3D-Video Compressie

Bart Cornelissen

21 augustus 2005

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Samenvatting

Deze thesis beschrijft hoe de extra bron van redundantie die 3D-video introduceert uitgebuit kan worden om zo efficiëntere compressie van de video sequenties te bekomen. Deze overbodige informatie onstaat doordat dezelfde scène gelijktijdig met verschillende camera's gefilmd wordt.

Er worden twee compressiesystemen voorgesteld. Het eerste is een uitbreiding van klassieke videocompressie, zodat ook de extra ruimtelijke redundantie gebruikt wordt. Het tweede systeem maakt gebruikt van aanvullende 3D-informatie van de scène, die berekend wordt door scène-reconstructie algoritmen.

Woord vooraf

Deze thesis werd gemaakt om de graad van Licentiaat in de Informatica, afstudeervariant multimedia, aan het Limburgs Universitair Centrum, sinds kort Universiteit Hasselt, te behalen.

Voor de realisatie van deze thesis wil ik nog enkele mensen bedanken. Uiteraard wil ik mijn promoter, en tevens begeleider, Prof. Dr. Philippe Bekaert bedanken voor de deskundige hulp tijdens het tot stand brengen van deze thesis. Ik wil ook de verschillende medeleerlingen bedanken die hier en daar wat hulp hebben geboden; in het bijzonder Yannick Francken, waar ik meer dan eens terecht kon voor zijn mening.

Ik wil ook mijn vriendin, Sylvie Hrynczak bedanken voor de vele steun, en de hulp bij het maken van de illustraties. Ik wil ook mijn ouders bedanken die het in de eerste plaats mogelijk maakten deze studies aan te vatten, en ook voor de steun. Bedankt ook aan de ouders van Sylvie, waarbij ik vaak verbleef tijdens het maken van deze thesis.

Hoofdstuk 1

Inleiding

Als een scène met een videocamera gefilmd wordt, wordt deze drie-dimensinale omgeving geprojecteerd op een twee-dimensinaal vlak. Hierdoor wordt de drie-dimensionale informatie niet vastgelegd door de camera. 3D-video is een uitbreiding van traditionele video om ook deze ruimtelijke informatie te kunnen vastleggen.

Een belangrijk gevolg van de beschikbare ruimtelijk informatie is dat de kijker, tijdens het afspelen van de videobeelden, controle kan krijgen over het camera-standpunt. Dit wordt 'free-viewpoint video' genoemd. Dit is in tegestelling tot traditionele video, waar enkel de tijd gecontroleerd kan worden; er kan terug- en verdergespoeld worden, maar het camerastandpunt kan niet meer aangepast worden. Deze extra vrijheidsgraad maakt het bijvoorbeeld mogelijk een voetbalmatch vanaf het standpunt van de scheidsrechter, of vlak achter de bal te bekijken.

3D-video belooft ook veel toepassingen te vinden binnen de tele-immersie. Zo wordt het bijvoorbeeld mogelijk gemaakt rond te lopen in een virtuele omgeving die overeenkomt met een echte omgeving op datzelfde moment ergens anders op de wereld.

De ruimtelijke informatie van een scène wordt vastgelegd door deze scène met verschillende camera's, vanop verschillende standpunten, te filmen. De bekomen twee-dimensionale beelden van de verschillende camera's worden dan gecombineerd om de drie-dimensionale scène te reconstrueren.

Dit leidt tot een gigantische hoeveelheid gegevens die opgeslagen of verstuurd moeten worden. Bestaande videocompressie technieken kunnen de hoeveelheid bits die nodig zijn om deze informatie over te brengen aanzienlijk beperken. De verschillende gegevensstromen, afkomstig van de verschillende camera's, kunnen telkens als een afzonderlijke videosequentie gecomprimeerd worden. Deze compressie techieken reduceren de hoeveelheid bits door redundante informatie binnen elk beeld, en ook tussen opeenvolgende beelden te elimineren.

Als op deze manier gecomprimeerd wordt, wordt nog steeds een grote hoeveelheid overbodige informatie gecodeerd. Doordat de verschillende camera's dezelfde scène, op hetzelfde moment filmen, zal een groot deel van de informatie

die een camera vastlegt ook te zien zijn in een andere camera. Dit maakt het mogelijk de verschillende videosequenties niet gewoon onafhankelijk van elkaar te comprimeren, maar ook rekening te houden met redundante informatie tussen verschillende videosequenties.

Onlangs[1] publiceerde de Moving Picture Experts Group[MPE] een oproep om voorstellen voor een 'Multi-view Video Coding' standaard in te dienen. Onderstaande citaten uit deze tekst wijzen op de relevantie en het doel van zo een standaard.

> "It has been recognized that multi-view video coding is a key technology that serves a wide variety of applications, including free-viewpoint and 3D video applications, including home entertainment and surveillance."

> "The objective of this Call for Proposal is to standardize new technologies for the encoding of multiview sequences improving the coding efficiency of current video coding solutions performing simulcast of independent views, for a similar degree of encoder optimization."

[1] De 'Call for Proposals' werd gepubliceerd tijdens de 73^e WG11 bijeenkomst op 25-29 juli 2005, en is te vinden op officiële MPEG website[MPE].

Hoofdstuk 2

3D-Video

3D-video is een uitbreiding van traditionele, twee-dimensionale video naar drie dimensies. Dit wordt gerealiseerd door eenzelfde scène gelijktijdig met verscheidene camera's, en dus vanuit verschillende standpunten, te filmen. Door deze drie-dimensionale informatie is het mogelijk beelden vanop willekeurige camera-standpunten te genereren, wat '*free-viewpoint video*' genoemd wordt.

Een traditionele, twee-dimensionale videosequentie is een door een camera vastgelegde serie afbeelingen, genomen op opeenvolgende tijdstippen, die telkens de gefilmde scène vanuit het standpunt van de camera op dat moment weergeven. Ieder beeld is de projectie van de scène op het beeldvlak van de camera. Door de projectie wordt een drie-dimensionale scène dus naar een twee-dimensionaal beeld herleid. Hierdoor gaat de drie-dimensionale informatie verloren. De afstand tot de camera, de diepte-informatie dus, wordt in het beeld niet vastgelegd. Iedere pixel komt nu overeen met een rechte, in plaats van een punt, in de ruimte. Andersom worden een hele reeks mogelijke 3D-punten, gelegen op een rechte die het optische centrum en het beeldvlak van de camera snijdt, op eenzelfde pixel geprojecteerd.

Door dezelfde scène ook vanuit een tweede camera-standpunt vast te leggen, kan deze diepte-informatie teruggewonnen worden. De reeks 3D-punten die in de ene camera allemaal op dezelfde pixel geprojecteerd werden, zullen in een tweede camera allemaal op verschillende pixels geprojecteerd worden[1]. Het vastgelegde 3D-punt kan dan opnieuw bekomen worden door de intersectie van de twee stralen, behorende bij de twee pixels waarop het punt geprojecteerd werd, te zoeken.

De reconstructie van een drie-dimensionale scène, uitgaande van een aantal camerabeelden, is echter niet zo eenvoudig. Niet ieder scène-punt is in iedere camera zichtbaar doordat tussenliggende punten de zichtbaarheid belemmeren. Uit de invoerbeelden kan ook niet zomaar bepaald worden of een punt, overeenkomend met een pixel in de ene camera, zichtbaar is een andere camera. Het is ook niet mogelijk het paar van pixels, afkomstig uit twee verschillende beelden, die hetzelfde, zichtbare punt vertegenwoordigen zomaar te bepalen. Hiervoor is

[1] Uitgaande van een 'logische' cameraopstelling, waarbij de twee camera's niet achter elkaar staan

namelijk juist de gezochte drie-dimensionale informatie nodig. Er kan gezocht worden naar pixels met dezelfde kleur, maar een beeld bevat gewoonlijk meerdere pixels met dezelfde kleur. Door reflectie-eigenschappen van het scène-element op dat punt kunnen de overeenkomstige pixels in de verschillende beelden ook verschillende kleuren hebben.

2.1 Overzicht

Het reconstrueren van een drie-dimensionale scène uit meerdere camera's is een klassiek probleem in de *computer visie*, een deeltak van de artificiële intelligentie die zich bezighoudt met het analyseren en begrijpen van de inhoud van afbeeldingen en video.

Om het scène-reconstructie probleem op te lossen, zijn reeds een hele hoop algoritmes geformuleerd. De verschillende benaderingen kunnen ruwweg in twee klassen opgesplitst worden: scène-gebaseerde benaderingen, en beeld-gebaseerde benaderingen. De eerste klasse bevat alle methodes die de vorm van een complexe scène trachten te reconstrueren. De beeld-gebaseerde benaderingen proberen rechtstreeks tussenliggende beelden te berekenen, zonder de scène expliciet te modelleren.

In deze sectie wordt een overzicht gegeven van een aantal belangrijke algorimtes, zowel scène- als beeld-gebaseerde. In de hieropvolgende secties wordt dieper op deze algoritmes inggegaan, zonder uit te weiden naar implementatie-details. Hiervoor wordt verwezen naar de betreffende bronnen.

2.1.1 scène-gebaseerde benaderingen

De scène-gebaseerde methodes [Dye01, AV89, MBR$^+$00, SVZ00, Col96, SD99, KS00] werken met een drie-dimensionaal volume waarin de scène wordt opgebouwd. Deze benadering voor het scène-reconstructie probleem wordt '*volumetric scène modeling*' genoemd.

'*Voxel occupancy*' is een methode die de drie-dimensionale vorm van een object tracht te reconstrueren door de scène voor te stellen als een set van drie-dimensionale voxels. De oplossing wordt dan gegenereerd door de verschillende voxels te markeren als 'gevuld' of 'leeg'. Deze methode gaat uit van de twee-dimensionale silhouetten van het object in de verschillende camera's. Het resultaat van deze methode is niet de echte drie-dimensionale vorm van het object, maar een benadering die de '*visual hull*' genoemd wordt.

Voxel occupancy maakt echter enkel gebruik van het twee-dimensionale silhouet van het object, en negeert de kleurenvariaties. Deze kleureninformatie kan als een extra richtlijn gebuikt worden bij het markeren van de voxels. De beperking dat een geldig punt in de verschillende beelden dezelfde kleur moet hebben, rekening houdend met het gepaste belichtings- en reflectiemodel, wordt *foto consistentie* genoemd. Twee bekende algoritmes die hiervan gebruik maken zijn 'voxel coloring' [SD99] en 'space carving' [KS00].

Een belangrijke beperking van voxel coloring en space carving is dat ze geen spatiale coherentie opleggen. Dit is nogthans belangrijk aangezien de data van de

invoerbeelden vrijwel altijd dubbelzinnig is. Door rekening te houden met spatiale coherentie kan een betere beslissing gemaakt worden bij het selecteren van een scène uit de de verschillende mogelijke reconstructies, die allemaal voldoen aan de data van de invoerbeelden. De spatiale coherentie laat toe de oplossing met de minst grillige vorm te kiezen.

Een andere beperking volgt uit het feit dat deze methodes 'harde' beslissingen nemen tijdens het doorlopen van de voxels; Beslissingen die later dus niet zomaar ongedaan gemaakt kunnen worden. Doordat de beelddata dubbelzinnig is, is de kans groot dat deze beslissingen incorrect zijn.

2.1.2 Beeld-gebaseerde benaderingen

Beeld-gebasserde benaderingen voor het scène-reconstructie probleem genereren hun oplossing niet door de scène expliciet te modelleren, zoals de scène-gebaseerde benaderingen. De berekeningen vinden plaats op beeld-niveau, in plaats van bewerkingen in een drie-dimensionale ruimte uit te voeren. scène-gebaseerde benaderingen construeren scène-elementen die consistent zijn met de invoerbeelden. Beeld-gebaseerde benaderingen zoeken overeenkomstige elementen in de verschillende invoerbeelden die onderling consistent zijn. De beeld-gebaseerde methodes geven dus ook een oplossing voor het scène-reconstructie probleem, hoewel ze de scène niet expliciet reconstrueren.

Een veel belovende familie van beeld-gebaseerde algoritmen is gebaseerd op een energie-formulatie van het scène-reconstructie probleem [Roy99, BK01, KZ01, KZ02, KZG03]. Door deze formulatie wordt het probleem herleid tot een energie-minimalisatie probleem. Het minimaliseren van de energie gebeurt aan de hand van een graaf, waarin de minimum-snede berekend wordt.

De energie-minimalisatie formulatie kent verschillende voordelen. Het zorgt in de eerste plaats voor een elegante specificatie van het op te lossen probleem. Bovendien maakt de energie-minimalisatie het gebruik van 'zachte' beperkingen, zoals spatiale coherentie, mogelijk. Dubbelzinnigheden kunnen opgelost worden op een manier die tot een spatiaal 'vloeiende' oplossing leidt.

2.2 Volumetric scène modeling

Volumetric scène modeling technieken stellen een reconstrie op van de scene in een drie-dimentinale ruimte. Hierbij wordt gebruik gemaakt van een set van silhouetten, of van foto-consistentie.

2.2.1 Reconstructie met silhouetten

Een silhouet wordt voorgesteld als een binaire afbeelding, die voor iedere pixel aangeeft of de straal, die vanuit het optische centrum door deze pixel gaat, al dan niet intersecteert met een object van de scène. Iedere pixel is dus ofwel deel van het silhouet, ofwel achtergrond. Deze binaire afbeelding kan bekomen worden door segmentatie algoritmes, of door 'blue screen' technieken [SH].

bevinden worden doorzichtig gemaakt. Hierdoor onstaat de benadering van de visual hull. Om deze voxels op een efficiënte manier in te kleuren, zijn verschillende algoritmen ontwikkeld [AV89, MBR+00, SVZ00].

2.2.2 Reconstructie door foto-consistentie

Wanneer er volledige beelden van de scène beschikbaar zijn, dus niet enkel silhouetten, kan deze extra informatie gebruikt worden om het reconstructieproces te verbeteren. De set van beelden van de verschillende camera's kan beschouwd worden als een set van beperkingen die opgelegd worden aan het te construeren scène-model. Dit is vergelijkbaar met de manier waarop de gebruikte silhouetten beperkingen oplegden aan het volume van het te reconstrueren object.

Een perfect gereconstrueerde scène die op het beeldvlak van één van de camera's geprojecteerd wordt, zou een gesynthesiseerd beeld moeten opleveren dat gelijk is aan het beeld van die camera, dat voordien gebruikt werd als invoer van het reconstructieproces. Een hypothetisch scène-model kan op deze manier dus geverifiëerd worden door echte en gesynthetiseerde beelden met elkaar te vergelijken, in plaats van de consistentie van 2D- of 3D-eigenschappen die afgeleid worden uit de invoerbeelden te evalueren.

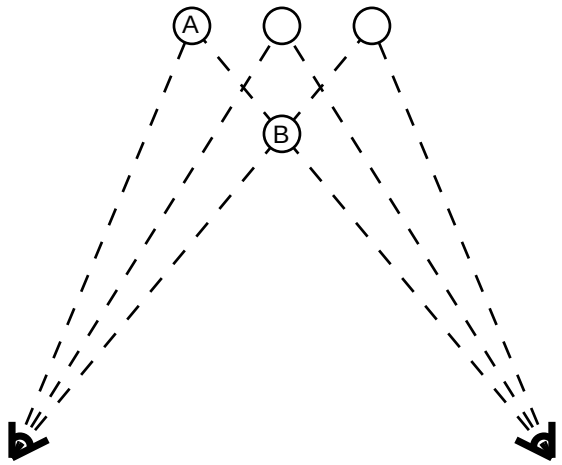
De vergelijking van het gesynthetiseerde beeld met het echte camerabeeld gebeurt op basis van kenmerken van het scène-model en het render proces. Een volledig model moet niet enkel rekening houden met de geometrie, maar ook met gepaste belichtings- en reflectiemodellen.

Een scène-element met een consistent voorkomen bij de verschillende camera posities wordt *foto-consistent* genoemd [SD99, KS00]. Een gegenereerd scène-element is dus foto-consistent met een set van camerabeelden als, voor iedere camera waarin het element zichbaar is, de projectie op het beeldvlak van de camera, rekening houdend met gepaste reflectie- en belichtingseigenschappen, gelijk is aan het invoerbeeld.

Een scène kan dus gereconstrueerd worden door alle voxels initeel als transparant te markeren, en ze dan één voor één te doorlopen. Door de voxels telkens op het beeldvlak van de verschillende camera's te projecteren, en met de overeenkomstige pixel(s) van het camerabeeld te vergelijken, kan de foto-consistentie van de voxel geëvalueerd worden. Iedere voxel die foto-consistent is met alle camerabeelden waarin de voxel zichtbaar is, wordt ingekleurd, en maakt dus deel uit van de te construeren scène.

Een voxel A is zichtbaar in camera i als er geen andere voxel B bestaat die (1) tot de scène behoort, en (2) voxel A bedekt in het beeld van camera i. Het probleem is dat er misschien wel zo een voxel B bestaat, maar dat die nog niet ingekleurd is op het moment dat dat voxel A getest wordt. Hierdoor bestaat dus het gevaar dat voxel A onterecht niet ingekleurd wordt. Dit fenomeen wordt geïllustreerd in figuur 2.2. Dit zichtbaarheidsprobleem kan opgelost worden door de voxels in een speciale volgorde te doorlopen, of door een algoritme te gebruiken dat de pixels meer dan één keer evalueert.

Het vermogen om na te kijken of een voxel foto-consistent is met de set van camera's, gecombineerd met het vermogen om te testen of de voxel zichtbaar is



Figuur 2.2: *De cirkels geven de voxels aan waaruit een correcte reconstructie zou moeten bestaan; De voxels die ingekleurd moeten worden dus. Als voxel B nog niet ingekleurd is op het moment dat voxel A geëvalueerd wordt, zal A foutief beschouwd worden als een voxel die zichtbaar is in de rechter camera. Hierdoor bestaat de kans dat voxel A onterecht als niet-foto-consistent geëvalueerd wordt. De projectie van voxel A op de rechter camera wordt immers vergeleken met de pixel die overeenkomt met voxel B. Indien voxel B al wel ingekleurd was, zou deze projectie niet meegerekend worden in de evaluatie van de foto-consistentie van voxel A, aangezien voxel A niet zichtbaar is in deze camera.*

in de verschillende camera's, vormt de basis voor een hele familie van methodes om de reconstructie van een 3D scène te genereren. 'Voxel coloring' [SD99] en 'space carving' [KS00] zijn twee van deze methodes, die onderling verschillen in de manier waarop de zichtbaarheid van voxels getest wordt.

Voxel coloring

Het '*voxel coloring*' [SD99] algoritme gaat uit van een camera opstelling die toelaat de voxels in verschillende lagen te doorlopen, telkens met toenemende afstand tot de camera's. Hierdoor wordt voor iedere voxel A gegarandeerd dat alle andere voxels die voxel A eventueel zouden kunnen bedekken reeds geëvalueerd zijn.

Om de foto-consistentie te testen, worden lambertiaanse (diffuse, matte) oppervlakken en constante belichting verondersteld. Hierdoor kan het gereflecteerde licht simpelweg door een kleurwaarde gekarakteriseerd worden. Een voxel die foto-constistent is met alle camerabeelden waarin de voxel zichtbaar is, wordt bijgevolg meteen gemarkeerd met de waargenomen kleur.

Space carving

Voor algemenere camera opstellingen is een 'one-pass' algoritme niet meer mogelijk, doordat de voxels niet langer topologische gesorteerd kunnen worden ten

opzichte van de verschillende camera standpunten. Het '*space carving*' [KS00] algoritme zal bijgevolg de set van voxels meerdere keren doorlopen. In iedere doortocht wordt mogelijk de markering van een aantal voxels aangepast, waardoor de zichtbaarheid, en bijgevolg ook de markering, van andere voxels aangepast dient te worden. Dit gebeurt in de daarop volgende doortocht. Het algoritme stopt wanneer er niets meer verandert.

Initiëel worden alle voxels als 'gevuld' gemarkeerd. De scène wordt dan gereconstrueerd door voxels 'weg te hakken', net zoals een beeldhouwer uit een blok steen een beeld 'hakt'. Het weghakken van voxels gebeurt door herhaaldelijk het '*plane-sweep*' [Col96] algoritme toe te passen.

Het plane-sweep algoritme maakt de veronderstelling dat de camera's en de voxels door een vlak gescheiden kunnen worden. Hierdoor kan gemakkelijk een topologische volgorde ten opzichte van de camera's gedefiniëerd worden op de voxels. De voxels kunnen namelijk in toenemende afstand tot dat vlak doorlopen worden. Hierdoor wordt gegarandeerd dat alle voxels die de zichtbaarheid van de huidige pixel kunnen belemmeren reeds geëvalueerd zijn.

Space carving gebeurt door het plane-sweep algoritme herhaaldelijk toe te passen, telkens gebruik makend van een ander scheidingsvlak. Aan de hand van dit vlak wordt een subset van de camera's, en een subset van de voxels geselecteerd waarmee het algoritme wordt uitgevoerd. De scheidingsvlakken kunnen bijvoorbeeld de vlakken van de gediscretiseerde ruimte waarin de scène wordt opgebouwd (meestal een kubus) zijn.

2.3 Energie minimalisatie

Vooraleer de energie-minimalisatie algoritmes toe te lichten, wordt eerst uitgelegd hoe de maximale stroom in een gewogen gerichte graaf gevonden kan worden. Dit wordt later gebruikt bij het minimaliseren van de energie-formulatie van het scène-reconstructie probleem.

2.3.1 Maximale stoom, minimale snede

Een gewogen gerichte *graaf* $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ bestaat uit een set van *knopen* $\mathcal{V}$ en een set van gewogen *takken* $\mathcal{E}$ die de knopen verbinden. Takken zijn koppels van knopen: $\mathcal{E} = \{(p, q) \mid p, q \in \mathcal{V}\}$. Merk op dat (p, q) en (q, p) twee verschillende elementen zijn, waardoor de graaf een *gerichte* graaf is. Het gewicht van een tak wordt gedefiniëerd door een functie $c : \mathcal{E} \mapsto \mathcal{W}$, waarbij $\mathcal{W}$ de verzameling van de mogelijke gewichten is.

Een *terminale* knoop is een knoop waar alleen maar takken uit weggaan of alleen maar takken in aankomen. Een knoop waar alleen maar takken uit weggaan, is een *bronknoop* of '*source node*'. Een knoop waar alleen maar takken in aankomen, is een *putknoop* of '*sink node*'. Een s/t *snede* C is een partitie van alle knopen in twee disjuncte deelverzamelingen $\mathcal{S}$ en $\mathcal{T}$, zodat de bronknoop $s \in \mathcal{S}$ en de putknoop $t \in \mathcal{T}$. Voor de eenvoud wordt een s/t snede hier simpelweg *snede* genoemd. Het *gewicht* van een snede $C = \{\mathcal{S}, \mathcal{T}\}$ is de som van de

gewichten van alle takken (p, q) waarbij $p \in \mathcal{S}$ en $q \in \mathcal{T}$. Merk op dat alleen de gewichten van takken die in de richting van de put lopen, gebruikt worden.

Het *maximale stroomprobleem* bestaat uit het vinden van een zo groot mogelijke stroom doorheen een graaf, volgens de richting van de takken, van een gegeven beginknoop naar een gegeven eindknoop, waarbij de gewichten van de takken een bovengrens voor de stroom bepalen. Een *stroom* is dus een toewijzing van niet-negatieve waarden aan de takken van de graaf, zodanig dat er in elke knoop behoud van stroom is. Dit wil zeggen dat de totale stroom die in een knoop binnenvloeit gelijk moet zijn aan de totale stroom die uit de knoop wegvloeit. De gewichten van de takken worden in deze context ook wel de *capaciteiten* van de takken genoemd.

Het maximale stroomprobleem kan opgelost worden door de snede met het minimale gewicht te vinden. Het theorema van Ford en Fulkerson [JF62] zegt dat, als de maximale stroom doorheen de graaf van s naar t gestuurd wordt, de set van takken die verzadigd geraken, de knopen van de graaf in twee disjuncte deelverzamelingen $\{\mathcal{S}, \mathcal{T}\}$ verdeelt, overeenstemmend met de minimale snede: De snede met het kleinste gewicht van alle snedes. De snede wordt in dit geval dus gedefiniëerd door de set van takken die 'doorgesneden' worden; Ofwel: Alle takken die een knoop uit $\mathcal{S}$ met een knoop uit $\mathcal{T}$ verbinden en vice versa.

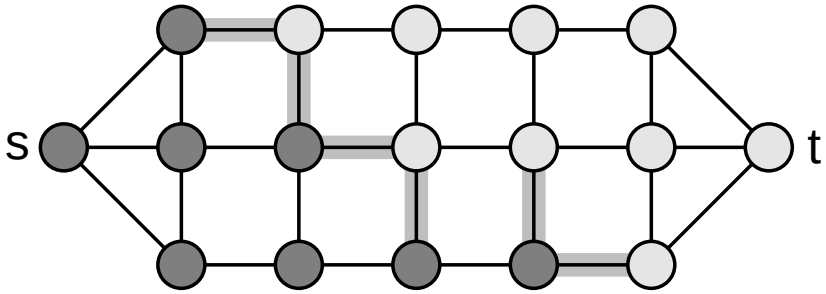
Dit kan ook als volgt bekeken worden. Iedere mogelijke snede geeft een bovengrens voor de maximale stroom doorheen de graaf. Deze bovengrens is gelijk aan de som van de capaciteiten van de takken waarlangs de stroom in de richting van de put kan wegvloeien; Het gewicht van de snede dus. De snede met het kleinste gewicht, de minimale snede, geeft de kleinste bovengrens. De maximale stroom die doorheen een graaf gestuurd kan worden is gelijk aan het gewicht van de minimale snede. Dit wordt geïllustreerd in figuur 2.3.

2.3.2 Probleem formulatie

Stel dat er n beelden gegeven zijn, die dezelfde scène, op hetzelfde moment, maar vanuit verschillende standpunten weergeven. $\mathcal{P}_i$ is de set van alle pixels in beeld i, en $\mathcal{P} = \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n$ de set van alle pixels. Een pixel $p \in \mathcal{P}$ komt overeen met een rechte in de drie-dimensionale ruimte. Beschouw nu de eerste intersectie van deze rechte met een object uit de scène. Het doel van deze methode is de diepte van dit punt voor de overeenkomstige pixels in de verschillende beelden te vinden. Dit komt neer op het zoeken van een functie $f : \mathcal{P} \mapsto \mathcal{L}$, waarbij $\mathcal{L}$ een discrete set van mogelijke dieptes is.

Het scène-reconstructie probleem komt dus neer op het zoeken van een functie f die aan iedere pixel een diepte toekent. Deze functie moet zo gekozen worden dat de foto-consistentie en de spatiale coherentie maximaal zijn. Dit wordt verwezenlijkt aan de hand van een energie-functie die aan iedere mogelijke definitie van de functie f een energiewaarde toekent. Kandidaten die grote inconsistenties veroorzaken, krijgen een hoge energie-waarde. Er wordt dus gezocht naar een invulling van de functie f die de toegekende energie minimaliseert.

Het minimaliseren van een energie-functie is een NP-probleem. Er zijn echter een hoop energie-minimalisatie algoritmes ontwikkeld, gebaseerd op snedes in



Figuur 2.4: *Vereenvoudigde versie van de graaf om de dieptes van de pixels te berekenen. De afbeelding waarvoor de dieptes berekend worden is in dit geval een één-dimensionale rij van 3 pixels. De takken die de snede bepalen zijn door de brede lijnen aangeduid. Horizontale takken in de snede bepalen de diepte van de pixel, en hebben dus capaciteiten die afhangen van de foto-consistentie van het punt dat overeenkomt met deze pixel en deze diepte. Vertikale takken in de snede duiden op minder spatiale coherentie. De twee onderste vertikale takken bijvoorbeeld komen overeen met een verschil van twee in diepte tussen de twee onderste pixels. Om spatiale coherentie te bekomen, wordt aan de vertikale takken simpelweg een constante capaciteit toegekend.*

capaciteit die afhangt van de foto-consistentie van de overeenkomstige pixel. Takken met een constante d-waarde krijgen een constante capaciteit K. Dit wordt geïllustreerd in figuur 2.4. De gevonden diepte voor iedere pixel is hier telkens de meest rechtse, donkere knoop op de overeenkomstige rij.

Hoofdstuk 3

Video compressie

Dit hoofdstuk begint met een korte introductie in digitale video en datacompressie, gevolgd door een bespreking van enkele veelgebruikte compressietechnieken. Daarna wordt uitgelegd hoe deze technieken gecombineerd kunnen worden om videosequenties te comprimeren. In de volgende paragraaf wordt de werking van een typisch videocompressie systeem echter al in grote lijnen toegelicht, zodat de later besproken compressietechnieken beter gesitueerd kunnen worden.

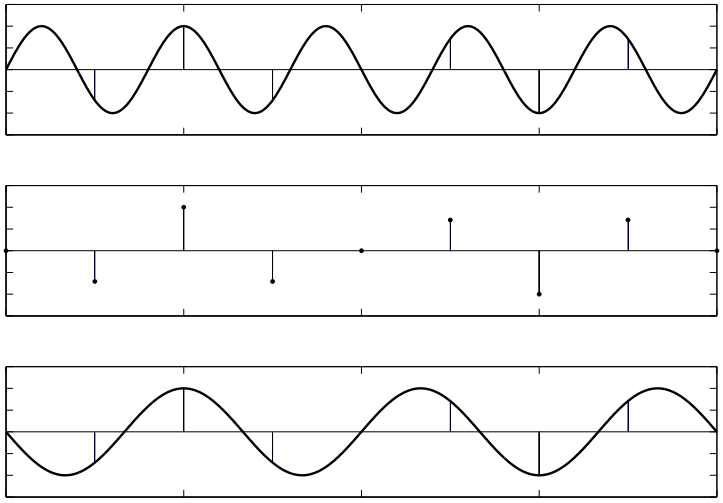
Een select aantal beelden, verspreid over de volledige videosequentie, wordt onafhankelijk van andere beelden gecodeerd door middel van beeldcompressietechnieken. Deze beelden worden gebruikt als referentie beelden om de overige, tussenliggende beelden te coderen. Deze tussenliggende beelden worden telkens voorspeld uitgaande van de informatie in de reeds gecodeerde referentie beelden. Hierdoor volstaat het enkel de informatie die nodig is om de voorspelling correct uit te voeren te coderen.

Een referentie beeld wordt als volgt gecodeerd: Eerst wordt de data door middel van de discrete cosinus transformatie herschreven in een vorm die beter geschikt is voor compressie. Daarna wordt het aantal mogelijke waarden van de bekomen coëfficiënten beperkt. Coëfficiënten die veel bijdragen tot de kwaliteit van het beeld krijgen meer mogelijke waarden toegekend dan coëfficiënten die minder bijdragen. Het resultaat wordt dan door middel van entropie codering met zo weinig mogelijk bits voorgesteld.

3.1 Digitale video

Video is de elektronische representatie van bewegende beelden. Bij *digitale* video wordt er gewerkt met discrete waarden, in tegenstelling tot analoge video waar een continu spectrum van waarden gebruikt wordt.

sterde versie is weergegeven in de middelste grafiek. Het is duidelijk dat deze monsters de vorm van de grafiek niet precies genoeg beschrijven. Reconstructie van een sinusgolf uit deze samples zou leiden tot de onderste grafiek in figuur 3.2: dit is een sinusgolf met een frequentie van 3 Hz, in plaats van 5 Hz. Beide functies komen perfect overeen met de monsters. Doordat de functie met een te lage frequentie bemonsterd werd, is de reconstuctie een golf met een andere frequentie. Dit fenomeen noemt men *vouwvervorming*, of *aliasing*.



Figuur 3.2: *De bovenste grafiek is een sinusgolf met een frequentie van 5 Hz, bemonsterd met een frequentie van 8 Hz. De middelste grafiek toont de resulterende monsters. De onderste grafiek toont een sinusgolf met een frequentie van 3 Hz, ook bemonsterd aan 8 Hz, die net dezelfde monsters oplevert als de bovenste grafiek.*

In het algemeen geldt dat voor ieder signaal met frequentie f_1, bemonsterd met frequentie f_s, een ander signaal met frequentie $f_2 = f_s - f_1$ geconstrueerd kan worden dat exact dezelfde monsters oplevert indien het bemonsterd wordt met frequentie f_s.

Merk op dat indien $f_1 < \frac{f_s}{2}$, de frequentie van het tweede signaal $f_2 > \frac{f_s}{2}$. Als de bemonsteringsfrequentie dus hoog genoeg gekozen wordt, zodat de frequentie van het te bemonsteren signaal kleiner is dan half de bemonsteringsfrequetie, kan het originele signaal steeds gereconstrueerd worden; dit is namelijk de variant met de laagste frequentie. Deze stelling noemt men het bemonsteringstheorema van Nyquist [Nyq28]:

Stelling 3.1 *Om een signaal te kunnen reconstrueren moet de bemonsteringsfrequentie minstens twee keer zo hoog zijn als de frequentie van het bemonsterde signaal.*

De frequentie gelijk aan half de bemonsteringsfrequentie heet dan de *Nyquist-frequentie*. Voor een foutloze reconstructie mag het bemonsterde signaal dus geen frequenties bevatten die de Nyquist-frequentie overschrijdt. Een signaal met een frequentie die hoger is dan de Nyquist-frequentie zal 'teruggevouwen' worden, resulterend in een signaal met een frequentie die lager is dan de Nyquist-frequentie. dit verklaart de term 'vouwvervorming'.

Vouwvervorming kan gemakkelijk geïllustreerd worden aan de hand van een analoge[1] klok, waarvan op regelmatige tijdstippen een foto genomen wordt. Aangezien de grote wijzer om het uur een volledige cirkel – de periode – aflegt, is de frequentie dus 1 periode per uur. De kleine wijzer maakt 1 periode per 12 uur.

Stel nu dat er om de 45 minuten een foto genomen wordt. De kleine wijzer is dan telkens $\frac{3}{4}$ van de hoek tussen twee opeenvolgende uren gedraaid. De grote wijzer legt telkens $\frac{3}{4}$ van de volledige cirkel af. Als de foto's achtereen getoond worden lijkt het echter alsof de grote wijzer steeds een kwart van de cirel aflegt in tegenwijzerzin. De Nyquist-frequentie is hier overschreden: $\frac{3}{4} > \frac{1}{2}$ periode per uur, en voor onze ogen lijkt de grote wijzer te draaien aan de teruggevouwen frequentie, namelijk $1-\frac{3}{4} = \frac{1}{4}$ periode per uur in de tegenovergestelde frequentie.

Merk op dat volgens Nyquist de frequentie van het bemonsterde signaal strikt kleiner moet zijn dan de Nyquist-frequentie. Als er om het half uur een foto genomen wordt, legt de grote wijzer telkens een halve cirkel af, precies de Nyquist-frequentie. In dit geval is het onmogelijk te zeggen in welke richting de grote wijzer draait. De teruggevouwen frequentie is gelijk aan de originele frequentie, maar in de tegenovergestelde richting. Dit voorbeeld is gebaseerd op een gelijkaardige uitleg in [Sym04].

Vouwvervorming is vaak duidelijk zichtbaar als een draaiend wiel van een auto gefilmd wordt. Het wiel lijkt dan soms in de tegenovergestelde richting te draaien. Dit fenomeen is gelijkaardig aan het voorbeeld van de klok.

3.1.2 Kwantisatie

Een afbeelding kan voorgesteld worden als een discrete twee-dimensionale functie $f(x, y)$. Voor iedere waarde van y worden telkens alle waarden voor een variërende x opgemeten. Zo wordt het beeld dus in twee dimensies bemonsterd. De functie $f(x, y)$ beeldt twee pixel coördinaten af op een kleur. Om video te representeren moet f uitgebreid worden naar drie dimensies: de tijdsdimensie wordt toegevoed. $f(t, x, y)$ is dan de kleur van pixel (x, y) op tijdstip t.

Door bemonstering kunnen deze drie dimensies dus discreet voorgesteld worden in plaats van continu. De mogelijke waarden van $f(t, x, y)$ kunnen dan echter nog steeds variëren over het volledige kleurenspectrum. Om dit te kunnen voorstellen in een digitale wereld, moeten de mogelijke waarden van de functie voor ieder monster gelimiteerd worden tot een vooraf bepaald, eindig aantal toegelaten waarden. Dit proces heet *kwantisatie*.

[1] Een analoge klok is een klassieke klok met wijzers; dit in tegenstelling tot een digitale klok, waar de tijd weergegeven wordt door cijfers.

3.2 Datacompressie

Datacompressie is het verminderen van de hoeveelheid data waarmee informatie wordt voorgesteld. Hierbij wordt gebruik gemaakt van het feit dat informatie van nature niet willekeurig is, maar regelmaat en patronen vertoont. De informatie kan in een andere vorm worden herschreven, waarbij overbodige data weggelaten wordt. Aangezien datacompressie specifieke kenmerken van de informatie uitbuit, hangt de gepaste compressietechniek dan ook af van de aard van de informatie.

Een nederlandstalige tekst bijvoorbeeld is niet zomaar een willekeurige aaneenschakeling van letters. Als een woord begint met de letters 'sc' is de kans groot dat hierna de letter 'h' volgt. Het is dus mogelijk een voorspelling te maken over een deel van de informatie op basis van een herkend patroon. Ontdekking van regelmaat en patronen leidt tot 'redundante' data: de data is niet noodzakelijk om de informatie over te brengen.

Ook in de digitale representatie van een tekst is er ruimte voor compressie. In standaard tekstbestanden worden de verschillende letters door unieke lettercodes met dezelfde bitlengte voorgesteld. In de nederlandse taal komen sommige letters echter veel vaker voor dan andere. Door aan vaak voorkomende tekens een kortere bitcode toe te kennen dan aan minder vaak voorkomende, wordt dezelfde informatie voorgesteld met minder bits.

De nood aan datacompressie heeft twee belangrijke oorzaken: beperkte opslagruimte en beperkte bandbreedte bij transmissie. Beide problemen komen in dit geval op hetzelfde neer: net zoals er een kost vasthangt aan het versturen van een bit, hangt er een kost aan het opslaan van een bit in het geheugen. Hoe complexer de informatie, hoe meer data er nodig is voor de representatie. Een standaard tekstbestand kan voorgesteld worden met 8 bits/karakter, of ongeveer 20 kbits per pagina. Muziek van CD kwaliteit vereist ongeveer 1500 kbits/seconde en PAL video, de europese televisie standaard, heeft ongeveer 158 Mbits/seconde nodig.

3.2.1 Exact omkeerbaar en niet-exact omkeerbaar

Soms is het niet noodzakelijk alle informatie te coderen. Irrelevante of bijna irrelevante informatie weglaten leidt uiteraard ook tot minder data. Dit heet *niet-exact omkeerbare compressie*, of verlieslatende compressie, van het Engelse 'lossy compression'. In dit geval is het niet mogelijk de oorspronkelijke data weer te bekomen uit de gecodeerde versie. Als er daarintegen geen informatie verloren gaat, spreekt men over *exact omkeerbare compressie*, of verliesloze compressie, van het Engelse 'losless compression'.

Het is bijvoorbeeld niet zinnig de kleureninformatie van een videofragment door te sturen als de ontvanger enkel over een zwart-wit scherm beschikt. De weggelaten informatie is in dit geval volledig irrelevant voor de ontvanger. Ook het weglaten van informatie die wel betekenisvol is, is een belangrijke compressietechniek. De te elimineren informatie moet zo gekozen worden dat het kwaliteitsverlies aanvaardbaar is in vergelijking met de uitsparing aan data.

Voor sommige applicaties is elk kwaliteitsverlies onaanvaardbaar: een tekstbestand of een binair bestand, zoals een computerprogramma, vereisen exact-omkeerbare compressie. Bij afbeeldingen, geluid of videobestanden kan er echter een veel hogere graad van compressie bereikt worden door de minst essentiële data weg te laten, wat resulteert in kwaliteitsverlies.

Het selecteren van deze bijna irrelevante informatie vereist kennis van de manier waarop de informatie waargenomen wordt door de ontvanger. Een videofragment met beperkt kwaliteitsverlies kan net dezelfde informatie overbrengen als het originele videofragment. Nochtans zijn er bij de compressie gegevens verloren gegaan. Dit komt doordat de informatie die *waargenomen* wordt wel hetzelfde gebleven is. Kleine veranderingen in het beeld storen niet of worden zelfs helemaal niet opgemerkt.

3.2.2 Symmetrische en asymmetrische systemen

Om de originele data weer te verkrijgen na compressie, moet de gecomprimeerde data eerst *gedecomprimeerd* worden. Dit proces voert dus de inverse bewerking van de compressie uit. Bij *symmetrische* systemen zijn beide processen even complex. Een voorbeeld hiervan is 'videoconferencing': om bruikbaar te zijn moet het systeem een minimum aan bandbreedte gebruiken, en de installatie voor verzenden en ontvangen mag niet te duur zijn. Door de beperkte bandbreedte is compressie noodzakelijk, en beide kanten moeten zowel kunnen comprimeren als decomprimeren. Het levert geen voordeel op het compressie systeem complexer of minder complex te ontwerpen dan het decompressie systeem.

Systemen waar de data eenmalig gecomprimeerd wordt, en dan verschillende keren gedecomprimeerd, zijn *asymmetrische* systemen. Denk bijvoorbeeld aan films op DVD: dezelfde gecomprimeerde versie wordt door miljoenen DVD spelers gedecomprimeerd. In dit geval is het interessanter een complexer of duurder compressie systeem te ontwerpen ten voordele van betere kwaliteit of eenvoudigere decompressie.

3.3 Entropie codering

Om een binair bestand juist te interpreteren is er kennis van de manier van coderen vereist. Meestal wordt elk symbool met even veel bits gecodeerd. Hierdoor is het gemakkelijk te zien wanneer de code van een nieuw symbool begint.

Entropie codering is een exact omkeerbare compressietechniek die aan ieder symbool een code toekent waarvan de lengte afhangt van de waarschijnlijkheid dat het symbool zal voorkomen. Denk aan het voorbeeld van het tekstbestand waarin aan vaak voorkomende tekens een kortere bitcode toegekend wordt dan aan minder vaak voorkomende.

Een bekend voorbeeld van entropie codering is de Morsecode, ontwikkeld in 1835 door Samuel Morse om door middel van telegrafie letters, leestekens en cijfers te verzenden. De kortste codes zijn voorbehouden voor de letters die het vaakst vorkomen.

De drie meest voorkomende entropie coderingstechnieken zijn Huffman codering, 'range encoding' en aritmetische codering. Nadat de begrippen *informatie* en *entropie* toegelicht zijn, volgt een bespreking van deze technieken. Het nut van entropie codering in video compressie komt hier nog niet aan bod. Dit zal in een latere sectie toegelicht worden.

3.3.1 Het begrip informatie

In 1948 definieerde Shannon [Sha48] de informatie geleverd door een gebeurtenis in functie van de waarschijnlijkheid van die gebeurtenis. Stel een bron $S = \{s_1, s_2, \ldots, s_m\}$ is een eindige verzameling van onafhankelijke gebeurtenissen. Een gebeurtenis s_i heeft waarschijnlijkheid p_i, waarbij $0 \leq p_i \leq 1$ en $\sum_{i=1}^{m} p_i = 1$. Neem als bron bijvoorbeeld een eerlijke dobbelsteen: de mogelijke worpen zijn de verschillende gebeurtenissen $s_1, s_2, \ldots, s_6$ waarbij $p_i = 1/6$ voor iedere i. De waarschijnlijkheid van iedere mogelijke worp is onafhankelijk van vorige gebeurtenissen en de som van de waarschijnlijkheden is gelijk aan 1.

Definitie 3.1 *De informatie $I(s_i)$ geleverd door een gebeurtenis s_i is gelijk aan*

$$I(s_i) = \log_2\left(\frac{1}{p_i}\right)$$

Aangezien $0 \leq p_i \leq 1$ zal $0 \leq I(s_i) \leq \infty$. Merk ook op dat $I(s_i) = 0$ als $p_i = 1$ en dat $I(s_i) = \infty$ als $p_i = 0$. Uit definitie 3.1 volgt dat hoe lager de waarschijnlijkheid van een gebeurtenis, hoe meer informatie er geleverd wordt door die gebeurtenis. Dit komt overeen met de intuïtie dat onverwachte gebeurtenissen ons meer vertellen. Een voorspelbare gebeurtenis vertelt ons immers niets nieuws. Verrassing vertaalt zich dus in informatie. Met het oog op compressie is het dus logisch om aan overwachte gebeurtenissen meer plaats toe te kennen in de informatiedrager.

De verzameling van elke mogelijke twee opeenvolgende gebeurtenissen is $S^2 = S \times S = \{(s_i, s_j) \,|\, i, j \in \{1, \ldots, m\} \wedge s_i, s_j \in S\}$. Volgens de principes van de kansrekening is de waarschijnlijkheid van een gebeurtenis (s_i, s_j) gelijk aan het product van de waarschijnlijkheden van de afzonderlijke gebeurtenissen: $p_{i,j} = p_i p_j$. Hieruit volgt:

$$\begin{aligned} I((s_i, s_j)) &= \log_2\left(\frac{1}{p_i p_j}\right) \\ &= \log_2\left(\frac{1}{p_i}\right) + \log_2\left(\frac{1}{p_j}\right) \\ &= I(s_i) + I(s_j) \end{aligned}$$

De informatie geleverd door twee opeenvolgende gebeurtenissen is dus gelijk aan de som van de hoeveelheden informatie geleverd door de twee gebeurtenissen afzonderlijk. Dit is wat men zou verwachten van een eenheid die de hoeveelheid informatie uitdrukt, en verklaart het gebruik van de logaritme in de definitie: het product van de kansen vertaalt zich in de som van de hoeveelheden informatie.

Informatie wordt uitgedrukt in bits. De hoeveelheid informatie geleverd door s_i met $p_i = \frac{1}{2}$ is dus gelijk aan 1 bit. Een gebeurtenis s_i met $p_i = \frac{1}{2^k}$ levert k bits informatie. Dit komt overeen met de vertrouwde betekenis van bits. Er zijn 2^k mogelijke bitcodes met lengte k. Er zijn dus k bits nodig om aan te duiden welke van de 2^k gebeurtenissen met gelijke kansen heeft plaatsgevonden.

3.3.2 Entropie

Entropie is een maat voor wanorde of onvoorspelbaarheid. Het wordt gedefinieerd als de verwachtingswaarde van de informatie I van een eindige verzameling onafhankelijke gebeurtenissen S.

Definitie 3.2 *De entropie $H(S)$ van een eindige verzameling onafhakelijke gebeurtenissen $S = \{s_1, s_2, \ldots, s_m\}$ is gelijk aan*

$$H(S) = \sum_{i=1}^{m} p_i I(s_i) = \sum_{i=1}^{m} p_i \log_2 \left(\frac{1}{p_i}\right)$$

waarbij p_i de waarschijnlijkheid van gebeurtenis s_i is, met $0 \leq p_i \leq 1$ en $\sum_{i=1}^{m} p_i = 1$.

Volgend voorbeeld verduidelijkt de betekenis van definitie 3.2. Beschouw een sequentie van symbolen uit het alfabet $S = \{a, b, c, d\}$. Stel dat elk symbool evenveel kans heeft om voor te komen in de sequentie. Ieder voorkomen van een symbool geeft dus $I(s_i) = \log_2 \left(\frac{1}{1/4}\right) = \log_2 4 = 2$ bits informatie. Dit is samengevat is tabel 3.1. Aangezien de entropie gedefinieerd is als de gemiddelde hoeveelheid informatie geleverd door een gebeurtenis, is $H(S)$ ook gelijk aan 2.

s_i	p_i	$I(s_i)$
a	0.25	2
b	0.25	2
c	0.25	2
d	0.25	2

Tabel 3.1: *De symbolden s_i, de waarschijnlijkheden van voorkomen p_i, en de informatie voortgebracht bij voorkomen $I(s_i)$ voor een alfabet met uniforme kansverdeling.*

Stel nu dat sommige symbolen vaker voorkomen dan andere. Tabel 3.2 toont voor ieder symbool van het alfabet de waarschijnlijkheid dat het voorkomt in de sequentie, samen met de informatie die uit deze gebeurtenis voortvloeit. De resulterende entropie is gelijk aan:

$$H(S) = 0.7 \times 0.5146 + 0.1 \times 3.3219 + 0.1 \times 3.3219 + 0.1 \times 3.3219 = 1.3568$$

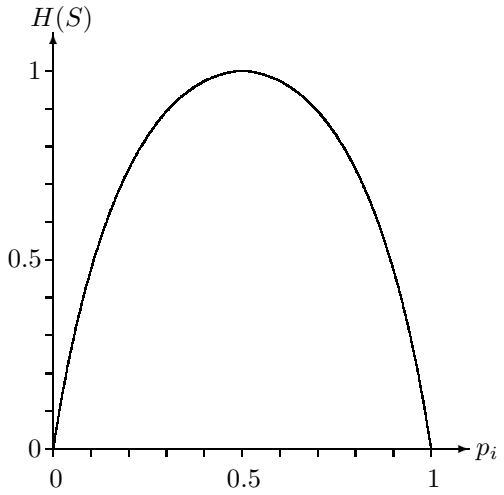
Hoewel b, c en d in het tweede voorbeeld veel informatie opleveren, dragen ze weinig bij tot de berekening van de entropie. Symbool a heeft daarentegen veel invloed op $H(S)$, maar geeft weinig informatie. Aangezien het waarschijnlijk is

s_i	p_i	$I(s_i)$
a	0.7	1.5146
b	0.1	3.3219
c	0.1	3.3219
d	0.1	3.3219

Tabel 3.2: *De symbolden s_i, de waarschijnlijkheden van voorkomen p_i, en de informatie voortgebracht bij voorkomen $I(s_i)$ voor een alfabet met niet-uniforme kansverdeling.*

dat a vaak zal voorkomen in een sequentie, levert dit weinig informatie op, en is het alfabet van het tweede voorbeeld voorspelbaarder. Het voorkomen van b, c of d zorgt voor een grotere verrassing, maar deze verrassingen vinden niet vaak genoeg plaats om veel invloed te hebben op de entropie. Een lagere entropie komt dus overeen met hogere voorspelbaarheid.

Merk op dat voor $S = \{s_1, s_2\}$ geldt dat $p_2 = (1 - p_1)$, dus $H(S)$ kan in functie van enkel p_1 uitgedrukt worden. Figuur 3.3 toont $p_1 \mapsto H(S)$, de entropie voor alle mogelijke verwachtingen van twee gebeurtenissen. De entropie is het grootst wanneer beide verwachtingen even groot zijn. Naarmate de verwachtingen meer verschillen wordt de entropie kleiner. Dit geldt ook voor het algemene geval met m mogelijke gebeurtenissen. Het absolute maximum is dan $\log_2\left(\frac{1}{1/m}\right) = \log_2 m$.



Figuur 3.3: $p_1 \mapsto H(S)$ *met* $H(S) = p_1 \log_2\left(\frac{1}{p_1}\right) + (1 - p_1)\log_2\left(\frac{1}{1-p_1}\right)$ *voor verschillende waarden van* p_1.

Met behulp van een entropie kan er een ondergrens bepaald worden voor exact omkeerbare compressie. Volgens Shannon [Sha48] is het gemiddeld aantal bits per symbool van de efficiëntst mogelijke codering groter dan of gelijk aan de

entropie van de bron.

Neem als bron opnieuw het alfabet uit tabel 3.1. De vier verschillende symbolen kunnen gecodeerd worden door telkens een bitcode van lengte 2 te gebruiken. Dit resulteert in een gemiddelde van 2 bits per symbool, wat exact evenveel is als de entropie. Een efficiëntere codering is hier dus niet mogelijk. Voor tabel 3.2 is de entropie slechts 1.3568. Hier is dus nog wel ruimte voor verbetering. Beschouw de bitcodes in tabel 3.3. Het gemiddeld aantal bits per symbool voor deze codering is $0.7 \times 1 + 0.1 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 1.5$. Hoewel dit nog steeds meer is dan de ondergrens, levert deze codering toch een aanzienlijke compressie op.

symbool	bitcode
a	0
b	10
c	110
d	111

Tabel 3.3: *Bitcodes voor het alfabet uit tabel 3.2*

Merk op dat de codering exact omkeerbaar moet zijn. Dit houdt in dat iedere gecodeerde sequentie ondubbelzinnig terug omgezet moet kunnen worden in de originele symbolen. Voor tabel 3.3 is dit inderdaad het geval. Beschouw volgende sequentie van symbolen uit het alfabet:

$$aacabaaaaadadaacaaab \tag{3.1}$$

Als we (3.1) coderen volgens de bitcodes in tabel 3.3 krijgen we:

$$00110010000001110110011000010 \tag{3.2}$$

Om de originele informatie te herstellen, moet sequentie (3.2) van links naar rechts doorlopen worden. Vanaf het moment dat de ingelezen bits overeenkomen met een bitcode uit tabel 3.2, worden deze vervangen door het bijhorende symbool. Dit resulteert opnieuw in sequentie (3.1). Merk op dat sequentie (3.2), bestaande uit 20 symbolen, gecodeerd wordt in 30 bits. Dit komt inderdaad overeen met 1.5 bits/symbool. Het aantal voorkomens van ieder symbool in de originele sequentie kwam dan ook perfect overeen met de vooropgestelde waarschijnlijkheden.

Maar kan het nu nog efficiënter? 1.5 bits/symbool is nog steeds meer dan de ondergrens van 1.3568 bits/symbool. Deze ondergrens zou bereikt kunnen worden door aan ieder symbool s_i een bitcode met lengte $I(s_i)$ toe te kennen. De gemiddelde code lengte zou dan gelijk zijn aan de gemiddelde hoeveelheid informatie, wat gelijk is aan de entropie. Het probleem is dat $I(s_i)$ vaak geen natuurlijk getal is, en het aantal bits in een code wel natuurlijk moet zijn. Hierdoor wordt de theoretische ondergrens dus niet altijd bereikt.

3.3.3 Huffman codering

In 1952 ontwikkelde David Huffman [Huf52] het naar hem genoemde entropie coderingsalgoritme. Het probleem met codes van variabele lengte is dat het

steeds mogelijk moet zijn de verschillende codes van elkaar te onderscheiden. Huffman bedacht een manier om een set van codes te genereren die nooit elkaars prefix zijn: als een sequentie bits herkend wordt als een code voor een symbool, is dit meteen het juiste symbool. De sequentie kan niet een deel van een nog langere bitcode zijn. Hiervoor moet de tabel van symbolen en bijbehorende codes natuurlijk wel bekend zijn bij de het reconstructieproces.

Algoritme

Het algoritme gaat ervan uit dat de symbolen van de te coderen sequentie bekend zijn, en voor ieder symbool ook het aantal voorkomens bekend is.

1. Maak voor ieder symbool een binaire boomstructuur, bestaande uit slechts één knoop[2], gelabeld met het overeenkomstige symbool. Associëer met iedere boom een gewicht, initiëel gelijk aan de frequentie van het symbool.
2. Voeg nu telkens de twee bomen met de laagste gewichten samen, door ze als deelbomen onder een niewe knoop te hangen. De boom met het laagste gewicht wordt de linker deelboom, de andere de rechter. De nieuwe boom krijgt als gewicht de som van de twee deelbomen. Herhaal dit tot er slechts één boom overblijft.
3. Het resultaat is een binaire boom waarvan de bladeren gelabeld zijn met de symbolen. De bitcode van ieder symbool wordt opgesteld door het pad van de wortel van de boom tot het blad te doorlopen: een vertakking naar het linkerkind levert een 0 op, een vertakking naar het rechterkind een 1.

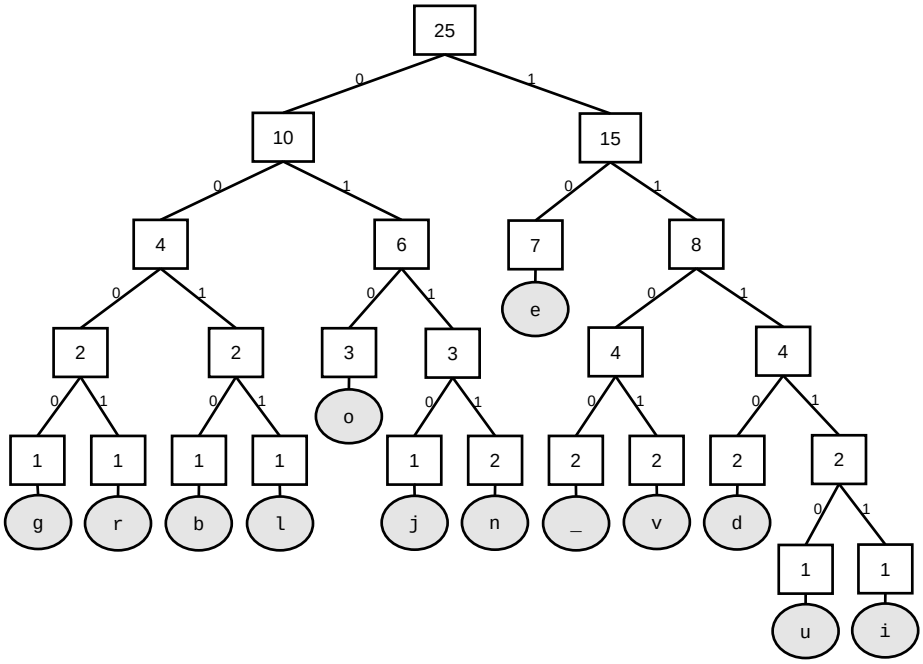
Beschouw bijvoorbeeld de sequentie "*een eenvoudig voorbeeldje*". Figuur 3.4 toont de volledige Huffman boom voor deze sequentie. In de eerste stap van het algoritme wordt voor iedere letter een binaire boom aangemaakt. Dit resulteert in 13 bomen, met gewichten variërend van 1 tot 7.

In stap 2 worden de bomen met de laagste gewichten geselecteerd. Neem bijvoorbeeld de bomen onstaan uit de letters 'g' en 'r'; beide komen slechts één keer voor in de sequentie. Deze worden samengevoegd door ze als kinderen onder een nieuwe knoop te hangen. Hierdoor blijven er nog maar 12 bomen over, waarvan er 11 bestaan uit slechts één knoop, en één boom bestaande uit 3 knopen. De nieuwe boom krijgt als gewicht 2, namelijk de som van de gewichten van de twee bomen.

Ook 'b', 'l', 'u' en 'i' komen slechts één keer voor, en worden dus op dezelfde wijze gehandeld. Dit resulteert in 7 bomen van telkens één knoop, en 3 bomen met ieder drie knopen. Hierna blijft er nog maar één boom met kost 1 over, namelijk dewelke uit de letter 'j' onstaan is. Deze wordt dus samengevoegd met een boom die kost 2 heeft, bijvoorbeeld de boom resulterend uit de letter 'n'.

De boom geconstrueerd uit 'g' en 'r', en die uit 'b' en 'l' hebben beide kost 2. Ze hebben dus de laagste kosten en moeten samengevoegd worden. Dit resulteert

[2]Knopen zijn elementen waaruit de boom, samen met vertakkingen, opgebouwd is. Zowel de wortel, als de interne knopen, als de bladeren van de boom zijn allemaal knopen.



Figuur 3.4: *De Huffman boomstructuur gegenereerd uit de sequentie "een eenvoudig voorbeeldje". De rechthoekjes bevatten telkens het gewicht van de onderliggende deelboom. De ellipsen zijn de labels van de bladeren van de boom.*

in een boom met kost 4, bestaande uit $3 + 3 + 1 = 7$ knopen. Als dit proces blijft herhaald worden, ontstaat er uiteindelijk de boom in figuur 3.4.

De vertakkingen worden gelabeld met de binaire symbolen zoals beschreven in stap 3. Om nu de bitcode voor bijvoorbeeld de letter 'o' te verkrijgen, volstaat het de boom vanaf de wortel tot aan het overeenkomstige blad te doorlopen. Eerst wordt dus de linker vertakking gekozen, wat resulteert in een 0. Dan de rechter, waardoor de bitcode aangevuld wordt met een 1. Ten slotte wordt nog eens de linker vertakking gekozen, zodat de uiteindelijke bitcode 010 wordt. Op dezelfde manier wordt voor de letter 'u' de bitcode 11110 verkregen.

Voor twee bitcodes gegenereerd door dit algoritme geldt dat de ene nooit een prefix van de andere kan zijn. Dit betekent dat er nooit een bitcode zal bestaan die gelijk is aan het eerste deel van een andere bitcode. Dit is noodzakelijk voor de decodering: als het gecodeerde bestand bit voor bit ingelezen wordt, is de eerste bitcode die voorkomt in de tabel met gegeneerde codes meteen de juiste code. Het kan niet voorkomen dat deze bitcode slechts het begin is van een nog langere code uit de tabel.

Deze eigenschap volgt uit de manier waarop de boom geconstrueerd is. Doordat twee bomen steeds samengebracht worden als kinderen van een nieuwe knoop, blijven de labels met de symbolen uit de sequentie steeds aan de bladeren van de boom zitten. Stel dat de code van een symbool 'a' een prefix is van de code van een symbool 'b', dan is het pad van de wortel van de boom tot aan de knoop

met label 'a' een deel van het pad van de wortel tot aan de knoop met label 'b'. De knoop met label 'a' is echter een blad. Er kan dus nooit een knoop met label 'b' onder knoop met label 'a' hangen. Hierdoor bestaat er dus geen symbool 'b' met een bitcode waarvan de code van 'a' een prefix is.

Het kan aangetoond worden dat de geconstrueerde boom optimaal is: er kan geen codeboom gevormd worden die een betere codering oplevert. Het gewogen gemiddelde van de code lengtes is dus minimaal bij het Huffman algoritme. Het bewijs hiervan is te vinden in [CP, Huf52].

3.3.4 Range encoding

Bij Huffman codering wordt de te coderen sequentie opgesplitst in de symbolen waaruit de sequentie opgebouwd is. Ieder symbool wordt dan vervangen door een codewoord. Bij '*Range encoding*' daarentegen, wordt de volledige sequentie in één getal gecodeerd. Ook *arithmetische codering* werk op deze manier. In tegenstelling tot arithmetische codering is range encoding echter vrij van patenten, en presteert het ongeveer even goed.

Gegeven de waarschijnlijkheden van de symbolen van het gebruikte alfabet, en een voldoende groot bereik van gehele getallen, werkt het range encoding algoritme als volgt. Het gegeven bereik wordt opgedeeld in deelbereiken, waarvan de groottes overeenkomen met de waarschijnlijkheden van de verschillende symbolen. Met ieder symbool komt dus een deelbereik overeen, waarvan de grootte ten opzichte van het volledige bereik gelijk is aan de waarschijnlijkheid van het symbool. Ieder symbool van de te coderen sequentie wordt dan om de beurt gecodeerd door het huidige bereik telkens te reduceren tot het bereik dat met dit symbool overeenkomt.

Als alle symbolen gecodeerd zijn, bevat het resulterende deelbereik genoeg informatie om de originele sequentie van symbolen opnieuw te genereren. Eén getal dat binnen dit bereik valt, bevat zelfs voldoende informatie, op voorwaarde dat op een of andere manier duidelijk gemaakt kan worden wanneer de volledige sequentie terug hersteld is.

s_i	p_i
a	0.6
b	0.2
\$	0.2

Tabel 3.4: *De symbolden s_i en de waarschijnlijkheden van voorkomen p_i voor een alfabet met een niet-uniforme kansverdeling. Het symbool '\$' wordt gebruikt om het einde van een sequentie aan te geven.*

Neem als voorbeeld het alfabet uit tabel 3.4. Het symbool '\$' wordt gebruikt om aan te geven dat het einde van de sequentie bereikt is. Het algoritme wordt nu geïllustreerd aan de hand van de sequentie "*aaba*\$" en het bereik $[0000000000, 1000000000)$[3]. In de eerste iteratie wordt dit volledige bereik dus als volgt opgedeeld in drie deelbereiken:

[3]De notatie $[x, y)$ wordt gebruikt om een half-open interval aan te duiden. Dit wil dus

a: $[0000000000, 0600000000)$
b: $[0600000000, 0800000000)$
\$: $[0800000000, 1000000000)$

Aangezien het eerste symbool van de sequentie een 'a' is, gaat het algoritme verder met als huidige bereik $[0000000000, 0600000000)$. Dit bereik wordt opnieuw onderverdeeld volgens de waarschijnlijkheden van de symbolen:

aa: $[0000000000, 0360000000)$
ab: $[0360000000, 0480000000)$
a\$: $[0480000000, 0600000000)$

Het volgende symbool in de sequentie is opnieuw een 'a', waardoor weer het eerste deelbereik gekozen wordt. Het verdere verloop gaat als volgt:

aaa: $[0000000000, 0216000000)$
aab: $[0216000000, 0288000000)$
aa\$: $[0288000000, 0360000000)$

$aaba$: $[0216000000, 0259200000)$
$aabb$: $[0259200000, 0273600000)$
aab\$: $[0273600000, 0288000000)$

$aabaa$: $[0216000000, 0241920000)$
$aabab$: $[0241920000, 0250560000)$
$abaa$\$: $[0250560000, 0259200000)$

Het resultaat van het algoritme is het bereik $[0250560000, 0259200000)$, wat dus overeenkomt met de sequentie "$abaa$\$". Het getal 0251000000 valt binnen dit bereik, en kan dus gebruikt worden om de originele sequentie te representeren.

De methode om die sequentie opnieuw te verkrijgen, uitgaande van dit getal, verloopt analoog. Eerst wordt het bereik $[0000000000, 1000000000)$ opgedeeld in drie deelbereiken die overeenstemmen met de waarschijnlijkheden van de drie symbolen. Aangezien 02510000000 binnen het eerste bereik ligt, het bereik van 'a' dus, wordt 'a' als eerste symbool van de sequentie gevonden. Dan wordt dit bereik opnieuw opgedeeld, en zo verder. Wanneer het symbool '\$' gevonden wordt, is de volledige sequentie opnieuw hersteld, en mag dus gestopt worden.

Het getal 0251000000 hoeft niet volledig gebruikt te worden om de sequentie te representeren. De eerste 0 mag weggelaten worden, aangezien alle mogelijke getallen binnen het bereik $[0000000000, 100000000)$ met een 0 beginnen. Doordat ook de lengte van het getal (in dit geval een lengte van 10, in decimale notatie) op voorhand bekend is, kunnen ook de achterste nullen weggelaten worden. De sequentie kan bijgevolg gecodeerd worden als het getal 251, wat overeenkomt met het binaire getal 11111011.

3.4 Voorspellende codering

In de vorige sectie werd de veronderstelling gemaakt dat de gebeurtenissen steeds onafhankelijk van elkaar zijn. Er werd dus verondersteld dat de waarschijn-

zeggen x nog wel tot het bereik hoort, maar y niet. Het gebruik van een half-open interval zorgt er in dit geval voor dat alle getallen, die binnen de sequentie liggen, met een 0 beginnen, waardoor deze niet gecodeerd moet worden.

lijkheid dat een symbool voorkomt niets te maken heeft met de voorgaande symbolen. In veel gevallen is de kans van voorkomen echter wel afhankelijk van voorgaande gebeurtenissen.

Shannon geeft in [Sha48] een zeer duidelijk illustratie van deze coherentie. Hij doet dit door een benadering van de Engelse taal te genereren, uitgaande van eigenschappen van bestudeerde Engelse teksten. Onderstaande zin is gegenereerd door volledig willekeurige letters uit het alfabet[4] achter elkaar te plakken.

> XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZLHJQD

Rekening houdend met de frequentie van voorkomen van iedere letter in de bestudeerde teksten werd volgende zin verkregen:

> OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL

Hoewel dit nog niet veel weg heeft van een Engelse zin, is er al een veel realistischere opsplitsing in woorden, en klopt de verhouding klinkers ten opzichte van medeklinkers. Er werd echter nog steeds geen rekening gehouden met voorgaande letters. Door de frequenties van paren van letters te beschouwen kan de kans op voorkomen afhankelijk van de voorgaande letter berekend worden. Dit resultueert in:

> ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

Deze zin vertoont al typische Engelse lettergrepen en patronen. Deze tendens wordt nog veel duidelijker zichtbaar als er rekening gehouden wordt met de twee voorgaande letters:

> IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

De volgende stap vereist de berekening van $27^4 = 373248$ kansen en wordt computationeel erg zwaar. Het is echter mogelijk een idee te krijgen van wat er verwacht kan worden indien deze methode verdergezet zou worden door met volledige woorden te werken in plaats van letters. Enkel uitgaande van de frequenties in de teksten levert dit niet veel meer op dan een willekeurig samenraapsel van woorden:

> REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TOOF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

[4] Het hier gebruikte alfabet bestaat uit de 26 verschillende letters, aangevuld met de spatie.

Maar als de frequenties berekend worden in functie van het vorige woord, heeft de resulterende zin iets weg van wat een overenthousiaste spreker zou kunnen uitbrengen:

> THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Beschouw nu een willekeurige Engelse tekst. Door de structuur van de taal is de data sterk gecorreleerd. Dit houdt in dat een groot deel van de data gebruikt wordt om de structuur te coderen. Als die structuur van de informatie gescheiden kan worden, is het mogelijk enkel de informatie te coderen.

Het voorgaande voorbeeld toont aan dat een groot deel van die structuur inderdaad uit de data gefilterd kan worden. Door willekeurig letters volgens een gegeven kansverdelingsfunctie te genereren ontstonden er duidelijk herkenbare Engelse patronen. Deze patronen moeten dus voortgevloeid zijn uit die kansverdelingsfuntie. Deze kansverdelingsfunctie beschrijft hoe zinnen gevormd kunnen worden volgens de structuur van de taal.

Welke zin er juist gevormd werd, werd bepaald door de willekeurige component. Dit is dus de informatie die gecodeerd moet worden. In dit geval hebben de zinnen echter geen betekenis. De informatie bestaat enkel uit ruis, gestructureerd zodat het lijkt op Engels. Wanneer diezelfde kansverdelingsfunctie echter gebruikt wordt om gelijkaardige informatie uit Engelstalige zinnen te extraheren is het resultaat wel betekenisvol.

Ook bij afbeeldingen, en dus ook bij videobeelden, is de data gestructureerd. Een eenvoudige benadering van de structuur kan als volgt omschreven worden: naburige pixels hebben de neiging op elkaar te lijken. Iedere pixel kan dus gecodeerd worden als het verschil met de naburige pixels. Figuur 3.5 illustreert deze methode. De rechtse foto lijkt opgebouwd te zijn uit de randen van de linkse foto. Dit zijn de plaatsen waar de foto het hardst afwijkt van de beschreven structuur, en waar dus de meeste data nodig is om de informatie te coderen.

Ondanks het feit dat de overbodige structuur uit de foto verwijderd is, is het aantal bits niet verminderd. De foto neemt zelfs meer bits in beslag dan voordien. De mogelijk waarden situren zich nu immers tussen -255 en 255 in plaats van tussen 0 en 255. Er is dus een extra bit per waarde nodig. De entropie is echter van 7.7832 naar 5.0465 gedaald. Door middel van entropie codering kan de overtollige informatie dus geëlimineerd worden. Als de voorspelling in de meeste gevallen ongeveer klopt, ontstaat er een grote hoeveelheid heel kleine waarden. Deze zullen dus de kleinste bitcodes krijgen.

In het eenvoudigste geval gebeurt de voorspelling van de waarde van een pixel aan de hand van één naburige pixel. Habibi [Hab70] toonde aan dat het gebruik van twee referentiepixels in plaats van één, en ook drie referentiepixels in plaats van twee gemiddeld een correctere benadering opleveren. Meer dan drie pixels leveren geen duidelijke verbeteringen meer op.

De meest gebruikte schema's voor voorspellende codering zijn $\hat{X} = \frac{1}{2}A + \frac{1}{2}C$, $\hat{X} = A - B + C$ en $\hat{X} = \frac{3}{4}A - \frac{1}{2}B + \frac{3}{4}C$, waarbij $\hat{X}$ de voorspelde waarde van

die nodig is om de mengkleur te verkrijgen, wordt uitgedrukt als een getal tussen 0 en 255.

Kleuren in een digitale afbeelding worden meestal voorgesteld door middel van het RGB kleursysteem, aangezien computerschermen dit model gebruiken. Het RGB model is gebaseerd op de fysieke manier waarop het menselijk oog licht waarneemt.

Het YUV model gebruikt een andere basis om deze kleurruimte voor te stellen. Kleurcomponenten ten opzichte van de RGB basis kunnen gemakkelijk getransformeerd worden naar de YUV basis. Hiervoor worden volgende vergelijkingen gebruikt:

$$\begin{aligned} Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\ U &= -0.147 \times R - 0.289 \times G + 0.436 \times B \\ &= 0.492 \times (B - Y) \\ V &= 0.615 \times R - 0.515 \times G - 0.100 \times B \\ &= 0.877 \times (R - Y) \end{aligned}$$

Het Y-kanaal vertegenwoordigt de helderheid, of luminantie; het U- en V- kanaal bepalen de chrominantie. Doordat het menselijke oog een vrij lage kleurenresolutie heeft, kunnen het U- en V-kanaal met mindere kwaliteit gecodeerd worden, zonder de kwaliteit van het gereconstrueerde beeld sterk te beïnvloeden. In de praktijk gebeurt dit meestal door de resolutie van het U- en V-kanaal te halveren volgens de horizontale as, en eventueel ook volgens de vertikale as.

3.5.2 Fourier transformatie

Periodische, continue functies kunnen uitgedrukt worden in termen van sinussen en cosinussen van verschillende frequenties. Deze ontbinding is de *Fourier analyse* van de functie, genoemd naar de Franse wiskundige Jean Baptiste Joseph Fourier die deze techniek in 1768 ontwikkelde. De verschillende sinussen en cosinussen waarin de functie wordt opgesplitst worden elk gekarakteriseerd door een amplitude en een fase. De variërende frequenties worden gegeven door een basisfrequentie en natuurlijke veelvouden van deze frequentie. De basisfrequentie is gelijk aan de frequentie waarmee de periodes van de functie voorkomen.

Een periodische, continue functie kan dus beschreven worden door een reeks amplitudes en fases ten opzichte van een basis van sinussen en cosinussen met gekende frequenties, de zogenaamde *harmonische functies*. De resulterende reeks coëfficiënten, telkens een amplitude en een fase, wordt een *Fourierreeks* genoemd. Door de verschillende sinussen en cosinussen op te tellen, bekomt men weer de originele functie. Het opbouwen van een functie door de geschikte sinussen en cosinussen te combineren wordt de *Fourier synthese* genoemd. Figuur 3.5.2 illustreert de Fourier synthese van een aantal functies.

Bij niet-periodische functies is het spectrum niet discreet en wordt de Fourierreeks veralgemeend tot een integraal. Deze transformatie wordt de *Fourier*

gebruik te maken van de formule van Euler[5]:

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{3.5}$$

Door (3.5) in (3.3) te substitueren, bekomt men:

$$f(x) = \int_{-\infty}^{\infty} F(u)\ [\cos(2\pi ux) + i\sin(2\pi ux)]\ \mathrm{d}u \tag{3.6}$$

Deze representatie illustreert duidelijk dat, door berekening van de Fourier transformatie, de functie $f(x)$ uitgedrukt kan worden als de integraal van gewogen sinus- en cosinus functies. Net zoals de Fourier analyse van een periodische functie resulteert in de som van sinussen en cosinussen, kan de Fourier transformatie van een niet-periodische functie gezien worden als een oneindige continue 'sommatie' van sinussen en cosinussen. De functie $f(x)$ wordt in dit geval dus opgesplitst in een continu spectrum in plaats van een discreet spectrum.

Ook (3.4) kan in een goniometrische vorm gelijkaardig aan (3.6) herschreven worden:

$$\begin{aligned} F(u) &= \int_{-\infty}^{\infty} f(x)\ [\cos(-2\pi ux) + i\sin(-2\pi ux)]\ \mathrm{d}x \\ &= \int_{-\infty}^{\infty} f(x)\ [\cos(2\pi ux) - i\sin(2\pi ux)]\ \mathrm{d}x \end{aligned} \tag{3.7}$$

Door het voorkomen van $i^2 = -1$ in (3.4) resulteert de representatie van de functie $f(x)$ in het frequentie domein in een reeks complexe getallen. $F(u)$ kan dus beschreven worden als een reëel en een imaginair deel:

$$F(u) = R(F(u)) + i\,I(F(u)) = R_u + iI_u$$

Anderzijds kan ook de representatie in polaire coördinaten gebruikt worden. $F(u)$ wordt dan opgesplitst in een amplitude spectrum λ_u en en fase spectrum ϕ_u:

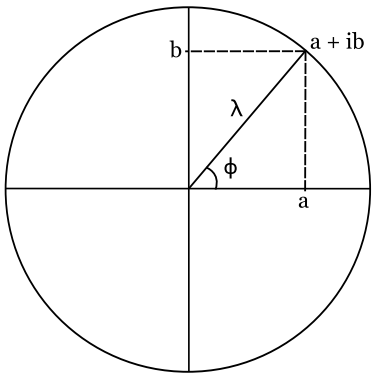
$$\begin{aligned} \lambda_u &= |F(u)| = \sqrt{(I_u)^2 + (R_u)^2} \\ \phi_u &= \tan^{-1}\left(\frac{I_u}{R_u}\right) \end{aligned}$$

Deze representatie met polaire coördinaten wordt geïllustreerd in figuur 3.8.

Fourier analyse voor beeldcompressie

Een monochrome afbeelding kan beschouwd worden als een discrete functie die iedere pixelcoördinaat associeert met een intensiteit. Bij afbeeldingen in kleur geldt hetzelfde voor iedere component van de kleurenruimte.

[5]De correctheid van gelijkheid (3.5) kan gemakkelijk aangetoond worden door de Taylor ontwikkeling van zowel $e^{i\theta}$ als van $\cos\theta$ en $\sin\theta$ te berekenen.



Figuur 3.8: *Representatie van een complex getal $a + ib$ in polaire coördinaten (λ, ϕ). Merk op dat $a = \lambda \cos\phi$ en $b = \lambda \sin\phi$. Anderzijds geldt dat $\lambda = \sqrt{a^2 + b^2}$ en $\phi = \tan^{-1}(\frac{a}{b})$.*

Het zou handig zijn deze functie te kunnen transformeren naar het frequentie domein. De Fourier transformatie van dit soort functies heeft namelijk de neiging een groot deel van de informatie die nodig is om de oorpronkelijke functie te reconstrueren te concentreren in de coëfficiënten van de laagste frequenties. De lage frequenties bepalen namelijk de ruwe vorm van de functie, terwijl de informatie in de hoge frequenties het 'fijne detail' bevat. Door de coëfficiënten van de hoge frequenties weg te laten[6], kan er dus een goede benadering van de functie gegeven worden met minder coëfficiënten.

Een afbeelding is echter een bemonsterd signaal, waardoor de overeenkomstige functie *discreet* is, en dus niet continu. Voor beelverwerking wordt er dus gewerkt met de discrete vorm van de Fourier transformatie. Aangezien afbeeldingen meestal voorgesteld worden als een twee-dimensionale rij van intensiteiten, wordt de Fourier transformatie eerst uitgebreid naar twee dimensies:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\ [\cos(2\pi(ux+vy)) - i\sin(2\pi(ux+vy))]\ \mathrm{d}x\,\mathrm{d}y \tag{3.8}$$

Voor de inverse transformatie geeft dit:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\ [\cos(2\pi(ux+vy)) + i\sin(2\pi(ux+vy))]\ \mathrm{d}u\,\mathrm{d}v \tag{3.9}$$

Discrete Fourier transformatie

Door (3.8) te discretiseren bekomt men de twee-dimensionale discrete Fourier transformatie (DFT):

[6] In de praktijk worden de coëfficiënten van de hoge frequenties meestal niet zomaar weggelaten, maar wel harder gekwantiseerd.

Definitie 3.3 *De discrete Fourier tranformatie van een functie $f(x, y)$ met $x = 0, 1, 2, \ldots, M-1$ en $y = 0, 1, 2, \ldots, N-1$ is gelijk aan:*

$$F(u,v) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \left[\cos\left(2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right) - i\sin\left(2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)\right] \tag{3.10}$$

voor $u = 0, 1, 2, \ldots, M-1$ en $v = 0, 1, 2, \ldots, N-1$.

De waarde van $F(u, v)$ voor ieder paar (u, v) is dus samengesteld uit de som van alle waarden van de functie $f(x, y)$. De $M \times N$ termen van $F(u, v)$ worden de *frequentie componenten* van de transformatie genoemd. Berekening van de DFT van een afbeelding van M op N pixels vraagt dus berekening van $(M \times N)^2$ frequentie componenten.

Op analoge wijze aan definitie 3.3 wordt ook, uitgaande van (3.9), de inverse van de twee-dimensionale discrete Fourier transformatie gedefiniëerd:

Definitie 3.4 *De inverse discrete Fourier tranformatie van $F(u, v)$ met $u = 0, 1, 2, \ldots, M-1$ en $v = 0, 1, 2, \ldots, N-1$ is gelijk aan:*

$$f(x,y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \left[\cos\left(2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right) + i\sin\left(2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)\right] \tag{3.11}$$

voor $x = 0, 1, 2, \ldots, M-1$ en $y = 0, 1, 2, \ldots, N-1$.

3.5.3 Discrete cosinus transformatie

De discrete cosinus transformatie, kortweg DCT, is gebaseerd op de discrete Fourier transformatie, maar gebruikt enkel reële getallen. Hierdoor is de DCT zeer aanlokkelijk voor de compressie van afbeeldingen. De DCT concentreert de energie ook beter in de lage coëfficiënten dan de DFT.

De afbeelding wordt meestal in blokken van 8×8 pixels opgesplitst vooraleer de DCT wordt toegepast. Hier volgt de definitie van de DCT voor het iets algemenere geval van $N \times N$ pixels:

Definitie 3.5 *De discrete cosinus tranformatie van een functie $f(x, y)$ met $x, y = 0, 1, 2, \ldots, N-1$ is gelijk aan:*

$$F(u,v) = \frac{2}{N} C_u C_v \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)\,u\pi}{2N}\right) \cos\left(\frac{(2y+1)\,v\pi}{2N}\right) \tag{3.12}$$

voor $u = 0, 1, 2, \ldots, N-1$ en $v = 0, 1, 2, \ldots, N-1$, en met C_w als volgt gedefiniëerd:

$$C_w = \begin{cases} \frac{1}{\sqrt{2}} & \text{als } w = 0 \\ 1 & \text{als } w \neq 0 \end{cases}$$

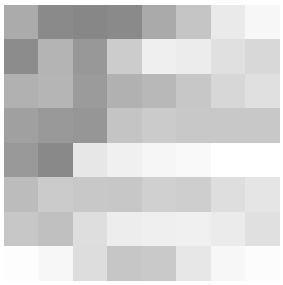
YUV-kleurenruimte transformatie

Aangezien het menselijk oog gevoeliger is voor helderheid dan voor kleur, wordt de afbeelding eerst getransformeerd van de gebruikelijke RGB-kleurenruimte naar de YUV-kleurenruimte. Hierdoor kan de helderheid onafhankelijk van de kleureninformatie gecodeerd worden. Het Y-kanaal bevat immers de intensiteit, terwijl de U- en V-kanalen de kleurinformatie bevatten.

JPEG biedt de mogelijkheid om de U- en V-kanalen in een lagere resolutie te coderen, om zo het aantal bits nodig om de foto te coderen meteen te verminderen. De standaard geeft de mogelijkheid de originele resolutie te behouden, de resolutie in horizontale richting te halveren, of de resoluite in zowel horizontale als verticale richting te halveren. Hierna doorlopen de drie kanalen afzonderlijk het coderingsproces, net zoals drie monochrome afbeeldingen apart gecomprimeerd zouden worden.

Discrete Cosinus Transformatie

De afbeelding wordt dan opgedeeld in blokken van 8×8 pixels, die elk getranformeerd worden naar het frequentie domein, gebruik makende van de Discrete Cosinus Transformatie. Dit resulteert in blokken, opnieuw van 8×8 pixels, waarvan het grootste deel van de informatie in de linkerbovenhoek geconcentreerd is.



Figuur 3.11: *Blok van* 8×8 *pixels afkomstig uit de foto in figuur 3.5.*

In figuur 3.11 is een blok van 8×8 pixels te zien dat gebruikt zal worden om het JPEG coderingsproces te illustreren. De waarden van iedere pixel van dit blok zijn hieronder weergegeven.

171	138	135	138	170	197	234	247
140	180	152	204	239	236	224	216
177	181	155	177	184	199	215	224
160	153	150	196	203	200	200	200
153	137	231	240	246	249	255	255
188	203	200	199	208	206	222	229
199	192	222	237	239	240	235	224
253	247	221	198	201	231	247	252

Wanneer de pixelwaarden in het interval 0–255 liggen, zoals gebruikelijk, valt de eerste coëfficiënt van de DCT in het interval 0–2040. Deze coëfficiënt, die zich dus helemaal in de linkerbovenhoek bevindt, is gelijk aan de gemiddelde waarde van de pixels in het originele blok, en wordt de DC[7] coëfficiënt genoemd. De overige coëfficiënten, de AC[8] coëfficiënten, variëren van -1023 tot $+1023$. Daarom wordt iedere pixelwaarde eerst met 128 verminderd voordat de DCT toegepast wordt, waardoor alle coëfficiënten tussen -1023 en 1023 komen te liggen. Het resultaat van deze substractie, gevolgd door toepassing van de DCT op het voorbeeldblok staat hieronder.

608	-161	4	15	10	5	-6	0
-126	-67	7	22	14	-11	-16	-7
23	20	66	19	-15	-12	-6	-2
0	-39	-18	-15	3	11	16	20
-5	-21	28	-3	4	19	20	12
-57	16	20	9	12	10	0	-1
-31	28	83	-8	-9	-4	-3	1
37	-29	-8	-15	-1	22	23	13

Kwantisatie

De DCT coëfficiënten zijn reeds gekwantiseerd doordat de reële getallen afgerond werden naar gehele getallen. Om goede compressie te bekomen zal er echter nog agressiever gekwantiseerd moeten worden. Om het verlies in kwaliteit te beperken wordt rekening gehouden met de gevoeligheid van het oog voor de verschillende coëfficiënten.

Zoals reeds verteld werd, is het menselijk oog is gevoelig voor kleine variaties in helderheid over een relatief grote oppervlake, maar kan moeilijk de exacte variatie in helderheid van een signaal met hoge frequentie bepalen. Een kleine afwijking in de coëfficiënten van de hoge frequenties zal dus minder snel opgemerkt worden dan een verschil in de coëfficiënten van de lage frequenties.

Kwantisatie gebeurt aan de hand van kwantisatie matrices. Een kwantisatie matrix bevat voor iedere positie in het blok een waarde die de agressiviteit van de kwantisatie, en dus het aantal mogelijke waarden, bepaalt. Dit gebeurt door simpelweg iedere coëfficiënt in het blok te delen door de overeenkomstige waarde in de kwantisatie matrix, en het resulterende getal af te ronden naar het dichtsbijzijnde gehele getal.

Door, in de kwantisatie matrix, rechtsonder grotere waarden te zetten dan linksboven, worden de coëfficiënten van de hoge frequenties harder gekwantiseerd. Dit leidt tot een groot aantal nullen, en voor de rest heel kleine waarden bij deze hoge frequenties. Een voorbeeld van een veel gebruikte kwantisatie matrix is weergegeven in figuur 3.12. Deze matrix is opgesteld door de gevoeligheid van het menslijke oog ten opzichte van verschillen in helderheid te observeren.

Het resultaat van de kwantisatie van het voorbeeld, gebruikt makende van de matrix in figuur 3.12, is hieronder weergegeven.

[7] DC is de afkorting voor 'Direct Current', wat 'gelijkstroom' betekent.
[8] AC staat voor 'Alternating Current', wat 'wisselstroom' betekent.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Figuur 3.12: *Veelgebruikte kwantiatie matrix die voorgesteld wordt in de JPEG standaard. De matrix is geen deel van de standaard, het is enkel een voorbeeldmatrix die door het comité voorgesteld wordt.*

38	-15	0	1	0	0	0	0
-10	-6	1	1	1	0	0	0
2	2	4	1	0	0	0	0
0	-2	-1	-1	0	0	0	0
0	-1	1	0	0	0	0	0
-2	0	0	0	0	0	0	0
-1	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0

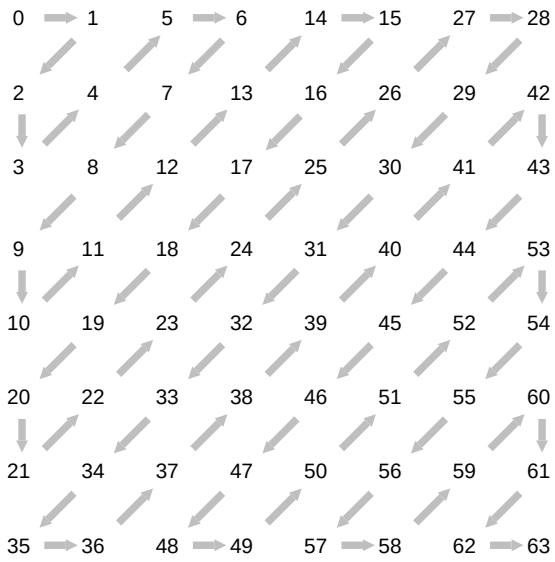
De afronding zorgt ervoor dat de compressie verlieslatend is. De rest van het proces is namelijk perfect omkeerbaar; door de bekomen reële waarden, nog voor de afronding plaats gevonden heeft, opnieuw te vermenigvuldigen met de kwantisatie matrix, de omgekeerde DCT toe te passen, en terug naar de RGB-kleurenruimte te transformeren, bekomt men opnieuw de originele afbeelding. Hoe hoger de kwantisatie-waarde, hoe meer verschillende coëfficiënten na deling en afronding op hetzelfde getal afgebeeld worden. Dit leidt enerzijds tot meer kwalitietsvelies, maar anderzijds tot data die beter geschikt voor entropie codering.

Entropie codering

Vervolgens worden de 8×8 waarden achter elkaar gezet om zo een sequentie van 64 waarden te bekomen. De volgorde van de sequentie wordt bepaald door het zig-zag patroon in figuur 3.13. Dit patroon is onststaan uit de observatie dat de DCT de hoge waarden in de linkerbovenhoek concentreert, een effect dat nog versterkt wordt door de kwantisatie. Door het blok op deze manier te doorlopen, zullen de grote getallen zich dus vooraan in de sequentie bevinden. Toegepast op het voorbeeld, geeft dit aftastingspatroon volgende sequentie:

38 -15 -10 2 -6 0 1 1 2 0 0 -2 4 1 0 0 1 1 -1 -1 -2 -1 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Het valt meteen op dat de sequentie eindigt met een lange rij nullen. Dit kan uitgebuit worden door een speciaal teken in te voeren dat het einde van het niet-nul-deel van de sequentie aanduidt. De hele rij nullen kan dan vervangen worden door dit ene teken, gewoonlijk EOB ('End Of Block') genoemd.



Figuur 3.13: *Zig-zag aftastingspatroon dat de volgorde bepaalt waarin het* 8×8 *blok doorlopen moeten worden om een sequentie van 64 waarden te krijgen.*

In de overgebleven sequentie komen nog een aantal aaneenschakelingen van meerdere opeenvolgende nullen voor. Ook hiervan kan gebruik gemaakt worden bij het coderen van de data. De sequentie wordt nu herschreven door alle nullen weg te laten, en vóór ieder waarde in de overgebleven sequentie het aantal weggelaten nullen te zetten. "0 0 0 -1" wordt dus "3,-1", aangezien er drie nullen voor het getal -1 stonden. De DC coëfficiënt, de eerste waarde in de sequentie, wordt voorlopig ook even weggelaten. Dit wordt later nog toegelicht. Voor het voorbeeld resulteert dit in onderstaande sequentie:

0,-15 0,-10 0,2 0,-6 1,1 0,1 0,2 2,-2 0,4 0,1 2,1 0,1 0,-1 0,-1 0,-2 0,-1 1,1 0,-1 10,1 1,1 EOB

Deze sequentie kan met behulp van entropie codering gecodeerd worden, door aan ieder koppel een uniek codewoord toe te kennen. Doordat het aantal mogelijk codewoorden bij deze methode zo groot is, werkt JPEG met een lichtjes aangepaste coderingsmethode. De gekwantiseerde DCT coëfficienten worden in tien verschillende categorieën onderverdeeld, genummerd van 1 tot 10. Deze cateogorieën zijn weergegeven in tabel 3.5. Aan ieder koppel, bestaande uit het categorienummer en een getal dat het aantal nullen aangeeft, wordt dan een codewoord toegekend.

Hierdoor zijn er minder verschillende codewoorden nodig, maar gaat er ook informatie verloren. Het is immers onmogelijk de juiste coëfficiënt terug te bekomen als enkel de categorie bekend is. Daarom wordt achter ieder codewoord een aantal bits geplakt, dat aangeeft welke coëfficiënt binnen de categorie bedoeld wordt. Aangezien het aantal coëfficiënten binnen iedere categorie gelijk is aan 2^n met n het categorienummer, is het aantal toegevoegde bits gelijk aan het categorienummer. De coëfficiënten -3, -2, 2 en 3 krijgen dus respectievelijk

Categorie	AC coëfficiënten
1	-1,1
2	-3,-2,2,3
3	-7,. . . , -4, 4,. . . , 7
4	-15,. . . , -8, 8,. . . , 15
5	-31,. . . , -16, 16,. . . , 31
6	-63,. . . , -32, 32,. . . , 63
7	-127,. . . , -64, 64,. . . , 127
8	-255,. . . , -128, 128,. . . , 255
9	-511,. . . , -256, 256,. . . , 511
10	-1023,. . . , -512, 512,. . . , 1023

Tabel 3.5: *Categorieën die toegekend worden aan de AC coëfficiënten.*

de bits 00, 01, 10 en 11 achter hun codewoord geplakt.

Als in de voorbeeldsequentie de coëfficiënten door de gepaste categorieën vervangen worden, resulteert dit in:

0,4 0,4 0,2 0,3 1,1 0,1 0,2 2,2 0,3 0,1 2,1 0,1 0,1 0,1 0,2 0,1 1,1 0,1 10,1 1,1 EOB

De volledige tabel met codewoorden voor deze koppels is weergegeven in [ITUb], zoals ook het algoritme om deze tabel te genereren. Deze tabel is opgesteld aan de hand van de statistieken van een groot aantal afbeeldingen. Gebruik makende van deze codewoorden, resulteert de codering van het voorbeeld in de volgende 106 bits:

10110000 10110101 0110 100001 11001 001 0110 1111100101 100100 001 111001 001 000 000 0101 000 11001 000 1111110101 11001 1010

Hiermee is het coderingsproces voor de AC coëfficiënten van het blok beëindigd. Nu moet de DC coëfficiënt nog gecodeerd worden. Aangezien de DC coëfficiënt gelijk is aan de gemiddelde waarde van het blok, zijn de DC coëfficiënten van naburige blokken gecorreleerd. Daarom worden ze voorspellend gecodeerd, telkens gebruik makend van de DC coëfficiënt van het vorige blok. De resulterende waarden worden dan analoog aan de AC coëfficiënten gecodeerd, maar met andere tabellen [ITUb].

Codering van de DC coëfficiënt van het voolbeeldblok, zonder rekening te houden met andere blokken, resulteerde in 10 extra bits. Het totale blok van 64 waarden werd dus gecodeerd in 116 bits. Zonder compressie zou de codering $64 \times 8 = 512$ bits ingenomen hebben. Bij de compressie van een volledige afbeelding wordt het aantal bits echter nog harder gereduceerd door de voorspellende codering van de DC coëfficiënten, door lagere resoluties van de U- en V-kanalen, en eventueel door het gebruik van agressievere kwantisatie matrices.

Reconstructie

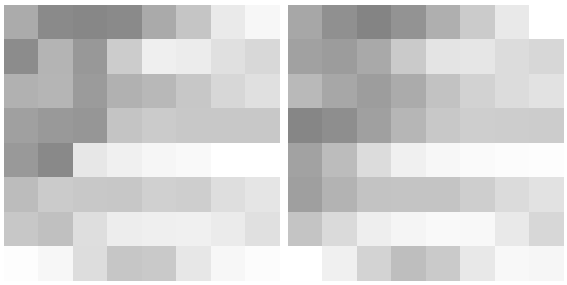
De reconstructie van de afbeelding gebeurt door het coderingsproces in de omgekeerde volgorde te doorlopen, en telkens de inverse bewerkingen uit te voeren. De gecodeerde AC coëfficiënten van een blok worden dus als volgt herseld. Er

wordt telkens gezocht naar het eerst voorkomende gekende codewoord in de data. Dit codewoord komt overeen met een categorie en een aantal nullen. Het categorienummer bepaalt het aantal extra bits dat ingelezen moet worden om de juiste coëfficiënt binnen deze categorie te identificiëren. De categorie en deze extra bits bepalen dus samen één coëfficiënt. Het tweede getal dat door het codewoord gecodeerd werd, bepaalt nog een aantal coëfficiënten, die allemaal gelijk zijn aan nul.

Door het zig-zag patroon omgekeerd te gebruiken, wordt uit deze sequentie opnieuw een blok bekomen. Dit blok wordt vermenigvuldigd met de kwantisatie matrix, gevolgd door de toepassing van de inverse DCT. De blokken worden dan samengevoegd en terug omgezet naar de RGB-kleurenruimte, waardoor de originele afbeelding, eventueel aangepast als gevolg van de verlieslatende compressie, opnieuw geconstrueerd wordt. Het resultaat van de reconstructie van het voorbeeldblok is hieronder weergegeven. Het eerste blok is het originele blok, het tweede blok is de reconstructie. Figuur 3.14 toont beide afbeeldingen.

171	138	135	138	170	197	234	247
140	180	152	204	239	236	224	216
177	181	155	177	184	199	215	224
160	153	150	196	203	200	200	200
153	137	231	240	246	249	255	255
188	203	200	199	208	206	222	229
199	192	222	237	239	240	235	224
253	247	221	198	201	231	247	252

167	144	132	147	175	203	233	255
161	156	169	202	228	230	220	215
185	167	157	171	194	210	220	226
134	142	160	182	200	206	205	204
162	188	220	240	247	250	252	253
159	179	195	196	196	206	219	226
196	218	238	245	249	248	233	215
255	240	211	190	202	232	248	246



Figuur 3.14: *Het linker blok is opnieuw het blok uit figuur 3.11, hier opnieuw afgebeeld als vergelijking. Het rechter blok is het gereconstrueerde blok na JPEG codering.*

3.8 Bewegingscompensatie

In de voorgaande secties werd een hele reeks compressietechnieken besproken die gecombineerd kunnen worden om afzonderlijke afbeeldingen te comprimeren: door toepassing van de discrete cosinus transformatie wordt de energie samengeperst in de linkerbovenhoek. Na kwantisatie is de overgebleven data uitermate geschikt voor entropie codering.

Digitale video is niet meer dan een serie afbeeldingen die op opeenvolgende tijdstippen getoond worden. Het is dus mogelijk deze afbeeldingen één voor één te comprimeren zoals hierboven besproken werd. De extra tijdsdimensie is echter een grote bron van redundantie: twee opeenvolgende video frames bevatten vaak voor een groot deel dezelfde informatie.

Een mogelijke compressietechniek die gebruik maakt van deze bron van redundantie zou dus simpelweg het pixelgewijze verschil tussen twee opeenvolgende beelden kunnen opslaan. Het resultaat, residu genaamd, bevat dan minder informatie en kan dus efficiënter gecodeerd worden. Deze techniek is in essentie hetzelfde als de voorspellende codering die voorheen besproken werd. Om goede resultaten op te leveren, mag er bij deze nogal naieve methode echter niet te veel beweging plaatsvinden.

Stel bijvoorbeeld dat een bewegend object voor een statische achtergrond gefilmd wordt met een stilstaande camera. Op de posities in het beeld waar de achtergrond te zien is, verandert er dus niets ten opzichte van de vorige frame. Op plaatsen waar het object in de voorgrond te zien is, is informatie te zien die waarschijnlijk voor een groot deel ook in het vorige beeld te zien was, maar dan van positie veranderd door de beweging. Het zou dus interessant zijn de beweging van de objecten in de scène te detecteren, zodat een beeld veel nauwkeuriger voorspeld kan worden uit het voorgaande beeld.

Deze techniek wordt *bewegingscompensatie* genoemd, of '*motion compensation*' in het Engels. Het is dus een manier om het verschil tussen twee opeenvolgende beelden te beschrijven aan de hand van de beweging die de verschillende objecten in de scène gemaakt hebben, om zo de overbodige gegevens in het tijdsdomein te elimineren. Het proces waarbij deze beweging wordt vastgesteld wordt *bewegingsschatting* of '*motion estimation*' genoemd.

3.8.1 Blokgebaseerde bewegingscompensatie

Bij *blokgebaseerde bewegingscompensatie* (BBMC)[9] wordt het te comprimeren beeld opgedeeld in blokken van bijvoorbeeld 16×16 pixels. Voor ieder blok wordt dan gezocht naar de positie waar dit deel van het beeld zich bevond in het voorgaande beeld. Het verschil tussen de bekomen positie en de positie van het blok in het huidige beeld resulteert in een 2D-vector die de *bewegingsvector* genoemd wordt. Deze bewegingsvector is dus de informatie die gecodeerd moet worden, eventueel aangevuld met een residu[10] indien het verschil tussen de twee

[9] afkomstig van het Engelse '*Block Based Motion Compensation*'

[10] Het residu is in dit geval dus het pixelgewijze verschil tussen het originele blok en het gevonden blok. Doordat beide blokken normaal gezien sterk op elkaar lijken bevat het residu weinig informatie, en kan het bijgevolg zeer efficiënt gecodeerd worden.

blokken te groot is.

In principe hoeft hiervoor niet noodzakelijk het voorgaande beeld gebruikt te worden; er kan evengoed een beeld van een nog vroeger tijdstip gebruikt worden, of zelfs een beeld dat na het huidige beeld komt. Het gebruikte beeld wordt het *referentie beeld* genoemd.

BMMC stelt het reconstructieproces dus in staat het huidige beeld te voorspellen uitgaande van een referentie beeld. Het is bijgevolg een vorm van voorspellende codering. De bewegingsvector geeft het reconstructrieproces de nodige informatie om het betreffende blok in het referentie beeld terug te vinden. Het residu bevat de informatie die nodig is om uit het gevonden blok het originele blok weer te bekomen.

Merk op dat de bewegingsvectoren niet noodzakelijk overeen moeten komen met de werkelijke beweging van het blok; iedere vector die het blok afbeeldt op een gelijkaardig blok in het referentie beeld zorgt voor een goed resultaat. Wanneer de bewegingvectoren echter een goede schatting zijn van de werkelijke beweging in de scène, is er veel meer coherentie tussen de verschillende bewegingsvectoren van naburige—zowel spatiaal als in de tijd—blokken. Hierdoor kunnen de bewegingsvectoren op hun beurt weer beter gecomprimeerd worden.

In figuur 3.18 is het resultaat te zien van een blokgebaseerde voorspelling van de beweging tussen twee beelden. Het gebruikte algoritme, EPZS, komt later nog aan bod.

Bij BBMC is de bewegingsschatting dus het proces waarbij de bewegingsvectoren bekomen worden. Hierbij moet gezocht worden naar een blok in het referentie beeld dat een grote gelijkenis vertoont met het huidige blok. Om deze gelijkenis te meten wordt de *voorspellingsfout* berekend, meestal door gebruik te maken van MAD[11] of SAD, afkomstig van het Engelse '*mean absolute difference*' en '*sum of absolute difference*' respectievelijk. MAD en SAD worden als volgt gedefinieerd:

Definitie 3.7 *De MAD en SAD voor een blok met grootte $N \times N$ dat zich op positie (x, y) in het huidige beeld bevindt, ten opzichte van een blok met grootte $N \times N$ dat zich op positie $(x + v_x, y + v_y)$ in het referentie beeld bevindt, worden gegeven door:*

$$\begin{aligned} MAD(v_x, v_y) &= \frac{1}{N^2} \cdot SAD(v_x, v_y) \\ SAD(v_x, v_y) &= \sum_{m,n=0}^{N-1} \left| I_t(x+m, y+n) - I_r(x+v_x+m, y+v_y+n) \right| \end{aligned}$$

waarbij I_t het huidige beeld, en I_r het referentie beeld is.

De voorspellingsfout wordt enkel voor het Y-kanaal, de helderheidscomponent dus, van de afbeelding berekend. In dit kanaal is de beweging het duidelijkst aanwezig. De bekomen bewegingsvector wordt dan gebruikt om alledrie de kanalen te coderen.

[11] Een andere vaak voorkomende benaming voor MAD is '*mean absolute error*' (MAE).

Full Search

De eenvoudigste manier om bewegingsschatting te implementeren, is door simpelweg de voorspellingsfout voor ieder blok in het referentie beeld ten opzichte van het huidige blok te berekenen. Het blok met de kleinste voorspellingsfout wordt geselecteerd als referentieblok en bepaalt dus de bewegingsvector voor het huidige te comprimeren blok. Deze methode wordt '*full search*' genoemd doordat alle mogelijke bewegingsvectoren beschouwd worden.

Betere resultaten worden bekomen door niet enkel rekening te houden met de voorspellingsfout, maar ook de hoeveelheid bits die nodig zijn om de bijhorende bewegingsvector te coderen, mee te laten tellen.

Het probleem met full search is dat het computationeel te zwaar is om praktisch buikbaar te zijn. Daarom zijn er een hoop alternatieve algoritmen voor bewegingsschatting ontwikkeld die in veel minder tijd een bewegingsvector selecteren, maar niet optimaal zijn wat betreft compressie.

Een eerste logische aanpassing om de berekentijd te verminderen is het gebruik van een zoekvenster dat de lengte van de bewegingvector beperkt. Door bijvoorbeeld enkel verplaatsingen van maximaal $2\times$ de beeldbreedte per seconde te beschouwen, wordt het zoekvenster in horizonatle richting beperkt tot $beeldbreedte/framerate$ pixels. Een beweging die de volledige beeldbreedte in minder dan een halve seconde aflegt zal dus nooit correct gedetecteerd worden.

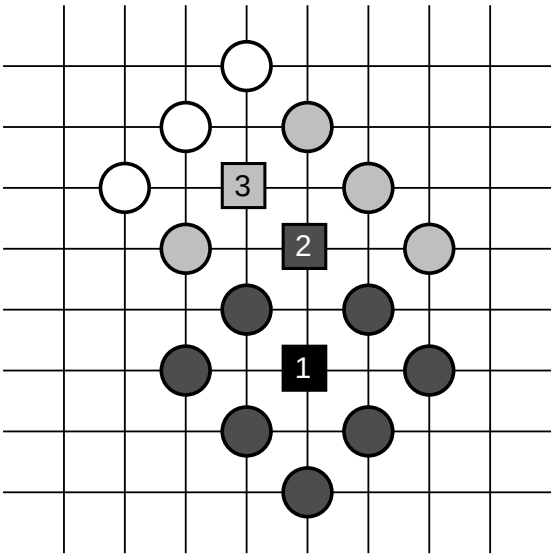
Aangezien een typische scène meer en snellere horizontale beweging bevat dan verticale en aangezien het menselijke oog ook minder gevoelig is voor verticale beweging, is het redelijk het zoekvenster in de verticale richting nog harder te beperken dan in de horizontale richting.

Alhoewel het kleinere zoekvenster het aantal berekingen aanzienlijk beperkt, moeten er per blok gewoonlijk toch nog enkele tienduizenden bewegingsvectoren met elkaar vergeleken worden om de beste kandidaat hieruit te selecteren. Deze vergelijkingen gebeuren aan de hand van een criterium dat de voorspellingsfout uitdrukt, zoals bijvoorbeeld SAD, wat voor een blokgrootte van 16×16 pixels resulteert in 256 vermenigvuldigingen per kandidaat bewegingsvector.

Andere technieken voor bewegingsschatting

Om de complexiteit van het zoekproces nog verder te beperken zijn een hele reeks algoritmen ontwikkeld. '*Three-Step Search*' (TSS) [KIHI81], '*2-D log Search*' (TDL) [JJ81] en '*Diamond Search*' (DS) [ZM00] zijn bekende voorbeelden van algoritmen die de complexiteit reduceren door het aantal testpunten te beperken. Ze doen dit door een bewegingsvector te selecteren uit een zeer beperkt aantal mogelijkheden binnen het zoekvenster, en dan het resultaat iteratief te verfijnen door telkens een kleiner zoekvenster gecentreerd rond het voorlopige resultaat te gebruiken. Het Diamond Search algoritme is geïllustreerd in figuur 3.15.

De observatie dat naburige bewegingsvectoren, zowel spatiaal als temporeel, sterk gecorreleerd zijn, leidde tot het gebruik van reeds berekende naburige vectoren als initiële schatting voor de bewegingsvector. '*Motion Vector Field Adaptive Search Technique*' (MVFAST) [HM99] gebuikt deze techniek in combinatie



Figuur 3.15: *Mogelijk verloop van het 'Diamond Search' algoritme voor het blok op positie 1. In de eerste iteratie worden de blokken op de donkerste punten, in de vorm van een ruit gecentreerd rond positie 1 gebruikt. Van deze negen blokken (de acht blokken aangegeven door de ruit + het middelste blok) wordt het blok met de laagste SAD gelecteerd; in dit geval het blok op positie 2. In de volgende iteratie wordt een ruit gecentreerd rond positie 2 gebruikt. Het algoritme stopt wanneer het middelste blok de laagste SAD heeft. De bewegingsvector is dan het verschil tussen de positie van dit blok en het blok op positie 1 (in dit geval dus positie 3 − positie 1).*

met een aangepast Diamond Search algoritme. Door middel van op voorhand vastgelegde criteria kan het algoritme vroegtijdig gestopt worden. Deze criteria leiden tot een efficiënter algoritme, terwijl de sterke correlatie tussen de verschillende bewegingsvectoren in het beeld voor betere compressie zorgt.

Het algoritme genaamd '*Predictive Motion Vector Field Adaptive Search Technique*' (PMVFAST) [TAL01] verbetert MVFAST door ook de mediaan van de naburige bewegingsvectoren, en de bewegingvector van het huidige blok in het voorgaande beeld[12] te gebruiken in de voorspelling. Een andere verbetering is dat de stopcriteria bij MVFAST niet langer op voorhand gedefiniëerd zijn, maar berekend worden in functie van het huidige blok. Die leidt tot betrouwbaardere criteria die onafhankelijke zijn van de video sequentie en gebaseerd zijn op de correlaties tussen naburige blokken.

Enhanced Predictive Zonal Search (EPZS)

Het '*Enhanced Predictieve Zonal Search*' (EPZS) [Tou02] algoritme is een verdere verbetering van PMVFAST door het gebruik van bijkomende bewegingvectoren

[12]MVFAST maakt enkel gebruik van spatiaal naburige bewegingsvectoren, niet van temporele buren.

en betere stopcriteria. Door de hogere efficiëntie wordt ook een eenvoudigere vorm van het Diamond Search algoritme gebruikt, waardoor de implementie minder complex is. Het algoritme wordt in deze sectie in detail besproken.

Zoals eerder gezegd zijn de bewegingsvectoren van spatiaal en temporeel naburige blokken sterk gecorreleerd. De bewegingsvectoren van naburige blokken die reeds berekend zijn kunnen dus gebruikt worden om de bewegingsvector van het huidige blok te voorspellen. Het EPZS algoritme selecteert hieruit de beste kandidaat, en tracht dit resultaat te verfijnen door een lokaal zoekalgoritme, zoals Diamond Search, uit te voeren rond de voorspelde bewegingsvector.

EPZS doet deze voorspelling onder andere aan de hand van de bewegingsvectoren die ook in PMVFAST gebuikt worden. Hiertoe behoren drie spatiaal naburige blokken, namelijk het blok boven, en het blok rechtsboven, en het blok links van het huidige blok. Verder worden ook de mediaan van deze drie vectoren en de $(0, 0)$ bewegingsvector, die dus aangeeft dat er helemaal geen beweging plaatsvindt, gebruikt als voorspellingen. Ook de bewegingsvector van het blok op dezelfde positie als het huidige blok in het vorige beeld wordt door PMVFAST gebruikt.

EPZS breidt deze set nog uit met de bewegingsvectoren van de spatiale buren van het reeds beschouwde blok in het vorige beeld. Door de beweging is de bewegingsvector van het huidige blok misschien niet gecorreleerd met die van het overeenkomstige blok in het vorige beeld, maar wel met die van de blokken daarrond. Stel bijvoorbeeld dat een blok telkens juist een blokbreedte naar rechts opschuift tussen twee opeenvolgende beelden. De bewegingsvector van dit blok is dan gelijk aan de bewegingsvector van het blok links hiervan in het vorige beeld. EPZS gebruikt niet alle aanliggende blokken. Van de acht buren worden de vier diagonale blokken niet gebruikt aangezien deze geen extra voordeel blijken op te leveren ten opzichte van enkel de overige vier.

De laatste vector die in rekening gebracht wordt, is de *acceleratie bewegingsvector*. Deze vector wordt berekend door niet alleen het vorige beeld, maar ook het beeld daarvoor te gebruiken. In beide gevallen wordt de bewegingsvector van het blok op dezelfde positie als het huidige blok gebruikt. De acceleratie bewegingsvector is gelijk aan de vector in het vorige beeld plus het verschil tussen deze vector en de vector van het beeld daarvoor. De vector wordt dus met de differentiaal verhoogd, waardoor niet alleen rekening gehouden wordt met constante snelheden, maar ook met blokken die versnellen.

Deze set van bewegingsvectoren wordt onderverdeeld in drie subsets. Stel dat de bewegingsvector van blok (x, y) in beeld t gezocht wordt. Stel ook dat $V((u, v), w)$ de bewegingsvector van het blok op positie (u, v) in beeld w geeft. De drie subsets zijn dan als volgt opgebouwd:

Subset A bevat de mediaan van de vectoren van de drie spatiaal naburige blokken:

- $mediaan[V((x-1, y), t), V((x, y-1), t), V((x+1, y-1), t)]$

Subset B bevat de rest van vectoren die in PMVFAST gebruikt worden:

- $(0, 0)$

- $V((x-1,y),t)$
- $V((x,y-1),t)$
- $V((x+1,y-1),t)$
- $V((x,y),t-1)$

Subset C bevat de 5 extra bewegingsvectoren die in EPZS geïntroduceerd worden:

- $V((x-1,y),t-1)$
- $V((x,y-1),t-1)$
- $V((x+1,y),t-1)$
- $V((x,y+1),t-1)$
- $V((x,y),t-1)+[V((x,y),t-1)-V((x,y),t-2)]$

Het gebruik van de extra bewegingsvectoren zorgt voor een betere voorspelling, maar introduceert ook extra berekeningen. Door de stopcriteria om het zoekproces vroegtijdig te beëindigen blijft de efficiëntie echter behouden. Het EPZS algoritme bestaat uit vier fases:

1. Test de bewegingsvector die voorgesteld wordt in subset A. Indien deze bewegingsvector voor het huidige blok een SAD oplevert die kleiner is dan een drempelwaarde T_1, beëindig dan het algoritme en neem deze vector als bewegingsvector voor het huidige blok. In het andere geval wordt de volgende fase ingezet.

2. Test nu ook alle bewegingsvectoren die voorgesteld worden in subset B. Selecteer de bewegingsvector die voor het huidige blok de kleinste SAD oplevert. Indien de SAD van deze vector kleiner is dan een drempelwaarde T_2, beëindig dan het algoritme en neem de geselecteerde vector als bewegingsvector voor het huidige blok. In het andere geval wordt de volgende fase ingezet.

3. Test nu ook alle bewegingsvectoren die voorgesteld worden in subset C. Selecteer de bewegingsvector die voor het huidige blok de kleinste SAD oplevert. Indien de SAD van deze vector kleiner is dan een drempelwaarde T_3, beëindig dan het algoritme en neem de geselecteerde vector als bewegingsvector voor het huidige blok. In het andere geval wordt de volgende fase ingezet.

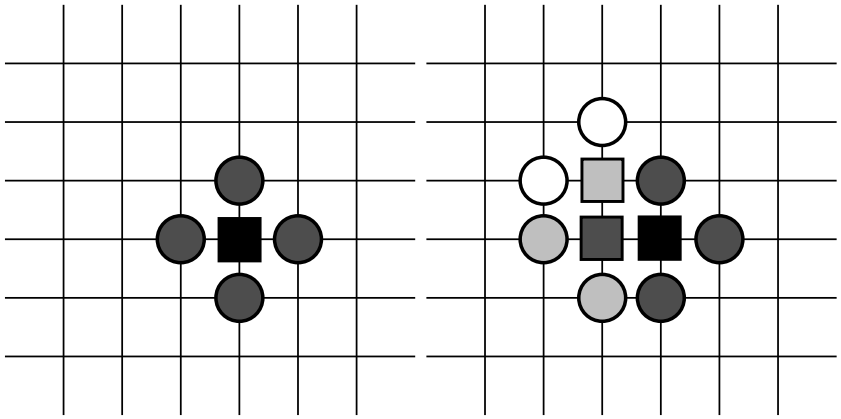
4. Voer een lokaal zoekalgoritme zoals Diamond Search uit, initieel gecentreerd rond de, in de vorige stap geselecteerde, bewegingsvector.

Merk op dat de bewegingsvector met de kleinste SAD ook uit de, in de vorige fase beschouwde, subsets gekozen wordt. In fase 2 wordt dus een bewegingsvector geselecteerd uit $A \cup B$.

Een gebruikelijke waarde voor T_1 voor blokken van 16×16 pixels is 256. T_2 en T_3 worden voor ieder blok afzonderlijk berekend. Een goede keuze voor zowel T_2 als T_3 is het minimun van de voorspellingsfouten, voortgebracht door de

bewegingsvectoren, van de naburige blokken waarvoor reeds een bewegingsvector gevonden is. Zowel de spatiaal naburige als de temporeel naburige blokken kunnen gebruikt worden.

Voor het lokaal zoekalgoritme in fase 4 worden twee varianten voorgesteld in [Tou02]. De eerste methode is een eenvoudig versie van het Diamond Search algoritme zoals geïllustreerd in figuur 3.16. De tweede variant maakt gebruik van een vierkantig patroon, zoals geïllustreerd in figuur 3.17. De EPZS implementatie die gebruikt maakt van het tweede patroon wordt EPZS^2 (EPZS square) genoemd.



Figuur 3.16: *De linkse figuur toont het ruitvormige patroon dat in EPZS gebruikt wordt. De rechtse figuur illustreert het verloop van het lokale zoekalgoritme met dit patroon. Opeenvolgende iteraties zijn aangegeven door lichter wordende kleuren. De posities aangeduid met een vierkant komen overeen met het beste resultaat in de betreffende iteratie. In de eerste iteratie worden dus de vijf donkerste posities gebruikt. De overeenkomstige blokken worden vergeleken met het huidige blok, en de positie van het blok met de kleinste fout wordt geselecteerd. In dit geval geeft de linkse positie het beste resultaat. In de volgende iteratie worden de posities in een ruitvorm rond de geselecteerd positie gebruikt. Het algoritme stopt wanneer de middelste positie het beste resultaat oplevert. In dit geval dus na drie iteraties. De resulterende bewegingsvector is dan het verschil tussen deze positie en de positie van het huidige blok.*

In figuur 3.18 is het resultaat te zien van het geïmplementeerde EPZS algoritme. Merk op de beweging van het been correct geschat werd, ondanks dat deze blokken sterke verschillen vertonen.

3.9 videocompressie

Om te illustreren hoe de beschreven beeldcompressie technieken, samen met de bewegingscompensatie gecombineerd kunnen worden tot een efficiënt videocompressie systeem, wordt hier de MPEG-1 [MPE] standaard toegelicht.

ontworpen als een asymmetrisch systeem; het coderingsproces is veel complexer dan het reconstructieproces.

De standaard definiëert enkel het formaat van de gegevensstroom. Elk proces dat data volgens dit formaat produceert is een geldig MPEG-1 coderingsproces. Hierdoor kan het coderingsproces nog verder evolueren nadat de standaard gepubliceerd is, maar blijft ieder geldig MPEG-1 reconstructieproces in staat de geproduceerde gegevenstroom af te spelen.

De beelden in een, volgens MPEG-1 gecomprimeerde, videosequentie hebben verschillende types. Deze types bepalen of de beelden gecodeerd worden met temporele compressie, of enkel aan de hand van de spatiale informatie in het beeld zelf. Deze beelden worden *niet-intrabeelden* en respectievelijk *intrabeelden* genoemd.

Intrabeelden (I-frames)

Intrabeelden, meestal I-frames genoemd, zijn beelden die gecomprimeerd worden zonder gebruik te maken van andere beelden. Er wordt dus geen temporele compressie toegepast; enkel spatiale informatie wordt gebruikt. De manier waarop intrabeelden gecomprimeerd worden volgens de MPEG-1 standaard lijkt sterk op de manier waarop JPEG beelden comprimeert. Deze intrabeelden worden gebruikt als referentiebeelden bij de bewegingscompensatie.

Niet-intrabeelden (P-frames & B-frames)

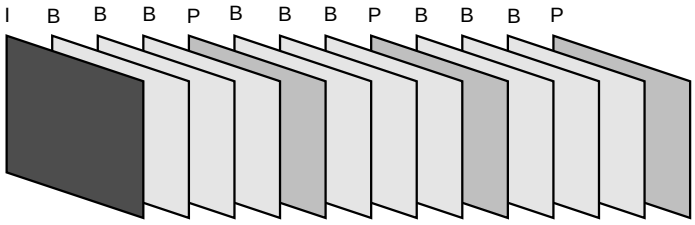
De niet-intrabeelden gebruiken, naast de informatie in het beeld zelf, ook informatie uit beelden die reeds gecomprimeerd zijn. Gebruik makende van bewegingscompensatie, wordt het beeld beschreven aan de hand van de verschillen ten opzichte van een referentiebeeld. Er zijn twee verschillende types van niet-intrabeelden: P-frames (Predicted frames) en B-frames (Bidirectional frames).

P-frames worden gecodeerd aan de hand van het laatst gecodeerde I- of P-frame. Het beeld wordt opgedeeld in blokken van 16×16 pixels, macroblokken genaamd. Voor ieder macroblok wordt een bewegingsvector berekend ten opzichte van het referentiebeeld. Deze bewegingsvectoren, en eventueel ook het residu, bevatten de nodige informatie om het beeld te kunnen reconstrueren.

B-frames worden gecodeerd aan de hand van niet alleen het vorige referentiebeeld, maar ook het eerstvolgende referentiebeeld. Er wordt dus gebruik gemaakt van zowel reeds gecodeerde beelden, als beelden die nog verwerkt moeten worden. Dit betekent dat het coderingsproces het coderen van B-frames zal uitstellen totdat de daaropvolgende beelden beschikbaar zijn. Het gebruik van twee referentiebeelden in plaats van één, zoals bij P-frames, leidt tot betere compressie. Zowel P- als I- frames kunnen als referenentiebeelden gebruikt worden; B-frames nooit.

Figuur 3.19 toont hoe een groep beelden gecodeerd kan worden als I-, P-, of B-frames. Het eerste beeld wordt als I-frame gecodeerd. Er wordt dus geen informatie uit andere beelden gebruikt. Aangezien er in dit geval nog geen andere beelden beschikbaar zijn, is dit de meest logische keuze. De volgende

drie beelden worden als B-frames gecodeerd. Dit wil zeggen dat ze gecodeerd worden aan de hand van het reeds gecodeerde I-frame, en het eerstvolgende P-frame. De drie B-frames kunnen dus pas verwerkt worden nadat het eerste P-frame beschikbaar is. Dit P-frame gebruikt het I-frame als referentie; de drie B-frames gebruiken zowel het I- als het P-frame.



Figuur 3.19: *Een typische aaneenschakeling van I-, P- en B-frames in een MPEG-1 codering.*

Na dit P-frame volgen weer drie B-frames. Er moet dus weer gewacht worden tot het volgende referentiebeeld beschikbaar is. Dit is opnieuw een P-frame. Dit P-frame wordt gecodeerd aan de hand van het vorige referentiebeeld, wat in dit geval ook een P-frame is. Dit betekent dat fouten in het eerste P-frame doorgevoerd worden in de voorspelling van het daaropvolgende P-frame, waardoor het residu groter wordt. Daarom wordt regelmatig een beeld als I-frame gecodeerd. Opeenvolgende B-frames dragen geen fouten over. Ze worden telkens aan de hand van de twee dichtstbijzijnde referentiebeelden gecodeerd, onafhankelijk van elkaar. De fouten in P-frames worden wel overgedragen in de B-frames.

Tot hiertoe werden twee redenen aangehaald om intrabeelden te gebruiken: voor het eerste beeld van de videosequentie[13], en later op regelmatige tijdstippen, om het voorplanten van fouten te beperken. Het regelmatig voorkomen van intrabeelden heeft nog een ander nut: het afspelen van de gecodeerde beelden kan dan vanop willekeurige posities gestart worden. Om het beeld van een willekeurig tijdstip te kunnen reconstrueren, moeten de nodige referentiebeelden immers eerst gereconstrueerd worden. Dit wil zeggen dat alle vorige P-frames, tot en met het eerste I-frame gedecodeerd moeten worden. Een laatste situatie waar intrabeelden gebruikt moeten worden, is bij het wisselen van scène. Het eerste beeld van een andere scène zal immers niet veel gemeen hebben met de laatste beelden van de vorige scène, en zal dus niet efficiënt voorspeld kunnen worden.

Aangezien P- en I-frames gebruikt worden als referentiebeelden voor bewegingscompensatie, worden deze beelden tijdens het coderingsproces reeds gereconstrueerd, nadat ze gecomprimeerd werden. Doordat MPEG-1 gebruik maakt van verlieslatende compressie, zullen de gereconstrueerde beelden niet identiek zijn aan de originiele beelden. Het coderingsproces gebruikt deze dan gereconstrueerde beelden, in plaats van de originele beelden, als referentiebeelden voor

[13] In principe hoeft het eerste beeld van een videosequentie niet als I-frame gecodeerd worden. De eerste beelden kunnen als B-frames, ten opzichte van een daaropvolgend I-frame gecodeerd worden. In ieder geval is het nodig ergens in het begin een intrabeeld te hebben, zodat er een referentie is voor voorspellende codering

de bewegingsschatting. Dit geeft nauwkeurigere resultaten, aangezien het reconstructieproces ook enkel over deze gereconstrueerde beelden zal beschikken.

Aangezien de B-frames zowel vorgaande als volgende referentiebeelden gebruiken, worden de gecodeerde beelden niet in de originele volgorde naar het MPEG-1 bestand weggeschreven. In het geval van figuur 3.19 zal eerst het gecodeerde I-frame weggeschreven worden, gevolgd door het eerste P-frame. Dan pas volgen de eerste drie B-frames. De resulterende volgorde is dus $IPBBBPBBBPBBB$.

Hierdoor wordt het reconstructieproces eenvoudiger. Het I-frame kan dadelijk gereconstureerd worden. Dan volgt het P-frame. Dit P-frame is niet het volgende beeld in de originele sequentie, maar is wel vereist om het volgende beeld te kunnen reconstureren. Het volgende beeld in het MPEG-1 bestand is het eerste B-frame. Dit kan meteen gereconstrueerd worden. Als de beelden in de originele volgorde waren weggeschreven, zouden zowel het I-frame, de drie B-frames als het eerste P-frame ingelezen moeten zijn om het eerste B-frame te kunnen reconstrueren.

Codering van macroblokken

Bij P-frames wordt voor ieder macroblok een bewegingsvector gezocht. Hoewel gewoonlijk enkel het Y-kanaal gebruikt wordt voor de bewegingsschatting, wordt deze bewegingsvector gebruikt voor de drie kanalen in het macroblok. Het macroblok wordt opgesplitst in blokken van 8×8 pixels voor de verdere codering.

Voor ieder 8×8 blok wordt het residu berekend. Dit is het pixelgewijze verschil tussen het te coderen blok, en het, volgens de bewegingsvector, overeenkomstige blok in het referentiebeeld. Het resultaat wordt vervolgens gecodeerd door toepassing van de DCT en kwantisatie, net zoals bij de codering van intrabeelden. De gebruikte kwantisatiematrix is, in tegenstelling tot die van intracodering, echter overal gelijk, zodat iedere coëfficiënt even hard gekwantiseerd wordt.

De DC coëfficiënten worden niet anders behandeld dan de AC coëfficiënten, doordat er minder coherentie is tussen deze waarden. De DC coëfficiënt komt immers overeen met de gemiddelde waarde van het residu-blok, een getal dat dicht bij nul zal liggen aangezien bij de bewegingsschatting de bewegingsvector met de kleinste gemiddelde fout gezocht werd. Alle 64 coëfficiënten worden vervolgens doorlopen volgens het zig-zag patroon, en gecodeerd door middel van entropie codering, net zoals bij de codering van intrabeelden.

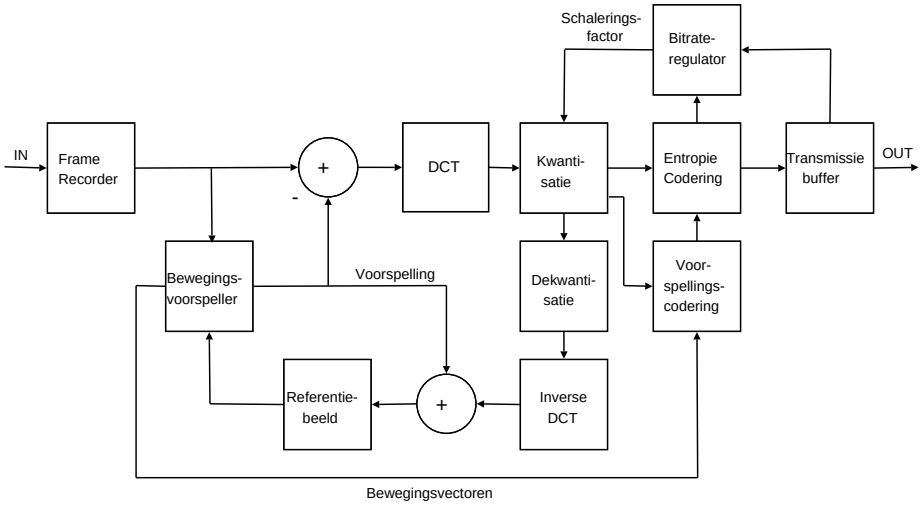
Indien het U- en V-kanaal geschaleerd werden, worden er per macroblok zes blokken van 8×8 pixels gedoceerd: vier voor het Y-kanaal, en voor het $U-$ en $V-$ kanaal telkens één. Als de beweging goed voorspeld werd, zullen veel blokken na kwantisatie uit alleen maar nullen bestaan. Deze blokken moeten dus niet gecodeerd worden, en kunnen vervangen worden door een speciale code die aangeeft dat dit blok, of zelfs het volledige macroblok overgeslagen werd.

De bewegingsvectoren worden voorspellend gecodeerd, door telkens het verschil met de vector van het vorige macroblok te nemen. Wanneer de geschatte beweging sterk aanleunt bij de eigenlijke beweging in de scène, zullen de vectoren immers sterk gecorreleerd zijn. De resulterende differentiale waarden worden vervolgens gecomprimeerd door middel van entropie-codering.

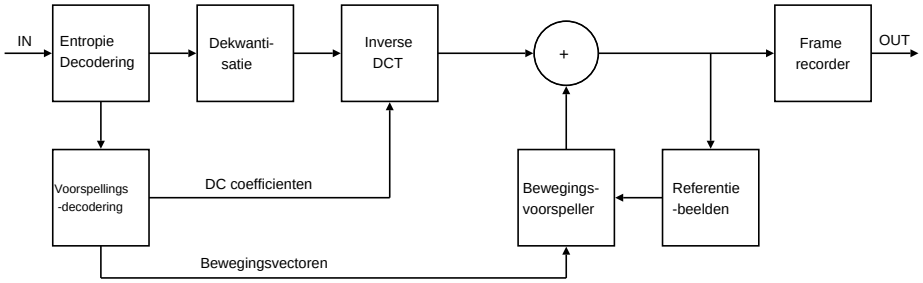
Bij een slechte voorspelling van het macroblok, zal het coderen van het residu veel bits in beslag nemen. Wanneer het aantal bits voor de codering van het residu, plus het aantal bits voor de bewegingsvector, groter is dan het aantal bits dat nodig is om het volledige macroblok te coderen, wordt er overgeschakeld op intracodering. De blokken in het macroblok worden dan gecodeerd zoals dit in een intrabeeld zou gebeuren. De bewegingsvector moet in dit geval niet gecodeerd worden.

Bij de codering van B-frames worden ook de bewegingsvectoren ten opzichte van het volgende referentiebeeld berekend. Dit biedt de mogelijkheid een macroblok te coderen aan de hand van het volgende referentiebeeld, in plaats van het vorige. Er bestaat ook nog en vierde methode voor de codering van macroblokken in B-frames: geïnterpoleerde voorspelling. Hierbij wordt het macroblok voorspeld aan de hand van het blok dat geconstrueerd wordt door de twee referentieblokken uit te middelen. Het te coderen residu is in dit geval dus gelijk aan het pixelgewijze verschil tussen het eigenlijke blok, en het pixelgewijze gemiddelde van de twee blokken die door de bewegingsvectoren bepaald worden.

Als één van deze drie voorspellende methodes een voorspelling geeft die goed genoeg is, kan de codering van het residu overgeslagen worden. Er moet dan nog wel aangegeven worden welke methode gebruikt werd. In het andere geval wordt het beste resultaat van deze drie methodes vergeleken met intracodering. De methode die in totaal tot het minste bits leidt, wordt geselecteerd. In dit geval wordt met een korte bitcode aangegeven welke methode gebruikt werd, gevolgd door het residu of, in het geval van intracodering, het volledige blok. Het volledige MPEG-1 coderingsproces is schematisch weergegeven in figuur 3.20. Het reconstructieproces is weergegeven in figuur 3.21.



Figuur 3.20: *Het volledige proces voor de generatie van gecomprimeerde MPEG-1 videobeelden.*



Figuur 3.21: *Het reconstructieproces voor gecomprimeerde MPEG-1 videobeelden.*

Hoofdstuk 4

Implementatie

In dit hoofdstuk wordt beschreven hoe de bestaande videocompressie technieken uitgebreid kunnen worden voor de implementatie van een 3D-video compressiesysteem.

4.1 Uitbreiding naar 3D

Stel dat een scène gefilmd wordt met n camera's, gepositioneerd op gelijke intervallen langs een cirkel. Als de beelden van de camera's op tijdstip t achter elkaar afgespeeld worden, in de volgorde van de camera's langs de cirkel, onstaat een videofragment waarbij de scène 'bevroren' lijkt, terwijl de camera errond cirkelt; een effect dat in fims wel eens gebruikt wordt.

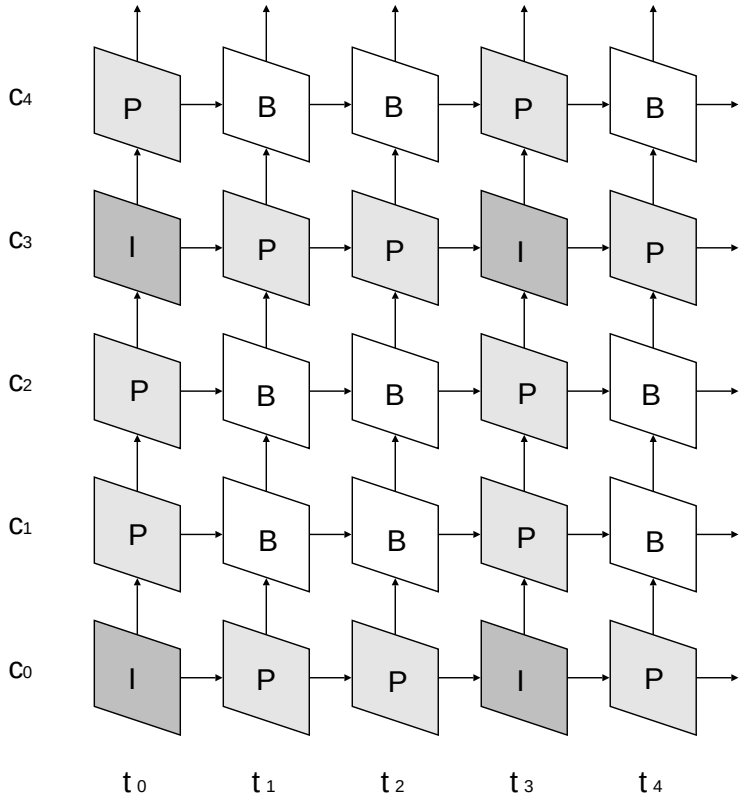
Deze transformatie van spatiale naburigheid naar temporele naburigheid wordt waargenomen als een beweging doorheen de tijd. Zoals in het hoofdstuk over videocompressie besproken werd, kan beweging in een videofragment gebruikt worden om redundante informate te verwijderen. Op dezelfde manier kan dus ook redundante spatiale informatie verwijderd worden.

Opeenvolgende beelden die enkel van elkaar verschillen door een lichte beweging van de camera, kunnen efficiënt gecomprimeerd worden door middel van bewegingscompensatie. Door spatiale naburigheid in 3D-video te behandelen zoals temporele naburigheid in traditionele videofragmenten, kan op dezelfde manier compressie van de 3D-videobeelden op een vast tijdstip bekomen worden.

Dit kan bereikt worden door het schema van I-, P- en B-frames bij MPEG, zoals in figuur 3.19, naar een hoger dimensionaal schema uit te breiden. Dit is geïllustreerd in figuur 4.1. Bij sommige camera's worden de beelden gecomprimeerd aan de hand van informatie uit beelden van andere camera's; bij andere camera's wordt enkel de informatie uit de eigen beelden gebruikt. Dit is analoog aan het onderscheid tussen niet-intrabeelden en intrabeelden.

De beelden van camera c_0 worden als I- en P- frames gecomprimeerd, zonder beelden van andere camera's te gebruiken. Bij camera c_1 wordt als eerste beeld een P-frame gebruikt in plaats van een I-frame. Dit beeld wordt gecomprimeerd

aan de hand van het I-frame in camera c_0 op datzelfde tijdstip t_0. Op tijdstippen t_1 en t_2 worden in camera c_1 twee B-frames gebruikt, in plaats van twee P-frames. Deze frames worden gecomprimeerd aan de hand van het vorige referentiebeeld in c_1, het P-frame op tijdstip t_0, en het P-frame uit camera c_0 op datzelfde tijdstip t_1 of t_2.



Figuur 4.1: *Compressieschema voor spatiaal naburige camera's c_0, c_1, c_2, c_3, c_4, ... op opeenvolgende tijdstippen t_0, t_1, t_2, t_3, t_4, ...*

Het P-frame van camera c_2 op tijdstip t_0 wordt ten opzichte van het overeenkomstige P-frame in camera c_1 gecodeerd. De hieropvolgende B-frames worden echter ten opzichte van de P-frames op tijdstip t_1 en t_2 van camera c_0 gecomprimeerd.

Dit schema wordt in beide richtingen herhaald. In de tijdsdimensie wordt een referentiebeeld steeds afgewisseld met twee voorspelde beelden: telkens IPP of PBB. Anderzijds worden camera's die enkel hun eigen beelden gebruiken voor de compressie telkens afgewisseld met twee camera's die ook de beelden van de voorgaande camera gebruiken.

Het aantal voorspelde beelden tussen twee referentiebeelden en het aantal voorspelde camera's tussen twee referentiecamera's kan uiteraard gewijzigd worden. Dit hangt af van de toepassing waarin de 3D-video beelden gebruikt worden. Hoe meer voorspeld wordt, hoe meer voorgaande beelden gebruikt moeten wor-

den tijdens de reconstructie van een beeld, waardoor het reconstructieproces complexer wordt.

Stel bijvoorbeeld dat het voor de gebruiker mogelijk moet zijn om vrij rond te lopen in de door de camera's vastgelegde wereld. Om het door de gebruiker gekozen standpunt te genereren worden de beelden van de naburige camera's gebruikt. De beelden van deze camera's moeten dus gecomprimeerd worden, waardoor ook de beelden van andere camera's die als referentie dienden gereconstrueerd moeten worden. Aangezien het compressieschema uitgaat van spatiale naburigheid, zullen de referentiecamera's gelukkig soms dezelfde zijn als de naburige camera's die al nodig waren om de tussenliggende beelden te genereren.

Als de gebruiker nu beweegt doorheen de virtuele wereld, moet er steeds overgeschakeld worden naar andere camera's om de tussenliggende beelden te genereren, die telkens gedecodeerd worden aan de hand van misschien nog een aantal andere camera's. Het is dus belangrijk om snel een beeld op een willekeurig tijdstip van een willekeurige camera te kunnen reconstrueren.

Om bijvoorbeeld het beeld van camera c_2 op tijdstip t_2 (kortweg (c_2, t_2)) uit figuur 4.1 te kunnen reconstrueren, zijn beelden (c_2, t_0) en (c_0, t_2) vereist, die op hun beurt beelden (c_1, t_0) en (c_0, t_1) nodig hebben, die op hun beurt dan weer beeld (c_0, t_0) gebruiken voor de reconstructie.

Een ander belangrijk punt in de kwaliteit van de beelden. De voorspelde beelden zijn gewoonlijk van mindere kwaliteit dan de intrabeelden. Bij de compressie wordt er immers vanuit gegaan dat kleine fouten tolereerbaar zijn, zodat het residu van verscheidene blokken achterwege gelaten kan worden. Bij 3D-video moet rekening gehouden worden met het feit dat de gereconstrueerde beelden nadien nog als referentiebeelden gebruikt worden om tussenliggende beelden te genereren. De beelden mogen dus niet te hard afwijken van de oorspronkelijke beelden, onafhankelijk van het feit dat het beeld als I-, p- of B-frame gecodeerd werd.

4.1.1 Testsysteem

In de implementatie werd telkens maar één beeld voorspellend gecodeerd tussen twee referentiebeelden. Het schema van figuur 4.1 wordt dus herleid naar $IPIPIP$ voor een camera met intracodering, en $PBPBPB$ voor een camera waarvan de beelden gecomprimeerd worden met behulp van informatie uit beelden van een andere camera. Ook bij de camera's werd afgewisseld tussen intra- en niet-intracodering. Hierdoor kan een willekeurig beeld van een willekeurige camera gemakkelijk gereconstrueerd worden.

Intrabeelden

De compressie van I-frames verloopt ongeveer zoals beschreven in de sectie over beeldcompressie. Het beeld wordt kanaal per kanaal vewerkt. De U- en V-kanalen worden telkens met de oorspronkelijke resolutie gecomprimeerd.

Eerst wordt het beeld opgesplitst in blokken van 8×8 pixels, waarna ieder blok de discrete cosinus tranformatie ondergaat. De resulterende coëfficiënten worden vervolgens gekwantiseerd. De gebruikte kwantisatie matrix is weergegeven

in figuur 4.2. Als iedere coëfficiënt door de overeenkomstige kwantisatiewaarde gedeeld wordt, veroorzaakt dit tamelijk agressieve kwantisatie. Daarom wordt iedere waarde in de kwantisatie matrix eerst vermenigvuldigd met een schaleringswaarde, in dit geval 1/5.

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Figuur 4.2: *Standaard MPEG-1 kwantisatie matrix voor intracodering.*

De resulterende DC- en de AC-coëfficiënten worden afzonderlijk gecodeerd. De AC-coëfficiënten van ieder blok worden volgens het zig-zag patroon uit figuur 3.13 doorlopen, en vervolgens gecodeerd met de aangepaste Huffman codering zoals die beschreven staat in de uitleg over JPEG. De DC-coëfficiënten van alle blokken in het beeld worden samen gecodeerd. De codering is verder analoog aan die van de AC-coëfficiënten, behalve dat telkens het verschil met de vorige waarde gecodeerd wordt, in plaats van de eigenlijke coëfficiënt.

Bewegingscompensatie

De detectie van de beweging in de scène gebeurt met behulp van het EPZS[2] algoritme. Voor ieder macroblok van 16×16 pixels wordt een bewegingsvector berekend, die de beweging ten opzichte van een gelijkaardig blok in het referentiebeeld voorstelt. In figuur 3.18 is de benadering van een te comprimeren beeld te zien, uitgaande van een referentiebeeld en de gevonden bewegingsvectoren.

Codering van bewegingsvectoren

Zowel om de berekentijd van de bewegingsschatting, als het aantal bits, nodig om een bewegingsvector te coderen, te beperken, wordt op voorhand een maximale waarde voor zowel de horizontale component als de vertikale component van de bewegingsvector vastgelegd.

De bewegingvectoren worden voorspellend gecodeerd, gebuik makend van vectoren boven, en linksboven, en links van de te coderen vector. De differentiale waarden worden dan met behulp van Huffman codering gecodeerd. De hiervoor gebruikte tabel met bitcodes wordt opgesteld aan de hand van de op voorhand vastgelegde maximale waarden van de vectoren. Het reconstructieproces kan deze tabel dus zelf opstellen indien de maximale beweging bekend is.

Voor de horizontale en de vertikale verplaatsingen worden aparte tabellen opgesteld. Iedere waarde krijgt een waarschijnlijkheid toegekend, die logaritmisch daalt bij stijgende verplaatsingen. De waarschijnlijkheid van 0 is dus dubbel zo groot als

die van -1 of 1, en vier keer zo groot als die van -3, 2, 2 of 3. Aan de hand van deze waarschijnlijkheden wordt de tabel moet codewoorden opgesteld, gebruik maken van het Huffman algoritme.

In de implementatie werd de maximale verplaatsing voor temporele beweging op $(32, 16)$ gezet, en voor spatiale beweging op $(64, 16)$. De eerste vector stelt dus een maximum op de snelheid van bewegingen die correct gedetecteerd kunnen worden. De tweede vector bepaalt de maximale spatiale verplaatsing, en hangt dus af van de cameraopstelling.

P-frames

P-frames worden opgedeeld in marcroblokken van 16×16 pixels, waarvoor telkens bijhorende bewegingsvector berekend wordt. Voor elk 8×8 blok binnen het macroblok wordt vervolgens het residu berekend. Dit zijn dus twaalf blokken; vier voor ieder kanaal.

Deze residu-blokken worden vervolgens analoog aan de blokken van intrabeelden gecodeerd. Zoals eerder vermeld heeft het in dit geval niet veel zin de DC-coëfficiënten anders te behandelen, dus worden ze gewoon samen met de AC-coëfficiënten verwerkt. De gebruikte kwantisatie matrix is weergegeven in figuur 4.3. Ook hier kunnen de kwantisatiewaarden geschaleerd worden om minder aggressieve kwantisatie te verkrijgen. In het testsysteem werd iedere waarde in de matrix eerst vermenigvuldigd met 1/2.

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

Figuur 4.3: *Standaard MPEG-1 kwantisatie matrix voor de codering van residublokken.*

Als het voorspelde blok sterk afwijkt van het te coderen blok zal de codering van het residu veel bits in beslag nemen. In dit geval wordt het blok niet aan de hand van het referentiebeeld gecomprimeerd, maar wordt het blok gecodeerd alsof het tot een intrabeeld behoorde. Deze beslissing wordt voor alle blokken binnen het macroblok tesamen genomen, aangezien al deze blokken met behulp van dezelfde bewegingsvector voorspeld worden. Voor ieder blok wordt in de resulterende bitstoom dan ook aangegeven of het al dan niet voorspellend gecodeerd wordt.

B-frames

De B-frames worden ten opzichte van het vorige P-frame in dezelfde videosequentie, en ten opzichte van het overeenkomstige P-frame van een naburige

camera gecomprimeerd. Bij B-frames worden voor ieder macroblok vier coderingsmanieren geëvalueerd: intracodering, voorspellende codering te opzichte van elke referentiebeeld, en geïnterpoleerde codering ten opzichte van allebei de referentiebeelden.

4.2 Compressie met diepte-informatie

Het berekenen van beelden voor een willekeurige gekozen camerastandpunt vereist 3D-informatie van de scène. Een scène-reconstructie algoritme kan deze informatie afleiden uit de beschikbare beelden van de verschillende camera's op een bepaald tijdstip, zoals beschreven in het hoofdstuk over 3D-video. Deze algorimtes berekenen voor elke pixel een diepte-waarde, zodat het overeenkomend 3D-punt bepaald wordt. Zoals eerder gezegd, komt iedere pixel overeen met een rechte in de drie-dimensionale ruimte. De berekende diepte-waarde duidt de afstand op deze rechte aan die afgelegd moet worden om het overeenkomende 3D-punt te vinden.

Een andere mogelijke oplossing bestaat erin voor iedere pixel p in camera A de verplaatsing te berekenen om de overeenkomstige pixel q in camera B te vinden. p en q zijn dan beide de projectie van hetzelfde 3D-punt op het beeldvlak van camera A respectievelijk camera B. Dit 3D-punt bevindt zich op het snijpunt van de twee rechten die bij deze twee pixels behoren. Als dus bekend is hoeveel pixel p verplaatst moet worden om de positie van de overeenkomstige pixel q in beeld B te bekomen, kan het 3D-punt berekend worden door de intersectie van de twee rechten te berekenen.

4.2.1 Tussenliggende beelden

Aan de hand van de beelden van twee camera's, en de berekende verplaatsingen voor iedere pixel, kunnen tussenliggende beelden gegenereed worden. Dit kan verwezelijkt worden door de verplaatsingen te interpoleren, en gebruik makende van de resulterende verplaatsingen de pixels op de juiste posities te 'verschuiven'.

Stel dat zowel de beelden van twee camera's A en B, als de verplaatsing die iedere pixel van A moet ondergaan om op de positie van de overeenkomstige pixel in B terecht te komen gekend zijn. Het beeld van B, of op zijn minst toch een benadering van dit beeld, wordt verkregen door iedere pixel in het beeld van A te verschuiven volgens de gegeven verplaatsingen. Een beeld van de camera die zich in het midden tussen A en B zou bevinden kan op dezelfde manier verkegen worden door de grootte van de verschuivingen te halveren.

In het algemeen kunnen tussenliggende beelden gegenereerd worden door de grootte van de verschuivingen te interpoleren afhankelijk van de positie van de gewenste camera ten opzichte van A en B. Als bovendien ook de verplaatsingen om een pixel uit een beeld van B naar de positie van de overeenkomstige pixel in A beschikbaar zijn, kunnen nauwkeurigere schattingen gemaakt worden door beide voorspellingen uit te middelen. Als een pixel p uit A en een pixel q uit B beide op dezelfde pixel r in het tussenliggende beeld afgebeeld worden, wordt aan r de gemiddelde kleur van p en q toegekend. Als de tussenliggende camera

niet precies in het midden tussen A en B ligt, wordt een gewogen gemiddelde gebruikt, afhankelijk van de positie van de camera ten opzichte van A en B.

4.2.2 Compressie

Aangezien het genereren van deze diepte-informatie een complex proces is, is het interessant deze gegevens op voorhand te berekenen, zodat in realtime beelden van willekeurige camerastandpunten gegeneerd kunnen worden. Deze informatie kan dan samen met de bijbehorende videosequenties gecomprimeerd worden.

Zoals in [GS98] gesuggereerd wordt, kunnen deze gegevens echter ook gebruikt worden om de videosequenties te comprimeren. Als de beelden van camera A, en de verplaatsingen om de beelden van A in die van B om te vormen beschikbaar zijn, kunnen de beelden van B aan de hand van deze informatie berekend worden, en hoeven ze dus niet gecodeerd te worden.

Dit is in principe net hetzelfde als het beeld van camera B, uitgaande van het overeenkomstige beeld in camera A, te voorspellen met behulp van bewegingscompensatie. Alleen wordt de beweging nu per pixel in plaats van per macroblok bepaald, en moet deze informatie niet meer berekend worden door een bewegingsschattingsalgoritme.

4.2.3 Testsysteem

Voor de generatie van de 3D-informatie van de scène wordt gebruikt gemaakt van de implementatie van Vladimir N. Kolmogorov [KZG03] van het minimale-snede-algoritme dat beschreven wordt in [KZ02].

Gegeven zijn drie camera's A, B en C, en de berekende verplaatingen van A naar B, van B naar A, van B naar C en van C naar B. Camera's A en C kunnen aan de hand van traditionele videocompressiessystemen gecomprimeerd worden. Camera B wordt berekend uit de gegeven informatie, zodat enkel het residu opgeslagen moet worden. Er wordt dus telkens afgewisseld tussen codering zonder informatie uit beelden van andere camera's, en voorspellende codering.

Camera B wordt voorspeld aan de hand van de beelden van camera's A en C, en de verplaatsingen van A naar B en van B naar C. Deze berekening gebeurt analoog aan de berekening van beelden van tussenliggende camerastandpunten. B bevindt zich immers tussen A en C. Het enige verschil is dat er geen verplaatsingen van A en C en van C naar A beschikbaar zijn. Er zijn echter wel verplaatsingen van A naar B en van C naar B beschikbaar. Dit maakt de berekeningen alleen maar gemakkelijker, aangezien dit juist de informatie is die nodig is. De verplaatsingen moeten in dit geval dus niet meer geïnterpoleerd worden.

De pixels van B die niet ingekleurd werden, doordat er geen enkele pixel uit A of C op de betreffende positie afgebeeld werd, worden nadien ingekleurd door te interpoleren over de kleuren van de naburige pixels. Het resultaat is dan de voorspelling van het beeld van camera B.

Het beeld van B wordt in blokken van 8×8 pixels opgedeeld, en voor ieder blok wordt het residu berekend. De compressie van dit residu verloopt op dezelfde manier als bij de codering van P-frames in het eerste testsysteem.

Als laatste moeten ook de verplaatsingen voor iedere camera gecodeerd worden. Deze verplaatsingen kunnen voorgesteld worden als afbeeldingen die aan ieder pixel een kleurwaarde toekennen die overeenstemt met de verplaatsing van de pixel op die positie in het camerabeeld. Hiervoor kunnen deze gegevens dus ook gemakkelijk gecomprimeerd worden door gebruik te maken van een traditioneel videocompressiessyteem.

Hoofdstuk 5

Resultaten

In dit hoofdstuk worden de resultaten van de hiervoor toegelichte implementaties besproken. Naar het eerst besproken 3d-video compressiesysteem, waarbij het IPB schema van MPEG werd uitgebreid, zal in deze uitleg verwezen worden als systeem 1; naar het compressiesysteem dat gebruik maakt van de 3D-informatie van de scène zal verwezen worden als systeem 2.

Het gebruikte beeldmateriaal is de 'Kung-Fu girl' dataset, een synthetische testsequentie voor 3D-video die door het 'Graphics-Optics-Vision' departement van het 'Max-Planck-Institut für Informatik' [GrO] beschikbaar gesteld werd voor onderzoek naar 3D-video.

De dataset bestaat uit 25 videosequenties, overeenkomstig met 25 camerastandpunten rondom de scène. De cameraposities zijn verdeeld over een halve bol rondom de scène. De resolutie van de beelden is 320×240 pixels. Ongecomprimeerd neem elk beeld dus $320 \times 240 \times 3 = 230400$ bytes, ofwel 225 KB, in. De resultaten werden berekend met een AMD Athlon 64 processor met een kloksnelheid van 2GHz.

5.1 Testsysteem 1

De videosequenties van de eerste twee camera's werden gecomprimeerd met systeem 1. De gereconstrueerde beelden zijn te zien in figuur 5.1 en figuur 5.2 voor camera 'b01' en respectievelijk camera 'b02'. De beelden van 'b01' werden gecomprimeerd zonder informatie uit beelden van andere camera's te gebruiken. De beelden van camera 'b02' werden gecomprimeerd met de beelden van 'b01' als referentie.

In tabel 5.1 en tabel 5.2 zijn de gegevens van het compressieproces weergegeven. De gecomprimeerde I-frames in camera 'b01' namen gemiddeld 6.705 KB in beslag. De overeenkomstig P-frames in camera 'b02', die ten opzicht van deze I-frames voorspeld werden, namen gemiddeld 4.789 KB in beslag. De P-frames in 'b01' en de overeenkomstige B-frames in 'b02' werden gecomprimeerd tot 2.786 KB en respectievelijk 2.776 KB.

Type	Aantal	Totaal bits	Totaal KB	KB/beeld	Tijd (s)
I-frame	100	5492545	670.477	6.705	1.414
P-frame	100	2281737	278.532	2.786	5.785
Totaal	200	7774282	949.009	4.745	7.199

Tabel 5.1: *Gegevens van de compressie van de beelden van camera 'b01'. De kolommen bevatten van links naar rechts: het type van de compressie, het aantal beelden volgens dit type, het totaal aantal bits nodig om deze beelden te representeren, het totaal aantal bits omgerekend in kilobytes, de gemiddelde grootte van een beeld, en de totale tijd die nodig was om deze beelden te comprimeren.*

Type	Aantal	Totaal bits	Totaal KB	KB/beeld	Tijd (s)
P-frame	100	3923268	478.915	4.789	11.464
B-frame	100	2273920	277.578	2.776	16.077
Totaal	200	6197188	756.493	3.782	27.541

Tabel 5.2: *Gegevens van de compressie van de beelden van camera 'b02'. De kolommen bevatten van links naar rechts: het type van de compressie, het aantal beelden volgens dit type, het totaal aantal bits nodig om deze beelden te representeren, het totaal aantal bits omgerekend in kilobytes, de gemiddelde grootte van een beeld, en de totale tijd die nodig was om deze beelden te comprimeren.*

satie er niet volledig in slaagde de fout van de voorspelling weg te werken voor het oog.

Dit kan te maken hebben met het feit dat de gebruikte beelden gesynthetiseerde beelden zijn. De DCT presteert vooral goed op fotgrafische beelden, waar de randen en de kleurovergangen natuurlijker zijn dan bij de meeste computergegereerde beelden. Enkele tests met dezelfde geïmplementeerde compressietechnieken op fotografische videobeelden leverde betere compressie op, met betere kwaliteit. Het is echter niet zo eenvoudig fotografisch beeldmateriaal vast te leggen dat geschikt is voor 3D-video, waardoor hier gebruik gemaakt werd van de beschikbare gesynthetiseerde beelden.

Het gebruik van een wavelet transformatie [SDS95a, SDS95b, DJL92] in plaats van de DCT zal hier waarschijnlijk betere resultaten geven, aangezien wavelet transformaties op het volledige beeld ineens toegepast worden, en dus niet blok per blok. Hierdoor worden er dus ook geen blokpatronen gevormd.

5.2 Testsysteem 2

Voor testsysteem werd gebruik gemaakt van de beelden uit camera's 'b01', 'b02' en 'b03'. De betreffende beelden zijn weergegeven in figuur 5.3. De gegenereerde 3D-informatie is weergegeven in figuur 5.4.

Uitgaande van deze informatie werden de beelden van camera 'b02' voorspeld. De voorspelling van het eerste beeld uit camera 'b02' is te zien in figuur 5.3. De fout van de voorspelling wordt gegeven door het residu in figuur 5.5. Na

Hoofdstuk 6

Conclusie

De bekomen resultaten illustreren dat de extra ruimteljke dimensie, net zoals de temporele dimensie, uitgebuit kan worden om compressie te verkrijgen.

De eerst voorgestelde methode maakt het mogelijk de 3D-video data in realtime te comprimeren, afhankelijk van het aantal processoren en het aantal camera's. De beelden kunnen dus meteen gecomprimeerd worden nadat ze door de camera's vastgelegd worden.

De complexteit van het coderingsproces stijgt lineair met het aantal camera's. Om in realtime te kunnen comprimeren, moeten er dus voldoende processoren beschikbaar zijn, afhankelijk van het aantal camera's. Het is logisch dat de benodigde rekenkracht stijgt per extra camera, aangezien iedere extra camera een extra beeld oplevert dat op datzelfde tijdstip gecomprimeerd moet worden.

De operaties die uitgevoerd worden om de beelden van één camera te comprimeren, zijn identiek aan de operaties die klassieke videocompressiesystemen uitvoeren. Het enige verschil is dat de gebruikte referentiebeelden niet alleen beelden zijn die door het compressieproces van de betreffende camera zelf verwerkt werden, maar ook beelden die door een ander proces, dat instaat voor de compressie van de beelden van een andere camera, verwerkt werden.

Als er dus vanuit gegaan wordt dat de referentiebeelden steeds beschikbaar zijn op het moment dat het coderingsproces van de betreffende camera deze nodig heeft, is de benodigde tijd voor de compressie van één camera gelijk aan de benodigde tijd voor de compressie van een traditionele videosequentie, met een tradioneel compressiesysteem.

Voor een camera waarvan de beelden als I- en P-frames gecomprimeerd worden, werden ongeveer 28 beelden per seconde verwerkt. Voor een sequentie met P- en B-frames ongeveer 7 beelden per seconde. Bestaande compressiesystemen laten zien dat dit proces nog efficiënter geïmplementeerd kan worden, onder andere door sommige delen rechtstreeks in efficiënte, processorspecifieke machinetaalinstructies te schrijven, en door het gebruik van programmeerbare grafische hardware. Deze systemen laten toe minstens één videosequentie in realtime te comprimeren op één processor.

De codering van B-frames is in deze implementatie tamelijk traag doordat

telkens alle mogelijke coderingsmethoden overlopen worden. Dit kan vermeden worden door ruwe schattingen te maken van het aantal bits dat iedere methode zal opleveren, en aan de hand daarvan enkel de geschikte methode uit te voeren. In de implementatie worden blokken waarvan de voorspelling nog steeds een hoge SAD oplevert meteen geselecteerd voor intracodering. Meer verfijnde methodes zouden de berekentijd nog verder kunnen reduceren.

Bij het tweede testsysteem werd verondersteld dat er ook 3D-informatie van de scène beschikbaar is. Dit levert betere compressie op dan de bewegingscompensatie die in testsysteem 1 gebruikt werd. Het eerste beeld van camera 'b02' werd door testsyteem 1 in 4.269 KB gecodeerd, en door testsyteem 2 in 2.841 KB. De mate van compresie en de kwaliteit van de resulterende videobeelden is bij het tweede testsysteem echter sterk afhankelijk van de kwaliteit van de gegenereerde 3D-informatie.

In tegenstelling tot het eerste testsysteem, kan het tweede testsysteem geen realtime compressie uitvoeren. De berekening van de 3D-informatie, uitgaande van twee camerabeelden, duurt ongeveer 8 seconden. Om dit systeem in realtime uit te voeren, zullen dus snellere algortimes ontwikkeld moeten worden. Het voordeel van het tweede systeem is dat de 3D-informatie samen met de videosequenties gecomprimeerd kan worden, waardoor de generatie van beelden vanuit willekeurige camerastandpunten uit de gecomprimeerde data gemakkelijk in realtime kan gebeuren.

Bibliografie

[AV89] N. Ahuja and J.E. Veenstra. Generating octrees from object silhouettes in orthographic views. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(2):137–149, 1989.

[BK01] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *EMMCVPR '01: Proceedings of the Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 359–374, London, UK, 2001. Springer-Verlag.

[Col96] R.T. Collins. A space-sweep approach to true multi-image matching. In *CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, page 358, Washington, DC, USA, 1996. IEEE Computer Society.

[CP] I. Craw and J. Pulham. Huffman codes. `http://www.maths.abdn.ac.uk/~igc/tch/mx4002/notes/node59.html`.

[DJL92] R.A. Devore, B Jawerth, and B.J. Lucier. Image compression through wavelet transform coding. *IEEE Trans. Information Theory*, 38, mar 1992.

[Dye01] C.R. Dyer. Volumetric scene reconstruction from multiple views. In L.S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, 2001.

[GrO] Graphics-Optics-Vision, Max-Planck-Institut für Informatik. `http://www.grovis.de`.

[GS98] N. Grammalidis and M.G. Strintzis. Disparity and occlusion estimation in multiocular systems and their coding for the communication of multiview image sequences. *IEEE Trans. on Circuits and Systems for Video Technology*, 8(3):327–344, jun 1998.

[Hab70] A. Habibi. Comparison of nth-order dpcm encoder with linear transformations and block quantization techniques. *IEEE Trans. on Communication Technology*, COM-19(6):948–956, dec 1970.

[HM99] P.I. Hosur and K.K. Ma. A new diamond search algorithm for fast block matching motion estimation. *Communications and Signal Processing (ICICS '99)*, dec 1999.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. *ireproc*, 40(9):1098–1101, sep 1952.

[ISO] International Organization for Standardization (ISO). `http://www.iso.org/`.

[ITUa] International Telecommunication Union (ITU). `http://www.itu.int/home/`.

[ITUb] International Telecommunication Union (ITU) Recommendation T.81. `http://www.w3.org/Graphics/JPEG/itu-t81.pdf`.

[JF62] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[JJ81] J.R. Jain and A.K. Jain. Displacement measurement and its application in interframe image coding. *IEEE Transactions on Communication*, COM-29(12):1799–1808, dec 1981.

[KIHI81] T. Koga, K. Iinuma, A. Hirano, and Y. Iijima. Motion compensated interframe coding for video conferencing. In *NTC' 81 Conference Record*, pages G5.3.1–G5.3.5, 1981.

[KS00] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *Int. J. Comput. Vision*, 38(3):199–218, 2000.

[KZ01] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions via graph cuts. pages 508–515, 2001.

[KZ02] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III*, pages 82–96, London, UK, 2002. Springer-Verlag.

[KZG03] V. Kolmogorov, R. Zabih, and S.J. Gortler. Generalized multi-camera scene reconstruction using graph cuts. In A. Rangarajan, M.A.T. Figueiredo, and J. Zerubia, editors, *EMMCVPR*, volume 2683 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 2003.

[Mat] Mathworld. `http://mathworld.wolfram.com/`.

[MBR+00] W. Matusik, C. Buehler, R. Raskar, S.J. Gortler, and L. McMillan. Image-based visual hulls. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[MPE] Moving Picture Experts Group (MPEG). `http://www.chiariglione.org/mpeg/`.

[Nyq28] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47:617–644, apr 1928.

[PNG] Portable Network Graphics (PNG) Specification. `http://www.w3.org/TR/PNG/`.

[Roy99] S. Roy. Stereo without epipolar lines: A maximum-flow formulation. *Int. J. Comput. Vision*, 34(2-3):147–161, 1999.

[SD99] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 25(3), November 1999.

[SDS95a] E.J. Stollnitz, T.B. DeRose, and D.H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[SDS95b] E.J. Stollnitz, T.B. DeRose, and D.H. Salesin. Wavelets for computer graphics: A primer, part 2. *IEEE Computer Graphics and Applications*, 15(4):75–85, 1995.

[SH] J. Starck and A. Hilton. Virtual view synthesis from multiple view video sequences. Technical report, Centre for Vision, Speech and Signal Processing, University of Surrey.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[SVZ00] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 345–353, 2000.

[Sym04] P. Symes. *Digital Video Compression*. McGraw-Hill, 2004.

[TAL01] A.M. Tourapis, O.C. Au, and M.L. Liou. Predictive motion vector field adaptive search technique (PMVFAST) - enhancing block based motion estimation. In *Proceedings of Visual Communications and Image Processing 2001 (VCIP'01)*, pages 883–982, jan 2001.

[Tou02] A.M. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. In C. C. Jay Kuo, editor, *VCIP*, volume 4671 of *Proceedings of SPIE*, pages 1069–1079. SPIE, 2002.

[ZM00] S. Zhu and K.K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–294, 2000.