LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Gammakorrektur (A208)
Projektaufgabe – Aufgabenbereich Bildverarbeitung

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage[1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen Assembler-Code anzufertigen ist, sind für die x86-64 Architektur unter Verwendung der SSE-Erweiterungen (bis SSE4.2) zu schreiben; alle anderen Bestandteile der Hauptimplementierung sind in C nach dem C17-Standard anzufertigen.

Der **Abgabetermin** ist **Sonntag 2. Februar 2025, 23:59 Uhr**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository auf dem `main` Branch. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **10.03.2024 –21.03.2025** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind und keine nachträglichen Änderungen akzeptiert werden können.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihre:n Tutor:in.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `https://asp.caps.in.tum.de`

2 Gammakorrektur

Im Zuge Ihrer Projektaufgabe werden Sie theoretisches Wissen aus der Mathematik im Anwendungszusammenhang verwenden, um einen Algorithmus in Assembler zu implementieren. Sie konzentrieren sich dabei auf das Feld des *Image Processing*, in welchem Pixelbilder, wie sie typischerweise Digitalkameras produzieren, als Eingabe für bestimmte Algorithmen verwendet werden und mathematische Überlegungen dadurch sichtbar gemacht werden.

Die zu implementierenden Algorithmen werden verwendet, um ein farbiges Bild in Graustufen zu konvertieren und anschließend eine Gammakorrektur durchzuführen.

2.1 Funktionsweise

Wir definieren ein Farbpixel an der Position (x, y) des Bildes P als einen Vektor mit drei Elementen für die Farbkanäle Rot (R), Grün (G) und Blau (B):

$$P_{(x,y)} = \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad (x, y) \in \mathbb{D} = \{0, \ldots, \text{Breite} - 1\} \times \{0, \ldots, \text{Höhe} - 1\}$$

Wir betrachten zunächst die Graustufenkonvertierung: Die Umwandlung eines Pixels in Graustufen geschieht durch das Berechnen eines gewichteten Durchschnitts D mittels der Koeffizienten a, b und c:

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

Das Pixel in Graustufen und somit das entsprechende Graustufenbild Q ist dann gegeben durch

$$Q_{(x,y)} = D$$

Das Graustufenbild soll anschließend noch einer Gammakorrektur unterzogen werden. Hierzu weist man jedem Graustufenpixel im Bild einen neuen Wert zu, der sich berechnet aus:

$$Q'_{(x,y)} = \left(\frac{Q_{(x,y)}}{255} \right)^{\gamma} \cdot 255 \tag{2}$$

Der Parameter γ ist dabei eine vom Benutzer zu spezifizierende Fließkommazahl.

2.2 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Antworten auf konzeptionelle Fragen

sollten an den passenden Stellen in Ihrem Vortrag in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihren Code reflektiert.

Wichtig: Mindestens eine Ihrer Implementierungen muss mit mathematischen Operationen auskommen, die ausschließlich grundlegende Berechnungen durchführen (im Zweifel: die vier Grundrechenarten, Shifts und logische Operationen), nicht jedoch Instruktionen, die komplexere Berechnungen durchführen (z.B. Wurzel, Logarithmus, Exponentiation, etc.).

2.2.1 Theoretischer Teil

- Machen Sie sich mit dem PPM[2] Bildformat vertraut. Sie werden 24bpp PPM (P6) Bilder als Eingabe für Ihr Programm verwenden.
- Wählen Sie ein geeignetes Netpbm[3] Format (auch mit Hinblick auf Effizienz) für die Ausgabe und machen Sie sich ebenfalls mit diesem vertraut.
- Bestimmen Sie Werte für a, b und c in Gleichung 1 zunächst durch intuitives Festlegen und passen Sie sie später anhand der entstehenden Graustufenbilder weiter an, sobald Sie die Aufgabe gelöst haben. Berücksichtigen Sie hierbei die Eigenschaften des Menschlichen Visuellen Systems (HVS) und ziehen Sie, wenn nötig, geeignete Sekundärliteratur hinzu.
- Finden Sie eine Möglichkeit die Exponentialfunktion, die in Formel 2 benötigt wird, nur mittels grundlegender mathematischer Operationen zu berechnen.
- Entwickeln Sie einen Algorithmus, welcher ein Bild erst in Graustufen konvertiert und anschließend eine Gammakorrektur auf allen Pixeln ausführt.

2.2.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie eine PPM Datei einlesen können und Ihrer Assemblerroutine einen Pointer auf die sequentiellen Bilddaten sowie die relevanten Metadaten (z.B. Höhe und Breite) übergeben können. Das Rahmenprogramm soll dann eine neue Datei in dem von Ihnen gewählten Netpbm Format erstellen und die berechneten Werte nach dem Aufruf dort hineinschreiben. Das Ergebnis sollte mit üblichen Bildbetrachtern lesbar sein.
- Implementieren Sie in der Datei mit Ihrem Assemblercode die Funktion:

[2] PPM Spezifikation: `https://netpbm.sourceforge.net/doc/ppm.html`
[3] Netpbm Formate: `https://netpbm.sourceforge.net/doc/index.html#formats`

```
void gamma_correct(
   const uint8_t* img, size_t width, size_t height,
   float a, float b, float c,
   float gamma,
   uint8_t* result);
```

Die Funktion nimmt dabei einen Pointer auf die RGB Pixelwerte des Bildes sowie weitere benötigte (Meta)Daten als Parameter entgegen, und speichert das Ergebnis der Graustufenkonvertierung, sowie der Gammakorrektur in `result`.. Allokieren Sie hierzu in Ihrem Rahmenprogramm Buffer passender Größe.

2.2.3 Rahmenprogramm

Ihr Rahmenprogramm muss bei einem Aufruf die folgenden Optionen entgegennehmen und verarbeiten können. Wenn möglich soll das Programm sinnvolle Standardwerte definieren, sodass nicht immer alle Optionen gesetzt werden müssen. Wird eine Option mit Argument gesetzt, so ist das entsprechende Argument zu benutzen. Die Reihenfolge der Optionen bei einem gültigen Aufruf muss irrelevant sein. Wir empfehlen Ihnen, die Kommandozeilenparameter mittels `getopt_long`[4] zu parsen.

- `-V` *<Zahl>* — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V 0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B`*<Zahl>* — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das *optionale* Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an, z. B. `-B` oder `-B5` .
- *<Dateiname>* — Positional Argument: Eingabedatei
- `-o` *<Dateiname>* — Ausgabedatei
- `--coeffs` *<FP Zahl>*,*<FP Zahl>*,*<FP Zahl>* — Die Koeffizienten der Graustufenkonvertierung a, b und c. Falls diese Option nicht gesetzt wird, sollen die von Ihnen bestimmten Werte verwendet werden.
- `--gamma` *<Floating Point Zahl>* — $\gamma \in [0, \infty)$
- `-h|--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet. Die Beschreibung soll mit "Help Message" anfangen.

Sie dürfen weitere Optionen implementieren, beispielsweise um vordefinierte Testfälle zu verwenden. Ihr Programm muss jedoch nur unter Verwendung der oben genannten Optionen verwendbar sein. Beachten Sie ebenfalls, dass Ihr Rahmenprogramm etwaige Randfälle korrekt abfangen muss und im Falle eines Fehlers mit einer Fehlermeldung,

[4] Siehe `man 3 getopt_long` für Details

die auf den konkreten Fehler hinweist und einer kurzen Erläuterung zur Benutzung **terminieren** sollte. Fehlermeldungen jeglicher Art sollen auf `stderr` ausgegeben werden. Die Wertebereiche der Argumente dürfen nicht unnötig eingeschränkt werden.

2.3 Allgemeine Bewertungshinweise

Beachten Sie grundsätzlich alle in der Praktikumsordnung angegebenen Hinweise. Die folgende Liste konkretisiert einige der Bewertungspunkte:

- Diese PDF-Datei darf nicht verändert, gelöscht oder umbenannt werden.
- Die in `README.md` geforderte Struktur des Projektrepositorys muss eingehalten werden.
- Der Exit-Code Ihrer Implementierung muss die erfolgreiche Ausführung oder das Auftreten eines Fehlers widerspiegeln. Benutzen Sie dafür die Makros aus `stdlib.h`.
- Stellen Sie unbedingt sicher, dass Ihre Implementierung auf der Referenzplattform des Praktikums (`lxhalle`) kompiliert und vollständig funktionsfähig ist.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. Achten Sie darauf, dass Ihr Programm keine x87-FPU- oder MMX-Instruktionen und SSE-Erweiterungen nur bis SSE4.2 verwendet. Andere ISA-Erweiterungen (z.B. AVX, BMI1) dürfen Sie nur benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne derartige Erweiterungen lauffähig ist.
- Sie dürfen die angegebenen Funktionssignaturen, bis auf die Änderung des Funktionsnamens für die Benennung der unterschiedlichen Implementierungen (siehe nächster Punkt) *nicht* ändern
- Verwenden Sie den angegebenen Funktionsnamen bzw. die angegebenen Funktionsnamen für Ihre Hauptimplementierung bzw. Hauptimplementierungen. Für alle weiteren Implementierungen gilt folgendes Schema: Hinter den Funktionsnamen wird das Suffix „`_V1`“, „`_V2`“, usw. angehängt.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Korrektheit/Genauigkeit, Performanz) und behandeln Sie *alle möglichen Eingaben*, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden; größere Eingaben sollten stattdessen stark komprimiert oder (bevorzugt) über ein abgegebenes Skript generierbar sein.
- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.

- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 16:9 als Folien-Format und fügen Sie Foliennummern hinzu.