LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

# Grundlagenpraktikum: Rechnerarchitektur

Gruppe 140 – Abgabe zu Aufgabe A208
Wintersemester 2023/24

Tianhao Gu Zhongfang Wang Julien Escaig

## 1 Einleitung

Im Rahmen dieser Projektaufgabe befassen wir uns mit der Herausforderung, einen Algorithmus in der Programmiersprache C zu entwerfen, der ein farbliches Bild in ein Graustufenbild transformiert und im Anschluss die Helligkeit des Graustufenbilds anpasst. Der Algorithmus lässt sich also in zwei distinkte Phasen unterteilen.

Die erste Phase ist die Graustufenkodierung. In diesem Teil wird als Input eine PPM Datei erwartet. Diese besteht aus einem Header mit Metadaten, worauf die Pixel-Informationen folgen. Im Datenteil der PPM Datei gibt es für jeden Pixel genau drei Werte, die jeweils die Stärke der Farben Rot, Grün und Blau speichern. Je größer der Wert desto stärker ist die Farbe in einem bestimmten Pixel vertreten (Siehe siehe Abb.1)

Im ersten Teil des Algorithmus wird nun diese PPM Datei in eine Bilddatei im PGM Format verwandelt. Eine PGM Datei besitzt ebenfalls einen Header mit Metadaten und darauf folgende Informationen über die Pixel. Da eine PGM Datei nur Graustufen darstellt, wird pro Pixel jeweils nur ein Integer-Wert gespeichert.(siehe Abb2)

Für die Verwandlung von einem Farbbild (PPM) in ein Graustufenbild (PGM) werden jeweils die 3 Werte pro Farbpixel zu einem Graupixel-Wert zusammengefasst. Diese Transformation ist im Grunde ein gewichteter Durchschnitt, der durch die folgende Formel berechnet wird:

$$D(x,y) = \frac{a*R+b*G+c*B}{a+b+c}$$

Kommen wir nun zur zweiten Phase des Algorithmus: Die Anpassung der Helligkeit. Dies passiert durch die sogenannte Gammakorrektur. Um diese besser verstehen zu können, betrachten wir erstmal eine simple Grafik (Abb3):

Das mittlere Bild ist das originale Bild. Wie man leicht erkennen kann, ist nach der Gammakorrektur das rechte Bild dunkler und das linke Bild heller. Die Auswirkung der Gammakorrektur hängt also von der Wahl des y Parameters (auch Gammawert genannt) ab.

Bei diesem Algorithmus wird die Gammakorrektur durch folgende mathematische Formel bestimmt:

$$D'(x,y) = \left(\frac{D(x,y)}{255}\right)^{\gamma} * 255$$

Die Werte D'(x,y) ergeben die Intensität der Graustufenkodierung für alle Pixel (x,y). Bei einem Gammawert kleiner 1, ist D'(x,y) stets kleiner als D(x,y) und das Bild wird

heller. Bei einem Gammawert größer 1, ist D'(x,y) stets größer ist als D(x,y) und das Bild wird dunkler.

Eine weitere Herausforderung dieses Projekt war es diese Gammakorrektur-Formel ohne das Verwenden der Power-Funktion aus der Math Lib zu implementieren. Dafür wurde eine Algorithmus entworfen der nur mit grundlegenden, arithmetischen Operqationen und der Hilfe der Taylorreihe den Wert der Gammakorrektur approximentiert.

## 2 Lösungsansatz

Unser Programm betrachtet aus der Perspektive einer Black Box, nimmt ein Bild im 24bpp PPM-Format entgegen und gibt dieses Bild nach einer Graustufen Konvertierung und Gamma-Korrektur aus, wobei es im PGM-Format gespeichert wird. PGM (Portable Graymap Format) und PPM (Portable Pixmap Format) sind Teil des Netpbm-Bildformats. PPM ist ein einfaches Dateiformat für Farbbitmapbilder. 24bpp (Bits pro Pixel) bedeutet, dass jedes Pixel mit 24 Bits repräsentiert wird, 8 Bits für jede Farbe (nämlich Rot R, Grün G, Blau B).

Unser Programm akzeptiert nur P6-Typ PPM-Eingaben. P6 bezieht sich auf das Binärformat, das im Gegensatz zum ASCII-basierten P3-Typ kompakter und schneller in der Lese-/Schreibgeschwindigkeit ist.Unten ist ein einfaches Beispiel, das dem folgenden Bild entspricht(P6 PPM).

```
p6
4
2
255
0,0,0        255,0,0    0,255,0     0,0,255
255,255,0    255,0,255  0,255,255.  255,255,255
```

Abbildung 1: ein Beispiel für P6 PPM

Abbildung 2: erzeugt durch den Beispielcode

Wir verwenden eine Formel für die Graustufen Konvertierung, um einen gewichteten Durchschnittswert der R-, G- und B-Werte jedes Pixels zu ermitteln und diesen in der PGM zu speichern.Wir wählen PGM, weil es Graustufenbilder speichert. Jeder Pixelwert liegt zwischen 0 und dem maximalen Grauwert, wobei 0 normalerweise Schwarz darstellt, der maximale Grauwert Weiß und die Zwischenwerte verschiedene Grautöne. Der maximale Grauwert ist üblicherweise 255 (8 Bits).Hier muss aber beachtet werden, wie die "Gewichte"von RGB gewählt werden.

Aufgrund der Eigenschaften des menschlichen visuellen Systems (HVS), wie Empfindlichkeit gegenüber verschiedenen Farben und Helligkeiten, könnten diese die optimalen Werte der Parameter beeinflussen. Beispielsweise ist das menschliche Auge empfindlicher für Grün als für Rot und Blau, so dass bei der Umwandlung von Farbbildern in Graustufenbilder das Gewicht der grünen Komponente größer sein könnte als das

der roten und blauen Komponenten.Wir setzen die Standardwerte von a, b, c wie folgt fest:a=0.2126 b=0.7152 c=0.0722 . Dann mitteln wir die RGB-Werte gewichtet.Das Graustufenbild von Bild (a), das durch Anwendung der Standardwerte a,b,c erhalten wird, sieht wie folgt aus:

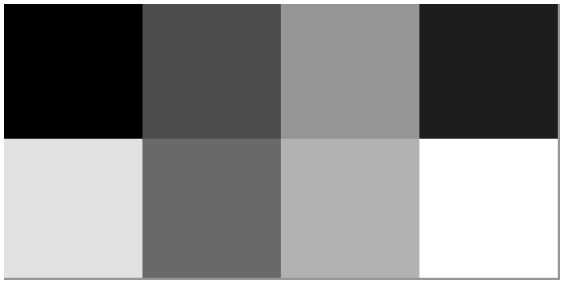

Abbildung 3: Graustufen Konvertierung von Abbildung2

Wir haben die Aufgabe grundsätzlich abgeschlossen, aber um die menschliche Sicht anzupassen, führen wir eine Gammakorrektur durch. Gammakorrektur ist eine Technik zur Anpassung der Helligkeit oder des Kontrastes von Bildern oder Videos. Ihr Zweck besteht darin, dass das Bild im menschlichen Sehsystem natürlicher wirkt. Der Gammawert ist der Parameter, der diese Korrektur steuert. Es ist eine positive Zahl, die normalerweise zwischen 1,0 und 2,2 liegt.

Unser Programm ist jedoch in der Lage, alle Gammawerte größer als 0 zu empfangen.Also je höher der Gammawert, desto höher der Kontrast des Bildes und desto größer die Unterschiede zwischen hellen und dunklen Bereichen. Umgekehrt ist der Kontrast des Bildes bei einem niedrigeren Gammawert geringer und die Unterschiede zwischen hellen und dunklen Bereichen sind geringer. Wir wenden die folgende Formel für jeden Graustufenwert an, wobei Gamma auf 1 gesetzt wird, wenn der Benutzer es nicht angibt:

Unten stehen drei Vergleiche für die Graustufenbild Abbildung3: Gammawert auf 0.1 eingestellt, ohne Gammakorrektur, Gamma-Wert auf 10 eingestellt.

Abbildung 4: Gamma=0.1

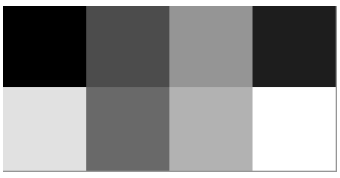

Abbildung 5: Gamma=1

Abbildung 6: Gamma=10

## 3 Genauigkeit

In diesem Teil wird die Genauigkeit bzw. Korrektheit des Algorithmus untersucht. Dafür ist erst einmal zu klären, welches der beiden Konzepte bei dieser Problemstellung sinnvoll anwendbar ist.

Da der Algorithmus zwischendurch mit Float-Werten arbeitet, kann es zu Ungenauigkeiten kommen, da nicht alle Zahlen als Float darstellbar sind. Der maximale Fehler bei 32 bit Float beträgt ungefähr xxx. Diese mögliche Ungenauigkeit wird bei der Gammakorrektur noch verstärkt (Die Ungenauigkeit hängt zudem noch von der internen Implementation der arithmetischen Operationen ab). Obwohl der Fehler immer noch sehr klein ist, kann es bei der Rundung zu Integern zu unterschiedlichen Werten kommen. Für das menschliche Auge macht es keinen Unterschied. Die Bilder sehen identisch aus, obwohl nicht alle Integerwerte in der PGM Datei übereinstimmen. Aus diesen Gründen macht es Sinn, das resultierende Bild auf Genauigkeit anstatt auf Korrektheit zu überprüfen.

Im Folgenden werden die 2 Teile des Programms erstmal getrennt voneinander auf Genauigkeit geprüft. Beginnen wir also mit der Graustufenkodierung.

Um unsere Implementierung der Graustufenkodierung auf Genauigkeit zu überprüfen, verwenden wir die von Netpbm bereitgestellte Funktion “ppmtopgm” als Referenz. Dazu haben wir eine Funktion entworfen die zwei PGM Dateien als Input nimmt und die betragsmäßige Abweichung über alle Pixel berechnet. Die Grafik1 zeigt unsere Ergebnisse für verschiedene PGM Dateien unterschiedlicher Größe.

Diese Abweichungen können jedoch niemals mit bloßem Auge erkannt werden, wie Abbildung 4 demonstriert.

Nun folgt die Genauigkeitsanalyse des zweiten Teils des Programms. Diese wird erneut in mehrere Teile aufgeteilt. Dafür überprüfen wir die Genauigkeit der Pow()-Funktion, die der Taylorreihe und abschließend noch die der SIMD Implementation. Als Referenz der für das Resultat nach der Gammakorrektur verwenden wir die Funktion “gammakorrekt” des Netpbm Packages. In der Grafik2 kann man nun die Abweichung unserer Implementierung von der Netpbm-Implentierung beobachten.

Nun beschäftigen wir uns mit der Genauigkeit der Taylorreihe. Die Taylorreihe ist eine Approximation, die erst bei unendlich vielen Gliedern exakt wird. In unserer Implementation der Taylorreihe brechen wir die Berechnung der Taylorreihe ab, sobald das nächste Reihenglied betragsmäßig kleiner als 0.00000001 ist. Dies lässt vermuten, dass die Abweichung bei dieser Implementation größer sein wird, als beim Verwenden der Pow-Funktion. In Grafik3 ist klar zu erkennen, dass es durch die Approximation der Taylorreihe insgesamt zu einer höheren Abweichung kommt.

Abschließend betrachten wir noch die SIMD Implementation. Es ist zu erwarten, dass es zu keinem Unterschied in der Abweichung kommt, da die gleichen Operationen auf den gleichen Variablen durchgeführt werden. Grafik4 unterstreicht diese Hypothese.

## 4 Performanzanalyse

## 5 Zusammenfassung und Ausblick

Bei[1] diesem Projekt ging es darum, eine Graustufen-Kodierung und Gammakorrektur selber zu programmieren. Das Umsetzen der Graustufen-Kodierung war relativ einfach, genau wie die Implementierung mit der Pow-Funktion. Die Aufgabe erhielt eine neue

Stufe an Komplexität, sobald es zur Implementierung mit der Taylorreihe und der SIMD Implementierung kommt. Die Taylorreihe führt zu einer geringeren Genauigkeit. Die SIMD-Funktion hat zu einer höhere Performanz von etwa ... Prozent geführt. Abschließend kann man sagen, dass dieses Projekt einen interessanten Einblick in verschiedene Bildformate, das Approximieren mit Hilfe der Taylorreihe und das Optimieren mit Hilfe von SIMD-Intrinsics geliefert hat.

## Literatur

[1] `https://www.cs.miami.edu/home/burt/learning/Csc521.101/notes/virtual-memory-notes.html`, 10 2009.