INF2310 – Digital bildebehandling

FORELESNING 16

REPETISJON – DEL I

Andreas Kleppe

Filtrering i bildedomenet
2D diskret Fourier-transform (2D DFT)
Kompresjon og koding
Morfologiske operasjoner på binære bilder

Filtrering i bildedomenet

- Anvendelsen av en **operator** som beregner ut-bildets verdi i hvert piksel *(x,y)* ved bruk av inn-bildets piksler i et **naboskap** rundt *(x,y)*.

$$g(x,y) = T(f(x,y))$$

g(x,y) angir ut-bildets verdi i *(x,y)* og *f(x,y)* angir inn-bildets piksler i et naboskap rundt *(x,y)*

Ut-bilde

Operator

Inn-bilde

1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

Et inn-bilde *f*

Filtrering

(3x3-middelverdifilter, speilende indeksering og bevart bildestørrelse)

2,7	2,9	2,7	2,1
3,1	2,9	2,4	2,2
3,3	3,0	3,0	3,3
2,6	2,2	2,9	3,7

Ut-bildet *g*

2D-konvolusjons-eksempel

Oppgave: Konvolver følgende filter og bilde:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3x3-middelverdifilter

1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

Inn-bilde *f*

Anta at bildet er 0 utenfor det definerte 4x4-området.

2D-konvolusjons-eksempel

Steg 1: Roter filteret 180 grader.

Ikke nødvendig her ettersom filteret er symmetrisk.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3x3-middelverdifilteret er symmetrisk

2D-konvolusjons-eksempel

Steg 2: Legg det roterte filteret over først posisjon der filteret og bildet overlapper.

1/9	1/9	1/9			
1/9	1/9	1/9			
1/9	1/9	1•1/9	3	2	1
		5	4	5	3
		4	1	1	2
		2	3	2	6

Inn-bilde *f*

2D-konvolusjons-eksempel

Steg 3: Multipliser filterets vekter med verdiene av de overlappende pikslene i bildet. Responsen er summen av produktene.

1/9	1/9	1/9			
1/9	1/9	1/9			
1/9	1/9	1•1/9	3	2	1
		5	4	5	3
		4	1	1	2
		2	3	2	6

Inn-bilde *f*

$1 \cdot 1/9 = 1/9 \approx 0{,}1$

0,1					

Foreløpig ut-bildet *g*

2D-konvolusjons-eksempel

Steg 4: Gjenta 3 for neste overlapp. Ikke flere: ferdig!

Steg 3: Multipliser filterets vekter med verdiene av de overlappende pikslene i bildet og summer.

1/9	1/9	1/9		
1/9	1/9	1/9		
1/9	1•1/9	3•1/9	2	1
	5	4	5	3
	4	1	1	2
	2	3	2	6

Inn-bilde *f*

$1 \cdot 1/9 + 3 \cdot 1/9$
$= 4/9 \approx 0{,}4$

0,1	0,4				

Foreløpig ut-bildet *g*

2D-konvolusjons-eksempel

… fjorten steg 3 senere:

Steg 3: Multipliser filterets vekter med verdiene av de overlappende pikslene i bildet og summer.

$3 \cdot 1/9 + 2 \cdot 1/9 + 1 \cdot 1/9 +$
$4 \cdot 1/9 + 5 \cdot 1/9 + 3 \cdot 1/9 +$
$1 \cdot 1/9 + 1 \cdot 1/9 + 2 \cdot 1/9$
$= 22/9 \approx 2{,}4$

1	3•1/9	2•1/9	1•1/9
5	4•1/9	5•1/9	3•1/9
4	1•1/9	1•1/9	2•1/9
2	3	2	6

Inn-bilde *f*

0,1	0,4	0,7	0,7	0,3	0,1
0,7	1,4	2,2	2,0	1,2	0,4
1,1	2,0	2,9	2,4		

Foreløpig ut-bildet *g*

2D-konvolusjons-eksempel

… og etter tjue steg 3 til:

Steg 4: Gjenta 3 for neste overlapp. Ikke flere: **ferdig**!

Løsningen er:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3x3-middelverdifilter

$*$

1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

Inn-bilde *f*

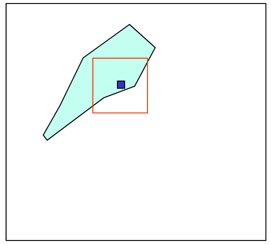
$=$

0,1	0,4	0,7	0,7	0,3	0,1
0,7	1,4	2,2	2,0	1,2	0,4
1,1	2,0	2,9	2,4	1,6	0,7
1,2	2,1	3,0	3,0	2,1	1,2
0,7	1,1	1,4	1,7	1,2	0,9
0,2	0,6	0,8	1,2	0,9	0,7

Ut-bildet *g*

Kant-bevarende støyfiltrering

- Ofte lavpassfiltrerer vi for å **fjerne støy**, men ønsker samtidig å **bevare kanter**.
- Det finnes et utall av «kantbevarende» filtre.
- Men det er et system:
  - Tenker at vi har flere piksel-populasjoner i naboskapet rundt *(x,y)*, f.eks. to:
  - Sub-optimalt å bruke all pikslene.



- Vi kan sortere pikslene:
  - Radiometrisk (etter pikselverdi)
  - Både geometrisk (etter pikselposisjon) og radiometrisk

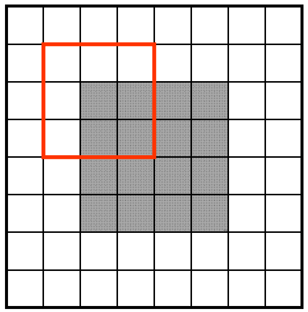
Middelverdi eller median?

Inn-bildet med tydelig salt-og-pepper-støy

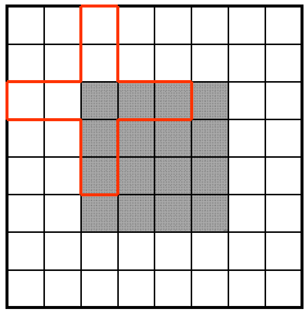
Etter middelverdifiltrering

Etter medianfiltrering

Medianfiltrering og hjørner



Med kvadratisk naboskap avrundes hjørnene



Med pluss-formet naboskap bevares hjørnene

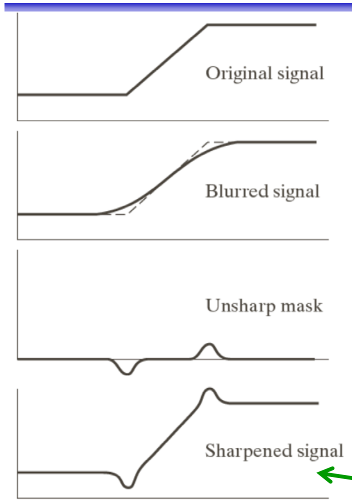
Kant-bevarende støyfiltrering

- Trimmet middelverdifilter:
  - Alpha-trimmet middelverdifilter (radiometrisk sortering):
    g(x,y) = middelverdien av de *mn-d* midterste verdiene
    (etter sortering) i *m*x*n*-naboskapet rundt *(x,y)*.
  - *K* Nearest Neighbour-filter (radiometrisk sortering):
    g(x,y) = middelverdien av de *K* pikslene i naboskapet
    rundt *(x,y)* som ligner mest på *(x,y)* i pikselverdi.
  - *K* Nearest Connected Neighbour-filter (også geometrisk):
    K Nearest Neighbour-filter med uendelig stort naboskap
    og der de valgte pikslene er tilkoblet ut-posisjonen *(x,y)*.
  - Max-homogenitet-filter (også geometrisk):
    g(x,y) = middelverdien av det mest homogene sub-naboskapet.
  - Symmetrisk nærmeste nabo-filter (også geometrisk):
    g(x,y) = middelverdien av de mest lignende fra
    hvert symmetrisk piksel-par rundt *(x,y)*.
- MinimalMeanSquareError-filter: Nær middelverdien når lokal varians tilsvarer anslått støyvarians (en parameter), ellers nær uendret.

Høypassfiltre

- Slipper gjennom høye frekvenser,
  og demper eller fjerner lave frekvenser.
  - Typisk fjernes den aller laveste frekvensen helt,
    d.v.s. at homogene områder får ut-verdi 0.
- Effekt:
  - Demper langsomme variasjoner, f.eks. bakgrunn.
  - Fremhever skarpe kanter, linjer og detaljer.
- Typiske mål: «Forbedre» skarpheten, detektere kanter.
- Q: Hva skjer med støy?

Unsharp masking og highboost-filtrering

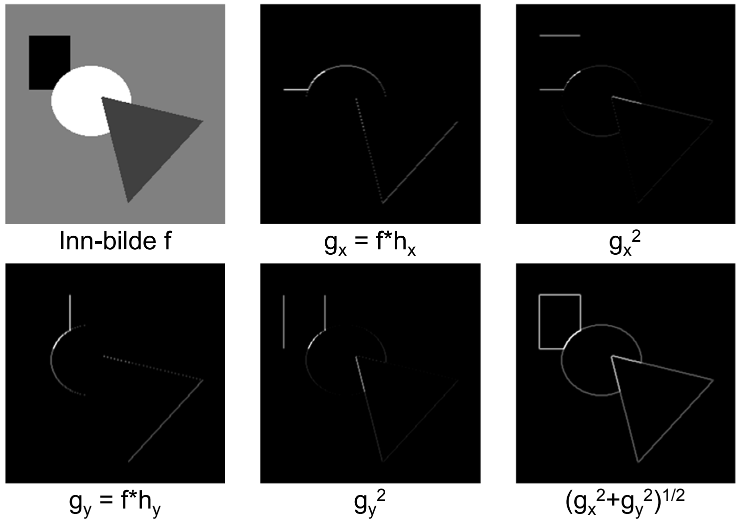


G&W fig. 3.39

- Gitt et bilde (original):
  (til venstre: et 1D-bilde av en rampe)

1. Lavpassfiltrer.
   (til høyre er originalen stiplet)
2. Beregn differansen:
   original – filtrering
3. Resultatet er:
   original + k • differansen
   - *k* er en positiv konstant.
   - **Unsharp masking**: *k* = 1
   - **Highboost-filtrering**: *k* > 1

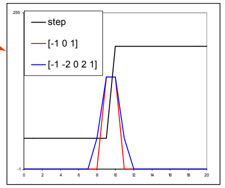
Eksempel: Gradient-beregning med Sobel-operatoren

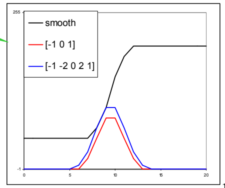


Merk:
Hvert bilde er skalert ved å dele på sitt maksimum.
De negative verdiene i g_x og g_y er satt til 0.

Gradient til kant-deteksjon

- Gradient-magnituden har «bred respons», men vi ønsker eksakt, tynn kant.
- For en steg-kant:
  - Bredden på responsen er avhengig av størrelsen på filteret.
- For en bred kant (glattet med [1 2 3 2 1]):
  - Bredden på responsen er avhengig av bredden på kanten.





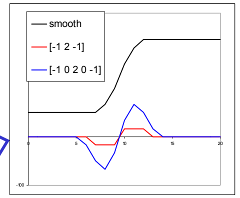
- **Maksimumet er likt og fornuftig lokalisert!**
  - Bruke den andrederiverte til å finne maksimumene?

Laplace-operatoren

- Laplace-operatoren er gitt ved:

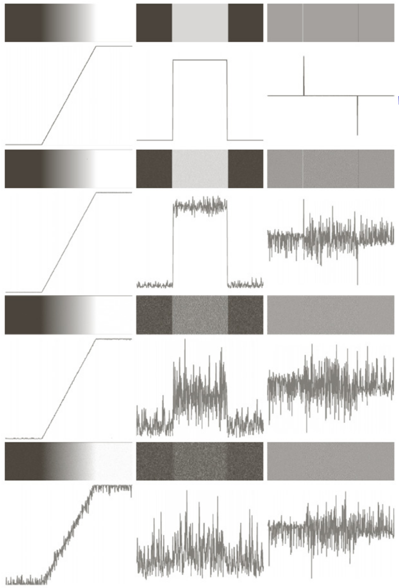
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

  - Den endrer fortegn der f et vendepunkt.
  - $\nabla^2 f = 0$ markerer kant-posisjon.
  - $|\nabla^2 f|$ har to ekstremverdier per kant; på starten og på slutten av kanten.
    - Derfor brukte vi den tidligere til å forbedre bildeskarpheten!



- Kantens eksakte posisjon er **nullgjennomgangen**.
- Dette gir tynne kanter.
- Vi finner bare kant-posisjoner, ikke kant-retninger.

Også lavpassfiltrere?

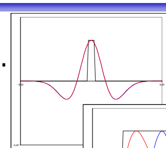


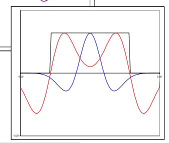
Signal Derivert Laplace

FIGURE 10.11 First column: Images and intensity profiles of a ramp edge corrupted by random Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

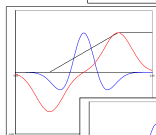
Kantdeteksjon ved LoG-nullgjennomganger

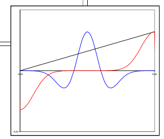
- Tommelfingerregel for strukturer:
  LoG-kjernen må være smalere enn strukturen.
  - Strukturen er mindre enn halvparten av LoG-kjernen => Nullgjennomgangene er utenfor kantskillene
  - Strukturen er større enn halvparten av LoG-filteret => Nullgjennomgangene er nøyaktig kantskillene
  - Et sted i mellom: Avhenger av diskretiseringen og tilnærmingen av LoG-filteret.





- Tommelfingerregel for ramper:
  LoG-filteret må være større enn rampen.
  - Rampen er bredere enn LoG-filteret, => Ingen nullgjennomgang, bare et null-platå.
  - Ellers: Nullgjennomgang midt på rampen (kan få én 0-respons akkurat på midten), altså en fornuftig definisjon av kantskillet til rampen.
    - P.g.a. støy krever ofte at nullpasseringen er skarp => LoG-filteret må være betydelig større enn rampen.





- => **Velg kjerne- og filterstørrelsen med omhu**!
  - Angis først og fremst av standardavviket til Gauss-funksjonen, som gir bredden av LoG-kjernen og antyder størrelsen av LoG-filteret.

Ideen til Canny

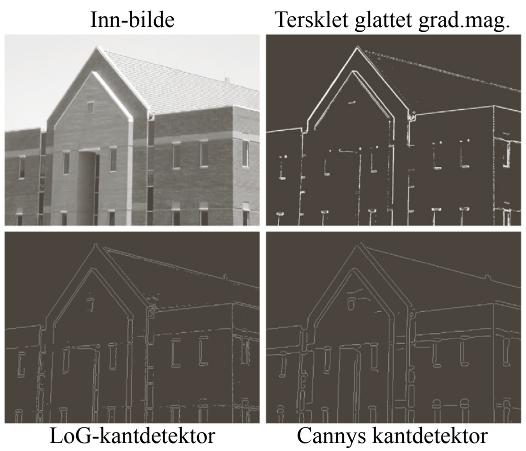
- Lag en kantdetektor som er optimal i forhold til følgende tre kriterier:
  - Best mulig deteksjon (alle kanter og bare kanter)
  - God kant-lokalisering
  - Én enkelt respons
- Optimer ved bruk av et bilde med støy.
- Resultat: Følgende enkle algoritme oppnår nesten optimumet:

Cannys algoritme

1. Lavpassfiltrer med Gauss-filter (med gitt σ).
2. Finn gradient-magnituden og gradient-retningen.
3. Tynning av gradient-magnitude ortogonalt på kant.
   - F.eks.: Hvis et piksel i gradient-magnitude-bildet har en 8-nabo i eller mot gradient-retningen med høyere verdi, så settes pikselverdien til 0.
4. Hysterese-terskling (to terskler, T_h og T_l) :
   a. Merk alle piksler der $g(x,y) \geq T_h$
   b. For alle piksler der $g(x,y) \in [T_l, T_h)$:
      - Hvis (4 eller 8)-nabo til et merket piksel, så merkes dette pikselet også.
   c. Gjenta fra trinn b til konvergens.

Eksempel: Kantdeteksjon

- **Oppgave**: Finn fremtredende kanter.

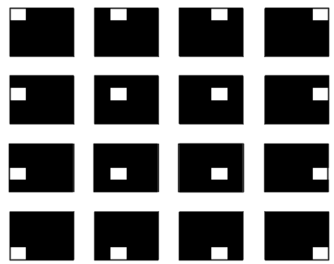


a b
c d

FIGURE 10.25 (a) Original image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1]. (b) Thresholded gradient of smoothed image. (c) Image obtained using the Marr-Hildreth algorithm. (d) Image obtained using the Canny algorithm. Note the significant improvement of the Canny image compared to the other two.

Standardbasis for matriser

Eksempel: Standardbasis for 4x4



- Et gråtonebilde representeres vanligvis som en matrise av gråtoneintensiteter.
- Dette tilsvarer å bruke den såkalte *standardbasisen* for matriser.
- Eksempel: 4x4-gråtonebilder
  - Standardbasisen er de 16 4x4-matrisene vist til venstre.
  - En vektet sum av disse matrisene kan unikt representere enhver 4x4-matrise/gråtonebilde.

Undereksempel:

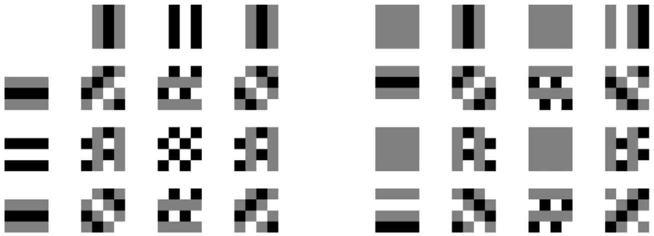
1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

= 1*  + 3*  +…+ 6* 

I bildene er sort 0 og hvitt 1.

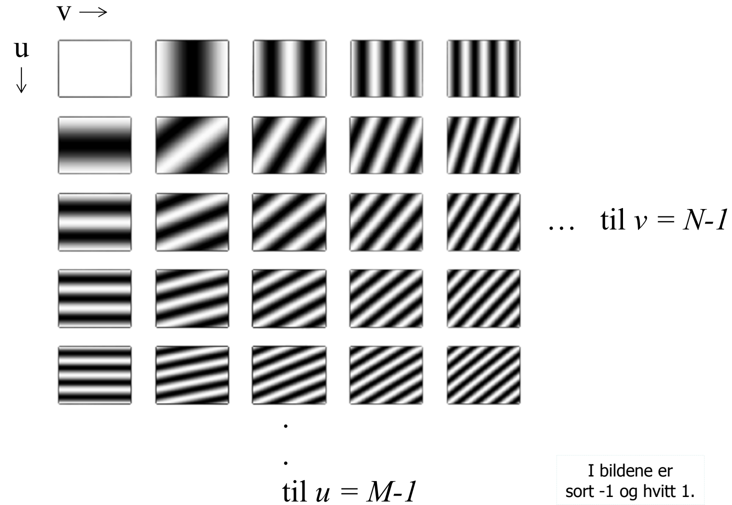
Alternativ basis

- Det finnes mange andre basiser for matriser.
  - Muligheten til å unikt representere enhver matrise ligger i *basis*.
- **2D DFT** bruker én slik basis som er basert på **sinuser og cosinuser med forskjellige frekvenser**.
  - Disse sinusene og cosinusene er faste for en gitt bildestørrelsen (MxN) og kan representeres som hver sin mengde av bilder:



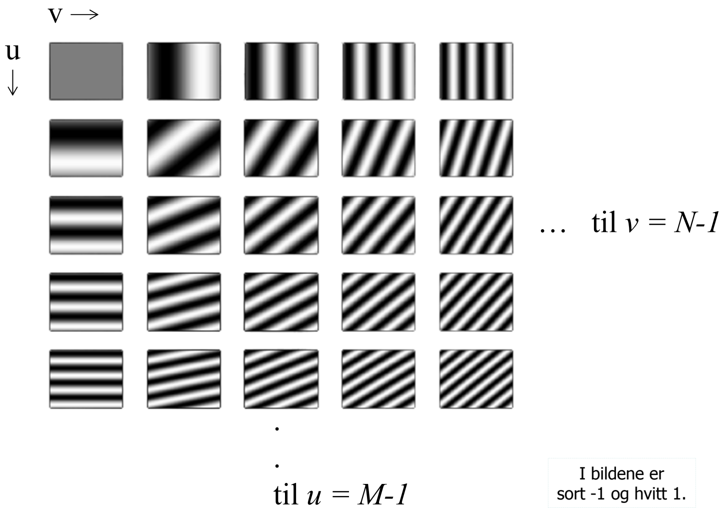
(i bildene er sort -1, grått er 0 og hvitt er 1)

Cosinus-bilder for større bilder



I bildene er sort -1 og hvitt 1.

Sinus-bilder for større bilder

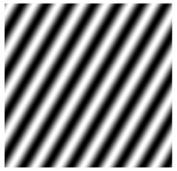


I bildene er sort -1 og hvitt 1.

Beregning av 2D DFT for en gitt frekvens

- Koeffisienten til 2D DFT av tidligere eksempelbilde for frekvens (0,1):
- real(F(0,1)) = sum($\begin{bmatrix} 1 & 3 & 2 & 1 \\ 5 & 4 & 5 & 3 \\ 4 & 1 & 1 & 2 \\ 2 & 3 & 2 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}$) = 2
- imag(F(0,1)) = sum($\begin{bmatrix} 1 & 3 & 2 & 1 \\ 5 & 4 & 5 & 3 \\ 4 & 1 & 1 & 2 \\ 2 & 3 & 2 & 6 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & 1 \end{bmatrix}$) = 1
- Altså er *F(0,1) = 2+j*.

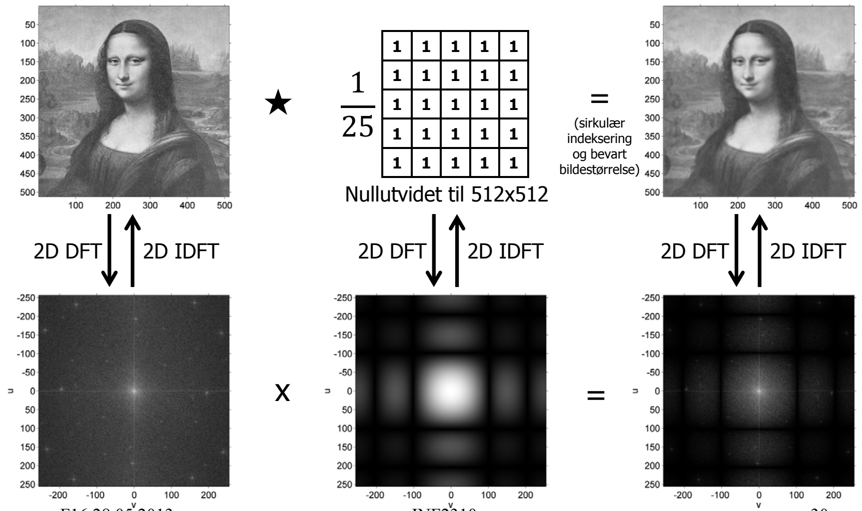
Grunnleggende om 2D DFT

- real(F(u,v)) = sum( x )

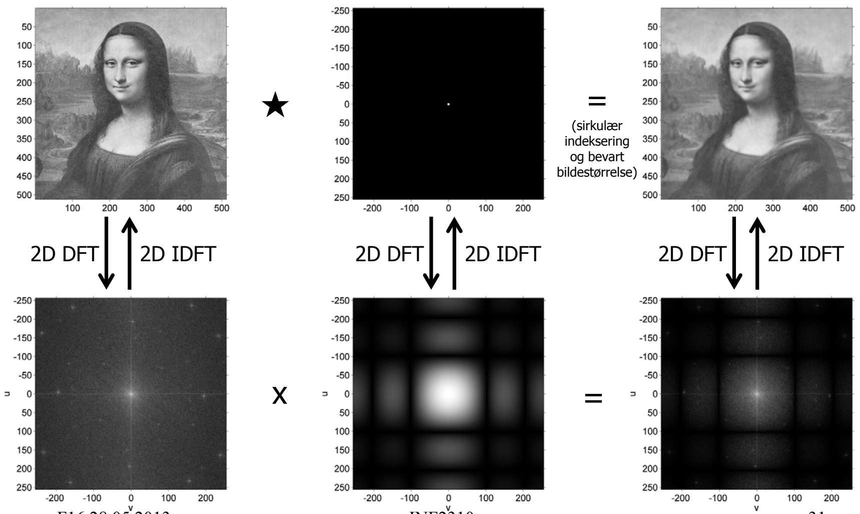
  realdelen til 2D DFT-en i frekvens (u,v) = sum(bildet punkt-multiplisert cosinus-bildet for frekvens (u,v))

- Tilsvarende for imaginærdelen og sinus-bildet.
- **Hvert punkt** i 2D DFT-en beskriver altså noe ved **hele bildet**.

Eksempel: 5x5-middelverdifiltrering og konvolusjonsteoremet



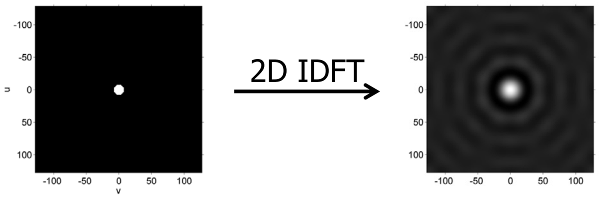
Eksempel: 5x5-middelverdifiltrering og konvolusjonsteoremet

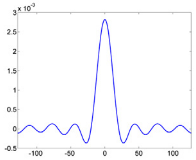


Anvendelse av konvolusjonsteoremet

- Design av konvolusjonsfiltre med bestemte frekvensegenskaper.
  - Designe konvolusjonsfilteret i Fourier-domenet slik at vi har bedre kontroll på dets frekvensegenskaper.
- Analyse av konvolusjonsfiltre.
  - 2D DFT-en til et konvolusjonsfilter gir oss innblikk i hvordan filteret vil påvirke de forskjellig frekvenskomponentene.
- Rask implementasjon av større konvolusjonsfiltre.

Filter-design i Fourier-domenet
Romlig representasjon av ideelt lavpassfilter

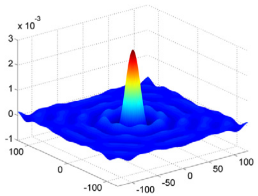




Den romlige representasjonen er en trunkert sinc-funksjon. Sinc-funksjonen er definert som:

$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

og sinc(0) = 1.



- Vi får *ringing* i bildet.
  - Husk også tommelfingerregel fra forrige forelesning: Smal/bred struktur i bildet ⇔ Bred/smal struktur i Fourier-spekteret

Eksempel: Ideelt lavpassfilter

Original

Filtrert med D_0=0.2min{M,N}/2

Filtrert med D_0=0.3min{M,N}/2

I god nok oppløsning kan striper/ringinger sees ut fra markante kanter i de to filtrerte bildene. Det er dette vi kaller *ringing*.

Gaussisk lavpassfilter

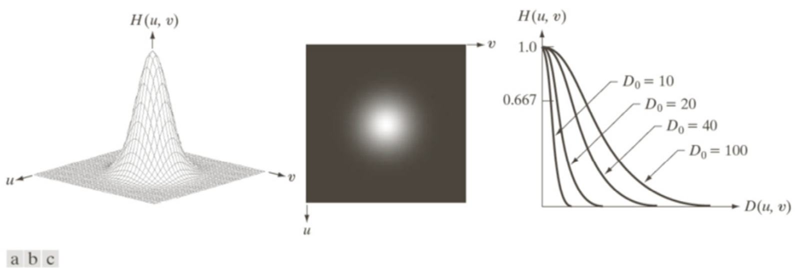


FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

Husk tommelfingerregelen:
Smal/bred struktur i bildet ⇔ Bred/smal struktur i Fourier-spekteret

Butterworth lavpassfilter

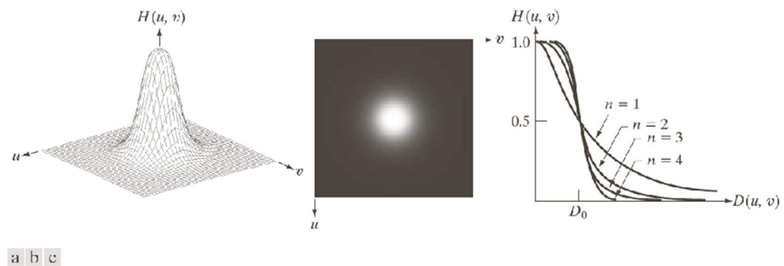


FIGURE 4.44 (a) Perspective plot of a Butterworth lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Eksempel: Butterworth lavpassfilter

n=11

n=41

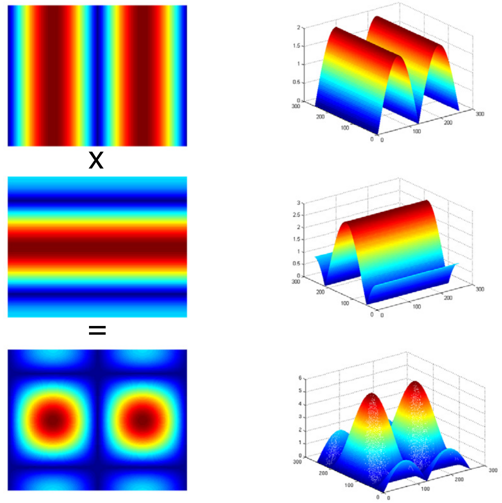
n=61

D_0 = 0.2min{M,N}/2 i alle filtreringene.

Analyse av filtre

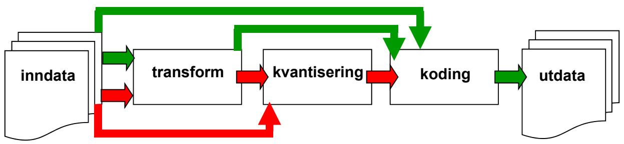
h_y i Prewitt-operatoren (en gradientoperator)

$$\begin{bmatrix}1 & 0 & -1\end{bmatrix} \star \begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\1 & 0 & -1\\1 & 0 & -1\end{bmatrix}$$



Kompresjon

- Kompresjon kan deles inn i tre steg:
  - **Transform** - representer bildet mer kompakt.
  - **Kvantisering** - avrunding av representasjonen.
  - **Koding** - produksjon og bruk av kodebok.



- Kompresjon kan gjøres:
  - **Eksakt / tapsfri** (eng.: *lossless*) – følg de grønne pilene.
    - Her kan vi rekonstruere den originale bildet eksakt.
  - **Ikke-tapsfri** (eng.: *lossy*) – følg de røde pilene.
    - Her kan vi ikke rekonstruere bildet eksakt.
    - Resultatet kan likevel være «godt nok».
  - Det finnes en mengde ulike metoder for begge kategorier.

Ulike typer redundans

- Psykovisuell redundans. → Mer generelt: **Irrelevant informasjon**: Unødvendig informasjon for anvendelsen, f.eks. for visuell betraktning av hele bildet.
  - Det finnes informasjon vi ikke kan se.
    - Enkle muligheter for å redusere redundansen: Subsample eller redusere antall biter per piksel.
- Interbilde-redundans.
  - Likhet mellom nabobilder i en tidssekvens.
    - Kode noen bilder i tidssekvensen og ellers bare differanser.
- Intersampel-redundans.
  - Likhet mellom nabopiksler.
    - Hver linje i bildet kan løpelengde-transformeres.
- Kodings-redundans.
  - Gjennomsnittlig kodelengde minus et teoretisk minimum.
    - Velg en metode som er "grei" å bruke og gir liten kodingsredundans.

Kompresjonsrate og redundans

- **Kompresjonsraten:**

$$CR = \frac{i}{c}$$

der i er antall bit per sampel originalt,
og c er antall bit per sampel i det komprimerte bildet.

- **Relativ redundans:**

$$R = 1 - \frac{1}{CR} = 1 - \frac{c}{i}$$

- **«Percentage removed»:**

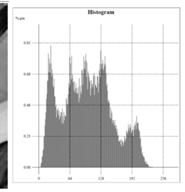
$$PR = 100\left(1 - \frac{c}{i}\right)\%$$

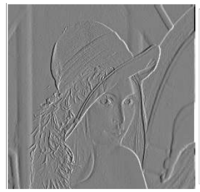
Differansetransform

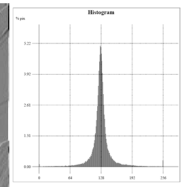
- Gitt en rad i bildet med gråtoner:
  $f_1, \ldots, f_N$ der $0 \le f_i \le 2^b-1$
- Transformer (reversibelt) til

$g_1 = f_1,\ g_2 = f_2 - f_1,\ \ldots\ ,\ g_N = f_N - f_{N-1}$

- Merk at: $-(2^b-1) \le g_i \le 2^b-1$
- Trenger derfor b+1 biter per g_i hvis vi skal tilordne like lange kodeord til alle mulig verdier.
- I differansehistogrammet vil de fleste verdiene samle seg rundt 0.
- Naturlig binærkoding av differansene er ikke optimal.







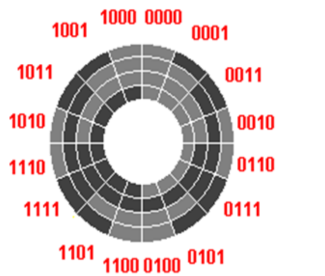
Løpelengde-transform

- Ofte inneholder bildet objekter med lignende gråtoner, f.eks. svarte bokstaver på hvit bakgrunn.
- Løpelengde-transformen (eng.: *run-length transform*) prøver å utnytte at **nabopiksler på samme rad ofte er like**.
  - Kompresjonen blir dårlig hvis dette ikke er tilfelle i det aktuelle bildet.
  - Løpelengde-transformen blir mer effektiv ettersom kompleksiteten i bildet blir mindre.
- Løpelengde-transformen er reversibel.
- Hvis pikselverdiene til en rad er:
  333333555555555544777777 (24 tall)
- Så starter løpelengde-transformen fra venstre og finner tallet 3 gjentatt 6 ganger etter hverandre, og returnerer derfor tallparet (3,6). **Formatet er: (tall, løpelengde)**
- For hele sekvensen vil løpelengdetransformen gi de 4 tallparene:
  (3,6), (5,10), (4,2), (7,6) (merk at dette bare er 8 tall)
- Kodingen avgjør hvor mange biter vi bruker for å lagre tallene.

Eksempel: Gray-koding

4-biters Gray- og naturlig binærkode:

Gray-kode	Naturlig binærk.	Desimal-tall
0000g	0000b	0d
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15



«Gray code shaft encoder»
Brukes for sikker avlesing av vinkel, f.eks. i styring av robot-armer.

Koden patentert av Gray i 1953, men ble brukt i Emilie Baudot's telegrafkode fra 1870.

Entropi

- Tar vi gjennomsnittet over alle symbolene s_i i alfabetet, får vi gjennomsnittlig informasjon per symbol.

$$\boxed{H = \sum_{i=0}^{2^b-1} p(s_i) I(s_i) = -\sum_{i=0}^{2^b-1} p(s_i) \log_2(p(s_i))}$$

- H er entropien til sekvensen av symbolene.

- **Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres.**
  - Gjelder bare hvis vi koder hvert symbol for seg.

Huffman-koding

- Huffman-koding er en algoritme for variabel-lengde koding som er **optimal** under begrensningen at vi **koder symbol for symbol**.
  - Med *optimal* menes her minst mulig kodings-redundans.

- Antar at vi kjenner hyppigheten for hvert symbol.
  - Enten spesifisert som en modell.
    - Huffman-koden er da optimal hvis modellen stemmer.
  - Eller så kan vi bruke symbol-histogrammet til sekvensen.
    - Huffman-koden er da optimal for sekvensen.
    - Ofte bruker vi sannsynlighetene i stedet, men vi kunne likegodt benyttet hyppighetene.

Eksempel: Huffman-koding

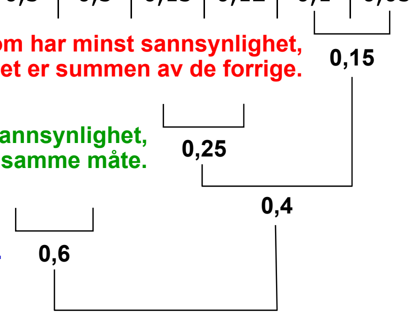
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

Slå sammen de to gruppene som har minst sannsynlighet,
Den nye gruppens sannsynlighet er summen av de forrige.

Finn de to som nå har minst sannsynlighet, og slå dem sammen på samme måte.

Fortsett til det er bare to igjen.

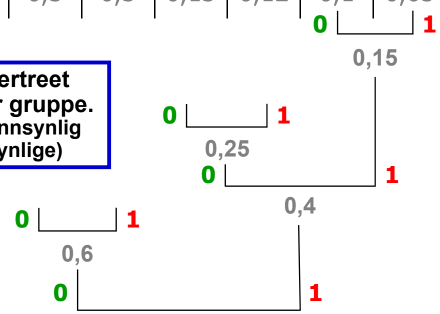


Eksempel: Huffman-koding

- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

Gå baklengs gjennom binærtreet og tilordne 0 eller 1 til hver gruppe.
(F. eks. kode 0 til den mest sannsynlig og kode 1 til den minst sannsynlige)



Eksempel: LZW-koding

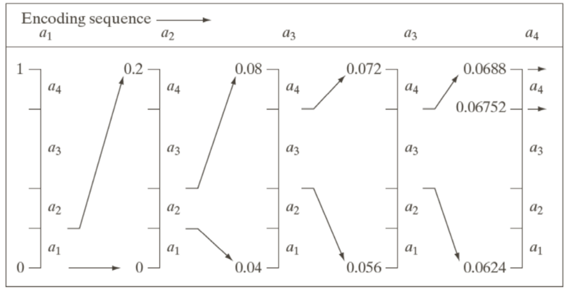
- Alfabetet: a, b og c med koder 0, 1 og 2, henholdsvis.
- Meldingen: ababcbababaaaaabab (18 symboler)
- LZW-sender: ny streng = sendt streng pluss neste usendte symbol
- LZW-mottaker: ny streng = nest siste streng pluss første symbol i sist tilsendte streng

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=0, b=1, c=2			a=0, b=1, c=2
a	0	ab=3	0	a	
b	1	ba=4	1	b	ab=3
ab	3	abc=5	3	ab	ba=4
c	2	cb=6	2	c	abc=5
ba	4	bab=7	4	ba	cb=6
bab	7	baba=8	7		

» Vi mottar kode 7, men denne koden finnes ikke i listen!
» Fra ny-streng-oppskriften vet vi at kode 7 ble laget ved: ba + ?
» Siden kode 7 nå sendes, må: ? = b => 7 = ba + b = bab

Eksempel: Aritmetisk koding

- Sannsynlighetsmodell: $P(a_1)=P(a_2)=P(a_4)=0{,}2$ og $P(a_3)=0{,}4$
- Melding/symbolsekvens: $a_1a_2a_3a_3a_4$



- a_1 ligger i intervallet [0, 0,2)
- a_1a_2 ligger i intervallet [0,04, 0,08)
- $a_1a_2a_3$ ligger i intervallet [0,056, 0,072)
- $a_1a_2a_3a_3$ ligger i intervallet [0,0624, 0,0688)
- $a_1a_2a_3a_3a_4$ ligger i intervallet [0,06752, 0,0688)

Eksempel: Representasjon av intervall

- Finn kortest mulig $N=0,c_1c_2c_3\ldots_2$ innenfor intervallet [0,6, 0,7).
- Hvis $n \geq k$ så er:

$$2^{-k+1} = 2^{-(k-1)} > c_k 2^{-k} + \ldots + c_n 2^{-n}$$

siden c_i er 0 eller 1.
- Derfor er:

$N = 0{,}\mathbf{1}\ldots_2 \quad \Rightarrow 0{,}5 \leq N < 1$

$N = 0{,}\mathbf{10}\ldots_2 \quad \Rightarrow 0{,}5 \leq N < 0{,}75$

$N = 0{,}\mathbf{100}\ldots_2 \quad \Rightarrow 0{,}5 \leq N < 0{,}625$

$N = 0{,}\mathbf{101}\ldots_2 \quad \Rightarrow 0{,}625 \leq N < 0{,}75$

- => Intervall kan kodes ved det binære kommatallet $0{,}\mathbf{101}_2$ (ekvivalent med $0{,}625_{10}$), altså med bare 3 biter.
  - Hvis vi vil kreve at øvre og nedre grense er innenfor intervallet; intervallet kan kodes ved $0{,}1010_2$ fordi:

    $N = 0{,}\mathbf{1010}\ldots_2 \Rightarrow 0{,}625 \leq N < 0{,}6875$

Ikke-tapsfri JPEG-kompresjon

1. Hver bildekanal deles opp i blokker på 8x8 piksler, og hver blokk i hver kanal kodes separat.
2. Dersom intensitetene er gitt uten fortegn; trekk fra 2^{b-1} der 2^b er antall intensitetsverdier.
   - Gjør at forventet gjennomsnittlig pikselverdi er omtrent 0.
   - Eks.: Intensitetsintervallet [0, 255]; 128 trekkes fra alle pikselverdiene.
3. Hver blokk transformeres med 2D DCT (diskret cosinus-transform).

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

2D DCT

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.05	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

- Mye av informasjonen i de 64 pikslene samles i en liten del av de 64 2D DCT-koeffisientene; nemlig de i øverste, venstre hjørne.

Ikke-tapsfri JPEG-kompresjon

JPEG-kompresjonsalgoritmen fortsetter med at:

4. 2D DCT-koeffisientene:
   a) punktdivideres med en vektmatrise og deretter
   b) kvantiseres til heltall.

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.05	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

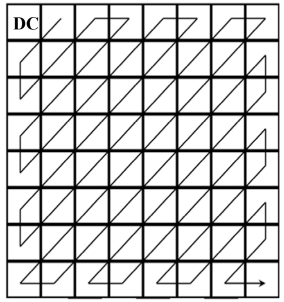
dividers med

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

resultatet avrundes til

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Ikke-tapsfri JPEG-kompresjon



DC- og AC-elementene behandles nå separat.

AC-elementene:

1. Sikk-sakk-skannes:
   – Ordner elementene i en 1D-følge.
   – Absoluttverdien av elementene vil stort sett avta utover i følgen.
   – Mange koeffisienter er null, spesielt litt uti følgen.
2. «Løpelengdetransform» av 1D-følgen.
   – Et «løpelengdepar» er her (antall 0-ere, antall biter i «ikke-0»).
3. «Løpelengdeparene» Huffman- eller aritmetisk kodes.
   – Både predefinerte og egendefinerte Huffman-kodebøker tillates.

Ikke-tapsfri JPEG-kompresjon

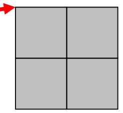
DC- og AC-elementene behandles nå separat.

DC-elementene:

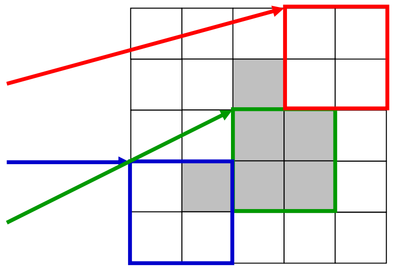
- For hver kanal samles DC-elementene fra alle blokkene.
- Disse er korrelerte og blir derfor differansetransformert.
- Differansene Huffman-kodes eller aritmetisk kodes.

Tre sentrale begrep

- Et **strukturelement** for et binært bilde er et **naboskap**.
  – Typisk definert ved en binær matrise der 1 markerer med i naboskapet.



- Når vi fører strukturelementet over det binære bildet vil vi finne:
  – Posisjoner der strukturelementet **ikke overlapper** objektet.
  – Posisjoner der strukturelementet delvis overlapper objektet, vi sier at elementet **treffer** objektet.
  – Posisjoner der strukturelementet ligger inni objektet, vi sier at elementet **passer** i objektet.



I figurene markerer grått med i mengden (forgrunnspiksel), og hvitt ikke med.

Anvendelse av erosjon: Kantdeteksjon

- Erodering fjerner piksler langs omrisset av et objekt.
- Vi kan finne kantene av objektene i bildet ved å subtrahere et erodert bilde fra originalbildet: **g = f - (f ⊖ S)**
- Strukturelementet avgjør 4- eller 8-tilkoblet:

Et bilde

```
0 0 1 1 1 1 0 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
1 1 1 1 0 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 1 0 0 0
```

erodert med

```
0 1 0
1 [1] 1
0 1 0
```

gir

```
0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 1 1 0 0
0 0 1 1 0 1 1 1 1 0 0
0 1 1 0 0 0 1 1 1 1 0
0 0 1 1 0 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

=> differanse

```
0 0 1 1 1 1 0 1 1 1 0
0 1 0 0 0 0 1 0 0 1 0
0 1 0 0 1 0 0 0 0 1 0
1 0 0 1 0 1 0 0 0 0 1
0 1 0 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 0 0 1 0
0 1 1 1 1 0 0 0 1 1 0
0 0 0 0 0 1 1 1 0 0 0
```

Sammenhengende kanter dersom man bruker 8-tilkobling

erodert med

```
1 1 1
1 [1] 1
1 1 1
```

gir

```
0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 1 0 0
0 0 1 0 0 0 1 1 1 0 0
0 0 1 0 0 0 1 1 1 0 0
0 0 1 0 0 0 1 1 1 0 0
0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

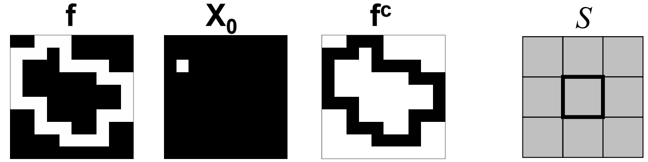
=> differanse

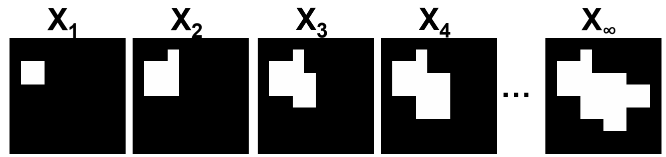
```
0 0 1 1 1 1 0 1 1 1 0
0 1 1 0 0 1 1 1 0 1 0
0 1 0 1 1 1 0 0 0 1 0
1 1 0 1 0 1 0 0 0 1 1
0 1 0 1 1 1 0 0 0 1 1
0 1 0 0 0 0 0 0 0 1 0
0 1 1 1 1 1 0 1 1 1 0
0 0 0 0 0 1 1 1 0 0 0
```

Sammenhengende kanter ved bruk av 4-tilkobling

Anvendelse av dilasjon: Region-fylling

- La X_0 inneholde et punkt i regionen som skal fylles.
- Deretter iterer $X_k = (X_{k-1} \oplus S) \cap f^c$ inntil konvergens:



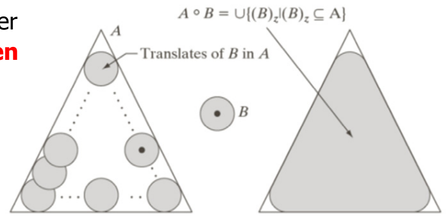


$(X_1 \oplus S) \longrightarrow$

(Bildene er hentet fra http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm)

Geometrisk tolkning av åpning

- Tenk at strukturelementet definerer **størrelsen og formen til spissen av en tusjpenn**.
- Det er bare tillatt å **fargelegge innenfor objekter**.
  - Et par detaljer: Man må holde tusjen vinkelrett på tegneflaten og med samme rotasjon som strukturelementet.
- **Åpningen er resultatet av å fargelegge så mye man har lov til.**
- For runde strukturelementer: Konvekse hjørner blir avrundet, konkave hjørner beholdes rette.
  - Akkurat som ved dilasjon (skyldes at enhver åpning avsluttes med en dilasjon).



(Deler av figur 9.8 i G&W)

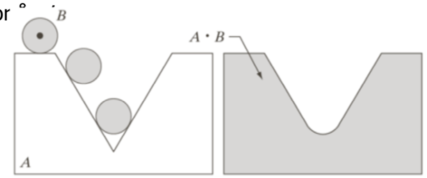
Åpning er idempotent:

$$(f \circ S) \circ S = f \circ S$$

d.v.s. at gjentatte anvendelser med samme strukturelement gir ingen endring.

Geometrisk tolkning av lukking

- Vi kan benytte **samme metafor** som for åpning
  - Strukturelementet definerer størrelsen og formen til spissen av en tusjpenn.
  - Man holder tusjen vinkelrett tegneflaten og fargelegger så mye man har lov til.
- **Denne gangen** er det bare tillatt å **fargelegge utenfor objekter**.
  - En detalj: Denne gangen skal tusjen holdes speilvendt (180° rotert) av strukturelementet.
- **Lukkingen er det som ikke fargelegges.**
  - Denne gangen fargelegger vi altså bakgrunnen, sist fargela vi forgrunnen.
- For runde strukturelementer: Konkave hjørner blir avrundet, konvekse hjørner beholdes rette.
  - Akkurat som ved erosjon (skyldes at enhver lukking avsluttes med en erosjon).



(Deler av figur 9.9 i G&W)

Også lukking er idempotent:

$$(f \bullet S) \bullet S = f \bullet S$$

Eksempel: Filtrering ved åpning og lukking

a b
d c
e f

FIGURE 9.11
(a) Noisy image.
(b) Structuring element.
(c) Eroded image.
(d) Opening of A.
(e) Dilation of the opening.
(f) Closing of the opening.
(Original image courtesy of the National Institute of Standards and Technology.)

Morfologisk rekonstruksjon

- Tidligere: Bilde f og strukturelement S
- Nå: Markørbilde F (startpunktene), maskebilde G og et strukturelement B som definerer tilkoblingstypen
- **Morfologisk rekonstruksjon ved dilasjon**:

$$F^k = (F^{k-1} \oplus B) \bigcap G$$

der $F^0 = F$.
- Med ord: Iterativt dilater F med B, og begrens resultatet med G, inntil ingen endring.

- Anvendelser:
  - Region-fylling (fire slides siden).
  - Helautomatisk hullfylling.
  - Kantrydding.
  - Åpning ved rekonstruksjon.