INSTITUT FOR MATEMATISKE FAG
AALBORG UNIVERSITET

Billedbehandling

Introduktion

Denne workshop omhandler billedredigering og den matematik der ligger bag visse måder at redigere billeder på. Vi vil modellere (sort-hvid) billeder ved hjælp af matricer. Ved et billede forstår vi altså en matrix $A = [a_{ij}]$. Vi kalder hver indgang (i, j) for en *pixel* og a_{ij} kaldes *intensiteten* af pixel (i, j). Grunden til at vi ikke behandler farvebilleder er at et farvebillede typisk udgøres at tre matricer R, G og B der indeholder intensiteter for hhv. rød, grøn og blå. Derfor kan mange teknikker der anvendes til sort-hvid billeder anvendes til farvebilleder ved at gentage dem for hver af de tre matricer R, G og B.

Vi vil skelne mellem to slags billeder: *Digitale billeder* og *ægte billeder*[1]. Forskellen på de to typer af billeder er hvordan intensiteterne er givet. For et ægte billede antager vi at intensiteterne a_{ij} er reelle tal, hvor 0 svarer til helt sort og jo højere en intensitet bliver jo mere hvid bliver den tilhørende pixel. Når billeder skal gemmes på en computer skal man træffe et valg om hvor meget plads (hvor mange bits) der skal bruges pr. pixel. Et typisk farvebillede bruger 24 bits pr pixel; 8 bits til hver af farverne rød, grøn og blå. Vi vil også antage at vores digitale (sort-hvid) billeder skal gemmes med 8 bits pr. pixel. Dermed er der 256 (2^8) mulige intensiteter og vi repræsenterer derfor intensiteterne i et digitalt billede med heltallene $0, 1, 2, \ldots, 255$[2]. Her svarer 0 til helt sort og 255 til helt hvid.

Vi skal i denne workshop se på to problemer relateret til billedredigering, og til dette skal vi bl.a. anvende Matlab. I Matlab er ægte billeder kendetegnet ved at være matricer hvor indgangene er af datatypen `double`, mens digitale billeder er af datatypen `uint8`. Datatypen `double` er (meget forenklet) måden vi repræsenterer reelle tal i en computer og datatypen `uint8` er heltal repræsenteret ved 8 bit (altså tallene $0, 1, \ldots, 255$). Man kan få Matlab til at vise et billede givet ved matricen A ved at anvende kommandoen `imshow(A)`. For et digitalt billede viser viser denne kommando billedet. For et ægte billede $A = [a_{ij}]$ er der først nødt til at ske en konvertering til et digitalt billede $D = [d_{ij}]$. Som default sker denne konvertering ved at man definerer

$$d_{ij} = \begin{cases} 0, & \text{hvis } a_{ij} \leq 0 \\ [255a_{ij}], & \text{hvis } 0 < a_{ij} < 1 \\ 255, & \text{hvis } a_{ij} \geq 1 \end{cases}, \tag{1}$$

[1] Terminologien ægte billeder er opfundet til denne workshop.

[2] Man kunne også have brugt f.eks. 16 bit pr. pixel. Så havde man haft 65536 mulige intensiteter.

hvor $[x]$ er funktionen der afrunder x til nærmeste heltal. Som man kan se tager denne konvertering kun højde for de intensiteter der ligger i intervallet $[0, 1]$. Hvis ikke man ønsker dette kan man bede Matlab om først at konvertere ens billede til et hvor intensiteterne ligger i intervallet $[0, 1]$. I praksis gøres dette ved at bruge kommandoen `imshow(A,[])` hvilket først konverterer $A = [a_{ij}]$ til matricen $A_1 = [a'_{ij}]$ hvor

$$a'_{ij} = \frac{a_{ij} - \min(A)}{\max(A)}, \tag{2}$$

hvor $\max(A)$ er den største indgang i A og $\min(A)$ er den mindste. Herefter vises billedet hvor man anvender konverteringen i (1) på A_1.

Delopgave 1 (intensitetstransformation)

I denne opgave betragter vi et ægte billede A. Dette billede ses i Figur 1, hvor vi har anvendt `imshow(A)` og `imshow(A,[])` til at vise billedet. Figur 1a fortæller os at en stor del af billedet har intensiteter der er større end 1 (de hvide områder). Figur 1b fortæller os at når billedet er blevet konverteret som i (2) og derefter til et digitalt billede vha. (1) så er næsten alle pixelsene sorte. Dette tyder på at $\max(A)$ er meget stor sammenlignet med de fleste andre pixels og at de 8 bit vi har til rådighed til at repræsentere farver simpelthen er for få til at vi kan se detaljerne i billedet.

En måde at redigere vores billede så det bliver nemmere at se hvad er foregår er ved at udføre en såkaldt intensitetstransformation. Dette gøres ved at danne et nyt billede B hvor

$$B = [f(a_{ij})], \tag{3}$$

for en funktion f. I vores tilfælde skal vi have en funktion der reducerer de høje intensiteter. I denne delopgave vil vi arbejde med en logaritmisk transformation:

$$f(x) = c \ln(1 + x), \tag{4}$$

hvor c er en positiv konstant.

(i) Bestem $f(0)$ og argumenter herefter for at $f(x) \geq 0$.

(ii) Vis at $g(x) = e^{\frac{x}{c}} - 1$ er den inverse til f. Hvordan kan man bruge den inverse funktion til f til at bestemme vores originale billede ud fra B i (3)?

(a) `imshow(A)` (b) `imshow(A,[])`

Figur 1: Billedet A der anvendes i delopgave 1

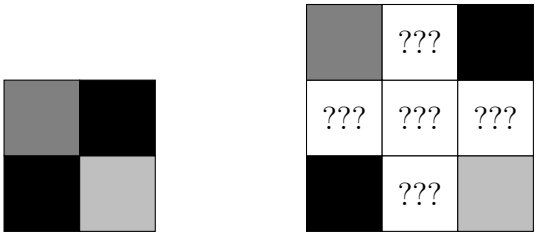
(iii) Implementer funktionen `log_transform(A,c)` i Matlabfilen `log_transform.m`. Denne skal tage et billede A og en værdi af c og så udføre transformationen i (3) med f givet som i (4). Hint: Man kan med fordel anvende funktionen `arrayfun(@(x),A)` der anvender funktionen `@(x)` på hver indgang i `A`.

(iv) Vores originale billede A findes i filen `delopgave1.txt`. Brug Matlabfilen `images.m` sammenligne billedet der fremkommer ved at transformere A med funktionen fra (iv) når $c = 1/7$ med billederne i Figur 1. Personen på billedet er matematikeren Irving Stringham, som var den første til at anvende notationen ln for den naturlige logaritme!

Delopgave 2

Et klassisk problem indenfor billedredigering er skalering af billeder, dvs. at gøre billeder større og eller mindre. Lad os antage at vi har et $m \times n$ billede. Hvis vi vil skalere billedet med 2 så skal vi lave et $2m \times 2n$ billede. Vores nye billede skal have 4 gange så mange pixels som det gamle, men hvor skal de pixels komme fra (se også Figur 2)? Vi skal altså bruge en måde at "gætte" på hvad de manglende pixels skal være. På tilsvarende vis, hvis vi skal skalere

et billede med $\frac{1}{2}$ så skal vi kun have det kvarte antal pixels af det originale billede. Men hvordan skal vi udvælge de pixels der bliver tilbage? Svaret på disse spørgsmål er interpolation. I denne delopgave vil vi se på interpolation i én dimension og i delopgave 3 skal vi prøve at udvide det til to dimensioner og anvende det på billeder.



Figur 2: Hvis 2×2 billedet til venstre skaleres med en faktor på 1.5 fås et 3×3 billede, men hvilken intensitet skal de hvide pixels have?

Interpolation går ud på at man ud fra nogle givne datapunkter prøver at finde en funktion der "forudsiger" hvad der sker mellem datapunkterne. Datapunkterne kunne f.eks. være intensiteterne i (en række af) et billede eller vigtige frames i en animation. Interpolation vil så hjælpe med at bestemme hvad der sker mellem to pixels eller to frames. Dette kan bruges hvis man ønsker at skalere et billede op, eller ikke ønsker at tegne alle frames i en animation i hånden. Andre anvendelser kan findes online.

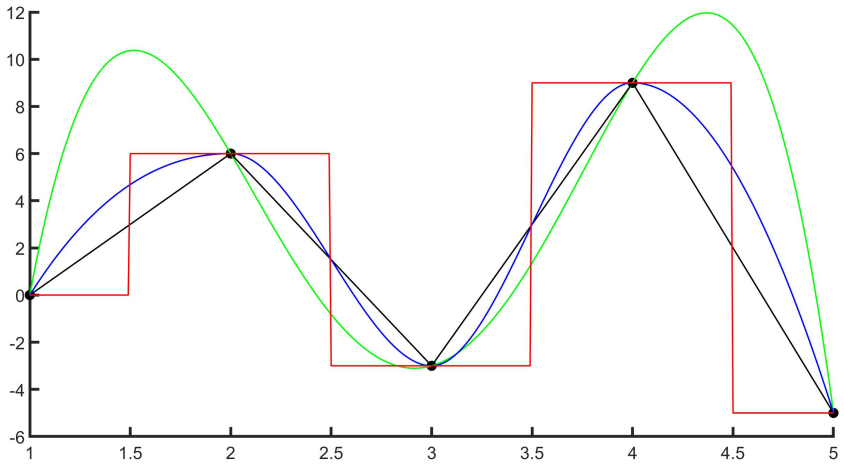
Vi vil tage udgangspunkt i datapunkterne $(x_1, y_1), (x_2, y_2), \ldots (x_k, y_k)$. Målet med interpolation er at bestemme en *interpolerende funktion*, dvs. en funktion der går gennem alle datapunkterne. Der findes forskellige måder at gøre dette på (se Figur 3), men i denne workshop ønsker vi at bestemme en stykkevist defineret funktion f:

$$f(x) = \begin{cases} f_1(x), & \text{hvis } x_1 \leq x < x_2 \\ f_2(x), & \text{hvis } x_2 \leq x < x_3 \\ \vdots & \\ f_{k-1}(x) & \text{hvis } x_{k-1} \leq x \leq x_k \end{cases}. \tag{5}$$

F.eks. er den simpleste stykkevist definerede interpolerende funktion kendt som *nærmeste nabo interpolation* (rød kurve i Figur 3) og denne er defineret ved

$$f_j(x) = \begin{cases} y_j, & \text{hvis } x_j \leq x \leq \frac{1}{2}(x_j + x_{j+1}) \\ y_{j+1}, & \text{hvis } \frac{1}{2}(x_j + x_{j+1}) < x \leq x_{j+1} \end{cases}.$$

Vi vil i denne delopgave beskæftige os med primært to andre slags: Lineær interpolation og kubisk Hermite interpolation.



Figur 3: Forskellige funktioner der interpolerer punkterne $(1,0)$, $(2,6)$, $(3,-3)$, $(4,9)$ og $(5,-5)$: Lagrangeinterpolation (grøn), nærmest nabo interpolation (rød), lineær interpolation (sort), kubisk Hermite interpolation (blå).

(i) Ved *lineær interpolation* ønsker vi at bestemme en stykkevist defineret funktion f som i (5), hvor $f_j(x) = a_j x + b_j$. Argumenter for at a_j og b_j løser ligningssystemet

$$\begin{bmatrix} x_j & 1 \\ x_{j+1} & 1 \end{bmatrix} \begin{bmatrix} a_j \\ b_j \end{bmatrix} = \begin{bmatrix} y_j \\ y_{j+1} \end{bmatrix}. \tag{6}$$

(ii) Lad

$$A = \begin{bmatrix} x_j & 1 \\ x_{j+1} & 1 \end{bmatrix}.$$

Vis at A er inverterbar hvis og kun hvis at $x_j \neq x_{j+1}$. Bestem herefter A^{-1} (under antagelse af at $(x_j \neq x_{j+1})$ og brug denne til at løse (6).

(iii) Konkluder ud fra (ii) at f_j er givet ved

$$f_j(x) = \frac{y_{j+1} - y_j}{x_{j+1} - x_j}(x - x_j) + y_j$$

(iv) Ved *kubisk Hermite interpolation* ønsker vi at bestemme f i (5) hvor f_j er et tredjegradspolynomium $f_j(x) = \alpha_j x^3 + \beta_j x^2 + \gamma_j x + \delta_j$ som udover at gå gennem (x_j, y_j) og (x_{j+1}, y_{j+1}) samtidig opfylder at $f_j'(x_j) = d_j$ og $f_j'(x_{j+1}) = d_{j+1}$. For at simplificere dette en smule antager vi at $x_j = 0$

og at $x_{j+1} = 1$. Vis at koefficienterne a_j, b_j, c_j, og d_j i dette tilfælde skal løse ligningssystemet

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_j \\ \beta_j \\ \gamma_j \\ \delta_j \end{bmatrix} = \begin{bmatrix} y_j \\ d_j \\ y_{j+1} \\ d_{j+1} \end{bmatrix} \tag{7}$$

(v) Vis ved at bruge Gauss-elimination at den entydige løsning til (7) er givet ved

$$\begin{aligned} \alpha_j &= 2y_j - 2y_{j+1} + d_j + d_{j+1} \\ \beta_j &= 3y_{j+1} - 3y_j - 2d_j - d_{j+1} \\ \gamma_j &= d_j \\ \delta_j &= y_j \end{aligned}$$

(vi) Antag at $f_j(x) = \alpha_j x^3 + \beta_j x^2 + \gamma_j x + \delta_j$ opfylder at $f_j(0) = y_j$, $f_j(1) = y_{j+1}$, $f_j'(0) = d_j$ og $f_j'(1) = d_j$. Vis at hvis at hvis $x_{j+1} = x_j + 1$ så opfylder $g_j(x) = f_j(x - x_j)$ at $g_j(x_j) = y_j$, $g_j(x_{j+1}) = y_{j+1}$, $g_j'(x_j) = d_j$ og $g_j'(x_{j+1}) = d_{j+1}$.

Det betyder at vi kan interpolere mellem punkterne (x_j, y_j) og $(x_j + 1, y_j)$ ved at erstatte x med $x - x_j$ i den interpolerende funktion mellem $(0, y_j)$ og $(1, y_j)$. Med andre ord kan vi "genbruge" koefficienterne fra (v). I delopgave 3 skal vi se nytten af dette.

(vii) Punkterne (x_j, y_j) er typisk givet ud fra anvendelsen, mens hældningerne d_j ikke er givet. I praksis (dvs. delopgave 3) vil vi anvende følgende formel

$$d_j = \frac{y_{j+1} - y_{j-1}}{2},$$

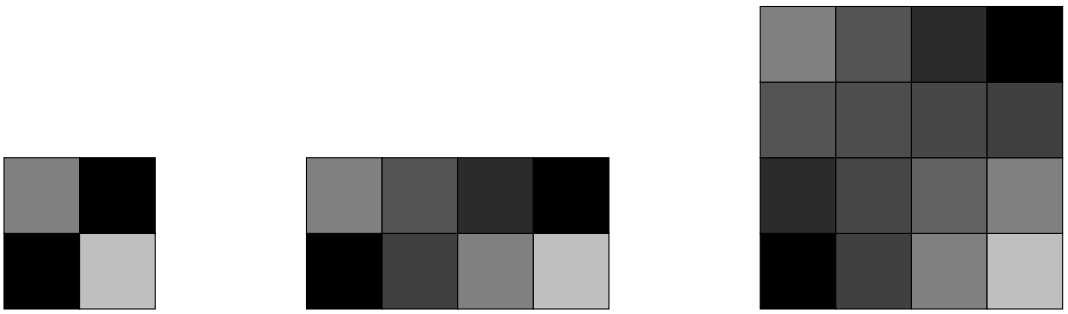
hvor vi definerer $y_0 = y_1$ og $y_{k+1} = y_k$. Argumenter ved hjælp af (v) og (vi) for at dette medfører at

$$\begin{aligned} f_j(x) = &\left(\frac{1}{2}y_{j-1} + \frac{3}{2}y_j - \frac{3}{2}y_{j+1} + \frac{1}{2}y_{j+2}\right)(x - x_j)^3 \\ &+ \left(y_{j-1} - \frac{5}{2}y_j + 2y_{j+1} - \frac{1}{2}y_{j+2}\right)(x - x_j)^2 \\ &+ \left(-\frac{1}{2}y_{j-1} + \frac{1}{2}y_{j+1}\right)(x - x_j) + y_j \end{aligned}$$

opfylder at $f_j(x_j) = y_j$, $f_j(x_{j+1}) = y_{j+1}$, $f'(x_j) = \frac{y_{j+1} - y_{j-1}}{2}$ og $f'(x_{j+1}) = \frac{y_{j+2} - y_j}{2}$.

Delopgave 3

I delopgave 2 så vi hvordan man kunne bestemme interpolerende funktioner af en variabel. Dette skal vi nu bruge til at kunne interpolere billeder. Vi vil gøre det som i følgende eksempel (se også Figur 4): Antag at vi har et $m \times n$ billede A, som vi gerne vil forstørre til et $2m \times 2n$ billede. Lad $\mathbf{r}_j$ være den j'te række i A. Vi vil tænke på intensiteterne $r_{j1}, r_{j2} \ldots, r_{jn}$ som punkterne $(1, r_{j1}), (2, r_{j2}), \ldots (n, r_{jn})$ i planen. Herefter finder vi en funktion f_j som interpolerer disse datapunkter og definerer et $m \times 2n$ billede A_1 hvor intensiteterne i den j'te række er givet ved $f_j(1), f_j(1 + \frac{n-1}{2n-1}), \ldots f_j(n)$. Vi laver nu samme procedure for søjlerne i A_1. Hvis $\mathbf{s}_j$ er søjle j i A_1 så tænker vi på intensiteterne $s_{1j}, s_{2j}, \ldots s_{mj}$ som punkterne $(1, s_{1j}), (2, s_{2j}), \ldots, (m, s_{mj})$ i planen. Vi bestemmer igen en funktion g_j som interpolerer disse datapunkter. Derefter definerer vi vores resulterende $2m \times 2n$ billede A_2 ved at definere intensiteterne i den j'te søjle som $g_j(1), g_j(1 + \frac{m-1}{2m-1}), \ldots g_j(m)$.



Figur 4: Venstre: 2×2 billedet A som skal forstørres til et 4×4 billede. Midten: 2×4 billedet A_1 hvor rækkerne er blevet interpoleret. Højre: Det forstørrede billede A_2 hvor søjlerne i A_1 er blevet interpoleret.

(i) I filen `resize_image.m` er der lavet en implementation af den procedure ovenfor der transformerer A til A_1. Denne funktion hedder

```
unit_step_row_interpolate(A,new_cols,method)
```

og kan anvende både nærmeste nabo interpolation og lineær interpolation. Bemærk at der for nærmeste nabo interpolation anvendes en hurtigere metode end den beskrevet ovenfor. Udvid denne funktion så den også kan lave kubisk Hermite interpolation.

(ii) Overvej at proceduren der transformerer A_1 til A_2 kan udføres ved at anvende `unit_step_row_interpolate(A,new_cols,method)` på $(A_1)^T$. Implementer denne procedure i filen `resize_image.m`.

(iii) Brug filen `images.m` til at teste forskellen på de forskellige interpolationsmetoder.