Projektaufgabe: Pixelwiederholung

Vortrag

Till-Ole Lohse (03696244)
Yannick Sihler (03698787)
Laert Llaveshi (03729916)

Donnerstag, 24.08.2023

Überblick

- Einleitung
- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

Überblick

- **Einleitung**
- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

Projektaufgabe: Pixelwiederholung

- Behandelt Bereich der **digitalen Bildverarbeitung**
- Pixelmanipulation wird auf Bild durchgeführt
- Bilder mit BMP24-Format (Erklärung folgt)

Überblick

- **Einleitung**
- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

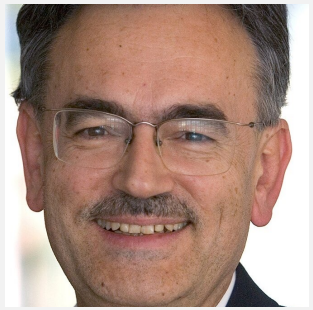
Überblick

- Einleitung
- **Aufgabe und generelle Funktionsweise**
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

Generelle Funktionsweise (1/2)

Beispiel

Ausgangsbild



Schritt 1: Bild ausschneiden

Schritt 2: Bild skalieren

BMP24-Format

- Bild-Daten liegen unkomprimiert im Speicher hintereinander
- Jeder einzelne Pixel besteht aus 24 Bit
- Setzt sich aus jeweils 8 Bit der Grundfarben Blau, Grün und Rot zusammen
- Über 16 Millionen verschiedene Farben pro Pixel möglich
- Besonders gut geeignet für hochwertige Digitalbilder
- Jedoch hoher Speicher nötig

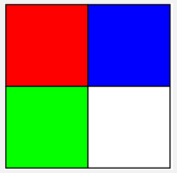
Generelle Funktionsweise (2/2)

Anforderungen

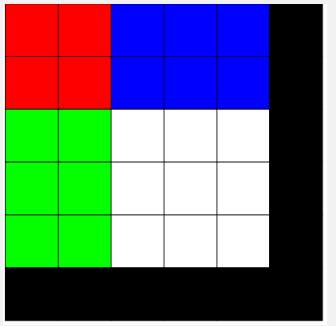
- Inputbild in BMP-Format
- **1. Teil: window:**
  - berechnet aus den Eingabedaten des Bildes ein Fenster mit den Pixelkoordinaten im Bereich von (x, y) bis (x + width, y + height)
- **2. Teil: zoom**
  - Skaliert den Fensterbereich um den Skalierungsfaktor und füllt die
- **Output:** verändertes Bild in BMP-Format

Illustration 2. Teil: zoom

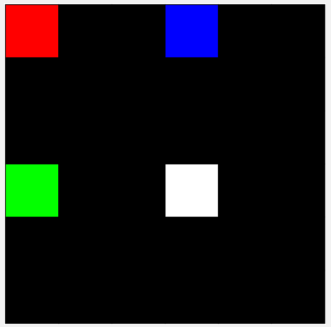
Ausgangsbild:



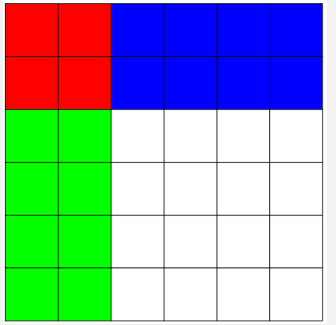
Lücken gefüllt mit direkten Nachbarn



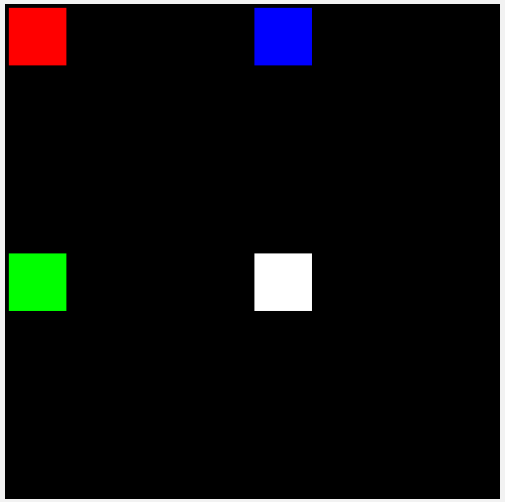
Skaliert mit Faktor:



Lücken gefüllt mit Tie-Breaker (*bottom-left*)

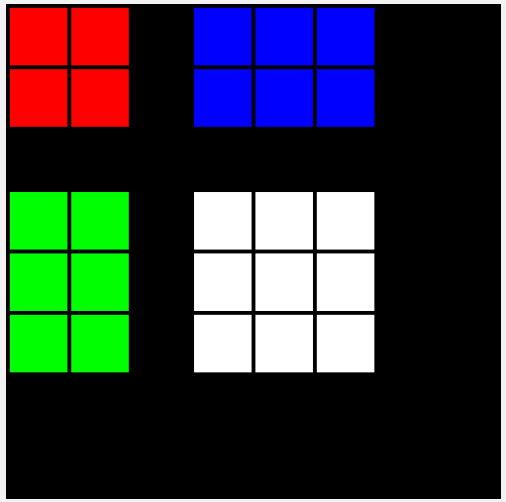


zoom() Funktion



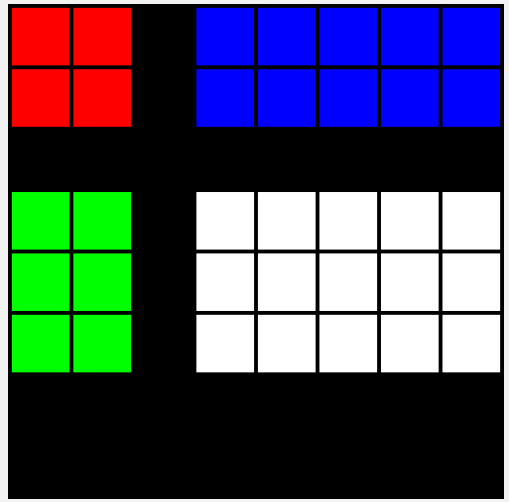
1. Schritt

- Pixel werden im Speicher abgelegt
- Skalierung zu beachten



2. Schritt

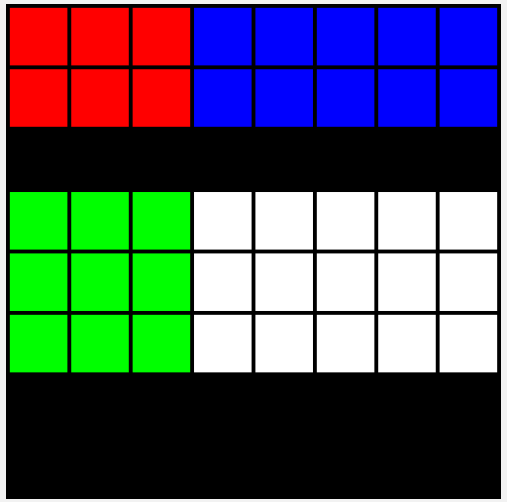
- Eindeutige Nachbarn werden gefärbt



3. Schritt

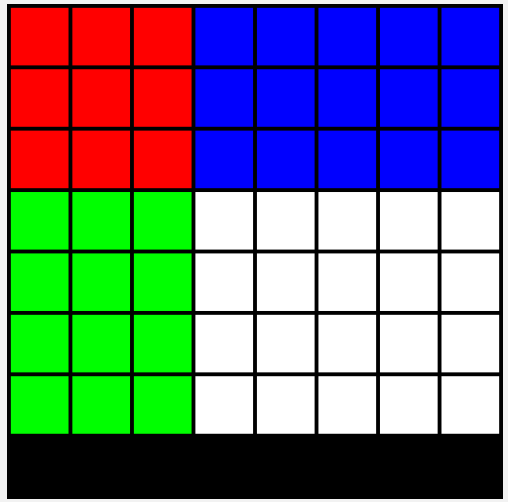
- Rechter Pixelrand wird gefärbt
- Selbe Farbe wie linksliegenden Pixel

zoom() Funktion



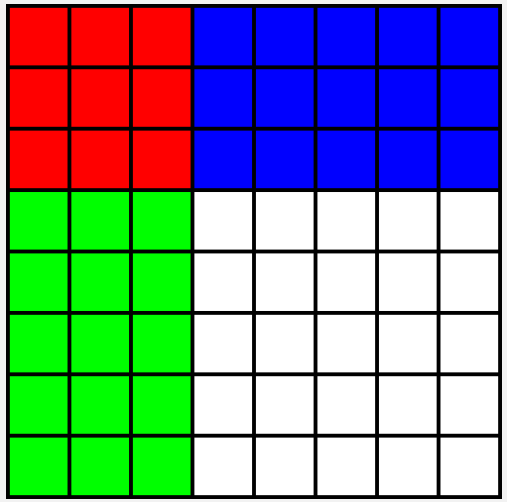
4. Schritt

- Vertikale Lücken werden geschlossen
- Nur bei geradem Skalierungsfaktor



5. Schritt

- Horizontale Lücken werden geschlossen
- Nur bei geradem Skalierungsfaktor



6. Schritt

- Unterer Pixelrand wird gefärbt
- Selbe Farbe wie darüberliegende Pixel

Überblick

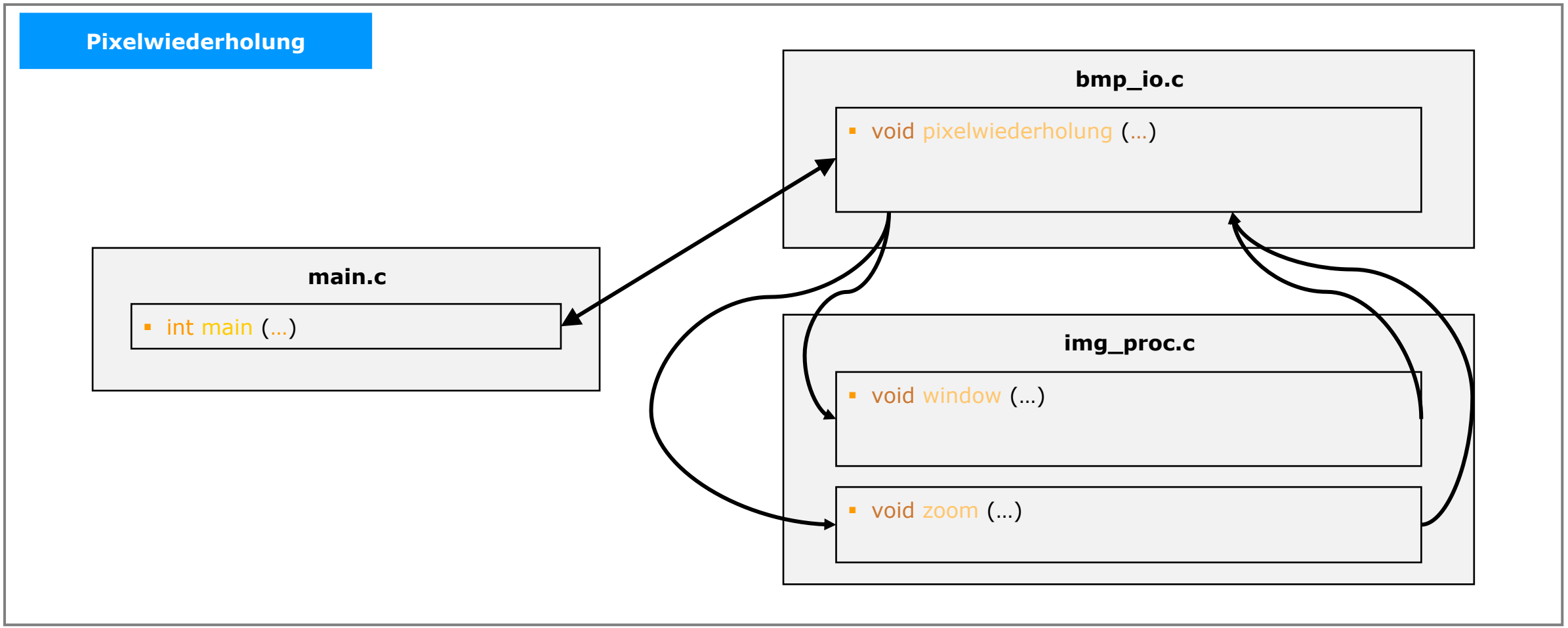
- **Aufgabe und generelle Funktionsweise**
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

Überblick

- Aufgabe und generelle Funktionsweise
- **Implementierung:**
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- Performance

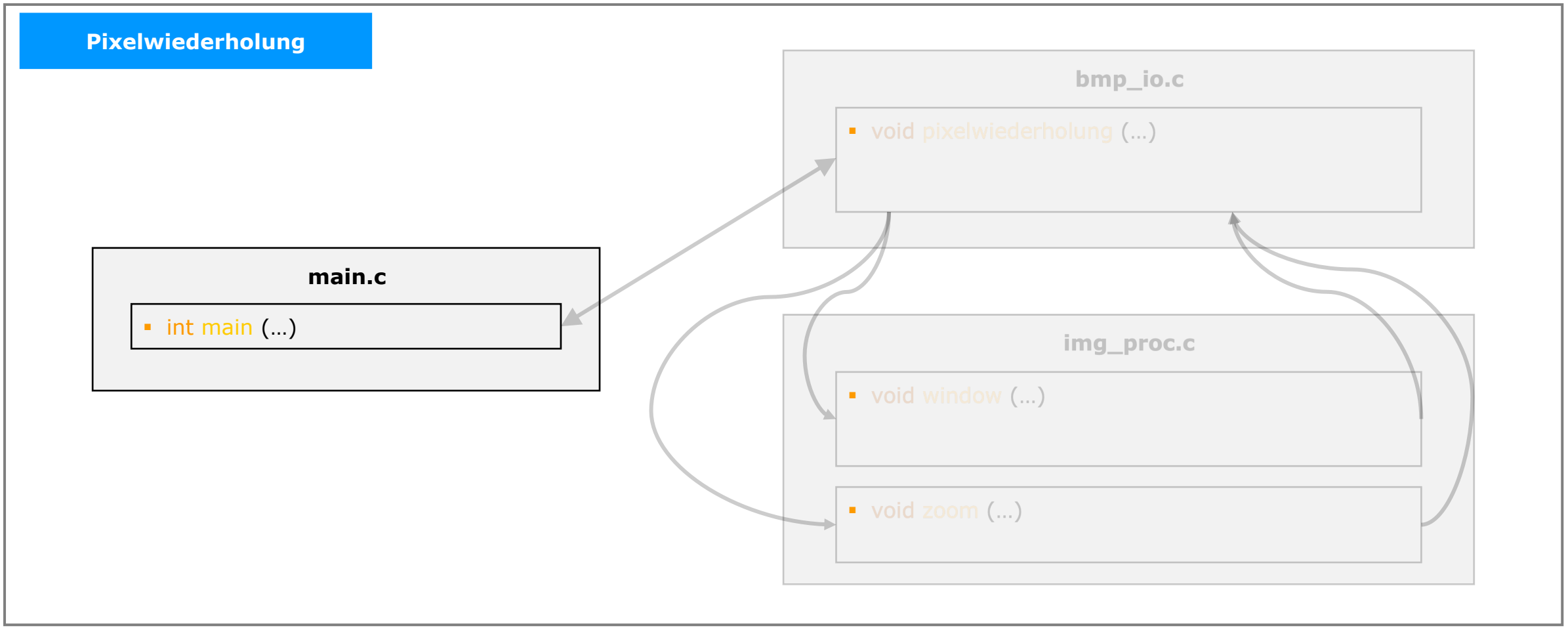
Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: main / Rahmenprogramm

Projektaufgabe: Pixelwiederholung

Verwendung von Pseudocode!

Anforderung → **Umsetzung**

Optionen:

- **-V<Zahl>:** gibt die zu nutzende Code Version an
- **-B<Zahl>:** Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs
- **<Dateiname>:** Eingabedatei
- **-s<Zahl>,<Zahl>:** Startpunkt des Fensterausschnitts (x,y)
- **-w<Zahl>:** Breite des Bildfensters
- **-h<Zahl>:** Höhe des Bildfensters
- **-f<Zahl>:** Skalierungsfaktor
- **-o<Dateiname>:** Ausgabedatei
- **--help/ -h:** Hilfe/ mehr Informationen

int main(int argc, char *argv[]):

--help / -h abfangen

while (…):

vergleiche Argument und setze Variable dementsprechend

ggf. **Standartwerte** setzen

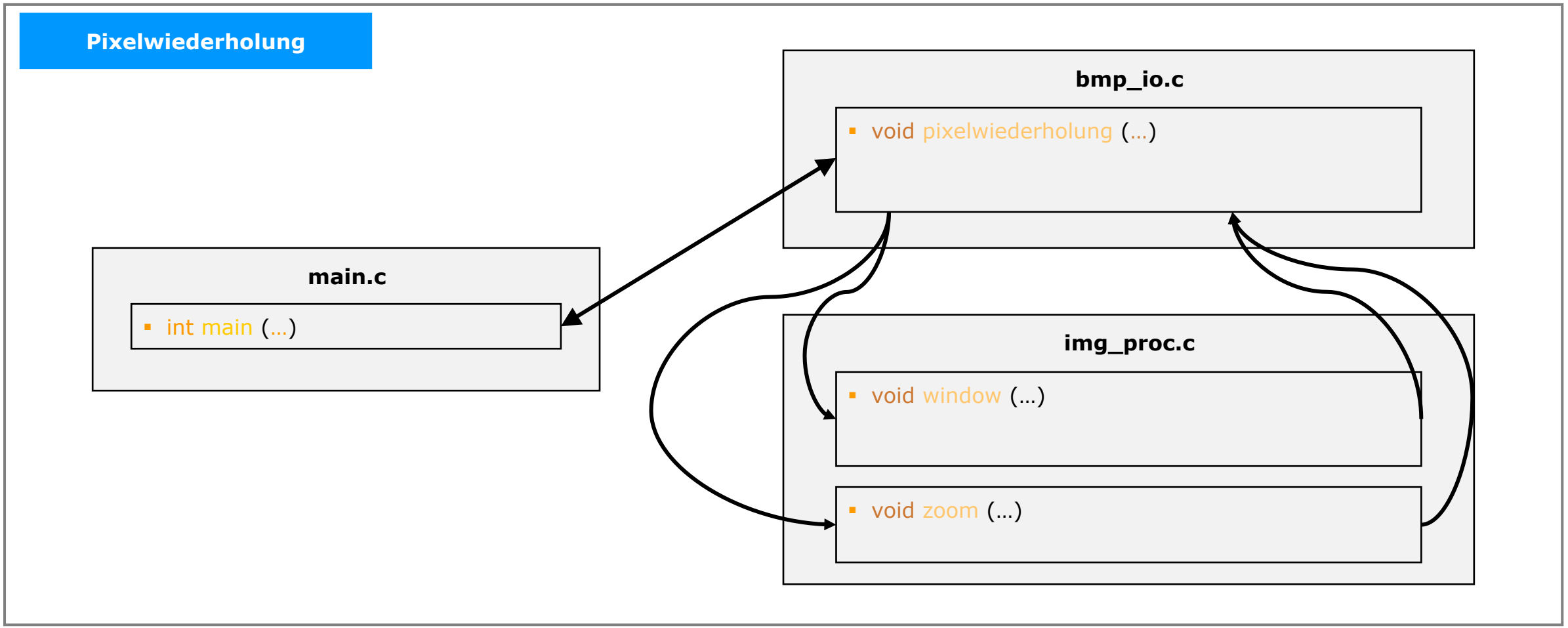
for (int i = 0; i < durchlaeufe; i++){
pixelwiederholung(input, output, x, y, width, height, scale, version); }

Ausführungszeit ausgeben

Beispielhafter Aufruf: -V 0 -B 1 ../images/hermann.bmp -s 2,15 -w 137 -h 5 -f 41 -o ../images/hermann_output.bmp

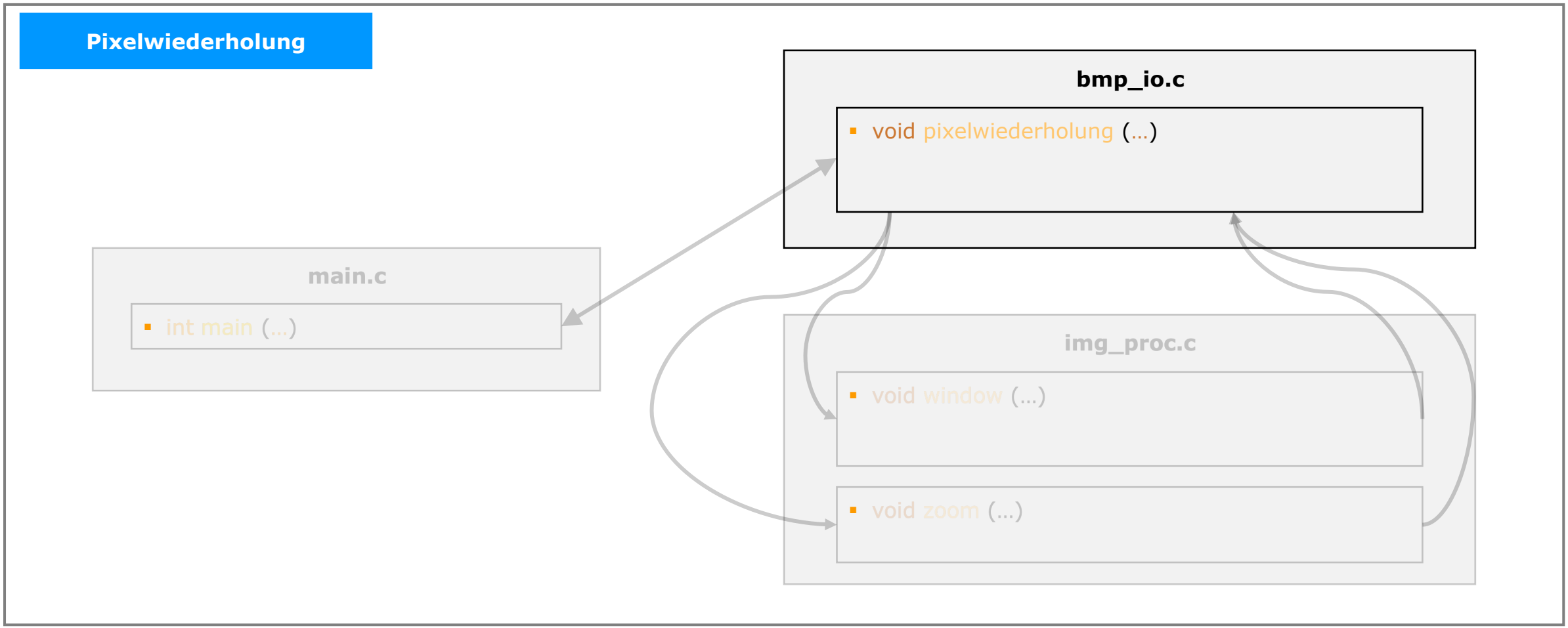
Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: pixelwiederholung

Projektaufgabe: Pixelwiederholung

Verwendung von Pseudocode!

Anforderung

- Öffnet die **Eingabedatei** und liest den **Header** sowie die relevanten Informationen aus.
- Überprüft die Gültigkeit der Eingabeparameter für den Ausschnitt des Bildes (x, y, width, height)
- **Liest** das **Pixel-Array** aus der Eingabedatei in den Speicher
- Ruft die Funktion **window** auf, um den gewünschten Bildausschnitt zu erhalten.
- Ruft die Funktion **zoom** auf, um den skalierten Ausschnitt des Bildes zu erhalten.
- **Erstellt** eine **neue Datei** mit dem angegebenen Ausgabepfad, schreibt den modifizierten Header und das **Pixel-Array** in die Datei und gibt den **Speicher frei**.

Umsetzung

- **void pixelwiederholung (...)**

fopen Inputdatei

ungültige Argumente für **x,y,width,heigth** abfangen (if-Bedingung)

window()

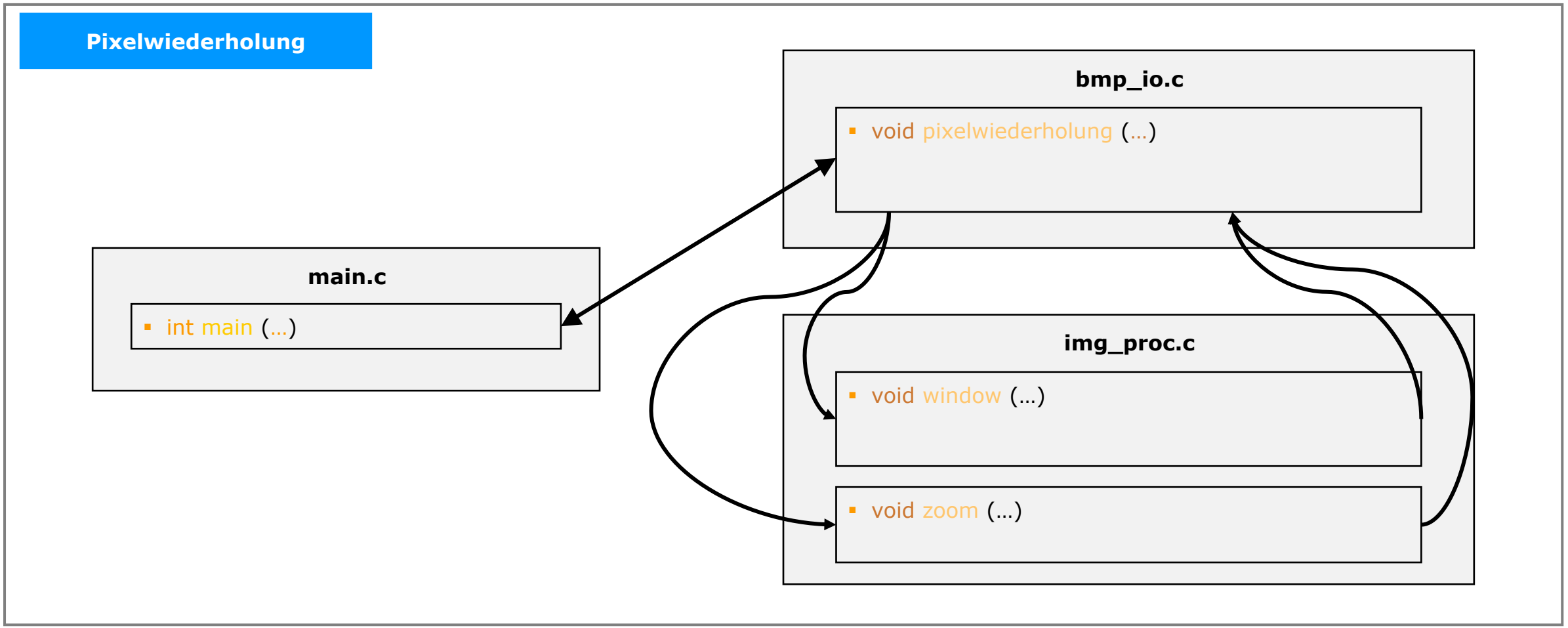
zoom()

fwrite () Outputdatei

free Memory

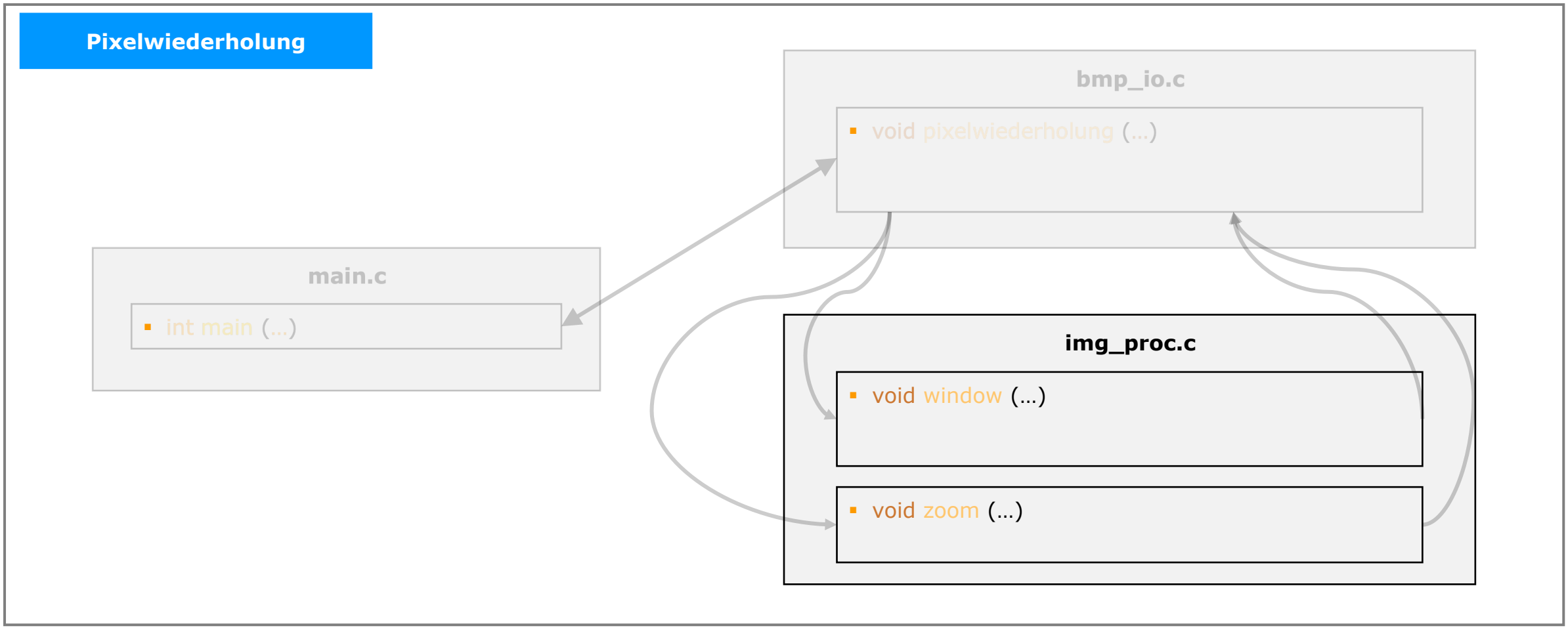
Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: Generelle Architektur

Projektaufgabe: Pixelwiederholung



Implementierung: img_proc.c (1/2) – window()

Projektaufgabe: Pixelwiederholung

Verwendung von Pseudocode!

Anforderung ⇄ **Umsetzung**

- **Eingabeparameter:** img, x, y, width, height, imageWidth, imageHeight
- **Ausgabeparameter**: result
- Berechnet den Zeiger **windowStart** zum ersten Pixel im Fenster, indem sie die **Ausgangskoordinaten** x und y, die Fensterbreite width und die **Fensterhöhe** height verwendet
- **Kopieren des Inhalts:** Die Methode kopiert den Inhalt des Fensters aus dem Eingabebild in das Ausgabebild result. Dabei werden die RGB-Werte (3 Bytes pro Pixel) für jede Zeile des Fensters kopiert und das Ergebnis entsprechend den Abmessungen von width und height im Speicher positioniert.
- **Berücksichtigung des Paddings:** ((width * 3) + (width % 4))

- **void window (...)**

windowStart berechnen

for (size_t row = 0; row < height; ++row){
kopieren der Zeile}

Implementierung: img_proc.c (2/2) – zoom()

Projektaufgabe: Pixelwiederholung

Verwendung von Pseudocode!

Anforderung → **Umsetzung**

- **Eingabeparameter:** img, width, height, scale_factor
- **Ausgabeparameter**: result
- Code berechnet verschiedene Größen und Parameter basierend auf der Eingabe und dem Skalierungsfaktor.
- **Schritt 1:** Es kopiert alle Originalpixel aus dem Eingabebild (**img**) in ihre skalierte Position im neuen skalierten Bild (**result**) und speichert die Adressen in einem temporären Array.
- **Schritt 2:** Es kopiert die benachbarten Pixel um jedes Pixel im skalierten Bild innerhalb eines angegebenen Radius.
- **Schritt 3:** Es füllt alle verbleibenden Lücken und Ränder im skalierten Bild, indem es RGB-Werte von benachbarten Pixeln kopiert.

- **void zoom (const uint8_t *img, size_t width, size_t height, size_t scale, uint8_t *result)**

width = 50, height = 35, scale = 10

Überblick

- Aufgabe und generelle Funktionsweise
- **Implementierung:**
  - **Rahmenprogramm**
  - **window()**
  - **zoom()**
- Testing
- Performance

Überblick

- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- **Testing**
- Performance

Testing: Edge Cases Überblick

Projektaufgabe: Pixelwiederholung

Rahmenprogramm

- **Übergabe falscher Optionen:** z.B. “-L <Zahl>”
- **Keine Angabe von Werten hinter Optionen:**
- **Keine Angabe von allen Optionen:**
  - **Keine Inputdatei/ falsches Dateiformat:** Fehler wird ausgegeben
  - **Keine Outputdatei:** das Originalbild wird überschrieben
  - **Alle anderen Optionen:** Standartwerte warden vergeben (Anzahl an Durchläufen = 1, Bildbreite = 40, Bildhöhe = 40)
- **Falsche Bildgröße:** Height/ width müssen > 0 sein
- **Falscher Skalierungsfaktor:** Skalierungsfaktor muss > 0 sein

pixelwiederholung()

- **Übergabe falscher Startpunkte:** Startpunkte (x,y) außerhalb des Bilder
- **Übergabe falscher Bildhöhe und Bildbreite:** Startpunkt x + width > Breite des Bildes oder Startpunkt y + height > Höhe des Bildes

```
test_zoom_scale_factor_3 erfolgreich durchgeführt.
test_window_single_pixel erfolgreich durchgeführt.
test_zoom_scale_factor_1 erfolgreich durchgeführt.
test_zoom_scale_factor_corner erfolgreich durchgeführt.
```

Überblick

- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- **Testing**
- Performance

Überblick

- Aufgabe und generelle Funktionsweise
- Implementierung:
  - Rahmenprogramm
  - window()
  - zoom()
- Testing
- **Performance**

Performance

Projektaufgabe: Pixelwiederholung

Zeitliche Performance

- Ausführungszeit verdoppelt sich etwa, wenn der Skalierungsfaktor verdoppelt wird (bei konstanter Fenstergröße).
- Der Skalierungsfaktor trägt ungefähr zu einer linearen (O(n)) Zeitkomplexität bei.
- Analog verhält es sich, wenn wir die Veränderung der Ausführungszeit betrachten, während die Fenstergröße vervierfacht wird.
- Sowohl die Fenstergröße als auch der Skalierungsfaktor trägt linear zur Zeitkomplexität bei.

Fenstergröße	Skalierungsfaktor	Durchschnittliche Ausführungszeit (Sek.)
128x128	1	0.014315
128x128	2	0.022671
128x128	4	0.036589
128x128	8	0.072948
512x512	1	0.060576
512x512	2	0.090332
512x512	4	0.202651
512x512	8	0.618855
1024x1024	1	0.163903
1024x1024	2	0.274887
1024x1024	4	0.758264
1024x1024	8	4.273953
2048x2048	1	0.339196
2048x2048	2	0.832601
2048x2048	4	3.018189
2048x2048	8	9.704604

Vielen Dank für die Aufmerksamkeit!

Gruppe 257