LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 158 – Abgabe zu Aufgabe A206
Wintersemester 2023/24

Lukas Li Yassine Hmidi Constantin Carste

1 Einleitung

Im Bereich der digitalen Bildverarbeitung führt die Schnittstelle zwischen mathematischen Prinzipien und realer Anwendung zu vielen innovativen Algorithmen, die den jetzigen Stand der Technik verbessert. Dieses Projekt befasst sich mit der Bildmanipulation und spezialisiert sich insbesondere auf der Umwandlung von Farbbildern in Graustufen und eine anschließende Skalierung durch bilineare Interpolation. Dazu werden theoretische Erkenntnisse aus der Mathematik genutzt, um einen praktischen C-Algorithmus zu entwickeln.
Jedes Farbbild besteht aus Pixeln, die durch ihre Position (x, y) eindeutig identifiziert sind und durch ihren Farbvektor (R, G, B) definiert werden. Die erste Herausforderung besteht darin, diese Farbpixel in Graustufen umzuwandeln. Dabei wird ein gewichteter Durchschnitt der Einträge des Farbvektors mit geeigneten Koeffizienten a, b und c berechnet. Bei der Berechnung der Grauwerte wird Rücksicht auf die Wahrnehmung des menschlichen visuellen Systems (HVS) genommen, damit das entstandene Graustufenbild besonders ansprechend für die menschlichen Augen ist. Anschließend wird die bilineare Interpolation angewendet, um das Bild passend zu skalieren.

Die theoretische Grundlage dieses Projekts basiert auf dem Verständnis des 24bpp PPM-Bildformats, welches als Eingabe erwartet wird, sowie auf den mathematischen Berechnungen der Graustufenkonvertierung und der bilinearen Interpolation.
Auf der praktischen Seite erstreckt sich die Aufgabe über die Umsetzung der theoretischen Erkenntnisse in der Programmiersprache C. Die Implementierung des Algorithmus umfasst ebenso ein Rahmenprogramm, das PPM-Dateien und weitere optionale Parameter über die Kommandozeile entgegennimmt und das Ergebnis der Berechnung als ein PGM-Bild abspeichert. Durch die I/O-Funktionen des Rahmenprogramms wird eine praktische Einbindung des Resultats in weitere Projekte und das Anpassen des Algorithmus an verschiedene Szenarien ermöglicht.
Während der Ausarbeitung des Projektes wird insbesondere der Fokus auf die Optimierung gelegt – theoretisch durch algorithmische Optimierung und praktisch durch Parallelisierung und SIMD.

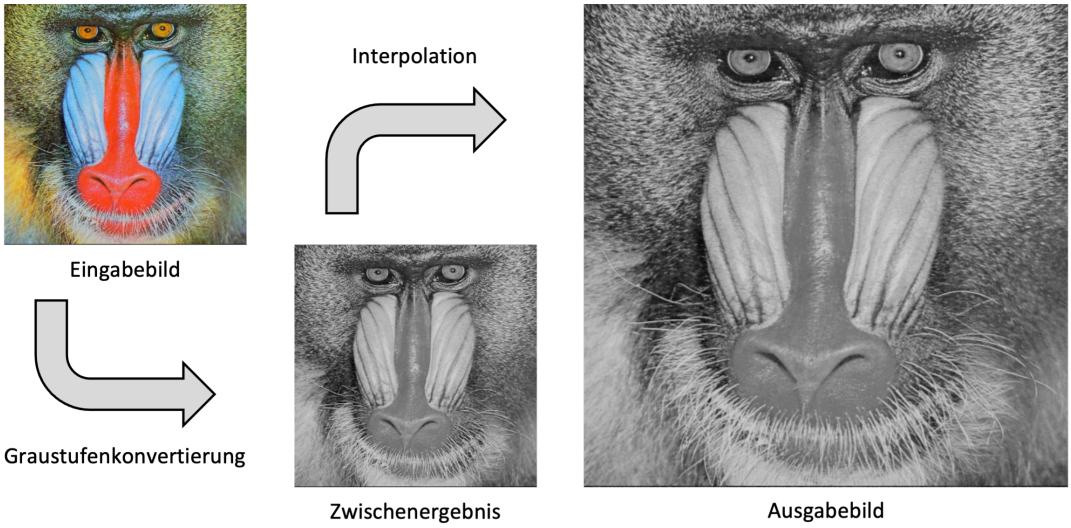

Abbildung 1: Verlauf der Eingabe zur fertigen Ausgabe

2 Lösungsansatz

Das Programm muss die Nutzereingaben korrekt erkennen und bei fehlerhaften Eingaben sinnvolle Alternativwerte verwenden.
Die eigentliche Konvertierung wird in drei Teilsektionen aufgeteilt. Die Graustufenkonvertierung konvertiert die PPM-Eingabe zu einem PGM-Bild, wobei die dafür vorgesehene Rundung nicht vorgenommen wird, die Interpolation skaliert das Grayscaling-Resultat mit dem vom Benutzer angegebenen Faktor und gibt dieses aus.

2.1 Netpbm-Format

Das Netpbm-Format enthält neben ppm auch das pgm-Bildformat. Dabei wird die Ausgabe ein P5 PGM sein. Die Entscheidung für ein PGM-Bild als Ausgabedatei lässt sich damit begründen, dass PGM für Graustufenbilder optimiert ist und nur ein Drittel der Speicher verbraucht wie das gleiche Bild als PPM abzuspeichern. Dabei verwenden PGM und PPM den gleichen Header (bei PPM P6 und bei PGM P5 als magischer Nummer). Anschließend speichert ein PPM-Bild für jeden Pixel die drei Farbkanäle Rot, Grün und Blau ab. Da beim Graustufenbild für alle drei Kanäle den gleichen Wert verwendet wird, trifft PPM-Bild die Vereinfachung, für jeden Pixel nur einen Wert abzuspeichern und braucht somit viel weniger Speicher als PPM.

2.2 Rahmenprogramm

Im Zuge der Implementierung war es nötig, ein Rahmenprogramm für die einfache Benutzererfahrung zu erstellen. Dabei wurde besonders viel Rücksicht auf die Benutzerfreundlichkeit genommen. Bis auf das Eingabebild sind alle Parameter optional

und müssen nicht gesetzt werden. Dabei wird als Eingabebild ein 24bpp P6 ppm Bild erwartet (das heißt, für jeden Farbkanal Rot, Grün und Blau darf nicht mehr als ein Byte verwendet werden, außerdem darf die Luminanz nicht gesetzt werden). Falls die Parameter nicht gesetzt oder falsch sind wie beispielsweise ein ungültiger Dateiname für Ausgabedateien, wird nach einer kurzen Warnung auf der Konsole auf Standardwerte zurückgegriffen. Bei fehlenden Parameter für das Eingabebild oder ungültiges Eingabebild wie falsches Bildformat wird das Programm mit einer Fehlermeldung terminiert. Die Funktion $interpolate(...)$ speichert das Graustufenbild in tmp und das interpolierte Bild in result ab. Anschließend schreibt das Rahmenprogramm das Bild als ein P5 PGM-Bild unter den gesetzten Parameter in den Speicher.

2.3 Graustufenkonvertierung

Die Graustufenkonvertierung sieht vor, das vom Benuter angegebene PPM Bild in ein PGM-Bild umzuwandeln. Ein PPM-Bild hat für jeden Pixel drei Werte gegeben (R, G, B), ein PGM-Bild nur einen, da das Bild für alle drei Kanäle die gleichen Werte verwendet. Die R, G und B Werte müssen also mit Koeffizienten (a, b und c) zu einem einzelnen Wert D gewichtet werden.

$$D_{neu}=\frac{a \cdot R + b \cdot G + c \cdot B}{a + b + c} \tag{1}$$

Um jede Konvertierung zu optimieren, muss $a + b + c = 1$ gegeben sein, da dadurch eine Division für jeden Pixel gespart werden kann. Die Koeffizienten werden also vor der Konvertierung miteinander skaliert.

$$D_{neu} \underset{a+b+c=1}{=} a \cdot R + b \cdot G + c \cdot B \tag{2}$$

2.3.1 Koeffizientenwahl

Die Standard-Koeffizienten $a = 0.299(\approx \frac{2}{7}), b = 0.587(\approx \frac{4}{7})$ und $c = 0.144(\approx \frac{1}{7})$ sind aufgrund der Annahme, dass das menschliche Auge Grün stärker als Rot und Rot stärker als Blau wahrnimmt, getroffen worden. Damit ergibt sich ein dem menschlichen Wahrnehmungssystem besonders ansprechende Graustufenbild. Diese Koeffizienten addieren somit standardgemäß bereits auf 1 auf, wodurch die Konvertierung übersprungen werden kann.

2.3.2 Naive Konvertierung

Die naive Konvertierung sieht vor, jeden Pixel einzeln zu berechnen und in einem Float-Array abzuspeichern. Dadurch wird mit verschiedenen Graustufenintensitäten die Helligkeitsstufen im ursprünglichen Farbbild sichergestellt, wodurch das resultierende Bild mehr Details erhält.

2.3.3 Konvertierung mit Tabellen

Für diesen Konvertierungsansatz werden vorberechnete Tabellen verwendet, die jedes mögliche Ergebnis der Multiplikation von 0 bis 255 (bzw. dem maximalen Farbwert) mit den entsprechenden Koeffizienten enthalten. Aufgrund der Größenlimitierung durch den Farbwert könnte ihre Zuweisung lokal im Stack erfolgen. Die Idee besteht darin, dass wir die Berechnung des Zwischenmultiplikationswertes für redundante Farbkanäle zu vermeiden. Statt der Multiplikation werden die Werte in der Tabelle nachgeschlagen und zum endgültigen D_{neu} addiert.

2.3.4 Konvertierung mit SIMD

In unserer Implementierung wird SIMD eingesetzt, um vier Pixel gleichzeitig zu verarbeiten und so die Effizienz gegenüber einer sequenziellen Bearbeitung zu steigern. Da SIMD ausgerichtete Daten voraussetzt, werden die letzten Pixel individuell und nicht mit SIMD verarbeitet.

Algorithm 1 Grayscale conversion using SIMD

$coef \leftarrow ((a,a,a,a),(b,b,b,b),(c,c,c,c))$ ▷ Init coeffs
$end \leftarrow h \cdot w \cdot 3 - (h \cdot w \cdot 3 \mod 12)$
for $i \leftarrow 0$ **to** end **step** 12 **do** ▷ Aligned data
    $vecR \leftarrow (R_1, R_2, R_3, R_4)$
    $vecG \leftarrow (G_1, G_3, G_3, G_4)$
    $vecB \leftarrow (B_1, B_3, B_3, B_4)$
    $Ra, Gb, Bc \leftarrow vecR \cdot coef[0], vecG \cdot coef[1], vecB \cdot coef[2]$
    $D \leftarrow Ra + Gb + Bc$ ▷ Grayscale
    $res[i/3] \leftarrow \text{Round}(D)$ ▷ Rounding
end for
for $i \leftarrow end$ **to** $h \cdot w \cdot 3$ **step** 3 **do** ▷ Remaining
    $res[i/3] \leftarrow \text{round}(a \cdot img[i] + b \cdot img[i+1] + c \cdot img[i+2])$
end for

2.4 Interpolation

Das Ziel der Interpolation ist es, eine Bildskalierung durchzuführen, also ein Bild so zu vergrößern, dass das menschliche Auge möglichst wenig Unschärfe erkennen kann. Hierfür werden einzelne Pixel mit einem Skalierungsfaktor k voneinander verschoben, wodurch Löcher von $k-1$ Pixeln im neu generierten Bild entstehen, welche mit Zwischenwerten aufgefüllt werden. Zwei zuvor aneinandergrenzende Pixel mit $k = 2$, einer schwarz, der andere weiß, sollten also in einem grauen Pixel resultieren. Da die Verschiebung in der Höhe sowie Breite durchgeführt wird entsteht pro Pixel ein neues Quadrat der Größe $k \cdot k$, mit dem ursprünglichen Pixel in der oberen linken Ecke. Durch Ausweitung des Quadrats auf $(k+1) \cdot (k+1)$ erhält man ein Quadrat, dessen Eckpunkte bekannt sind. Anhand der bekannten Pixel kann man die restlichen Pixel

mit der folgenden Formel berechnen und befüllen. Für die mathematische Berechnung werden die Pixel in einem Koordinatensystem mit $x, y \in [0, k]$ angegeben. Also haben die vier originalen Eck-Pixel folgende Koordinaten: $(0, 0)$, $(0, k)$, $(k, 0)$ und (k, k). Für die unbekannten zu interpolierenden Pixel an (x, y) gilt dann die Formel

$$Q_{neu}=\frac{1}{(k)^2} \cdot \begin{pmatrix} k-y & y \end{pmatrix} \cdot \begin{pmatrix} Q_{(0,0)} & Q_{(0,k)} \\ Q_{(k,0)} & Q_{(k,k)} \end{pmatrix} \cdot \begin{pmatrix} k-x \\ x \end{pmatrix} \tag{3}$$

mit (x, y) als Koordinaten des zu berechnenden Pixels. Diese Funktion funktioniert für fast alle Quadrate, mit Ausnahme der rechten und unteren Randpixel, da diese keine Nachbar-Pixel haben, auf die das Quadrat ausgeweitet werden kann. Um die Interpolation weiter durchzuführen müssen also sinnvolle Alternativwerte gefunden werden.

2.5 Interpolation der Randpunkte

Bei der Interpolation der rechten sowie unteren Randpixel fehlen jeweils zwei bis drei der vier benötigten Eckpunkte (letzteres nur in der unteren rechten Ecke). Für diese Interpolation müssen also Alternativwerte gesucht werden.
Zum Verständnis sehen wir das Bild als Zylinder an. Dadurch berühren sich zwei Kanten, als Beispiel die linke und rechte Kante, und löst das Problem der unbekannten Eckpunkte für eine Kante auf. Im Klartext füllt man somit die unbekannten Eckpunkte mit den ersten Eckpunkten der aktuell bearbeitenden Reihe / Spalte. Dieser Ansatz würde jedoch beispielsweise bei einem Verlaufsbild von schwarz nach weiß zu einer ungewollten rechten Kante führen. Die meisten Bilder, haben jedoch einen gleichfarbigen Farbverlauf von links nach rechts (wie beispielsweise der Horizont eines Portraits), der unwahrscheinliche Fall eines Verlaufsbildes kann also in Kauf genommen werden.

2.6 Interpolationsansätze

2.6.1 Naiver Interpolationsansatz

In unserem ersten Interpolationsansatz iterieren wir über jedes Pixel des skalierten Bildes. Dabei werden Zwischenwerte, die für verschiedene x-Indizes bei gleichem y-Index gleich bleiben, im Voraus berechnet. Wir ermitteln die benötigten Pixelwerte aus dem Eingabebild in Graustufen, indem wir die entsprechenden Indizes berechnen. Anschließend führen wir die Interpolationsberechnung durch, konvertieren das Ergebnis und speichern es als $uint8\_t$-Wert im Ausgabebild. Dieses Format entspricht dem PGM-Format unseres Ausgabebildes.

2.6.2 Algorithmisch optimierter Interpolationsansatz

Der optimierte Interpolationsansatz bearbeitet das neue Bild in Sektoren - also in Quadraten der Größe $k \cdot k$, statt $(k + 1) \cdot (k + 1)$, da das in einer doppelten Berechnung der unteren und rechten Kante resultieren würde. Dabei sind immer drei der bekannten

Randpunkte außerhalb des eigentlichen Quadrates, mit dem oberen linken Randpunkt als bekannter Pixel. Für jedes Quadrat werden die Eckpunkte aus dem PGM Bild ausgelesen und in Variablen abgespeichert, dass nicht jeder neue Pixel erneut auf die vier Pixel zugreifen muss. Daraufhin wird die Funktion für jeden Pixel ausgeführt - der resultierende Float wird gerundet und an korrekter Stelle im neuen Bild abgespeichert. Der uns bereits bekannte obere linke Pixel wird nicht berechnet sondern direkt eingesetzt. Falls der Skalierungsfaktor Eins entspricht, muss dieser Ansatz nicht ausgeführt werden beziehungsweise kann zu Beginn terminiert werden, da sich das Bild nicht verändert.

2.6.3 Algorithmisch weiter optimierter Interpolationsansatz

Der weiter-optimierte Interpolationsansatz basiert ebenfalls auf Sektoren. Die Idee hierbei ist, dass in horizontaler Richtung folgende Funktion genutzt werden kann, um die Werte zu berechnen. In der folgenden Funktion sei $k =$ Skalierungsfaktor, $x =$ Wert im Quadrat

$$Q_x{=}Q_0 \cdot \frac{k-x}{k} + Q_k \cdot \frac{x}{k} \tag{4}$$

Dieser Ansatz wird auf die vertikale Richtung weitergeführt, wobei die entsprechenden Q_0 und Q_k Werte berechnet werden müssen, da sie im neuen Bild bereits gerundet wurden. Die Berechnung der Werte unterscheidet sich dadurch von dem zweiten Ansatz eigentlich nur in der Reihenfolge.

2.6.4 Interpolation mit SIMD

Die Idee hier ist, dass wir das eingegebene Graustufenbild durchiteriert wird und die Tatsache ausgenutzt wird, dass jeder Pixel für einen Sektor als $Q_{0,0}$ interpretiert wird. Deshalb werden vier $Q_{0,0}$-Werte, die 4 aufeinanderfolgenden Sektoren entsprechen geladen. Das Gleiche gilt für die $Q_{s,0}$, wobei jetzt jede Zeile jeweils bei $x = 1$ beginnt. Das gleiche Konzept gilt für c und d, ausgenommen der ersten Linie. Die letzte Zeile und letzte Spalte, die zu den nicht ausgerichteten Daten hinzugefügt wurden, müssen einzeln behandelt werden.

3 Genauigkeit

3.1 Rechnen mit Floats

Im allgemeinen wird bei der Graustufenkonvertierung sowie Interpolation mit Floats gearbeitet. Floats sind auf 32 Bit begrenzt und für ihre Ungenauigkeit bei der Multiplikation bekannt. Im Falle der Koeffizienten für die Graustufenkonvertierung können die Endresultate für die Optimierung $a + b + c = 1$ meistens nicht perfekt in Floats gespeichert werden. Als Alternative könnte man auf Doubles zurückgreifen, was aber einen Eingriff in den SIMD-Prozess aufgrund größerer Datenstrukturen ineffizient machen würde.

Algorithm 2 SIMD-Interpolation Implementierung

$breiteNeu \leftarrow breite \times faktor$
$invFaktor \leftarrow 1/faktor$
Vorbereiten der Zwischenwerte außerhalb der Schleifen
for $y \leftarrow 0$ **to** $hoehe - 2$ **do**
    for $x \leftarrow 0$ **to** $(breite - 1) - ((breite - 1)\%4)$ **do**
        $Q00 \leftarrow$ Lade 4 aufeinanderfolgende Pixel
        $Qs0 \leftarrow$ Lade 4 aufeinanderfolgende Pixel startend von $x + 1$
        $Q0s \leftarrow$ Lade 4 Pixel von Zeile
        $Qss \leftarrow$ Lade 4 diagonalen Pixel startend von $x + 1$
        for $i \leftarrow 0$ **to** $faktor - 1$ **do**
            for $j \leftarrow 0$ **to** $faktor - 1$ **do**
                $a, b, c, d \leftarrow$ Lade Zwischenwerte mit Shuffle
                $interpolated \leftarrow Q00 * a + Qs0 * b + Q0s * c + Qss * d$
                Speichere gerundet und konvertierte interpolierte Werte
            end for
        end for
    end for
end for
Verarbeite Randpunkte mit nicht-SIMD-Prozess

3.2 Graustufenkonvertierung

Bei der Entwicklung der Methoden zur Graustufenkonvertierung war es wichtig, eine einheitliche Rundungsstrategie zu verfolgen, um Konsistenz über alle Implementierungen hinweg zu gewährleisten. Bei der Anwendung der Lookup-Tabellen und der naiven Interpolationsmethode wurde dies beachtet.
Im Fall der SIMD-Implementierung musste aufgrund der fehlenden direkten Entsprechung einer exakten Rundungsmethode für einzelne Fließkommazahlen ein alternativer Ansatz gewählt werden. Hierbei wurde auf jeden Wert 0.5 addiert und die Fließkommazahl durch $\_mm\_floor\_ps$ abgerundet. Diese Vorgehensweise kann im Vergleich zu anderen Methoden leicht langsamer sein, sichert jedoch eine exakte Rundung und garantiert, dass alle Konvertierungen unabhängig von der Eingabe konsistente und präzise Resultate liefern. Obwohl diese Methode möglicherweise nicht die schnellste ist, wurde sie gewählt, um die Genauigkeit zu maximieren und zu gewährleisten, dass die Ergebnisse exakt den mathematischen Erwartungen entsprechen.

3.3 Interpolationsrundungen

Die von der Interpolation berechneten neuen Pixel sind in den meisten Fällen Werte mit Nachkommastellen (Beispiel: $k = 2$ mit $P_{0,0} = 10$, $P_{1,0} = 11$ ergibt $Q_{0,0} = 10$, $P_{1,0} = 10.5$, $P_{2,0} = 11$, ...). Der resultierende Wert muss also gerundet werden. Im Falle eines maximalen Farbwertes von 255 kann diese Rundung eher vernachlässigt werden,

würde aber bei niedrigen maximalen Farbwerten theoretisch Probleme bereiten. Daher wurde die in der Graustufenkonvertierung erwähnte Rundung via 0.5 verwendet, um bei allen Implementationen die gleichen Werte zu erzielen, bevor die Resultate in einem $uint\_8$ abgespeichert werden.

3.4 Genauigkeit-Test

Die Genauigkeit wurde über ein breites Spektrum an Bildgrößen hinweg getestet, mit einem besonderen Fokus auf den Skalierungsfaktor 2. In jedem Fall, von kleinsten Bildern bis hin zu solchen mit 16 Millionen Pixeln, waren die Resultate identisch. Dies zeigt die Zuverlässigkeit der Implementierungen und die Effektivität der Rundungsmethoden, insbesondere der strategischen Addition von 0.5 vor der Abrundung in der SIMD-Implementierung.
Die Algorithmen demonstrieren eine hohe Präzision und Konsistenz, was ihre Anwendbarkeit in der Praxis bestätigt.

Skalierungsfaktor	Naive vs Optimized	Naive vs SIMD	Optimisiert vs SIMD
$1, 2, 3, 4$	0	0	0
$6, 7, 8, 9$	75763	51546	79451
$10, 11$	112437	85435	137018
$12, 13$	262964	189261	298229
$14, 15, 16, 17$	325182	229889	375447
$18, 19$	438917	352814	519551
20	535449	471138	687737

Tabelle 1: Vergleiche der Genauigkeit verschiedener Ansätze bei eine4 512x512 Eingabe

4 Performanzanalyse

Um die Performanz der vorgestellten und implementierten Lösungsansätze zu vergleichen, überprüfen und beurteilen zu können wurden sie auf einem Windows 11 Laptop mit einer i5-10300H CPU (L1: 256KB, L2: 1MB, L3: 8MB) mit 24GB 2600MHz RAM verglichen. Das Programm wurde mit GCC 11.4.0 -O0 kompiliert.

4.1 Komplexität

In der folgenden Sektion gilt $n =$ Anzahl der Pixel.

4.1.1 Rahmenprogramm

Das Rahmenprogramm besitzt eine Laufzeit von $O(n)$, da das Bild einmalig in ein Array eingelesen werden muss. Die Laufzeit der Überprüfung von Argumenten kann für die Komplexitätsberechnung vernachlässigt werden.

4.1.2 Grayscaling

Jede Grayscale-Methode ist auf $O(n)$ zurückzuführen. Auch wenn der Code beispielsweise beim Lookup-Table auf $O(1)$ für einen Pixel optimiert wurde oder durch SIMD mehrere Pixel gleichzeitig berechnet werden, wird diese Laufzeit trotzdem mit n in der Komplexität multipliziert. Durch Nachschlagen der Werte kann zwar eine Beschleunigung des Programmes erreicht werden, da dies aber nur leicht schneller als eine Multiplikation mit kleinen Zahlen ist, bleibt der Speed-Up marginal. Für ein großes n kann SIMD ein SpeedUp-Faktor von 4 erreichen.

4.1.3 Interpolation

Die Interpolation ist ebenfalls auf $O(n)$ zurückzuführen. (Genauer: $O(n \cdot k^2)$ mit $k =$ Interpolationsfaktor). Man erwartet aber ein deutlich besseres Laufzeitverhalten durch Code-Optimierungen

4.2 Laufzeitverhalten

Die Laufzeit des Programmes jeder Implementierung steigt linear mit der Eingabegröße. Zu erwarten war ein Laufzeitverhalten von $O(n)$
Wie zu erwarten sinkt die Laufzeit des Programmes durch Parallelisierung und Optimierung bezüglich der naiven Implementierung.

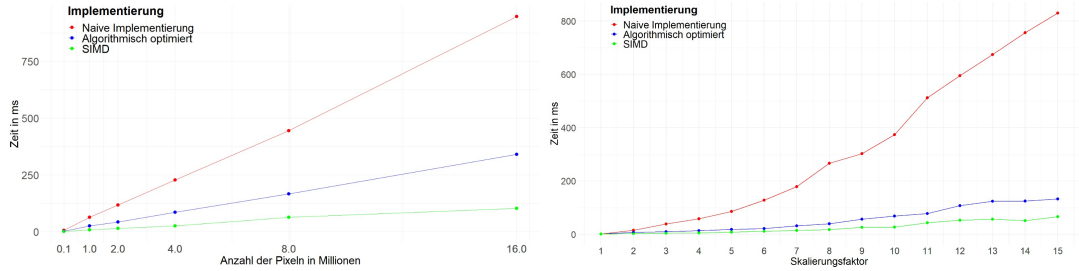

Abbildung 2: Links: Benötigte Zeit für ein Bild mit Skalierungsfaktor 2 anhand der Anzahl von Pixeln, Rechts: Benötigte Zeit für ein Bild von 512x512 anhand des Skalierungsfaktors

4.3 Cache

Aufgrund der Größe des resultierenden Bildes passt das Bild in keinen der L-Caches, wodurch die Speicherung vermutlich auf dem Hauptspeicher erfolgt und damit die Laufzeit des Programmes beeinträchtigt.

Pixel	Naiv (ms)	Opt. (ms)	SIMD (ms)	Naiv/SIMD (%)	Naiv/Opt. (%)
100k	5.5	2.12	0.73	753.42	259.62
500k	27.88	11.1	3.29	847.05	251.17
1 Mil	63.66	24.92	8.92	714.29	255.68
2 Mil	114.74	42.0	14.94	768.41	273.19
4 Mil	227.27	85.10	26.0	874.12	267.13
8 Mil	445.08	166.32	62.99	706.65	267.62
16 Mil	947.94	341.10	102.36	926.44	277.98

Tabelle 2: Laufzeitgergleich der Implementationsmethoden mit Skalierungsfaktor 2

5 Zusammenfassung und Ausblick

Dieses Projekt setzt sich mit der Bildmanipulation auseinander. Dabei liegt der Fokus auf die Umwandlung von Farbbildern in Graustufen und eine anschließende Skalierung durch bilineare Interpolation. Das Programm beinhaltet neben einem naiven Ansatz zum Nachvollziehen der mathematischen Berechnungen und der theoretischen Überlegungen auch verschiedene optimierte Ansätze, die je nach Kontext und Einbettung des Programms bessere Resultate liefern. Hierbei kommen algorithmische Optimierungen wie Look-up-Tables, die vorab Werte berechnen, sowie Parallelisierung durch SIMD zum Einsatz, um eine maximale Leistungsfähigkeit des Codes zu gewährleisten.

Dieses Projekt ist nicht nur theoretisch von besonderer Bedeutung, auch praktisch finden sich Graustufenkonvertierung und bilineare Interpolation viele sinnvolle Anwendungsmöglichkeiten. So wird die Graustufenkonvertierung oft in der Bild- und Videoverarbeitung verwendet, um Retro-Effekte nachzuahmen oder um Speicher zu sparen, wenn die Farbinformationen keine Rolle spielen. Die Integration der bilinearen Interpolation macht den Algorithmus auch bedeutsam für Videokompression und Computerspiele. So lässt sich ein Bild zuerst komprimieren und dann interpolieren, um Speicher zu sparen.
Dieses Projekt demonstriert, wie mathematischen Erkenntnisse und technische Grundlagen innovative Bildverarbeitungstechniken hervorbringen können, die die Bildqualität und die Bildrepräsentation revolutionieren können. Insgesamt kann das Projekt als Grundlage für viele weitere Arbeiten im Bereich der Bildmanipulation dienen.

Literatur

[1] *PPM Format Specification*, October 2016. `https://netpbm.sourceforge.net/doc/ppm.html`, visited 2023-12-18.

[2] *User Manual for NetPBM*, August 2020. `https://netpbm.sourceforge.net/doc/index.html#formats`, visited 2023-12-18.

[3] *Grafik/Grundbegriffe Graustufen*, January 2022. `https://wiki.selfhtml.org/wiki/Grafik/Grundbegriffe/Graustufen`, visited 2024-01-14.

[4] *Wahrnehmung von Helligkeiten / Grautönen*. `https://www.geowissensbildung.de/startseite-final-2/2-wahrnehmung-von-farben-final/2-6-wahrnehmung-von-helligkeitengrautoenen-final/`, visited 2024-01-14.

[5] *Bilineare Interpolation*, December 2022. `https://de.wikibrief.org/wiki/Bilinear_interpolation`, visited 2024-01-14.

[6] Paul Bourke. *PPM, PGM, PBM Image Files*, July 1997. `https://paulbourke.net/dataformats/ppm/`, visited 2024-01-20.

[7] University of Tennessee. *ECE472/572 Digital Image Processing*. `https://web.eecs.utk.edu/~hqi/ece472-572/testimage.htm`, visited 2023-12-25.

[7] [1] [2] [6] [3] [5] [4]