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

Das Ziel dieser Projektarbeit ist es, einen Algorithmus in C zu entwickeln, der ein farbiges Bild in ein Graustufenbild umwandelt und anschließend die Helligkeit des Graustufenbilds mithilfe der Gammakorrektur anpasst.

Als Eingabe akzeptiert unser Programm nur PPM-Dateien [5] des Typs P6. P6 bezieht sich dabei auf das Binärformat der Pixeldaten. 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. Unterhalb ist ein einfaches Beispiel eines solchen Bildes.

```
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

Die erste Phase des Projekts ist die Graustufenkodierung [8]. Dafür verwenden wir die Formel (1) unten. Dabei wird der gewichtete Durchschnitt der Rot-, Grün- und Blau-Werte jedes Pixels ermittelt. Ein Beispiel der Graustufenkodierung ist in Abbildung 3 zu sehen.

$$D(x,y) = \frac{a * R + b * G + c * B}{a + b + c} \tag{1}$$

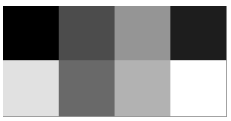

Abbildung 3: Graustufen Konvertierung von Abbildung2

Im zweiten Teil der Aufgabe befassen wir uns nun mit einem anderen Aspekt des menschlichen visuellen Systems (HVS) und zwar der Helligkeit [6]. Diese hat einen großen Einfluss darauf, wie natürlich ein Bild auf uns Menschen wirkt. Die Gammakorrektur ändert die Helligkeit bzw. den Kontrast eines Bildes, und hängt von der Wahl des γ Parameters ab. Bei diesem Algorithmus wird die Gammakorrektur durch folgende mathematische Formel (2) bestimmt. Die Werte D'(x,y) ergeben die Intensität der Graustufenkodierung für alle Pixel (x,y). Ein kleinerer Gammawert führt zu einem helleren Bild, während ein größerer Gammawert zu einem dunkleren Bild führt.

$$D'(x,y) = \left(\frac{D(x,y)}{255}\right)^{\gamma} * 255 \tag{2}$$

Um den Effekt der Gammakorrektur zu visualisieren, betrachten wir nochmal die Abb.3. Unterhalb in Abb4. kann man 3 verschiedene „Helligkeits-Versionen“ von Abb.3 vergleichen.

Abbildung 4: Gamma=0.1

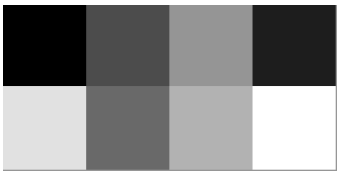

Abbildung 5: Gamma=1

Abbildung 6: Gamma=10

Die Hauptfunktion unseres Programms erhält 6 Parameter. Zwei davon sind `input_file_name` und `output_file_name`, welche die Pfade des Eingabebildes und des Ausgabebildes repräsentieren. Der Parameter `version` vom Datentyp `int` gibt die Version an. Der Parameter `benchmark_number` repräsentiert die Anzahl der Benchmark-Zyklen. Die Parameter `a`, `b`, und `c` vom Datentyp `float` stellen die RGB-Gewichte für die Graustufenumwandlung dar. Schließlich erhält die Funktion noch einen Parameter `_gamma` vom Datentyp `float`, der für den Gammawert steht.

Obwohl die oben genannten Parameter zusätzlich zu den Ein- und Ausgängen standardmäßig mit sinnvollen Defaultwerten besetzt sind, kann der Nutzer diese Parameter auch selbst mit spezifischen und sinnvollen Werten ersetzen. Wenn die gesetzten Werte nicht sinnvoll sind, wird eine Fehlermeldung ausgegeben und das Programm wird beendet. Hier ist eine Übersicht an Optionen, die der Nutzer beim Aufrufen des Programms setzen kann:

- Option *-V<Zahl>* Die Option -V 0 ist die Standardimplementierung. -V 1 steht für die Implementierung V1 mit Taylorreihe. -V 2 steht für die Implementierung V2 mit SIMD. Andere Werte sind nicht erlaubt.
- Option *-B<Zahl>* Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das Argument gibt die Anzahl an Wiederholungen des Funktionsaufrufs an. Es darf nicht kleiner als 1000 sein.

- Positionales Argument <*Dateiname*> Der Pfad zur Eingabedatei ist ein obligatorischer Parameter. Falls dieser ungültig ist, wird eine Fehlermeldung ausgegeben und das Programm wird beendet.
- Option *-o*<*Dateiname*> Der Pfad zur Ausgabedatei ist ein obligatorischer Parameter. Falls dieser ungültig ist, wird eine Fehlermeldung ausgegeben und das Programm wird beendet.
- Option *--coeffs*<*FP Zahl*>,<*FP Zahl*>,<*FP Zahl*> Die drei Argumente müssen stets nicht-negativ sein und dürfen nicht den Wert Infinity und NaN annehmen. Ihre Summe darf auch nicht 0 betragen.
- Option *--gamma*<*Floating Point Zahl*> Der Gammawert kann nicht negativ sein. Wenn diese Option nicht gesetzt wird, wird standardmäßig der Wert 1 verwendet.
- Option *-h* | *--help* Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

2 Lösungsansatz

Nun wird die Implementierung des Programms betrachtet. Zu Beginn erhält das Programm die Optionen und ihre Argumente von der Eingabe des Nutzers durch die Methode `getopt_long`[3]. Nach erfolgreichem Einlesen der Eingabe wird die Funktion `check_values` aufgerufen, um zu überprüfen, ob alle Argumente für das Programm legal sind. Bei illegalen Argumenten wird das Programm mit einer Fehlermeldung beendet.

Die Versionsnummer der Implementierung kann entweder 0, 1 oder 2 sein. Bei den Versionen 0 und 1 werden lediglich sequenzielle Anweisungen verwendet. Version 2 wiederum verwendet SIMD-Instruktionen zur Berechnung der Graustufenkonvertierung.

In allen aufgerufenen Funktionen wird zunächst die Eingabedatei gelesen. Gemäß der Netpbm-Dateibeschreibung ist die ppm-Datei in zwei Teile unterteilt: Metadaten und Pixelbereich. Es wird darauf geachtet, dass in den Metadaten ein Kommentar an beliebiger Stelle eingefügt werden kann. Daher wird beim Lesen der Metadaten in der ppm-Datei ein Zustandsautomat entworfen, um alle möglichen Randfälle abzufangen und zu behandeln. Der Automat wird in Abbildung 7 gezeigt.

Der Speicher für die Eingabedaten und Ausgabedaten wird basierend auf den Metadaten der Eingabedatei allokiert. Bei der Implementierung von V0 und V1 werden die drei Farben jedes Pixels beim Lesen der Eingabedatei in der originalen Reihenfolge gespeichert. Bei der Implementierung von V2 werden beim Einlesen die drei Farben jedes Pixels getrennt gespeichert. Dadurch werden drei Speicherbereiche im Heap erstellt, die jeweils für eine bestimmte Farbe alle Pixel-Werte in fortlaufender Reihenfolge speichern. In der `main`-Funktion wird eine Switch-Anweisung verwendet, um verschiedene Funktionen gemäß der gewählten Versionen aufzurufen.

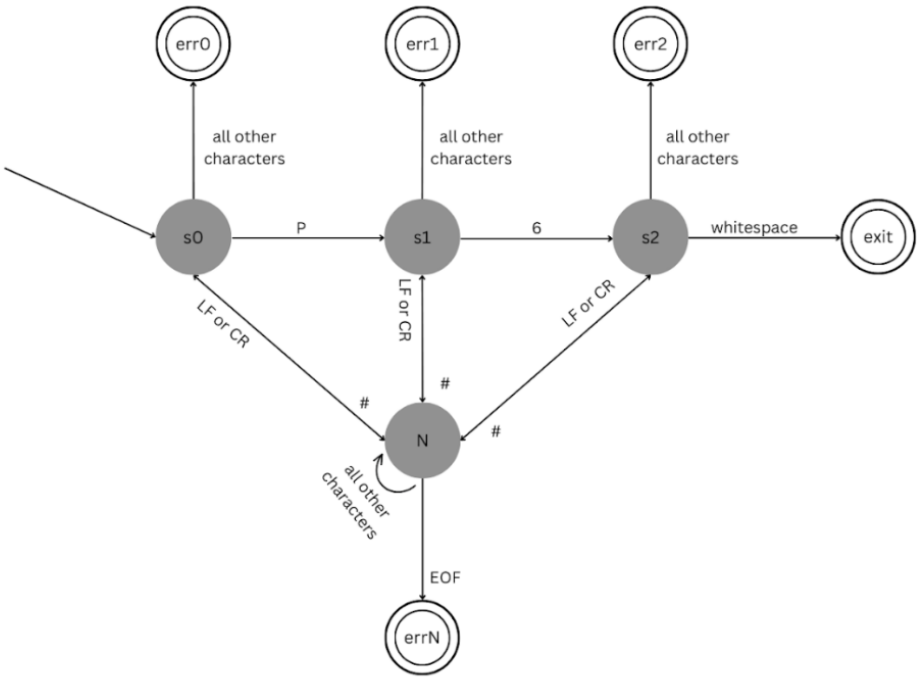

Abbildung 7: Zustandsautomat für das Lesen der Metadaten in der ppm-Datei

In V2, der SIMD-Implementierung, wird `aligned_alloc`[2] verwendet, um Speicher zu reservieren. Die Startadresse des Speichers wird auf ein Vielfaches von 16 ausgerichtet und erhöht damit die Geschwindigkeit des Ladens und Zurückschreibens von und aus dem Speicher in die xmm-Register. Anschließend wird die ausgewählte Version von gamma_correct aufgerufen. In V0 und V1 wird die Graustufenkonvertierung für jeden Pixel sequentiell berechnet. V2 verwendet SIMD-Anweisungen, welche es ermöglichen die Graustufenwerte von vier Pixeln gleichzeitig zu berechnen. Hier noch eine Bemerkung: Wir setzen die Standardwerte von a, b, c wie folgt fest: a=0.299 b=0.587 c=0.114. Diese Gewichtungen werden häufig verwendet[8], da sie in der Praxis gute Ergebnisse liefern und die Luminanzeigenschaften der Farben natürlich für das menschliche Auge wirken. Nachdem die Graustufenwerte der Pixel berechnet wurden, verwenden V0 und V2 die `pow`-Funktion der `math.h` Bibliothek, um die Graustufenwerte nach der Gammakorrektur für jedes Pixel sequentiell zu berechnen. V1 verwendet keine Bibliotheksfunktionen. Im nächsten Teil werden wir sehen, wie V1 grundlegende mathematische Operationen verwendet um die pow Funktion durch die Taylor-Entwicklung zu approximieren.

Die Taylor Entwicklung [9] dient der lokalen Approximation eines Funktionswertes und hat die allgemeine Form $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \frac{f'''(a)(x-a)^3}{3!} + \ldots + \frac{f^n(a)(x-a)^n}{n!} + R_n(x)$ Wir verwenden den Fall $a = 1$ und erhalten dann folgende Formel:

$$f(x) = 1 + \frac{(x-1)^1}{1!}(\gamma) + \frac{(x-1)^2}{2!}(\gamma)(\gamma-1) + \cdots + \frac{(x-1)^n}{n!}\prod_{i=0}^{n-1}(\gamma - i) + R_n(x) \quad (3)$$

Betrachten wir zunächst drei mögliche Grenzfälle:

1. Wenn die Basis kleiner oder gleich 0.5 ist und Gamma größer oder gleich 150 ist, ist das Ergebnis immer 0 (**Underflow** für `float`).
2. Wenn die Basis 1.0 beträgt, ist das Ergebnis immer 1.
3. Wenn Gamma größer ist als die magische Zahl 67075968, ist das Ergebnis 0, wenn die Basis kleiner als 1 ist (wieder **Underflow**).

Eine weitere logische Schlussfolgerung für Formel (2) ist, dass die Basis $D(x, y)/255$ immer eine Zahl zwischen 0 und 1 ist.(255 ist maxValue) *Gamma* ist laut Angabe so zu wählen, dass es größer als 0 und nicht *Infinity* oder *NaN* ist. Bei der Entwicklung haben wir festgestellt, dass die Berechnung von $\prod_{i=0}^{n-1}(\gamma - i)$ bei größeren Gammawerten leicht zu einem **Overflow** führt. Um dieses Problem zu lösen, zerlegt V2 *Gamma* in eine ganze Zahl und eine Dezimalzahl und berechnet diese separat.

Für den Ganzzahlanteil ist unser Design Gedanke, den Exponenten in eine binäre Zahl umzuwandeln und dann zu zerlegen. Im folgenden Beispiel wird die 29. Potenz von 0,79 berechnet

$$29 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4$$
$$0.79^{29} = 0.79^{2^0} \cdot 0.79^{2^2} \cdot 0.79^{2^3} \cdot 0.79^{2^4}$$

Da Gamma kleiner als 67075968 ist, also zwischen 2^{25} und 2^{26} liegt, benötigen wir unabhängig vom Exponenten höchstens 26 Schleifen.

Danach verwenden wir Formel (3) für den Dezimal-Anteil, und die Akkumulation endet, wenn der Expansionsterm klein genug ist. Hier ist zu beachten, dass wir das Ergebnis des i-ten Expansionsterms zur Berechnung des (i+1)-ten Expansionsterms verwenden können.

Die Graustufenwerte nach der Gammakorrektur werden über einen weiteren Funktionsaufruf im Format von PGM in die Ausgabedatei geschrieben. PGM (Portable Gray Map) Format[5] wird zur Speicherung von Graustufenbildern verwendet, weil es ein einfaches, leicht umsetzbares Bildformat ist. Im Vergleich zu anderen komplexen Bilddateiformaten ist das PGM-Format sehr einfach. Es speichert die Pixelwerte des Bildes und kann leicht in Textform bearbeitet und gelesen werden. Dieses Format erfordert keine komplizierten Dekodierer oder zusätzliche Datenkompression, was es ideal für die schnelle Speicherung und Verarbeitung von Graustufenbilddaten macht. Bevor alle Ressourcen freigegeben werden, wird `gamma_correct()` wiederholt ausgeführt, wenn der Benutzer die Option `-B` in den Optionen aktiviert hat. Die Zeit wird zu Beginn und am Ende des Zyklus aufgezeichnet. Diese Laufzeit des Zyklus wird später für die Leistungsanalyse verwendet.

Am Ende des Programms müssen alle Ressourcen freigegeben werden. Zusätzlich wird an sämtlichen Stellen im Programm, an denen eine Allocation oder eine IO-Operation durchgeführt wird, überprüft, ob sie erfolgreich war.

Das Programm verteilt alle Funktionen sinnvoll auf die verschiedenen Funktionen. Jede Funktion ist angemessen groß, logisch korrekt, leicht zu warten und vermeidet Blob Muster.

3 Genauigkeit

In diesem Teil der Ausarbeitung beschäftigen wir uns mit der Genauigkeit unserer Implementierung.

Schauen wir uns die Ungenauigkeit an, die entstanden wäre, wenn das Programm in Version 2 anstelle von pow in stadard lib die Formel (3) verwendet hätte. Dies liegt daran, dass wir lediglich eine beschränkte Anzahl von Termen der Taylorreihe berechnen können. Dadurch werden Terme ab dem (n+1)-ten vernachlässigt, was zu Genauigkeitsverlusten führt. Zur Schätzung der Präzision ziehen wir das Peano'sche Restglied[4] heran.

$$o\left((x-1)^n\right)$$

Es zeigt sich, dass das Restglied bei einem Wert von x nahe null langsam und bei x nahe eins schneller konvergiert. Eine präzisere Beschreibung ermöglicht das Lagrange'sche Restglied[1], insbesondere da der Exponent ein Bruchteil des Gammas ist und wir uns auf Werte zwischen 0 und 1 konzentrieren. Wir setzen einen Anpassungspunkt von 1 ein und vereinfachen die Lagrangesche Gleichung wie folgt:

$$\frac{\prod_{i=0}^{n}(\gamma - i) \cdot (x-1)^{n+1}}{(n+1)!}$$

Wir werden dies im Folgenden unter zwei Gesichtspunkten analysieren: Zum einen soll untersucht werden, wie viele Konvergenzterme erforderlich sind, um sich dem minimalen Wert von float zu nähern, und zum anderen soll untersucht werden, welche Kombinationen von Gamma und x zu Fehlern führen, die größer als $1/255$ und welche kleiner als $1/255$ sind. Der Fehler des endgültigen gamma-korrigierten Ergebnisses beträgt höchstens 1, wenn der Fehler weniger als $1/255$ beträgt.

Beginnen wir mit der ersten Perspektive, bei der wir gamma bzw. x fixieren und die Auswirkungen einer Änderung einer anderen Variablen auf die Anzahl der Iterationen untersuchen. Die Ergebnisse werden mit der Software Wolframalpha berechnet. Die Schlussfolgerung ist in Abb.8 und Abb.9 dargestellt.

Daraus folgt, dass die Auswirkung von Gamma linear und die von der Basis krummlinig ist. Wenn also die Basis nahe null liegt, konvergiert sie langsam, was ebenfalls zu größeren Ungenauigkeiten führt.

Anschließend wird die Auswirkung der Änderung von (Gamma,x) auf den Fehler weiter untersucht. Hier verwenden wir den Referenzstandard netpbm/v0. Die Schlussfolgerung ist in Abb.10 dargestellt.(todo by zhongfang)

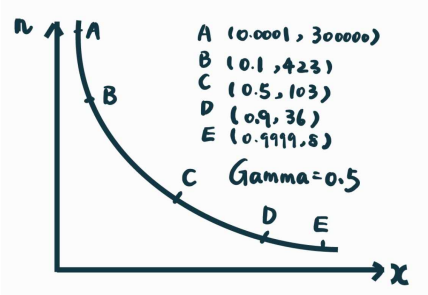

Abbildung 8: Anzahl der erforderlichen Iterationen für verschiedene x, wenn gamma auf 0,5 festgelegt ist

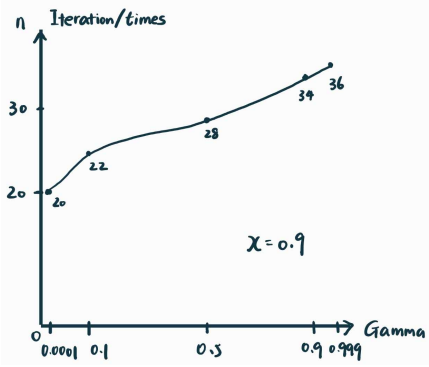

Abbildung 9: Für verschiedene Gamma, wenn x auf 0,9 festgelegt ist

Die Ergebnisse der Ausführung unseres Programms bestätigen auch die Schlussfolgerungen der mathematischen Berechnungen, d.h. der Wert von Gamma hat einen geringen Einfluss auf den Fehler, während x sehr nahe bei 0 zu einem großen Fehler führt.

Aber wirkt sich das sehr negativ auf unsere Ergebnisse aus?In Anwendungen tatsächlich nicht wirklich schlimm.Begründen ist wie folgt:

1. Die Wahrscheinlichkeit ist sehr gering, wenn die Basis kleiner als 0,0001 ist.
2. Wenn die Basis klein und gleichzeitig Gamma groß ist, wird dies bereits als Grenzfall berücksichtigt.
3. Wenn das Gamma einen ganzzahligen Teil enthält, ist das Ergebnis des ganzzahligen Teils kleiner als 1, weil die Basis kleiner als 1 ist. Der absolute Fehler aus dem gebrochenen Teil der Berechnung wird reduziert, wenn wir das Ergebnis mit dem ganzzahligen Teil multiplizieren.

4 Performanzanalyse

Im vorletzten Teil der Ausarbeitung untersuchen wir nun die Leistung unserer verschiedenen Implementierungen. Dabei analysieren wir bezogen auf die Laufzeit und den Speicherbedarf.

Eine theoretische Laufzeitanalyse ist für keine der drei Implementierungen praktisch umsetzbar. Bei Version 0 und Version 2 wird die pow()-Funktion der Math.h - Bibliothek verwendet. Dabei handelt es sich um eine Blockbox-Funktion, deren Laufzeit wir nicht theoretisch bestimmen können. Bei Version 1 findet eine Approximation durch die Taylorreihe statt. Dabei wird die Reihe abgebrochen, sobald das neueste Reihenglied betragsmäßig kleiner als ein bestimmtes Epsilon ist. Dies hat zur Folge, dass es praktisch nicht möglich ist vorherzusagen, wie viele Glieder im Durchschnitt berechnet werden

müssen, da die Größe und Entwicklung der Reihenglieder stark von Basis und Exponent abhängen.

Für die praktische Analyse der Laufzeit wurden die 3 Versionen des Algorithmus auf einem Gerät mit folgender Spezifikation evaluiert:

- AMD Ryzen 5 3600 6-Core Processor 3.59 GHz
- 8.00 GB DDR4 RAM
- ...

In der Grafik1 sehen wir das Resultat der Laufzeitanalyse für die 3 verschiedenen Versionen bei unterschiedlichen Benchmarking-Zyklen.

GRAFIK1 - TODO

Die Version 2 war wie zu erwarten langsamer als Version 1, da sie nicht die Pow-Funktion der Library benutzt, sondern selbst die Taylorreihe implementiert. Durch das Verwenden von SIMD Anweisungen konnte die Laufzeit des Programms um etwa 30 Prozent verbessert werden.

Bei der Analyse des Speicherbedarfs verwenden wir das Tool “Valgrind”. Dabei handelt es sich um eine Sammlung von Programmierwerkzeugen, die unter anderem die Speichernutzung eines Programms protokollieren kann. Beim Ausführen unseres Programms mit 9 verschiedenen Eingabe-Bildern unterschiedlicher Größe kamen wir zu folgenden Ergebnissen(Grafik2).

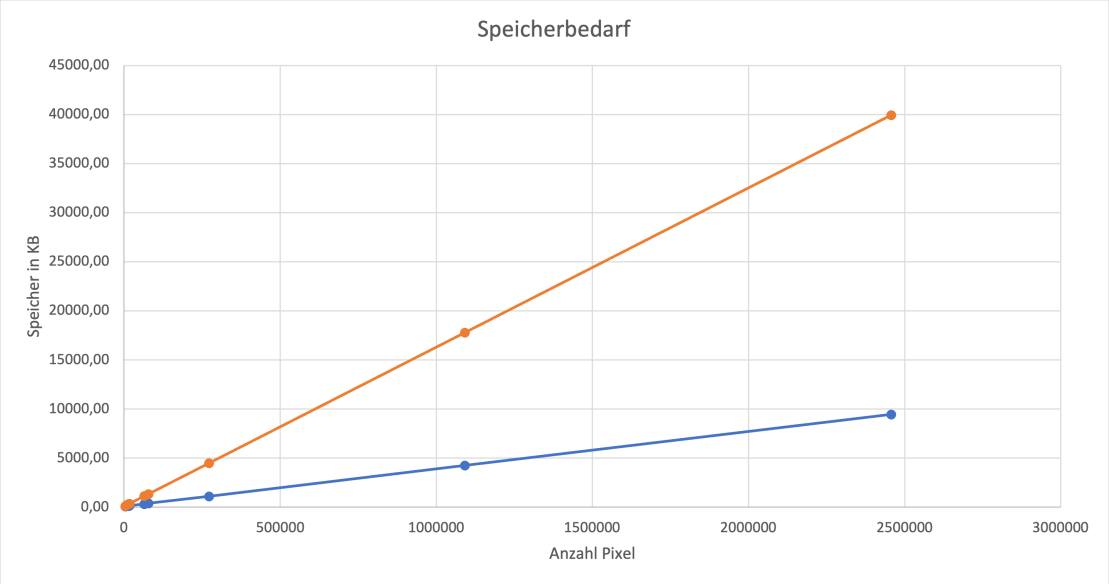

Abbildung 10: Speicherbedarf2

Für alle 3 Implementierungen kann man einen linearen Zusammenhang zwischen Bildgröße und Speichernutzung beobachten. V0 und V1 haben jeweils sehr ähnliche

Resultate geliefert, weshalb sie in Grafik 2 zusammengefasst wurden. Beide Implementationen haben ungefähr eine Speichernutzung von 3,7 KB pro 1000 Pixel. Die Implementierung V2 verbraucht mit 16 KB pro 1000 Pixel im Vergleich deutlich mehr Speicher.

5 Zusammenfassung und Ausblick

Bei 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.

Außerdem erörterten die Diskussionsteilnehmer, wie das Problem der langsamen Iteration gelöst werden kann, wenn x klein ist; eine Lösung ist die Verwendung von Lookup Table[7].

todo by zhongfang: explain how to use lookup table to solve the problem of slow convergence when x is small

Literatur

[1] biancahoegel.de. Taylor-formel. `https://www.biancahoegel.de/mathe/analysis/taylor-formel.html`, visited 2024-02-02, January 2021.

[2] GNU. aligned_alloc(3) - linux man page. `https://man7.org/linux/man-pages/man3/aligned_alloc.3.html`, visited 2024-01-25, March 2008.

[3] GNU Project. *GNU C Library: getopt_long(3)*. GNU, December 2022. `https://man7.org/linux/man-pages/man3/getopt_long.3.html`, visited 2024-02-01.

[4] Mathematik.net. Reihen und taylor-polynome. `https://www.mathematik.net/reihen-taylor-polynome/tp3s20.htm`, visited 2024-02-01, December 2009.

[5] Netpbm community. Netpbm: Open-source graphics tools and file formats. `https://netpbm.sourceforge.net/doc/`, visited 2024-02-02, August 2020.

[6] Wikipedia contributors. Gammakorrektur — wikipedia, die freie enzyklopädie. `https://de.wikipedia.org/wiki/Gammakorrektur`, visited 2024-02-01, December 2023.

[7] Wikipedia contributors. Lookup table — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Lookup_table`, visited 2024-02-01, December 2023.

[8] Wikipedia contributors. Grayscale — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Grayscale`, visited 2024-01-31, January 2024.

[9] Wolfram Research. Taylor series — from wolfram mathworld. `https://mathworld.wolfram.com/TaylorSeries.html`, visited 2022-02-01, January 2024.