FDS2 - Übung 2: Bitmap Implementation

Tim Peko

SS 2025

1. Lösungsansatz

1.1. Datenstruktur

Die Implementierung basiert auf einer einfachen Struktur, die die Dimensionen des Bildes und einen Zeiger auf die Pixeldaten speichert:

```
struct bitmap {
    long_type width;
    long_type height;
    pixel_type* pixels;
};
```

Die Pixeldaten werden als eindimensionales Array gespeichert, wobei die Pixel zeilenweise angeordnet sind. Der Zugriff auf ein Pixel an der Position (x, y) erfolgt über den Index `y * width + x`.

1.2. Speicherorganisation

Das BMP-Format speichert Bilder zeilenweise von unten nach oben, wobei jede Zeile auf ein Vielfaches von 4 Bytes aufgefüllt wird (Padding). Die Pixeldaten werden im BGR-Format gespeichert (Blau, Grün, Rot).

Dabei war wichtig zu beachten, dass die verwendeten C++ `structs` die Byte-Ausrichtung 1 haben müssen, da die BMP-Dateien in dieser Ausrichtung gespeichert werden. `#pragma pack(push, 1)` wurde verwendet, um die Byte-Ausrichtung zu gewährleisten.

1.3. Implementierte Funktionen

1.3.1. Erzeugung von Bitmaps

Die Funktionen `generate_bitmap` macht sich den Konstruktor von `bitmap` zunutze, um ein neues Bitmap-Objekt zu erzeugen und mittels `new` den Speicher auf dem Heap zu reservieren.

1.3.2. Größenänderung

Bei der Größenänderung werden die Pixeldaten neu allokiert, die alten Pixeldaten werden mittels `free` freigeben und die neuen Pixeldaten werden mittels `new` angelegt.

1.3.3. Bildmanipulation

Verschiedene Funktionen zur Bildmanipulation wurden implementiert:

- `detect_edges`: Kantenerkennung mittels Sobel-Operator
- `fill`: Füllen des Bildes mit einer Farbe
- `invert`: Invertieren der Farben
- `to_gray`: Umwandlung in Graustufen auf Basis des Luminanz-Werts

1.3.4. Kantenerkennung

Der Algorithmus berechnet den Gradienten des Bildes in x- und y-Richtung mithilfe von Sobel-Operatoren und kombiniert diese zu einem Gesamtgradienten, der die Stärke der Kante angibt.

Vor der Anwendung des Sobel-Operators wird das Bild in Graustufen umgewandelt.

1.3.5. Graustufen

Die Umwandlung in Graustufen erfolgt mithilfe des Luminanz-Werts.

$$L = \sqrt{0.299 \cdot R^2 + 0.587 \cdot G^2 + 0.114 \cdot B^2}$$

Dieser wird für jeden Pixel berechnet und alle Farbkanäle werden durch diesen Grauwert ersetzt.

$$R' = G' = B' = L$$

1.3.6. Invertieren

Die Invertierung der Farben erfolgt, indem bei jedem Farbkanal der maximal mögliche Wert W mit dem aktuellen Wert C_i subtrahiert wird:

$$C_{i'} = W - C_i$$

2. Testfälle

Die Implementierung wurde mit verschiedenen Testfällen überprüft. Die Testfälle sind in der Datei `test_bitmap.cpp` zu finden.

2.1. Erzeugung von Bitmaps

Es werden verschiedene Bitmaps mit unterschiedlichen Größen und Farben erzeugt und deren Größe überprüft.

Ergebnis: Erfolgreich

2.2. Kopieren von Bitmaps

Es wird ein Bitmap erzeugt und ein Kopie erstellt. Die ursprüngliche Bitmap wird dann gelöscht und die Kopie wird überprüft.

Ergebnis: Erfolgreich

2.3. Lesen und Schreiben von BMP-Dateien

Es wird ein Bitmap erzeugt und in eine BMP-Datei geschrieben. Die Datei wird dann wieder gelesen und die Pixel werden überprüft.

Ergebnis: Erfolgreich

2.4. Anwendung der Bildmanipulationsfunktionen

Es wird ein Bitmap erzeugt und die Bildmanipulationsfunktionen werden aufgerufen. Die Pixel werden überprüft, ob sie die erwarteten Werte haben.

Ergebnis: Erfolgreich

2.5. Beispiel Anwendung (`example::main`)

Es wird eine gegebene BMP-Datei geladen und die Bildmanipulationsfunktionen werden aufgerufen. Die Ergebnisse werden in eine neue BMP-Datei geschrieben.

Dieser Testfall wird durch manuelles Überprüfen der Ergebnisse bestätigt.

Ergebnis: Erfolgreich

3. Fazit

Die Tests zeigen, dass die Implementierung korrekt funktioniert und die geforderten Funktionen erfüllt.

Aufwand in Stunden: 8