4. Aufgabe: Medizinische Bilder

1 Einleitung

In der modernen Medizin gibt es zahlreiche bildgebende Verfahren, die eine zuverlässige Diagnose unterstützen. Eine große Rolle spielen dabei das *klassische Röntgen*, *Computertomografie* (CT), *Ultraschall*, *Magnetresonanztomografie* (MRT, auch *Kernspintomografie* genannt) sowie Verfahren aus dem Bereich der Nuklearmedizin.

Um eine Kompatibilität zwischen den verschiedenen bildgebenden Verfahren und Herstellern zu gewährleisten, wurde ein Standard definiert: *Digital Imaging and Communications in Medicine* (DICOM). Zusätzlich zu diesem Standard werden lokal auf den Systemen die erstellten Bilder in einem proprietären Format abgelegt. Dabei ist es üblich, die Bilder unkomprimiert und vor allem Verlustfrei zu speichern.

Eine einfache Methode solche Bilder einzulesen, ist, nur die Pixeldaten direkt aus der Datei zu extrahieren: Es werden die Pixelwerte in *RAW-Format* importiert. Um eine Bild-Datei auf diese Weise zu lesen, müssen ein paar Informationen über das Bildformat vorliegen:

Name	Bedeutung
header	Größe des Headers vor den Pixeldaten
width	Anzahl der Pixel pro Zeile
height	Anzahl der Zeilen
bytesPerPixel	Anzahl der Bytes pro Pixel
isSigned	Haben die Pixelwerte ein Vorzeichen?
littleEndian	Wird erst das niederwertigste Byte gespeichert?

Tabelle 1: Informationen zu einem Bild im RAW-Format

Wird eine Datei im RAW-Format eingelesen, so wird der *Header* zu Beginn der Datei ignoriert. Wichtig ist nur die Info, wie groß der Header ist, damit direkt mit dem Lesen der Pixelwerte begonnen werden kann.

Da der Header nicht ausgewertet wird, muss die Größe des Bildes (*width* und *height*) bekannt sein. Ggf. muss versucht werden, die Größe des Bildes mit einem Hex-Editor aus dem Header der Datei auszulesen. Die Art und Weise, wie die Größe angegeben wird, hängt vom Bildformat ab. Für dieses Praktikum wurden ein paar Bilder mit den benötigten Informationen zum Einlesen zur Verfügung gestellt.

In der Medizintechnik werden die Bilder häufig mit mehr als 256 Graustufen gespeichert, so dass pro Pixelwert mehr als ein Byte nötig ist. Typisch sind dabei 10 oder 12 Bit, d.h. 1024 oder 4096 Pixelwerte. Die übrigen Bits werden ignoriert und üblicher Weise auf null gesetzt.

Die Pixelwerte können mit und ohne Vorzeichen gespeichert werden. Das hier zu erstellende Programm soll die Pixel als *short*-Variablen speichern, d.h. 16 Bit mit Vorzeichen. Sind die Pixel in der Datei ohne Vorzeichen gespeichert, so können diese auch in *short* gespeichert werden, wenn die Anzahl der Bits 15 nicht übersteigt.

Little Endian (auf deutsch „*Klein-Ender*“) bedeutet, dass das kleinstwertige Byte zuerst und das höchstwertige Byte zuletzt gespeichert wird (ähnlich wie beim Datum: Tag.Monat.Jahr). Das entspricht der Reihenfolge auf unseren Rechnern, so dass die Bytes direkt in eine Variable gespeichert werden können.

Big Endian (auf deutsch „*Groß-Ender*“) ist das Gegenteil von *little endian* und bedeutet, dass das höchstwertige Byte zuerst gespeichert wird. Hier müssen die Bytes einzeln eingelesen

und entsprechend anders herum in eine Variable gespeichert werden.

In diesem Praktikum wollen wir medizinische Röntgenbilder im RAW-Format importieren, verarbeiten und als Bitmap speichern. Dabei soll mit einer *Look-Up-Table* (LUT) eine Anpassung der Grauwerte durchgeführt werden.

2 Bild einlesen und als Bitmap speichern

Erstellen Sie zunächst eine Header- und Quellcode-Datei mit den Namen *rawImage.h* und *rawImage.c*. In diesen Dateien werden die nachfolgenden Strukturen und Funktionen implementiert. Binden Sie des Weiteren die Dateien *bitmap.h* und *bitmap.c* der letzten Praktikumsaufgabe in Ihr Projekt ein.

2.1 Strukturen erstellen

2.1.1 Struktur für RAW-Einstellungen

Erstellen Sie in der Header-Datei *rawImage.h* eine Struktur gemäß Tabelle 1. Wählen Sie für alle Elemente den Typ *unsigned* und definieren Sie damit den neuen Datentyp *tRawInfo*.

2.1.2 Struktur für eingelesenes Bild

Erstellen Sie in der Header-Datei *rawImage.h* eine Struktur für ein eingelesenes Bild gemäß Tabelle 2 und definieren Sie dafür den neuen Datentyp *tImage*.

Typ	Name	Beschreibung
unsigned	width	Breite des Bilds
unsigned	height	Höhe des Bilds
short**	pixel	Pixelwerte des Bilds

Tabelle 2: Struktur für ein eingelesenes Bild.

2.2 Bild im RAW-Format einlesen

Erstellen Sie eine Funktion zum Einlesen eines medizinischen Bilds im RAW-Format:

```
tImage *importRawImage(char *filename, tRawInfo rawInfo);
```

Die Funktion reserviert dynamisch Speicher für die Struktur *tImage* und einen zweidimensionalen Vektor innerhalb der Sturktur. Danach liest die Funktion das Bild gemäß der übergebenen Parameter aus der Datei ein. Bei einem Fehler wird *NULL* zurückgegeben.

Hinweis: Mit der Funktion *fseek()* aus der Header-Datei *stdio.h* kann in einer Datei vor und zurück gesprungen werden.

2.3 Bild als Bitmap speichern

Schreiben Sie eine Funktion, welche das Bild als Bitmap in eine Datei schreibt:

```
int exportImageBmp(char *filename, tImage *image);
```

Die Funktion erstellt eine Bitmap gleicher Breite und Höhe wie das übergebene Bild. Danach ermittelt die Funktion im übergebenen Bild den kleinsten und größten Pixelwert. Mithilfe dieser Extremwerte werden die Pixelwerte des übergebenen Bilds auf die Werte 0 bis 255 umgerechnet und in das Bitmap kopiert. Anschließend wird das Bitmap in eine Datei mit dem übergebenen Namen gespeichert. Bei Erfolg soll der Wert null, im Falle eines Fehlers ein Wert ungleich null zurückgegeben werden.

2.4 Freigabe des Speichers

Erstellen Sie eine Funktion zur Freigabe des Speichers eines Bilds:

```
void freeImage(tImage *image);
```

Die Funktion muss dreimal Speicher Freigeben: a) Speicher der Pixeldaten, b) Speicher des Zeigervektors für zweidimensionalen Vektor und c) Speicher der Struktur *tImage*.

2.5 Test der Funktionen

Gehen Sie im Hauptprogramm wie folgt vor:

1. Erstellen Sie im Hauptprogramm eine Variable für die Struktur *tRawInfo* und initialisieren Sie diese mit den Parametern für eines der zur Verfügung gestellten Bilder.
2. Erstellen Sie einen Zeiger auf die Struktur *tImage* und initialisieren Sie ihn mit *NULL*.
3. Rufen Sie die Funktionen *importRawImage()* zum Importieren eines Bilds auf.
4. Rufen Sie die Funktion *exportImageBmp()* zum Speichern des Bilds als Bitmap auf.
5. Geben Sie den Speicher für das Bild frei, indem Sie die Funktion *freeImage()* aufrufen.

Hinweis: Aller Voraussicht nach wird mit normalen Programmen zum Betrachten der Bitmaps das Bild vertikal gespiegelt („auf dem Kopf“) erscheinen. Um das zu beheben sollten Sie in der Funktion *importRawImage()* die Pixel entsprechend vertikal gespiegelt speichern.

3 Anwendung einer Look-Up-Table (LUT)

Bei der Auswertung von medizinischen Bildern will der Radiologe häufig Kontraste in bestimmten Bereichen erhöhen oder absenken. Zur Realisierung wird dabei eine *Look-Up-Table*, kurz *LUT* (auf deutsch *Tabellen zum Nachschauen*) implementiert: Es wird eine Tabelle angelegt, die für jeden Pixelwert einen Grauwert angibt. Beim Speichern des Bilds in eine Bitmap wird für jeden Pixelwert über die LUT der dazugehörige Grauwert ermittelt.

Eine LUT wird als ein Vektor angelegt, dessen Elemente vom Typ *unsigned char* sind. Die Anzahl der Elemente ergibt sich aus der Anzahl der vorhandenen Graustufen im Bild.

3.1 Struktur für eingelesenes Bild erweitern

Bereiten Sie die Struktur *tImage* für eine LUT vor:

1. Erweitern Sie die Struktur *tImage* um zwei weitere Elemente gemäß Tabelle 3.
2. Initialisieren Sie in der Funktion *importRawImage()* den Zeiger für die LUT mit NULL.

Typ	Name	Beschreibung
unsigned char*	LUT	Zeiger auf dynamischen Vektor für LUT
short	minVal	Minimaler Pixelwert im Bild

Tabelle 3: Erweiterung der Struktur aus Tab. 2 zur Implementierung einer LUT.

3. Wenn der Zeiger für die LUT nicht den Wert NULL hat, geben Sie den Speicher in der Funktion *freeImage()* frei.

3.2 Erstellen einer linearen LUT

Erstellen Sie eine Funktion, die eine lineare LUT erzeugt:

```
int createLutLinear(tImage *image);
```

Gehen Sie wie folgt vor:

1. Ermitteln Sie im übergebenen Bild den Minimal- und Maximalwert.
2. Reservieren Sie für die LUT Speicher, so dass für jeden Pixelwert vom Minimum bis zum Maximum ein Grauwert gespeichert werden kann. Hat der Zeiger *LUT* nicht den Wert NULL, so muss zunächst der vorher reservierte Speicher freigegeben werden.
3. Initialisieren Sie den Vektor linear mit Grauwerten von null bis 255.
4. Speichern Sie den minimalen Pixelwert in das Strukturelement *minVal* der Struktur *tImage*.
5. Bei Erfolg soll null, bei einem Fehler ungleich null zurückgegeben werden.
6. Rufen Sie in der Funktion *importRawImage()* nach dem Einlesen des Bilds die eben erstellte Funktion *createLutLinear()* auf.

3.3 Bild mit LUT als Bitmap speichern

Modifizieren Sie die Funktion aus Abschnitt 2.3. Gehen Sie wie folgt vor:

1. Beim Kopieren eines Pixels ziehen Sie vom Pixelwert den Minimalwert ab und verwenden diese Differenz als Index für den LUT-Vektor.
2. Den Wert aus der LUT speichern Sie in die Bitmap.

Ein Test sollte zum selben Ergebnis führen wie im Abschnitt 2.

3.4 Erstellen einer LUT mit angepasstem Fenster

Jetzt soll eine LUT erstellt werden, mit der Helligkeit und Kontrast im Bild angepasst werden können. Erstellen Sie dafür eine Funktion wie in Abschnitt 3.2 mit den zwei zusätzlichen Parametern für die Pixelwerte, die weiß (p_{ws}) bzw. schwarz (p_{sz}) dargestellt werden sollen:

```
int createLutMinMax(short pvBlack, short pvWhite, tImage *image);
```

Die Pixelwerte dazwischen werden entsprechend linear auf die Graustufen abgebildet. Die Pixelwerte außerhalb dieser Grenzen werden entsprechend schwarz bzw. weiß dargestellt. Zwei Beispiele sind in Abbildung 1 skizziert.

Testen Sie die Funktion für verschiedene Pixelwerte. Testen Sie auch Varianten mit $p_{\mathrm{sz}} > p_{\mathrm{ws}}$ und $p_{\mathrm{sz}}, p_{\mathrm{ws}} \notin [p_{\min}, p_{\max}]$.

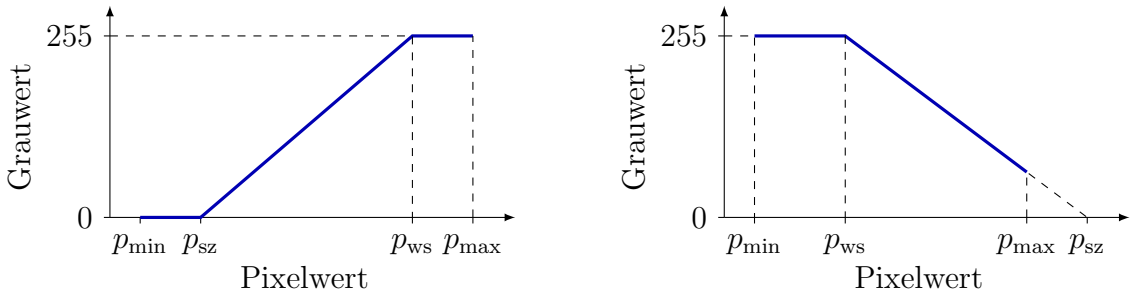

Abbildung 1: Beispiele einer LUT mit unterschiedlichen Pixelwerten für schwarz und weiß.

4 Optionale Aufgaben

4.1 Menü zum Steuern des Programms

Erstellen Sie ein Menü, mit dem der Benutzer das Programm steuern kann. Es soll möglich sein, die Pixelwerte für schwarz und weiß anzugeben. Weitere Optionen sind die Wahl der Eingabedatei mit der Angabe der Parameter aus Tabelle 1 sowie die Wahl einer Ausgabedatei.

4.2 Histogramm

Ein Histogramm gibt die Anzahl der Pixel als Funktion der Pixelwerte an. Die Form des Histogramms gibt Aufschluss über die Pixelwertverteilung und kann zur Optimierung der Darstellung verwendet werden. Erstellen Sie eine Funktion, die aus einem Bild das Histogramm ermittelt und in eine CSV-Datei schreibt.

4.3 Histogrammäqualisation

Bei der *Histogrammäquilisation* sollen alle Grauwerte mit der gleichen Häufigkeit erscheinen. Für das Histogramm hat das zur Folge, dass der Verlauf der Funktion sich einer Konstante nähert, siehe Abbildung 2.

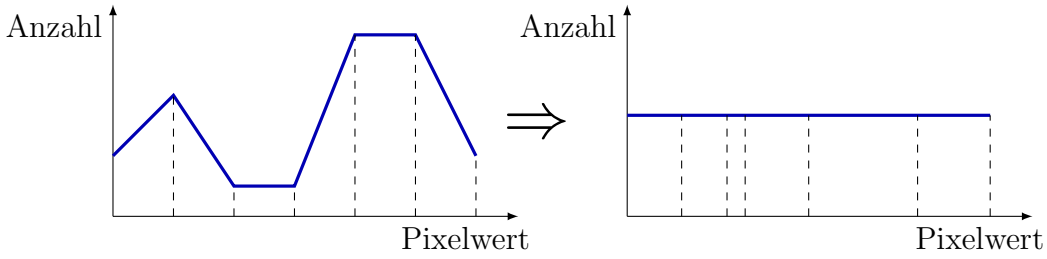

Abbildung 2: Histogrammäquilisation

Erstellen Sie eine Funktion, die eine LUT erstellt, deren Anwendung zur einer Histogrammäquilisation führt. Gehen Sie wie folgt vor:

1. Erstellen Sie für das eingelesene Bild ein Histogramm.
2. Bilden Sie das Integral. D.h. jedes Element ist die Summe aller vorigen Elemente.
3. Skalieren Sie das Integral auf den maximalen Grauwert (d.h. 255).

4. Speichern Sie das skalierte Integral als LUT.

Wenden Sie diese LUT auf die zur Verfügung gestellten Bilder an und betrachten Sie das Ergebnis.

Viel Spaß beim Programmieren!