
Prof. Dieter Schmalstieg

Dr. Guido Reina

7. Übungsblatt zur Vorlesung Computergraphik im WS 2025/26

Besprechung am Montag, Dienstag und Mittwoch, 12.01., 13.01 & 14.01.2026

Aufgabe 1 *Framebuffer [1 Voterpunkte]*

1. Was ist ein Framebuffer und weshalb wird er eingesetzt?
2. Ein Highcolor-Framebuffer mit einer Farbtiefe von 16 Bit verwendet i.d.R. jeweils 5 Bit um Rot und Blau, sowie 6 Bit um Grün zu repräsentieren. Warum wird die Farbe Grün bevorzugt?
3. Wieviel Speicherplatz wird für einen unkomprimierten Film in 4K-Auflösung (3840 x 2160), 36 Bit Farbtiefe und einer Dauer von einer Stunde bei 24 Bildern pro Sekunde benötigt?

Aufgabe 2 *Repräsentation von Bildern [2 Voterpunkte]*

1. Warum ist Aliasing ein inhärentes Problem bei Rastergrafiken?
2. Wie entsteht Aliasing und wie kann man dem entgegenwirken?
3. Nennen Sie drei verschiedene Arten von Aliasing; denken Sie hierbei an Diskretisierungen im Allgemeinen.

Auf diesem Übungsblatt werden Sie sich mit der Implementierung von Filteroperationen auf Bildern beschäftigen. Das gegebene Programmskelett für dieses Übungsblatt baut auf einer angepassten Version der Bildklasse und der Funktionen des vorangegangenen Übungsblatts auf. Neben dem neuen Namespace `cg::filter::`, in dem Sie grundlegende Funktionen für Filteroperationen implementieren bzw. ergänzen werden, wurde das Programmskelett durch einen OpenGL-ImageViewer erweitert, den Sie in diesem Übungsblatt verwenden und vervollständigen werden. Für die Bearbeitung der Aufgaben werden keine OpenGL-Kenntnisse benötigt – diese werden Sie im weiteren Verlauf der Vorlesung aber noch erlangen.

Hinweis: Beachten Sie bitte, dass das Programmskelett für die Verwendung von OpenGL auf externe Open-Source-Bibliotheken zurückgreift. In der mitgelieferten CMakeLists.txt sind die Bibliotheken bereits eingebunden. Im gegebenen Programmskelett sind `glad`, `glfw`, `gowl` und `imgui` bereits als Quellcode enthalten. Diese Bibliotheken werden von CMake automatisch in den Buildprozess eingebunden. Somit müssen sie nicht separat installiert werden.

Das Programm-Skelett sollte bereits fehlerfrei kompilieren und ausführbar sein. Sobald Sie das Programm starten, öffnet sich ein Fenster, in dem ein Bild und eine simple GUI angezeigt wird. In der GUI können Sie den Pfad zu einem `.ppm`-Bild angeben, das Sie anzeigen und filtern möchten. Darüber hinaus bietet Ihnen die GUI die Wahl zwischen CPU- und GPU-Berechnung der Filteroperationen, sowie jeweils eine Auswahl unterschiedlicher Filter mit entsprechenden Einstellungen. Mit dem *Update*-Button wird der ausgewählte Filter auf das Eingabebild angewandt und das Ergebnis direkt im Fenster angezeigt. Je nach Konfiguration kann die Berechnung einige Sekunden in Anspruch nehmen. Im gegebenen Zustand wird jedoch noch keine Filterkonfiguration eine Änderung des Bildes verursachen. (Ausnahme: Die Ausführung des GPU-Filters funktioniert bereits.)

Hinweis: Beachten Sie bitte, dass in diesem Übungsblatt für die Verwendung von GPGPU-Computing mit einem OpenGL *Compute Shader* ein OpenGL-Kontext der Version 4.3 oder höher benötigt wird. OpenGL verwendet *Shader*-Programme, in der C-ähnlichen Sprache GLSL geschrieben, für die Programmierung moderner Grafikkarten. Diese Art der Programmierung werden Sie im Verlauf der Vorlesung näher kennen lernen.

Die erste Aufgabe dieses Aufgabenblattes kann optional ohne die GUI und den OpenGL-Kontext ausgeführt werden. Rufen Sie hierzu das kompilierte Programm über die Kommandozeile mit Parametern für das Ein- und Ausgabebild auf:

```
./ImageViewer[.exe] .../bilder/gingko.ppm .../bilder/gingko_edge.ppm
```

Bitte beachten Sie, dass die relativen Pfade für das Programm und die Ein- und Ausgabebilder vom aktuellen Ausführungsverzeichnis abhängig sind und ggf. angepasst werden müssen! Die (erzeugten) Bilder können z.B. mit einem Bildbearbeitungsprogramm wie GIMP oder Adobe Photoshop geöffnet werden.

Die folgenden Aufgabenblätter setzen eine OpenGL 4.3-kompatible Umgebung voraus. Sollten Sie keinen Zugang zu einem geeigneten Computer haben, können Sie einen VIS-Account beantragen und die PCs in den VIS-/VISUS-Pools verwenden.

Aufgabe 3 Filteroperationen auf der CPU [7 Votierpunkte]

Für Filteroperationen auf (diskreten) Bildern werden üblicherweise sog. Kernel verwendet. Eine entsprechende Klasse `Kernel` ist in `ImageFilter.h` bereits fertig implementiert und steht Ihnen zur Verfügung. Lesen Sie die Kommentare zu den einzelnen Funktionen der Klasse aufmerksam durch, um die Klasse korrekt zu verwenden!

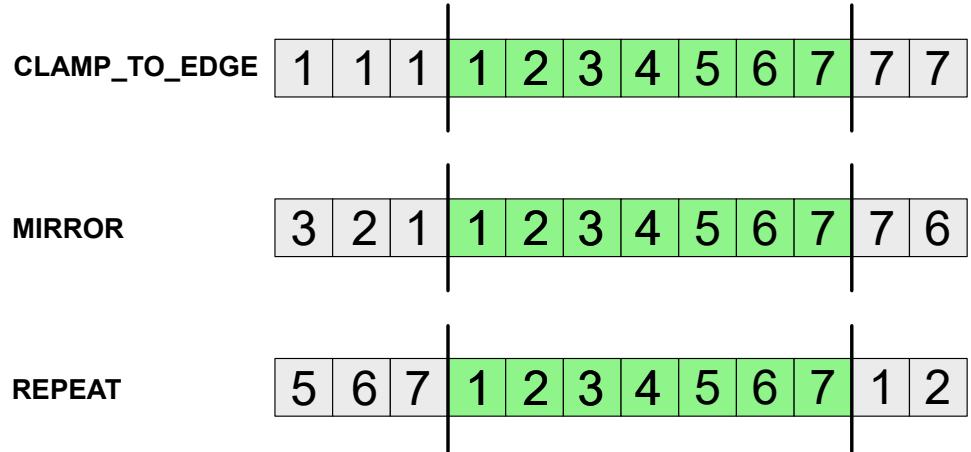
1. Implementieren Sie in der Funktion `filterImage`, in der Datei `ImageFilter.h`, die Anwendung des Filter-Kernels auf das Eingabebild. Iterieren Sie dazu über den Bereich des Filter-Kernels und berechnen Sie die Summe der Multiplikationen der Kerneleinträge mit den entsprechenden Bildwerten (siehe Kapitel 1, Folien 145ff. der Vorlesung). Verwenden Sie die Funktion `offsetImageCoordinates` zur Berechnung der passenden Bildkoordinaten für eine gegebene Position des Kernels *über* dem Eingabebild (in Bildkoordinaten) und einer gegebenen Position innerhalb des Kernels (Koordinaten relativ zur Mitte des Kernels).
2. Implementieren Sie anschließend die Funktion `buildEdgeDetectionKernel`, in der ein einfacher 3×3 Filter-Kernel zur Kantenerkennung erstellt werden soll (siehe Kapitel 1, Folie 158 der Vorlesung).
3. Um das Ergebnis des Kantenerkennungs-Filters anzuzeigen, müssen Sie nun in der Datei `ImageFilter.h` die Funktion `edgeDetection2D` ergänzen. Erstellen Sie in der Funktion einen entsprechenden Filter-Kernel und verwenden Sie die Funktion `filterImage` um den Filter-Kernel auf das Eingabebild anzuwenden. Wandeln sie dieses Zwischenergebnis anschließend mithilfe der Funktion `absValueImage` in ein Bild um, welches ausschließlich positive Werte in den einzelnen Farbkanälen enthält.
4. Für das *Weichzeichnen* eines Bildes verwendet man in der Computergrafik häufig einen Gauß-Filter. Dabei wird für jeden Pixel im Bild seine lokale Nachbarschaft gemäß einer zweidimensionalen Gauß-Glocke der Breite σ gewichtet. Die Funktionen `build2DGaussianKernel`, `build1DHorizontalGaussianKernel` und `build1DVerticalGaussianKernel` verwenden die Funktion `setGaussianValues` zur Berechnung der Einträge eines solchen Gauß-Filters. Implementieren Sie die Funktion `setGaussianValues` mit folgender Formel:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

wobei x und y der horizontale und vertikale Abstand vom Zentrum des Kernels in Pixeln sind. Normalisieren Sie anschließend die Einträge des Kernels. Was passiert, wenn keine Normalisierung angewendet wird?

5. Ergänzen Sie die Funktionen `gaussian2D` und `seperatedGaussian2D` in `ImageFilter.h` analog zur Implementierung von `edgeDetection2D`, um die Gauß-Filter im Programm verwenden zu können. Für den separierten Gauß-Filter wird zunächst ein eindimensionaler Filter zunächst in horizontaler Richtung und anschließend nochmals in vertikaler Richtung auf das Bild angewandt. Welchen Vorteil hat dieses Vorgehen?
6. Ein wesentlicher Aspekt bei der Anwendung von Filtern ist das Verhalten am Bildrand, da dort für einen Teilbereich des Filter-Kernels keine zugehörigen Bildwerte mehr verfügbar sind. In der Funktion `offsetImageCoordinates` in `ImageFilter.h` werden potentielle Zugriffe außerhalb des Bildbereiches erkannt und nach unterschiedlichen Strategien auf gültige Koordinaten innerhalb des Bildes abgebildet. Die Strategie `CLAMP_TO_EDGE`, die alle Zugriffe außerhalb des Bildbereiches auf den Rand des Bildes abbildet, ist bereits implementiert. Ergänzen Sie die Strategie `MIRROR`, die Zugriffe am Bildrand spiegelt, sowie die Strategie `REPEAT`, die das Bild in allen Richtungen virtuell wiederholt, indem Zugriffe außerhalb des

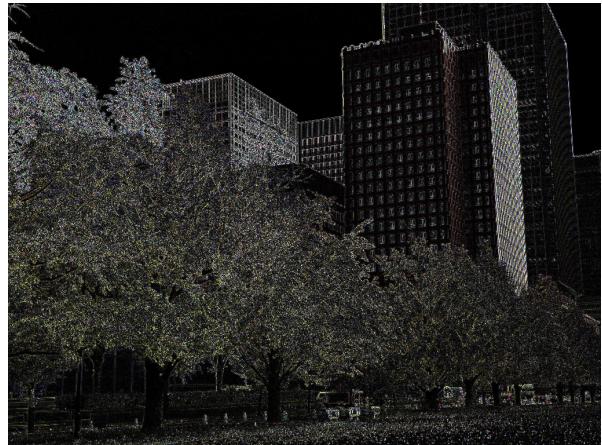
Bildbereiches auf gültige Koordinaten am gegenüberliegenden Bildrand abgebildet werden. Das ist beispielhaft hier dargestellt:



7. Experimentieren Sie mit unterschiedlichen Konfigurationen der CPU-basierten Filter, um Ihre Implementierung zu testen. In Abbildung 1 können sie einige Beispielbilder finden, um Ihre Lösung zu vergleichen. Erklären Sie, welches Problem bei der Kombination aus Kantendetektion und REPEAT entstehen könnte.



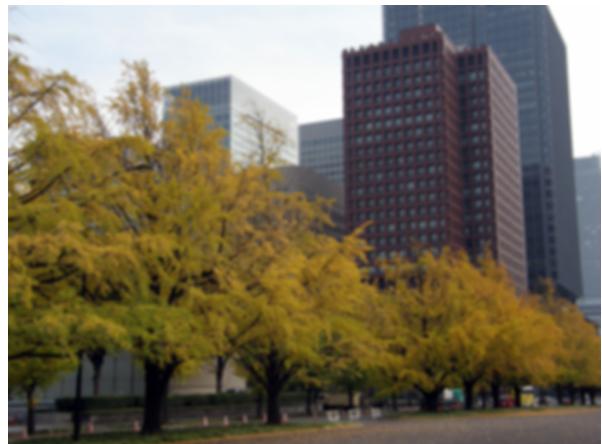
(a) Original



(b) Kantendetektion



(c) Gauß, 2×2 Kernel, $\sigma = 3$



(d) Gauß, 5×5 Kernel, $\sigma = 3$



(e) Gauß, 4×1 Kernel, $\sigma = 3$



(f) Gauß, 1×4 Kernel, $\sigma = 3$

Abbildung 1: Original und mögliche Lösungen für unterschiedliche Einstellungen.

Quelle: Tomi Mäkitalo “Ginkgo biloba in autumn near Tokyo Station 秋の行幸通りのイチョウ並木 - panoramio”, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>