Practical generation of video textures the auto-regressive process. *British Machive Vision Conference*, page 434 tot 443, 2004.

[CGPP03] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts and shadows in video streams. *IEEE Trans. on Patt. Anal. and Machine Intell*, Vol.25(10):p.1337 tot 1342, 2003.

[Che95] S. E. Chen. Quicktime vr - an image-based approach to virtual environment navigation. *Proceedings of SIGGRAPH 95*, page 29 tot 38, 1995.

[Coh92] M. F. Cohen. Interactive spacetime control for animation. *Computer Graphics (Proceedings of SIGGRAPH 92)*, Vol.26:p.293 tot 302, 1992.

[Da99] P. Debevec and al. Image-based modeling, rendering, and lighting. *SIGGRAPH99 Course 39*, 1999.

[Dec04] B. De Decker. Simulatie en visualisatie van vloeistoffen. Master's thesis, Universiteit Hasselt, 2004.

[dJB04] C. de Juan and B. Bodenheimer. Cartoon textures. *In Eurographics Symposium on Computer Animation*, 2004.

[EF01] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH*, 2001.

[EFS56] P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *RE Transactions on Information Theory IT-2*, page 117 tot 119, 1956.

[EL99] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *Seventh International Conference on Computer Vision (ICCV99)*, page 1033 tot 1038, 1999.

[FAH91] W. T. Freeman, E. H. Adelson, and D. J. Heeger. Motion without movement. *Computer Graphics (Proceedings of SIGGRAPH 91)*, Vol.25:p.27 tot 30, 1991.

[Gle98] M. Gleicher. Retargeting motion to new characters. *Proceedings of SIGGRAPH 98*, page 33 tot 42, 1998.

[Gre86] N. Greene. Environment mapping and other applications of word projections. *IEEE Computer Graphics and Applications*, Vol.6:p.21 tot 29, 1986.

[GTGB84] C. Goral, K.E. Torrance, D.P. Greenberg, and Battaile B. Modelling the interaction of light between diffuse surfaces. *Computer Graphics (Proceedings of SIGGRAPH 1984)*, Vol.18:p.213 tot 222, 1984.

[GW01] R. C. Gonzalez and R. E. Woods. *Digital image processing*. Prentice hall, tweede edition, 2001.

[HB95] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 1995*, page 229 tot 238, 1995.

[HFR05] W. Van Haevre, F. Di Fiore, and F. Van Reeth. Uniting cartoon textures with computer assisted animation. 2005.

[HJO+01] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. *Proceedings of SIGGRAPH*, 2001.

[HR05] W. Van Haevre and F. Van Reeth. Video textures exploiting symmetric movements. *Short Paper Proceedings of Eurographics 2005*, page p.57 tot 60, 2005.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *In Proceedings of SIGGRAPH 2002, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series*, 2002.

[KLM96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *In Journal of Artificial Intelligence Research*, Vol.4:p.237 tot 285, 1996.

[KSEB03] V. Kwatra, A. Schödl, I. Essa, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *In Siggraph 2003, Computer Graphics Proceedings*, 2003.

[Lau94] Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans.Pattern Anal. Machine Intell.*, Vol.16(2), 1994.

[LCR+02] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *In Proceedings of SIGGRAPH 2002, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series*, 2002.

[LV00] B.P.L. Lo and S.A. Velastin. Automatic congestion detection system for underground platforms. *Proc. of 2001 Int. Symp. on Intell. Multimedia, Video and Speech Processing*, page 158 tot 161, 2000.

[MB95] L. Mcmillan and G. Bishop. Plenoptic modeling: An imagebased rendering system. *Proceedings of SIGGRAPH 95*, page 39 tot 46, 1995.

[MPC+05] M. Magnor, M. Pollefeys, G. Cheung, W. Matusik, and C. Theobalt. Video-based rendering. *ACM Siggraph*, 2005.

[ORP00] N. M. Oliver, B. Rosario, and A. P. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Trans. on Patt. Anal. and Machine Intell*, Vol.22(8):p.831 tot 843, 2000.

[Par05] R. Parent. *Computer animation: algorithms and techniques*. Morgan Kaufmann publishers, 2005.

[PBMH02] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):p. 703 tot 712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[PD80] P. E. Pfeifer and S. J. Deutsch. A three-stage iterative procedure for space-time modeling. *Technometrics*, Vol.22:p.35 tot 47, 1980.

[PH98] S. Pollard and S. Hayes. 3d video sprites. *Digital Media Department*, 1998.

[Pic04] M. Piccardi. Background subtraction techniques: a review. *University of Technology, Sydney*, 2004.

[Pot87] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision Graphics And Image Processing*, Vol.40:p.1 tot 29, 1987.

[PPHL98] S. Pollard, M. Pilu, S. Hayes, and A. Lorusso. View synthesis by trinocular edge matching and transfer. *British MachineVision Conference (BMVC98)*, 1998.

[PW99] Z. Popovic and A. Witkin. Physically based motion transformation. *Proceedings of SIGGRAPH 99*, page 11 tot 20, 1999.

[Rey86] C. Reynolds. Flocking behavior from the simulation program boids. 1986.

[RKB04] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. 2004.

[SDW01] S. Soatto, G. Doretto, and Y. N. Wu. Dynamic textures. *In Proceeding of IEEE International Conference on Computer Vision*, page 439 tot 446, 2001.

[SE01] A. Schödl and I. Essa. Machine learning for videobased rendering. *In Advances in Neural Information Processing Systems*, Vol.13:p.1002 tot 1008, 2001.

[SE02] A. Schödl and I. Essa. Controlled animation of video sprites. *In ACM Symposium on Computer Animation on ACM Siggraph*, 2002.

[SG00] C. Stauffer and W.E.L. Grimson. Adaptive background mixture modelsfor real-time tracking. *Proc. of 2001 Int. Symp. on Intell. Multimedia, Video and Speech Processing*, page 158 tot 161, 2000.

[SHP04] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. 2004.

[SP96] M. Szummer and R. W. Picard. Temporal texture modeling. *In Proceeding of IEEE International Conference on Image Processing*, Vol.3:p.823 tot 826, 1996.

[SSSE00] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. *Proceedings of SIGGRAPH 2000*, page 489 tot 498, 2000.

[SWND02] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL programming guide*. Addison-Wesley, vijfde edition, 2002.

[TLMS03] C. Theobalt, M. Li, M. Magnor, and H. Seidel. A flexible and versatile studio for synchronized multi-view video recording, 2003.

[VMPX04] A. Vetro, W. Matusik, H. Pfister, and J. Xin. Coding approaches for end-to-end 3d tv systems. *CM Siggraph*, 2004.

[WADP97] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: real-time tracking of the human body. *IEEE Trans. on Patt. Anal. and Machine Intell*, Vol.19(7):p.780 tot 785, 1997.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM 23*, Vol.6:p.343 tot 349, 1980.

[wik] Wikipedia foundation. http://en.wikipedia.org/wiki/.

[WK88] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics (Proceedings of SIGGRAPH 88)*, Vol.22:p.159 tot 168, 1988.

[WL00] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH*, 2000.

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Video-gebaseerde Computeranimatie
Richting: **Master in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Stijn STEEGEN

Datum: