

Algorithmik kontinuierlicher Systeme Aufgabenblatt 5 — Quantisierung und Downsampling

Allgemeines:

- Die Abgabe der Programmieraufgaben erfolgt über das Exercise Submission Tool:
<https://est.cs.fau.de>
- Sie können während der Bearbeitungszeit Ihre Abgaben im EST beliebig oft aktualisieren. Nur die aktuellste Abgabe, die in der Bearbeitungszeit hochgeladen wurde, wird gewertet.
- Auf der Übungswebsite finden Sie zu jedem Übungsblatt eine Vorlage, sowie zu jeder Aufgabe eine Datei `*_test.py` mit der Sie ihre Lösungen jederzeit selbst überprüfen können.
- Damit Sie auf eine Teilaufgabe Punkte bekommen, muss sie mit Python 3.5 im CIP Pool funktionieren. Das bestehen der mitgelieferten Tests ist notwendig, aber nicht hinreichend dafür, dass Sie Punkte bekommen.

In diesem Arbeitsblatt werden Sie das erste mal Bildverarbeitung in Python betreiben. Da auf den Bearbeitungszeitraum ein Feiertag und die Bergkirchweih fällt, hat dieses Blatt etwas weniger Punkte :-)

Aufgabe 1 — Median Cut (8 Punkte)

mcut.py

In dieser Aufgabe implementieren Sie den rekursiven Median-Cut-Algorithmus, um den Farbraum eines Bildes zu quantisieren. Arbeiten Sie dazu die Teilaufgaben der Reihe nach durch. In der `main()`-Funktion finden Sie exemplarisch einen Aufruf des Algorithmus, inklusive dem Lesen und Anzeigen eines Bildes.

a) Pixelliste erzeugen: Schreiben Sie eine Funktion `image2pixels`, die ein Bild in eine Liste von Pixeln konvertiert. Jedes Pixel soll als `(R, G, B, X, Y)` Tupel repräsentiert werden. Das übergebene Bild ist ein dreidimensionales Array der Form `[Zeile, Spalte, RGB]` mit den Abmessungen $(y_{\max}, x_{\max}, 3)$ und dem Elementtyp `uint8`, d.h. jeder Farbkanal wird durch eine Ganzzahl zwischen 0 und 255 dargestellt. Die Pixel sollen zeilenweise in der zurückgegebenen Liste liegen.

Hinweis: Die Funktion `scipy.misc.imread` mit Argument `'RGB'` liefert standardmäßig dieses Format und eignet sich daher hervorragend zum Testen.

b) Pixelliste zurückkonvertieren Schreiben Sie nun die Funktion `pixels2image`, welche aus einer Liste von `(R, G, B, X, Y)` Tupeln, wieder ein dreidimensionales Array mit Elementtyp `uint8` liefert. Die Reihenfolge der übergebenen Pixel darf dabei keine Rolle spielen.

c) Maximaler umgebender Quader Schreiben Sie die Funktion `bounding_box`, welche den maximalen umgebenden Quader *um die Farbwerte* einer Liste von `(R, G, B, X, Y)` Tupeln berechnet. Der Quader soll in einem geschachtelten Tupel der Form `((Rmin, Rmax), (Gmin, Gmax), (Bmin, Bmax))` gespeichert werden.

d) Durchschnittsfarbe Schreiben Sie die Funktion `color_average`, welche eine Liste aus `(R, G, B, X, Y)` Tupeln übergeben bekommt, den Mittelwert aller Farben darin bestimmt und eine neue Liste der gleichen Form erzeugt, in der jedes Tupel den Mittelwert als Farbwert hat. Verwenden Sie dabei die Python Funktion `round`, um den Mittelwert in jedem Farbkanal von einer Gleitkommazahl in eine Ganzzahl zwischen 0 und 255 zu konvertieren.



Abbildung 1: Originalbild.



Abbildung 2: Median-Cut-Quantisierung mit 16 Farben im RGB-Farbraum.

e) **Schnittdimension** Implementieren Sie eine Hilfsfunktion `cut_dimension`, welche den Index der längsten Kante des umgebenden Farbquaders zurückgibt. Der Quader wird in dem Format aus Aufgabe c) erwartet. Der zurückgegebene Index soll bei Rot den Wert 0, bei Grün den Wert 1 und bei Blau den Wert 2 liefern. Bei Gleichstand wird Rot und danach Grün bevorzugt.

f) **Median Cut** Implementieren Sie nun die Funktion `recursive_median_cut`, die den Hauptteil des Median-Cut-Algorithmus rekursiv implementiert. Die Funktion erwartet eine Liste von (R, G, B, X, Y) Tupeln und eine nicht-negative Ganzzahl N , die angibt, wie viele rekursive Schritte noch durchgeführt werden sollen. Ist $N = 0$, oder der Farbraum bereits vollständig unterteilt, wird die Rekursion abgebrochen und alle Elemente auf die Durchschnittsfarbe gesetzt. Optional darf der maximale umgebende Farbquader übergeben werden, falls dieser bereits bekannt ist.

Die Unterteilung des Farbraumes wird in jedem rekursiven Schritt entlang des Indexes von Aufgabe e) durchgeführt, d.h. die Liste der Tupel wird bzgl. der längsten Seite der Bounding Box sortiert und dann in der Mitte geteilt (Berechnung des Median). Bei einer ungeraden Länge wird beim abgerundeten Index geteilt. Der Datentyp `list` verfügt in Python über eine Methode `sort(key=func)`, die die Elemente aufsteigend sortiert. Eine Lambda-Funktion kann ihnen dabei helfen den richtigen `key` auszuwählen.

Aufgabe 2 — Moiré-Effekt (3 Punkte)

moire.py

Ziel der Aufgabe ist es ein Downsampling zu implementieren, so dass im Schwarz-Weiß-Bild `moire.png` ein Moiré-Muster zu erkennen ist.

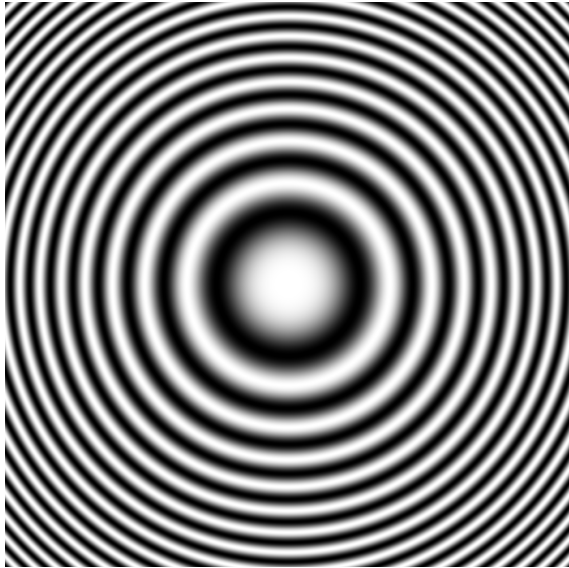


Abbildung 3: Originalbild.

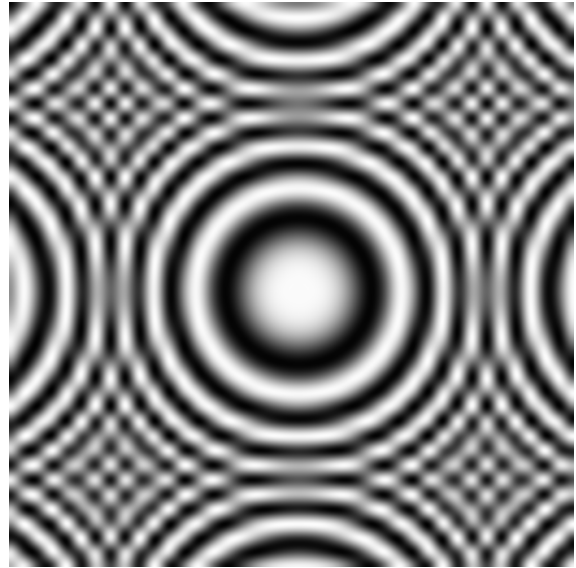


Abbildung 4: Moiré-Effekt durch Downsampling.

a) **Interpolation** Implementieren Sie die Funktion `stretch`, die die Höhe und Breite eines gegebenen Graustufenbilds um den Faktor `hstretch` und `wstretch` streckt. Zum interpolieren soll Spline-Interpolation dritter Ordnung verwendet werden.

Hinweis: Sie dürfen dazu Funktionen aus dem Modul `scipy.ndimage` verwenden. Mit der richtigen Hilfsfunktion ist diese Aufgabe sehr einfach.

b) **Downsampling** Implementieren Sie die Funktion `downsample`, die vom übergebenen Bild von links oben `img` jede `N`te Zeile und Spalte behält und die anderen entfernt.

Hinweis: Es wird nur ein Schwarz-Weiß-Bild eingelesen, d.h. jedem Pixel ist nur ein Skalar zugeordnet und keine Liste.

c) **Anwendung** Bestimmen Sie nun den Wert `N` in der Funktion `downsample`, so dass aus der Datei `moire.png` das Moiré-Muster auf diesem Aufgabenblatt erkennbar wird. Lassen Sie die Funktion `moire_parameter` diesen Wert zurückgeben.

Hinweis: Der gesuchte Wert `N` ist ein Vielfaches von 5.