Musteraufgaben

Musterfragen

Verständnis:

Erklären Sie die Idee sowie die Formeln von bilinearer Interpolation mit einem Schaubild. Geben Sie Formeln an.

Verständnis und Rechnen:

Angenommen Sie haben eine Orthonormalbasis $\{\vec{z}^k \in \mathbb{R}^n \mid k \in \{0, ..., n-1\}\}$ von $\mathbb{R}^n$. Was sind die Koeffizienten für die Darstellung eines beliebigen Vektors f als Linearkombination der z^k? Was hat dies mit der Kosinustransformation zu tun? Bonuspunkt: Beweisen Sie ihre Antwort auf die erste Frage.

Rechnen:

Berechnen Sie den 3x3 Medianfilter auf folgendem Bild / folgender Matrix:

$$\begin{pmatrix} 9 & 7 & 1 & 4 & 1 \\ 2 & 7 & 6 & 1 & 3 \\ 7 & 0 & 10 & 9 & 8 \\ 1 & 4 & 1 & 1 & 5 \\ 6 & 9 & 8 & 6 & 9 \end{pmatrix}$$

Es genügt wenn Sie den gültigen Bereich des Filters (ohne Randbedingungen) ausrechnen (sodass das Ergebnis kleiner ist als die Eingabe).

Programmieren:

Schreiben Sie eine Funktion *projTransformation*, die als Eingabe ein Grauwertbild $f \in \mathbb{R}^{n_y \times n_x}$ (in Form eines numpy Arrays) sowie eine Matrix $H \in \mathbb{R}^{3\times 3}$ bekommt und als Ausgabe ein Bild $g \in \mathbb{R}^{n_y \times n_x}$ zurück gibt welches den Wert an der Stelle (i, j) durch eine projektive Transformation mit H aus den Werten des Bildes f interpoliert.

Sie können davon ausgehen, dass eine Funktion $interpolate(x, f)$ existiert, die den Intensitätswert an der Stelle x aufgrund der Intensitätsinformationen in f interpoliert.

Musterantworten

Verständnis:

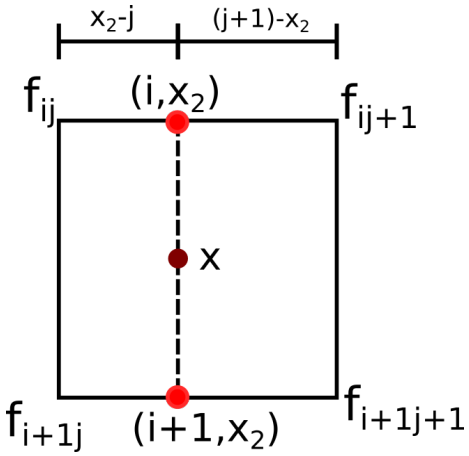

Figure 1: Schaubild

Seien $x = (x_1, x_2)$ die Koordinaten des Punktes an dem ein Wert durch bilineare Interpolation bestimmt werden soll. Wie im Schaubild dargestellt such man sich zunächst die vier umliegenden bekannten Pixel mit Werten $f_{i,j}, f_{i,j+1}, f_{i+1,j}, f_{i+1,j+1}$ mittels $(i, j) = (textfloor(x_1), \text{floor}(x_2))$. Um den Wert an der Stelle x zu bestimmen interpoliert man nun erst linear entlang einer Koordinatenachse um den Wert zweier Hilfspunkte zu bekommen, welche dann benutzt werden um lineare entlang der anderen Koordinatenachse zu interpolieren. Man kann zeigen, dass die Reihenfolge hierbei keine Rolle spielt.

Die Formel zur Berechnung des Wertes $f(x)$ an der Stelle x lauten:

$$f(i, x_2) = (x_2 - j)f_{i,j+1} + (j + 1 - x_2)f_{i,j}$$

$$f(i + 1, x_2) = (x_2 - j)f_{i+1,j+1} + (j + 1 - x_2)f_{i+1,j}$$

$$f(x) = (x_1 - i)f(i + 1, x_2) + (i + 1 - x_1)f(i, x_2)$$

Verständnis und Rechnen:

Die Koeffizienten c_k der Linearkombination der $\vec{z}^k$ zur Darstellung des Vektors $f = \sum_{k=0}^{n-1} c_k \vec{z}^k$ sind gegeben durch

$$c_k = \langle \vec{z}^k, f \rangle.$$

Der resultierende Vektor $\vec{c}$ aus Koeffizienten ist genau die diskrete Kosinustransformation von f, falls die $\vec{z}^k$ Auswertungen von (skalierten) Kosinusfunktionen unterschiedlicher Frequenz sind.

Bonusaufgabe: Aufgrund der Darstellung $f = \sum_{k=0}^{n-1} c_k \vec{z}^k$ gilt:

$$\begin{aligned}\langle f, \vec{z}^s \rangle &= \langle \sum_{k=0}^{n-1} c_k \vec{z}^k, \vec{z}^s \rangle \\ &= \sum_{k=0}^{n-1} c_k \langle \vec{z}^k, \vec{z}^s \rangle \\ &= c_s\end{aligned}$$

wobei im letzten Schritt die Tatsache verwendet wurde, dass die $\vec{z}^k$ eine Orthonormalbasis bilden, also

$$\langle \vec{z}^k, \vec{z}^s \rangle = \begin{cases} 1 & \text{falls } = s, \\ 0 & \text{sonst.} \end{cases}$$

Rechnen:

Die Anwendung des Medianfilters resultiert in folgendem Bild/Matrix:

$$\begin{pmatrix} 7 & 6 & 4 \\ 4 & 4 & 5 \\ 6 & 6 & 8 \end{pmatrix}$$

Programmieren:

```
import numpy as np
def projTransformation(f, H):
    result = np.copy(f)
    it = np.nditer(f, flags=['multi_index'])
    while not it.finished:
        z = np.append(np.array(it.multi_index),[1])
        x = np.matmul(H, z)
        x = x/x[2]
        result[it.multi_index] = interpolate(x,f)
        it.iternext()
    return result
```

Alternative ebenfalls richtige Lösung ohne Verwendung von Multiindizes

```
import numpy as np
def projTransformation(f, H):
    result = np.copy(f)
    ny,nx = f.shape
    for i in range(0,ny):
        for j in range(0,ny):
                z = [i,j,1]
                x = np.matmul(H, z)
                x = x/x[2]
                result[i,j] = interpolate(x,f)
    return result
```