NTNU
Det skapende universitet

3D-videokoding med kontekstbasert, adaptiv, binæraritmetisk koding

Espen Falkevik

Master i elektronikk
Oppgaven levert: Juni 2006
Hovedveileder: Tor Audun Ramstad, IET
Medveileder(e): Arild Fuldseth, TANDBERG

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

Oppgaven tar utgangspunkt i en tidligere utviklet 3D delbåndskoder for video, og har som formål å videreutvikle denne. Videreutviklingen skal i hovedsak skje ved å innføre en mer egnet kodingsmetode for lavpassdelbåndet, samt ved å innføre en tilpasset versjon av Context-Based Adaptiv Binary Arithmetic Coding (CABAC). Oppgaven har som formål å se hvordan ytelsen på koderen endres når man benytter en adaptiv entropikoder, samt å se på hvordan dette påvirker kjøretiden til koderen. Den aktuelle 3D-videokoderen vil også sammenlignes med den siste ITU-T/ ISO/IEC standarden H.264/AVC.

Oppgaven gitt: 16. januar 2006
Hovedveileder: Tor Audun Ramstad, IET

Sammendrag

Masteroppgaven har hatt som formål å videreutvikle en tidligere utviklet 3D delbåndskoder for video. Videreutviklingen har medført at den foreslåtte koderen kan benytte seg av høyere ordens statistikk, samt betingete sannsynligheter mellom ulike symboler. Dette er gjort ved at en mer egnet kodingsmetode i lavpassdelbåndet er valgt, samt at en kontekstbasert, adaptiv, binæraritmetisk koder (CABAC) er implementert.

Den foreslåtte koderen benytter diskret cosinus transformasjon (*DCT*) for temporal dekomposisjon, samt en romlig dekomposisjon bestående av en 64-delbånds ikke-unitær filterbank med oktavbånd dekomposisjon i inntil tre nivåer i lavpassdelbåndet. Det dekomponerte signalet blir kvantisert med en skalar kvantiserer med dødsone, hvor dødsonen angir størrelsen på nullområdet. Grunnet effektkorrelasjonen mellom bildepunkter i de ulike nivåene i lavpassdelbåndet, kodes denne informasjonen med en modifisert utgave av "*zerotree*" koding. De resterende delbåndene kodes vha. zig-zag skanning og CABAC. Fordelen med denne delingen er at den foreslåtte koderen får utnyttet effektkorrelasjonen mellom nabobildepunkter innad i delbåndene (intrabånd effektkorrelasjon), effektkorrelasjonen mellom bildepunkter i de ulike delbåndene (interbånd effektkorrelasjon), samt det faktum at de fleste kilder har et fallende effektspekter. Ved å velge bort bevegelseskompensasjon, senkes kompleksiteten i den foreslåtte koderen, samtidig som den romlige dekomposisjonen kan utføres mer effektivt i form av en filterbank. "*Zerotree*" kodingen og implementasjonen av CABAC gjør det mulig for koderen å tilpasse seg ulike kilder, samt hver kildes temporalt og romlig varierende lokale statistikk.

Optimaliseringen av koderen viser at dens ytelse er sterkt avhengig av innholdet til orginalsekvensen. Den manglende bevegelseskompensasjonen medfører at koderen oppnår høyere ytelse for rolige sekvenser enn for sekvenser med mye objekt- og kamerabevegelser. Simuleringene viser også at koderens ytelse for rolige sekvenser er sterkt avhengig av antallet rammer den temporale dekomposisjonen utføres over. Sekvenser med mye bevegelser vil derimot oppnå lavere ytelse dersom antallet rammer i den temporale dekomposisjonen øker over et visst antall.

Simuleringene viser at innføringen av CABAC medfører en bitratebesparelse på 10-20% sammenlignet med en referansekoder hvor blokkbasert aritmetisk koding er benyttet. Innføringen av CABAC medfører derimot en liten økning i kjøretiden, noe som tyder på at kompleksiteten i CABAC er noe høyere enn kompleksiteten i en vanlig aritmetisk koder. Dette skyldes i hovedsak at flere kontekstmodeller benyttes, samt at modellene er adaptive.

Sammenligninger med den siste standardiserte koderen fra ITU-T/ISO/IEC *H.264/AVC*, viser at *PSNR* verdiene til den foreslåtte koderen ligger 1,5-2 dB lavere enn *H.264/AVC main profile* for sekvensen *news*. Den foreslåtte koderens ytelse for sekvensen *foreman* er 4-5 dB lavere enn *H.264/AVC*. Dette skyldes i hovedsak den foreslåtte koderens temporale dekomposisjon uten bevegelsesestimering.

Forord

Masteroppgaven ble utført våren 2006, som en del av studiet for graden sivilingeniør/Master i teknologi innen Elektronikk. Oppgaven ble utført ved institutt for elektronikk og telekommunikasjon ved Norges teknisk-naturvitenskapelige universitet (NTNU).

Jeg vil takke min veileder ved NTNU, professor Tor A. Ramstad, for den hjelpen og de innspillene jeg har fått gjennom prosessen. Jeg vil også takke min veileder ved TANDBERG, Dr. Ing. Arild Fuldseth, for veiledning og nyttige innspill underveis i prosessen.

Trondheim, 9. juni 2006

Espen Falkevik

Innhold

Figurer

Tabeller

Forkortelser og symboler

σ_x^2	varians inngangssignal
σ_ϵ^2	varians kvantiseringsstøy
σ	Kontekstmodellens tilstand
ϖ	Verdien til det mest sannsynlige symbolet (*MPS*)
γ	Kontekstmodellens indeks
Γ_S	Nedre grense for kontekstindeksen
χ_S	Kontekstindekstillegget
ρ	Indeks til tilnærmet bredde på intervallet (R)
p_n	Sannsynlighet til hendelse n
$\hat{l}_b$	Midlere kodeordslengde
$H(\mathbf{p})$	Kildens entropi mhp. sannsynlighetsdistribusjonen $\mathbf{p}$
$I(p_n)$	Egeninformasjonen til hendelse n

AR	Autoregressiv
AVC	Advanced Video Coding
bpp	Bit pr. piksel
CIF	Common Intermediate Format
DCT	Discret Cosine Transform
E	Antall etterfølgerbit
EGk	*k*te ordens Exp-Golomb
EZW	Embedded Zerotree Wavelet
FIR	Finite Impulse Response
FL	Fixed-Length
KLT	Karhunen-Loève Transform
L	Low (nedre grense på intervallet)
LPS	Least Probable Symbol
LP-LP	Lavpass-Lavpass
MB	Makroblokk
MPS	Most Probable Symbol
PCM	Pulse Code Modulation
PR	Perfekt Rekonstruksjon
PSNR	Peak Signal-to-Noise Ratio
R	Range (bredden på intervallet)
RLC	Run-Length Coding
TU	Truncated Unary
UEGk	Unary/*k*te ordens Exp-Golomb
VLC	Variable Length Coding

Kapittel 1

Innledning

1.1 Bakgrunn og motivasjon

Dagens samfunn beveger seg i en retning hvor bruken av video som kommunikasjonskanal blir mer og mer vanlig. Mange selskaper opererer i globale marked, og har kontorer ved flere lokasjoner. Dette kan medføre kommunikasjonsproblemer dersom det kun benyttes tradisjonelle kommunikasjonsmetoder. En løsning på disse problemene har for mange blitt bruken av videokonferanseutstyr, som erstatning eller supplement til telefon og e-post.

Et problem med de tradisjonelle videokonferansene er at brukerne må ha nødvendig utstyr tilgjengelig, noe som ikke alltid er like lett. Løsningen på dette er bruken av mobilt, trådløst utstyr (f.eks. mobiltelefoner). Innføringen av tredje generasjons mobilnett vil medføre at en rekke nye tjenester blir lansert. Eksempler på dette er videosamtaler og muligheten for å se TV på mobiltelefonen. En slik utvikling, hvor nye tjenester gjøres tilgjengelig, medfører nye krav til kompresjonsteknikkene som skal benyttes. Videokompresjon for mobiltelefoner vil i tillegg til å kreve høy kompresjon pga. båndbreddebegrensningen, også kreve at kompleksiteten ikke blir for høy, grunnet begrensede ressurser som prosesseringskraft og batterilevetid.

Hovedvekten av forskning og standardisering av videokodere de siste tiårene har vært gjort på hovedgruppen hybridkodere, hvor det benyttes differensiell koding i den temporale retningen. Hovedproblemet med denne gruppen kodere er den rekursive strukturen som benyttes i koderen/dekoderen, noe som medfører forplantning av eventuelle feil. Et annet problem er blokkeffekten som oppstår ved lavratekoding, grunnet den blokkbaserte dekomposisjonen (bevegelsesestimering og -kompensasjon, samt blokkbasert romlig dekomposisjon med f.eks. *DCT*). Fordelen med denne gruppen kodere er den høye effektiviteten, samt det faktum at teknikken har vært benyttet i mange år, noe som har medført at det er utviklet nødvendig hardware og software.

Den andre hovedgruppen videokodere kalles 3D-kodere, og skiller seg fra hybridkodere ved at det benyttes filterbanker eller transformkoding i samtlige tre dimensjoner (temporalt og romlig). En fordel med dette er at den rekursive strukturen unngås. Innen 3D-kodere skilles det gjerne mellom kodere med og uten bevegelseskompensasjon. Kodere uten bevegelseskompensasjon vil effektivt kode rolige sekvenser, og har sin fordel ved at de kan benytte mer avanserte romlige dekomposisjonsmetoder enn kodere med bevegelseskompensasjonen. Sekvenser med både objekt- og kamerabevegelse vil derimot kunne komprimeres bedre vha. bevegelseskompensasjon, men dette skjer på bekostningen av økt kompleksitet i koderen. Et kjent problem med 3D-kodere er forsinkelsen gjennom systemet som skjer når flere rammer

må bufres opp før den temporale dekomposisjonen kan utføres. Dette kan, avhengig av størrelsen på forsinkelsen, ikke være ønskelig i enkelte system (f.eks. videokonferanser). Ulike eksperiment viser at 3D-kodere kan oppnå like høy ytelse som hybridkodere. J.R. Ohm [1] viste tidlig hvordan transformkoding med bevegelseskompensasjon kunne oppnå bedre ytelse enn hybridkodere. Lin Luo et. al har senere vist hvordan 3D-kodere med bevegelseskompensasjon og *lifting* filter overgår ytelsene til MPEG-4 [2].

Et generelt problem med tidligere standardiserte videokodere er valget av entropikoder. Entropikodingen har som mål å tilpasse ett sett av kodingssymbol til kildesymbolene, slik at det benyttes færrest mulig bit. Sammensetningen av løplengde koding (*RLC*) og variabellengde koding (*VLC*) har vært basert på faste tabeller. Dette har medført at entropikoderen ikke har mulighet til å tilpasse seg ulike signalers statistikk, både i form av forskjellene mellom ulike kilder, samt hver kildes temporalt og romlig varierende lokale statistikk. En lenge kjent løsning på dette problemet har vært å benytte seg av adaptiv aritmetisk koding [3], hvor det er et klart skille mellom modelleringsfasen og kodingsfasen. Likevel har dette kun blitt benyttet i mindre grad innenfor hybridkodere. Den siste standardiserte koderen fra ITU-T/ISO/IEC, *H.264/AVC*, har derimot standardisert en kontekstbasert, adaptiv, binæraritmetisk koder (CABAC) som viser seg å gi bitratebesparelser på 9-14% sammenlignet med en kontekstbasert, adaptiv, variabellengde koder, også er spesifisert i *H.264/AVC* [4].

Målsetningen til denne oppgaven var å videreutvikle en 3D-koder uten bevegelsesestimering, ved å innføre en mer egnet metode for kodingen av informasjonen i LP-LP delbåndet, samt ved å innføre CABAC. Oppgavens hovedformål var å tilpasse koderen slik at den oppnådde høyest mulig ytelse, samtidig som den perseptuelle kvaliteten ble opprettholdt. Oppgaven skulle også sammenligne ytelsen til CABAC med en tidligere benyttet blokkbasert aritmetisk koder, samt se hvordan kompleksiteten endret seg ved innføringen av CABAC. Ytelsen til koderen ble målt vha. Peak Signal-to-Noise Ration (*PSNR*) for sekvensene *news* og *foreman*, og sammenlignet med *H.264/AVC main profile*.

1.2 Oppbygging av rapporten

Rapporten er bygd opp på følgende måte:

- Kapittel 2 inneholder bakgrunnsteori angående videokompresjon
- Kapittel 3 inneholder en nærmere beskrivelse av CABAC
- Kapittel 4 inneholder en inngående beskrivelse av oppbyggingen til den foreslåtte koderen.
- Kapittel 5 inneholder numeriske og visuelle resultater for den foreslåtte koderen, samt sammenligninger med *H.264/AVC*.
- Kapittel 6 inneholder en vurdering av de oppnådde resultatene
- Kapittel 7 inneholder konklusjonen
- Appendiks A inneholder en beskrivelse av hvordan programmet kjøres.

Kapittel 2

Videokompresjon

Kapittelet har som formål å forklare grunnlaget for videokompresjon, samt utdype og forklare teorien bak de metodene en videokoder benytter.

Hovedårsaken til at video kan komprimeres uten at det visuelle inntrykket blir betydelig degradert ligger både i oppbyggingen av videosignaler, og i øyets og menneskets begrensninger. De tre faktorene som ligger til grunn for all kompresjon er

- redundans
- irrelevans
- toleranse

Redundans vil si at det er statistisk avhengighet mellom nærliggende bildepunkter i romlig og temporalt plan. Ved å bruke reversible transformer, som er tilpasset statistikken, kan informasjonen til en diskret kilde komprimeres tapsfritt. For å oppnå enda høyere kodegevinst benyttes denne teknikken i kombinasjon med andre ikke-tapsfrie teknikker.

Irrelevans hentyder på øyets og menneskets begrensninger, samt måten vi oppfatter visuelle inntrykk på. Irrelevans kan tolkes som mengden distorsjon som kan innføres uten at det visuelle inntrykket endres.

Toleranse henviser til mengden av synlig distorsjon som mennesker er villige til å tolerere. Dette kan variere mellom forskjellige mennesker, samt være avhengig av omgivelsene og miljøet hvor kilden observerers.

Hoveddelene i en videokoder er dekomposisjon av signalet, kvantisering og koding av kvantiserte verdier.

2.1 Dekomposisjon

En av grunnene til at videosekvenser kan komprimeres kraftig uten at distorsjonen blir for merkbar, er korrelasjonen mellom påfølgende rammer, samt korrelasjonen som er innad i hver ramme. Den temporale korrelasjonen kommer av at typiske sekvenser inneholder mange bilder som er svært like, hvor kun enkelte objekter beveger seg, mens korrelasjonen innad i hver ramme kommer av at naturlige bilder gjerne inneholder båndbegrensete områder. Dekomposisjonens oppgave er å fjerne mest mulig av redundansen i signalet.

Kodingsgevinsten for en delbåndskoder er definert som forholdet mellom støybidraget ved direkte bruk av skalar kvantisering (*PCM*) på inngangssignalet og støybidraget fra en delbåndskoder ved kvantisering basert på optimal bitallokering [5].

Effektiviteten til ulike dekomposisjonsmetoder er avhengig av en rekke kriterier, og kan bestemmes utifra bl. a.:

- Hvor dekorrelert signalet blir, samt hvor mye av energien til det orginale bildet som er komprimert til et minimum av dekomponerte verdier for en gitt signalkarakteristikk.
- Hvilke typer visuelle artefakter som innføres ved lavratekoding (f.eks. blokk- og ringing-effekt)
- Om den inverse dekomposisjonen innebærer perfekt rekonstruksjon (*PR*) eller ikke.

Dekomposisjon deles ofte opp i transformdekomposisjon og filterbankdekomposisjon, selv om transformkoding kan sees på som en undergruppe av filterbanker. Hovedforskjellen mellom de to metodene kan forklares ved at transformkoding som regel innebærer at kilden deles opp i separate blokker, og deretter utføres transformkodingen på hver blokk, mens filterbankdekomposisjon i prinsippet kan utføres på hele kilden. I tillegg skilles også differensiell (prediktiv) koding gjerne ut som en egen gruppe innenfor dekomposisjon.

2.1.1 Optimal kvadratisk transform

Den optimale blokktransformen for full dekorrelasjon innad i en blokk er *Karhunen-Loève* transformen (*KLT*). Transformkoeffisientene beregnes ved å benytte egenvektorene til autokorrelasjonsmatrisen. Problemet med denne transformen er at den er signalavhengig. Dette medfører at den må beregnes på nytt når signalet endrer statistikk, noe som er et problem for video og bilder, hvor den lokale statistikken varierer innenfor hver ramme, samt over tid.

En mer brukt transform er *diskret cosinus transform (DCT)*. Denne transformen er gitt av

$$C(0) = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} s(k)$$

$$C(u) = \sqrt{\frac{2}{M}} \sum_{k=0}^{M-1} s(k) cos[\frac{(2k+1)u\pi}{2M}] \quad u = 1, 2, ..., M-1 \tag{2.1}$$

med den inverse transformen

$$s(k) = \frac{1}{\sqrt{M}} C(0) + \sqrt{\frac{2}{M}} \sum_{u=1}^{M-1} C(u) cos[\frac{(2k+1)u\pi}{2M}] \tag{2.2}$$

Dersom signalet kan antas å være stasjonært, og at signalet kan modelleres som en *AR(1)* prosess med en prediksjonskoeffisient ρ nær 1, vil *DCT* oppnå en energipakkingseffektivitet[1] svært nær *KLT*, samtidig som dekorreleringsegenskapene kun er litt redusert i forhold til *KLT* [5][6]. Fordelen med *DCT* er dens lave kompleksitet, samt at den er separabel, dvs. at den todimensjonale dekomposisjonen kan utføres som to endimensjonale operasjoner. En annen fordel er at den er signaluavhengig.

Et problem med transformdekomposisjon er at den ikke utnytter interblokkredundansen ved at selve transformasjonen utføres på separate blokker.

[1] definert av Clarke[6, kapittel 7] som andelen av den totale energien som ligger i de N første koeffisientene etter transformen

2.1.2 Optimale filterbanker

En filterbank har flere frihetsparametere enn en kvadratisk transform, og den bør derfor kunne kunne oppnå høyere kodingsgevinst. Oppbyggingen av de ulike filterene i filterbanken begrenses kun av filterbankens ønskede egenskaper.

Fullstendig dekorrelasjon mellom delbåndene (interbånd dekorrelasjon)

Fullstendig interbånd dekorrelasjon er et viktig kriterie for maksimal kodingsgevinst, grunnet at den samme informasjonen ikke bør representeres flere ganger (korrelasjon mellom flere delbånd). Dette kan oppnås ved å benytte ideelle firkantfilter med perfekt separasjon mellom filterene i frekvensplanet:

$$|H_n(e^{j\omega})| = \begin{cases} \neq 0 & \text{for } \omega \in \left[\frac{\pi n}{N}, \frac{\pi(n+1)}{N}\right] \\ 0 & \text{ellers} \end{cases}, n = 0, 1, \ldots, N-1, \tag{2.3}$$

hvor N er antallet delbånd.

Perfekt rekonstruksjon

Et viktig poeng med dekomposisjon er at operasjonen er reversibel dersom kvantisering (og invers kvantisering) ikke benyttes. Dersom inngangssignalet til analysefilterbanken og utgangssignalet av syntesefilterbanken er identisk, oppfyller dekomposisjonen kriteriet for perfekt rekonstruksjon (*PR*). Gitt betingelsen for full interbånd dekorrelasjon i ligning 2.3, kan perfekt rekonstruksjon oppnås ved følgende betingelse for sammenhengen mellom analysefilteret $H_n(e^{j\omega})$ og syntesefilteret $G_n(e^{j\omega})$:

$$|H_n(e^{j\omega})| = \begin{cases} \frac{N}{|G_n(e^{j\omega}|} & \text{for } \omega \in \left[\frac{\pi n}{N}, \frac{\pi(n+1)}{N}\right] \\ 0 & \text{ellers} \end{cases}, n = 0, 1, \ldots, N-1, \tag{2.4}$$

Unitær filterbank

Unitære filterbanker er en type filterbanker med *PR* egenskaper. Denne gruppen filterbanker har en oppbygging hvor analyse- og syntesefilterene har samme amplitude på frekvensresponsen, noe som gir:

$$|H_n(e^{j\omega})|^2 = \begin{cases} N & \text{for } \omega \in \left[\frac{\pi n}{N}, \frac{\pi(n+1)}{N}\right] \\ 0 & \text{ellers} \end{cases}, n = 0, 1, \ldots, N-1, \tag{2.5}$$

Ikke-unitære filterbanker

Bedring av kodingsgevinsten kan gjøres ved å benytte seg av ikke-unitære filterbanker, som medfører at det benyttes ulike frekvensamplituder i analyse- og syntesefilterene. Hovedgrunnen til dette er at korrelasjonen innad i hvert delbånd bør fjernes (intrabånd korrelasjon). Dette kan gjøres ved å innføre en halvhvitingsmekanisme innad i delbåndene før kvantiseringen. Denne prosessen må reverseres i syntesedelen, noe som impliserer at amplitudene må

være ulike. For maksimal kodingsgevinst kan det vises at analysefilteret må være på formen:

$$|H_n(e^{j\omega})| = \begin{cases} c_2 \left[\frac{S_{XX}(F)}{\sigma_x^2}\right]^{-1/4} & \text{for } \omega \in \left[\frac{\pi n}{N}, \frac{\pi(n+1)}{N}\right] \\ 0 & \text{ellers} \end{cases}, n = 0, 1, \ldots, N-1, \qquad (2.6)$$

hvor c_2 er en tilfeldig konstant, σ_x^2 er variansen til inngangssignalet og $S_{XX}(F)$ er kildens effektspekter [5].

For full dekorrelasjon mellom delbåndene, perfekt rekonstruksjon og full intrabånd dekorrelasjon kreves det at analysefilterene har ideell frekvensseparasjon mellom delbåndene, samt halvhviteningfiltrering innad i hvert delbånd.

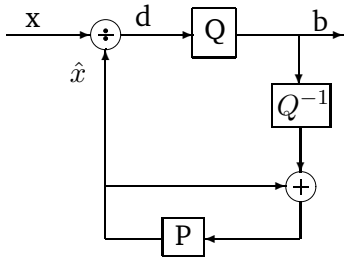
2.1.3 Kodingsgevinst

Kodingsgevinsten ved ulike dekomposisjonsmetoder avhenger av valg av metode, lengden på filterene samt inngangsfordelingen til signalet. Når det gjelder valg av dekomposisjonsmetode vil ikke-unitære filterbanker ha bedre ytelse enn unitære filterbanker, som igjen har bedre ytelse enn transformdekomposisjon. Kodingsgevinsten vil i tillegg øke når antallet delbånd i dekomposisjonen øker.

Dersom et bilde modelleres som en *AR(1)* prosess, viser det seg at kodingsgevinsten øker etterhvert som filterlengden øker. Et problem med å øke filterlengden for mye er at den varierende lokale statistikken i naturlige bilder da ikke kan ytnyttes.

Forsøk viser også at bruk av filterbanker uten *PR*, kan øke kodingsgevinsten ved lavratekoding [5].

2.2 Differensiell koding

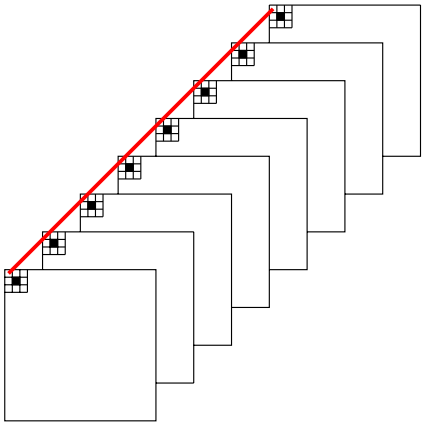


Figur 2.1: *closed-loop* differensiell koder hvor kvantisereren inngår i en tilbakekobling. Prediktoren **P** benyttes for å prediktere signalet basert på tidligere data.

Closed-loop differensiell koding skiller seg fra den vanlige kodingsoppbyggingen med dekomposisjon, kvantisering og entropikoding ved at kvantiseringen skjer inn i en tilbakekobling som vist i figur 2.1. Prediksjonen P utføres hele tiden på den informasjonen som vil være tilgjengelig i dekoderen. Optimal kodingsgevinst i *closed-loop* differensiell koding oppnås når prediktoren gjør inngangssignalet til kvantisereren hvitt.

2.3 3D-videokoding

3D-kodere skiller seg fra hybridkodere ved at det benyttes transformasjoner eller filterbanker for dekomponering i alle tre retninger, i motsetning til hybridkodere som benytter differensiell koding i temporal retning.



Figur 2.2: Transformasjon i tidsretningen. Transformasjonen utføres på bildepunkter med samme koordinater

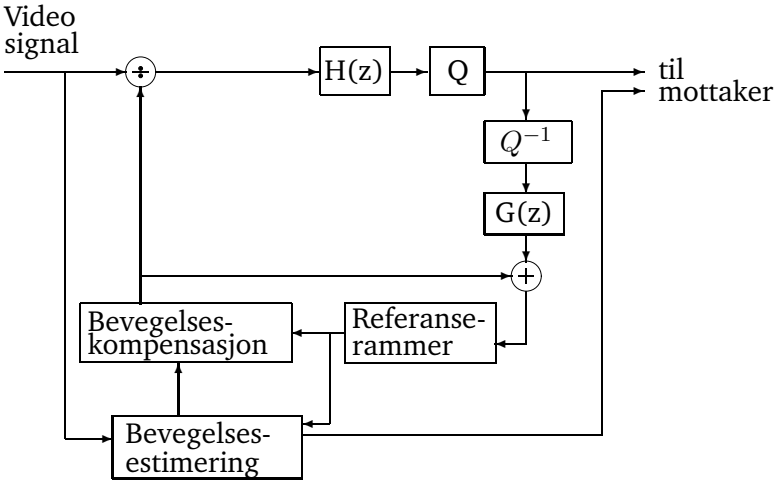
Dekomposisjon i temporal retning vha. transformasjoner baserer seg på antagelsen om at det finnes flere påfølgende rammer som har høy grad av korrelasjon. En måte å gjennomføre dekomposisjonen på er å benytte en egnet transform på bildepunktene i påfølgende rammer som har samme romlige koordinater (innad i hver ramme). Dette er vist i figur 2.2. Den romlige dekomposisjonen innenfor 3D-videokodere kan utføres ved å benytte transformkoding eller filterbanker.

Hovedproblemet med den temporale dekomposisjonsmetoden vist i figur 2.2 er at ytelsen synker når sekvensen som skal kodes inneholder mye bevegelse. En løsning på dette problemet er å benytte bevegelseskompensasjon. Dette kan gjøres ved å estimere bevegelseskurver, og deretter benytte denne informasjonen i en blokkbasert temporal *lifting* struktur [1][2][7]. En slik oppbygging har vist seg å kunne gi like gode resultater som bevegelsesestimeringen og -kompensasjonen som utføres i hybride videokodere.

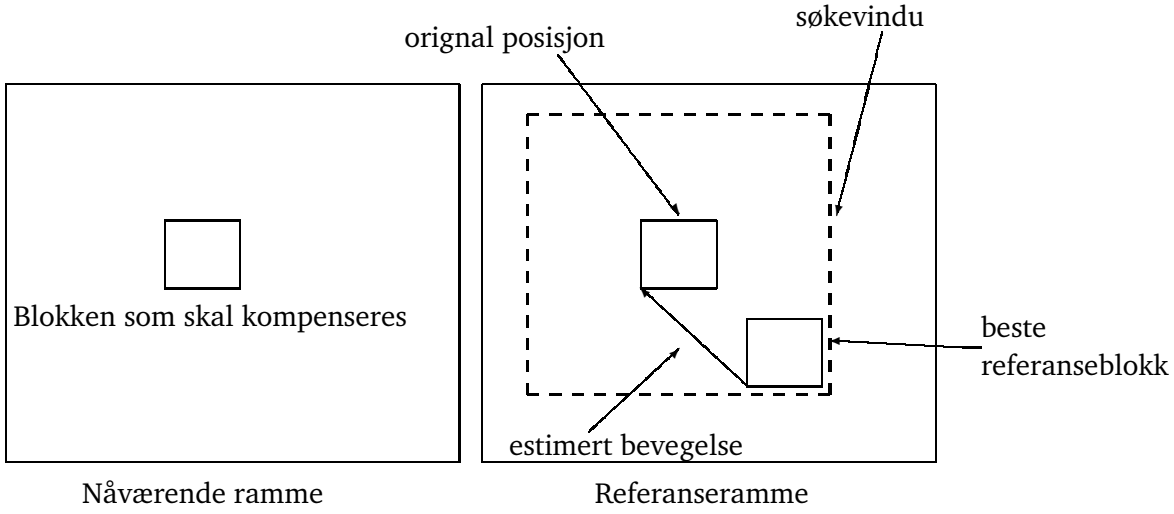
2.4 Hybrid videokoding

Hybridkodere skiller seg fra 3D-kodere ved at det benyttes differensiell dekomposisjon i temporal retning. En skjematisk hybridkoder er vist i figur 2.3.

Hybridkodere benytter seg av blokkvis bevegelsesestimering og -kompensasjon for å prediktere signalet (se figur 2.4). Metoden baserer seg på at bildet deles opp i adskilte blokker, og deretter leter systemet etter den blokken i et referansebilde som er mest lik den aktuelle blokken, der likheten kan måles på flere måter (f.eks. minste kvadratiske feil). Når referanseblokken er bestemt, estimeres en bevegelsesvektor som beskriver bevegelsen til blokken. Den videre kodingen skjer deretter på differansesignalet mellom de to blokkene.



Figur 2.3: Skjematisk oppbygging av hybrid videokoder. Temporal dekomposisjon skjer vha. differensiell koding, mens romlig dekomposisjon skjer vha. blokkvis transformasjon.



Figur 2.4: Blokkvis bevegelsesestimering, figur hentet fra [5]

Bevegelsesestimering og -kompensasjon vil i tillegg til å fjerne redundansen i områder uten bevegelse, også kunne fjerne redundansen mellom bevegelige objekter i påfølgende rammer. Dekomposisjonen kan dermed fjerne redundans som skyldes forflytning av objekter eller generell kamerabevegelser som zooming og panorering.

Den mest vanlige romlige dekomposisjonsmetoden i hybridkodere er blokkvis transformasjoner i form av *DCT*. Hovedårsaken til dette er at den blokkvise bevegelsesestimeringen og -kompensasjonen kan resultere i kantoverganger (høye frekvenser) mellom de ulike blokkene.

En fremgangsmåte som er forsket på i senere tid, er å utføre den romlige dekomposisjonen før bevegelsesestimeringen og -kompensasjonen utføres for hvert delbånd. En gevinst med denne fremgangsmåten er at dersom man anser bevegelsene i kilden som naturlig bevegelse, vil det være sammenhenger mellom bevegelsesestimeringen i alle delbånd, noe som kan utnyttes [5]. En annen fordel med denne metoden er at den kan benytte seg av filterbanker for romlig dekomposisjon, noe som vil øke effektiviteten i forhold til transformdekomposisjon.

2.5 Statistiske avhengigheter etter romlig dekorrelasjon

Ulike forsøk med dekomposisjon basert på *wavelets* (med oktavbånd dekomposisjon av lavpassdelbåndet), viser at en den lineære korrelasjonen (ligning 2.7) mellom forelderbildepunkter i et delbåndsnivå, og barnbildepunktene i et delbånd på et lavere nivå (se figur 2.8) er tilnærmet lik null.

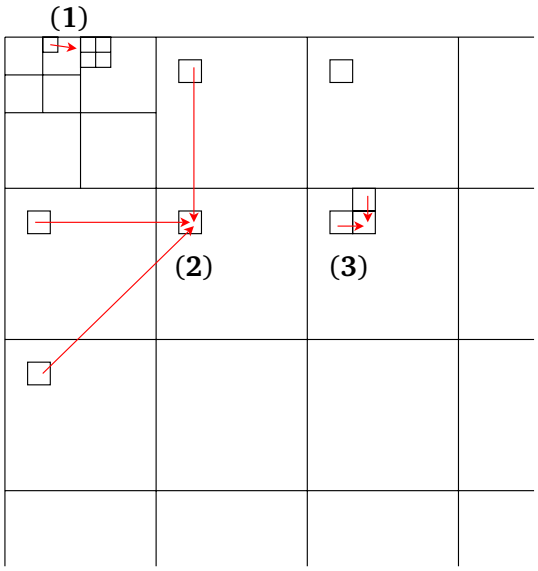
$$\mathrm{E}\left[X,Y\right] = \frac{\mathrm{E}\left[XY\right]}{\sqrt{\mathrm{E}\left[X^2\right]\mathrm{E}\left[Y^2\right]}} \tag{2.7}$$

Derimot viser forsøk gjort på ulike typer bilder at korrelasjonskoeffisienten til effekten (ligning 2.8) av forelder og barn ligger på mellom 0,2 og 0,6, hvor hoveddelen ligger rundt 0,35 [8].

$$\mathrm{E}\left[\left|X\right|^2,\left|Y\right|^2\right] = \frac{\mathrm{E}\left[\left|X\right|^2\left|Y\right|^2\right]}{\sqrt{\mathrm{E}\left[\left|X\right|^4\right]\mathrm{E}\left[\left|Y\right|^4\right]}} \tag{2.8}$$

En forklaring på dette fenomenet er at naturlige bilder kan betraktes som en sammensetning av to hoveddeler, trender og anormaliteter. Trendene er områder med høy romlig korrelasjon (typisk områder med liten aktivitet og lave frekvenser), mens anormalitetene er områder bestående av kanter og grenseområder (områder med høye frekvenser og relativt lite energibidrag i bildet, men med høy perseptuell signifikans) [8]. Etter en romlig dekomposisjon, vil anormalitetene gjenspeiles som aktivitet i flere delbånd (grunnet at de består av et bredt spekter av frekvenskomponenter), og denne aktiviteten vil skje på samme lokasjon innad i hvert delbånd. Dette gjenspeiles i effektkorrelasjon mellom de ulike delbåndene.

Den samme forklaringen kan benyttes for å forklare avhengigheten mellom effekten til nabopiksler innad i hvert delbånd. Dersom dekomposisjonen er effektiv, vil den lineære korrelasjonen være liten, men også her kommer aktivitetsbetraktningen av bilder inn når effektkorrelasjonen vurderes. Dersom et piksel har en nabo med en stor effekt, vil det være mer sannsynlig at bildepunktet har stor effekt, enn om naboen har liten effekt. Dette kan forstås utifra at naturlige bilder har objekter med kanter som går over større eller mindre områder (som f.eks en husvegg eller en fjelltopp).



Figur 2.5: Ulike typer effektkorrelasjon. (**1**) viser hvordan effektkorrelasjon opptrer mellom forelderbildepunkt og barnbildepunkt i en trestruktur, (**2**) viser effektkorrelasjon mellom bildepunkter på samme lokasjon i ulike delbånd og (**3**) viser effektkorrelasjon mellom ulike bildepunkter innad i hvert delbånd.

2.6 Kvantisering

Kvantisering benyttes for å tilnærme et signal til et endelig sett av verdier, som kan kodes. Avhengig av inngangsfordelingen til signalet, ønsket kompleksitet og forsinkelse kan det benyttes ulike typer kvantiserere.

2.6.1 Skalar uniform kvantiserer

Den enkleste formen for kvantisering er en skalar uniform kvantiserer hvor alle representasjonsnivåene er plassert med like mellomrom innenfor et gitt intervall $[-A\ ,\ A]$. Det eksisterer to hovedtyper skalar uniforme kvantiserere, *midtread* kvantisereren som har muligheten til å kvantisere verdier til null, og *midrise* som ikke har et nullområde (se figur 2.6).

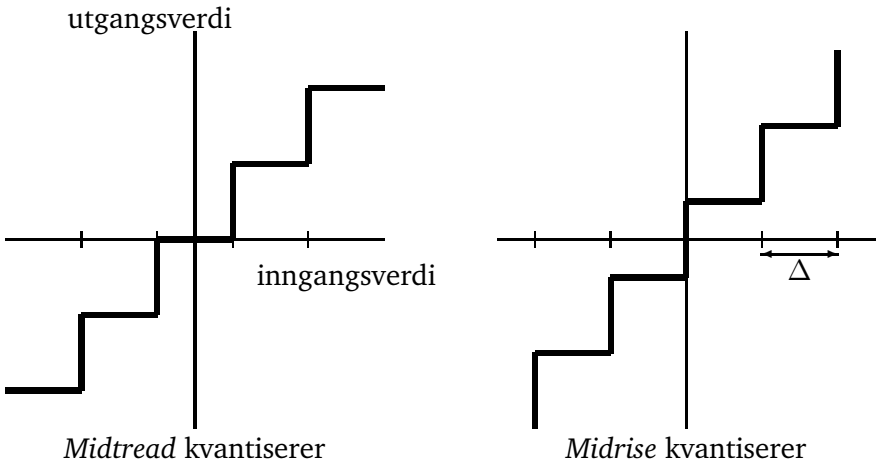
Det mest vanlige målet for støybidraget til en kvantiserer er variansen til kvantiseringsfeilen:

$$\sigma_\epsilon^2 = E\left[\epsilon^2\right], \tag{2.9}$$

hvor $\epsilon = x - r_n$ tilsvarer kvantiseringsfeilen når inngangssignalet x blir kvantisert til representasjonsverdien r_n. Dersom x er uniformt fordelt innenfor $[-A\ ,\ A]$ blir feilvariansen innenfor hvert kvantiseringstrinn lik, og sannsynligheten for å havne i hvert intervall lik, noe som resulterer i følgende kvantiseringsfeil[9]

$$\sigma_\epsilon^2 \quad = \quad \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} (\epsilon)^2\, \mathrm{d}\epsilon = \frac{\Delta^2}{12} \tag{2.10}$$

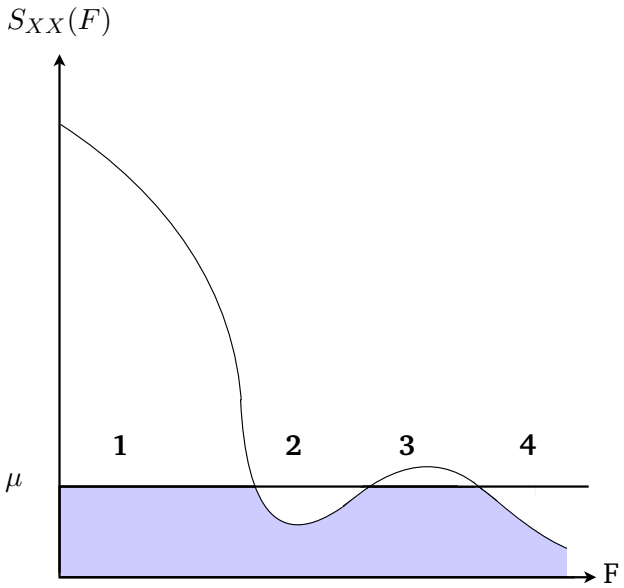
hvor Δ er trinnstørrelsen på kvantisereren. Resultatet i ligning 2.10 kan generaliseres til å gjelde for samtlige sannsynlighetsfunksjoner dersom $\Delta \ll \sigma_x$.



Figur 2.6: Skalar uniform kvantiserer

Et alternativ til skalar kvantisering er vektorkvantisering, hvor en vektor av inngangssignalet blir sammenlignet med et sett av kodevektorer i en kodebok. Indeksen til kodevektoren som er mest lik inngangsvektoren, overføres til mottakeren, som gjør et oppslag i en tilsvarende kodebok og henter ut representasjonsvektoren for indeksen. Innføringen av en slik kvantiserer vil øke kompleksiteten og forsinkelsen av kodingen, men det kan vises at for en gitt kompresjonsgrad, vil en vektorkvantiserer med optimalt design alltid være den kildekoderen som gir best rate-distorsjonsytelse[9].

2.7 Effektspekter og ressursallokering



Figur 2.7: Effektspekteret til en kilde, hvor det er innført støy i form av kvantisering (μ).

Effektspekter til naturlige bilder vil typisk være fallende, dvs. at det er flere lavfrekvente

komponenter enn høyfrekvente (bilder kan typisk modelleres som en *AR(1)* prosess med korrelasjonskoeffisient $\rho \approx 0,9 - 0,95$ [5]). For en gitt kilde med effektspekteret $S_{xx}(F)$ som er påvirket av en støykilde (f.eks. kvantiseringsstøy) μ, vil den optimale raten være gitt ved:

$$R(\mu) \;=\; \int_0^W \max\left[0\;,\; \log_2(\frac{S_{XX}(F)}{\mu})\right] dF \text{ bits pr sekund} \tag{2.11}$$

som resulterer i følgende distorsjon:

$$\sigma_D^2(\mu) \;=\; \int_0^W \min\left[\mu\;,\; S_{XX}(F)\right] dF \tag{2.12}$$

Dette vil si at den optimale raten for kilden er null i de områdene hvor kvantiseringsstøyen er større enn effekten til kilden (område **2** og **4** i figur 2.7) og lik $\log_2(\frac{S_{XX}(F)}{\mu})$ i de områdene hvor effekten til kilden er større enn kvantiseringsstøyen (område **1** og område **3**).

2.8 *"Zerotree"* koding

"Embedded Zerotree Wavelet" (*EZW*) koding, først introdusert av J.M. Shapiro [8], er en meget effektiv og lite kompleks bildekodingsteknikk. Kodingsteknikken baserer seg på en *wavelet* transformasjon hvor det innføres en dyadisk splitt (oktavbånd dekomposisjon) i LP-LP delbåndet. Kodingen er effektiv pga. at den utnytter det faktum at de fleste effektspekter er synkende funksjoner, samt at det eksisterer en effektkorrelasjon mellom forelderpiksel og barnpiksel (som beskrevet i del 2.5). Dette benyttes i *EZW* ved at dersom en forelderpiksel har amplitude lavere enn en gitt grenseverdi, vil det også være stor sannsynlighet for at samtlige barnpiksler til denne pikselen har amplituder mindre enn grenseverdien. Dersom dette stemmer kalles dette for et *"zerotree"*.

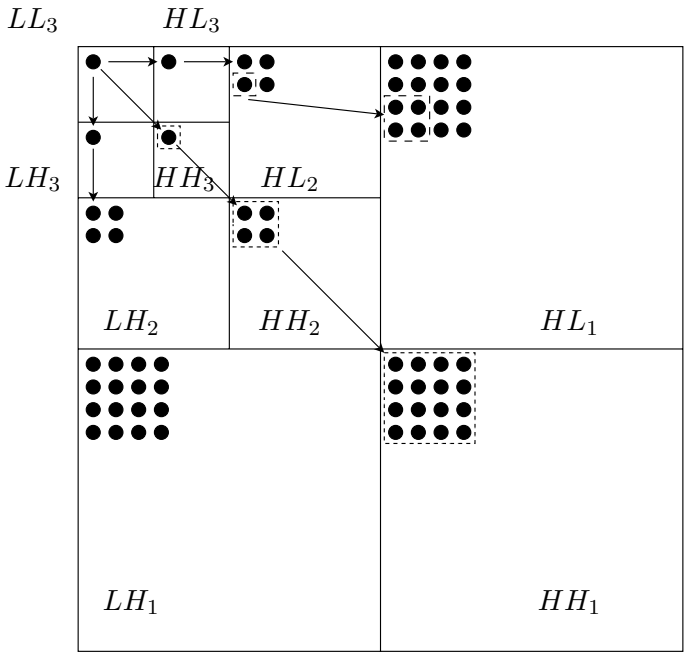
EZW kodingen (beskrevet i [8]) skjer vha. en iterativ algoritme, bestående av to operasjoner:

- Koding av signifikanskart (kalt dominant kodingsløp), hvor det for en gitt grenseverdi T, bestemmes om pikslene i de ulike delbåndene har amplituder som er større eller mindre enn grenseverdien. Dersom en koeffisient er signifikant, legges amplituden til en underordnet liste, og koeffisienten settes til null i den neste iterative kjøringen.
- Oppdatering av signifikante koeffisienter (kalt underordnet kodingsløp), som innebærer at usikkerhetsintervallet til de signifikante koeffisientenes amplitude halvveres.

2.8.1 Dominant kodingsløp

Det dominante kodingsløpet innebærer at signifikansen til koeffisientene kodes til en av følgende verdier:

- Dersom både forelderpikselen og alle barnpikslene har amplituder lavere enn grenseverdien T, kodes dette som et *"zerotree"*.
- Dersom forelderpikselen har amplitude større en T, kodes pikselen som signifikant, og amplituden legges i en underordnet liste, før koeffisienten settes lik null.
- Dersom forelderpikselen har amplitude som er lavere enn T, men en barnpiksel av forelderen har amplitude høyere enn T, kodes pikselen som isolert-nullkoeffisient.



Figur 2.8: *"Zerotree"* koding på bilde dekomponert med oktavbånd filterbank. Figuren viser avhengigheten mellom forelderpiksel og barnpiksel, hvor hvert forelderpiksel på ett nivå har fire barnpiksel i det neste nivået i trestrukturen.

2.8.2 Underordnet kodingsløp

Det underordnete kodingsløpet har som oppgave å oppdatere nøyaktigheten til de signifikante koeffisientene. Dette gjøres ved å halvere størrelsen på grenseverdien, og bestemme om amplituden ligger i øvre eller nedre del av usikkerhetsintervallet (dersom en koeffisient kodes som signifikant ved en grenseverdi T_0 vil usikkerhetsintervallet i første underordnete kodingsløp være $\langle\ T_0\ ,\ 2 * T_0\ \rangle$). Etter hvert underordnet kodingsløp utføres et nytt dominant kodingsløp med den nye grenseverdien.

2.9 Entropikoding

Oppgaven til en entropikoder er å kode et sett av hendelser med så få bit som mulig. Hovedidèen består i å gi korte kodesymboler til hendelser med høy sannsynlighet, og lengre kodeord til mindre sannsynlige hendelser.

Gitt en diskret, minneløs kilde med symbolalfabetet $\{x_0, x_1, \ldots, x_{N-1}\}$ hvor N er antallet symboler og sannsynligheten for komponent n, $p_n\ =\ P(X = x_n)$. Egeninformasjonen $I(p_n)$ til hvert symbol er da definert ved:

$$I(p_n) = -\log_2(p_n) \quad \text{informasjonsbit pr kildesymbol,} \tag{2.13}$$

og kan tolkes som det minste antallet bit som må benyttes for å spesifisere at hendelsen x_n har blitt observert.

Egeninformasjonen beskriver kun informasjonsinnholdet i en enkelt hendelse. En mer nyttig verdi vil for datakompresjon være gjennomsnittlig informasjonsinnhold for en kilde. Dette

gjennomsnittet er gitt ved forventningsverdien av egeninformasjonen mhp. sannsynlighetsdistribusjonen **p** og kalles kildens entropi[9].

$$H(\mathbf{p}) = -\sum_{n=0}^{N-1} p_n * I(p_n) = -\sum_{n=0}^{N-1} p_n \log_2(p_n) \quad \text{informasjonsbit pr. kildesymbol.} \quad (2.14)$$

For å kunne kode enhver sekvens av kodesymboler fra denne kilden entydig (gitt at den er minneløs), med et sett av kildesymboler gjelder det alltid at

$$\bar{l_b} \geq H(\mathbf{p}) \quad (2.15)$$

hvor $\bar{l_b}$ er forventet antall kodesymbol pr. kildesymbol [9, Shannon's kildekodingsteorem].

Det finnes to hovedmetoder for entropikoding. Universelle kodere, som ikke antar kjennskap til kilden som skal kodes, og modellbaserte kodere som antar en kjent sannsynlighetsfordeling på kilden. Eksempel på den første gruppen er Lempel-Ziv algoritmen, mens den andre gruppen kan deles opp i to hovedgrupper, Huffmankoding og aritmetisk koding.

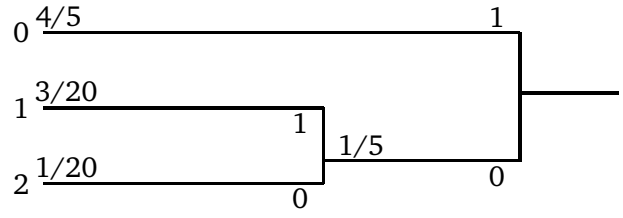
Huffmankoding har sin fordel i sin lave kompleksitet. Algoritmen baserer seg på iterative operasjoner hvor man slår sammen de to symbolene med lavest sannsynlighet, sorterer de nye symbolene etter deres sannsynlighet, og deretter tilegner de bit etter en fast regel. En ulempe med denne er at alle kodeord må ha heltalls lengde og være minimum ett bit. Dette kan løses ved å kode blokker av symbol, men dette vil medføre at alfabetstørrelsen øker betraktelig[10].

Gitt en kilde med tre diskrete symboler med sannsynlighetsfordeling som vist i tabell 2.1.

symbol	0	1	2
sannsynlighet	$\frac{4}{5}$	$\frac{3}{20}$	$\frac{1}{20}$

Tabell 2.1: Sannsynlighetsfordeling til eksempelkilde

Entropien til denne kilden (gitt av ligning 2.14) er lik $0,8842$ bit pr kildesymbol. Huffmankoden for den gitte kilden er vist i figur 2.9. Gjennomsnittlig kodeordslengde ($\bar{l_b}$) er lik $1,2$ bit pr. kildesymbol. Gitt symbolsekvensen: $0-1-0-2-0-0$, vil dette kodes av Huffmankoderen som: $1-01-1-00-1-1$.



symbol	0	1	2
kodeord	1	01	00

Figur 2.9: Huffmankoding

Teorien bak aritmetisk koding baserer seg på at det tilegnes et kodeord til hvert mulige sett av data som kan forekomme. Selve kodeordet består av halvåpne delintervall i det halvåpne

enhetsintervallet $[0\ ,\ 1\rangle$. Delintervallet kan også beskrives vha. dets nedre grense (ofte kalt *low*) og bredden på intervallet (kalt *range*). Det endelige delintervallet uttrykkes ved at det tilegnes nok bit til kodeordet for å skille intervallet fra alle andre mulige intervall [11]. Kortere kodeord vil da tilsvare større delintervall, og dermed sett av data med høyere sannsynlighet.

Selve kodingen foregår i to trinn for hver hendelse:

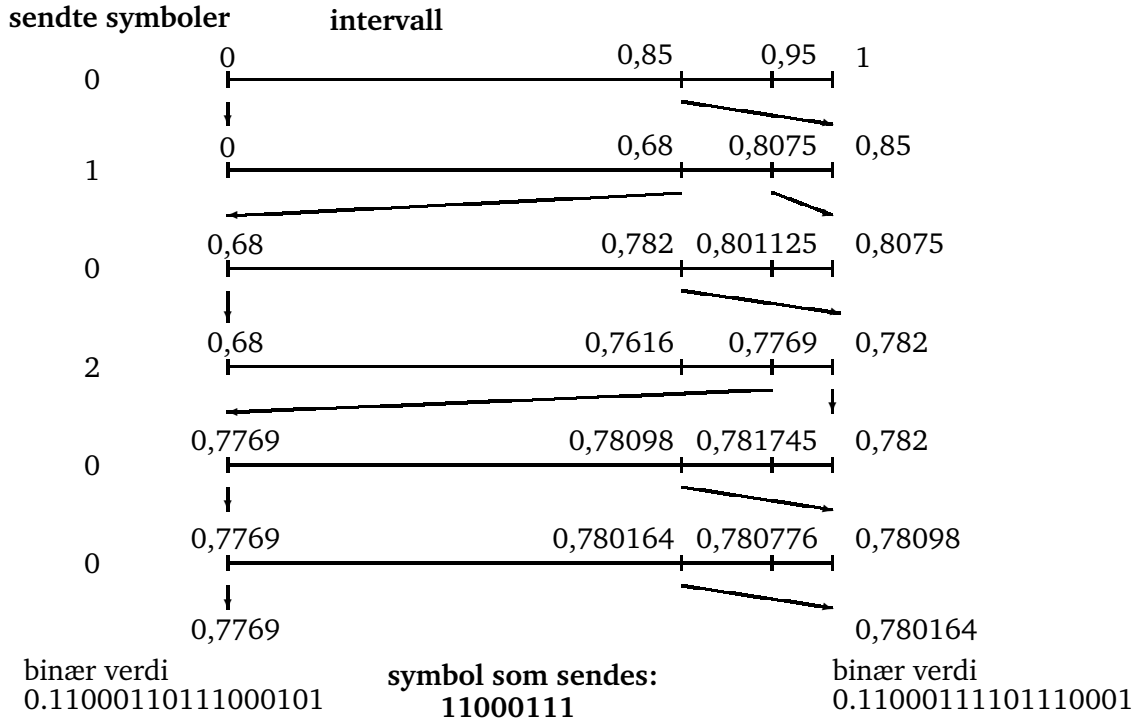
1. Oppdeling av det nåværende intervallet til delintervall, ett for hver hendelse. Størrelsen på delintervallet er proposjonal med den estimerte sannsynligheten for at hendelsen skal inntre.

2. Utvelgelse av det delintervallet som tilsvarer hendelsen som inntreffer, og deretter gjøre dette delintervallet om til det nåværende intervallet.

Etter alle hendelsene sendes det nok bit slik at det endelige intervallet kan skilles entydig fra alle andre intervall. Det totale antallet bit (B) som benyttes for å skille det endelige intervallet fra andre delintervall er øvre begrenset av:

$$B = \lfloor -\log_2(\ p\ )\rfloor\ + 2 = \lfloor -\log_2(\ p_0\ * p_1\ * \ldots *\ p_{N-1}\ )\rfloor + 2 = \\ \lfloor -\log_2(\ p_0\ )\ -\ \log_2(\ p_1\ )\ -\ \ldots\ -\ \log_2(\ p_{N-1}\ )\rfloor + 2 \tag{2.16}$$

hvor p tilsvarer produktet av de individuelle sannsynlighetene til de N kodete hendelsene [11]. Ligning 2.16 viser at den aritmetiske koderens gjennomsnittlige kodeordslengde (B/N) vil nærme seg kildens entropi når antallet hendelser øker.



Figur 2.10: Aritmetisk koding av eksempelkilde

Aritmetisk koding av den gitte sekvensen og den aktuelle kilden er vist i figur 2.10. Symbolsekvensen 0-1-0-2-0-0, vil bli kodet som 11000111, siden dette er den korteste binære

strengen som har en verdi som ligger innenfor det halvåpne delintervallet. Totalt gir også denne kodingsmetoden 8 bit.

Fordelen med aritmetisk koding fremfor Huffmankoding er at Huffmankoder krever minimum ett bit pr symbol, noe som resulterer i lav kodingseffektivitet for kilder som har en fordeling hvor en hendelse har $p_n > 0,5$ (dette kan løses ved å kombinere Huffmankoding med f.eks. *RLC* [9]). En annen fordel med aritmetisk koding er at den har et klart skille mellom sannsynlighetsmodelleringen og selve kodingsprosessen [11]. Dette medfører at sannsynlighetsmodellene kan være adaptive (dynamisk estimere sannsynligheten basert på alle tidligere hendelser), semi-adaptive (estimere sannsynligheten basert på en treningssekvens eller på et utvalg av tidligere hendelser) eller ikke-adaptive, uten at dette medfører endringer i selve kodingsprosessen (ved Huffmankoding ville alfabetet måtte regenereres for hver gang sannsynlighetsmodellen ble endret).

Den aritmetiske kodingen som er beskrevet ovenfor har to ulemper:

- Når intervallet gjøres mindre, stilles det større og større krav til presisjonen i delintervallberegningene.
- Koderen sender ikke ut noen verdier før samtlige hendelser er kodet.

En løsning på begge disse problemene er at koderen sender de mest signifikante bitene så snart de er kjent. Deretter benyttes lineær interpolasjon for å doble lengden på intervallet, slik at det kun reflekterer den ukjente delen av det endelige intervallet. Det kan også benyttes en mekanisme som forhindrer at intervallet blir veldig lite dersom endepunktene befinner seg nært 1/2, men på hver sin side av 1/2. En slik situasjon innebærer at koderen ikke vet hva neste bit vil være, men at det påfølgende bit'et vil være motsatt av det neste. Dette benyttes ved å inkrementere en etterfølgerteller, og deretter ekspandere intervallet symmetrisk om 1/2 [12].

Hovedtrekkene i utvidede løsningen medfører at følgende rekursive kode legges til etter steg 2, i den tidligere nevnte algoritmen .

3a Dersom det nye delintervallet ikke ligger helt innenfor et av intervallene $[0, 1/2\rangle$,$[1/4, 3/4\rangle$ eller $[1/2, 1\rangle$ avsluttes den rekursive algoritmen.

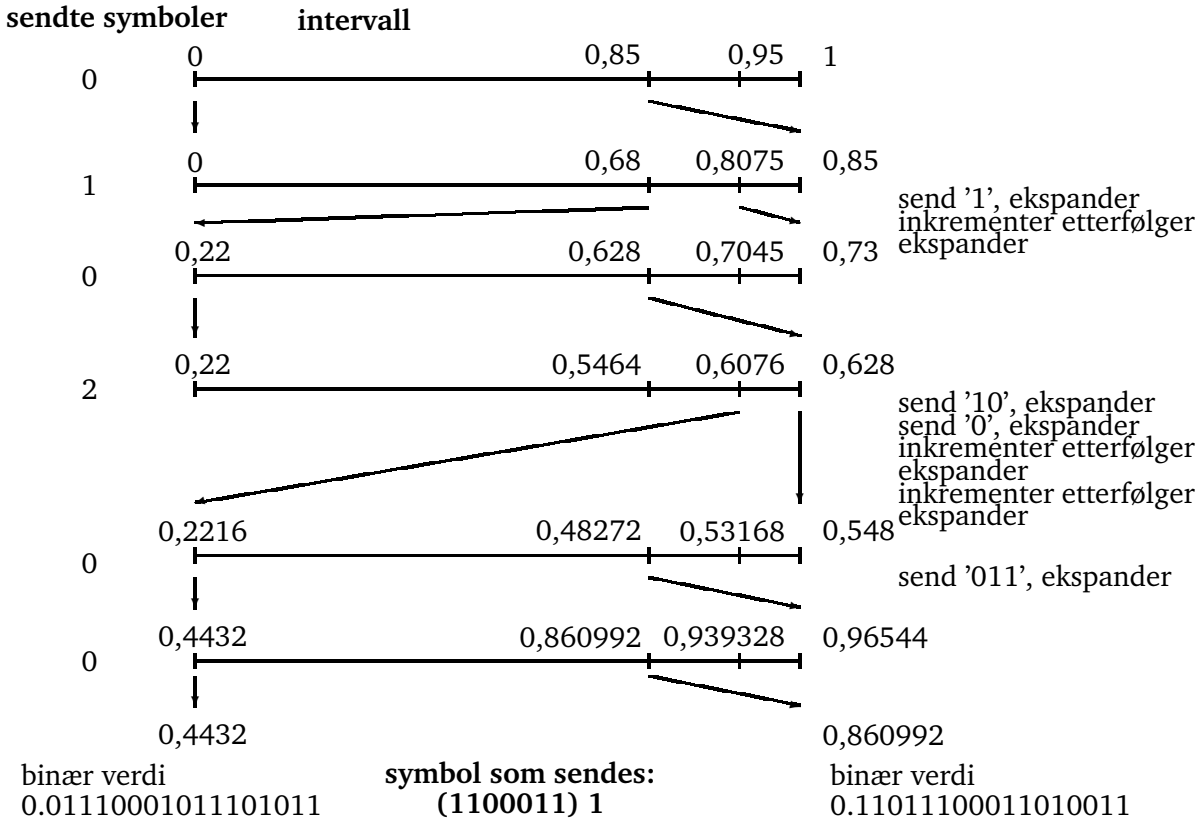
3b Dersom det nye delintervallet ligger helt innenfor $[0, 1/2\rangle$ sender koderen ut en 0, deretter dobles størrelsen på delintervallet ved å lineært ekspandere $[0, 1/2\rangle$ til $[0, 1\rangle$

3c Dersom det nye delintervallet ligger helt innenfor $[1/2, 1\rangle$ sender koderen ut en 1, deretter dobles størrelsen på delintervallet ved å lineært ekspandere $[1/2, 1\rangle$ til $[0, 1\rangle$

3d Dersom det nye delintervallet ligger helt innenfor $[1/4, 3/4\rangle$ inkrementeres etterfølger telleren. Deretter dobles størrelsen ved å lineært ekspandere $[1/4, 3/4\rangle$ til $[0, 1\rangle$.

Den utvidede kodingsprosessen er vist i figur 2.11.

Et annet praktisk problem med aritmetisk koding er at hver hendelse krever en multiplikasjonsoperasjon for å skalere det valgte delintervallet. Muliplikasjoner er kostbare operasjoner i både software og hardwareimplementasjon. En løsning på dette problemet er at det benyttes tilnærmelser (f.eks. i form av kvantisering) på størrelsen av intervallet, og tilnærmelser på sannsynlighetene for de ulike hendelsene [13][4]. Ved å gjøre dette kan multiplikasjonsoperasjonen erstattes med f.eks. tabelloppslag i ferdig definerte tabeller.



Figur 2.11: Aritmetisk koding hvor signifikante bit blir sent når de er kjent, og intervallstørrelsen dobles vha. lineær interpolasjon. De sju første bitene sendes underveis i kodingsprosessen, mens det siste benyttes for å skille det endelige intervallet entydig fra andre delintervall.

2.10 Kvalitetsvurderinger

For å vurdere kvaliteten på det komprimerte signalet må det benyttes et kvalitetsmål som forteller om ulikhetene mellom to signal. Det mest brukte målet for dette er Peak Signal-to-Noise Ration (*PSNR*).

Gjennomsnittlig *PSNR* for en videosekvens beregnes ved å ta gjennomsnittet av den kvadratiske feilen pr ramme. Gitt en sekvens bestående av F rammer, hvor hver ramme har oppløsningen $M * N$ piksel, og hver koeffisient består av k bit, så vil gjennomsnittlig *PSNR* beregnes som:

$$\textit{PSNR}(dB) = 10\ \log_{10}\ \frac{(2^k-1)^2}{\frac{1}{MNF}\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}\sum_{f=0}^{F-1}|x[n,m,f]-\hat{x}[n,m,f]|^2} \tag{2.17}$$

hvor x er det orginale signalet og $\hat{x}$ er det dekodede signalet.

PSNR verdien gir ikke nødvendigvis et resultat som er meningsfylt i seg selv, men den kan benyttes for å sammenligne ulike kompresjonsteknikker. De negative sidene ved bruk av *PSNR* som kvalitetsmål er at metrikken er lite tilpasset menneskets visuelle persepsjon. Øyet er veldig følsomt for feil i kanter og i deterministiske mønster, samtidig som øyet er lite følsomt for feil i stokastiske strukturer [5]. Det positive med *PSNR* som kvalitetsmål er at målingen er lite komplisert, samt at det gir et overslag av kvaliteten på det komprimerte bildet.

Kapittel 3

Kontekstbasert, adaptiv, binæraritmetisk koding

Kapittelet har som formål å nærmere utdype teorien bak CABAC, samt å forklare de ulike bestanddelene i CABAC rammeverket.

CABAC er en av de to spesifiserte entropikodingsmetodene i den nye *ITU-T/ISO/IEC* standarden for videokoding, *H.264/AVC*. Tidligere standardiserte hybride videokodere har benyttet seg av entropikodere basert på faste tabeller for Huffmankoding sammen med *RLC*. Problemet med denne fremgangsmåten er at tabellene ikke er effektive dersom et symbol har sannsynlighet $> 0{,}5$.

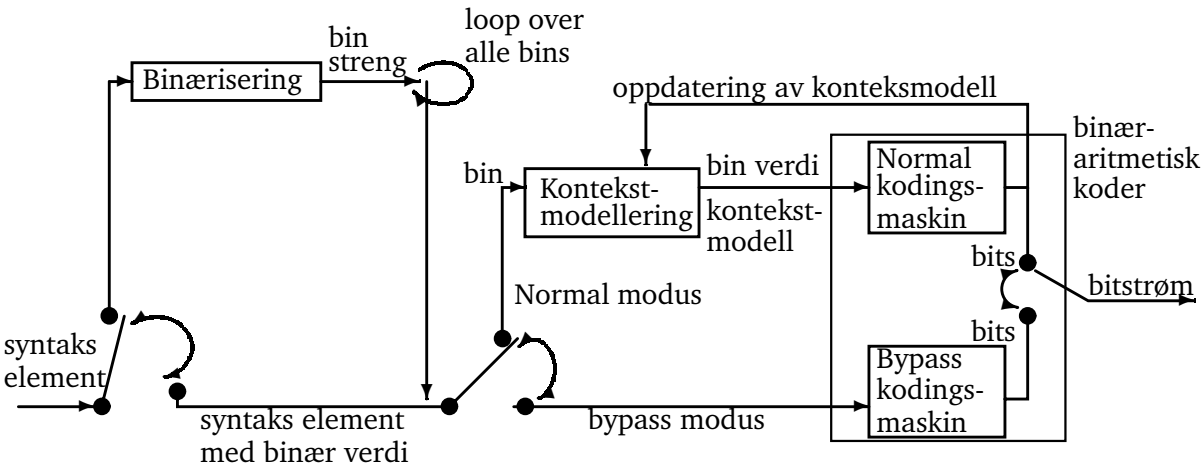
Et annet problem ved bruken av faste tabeller er at de ikke har mulighet til å tilpasse entropikodingen til den aktuelle symbolstatistikken, som kan variere temporalt og romlig, samt for ulike kilder og ulike parameteroppsett i koderen [4]. Det ble tidlig vist at disse problemene kunne overkommes ved å benytte seg av aritmetisk koding [3], ved at det er et klart skille mellom kontekstmodelleringen og selve kodingsoperasjonen. Til tross for dette er aritmetisk koding benyttet i svært liten grad innenfor hybride videokodere, før *H.264/AVC* ble standardisert (kun benyttet i H.263 - tillegg 3[4]). Innenfor 3D-videokodere har aritmetisk koding vært benyttet i noe større grad.

3.1 CABAC rammeverket

Figur 3.1 viser et blokkdiagram for kodingen av et syntakselement i CABAC. Selve kodingsprosessen består av tre operasjoner:

1. Binærisering
2. Kontekstmodellering
3. Binæraritmetisk koding

Den første delen avbilder et ikke-binært syntakselement om til en binær sekvens, også kalt binstreng. Dersom syntakselementet som skal kodes allerede er binært, hoppes dette steget over, som vist i nedre gren i figur 3.1. For hvert element (kalt bin) i binstrengen, eller for hvert binære syntaks element gjøres ett eller to av de følgende stegene, avhengig av kodingsmodusen:



Figur 3.1: Blokkdiagram av CABAC koder, figuren er hentet fra [4]

1. Ved normalmodus (*regular coding mode*) sendes hver bin til en kontekstmodelleringsmaskin, hvor en sannsynlighetsmodell blir valgt. Deretter sendes modellen og binverdien til en kodingsmaskin hvor elementet blir kodet, før det gjøres en oppdatering på sannsynlighetsmodellen.

2. Bypassmodus benyttes for å senke kompleksiteten på enkelte elementer. Dette gjøres ved å droppe kontekstmodelleringssteget, og benytte lik sannsynlighet for det mest sannsynlige symbolet (*MPS*) og det minst sannsynlige symbolet (*LPS*).

3.2 Binærisering

CABAC benytter seg av fire basismetoder for binærisering: *unary* kode, *truncated unary* kode, *k*te ordens Exp-Golomb kode og fastlengde kode. De fire basismetodene kan også slåes sammen og danne nye binæriseringsmetoder.

3.2.1 *Unary* kode og *Truncated Unary (TU)* kode

For et gitt heltall $x \geq 0$ består *unary* kodeordet av x antall '1' bit pluss en terminerende '0' bit. *TU* kodeordet skiller seg fra *unary* kodeordet ved at det kun er definert for $0 \leq x \leq S$. Kodeordene er like for de to metodene for $x < S$, men for $x = S$ blir den terminerende '0' bit'en neglisjert, slik at *TU* kodeordet består kun av S antall '1' bit.

3.2.2 *k*te ordens Exp-Golomb kode

Exp-Golomb koder består av en prefiks- og en suffiksdel. Prefiksdelen består av $l(x) = \lfloor \log_2(x/2^k + 1) \rfloor$ antall '1' bit, mens suffiksdelen beregnes som binærrepresentasjonen av $x + 2^k(1 - 2^{l(x)})$ når det benyttes $k+l(x)$ signifikante bit. Ved å velge en passende verdi for k kan en god førsteordens tilnærming av en ideell prefiksfri kode for halen til typisk observerte sannsynlighetstetthetsfunksjoner oppnås [4].

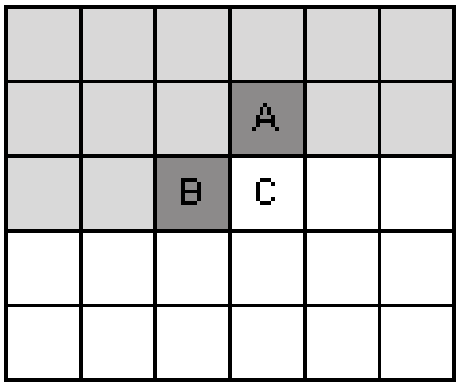
3.2.3 Fastlengde binærisering (*FL*)

FL binærisering baseres på at syntakselementet har et alfabet med en gitt størrelse. Gitt verdien x til et slikt element, hvor $0 \leq x < S$, vil *FL* representasjonen av x være gitt ved binærrepresentasjonen av x med et minimum fast antall bit lik $\lceil \log_2 S \rceil$. *FL* benyttes vanligvis for syntakselementer tilnærmet uniform fordeling.

3.3 Kontekstmodellering

En fordel med aritmetisk koding er at kontekstmodelleringen og selve kodingen skjer i to trinn. Dette medfører at det er mulig å utnytte høyere ordens statistiske sammenhenger (betingete sannsynligheter mhp. flere nabosymbol), samt muligheten for å oppdatere modellene underveis, uten at det påvirker selve kodingen.

Et problem med å benytte høyere ordens statistiske sammenhenger, er at dette krever svært mange estimeringer av betingede sannsynligheter. CABAC har løst dette problemet ved å velge en middelvei mellom ønsket ytelse og kompleksitet. Selve kontekstmodelleringen utføres ikke på samtlige elementene i binstrengen, samt at det kun benyttes et fåtall av nabosymbol for estimering.



Figur 3.2: Kontekstmønster hvor to nabosymbol (A og B) benyttes for estimering av det nåværende symbolet C. De ulike symbolene kan f.eks. være en blokk av koeffisienter. Figuren er laget fra en skisse i [4]

Innenfor CABAC benyttes det fire ulike designtyper for kontekstmodellering:

1. Konteksten avhenger av opp til to tidligere kodede nabosymbol, hvor regelen for hva som er nabosymbol kommer an på typen av symbol som skal kodes. Den mest vanlige metoden er basert på nabosymbolet over (A) og nabosymbolet til venstre (B) for det aktuelle symbolet (C), som vist i figur 3.2.

2. Konteksten avhenger av verdiene til samtlige tidligere kodede elementer av samme type.

3. Den tredje typen benyttes i kodingen av transformkoeffisienter og avhenger av blokktypen (definert i *H.264/AVC*) samt posisjonen i en zig-zag skan.

4. Den fjerde typen benyttes også i kodingen av transformkoeffisienter, og avhenger i likhet med den tredje typen også av blokktypen. I tillegg avhenger konteksten av det akkumulerte antallet tidligere kodede elementer med denne verdien.

Totalt eksisterer det 398 kontekstmodeller i CABAC i *H.264/AVC*, som hver har en kontekstindeks γ. Hver kontekstmodell har en sannsynlighetsmodell som kan beskrives som en sum av to verdier, indeksen til sannsynlighetstilstanden σ_γ, samt den binære verdien til *MPS* symbolet ϖ_γ. En oversikt over hvilke kontekstmodeller som benyttes for koding av ulike symboler i *H.264/AVC* finnes i [4].

3.4 Binæraritmetisk koding

Ved bruk av en binæraritmetisk koder vil kodingsprosessen beskrevet i eksempelet i del 2.9 bli noe forenklet siden det kun er to symboler, *MPS* og *LPS*.

Gitt sannsynligheten til *LPS*, $p_{LPS} \in \langle 0\ ,\ 0,5]$, og det aktuelle intervallet, beskrevet av intervallets nedre grense L, og intervallbredden R. Intervallet kan da deles opp i to delintervall, hvor bredden på det ene er gitt ved:

$$R_{LPS} \ = \ R{*}p_{LPS} \tag{3.1}$$

Bredden på det andre er gitt ved $R_{MPS} \ = \ R - R_{LPS}$ og sannsynligheten for *MPS* er gitt ved $1 - p_{LPS}$. Avhengig av hendelsen som skal kodes, velges deretter et av delintervallene som nytt intervall.

Hovedproblemet ved implementeringen av en slik løsning, er kostnaden ved multiplikasjonen i ligning 3.1. For å løse dette problemet er det i CABAC innført multiplikasjonsfri variant av aritmetisk koding, kalt modulokoder (M-koder). Koderen tilnærmer både bredden ($R \ \in \ [R_{min}\ ,\ R_{max}\rangle$) og verdien til p_{LPS} til et sett av endelige verdier $\mathbf{Q} = \{Q_0, Q_1, Q_2, Q_3\}$ og $\mathbf{P} \ = \ \{p_0, p_1, \ldots, p_{63}\}$. Den høyre delen av ligning 3.1 kan dermed erstattes med et tabelloppslag i en ferdig definert tabell (med dimensjonen 4x64) som inneholder produktene $Q_\rho * p_\sigma$ for $\{0 \ \leq \ \rho \ \leq \ 3\}$ og $\{\ 0 \ \leq \sigma \ \leq \ 63\}$ [4].

3.4.1 Normal- og Bypassmodus

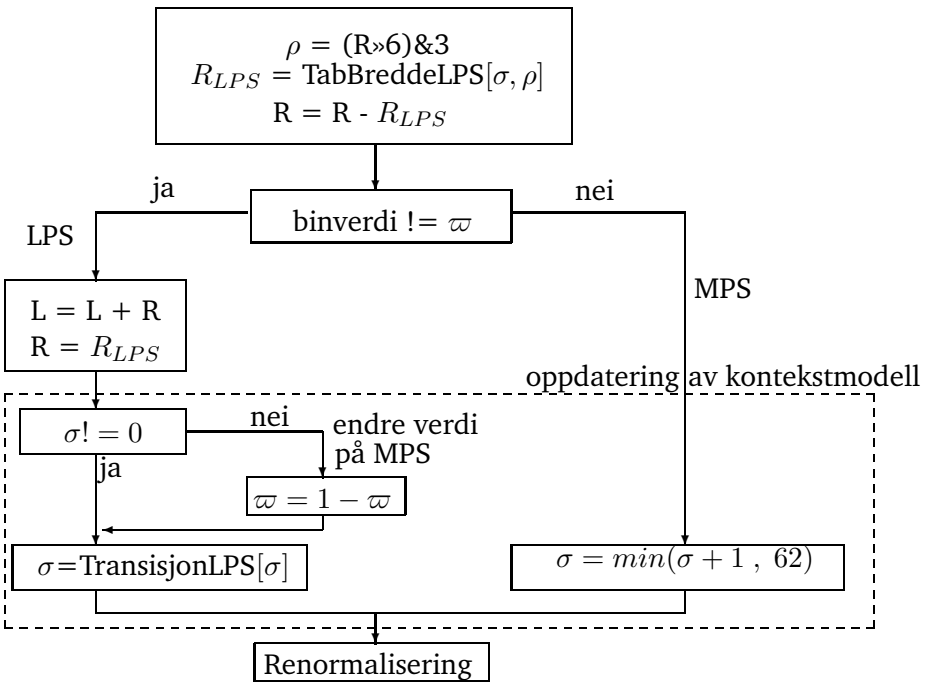
Kodingen i normalmodus er vist i figur 3.3. Koderen benytter en gitt kontekstmodell γ, som er bestemt av tilstanden σ_γ, og *MPS* verdien ϖ_γ. Kodingsintervallet er beskrevet ved den nedre grensen L, samt bredden på intervallet R.

Kodingen i normalmodus skjer i en firestegs operasjon. Den første operasjonen er at intervallet blir delt i to delintervall. Dette gjøres ved å benytte en tilnærmet verdi for bredden på intervallet. Istedenfor å direkte benytte seg av den tilnærmede verdien, klassifiseres verdien med en indeks ρ. Indeksen beregnes ved å dividere bredden R med 64, og deretter benytte to signifikante bit for å representere indeksen. Dette implementeres effektivt ved å benytte:

$$\rho = (R >> 6)\&3 \tag{3.2}$$

Deretter gjøres det et tabelloppslag, hvor ρ og kontekstilstanden σ_γ benyttes, og får ut bredden på delintervallet R_{LPS}.

Den neste operasjonen sjekker om binverdien tilsvarer *MPS* (ϖ) eller *LPS*. En binverdi lik *MPS* tilsvarer høyre gren i figur 3.3, mens venstre gren tilsvarer *LPS*.



Figur 3.3: Blokkdiagram over den aritmetiske kodingen i normalmodus

Det tredje steget innebærer oppdatering av kontekstmodellen, nærmere beskrevet i del 3.5.1, hvor modellens nye tilstand avhenger om hendelsen som ble kodet tilsvarte *MPS* eller *LPS*.

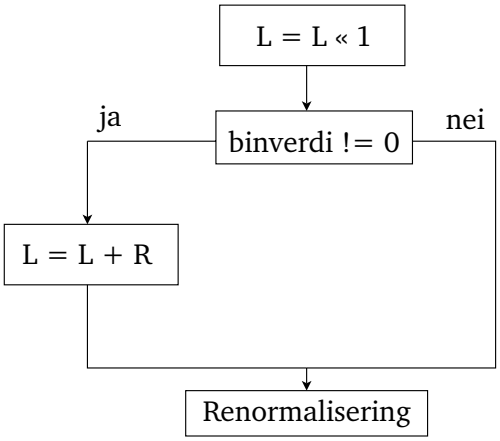
Det siste steget er renormalisering, som forhindrer at bredden på intervallet går utenfor sitt lovlige område. Dette er nærmere beskrevet i del 3.6.

Kodingen i bypassmodus skjer for å øke hastigheten på koding og dekodingen av symbol. Dette innebærer at det ikke benyttes en kontekstmodell, og delintervallbredden beregnes ved $R - R_{LPS} \approx R_{LPS} \approx R/2$. Den første operasjonen i bypassmodusen er at den nedre grensen til intervallet dobles ($L = L << 1 = L * 2$). Dette gjøres istedenfor å halvvere bredden på intervallet, noe som medfører at en slipper doblingen av L og R i renormaliseringsfasen. Deretter velges et av delintervallene utifra binverdien, før L renormaliseres. Kodingsforløpet i bypassmode er vist i figur 3.4.

3.5 Sannsynlighetsestimering i CABAC

Hovedidéen bak en multiplikasjonsfri aritmetisk koder er at estimerte sannsynlighetsverdier kan representeres av et tilstrekkelig stort sett av representasjonsverdier. For CABAC benyttes et utvalg av 64 representasjonsverdier $p_\sigma \in [0,01875\ ,\ 0,5]$, som er utledet fra følgende rekursive ligning:

$$p_\sigma = \alpha * p_{\sigma-1} \quad \text{for } \sigma = 1, \ldots, 63$$
$$\text{hvor } \alpha = \left(\frac{0,01875}{0,5}\right)^{1/63} \qquad \text{og } p_0 = 0,5 \tag{3.3}$$



Figur 3.4: Blokkdiagram over den aritmetiske kodingen i bypassmodus

Her er både skaleringsfaktoren $\alpha \approx 0,95$ og N=64 et kompromiss mellom muligheten for rask tilpasning til kilden ($\alpha \rightarrow$0 og liten N) og nødvendig nøyaktighet ($\alpha \rightarrow$1 og stor N).

Et resultat av denne oppbyggingen er at hver kontekstmodell kan fastsettes av to parametere: en 6-bits tilstandsindeks (som angir sannsynligheten til *LPS*), samt en 1-bits verdi ϖ, som representerer *MPS* verdien. Med bakgrunn i denne oppbyggingen, benyttes det totalt 128 (2^7) ulike sannsynlighetstilstander i CABAC, hvorav en tilstand (σ=63) er en ikke-adaptiv tilstand som benyttes for terminering av kodeord [4].

3.5.1 Oppdatering av kontekstmodell

Oppdateringen av kontekstmodellen skjer etter kodingen av hver bin i normalmodus. Oppdateringen medfører at modellens tilstand blir oppdatert etter følgende regler:

1. Dersom hendelsen som kodes er lik *MPS*, øker tilstandsindeksen med en, med mindre den allerede er i tilstand 62 hvor p_{LPS} allerede er på minimum.

2. Dersom hendelsen som kodes er lik *LPS*, og tilstanden er lik 0 (lik sannsynlighet for *LPS* og *MPS*), forblir tilstandsindeksen uforandret, men verdien til *MPS* (ϖ) endres.

3. Dersom hendelsen som kodes er lik *LPS*, og tilstanden ikke er lik 0, gjøres det et tabelloppslag i en TransisjonLPS tabell (som vist i figur 3.3), som angir den nye indeksen. Transisjonspilene fra gammel til ny tilstand er vist i figur 3.5.

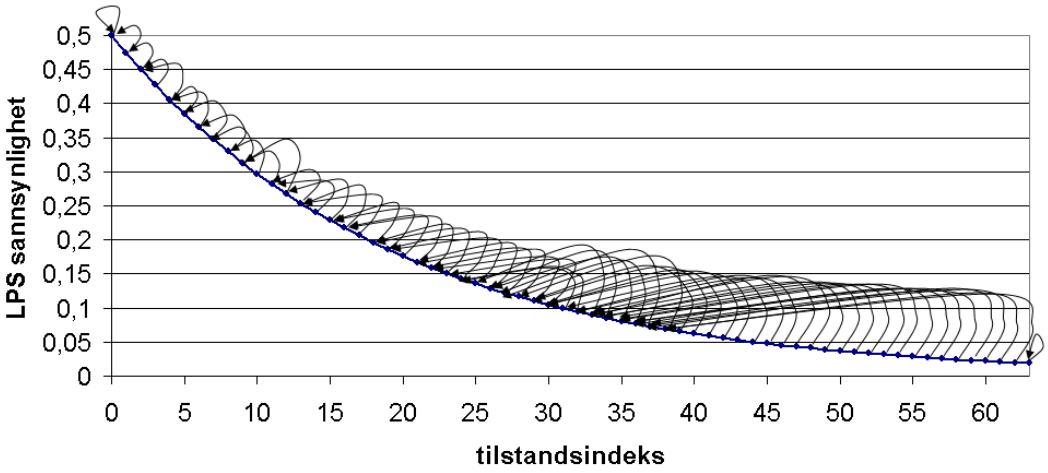
Den nye sannsynligheten for *LPS* kan også beregnes utifra følgende ligning:

$$p_{ny} = \begin{cases} \max(\alpha * p_{gammel}, p_{62}), & \text{dersom } \textit{MPS} \text{ forekommer} \\ \alpha * p_{gammel} + (1 - \alpha), & \text{dersom } \textit{LPS} \text{ forekommer.} \end{cases} \tag{3.4}$$

hvor α er gitt i ligning 3.3.

3.5.2 Initialisering av kontekstmodell

Initialisering av kontekstmodeller må gjøres for hver gang en ny uavhengig enhet skal kodes. En slik enhet kan f.eks. være en *slice* i *H.264/AVC*. Siden en slik enhet ikke kan benytte seg



Figur 3.5: Transisjonstilstandsdiagram for kontekstmodellens nye tilstand etter oppdateringen. Pilene går fra nåværende tilstand til ny tilstand dersom *LPS* kodes. Dersom *MPS* kodes vil tilstandsindeksen øke med en.

av andre enheters adapsjon til kilden, og den selv ikke har noe kunnskap om den aktuelle kilden, må kontekstmodellene re-initialiseres før kodingen/dekodingen kan starte.

Den enkleste formen for re-initialisering vil være å sette kontekstmodellene til tilstand '0', hvor det er lik sannsynlighet for *MPS* og *LPS*. Et problem med denne løsningen er at koderen vil bruke tid på å tilpasse seg kilden på nytt, som vil resultere i lavere ytelse i adapsjonsfasen.

CABAC løser dette problemet ved å tilby en løsning som gir kontekstmodellene noe kunnskap om kilden ved initialiseringen, basert på kodingsparameterene. Løsningen tilbyr tilpasning av kontekstmodellenes starttilstand på to nivåer.

Initialisering avhengig av kvantiseringstrinnstørrelsen (*SliceQP*)

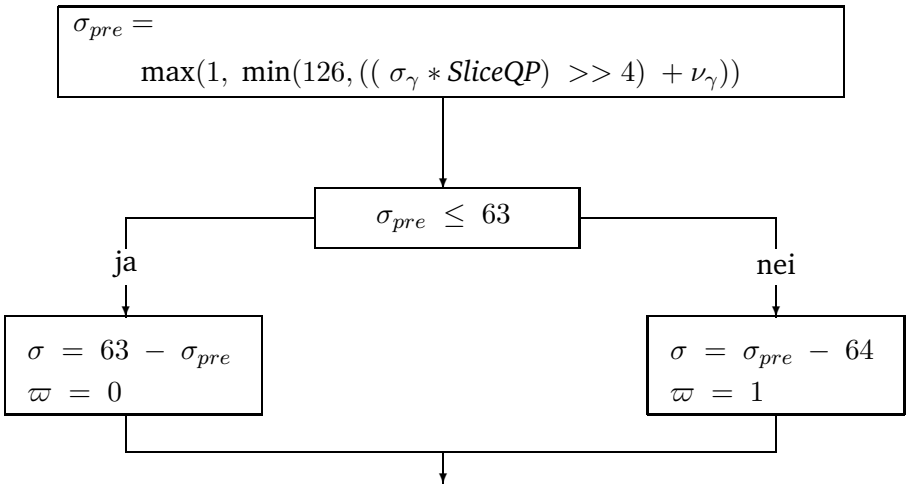
Valg av starttilstand på de ulike kontekstmodellene avhengig av kvantiseringsparameteren, er basert på målinger gjort på treningssekvenser. Ved hjelp av lineær regresjon har det for hver modell blitt beregnet to parametere (μ_γ, ν_γ), som bestemmer starttilstanden til kontekstmodellen utifra ligningen vist i figur 3.6.

Slice-avhengig initialisering

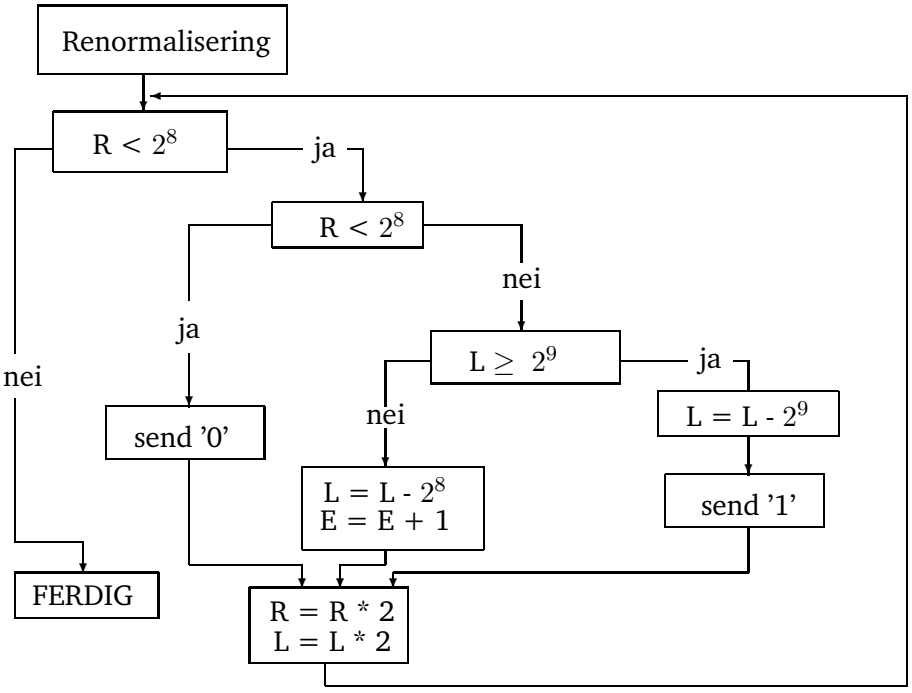
Metoden baserer seg på at det kan benyttes ulike initialiseringsparametere spesielt for de kontekstmodellene som benyttes for koding av P- og B-*slices* i *H.264/AVC* [4]. På denne måten koderen velge mellom flere initaliseringstabeller for kodingen av disse elementene. Koderen kan dermed oppnå bedre tilpasning til ulike kodingsscenarioer og videoinnhold.

3.6 Renormalisering i CABAC

Renormalisering benyttes i aritmetisk koding for å begrense kravet til nøyaktigheten, samt at dette lar koderen sende de signifikante bit'ene etterhvert som de bestemmes. Renormaliseringsalgoritmen som er valgt i CABAC benyttes dersom bredden på intervallet kommer



Figur 3.6: Kvantiseringstrinnavhengig initialisering



Figur 3.7: Renormalisering i CABAC-koder. **L**=nedre grense på intervallet, **R**=bredden på intervallet og **E**=etterfølger bit som beskrevet i 2.9. Figuren er laget etter en skisse i [14]

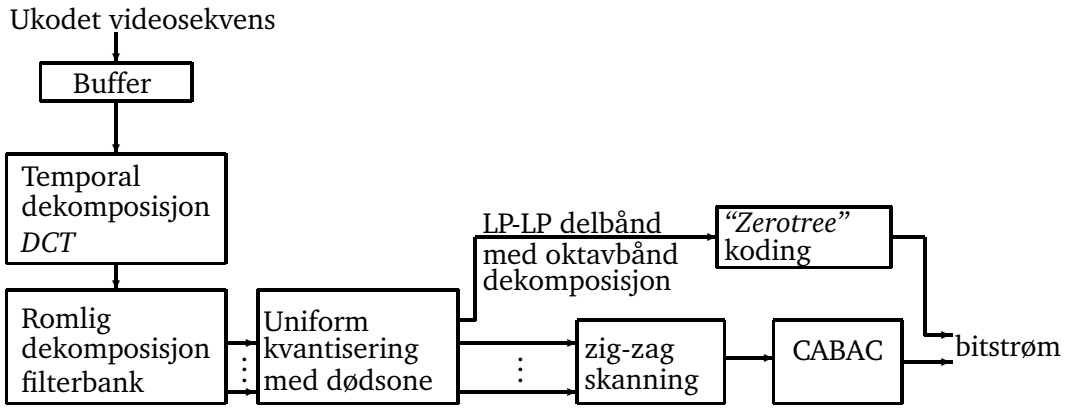
utenfor det lovlige området $\left[\ 2^8\ 2^9\ \right\rangle$, og er hentet fra [15]. Kun dekodingsalgoritmen er spesifisert i CABAC [14], men en mulig implementasjon i koderen er vist i figur 3.7.

Kapittel 4

Fremgangsmåte

Kapittelet har som formål å forklare den foreslåtte koderens oppbygging, samt grunngi hvorfor de ulike delene er valgt.

4.1 Foreslått koder



Figur 4.1: Foreslått 3D-koder

Blokkdiagrammet over den foreslåtte koderen er vist i figur 4.1. Koderen bufrer opp et sett av rammer, dekomponerer sekvensen temporalt vha. *DCT* og fjerner romlig redundans vha. en 64-delbånds filterbank med oktavbånd struktur i LP-LP delbåndet. Koeffisisentene blir kvantisert med en skalar uniform kvantiserer med dødsone. LP-LP delbåndet blir kodet med en *"zerotree"* koder som utnytter effektkorrelasjonen mellom de ulike nivåene i oktavbånd strukturen, mens koeffisientene i de resterende delbåndene blir zig-zag skannet og deretter entropikodet med CABAC.

4.2 Temporal dekomposisjon

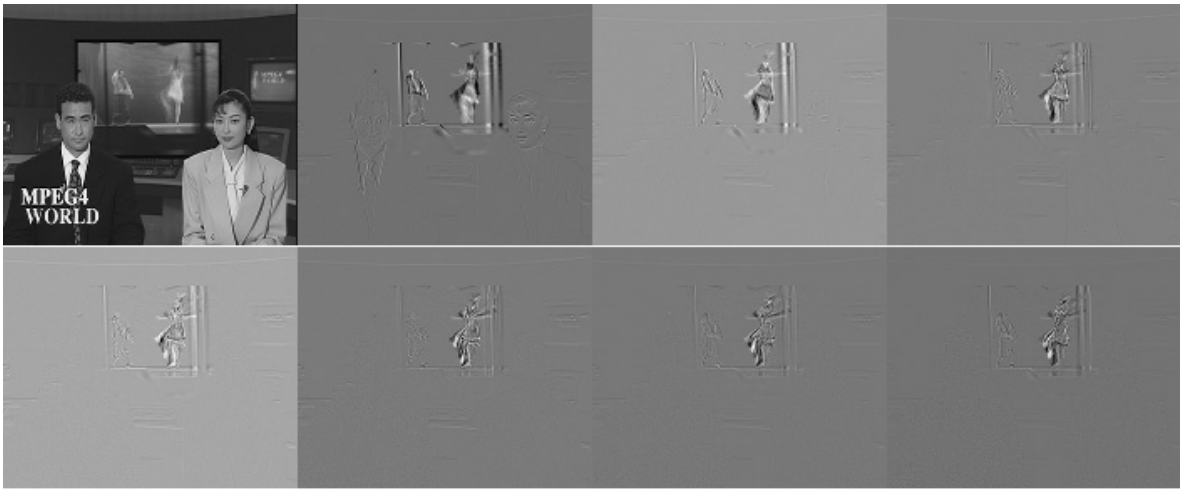
Oppbyggingen av videosekvenser innebærer ofte mange påfølgende rammer med svært lignende karakter. Forskjellene mellom påfølgende rammer skyldes ofte forflytningen av objekter, objekter som skjules bak andre eller kommer frem fra andre, eller generell kamerabeve-

gelse (som f.eks. panorering eller zooming). Bevegelsesestimering og -kompensasjon kan helt eller delvis kompensere for de fleste av endringene mellom rammene. Problemet er at dette er en kostbar og krevende operasjon for koderen, i tillegg til at det legger begrensninger på valget av romlig dekomposisjonsmetode.

Et problem med 3D-videokodere (med eller uten bevegelseskompensasjon) er at de må bufre opp rammer før den temporale dekomposisjonen kan utføres. Denne operasjonen medfører en forsinkelse i systemet, samt at det øker minnebehovet til koderen og dekoderen.

Den foreslåtte koderen benytter seg av *DCT* som temporal dekomposisjonsmetode. Dette valget er basert på en balanse mellom ønsket kompleksitet og ytelse, samt at det ikke legger noen begrensning på den romlige dekomposisjonsmetoden. *DCT* har dekorreleringsegenskaper nær *KLT* for *AR(1)* prosesser, og er i tillegg signaluavhengig. Metoden vil dermed kunne pakke informasjonen godt i de områdene hvor det er stor likhet mellom påfølgende rammer, mens bevegelser av objekter vil kunne merkes som høye frekvenskomponenter. Ved bruk av romlig dekomposisjon og effektive entropikodere kan disse komponentene kodes effektivt. Et problem med den valgte metoden er at den ikke har mulighet til å kompensere for kamerabevegelser som panorering og zooming, noe som vil medføre at den vil yte dårligere enn 3D-kodere med bevegelseskompensasjon for sekvenser hvor dette er utbredt.

Utgangssignalet etter temporal dekomposisjon er vist i figur 4.2, og den viser tydelig hvordan objektenes bevegelser vises som høye frekvenser i de ulike rammene.



Figur 4.2: Skalert versjon av utgangsignalet fra den temporale dekomposisjonen (ved bufferstørrelse 8). Hver ramme er skalert uavhengig av de andre rammene, for at verdiene i rammen skal være tilpasset intervallet [0 255].

4.3 Romlig dekomposisjon

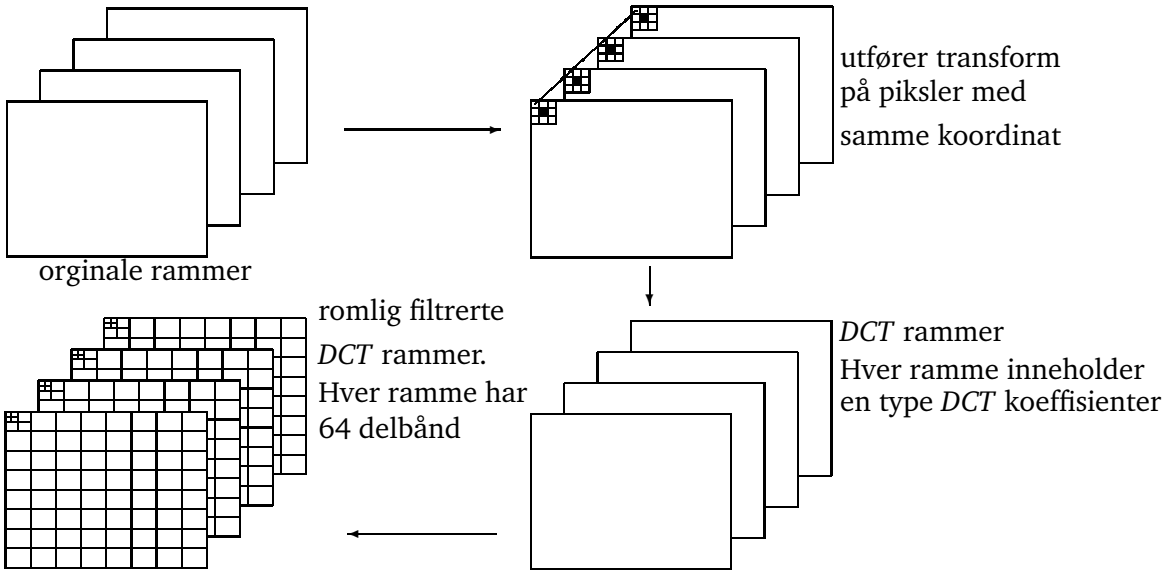
En ulempe ved å benytte blokkbasert bevegelsesestimering og -kompensasjon, er at det kan oppstå kantoverganger mellom de ulike blokkene. Et resultat av dette er at det må benyttes blokkbasert romlig dekomposisjon, noe som ikke vil kunne utnytte interblokk redundansen. Dersom det benyttes en temporal dekomposisjon uten blokkbasert bevegelsesestimering og -kompensasjon vil koderen slippe kravet om blokkbasert romlig dekomposisjon, og dermed

benytte seg av mer effektive dekomposisjonsmetoder som f.eks. filterbanker.

Den foreslåtte koderen benytter en 64-delbånds, ikke-unitær *FIR* filterbank utviklet av Aase [16]. Filterbanken er ikke *PR*, og er utviklet for å maksimere kodingsgevinsten samt minimere visuelle artefakter ved lavratekoding. Lavpassdelbåndet deles opp i en trestruktur ved innføring av en oktavbånd filterbank i inntil tre nivåer [17]. Oktavbånd filterbanken er nøye utvalgt fra et stort utvalg *PR FIR* filterbanker [5].

Fordelen med å dekomponere kilden til 64 delbånd er at koderen oppnår god frekvensoppløsning på dekomposisjonen, på bekostningen av den romlige oppløsningen innad i hvert delbånd (grunnet kritisk nedsampling). Trestrukturen i lavpassdelbåndet bidrar til at effektkorrelasjonen mellom de ulike nivåene kan utnyttes vha. *"zerotree"* koding.

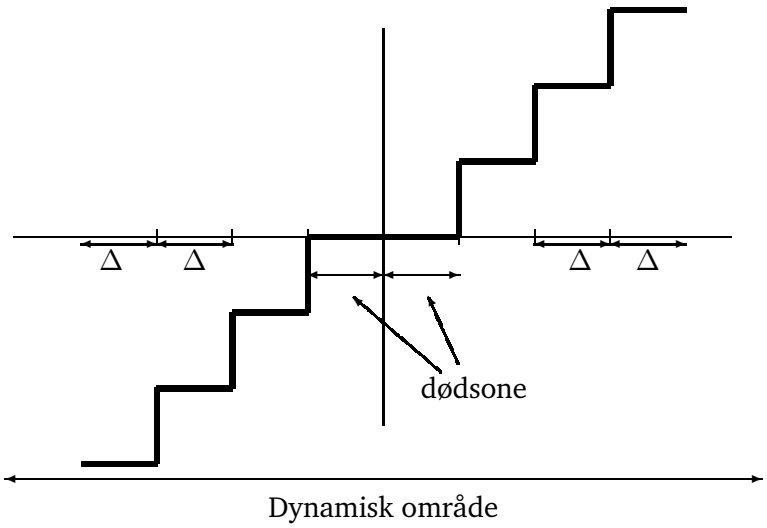
En oversikt over hele dekomposisjonen er vist i figur 4.3.



Figur 4.3: 3D-dekomposisjon av en videosekvens. Orginale rammer bufres opp, før *DCT* operasjonen utføres på pikselnivå i tidsretningen. Resultatet blir nye frekvensrammer, som romlig dekomponeres vha. filterbanken. Utgangssignalet blir nye rammer som hver er inndelt i 64-delbånd.

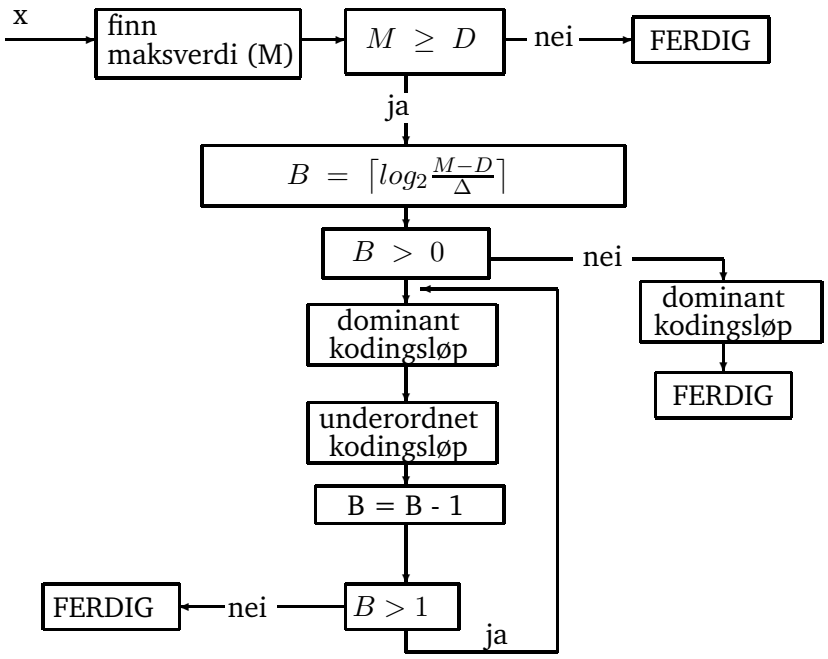
4.4 Kvantisering

Kvantisererens oppgave er å tilpasse kilden til et endelig sett av representasjonsverdier. Den temporale og romlige dekomposisjonen har som oppgave å gjøre signalet inn i kvantisereren ukorrelert (hvitt). Dette medfører at det kan benyttes skalar kvantisering og entropikoding, uten at ytelsen blir betydelig lavere enn ved vektorkvantisering. Bruk av skalar kvantisering og entropikoding vil også kunne implementeres med langt lavere kompleksitet enn vektorkvantisering. Kvantisereren som er benyttet i den foreslåtte koderen er en 11-bits skalar uniform *midtread* kvantiserer med dødsone rundt nullområde, som vist i figur 4.4. Dødsonen er benyttet for å kunne kvantisere flere koeffisienter med lav verdi til null, noe som har vist seg å gi økt kodingsgevinst innenfor bildekoding [10].



Figur 4.4: Skalar uniform kvantiserer med dødsone. Dødsonen er her satt lik Δ, noe som tilsvarer alle verdier i området $[-\Delta\ ,\ \Delta]$ vil bli kvantisert til null.

4.5 *"Zerotree"* koding

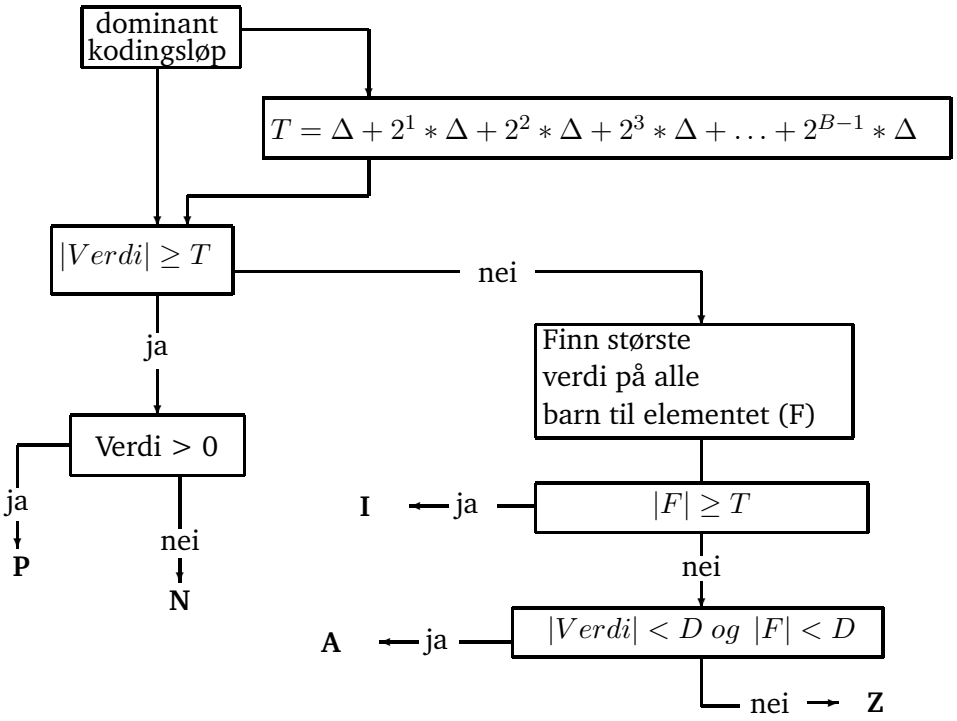


Figur 4.5: *"Zerotree"* koding av lavpassdelbåndet. **M**=amplituden til det største elementet i delbåndet, **D**=størrelsen på dødsonen, **B**=antall bit kvantisereren benytter for å kvantiserer **M** med trinnstørrelsen Δ.

Dekomposisjonsmetoden i den foreslåtte koderen medfører effektkorrelasjon mellom punkter i de ulike nivåene i lavpassdelbåndet. Dette utnyttes i den foreslåtte koderen ved at del-

båndet kodes med en modifisert utgave av *EZW*.

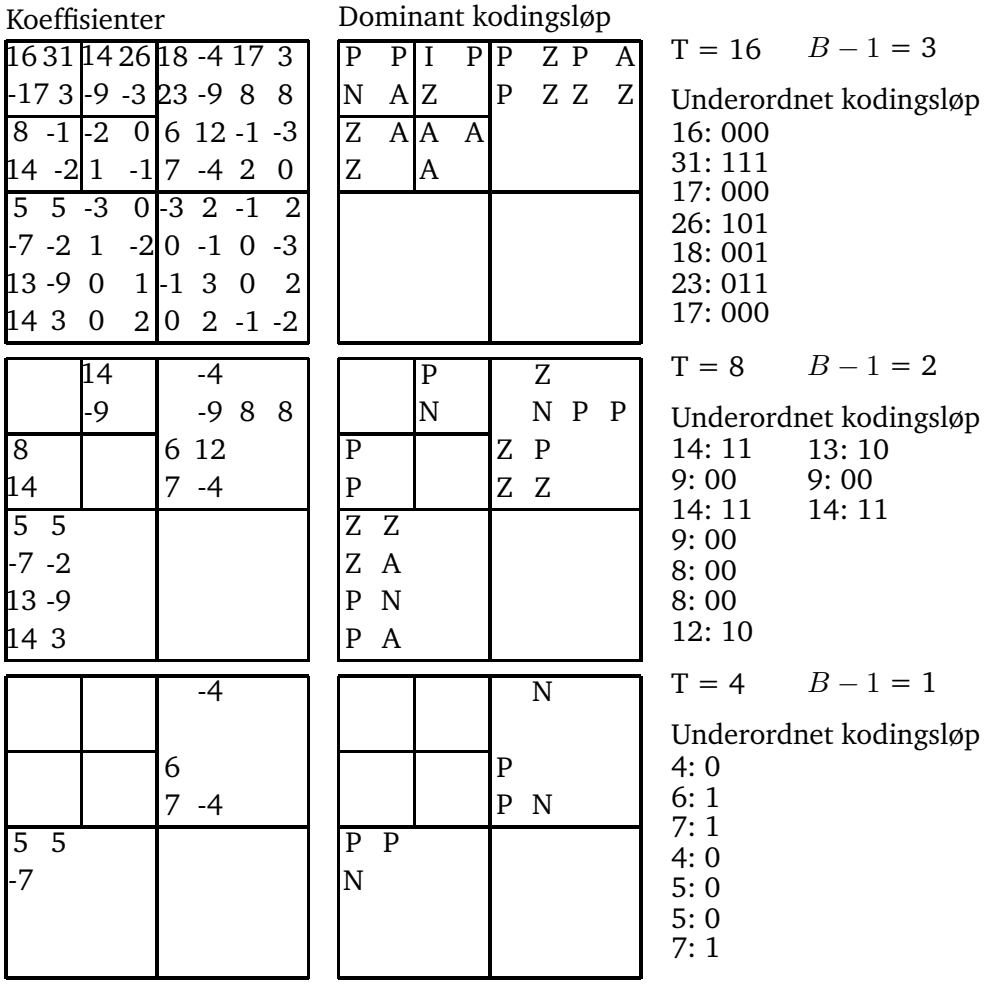
De ulike operasjonene i den implementerte *"zerotree"* algoritmen, er vist i figur 4.5. En mer inngående beskrivelse av det dominante kodingsløpet er vist i figur 4.6. Alle koeffisienter som blir kodet som signifikant (positiv eller negativ) får absoluttverdien kodet med *PCM* koding i det underordnete kodingsløpet. *PCM* verdien beregnes etter formelen $\left\lfloor \frac{|Verdi|-T}{\Delta} \right\rfloor$ med $B-1$ bit, og den foreslåtte koderen skiller seg dermed fra *EZW* koding ved at den ikke er *embedded*.



Figur 4.6: Dominant kodingsløp i *"zerotree"* kodingen. Koeffisienten kodes som positiv (**P**) eller negativ (**N**) signifikant dersom den har amplitude større enn grenseverdien (**T**). Grenseverdien (**T**) avhenger av dødsonen (**D**), kvantiseringstrinnstørrelsen (Δ), samt antallet bit (**B**) som behøves for å kvantisere den største verdien i delbåndet. Dersom koeffisienten ikke er signifikant, hentes den største amplituden (**F**) fra dens barn (i de lavere delbåndene), og koeffisienten kodes som isolert nullverdi (**I**) hvis ett eller flere barn er signifikante. Dersom koeffisienten og samtlige barn er mindre enn dødsonen (**D**) kodes koeffisienten og alle barnene som "absolutt *zerotree*" (**A**), hvis ikke kodes de som *"zerotree"* (**Z**)

Det dominante kodingsløpet benytter en fast Huffmantabell for å kode signifikanskartet. Denne tabellen er tilpasset fordelingen av de ulike symbolene, hvor *"zerotree"* symbolet har høyest sannsynlighet. Et numerisk eksempel på det dominante og det underordnete kodingsløpet er vist i figur 4.7.

Et viktig poeng er at dersom delbåndet som skal kodes tilhører den første rammen ut fra DCT dekomposisjonen, vil middelverdien i øverste nivået (LL_3 i figur 2.8) være lik gjennomsnittsintensiteten i rammene. Denne middelverdien må kompenseres for ved at den beregnes, sendes til mottakeren og deretter subtraheres fra koeffisientene i delbåndet før *"zerotree"* kodingen utføres.



Figur 4.7: Dominant og underordnet kodingsløp i *"zerotree"* kodingen. Figuren viser koeffisientene og verdiene som kodes i det dominante kodingsløpet, med kvantiseringstrinn $\Delta = 2$ og dødsone 2*Δ=4.
T=grenseverdi, **P**=positiv signifikant verdi, **N**=negativ signifikant verdi, **I**=isolert nullverdi, **Z**=*"zerotree"* og **A**="absolutt *zerotree*". Det underordnete kodingsløpet koder koeffisientens usikkerhetsintervall (verdi minus grenseverdi) med $B-1$ bits *PCM* koding. Merk at samtlige koeffisienter som blir kodet som **P**, **N** eller **A** ikke tas med i det neste dominante kodingsløp.

4.6 CABAC

Entropikoderen har som oppgave å kode kildens alfabet vha. en eller flere passende sannsynlighetsmodeller. Ulike sekvenser vil ha ulike statistiske egenskaper grunnet ulikt innhold og/eller måten sekvensen er filmet. I tillegg vil en sekvens ha temporalt og romlig varierende lokal statistikk. Dette medfører at en entropikoder som benytter seg av høyere ordens statistiske egenskaper vil oppnå høyere ytelse enn faste stasjonære entropikodere, som ble benyttet i tidligere videokodingsstandarder.

CABAC er implementert i den foreslåtte koderen som en løsning på problemene nevnt ovenfor, noe som medfører at kontekstmodelleringen og den aritmetiske kodingen er skilt fra hverandre. En følge av dette er at ulike kontekstmodeller kan benyttes for forskjellige elementer, samt at de ulike kontekstmodellene kan oppdateres underveis i kodingsprosessen. Dette medfører at koderen kan tilpasse seg kildens statistikk. CABAC-implementasjonen i den foreslåtte koderen er basert på CABAC-delen av *H.264/AVC* referansekoderen funnet i [18, versjon JM10.2]

Selve entropikodingsprosessen i den foreslåtte koderen starter ved initialisering av kontekstmodellene (beskrevet i del 4.6.1), deretter avbildes signalet fra et todimensjonalt til et endimensjonalt signal vha. zig-zag skan (beskrevet i del 4.6.2). De ikke-binære dekomposisjonskoeffisientene binæriseres vha. *UEG0* (beskrevet i del 4.6.3), og deretter velges det en kontekstmodell for hver bin (beskrevet i del 4.6.4). Kontekstmodellen og binverdien sendes deretter inn en artimetisk koder som utfører kodingen av elementet (beskrevet i del 4.6.5 og i del 4.6.6). Til slutt renormaliseres intervallet (som tidligere beskrevet i del 3.6) og kontekstmodellen oppdateres (beskrevet i del 3.5.1).

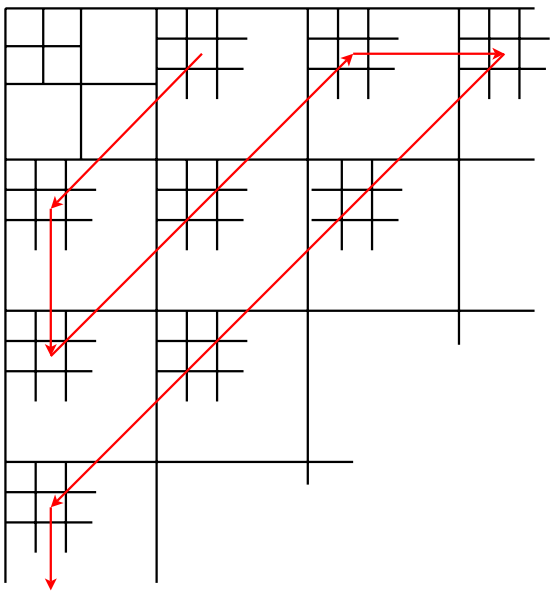
4.6.1 Initialisering av kontekstmodell

De ulike kontekstmodellene initialiseres for hver dekomponerte ramme i koderen. På denne måten vil entropikoderen lettere kunne tilpasse seg statistikken til de ulike rammene, samt at hver ramme kan dekodes uavhengig av de andre rammene.

Den foreslåtte koderen benytter initialisering med hensyn på kvantiseringstrinnstørrelsen Δ, samt at den skiller mellom de ulike rammene ut fra den temporale dekomposisjonen. Initialiseringen mhp. kvantiseringstrinnet gjøres ved at $SliceQP$ i ligningen i figur 3.6 erstattes med min$(52\ , 52 * \frac{\Delta}{10})$. Dette gjøres for å tilpasse verdiene i ligningen til kvantiseringstrinnstørrelsene som er aktuelle for den foreslåtte koderen. Initialiseringen mhp. rammetypen skjer ved at koderen benytter ulike sett av parameterene (μ_γ, ν_γ) for den første rammen ut fra den temporale dekomposisjonen og de resterende rammene. Bakgrunnen til dette er at den første rammen vil inneholde gjennomsnittsintensiteten til samtlige rammer i bufferet, noe som medfører at rammens statistikk skiller seg fra de andre rammenes statistikk etter den temporale dekomposisjonen. Den første rammen initialiseres derfor med de samme parameterene (μ_γ, ν_γ) som benyttes for kodingen av I-*slices* i *H.264/AVC*, mens de resterende rammene benytter parametere tilpasset koding av P-*slices*.

4.6.2 Zig-zag skan

Den foreslåtte koderen benytter en zig-zag skan for å avbilde et todimensjonalt signal til et endimensjonalt signal, som vist i figur 4.8. Dette gjøres for å utnytte effektkorrelasjonen



Figur 4.8: Zig-zag skan benyttet i den foreslåtte koderen. Skanningsmønsteret tar alle pikslene med samme lokasjon innad i hvert delbånd og samler dem til ett endimensjonalt signal kalt en makroblokk. Merk at koeffisientene fra lavpassdelbåndet ikke tas med, slik at hver makroblokk totalt har 63 koeffisienter.

mellom pikslene i de ulike delbåndene, samt faktumet at de fleste effektspekter er synkende funksjoner, noe som medfører at amplitudene er lavere i de høyfrekvente delbåndene.

Det endimensjonale signalet kalles for en makroblokk i koderen. Makroblokken inneholder i tillegg til de skannede koeffisientene, et makroblokkflagg som signaliserer om den inneholder signifikante koeffisienter. Dette flagget settes til '1' dersom makroblokken har koeffisienter ulik null, og '0' dersom makroblokken ikke har signifikante koeffisienter.

4.6.3 Binærisering

Den første operasjonen som utføres i CABAC rammeverket er å gjøre alle ikke-binære elementer om til en binstreng vha. binærisering. Makroblokkflagget er allerede på binærform, og behøver derfor ingen binærisering. Absoluttverdiene til de kvantiserte koeffisientene i hver makroblokk binæriseres vha. en sammensetning av binæriseringsmetodene *TU* og *EGk*, kalt *UEGk*. *TU* prefiksdelen er definert for $0 \leq x \leq 14$, hvor x tilsvarer absoluttverdien til koeffisienten minus en (dette gjøre pga. kodingen av signifikanskartet som forteller om koeffisienten er ulik null, som beskrevet i del 4.6.6). For absoluttverdier ≥ 15, benyttes det i tillegg til *TU* prefiksen en suffiks bestående av et *EG0* kodeord for absoluttverdien minus 15. Tabell 4.1 viser hvordan absoluttverdiene til koeffisientene blir binærisert.

4.6.4 Valg av kontekstmodell

Indeksen γ til kontekstmodellen som skal benyttes for å kode et gitt syntakselement i CABAC, kan beregnes utifra følgende ligning:

$$\gamma = \Gamma_S + \chi_S, \tag{4.1}$$

abs_level	Bin streng																		
	TU prefiks														EG0 suffiks				
1	0																		
2	1	0																	
3	1	1	0																
4	1	1	1	0															
5	1	1	1	1	0														
...	...	...	...	...	...	...													
...	...	...	...	...	...	...	...	...	...	...	...	...							
13	1	1	1	1	1	1	1	1	1	1	1	1	0						
14	1	1	1	1	1	1	1	1	1	1	1	1	1	0					
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0				
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0		
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1		
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
bin	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19 ...

Tabell 4.1: *UEG0* binærisering av absoluttverdiene til de kvantiserte koeffisientene. Merk at binæriseringen utføres på absoluttverdien minus en. Tabellen er basert på en skisse i [4]

hvor Γ_S er en nedre grense til kontekstindeksen for den aktuelle typen syntakselement (angitt i [4, Tabell II]), mens χ_S er et kontekstindekstillegg, som bestemmes for det aktuelle syntakselementet.

4.6.5 Koding av makroblokkflagg

Makroblokkflagget indikerer om makroblokken har signifikante koeffisienter (se del 4.6.2), og grunnet effektkorrelasjonen mellom nabopiksler innad i hvert delbånd (intrabånd effektkorrelasjon), vil det eksistere korrelasjon mellom makroblokkflaggene til nabomakroblokker. Dette utnyttes i CABAC ved at kontekstindekstillegget for kodingen av makroblokkflagget til makroblokk C beregnes utifra:

$$\chi_{MBflagg(C)} = !MBflagg(A) + !MBflagg(B) \tag{4.2}$$

hvor !-tegnet betyr invertert verdi og hvor figur 3.2 viser sammenhengen mellom den romlige plasseringen til makroblokkene A,B og C. Kontekstindekstillegget blir lik '0' dersom begge nabomakroblokkene er signifikante, '1' dersom en av nabomakroblokkene ikke har signifikante koeffisienter, og '2' dersom begge ikke er signifikante.

4.6.6 Koding av transformkoeffisienter

Kodingen av transformkoeffisientene i de signifikante makroblokkene skjer i en tostegs operasjon med følgende elementer:

- Koding av et signifikanskart, som forteller hvilke koeffisienter som er ulik null

- Koding av absoluttverdier og fortegn på de signifikante koeffisientene.

Koding av signifikanskart

Signifikanskartet er et binært kart som forteller hvilke koeffisienter i makroblokken som er ulik null. Hver koeffisient i makroblokken markeres med et "signifikant koeffisientflagg" (*sig_koeff_flagg*), og dersom en koeffisient er signifikant, benyttes det et "siste signifikante koeffisient flagg" (*siste_sig_koeff_flagg*), som markerer om koeffisienten er den siste signifikante i makroblokken. Et eksempel på signifikanskartet er vist i tabell 4.2

Posisjon i skanningsmønsteret	1	2	3	4	5	6	7	9	...
Verdi på transformkoeffisient	18	-13	0	7	2	0	4	0	...
sig_koeff_flagg	1	1	0	1	1	0	1		
siste_sig_koeff_flagg	0	0		0	0		1		

Tabell 4.2: Signifikanskart bestående av ett-bits symbol for signifikante koeffisienter (*sig_koeff_flagg*), samt et ett-bits symbol for å markerer om koeffisienten er den siste signifikante i makroblokken (*siste_sig_koeff_flagg*).

Kontekstmodellene som benyttes for å kode *sig_koeff_flagg* og *siste_sig_koeff_flagg* avhenger av posisjonen til koeffisienten i makroblokken. Kontekstindekstillegget til en koeffisient i posisjonen k er gitt ved:

$$\begin{aligned} \chi_{signifikant}(\text{koeffisient}\,[k]) &= \text{PosisjonKontekstSig}\,[k] \\ \chi_{siste}(\text{koeffisient}\,[k]) &= \text{PosisjonKontekstSiste}\,[k]\,, \end{aligned} \tag{4.3}$$

hvor PosisjonKontekstSig[k] og PosisjonKontekstSiste[k] er vist i tabell 4.3. Totalt benyttes det 14 kontekstmodeller for koding av *sig_koeff_flagg*, og 8 kontekstmodeller for koding av *siste_sig_koeff_flagg*.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
PosKontekstSig[k]	1	2	3	4	5	5	4	4	3	3	4	4	4	5	5	4	4	4	4	3	3
PosKontekstSiste[k]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2

k	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
PosKontekstSig[k]	6	7	7	7	8	9	10	9	8	7	7	6	11	12	13	11	6	7	8	9	14
PosKontekstSiste[k]	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4

k	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
PosKontekstSig[k]	10	9	8	6	11	12	13	11	6	9	14	10	9	11	12	13	11	14	10	12	14
PosKontekstSiste[k]	4	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8

Tabell 4.3: Tabellen viser kontekstindekstillegget $\chi_{signifikant}(\text{koeffisient}\,[k]$ for kodingen av *sig_koeff_flagg*, samt kontekstindekstillegget $\chi_{siste}(\text{koeffisient}\,[k]$ for kodingen av *siste_sig_koeff_flagg*. Begge tilleggene er avhengig av koeffisentens posisjon k

Koding av koeffisientenes absoluttverdi og fortegn

Koeffisientenes absoluttverdi binæriserers med *UEG0*, som vist i tabell 4.1. Deretter kodes de signifikante koeffisientene i reversert rekkefølge (de siste signifikante først), noe som resulte-

rer i at de første absoluttverdiene som kodes har større sannsynlighet for å være lik 1 (pga. kilders naturlige fallende effektspekter). CABAC utnytter dette ved å benytte to sett av kontekstmodeller, et sett for det første binelementet i binstrengen (binindeks 1, som er markert med mørk grå i tabell 4.1) og et sett for resten av binelementene i *TU* prefikset av kodeordet (binindeks 2-14).

Valg av kontekstmodell for det første binelementet i binstrengen baserer seg på antallet tidligere kodede elementer med absoluttverdi lik 1, samt om antallet kodede elementer med absoluttverdi større enn 1. Gitt $NumEn\,[k]$, tilsvarende antallet kodede elementer i makroblokken med absoluttverdi lik 1, og $NumGr\,[k]$, tilsvarende antallet kodede elementer i makroblokken med absoluttverdi større enn 1, kan kontekstindekstillegget til det første binelementet til koeffisienten i posisjon k beregnes ved:

$$\chi_{ABS}\,[k, \text{binindeks} = 1] = \begin{cases} 4, & NumGr\,[k] > 0 \\ \min(3,\ NumEn\,[k]\ ), & \text{ellers} \end{cases} \tag{4.4}$$

Kontekstmodellen for koding av binelementene med indeks 2-14 (markert med lys grå i tabell 4.1) velges utifra antallet koeffisienter med absoluttverdi større enn 1, $NumGr\,[k]$. Kontekstindekstillegget for koeffisient med posisjon k beregnes ved:

$$\chi_{ABS}\,[k, \text{binindeks} = 2 - 14] = 5 + \min(4, NumGr\,[k]) \tag{4.5}$$

Begge formlene for kontekstindekstillegget er basert på effektkorrelasjonen mellom koeffisientene makroblokken og antagelsen om at kilden har et fallende effektspekter.

Koeffisienter med absoluttverdi større enn 14 får suffiksdelen av kodeordet (binindeks ≥ 15, tilsvarende *EG0* delen av kodeordet) kodet i bypassmodus. Denne modusen benyttes også for kodingen av fortegnet til samtlige signifikante koeffisienter.

Kapittel 5

Resultater

Den foreslåtte koderen har blitt testet med sekvensene *news* og *foreman*, hvor den ene sekvensen er en veldig rolig sekvens med lite bevegelser, og den andre er en sekvens bestående av mye bevegelser, både av objekter og kamerabevegelser. Begge sekvensene er i *CIF* oppløsning, tilsvarende 352*288 piksel pr ramme og 30 rammer pr sekund. Kodingen av testsekvensene har kun skjedd på luminansdelen av signalet.

Resultatkapittelet er delt i tre hoveddeler, først optimaliseres den foreslåtte koderen med hensyn på de ulike variablene bufferstørrelse og dødsone, deretter sammenlignes den mot en tilsvarende 3D-koder uten CABAC. Til slutt sammenlignes koderen mot den siste koderen fra ITU-T/ISO/IEC, *H.264/AVC*.

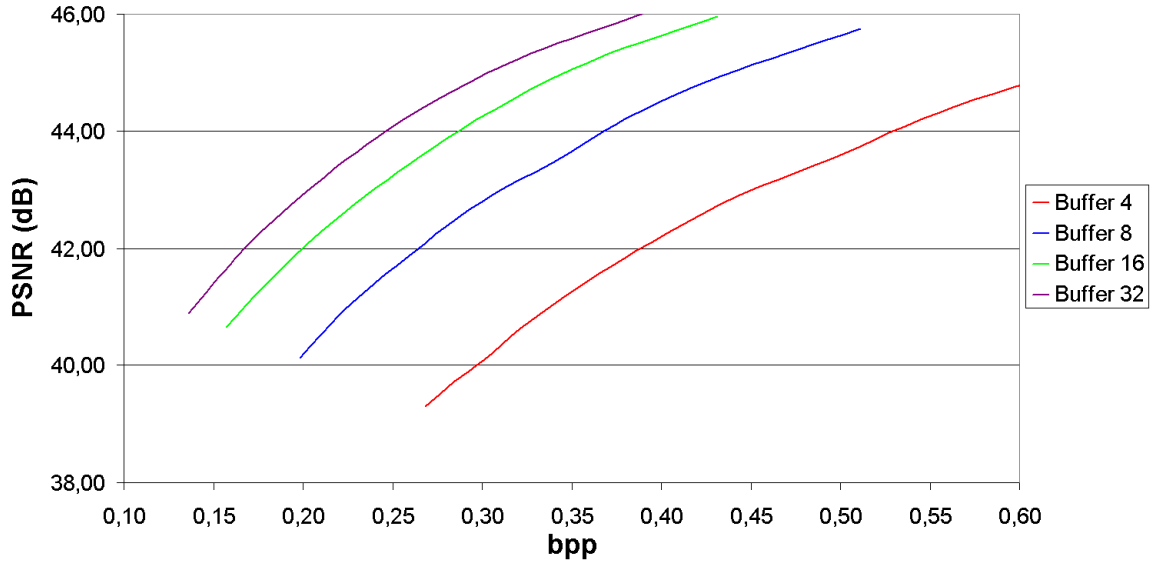
5.1 Optimalisering av 3D-koderen

5.1.1 Bufferstørrelse

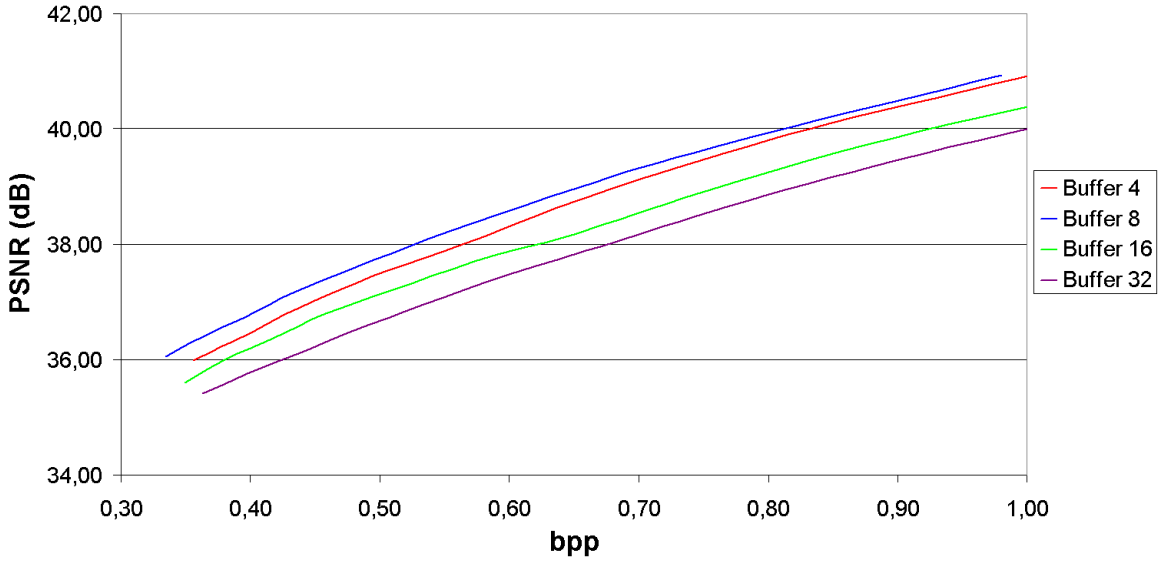
Bufferstørrelsen angir antallet rammer den temporale dekomposisjonen utføres på. Ved å øke bufferstørrelsen vil den temporale redundansen fjernes mellom flere rammer, gitt at det finnes en temporal korrelasjon mellom samtlige av de påfølgende rammene. Sekvenser med få bevegelser vil dermed kunne kodes mer effektivt dersom bufferstørrelsen økes. Problemene med temporal dekomposisjon over mange rammer er at minnekapasiteten i koderen og dekoderen må økes tilsvarende, samt at forsinkelsen gjennom systemet øker. Bufferstørrelsen påvirker også antallet rammer som blir påvirket av en eventuell transmisjonsfeil.

Sekvenser hvor objektene har større bevegelser vil i motsetning til rolige sekvenser ikke kunne kodes mer effektivt når bufferstørrelsen øker. Dette skyldes at den temporale korrelasjonen mellom piksler (med samme romlige koordinater) i påfølgende rammer blir mindre når antallet rammer økes. Figur 5.1 og figur 5.2 viser sammenhengen mellom ytelse og bufferstørrelse for de to sekvensene. Figuren viser at for sekvensen *news* øker ytelsen når bufferstørrelsen øker, mens for sekvensen *foreman* vil den maksimale ytelsen oppnås ved bufferstørrelse 8. Denne bufferstørrelsen velges for videre simuleringer, som et kompromiss mellom god ytelse for begge sekvenser, samt for å begrense forsinkelsen og minnebehovet i koderen og dekoderen.

Bufferstørrelsen påvirker også typen distorsjon som innføres ved lavratekoding. Et stort buffer vil medføre mer ringing rundt objekter i bevegelse, mens en liten bufferstørrelse vil kunne gjenskape bevegelsene mer naturlig og med mindre ringing, på bekostningen av skarp-



Figur 5.1: Figuren viser sammenhengen mellom bit pr. piksel og *PSNR*, når bufferstørrelsen varierer mellom 4, 8, 16 og 32 for sekvensen *news*. Dødsonen er satt til $1.25 * \Delta$



Figur 5.2: Figuren viser *bpp* - *PSNR* kurven for sekvensen *foreman* ved ulike bufferstørrelser. Dødsonen er satt til $1.25 * \Delta$

Figur 5.3: Figuren viser ramme 17 i den dekodede *foreman* sekvensen ved bruk av bufferstørrelse 8 (til venstre) og 32 (til høyre). Begge sekvensens er kodet med 0,363 *bpp*.

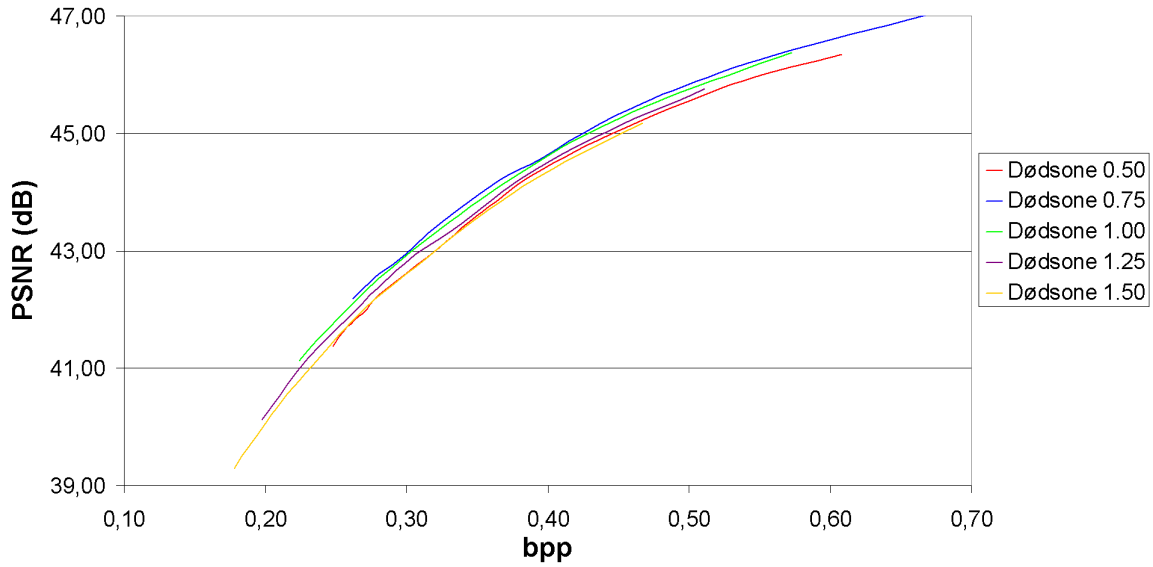
heten til de statiske objektene. Figur 5.3 viser to dekodede rammer, kodet med ulik bufferstørrelse for sekvensen *foreman*. De største forskjellene mellom sekvensene vises i områdene rundt hodet, hvor bildet til høyre har mer ringing enn det til venstre.

5.1.2 Dødsone

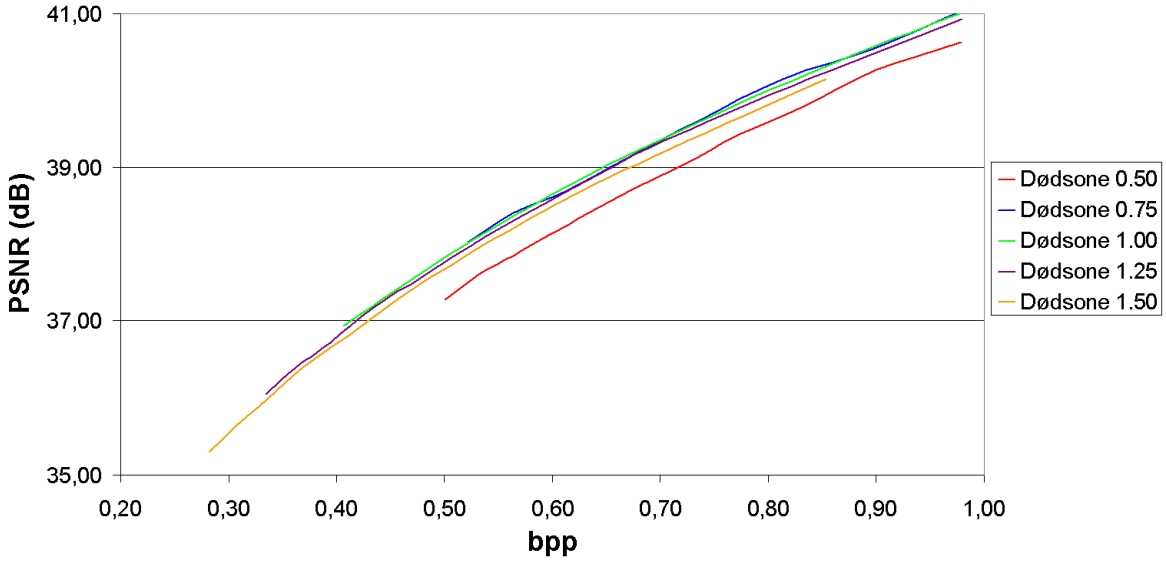
Dødsonestørrelsen angir bredden på nullområdet i kvantisereren, dvs. hvor stor del av koeffisientene som blir kvantisert til null. En dødsonestørrelse på $0.5 * \Delta$ tilsvarer at bredden på nullområdet er like stort som et vanlig kvantiseringstrinn. Figur 5.4 og figur 5.5 viser sammenhengen mellom dødsonestørrelsen og ytelsen for de to sekvensene. En bred dødsone medfører at flere koeffisienter med liten amplitude blir satt til null, samtidig som det kan benyttes mindre kvantiseringstrinn (høyere oppløsning) på de resterende koeffisientene. Figurene viser at ytelsen til koderen er ganske lik så lenge dødsonen tilsvarer $0.75\Delta - 1.25\Delta$. Videre simuleringer er gjennomført med en dødsone lik 1.25Δ. Figur 5.6 viser to dekodede bilder av sekvensen *news*, kodet med ulik dødsonenstørrelse.

5.2 Sammenligning mot 3D-koder uten CABAC

Hovedfordelen med å benytte seg av CABAC ved entropikoding, er at den har muligheten til å tilpasse seg ulike kilder, og at den kan implementeres effektivt. En tilnærmet identisk 3D-koder er benyttet for å teste ytelsen til CABAC. Referansekoderen benytter samme dekomposisjonsmetode og kvantiserer som den foreslåtte koderen, men CABAC er erstattet av adaptiv entropikoding i form av blokklassifisering basert på variansestimering, samt ulike aritmetiske kodere tilpasset sannsynlighetsfordelingen til ulike varianskategorier. Blokkstørrelsen er satt til 11*6 bildepunkter, grunnet størrelsen til hvert delbånd (44*36 piksel) etter dekomposisjonen. Den gitte blokkstørrelsen er et kompromiss mellom den totale mengden sideinformasjon som må kodes, samt ønsket om å kunne utnytte den varierende lokale statistikken.

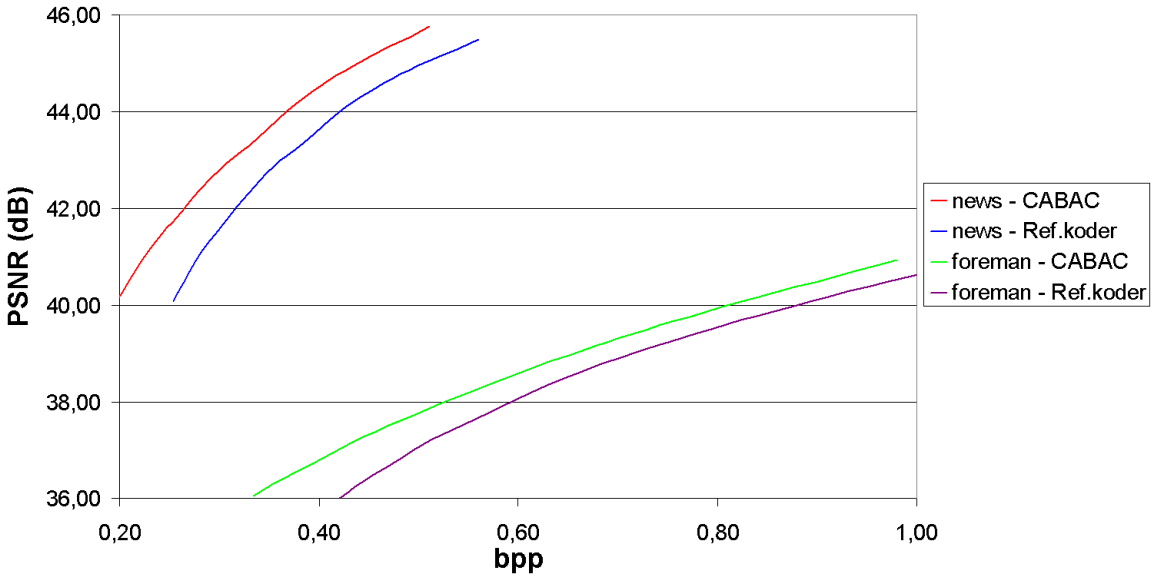


Figur 5.4: Figuren viser *bpp* - *PSNR* kurven for ulike dødsonestørrelser for sekvensen *news*. Bufferstørrelsen er satt til 8 rammer.



Figur 5.5: Figuren viser hvordan dødsonestørrelsen påvirker *bpp* - *PSNR* kurven for sekvensen *foreman*. Bufferstørrelsen er satt til 8 rammer.

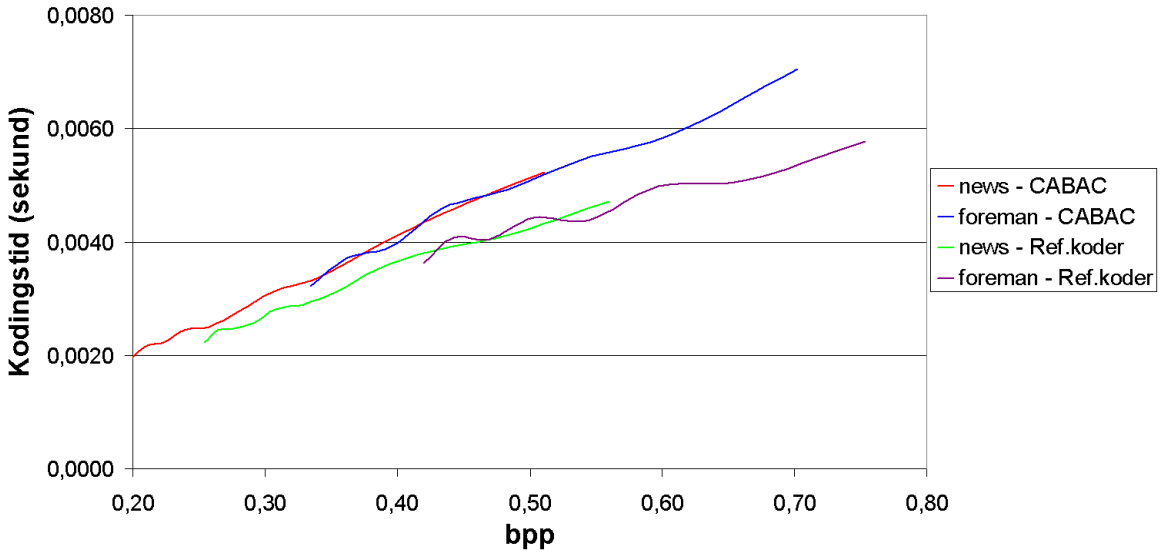
Figur 5.6: Figuren viser dekodet ramme nummer 32 i *news* sekvensen ved bruk at dødsone 1.25Δ (venstre) og 2.50Δ (høyre). Begge sekvensene er kodet med bufferstørrelse 8 og med en rate lik 0,210 *bpp*.



Figur 5.7: Figuren viser ytelsen til to 3D-kodere, hvor den ene benytter CABAC som entropikoder (merket CABAC) og den andre benytter blokklassifisering og tre ulike aritmetiske kodere (merket Ref.koder). Ytelsen er målt med bufferstørrelse 8 og dødsone lik 1.25Δ.

Variansberegningen i hver blokk benyttes for å klassifisere blokkene til en av fire kategorier. Blokkene tilhørende kategori 1, tilsvarene lavest varians, blir ikke kodet, mens de resterende blokkene, tilhørerende de andre klassifiseringskategoriene, kodes med en aritmetisk koder tilpasset inngangsfordelingen til den aktuelle kategorien. De aritmetiske koderene i referansekoderen benytter, imotsetning til CABAC, multiplikasjonsoperasjoner for å beregne delintervallene. Forskjellen mellom ytelsen til den foreslåtte koderen og referansekoderen er vist i figur 5.7. Figuren viser at CABAC gir en bitratebesparelse på 10-20% for de ulike sekvensene.

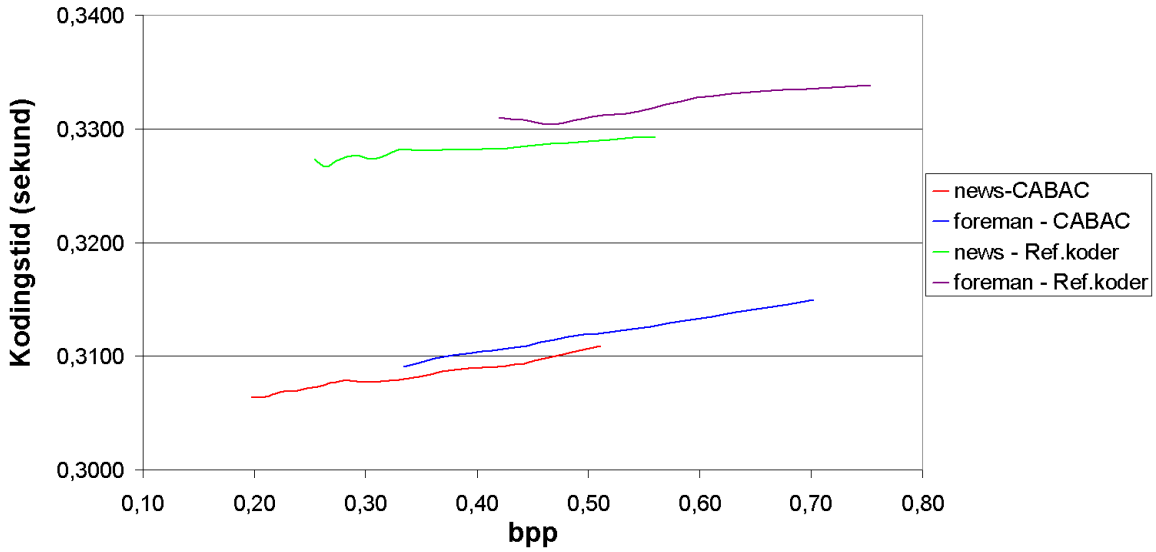
5.2.1 Effektivitet



Figur 5.8: Sammenligning mellom kjøretiden for den foreslåtte koderen og referansekoderen. Grafen er basert på kjøretidsmålinger kun for den aritmetiske kodingen, og innebærer ikke blokklassifiseringen i referansekoderen. Kjøretiden er beregnet som gjennomsnittlig kjøretid pr ramme for simuleringssekvenser av lengde 32 rammer. Gjennomsnittet er tatt av 30 simuleringer for hvert parameteroppsett.

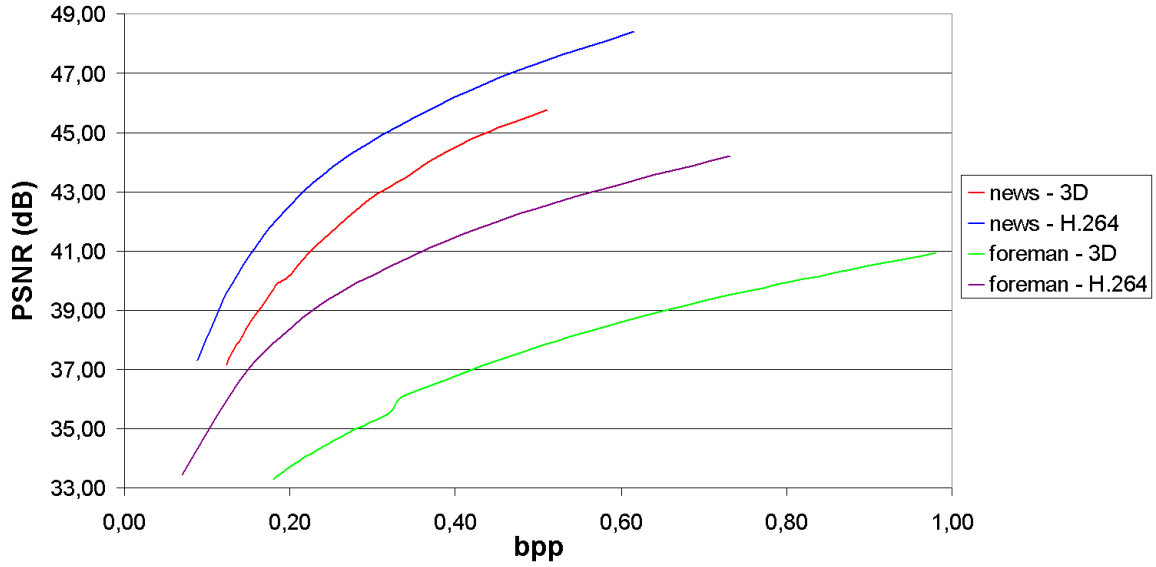
Et problem med en adaptiv entropikoder er at adapsjonen medfører økt kompleksitet i form av tilbakekoblinger samt estimeringer av betingede sannsynligheter. En indikasjon på hvordan kompleksiteten endres ved å bytte ut den aritmetiske kodingen i referansekoderen med CABAC er vist i figur 5.8. Figuren viser at CABAC kan øke kjøretiden for entropikodingen med 5-10%. Dette er en indikasjon på at kompleksiteten i CABAC er noe høyere enn ved bruk av standard aritmetiske kodere, siden kjøretiden er svært avhengig av optimaliseringsgraden til kildekoden.

Dersom hele kjøretiden betraktes for de to koderne, viser det seg derimot at den foreslåtte koderen har en gjennomsnittlig kjøretid pr ramme, tilnærmet 5-10% lavere enn referansekoderen (se figur 5.9). Dette kan skyldes at den foreslåtte koderen har en mer optimalisert kildekode, samt at referansekoderen benytter variansberegning, bestående av en rekke implementeringskostbare multiplikasjonsoperasjoner.



Figur 5.9: Sammenligning mellom den totale kjøretiden (total tid for kodingen av en ramme) for den foreslåtte koderen og referansekoderen. Den totale kjøretiden er beregnet som gjennomsnittlig kodingstid for en ramme ved kodingen av sekvenser med lengde 32 rammer. Gjennomsnittet er tatt av 30 simuleringer for hvert parameteroppsett.

5.3 Sammenligning mot H.264/AVC



Figur 5.10: Sammenligning mellom den foreslåtte koderen og *H.264/AVC*.

Ytelsen til den foreslåtte koderen sammenlignet med *H.264/AVC* er vist i figur 5.10. *H.264/AVC* koderen benytter en *GOP* størrelse 9, og med 2 B-rammer for hver I/P-ramme (tilsvarende en *GOP* bestående av følgende rammesekvens: IBBPBBPBB). Figuren viser at for

sekvensen *news* oppnår den foreslåtte koderen 1,5-2 dB lavere *PSNR* verdier enn *H.264/AVC*, mens for sekvensen *foreman* oppnår den foreslåtte koderen *PSNR* verdier tilsvarende 4-5 dB lavere enn *H.264/AVC*.

Den visuelle forskjellen mellom de to koderne er vist i figur 5.11 og 5.12. De største ulikhetene vises i de områdene med mye bevegelser, hvor det tydelig fremkommer ringingeffekt i 3D-koderen.

Figur 5.11: Figuren viser ramme nummer 8 i sekvensen *foreman*, hvor det venstre bildet er kodet med 3D-koderen, og det høyre er kodet med H.264/AVC. Begge bildene inngår i sekvenser kodet med 0,188 *bpp*.

Figur 5.12: Visuell sammenligning mellom 3D-koder og H.264/AVC. Bildet viser ramme nummer 8 i sekvensen *news*, hvor bildet til venstre er kodet med 3D-koderen og bildet til høyre er kodet med H.264/AVC. Begge bildene inngår i sekvenser kodet med 0,210 *bpp*.

Kapittel 6

Diskusjon

Videokompresjon har som formål å komprimere det orginale signalet mest mulig, samtidig som brukerens visuelle opplevelse påvirkes minst mulig. Ulike brukere vil ha forskjellig oppfatning av den visuelle kvaliteten til et degradert signal, noe som gjør optimalisering og sammenligning mellom kodere til et komplekst tema. Optimaliseringen av den foreslåtte koderen er basert på målte *PSNR* verdier. Hovedårsaken til dette er at det er et enkelt sammenligningskriterie, selv om det ikke nødvendigvis optimaliserer den visuelle opplevelsen.

3D-videokoding har en fordel fremfor hybridkodere ved at den rekursive strukturen unngås. På denne måten unngås forplantningen av eventuelle transmisjonsfeil. Fordelen med en 3D-videokoder uten bevegelseskompensasjon er den lave kompleksiteten i den temporale dekomposisjonen, samt at dette muliggjør bruk av en ikke-blokkbasert romlig dekomposisjon. En slik oppbygging medfører at kompleksiteten i koderen ikke behøver å overgå kompleksiteten i dekoderen, slik tilfellet er dersom bevegelseestimering benyttes i koderen. Problemet ved å ikke benytte bevegelseskompensasjon i den temporale dekomposisjonen er måten bevegelser gjenskapes i det dekodede signalet, samt at koderens ytelse vil være veldig avhengig av kildens innhold.

Den foreslåtte koderens ytelse er avhengig av hvilken sekvens som skal kodes, samt koderens inngangsparameter. Ytelsen ved koding av *news* sekvensen vil være avhengig av bufferstørrelsen, og generelt øke når bufferstørrelsen øker (som vist i figur 5.1). Dette skyldes at de påfølgende rammene har høy temporal korrelasjon (få bevegelser), noe som utnyttes best ved å utføre *DCT* dekomposisjonen over mange rammer. Ytelsen ved kodingen av sekvensen *foreman* vil ikke øke dersom bufferstørrelsen økes, grunnet at sekvensen inneholder mye bevegelser, og at disse bevegelsene ikke er ensformige (lik bevegelse fra ramme til ramme) over lengre perioder. Dødsonestørrelsen spiller inn ved at den utvider bredden på nullområdet, noe som viser seg å øke ytelsen så lenge dødsonen ikke blir for stor.

Simuleringene viser at implementeringen av CABAC medfører en bitratebesparelse på 10-20% i forhold til blokkbasert aritmetisk koding. Dette skyldes i hovedsak at CABAC benytter flere kontekstmodeller, samt at intra- og intereffektkorrelasjonen utnyttes bedre. En annen årsak til denne bitratebesparelsen skyldes at CABAC benytter adaptive kontekstmodeller, noe som gjør koderen i stand til å tilpasse seg kildens innhold, samt dens temporalt og romlige varierende lokale statistikk. Bruken av flere adaptive kontekstmodeller medfører at kompleksiteten i CABAC generelt er høyere enn for en ikke-adaptiv aritmetisk koder. Et resultat av dette er at kjøretiden for CABAC kodingen er 5-10% høyere enn kjøretiden for referansekoderen. Grunnen til at økningen ikke er større skyldes i hovedsak at den aritmetiske koderen

i CABAC er en multiplikasjonsfri modulokoder, samt at bypassmodusen benyttes for enkelte element.

Sammenligningen mellom den forslåtte koderen og *H.264/AVC* viser at *PSNR* verdiene til den foreslåtte koderen ligger 1,5-2 dB lavere enn *H.264/AVC* for sekvensen *news*, og ca 4-5 dB lavere for sekvensen *foreman*. Den høye ytelsen for *news* sekvensen, viser at koderen fungerer godt for rolige sekvenser. Koderens ytelse for *foreman* sekvensen er også som forventet, grunnet sekvensens oppbygging, og koderens manglende bevegelseskompensasjon i den temporale dekomposisjonen.

Distorsjonen som innføres i den foreslåtte koderen er i hovedsak ringing og sløring. Begge fenomenene skyldes at høyfrekvente komponenter blir fjernet når kvantiseringstrinnet økes. Ringingeffekten kommer som et resultat av bruken av filterbank som romlig dekomposisjon, og tilsvarer blokkeffekten ved bruk av blokkvis transformdekomposisjon. Andelen av ringing og sløring har sammenheng med valget av bufferstørrelse. Et stort buffer vil gjerne medføre økt ringing rundt bevegelser, samtidig som statiske objekter kan gjenskapes med skarpere kantoverganger. Ved bruk av mindre buffer, vil bevegelsene gjenskapes mer naturlig, på bekostningen av skarpheten i andre objekter.

Kapittel 7

Konklusjon

Den foreslåtte koderen benytter en sammensetning av både enkle og svært avanserte signalbehandlingsteknikker for å komprimere videosignal med en høyest mulig kompresjonsgrad, samtidig som den visuelle degraderingen blir minst mulig. Koderen benytter en enkel temporal dekomposisjon uten bevegelseskompensasjon, noe som muliggjør bruken av en ikke-blokkbasert, 64-delbånds filterbank for romlig dekomposisjon. Fordelen med denne fremgangsmåten er at kompleksiteten i kodingsprosessen ikke behøver å overgå kompleksiteten i dekoderen, samtidig som det kan benyttes mer effektive romlige dekomposisjonsmetoder. Et problem med den valgte dekomposisjonen er at koderens ytelse blir svært avhengig av kildens innhold.

Koderens største fordel er bruken av kontekstbasert, adaptiv, binæraritmetisk koding (CABAC). Ved å benytte seg av aritmetisk koding oppnås et klart skille mellom sannsynlighetsmodelleringen og selve kodingen. Dette muliggjør bruken av ulike kontekstmodeller for ulike deler av symbolene, samt muligheten for å benytte adaptive modeller. Den foreslåtte koderen utnytter effektkorrelasjon innad i delbåndene, samt mellom delbåndene for å kode koeffisientene mest mulig effektivt. Samtidig oppdateres kontekstmodellene etter hver hendelse, noe som medfører at koderen kan tilpasse seg ulike kilders statistikk. En slik fremgangsmåte medfører at kompleksiteten, i form av kjøretid, øker med 5-10%, men gir bitratebesparelser på 10-20% i forhold til aritmetisk koding basert på blokklassifisering.

Et resultat av koderens dekomposisjonsmetode er innføringen av ringing og sløring ved lavratekoding. Ringingen vises rundt objekter i bevegelse, mens sløringen vises både på objekter i bevegelse og på statiske objekter. Begge fenomenene skyldes fjerningen av mange høyfrekvente komponenter med lav amplitude. Koderen opplever derimot ingen blokkeffekt, som er et kjent problem med hybridkodere. Den foreslåtte koderen har ingen problemer med å gjenskape bevegelser på en naturlig måte, til tross for dens manglende bevegelseskompensasjon.

Grunnet valget av dekomposisjonsmetode, påvirkes koderens ytelse av inngangssignalet. Koderen oppnår gode resultater for sekvenser med få bevegelser (*news*), mens for sekvenser hvor både objekt- og kamerabevegelser er utbredt (*foreman*), vil koderens ha lavere ytelse enn kodere som benytter bevegelsesestimering og -kompensasjon. Ved sammenligning med den siste standardiserte koderen fra ITU-T/ISO/IEC, *H.264/AVC*, viser det seg at den foreslåtte oppnår *PSNR* verdier lik 1,5-2 dB lavere for sekvensen *news*, mens den for sekvensen *foreman* oppnår *PSNR* verdier 4-5 dB lavere enn *H.264/AVC*. Tatt i betraktning at koderen ikke benytter bevegelseskompensasjon, viser disse sammenligningen at koderen yter som forventet.

7.1 Videre arbeid

Koderen utnytter inter- og intrabånd effektkorrelasjonen innad i hver dekomponerte ramme. En slik effektkorrelasjon vil også kunne eksistere i den temporale retningen, og en naturlig videreutvikling vil derfor være å se metoder for å utnytte dette. Den enkleste formen for utnyttelse vil være å benytte flere betingede sannsynligheter for koding av bl. a. makroblokkflagget.

Et annet problem med den foreslåtte koderen er at den benytter en fast sannsynlighetsmodell for *"zerotree"* kodingen av LP-LP delbåndet. Kodingen er basert på en fast Huffmantabell, og utnytter derfor ikke det faktum at symbolsannsynlighetene i det dominante kodingsløpet vil variere over tid. En videreutvikling kan implementeres ved å utnytte betingete sannsynligheter, samt benytte adaptive kontekstmodeller og aritmetisk koding. En måte å gjøre dette på er å implementere en versjon av Said og Perlmans "Set Partitioning in Hierarchical Trees" (*SPITH*) algoritme [19]. Grunnet effektkorrelasjonen i den temporale retningen vil det også være mulig å oppnå høyere ytelse dersom *SPITH* algoritmen videreutvikles til å gjelde 3D signaler.

Grunnet ringingeffekten som oppstår ved lavratekoding, vil det visuelle inntrykket kunne bedres dersom det innføres en post-prosessering i dekoderen. Dette kan gjøres ved å benytte en algoritme for å påvise områder med ringingeffekt og deretter et egnet filter for å fjerne denne uønskede effekten. Et tilsvarende blokkeffektfilter benyttes i *H.264/AVC* for å fjerne blokkeffekten som oppstår grunnet den blokkbaserte dekomposisjonen. Flere forsøk har vist lovende resultater med hensyn på filter som bevarer skarpheten, samt fjerner den uønskete ringingeffekten [20][21].

Bibliografi

[1] Jens-Rainer Ohm. Three-dimensional subband coding with motion compensation. *IEEE Transaction on Image Processing*, 3(5):559–571, September 1995.

[2] Lin Luo, Jin Li, Shipeng Li, Zhenquan Zhuang, and Ya-Qin Zhang. Motion compensated lifting wavelet and its application in video coding. *IEEE International Conference on Multimedia and Expo*, pages 365–368, August 2001.

[3] Rissanen and Langdon. Universal Modelling and Coding. *IEEE Transactions on Information Theory*, 27(1):12–23, Januar 1981.

[4] Marpe, Schwarz, and Wiegand. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, Juli 2003.

[5] Tor A. Ramstad. Digital image communication, October 2005.

[6] Gibson, Berger, Lookabaugh, Lindbergh, and Baker. *Digital Compression for Multimedia*. Morgan Kaufman Publishers, 1998.

[7] Béatrice Pesquet-Popescu and Vincent Bottreau. Three-dimensional lifting schemes for motion compensated video compression. *Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3:1793–1796, Mai 2001.

[8] J.M. Shapiro. Embedded image coding using zerotrees of wavelets coefficients. *IEEE Transactions on Signal Processing*, 41:3445–3462, Desember 1993.

[9] Øien and Lundheim. Informasjonsteori, koding og kompresjon, 2004.

[10] John Markus Lervik. *Subband image communication over digital transparent and anaolog waveform channels*. PhD thesis, NTNU, 1996.

[11] Paul G. Howard and Jeffrey Scott Vitter. Arithmetic coding for data compression. *Proceedings of the IEEE*, 82:857–865, Juni 1994.

[12] Witten, Neal, and Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, Juni 1987.

[13] Pennebaker, Mitchell, Langdon, and Arps. An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726, November 1988.

[14] ITU-T and ISO/IEC JTC-1. Advanced Video Coding for generic audiovisual services. *ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) AVC*, 2003.

[15] Moffat, Neal, and Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16:256–294, Juli 1998.

[16] Sven Ole Aase. *Image Subband Coding Artifacts: Analysis and Remedies*. PhD thesis, NTNU, 1993.

[17] Balasingham, Fuldseth, and Ramstad. On optimal tiling of the spectrum in subband image compression. *Proceedings of the 1997 International Conference on Image Processing*, 1:49–52, Oktober 1997.

[18] JVT. H.264/AVC reference software. http://iphome.hhi.de/suehring/tml.

[19] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, Juni 1996.

[20] Kong, Nie, Vetro, Sun, and Barner. Coding artifacts reduction using edge map guided adaptive and fuzzy filter. *IEEE International Conference on Multimedia and Expo*, 2:1135–1138, Juni 2004.

[21] Hao-Song Kong, Anthony Vetro, and Huifang Sun. Edge map guided adaptive post-filter for blocking and ringing artifacts removal. *Proceedings of the 2004 International Symposium on Circuits and Systems*, 3:929–932, Mai 2004.

Tillegg A

Appendiks

DVDen vedlagt rapporten inneholder følgende:

- Videokoder, Videodekoder og et program for måling av *PSNR*
- Dekodede sekvenser, kodet med forskjellige parametre
- Orginale versjoner av *news* og *foreman* sekvensene
- Program for å vise de dekodede sekvensene (*YUVViewer*)
- Excel dokument med alle simuleringer og målinger som er utført.

Testing av programmet

For å teste programmet kreves det at man har installert en C/C++ kompilator på maskinen (f.eks. *MinGW* som kan lastes ned fra *http://sourceforge.net/projects/mingw*). Kompileringen gjøres ved å benytte de respektive *makefilene* i hver mappe. Etter kompilering kan de ulike programmene kjøres.

VideoKoder

Videokoderen tar følgende innparameter:
-i Ukomprimert sekvense (av formatet .yuv)
-m Oppløsning (1=QCIF, 2=CIF, 3=SIF (240x352) og 4=512x512. default = 2)
-b Bufferstørrelse (default = 8)
-f Antall rammer
-q Kvantiseringstrinnstørrelse (default = 8)
-t Dødsone størrelse (default = 1.25)
-n Kvantiseringstrinnstørrelse for LP-LP delbåndet (default lik -q)
-l Dødsonestørrelse for LP-LP delbåndet (default lik -t)
Kodet informasjonen blir lagret i filen *CODEFILE*. Programmet beregner gjennomsnittlig bit pr. piksel, *bpp* for "*zerotree*" kodingen, *bpp* for CABAC delen, samt kjøretiden for en ramme, og skriver dette ut til brukeren

eksempel: VideoEncoder -i news-cif-ori90.yuv -b 8 -q 7 -t 1.5 -f 32 -m 2

vil kode sekvensen *news-cif-ori90.yuv* som må være i *CIF* oppløsning, med bufferstørrelse 8, kvantiseringstrinn 7, dødsone lik 1.5 (merk at total dødsone blir lik 7*1.5= 10.5). Totalt vil 32 rammer kodes.

VideoDekoder

Videodekoderen tar følgende innparameter:
- -i Filen med kodet informasjon (*CODEFILE*)
- -o Dekodet videofil i formatet .yuv

eksempel: VideoDecoder -i CODEFILE -o decoded-movie-news.yuv
vil dekode filen *CODEFILE* og lagre dekodet video i filen *decode-movie-news.yuv*.

Måling av *PSNR*

Programmet VideoPSNR tar følgende parameter:
- -c Dekomprimert videosekvens (fra dekoderen)
- -o Orginal videosekvens
- -m Oppløsning (sammen verdier som i koderen)
- -f Antall rammer den gjennomsnittlige *PSNR* verdien skal beregnes på

Programmet beregner gjennomsnittlig *PSNR* verdi og skriver dette ut til brukeren.

eksempel: VideoPSNR -o news-cif-ori90.yuv -c decoded-movie-news.yuv -m 2 -f 32
vil beregne gjennomsnittlig *PSNR* over 32 rammer for de to filene *news-cif-ori90.yuv* og *decoded-movie-news.yuv*, hvor begge filene må være i *CIF* oppløsning.