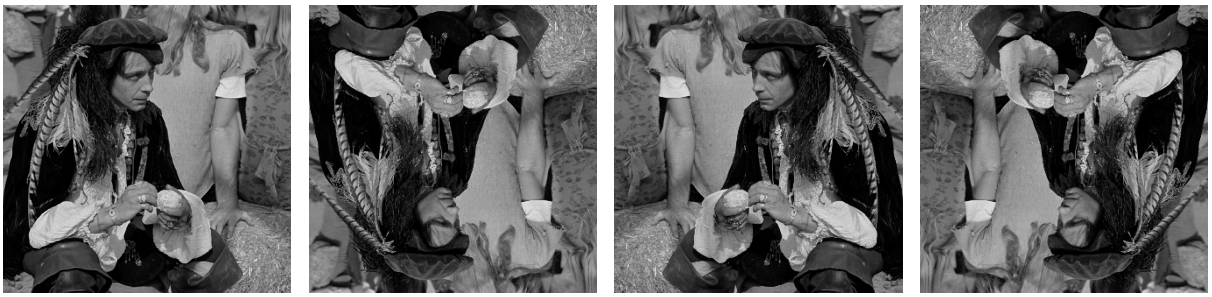
GdCV

Lab 1



(a) (b) (c) (d)

Abbildung 1 Das Originalbild (a) wird mittels einfacher geometrischer Operatoinen „geflippt“ (b), „gefloppt“ (c) und geflipfloppt“ (d).

In diesem Praktikum werden wir grundlegende Bildverarbeitungsoperationen besprechen. Die Methoden die sie implementieren werden von Testmethoden in `main.cpp` aufgerufen. Die Ausgabebilder finden Sie in Ihrem Build-Ordner `GdCVBuild\exercises\BasicImageProcessing`.

Woche 1: Simple Geometric Transformations

Testmethode: `testSimpleGeometricTransformations.`

Code-Dateien: `SimpleGeometricTransforms.h, SimpleGeometricTransforms.cpp, Image.h, Image.cpp`

Aufgabe 1 Image-Klasse

Die Klasse `Image` verwaltet ein Bild im Arbeitsspeicher. Der Datentyp des Pixels wird als Template-Argument `T` übergeben. Implementieren Sie die *Rule-of-Five* und die fehlenden Methoden. Beachten Sie hierzu die Dokumentation in `Image.h`.

Nun werden in `Image.h` folgende Datentypen deklariert:

- ImageG8 für Graustufenbilder mit 8 Bit Farbtiefe und
- ImageRGB8 für Farbbilder mit 8 Bit Tiefe je Farbkanal.

Dazu wird als Template-Argument dazu `uint8` bzw. `ui8vec3` und die notwendig Template-Instanziierungen werden in `Image.cpp` durchgeführt.

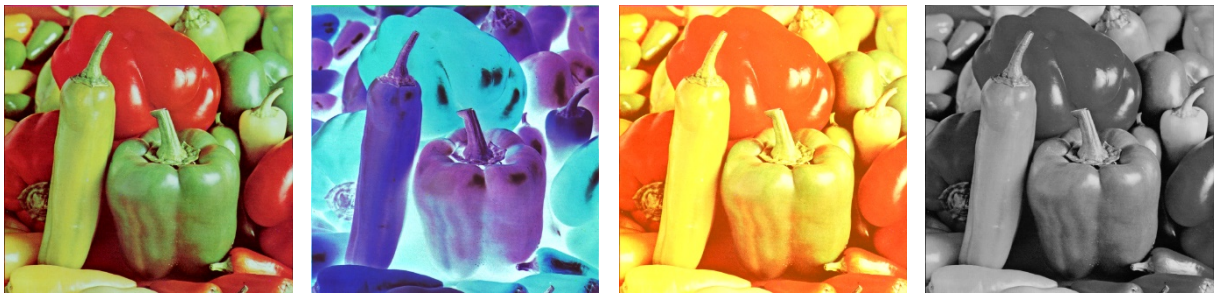
Aufgabe 2 Simple Geometric Transforms

In `SimpleGeometricTransforms.h` finden Sie oberhalb der Deklaration von den Methoden einen kurzen Kommentar zu deren Funktionalität. Vervollständigen Sie nun die Definition der Methoden in `SimpleGeometricTransforms.cpp` anhand dieser Kommentarinformationen. Abbildung 1 zeigt Ergebnisse einiger einfacher geometrischer Transformationen.

Woche 2: Point Transformations

Testmethode: `testPointTransformations.`

Code-Dateien: `PointTransformations.cpp, PointTransformations.h`



(a) (b) (c) (d)

Abbildung 2 Aus einem Bild (a) wir ein Negativbild (b), ein Bild mit Sepiaeffekt (c) und ein Graustufenbild (d) mittels Punkttransformationen berechnet.

Aufgabe 3 Saturation Arithmetik

Im anonymen Namespace von `PointTransformations.cpp` müssen Sie die beiden Funktion `sat` gemäß deren Beschreibung in den Kommentaren implementieren.

Aufgabe 4 Point Transformations

Nutzen Sie nun die beiden Methoden `sat` und die Informationen aus der Vorlesung um die freien Methoden `negativeImage`, `gain`, `bias`, `toGrayScale` und `blend` gemäß deren Beschreibung in den Kommentaren zu implementieren. Sie sollten Ergebnisse wie in Abbildung 2 erhalten.

Woche 3: Linear Transformations

Testmethode: `testGeometricTransformations`.
Code-Datei: `GeometricTransforms.h, GeometricTransforms.cpp`

Aufgabe 5 Lineare Transformationen

Implementieren Sie `GeometricTransforms::applyTransform`, welche ein Bild gegen den Uhrzeigersinn rotiert. Dabei soll für jeden Zielpixel die Koordinate mit einer homogenen Transformation transformiert werden. Die resultierende Koordinate soll genutzt werden, um vom Quellbild den zugehörigen Pixel zu lesen und auf das Zielbild zu übertragen.

Sollte die resultierende Koordinate jedoch außerhalb des gültigen Bereichs des Quellbilds liegen, so soll das Zielbild an der Koordinate die Farbe schwarz annehmen.

Hinweise:

- `f32mat3` repräsentiert eine 3x3 Matrix. Der Konstruktor nimmt neun Werte entgegen. Achtung: Diese befinden sich jedoch in Column-Major- (und nicht Row-Major-) Anordnung!
- `f32vec3` repräsentiert einen 3er Vektor.
- Dank C++ Operator-Overloading können Sie ein `f32mat3` Objekt mit einem `f32vec3` Objekt mittels des `*`-Operators verknüpfen. Das Ergebnis ist ein `f32vec3` Objekt, welches das Ergebnis der entsprechenden Matrix-Vektor Multiplikation beinhaltet.

Abbildung 3 Bild (Links) wird um +30° (Mitte) und um -30°rotiert um das Bildzentrum mittels linearer Transformationen rotiert.

Aufgabe 6 Rotation ums Bildzentrum

Implementieren Sie `GeometricTransforms::rotateAroundCenter` und nutzen Sie dabei `GeometricTransforms::applyTransform`. Konstruieren Sie die entsprechende Transformation, welche eine Rotation um das Bildzentrum durchführt.

In Abbildung 3 finden Sie eine mögliche Ausgabe des Testprogramms.

Woche 4: Filter

Die Testmethode: `testFilter`.
Code-Datei: `Filter.h, Filter.cpp`

Aufgabe 7 1D Filter in horizontaler Richtung

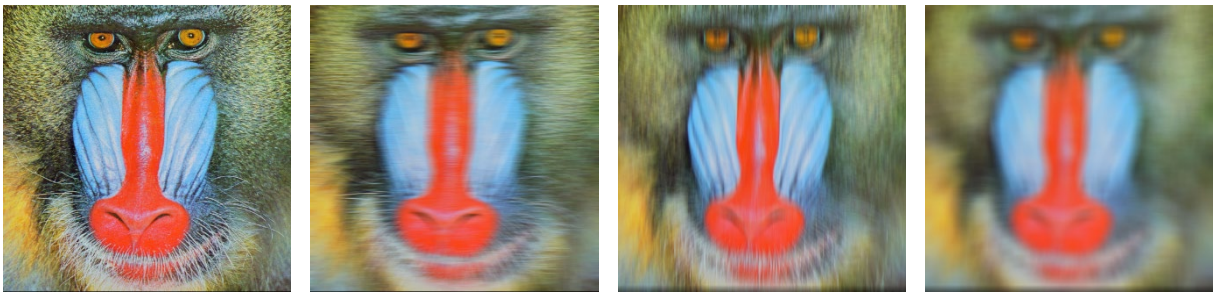
Implementieren Sie `filter1D_X_Clamp`, welche die Filterung eines Bildes mit einem Filter-Kern in horizontaler Richtung ausführt (vgl. Abb. Abbildung 4b). Verwenden Sie dabei die Clamp-Randbehandlung. Achten Sie auf Saturierungsarithmetik!

Aufgabe 8 1D Filter in vertikaler Richtung

Implementieren Sie analog zu Aufgabe 2 `filter1D_Y_Clamp`, welche die Filterung eines Bildes mit einem Filter-Kern in vertikaler Richtung ausführt (vgl. Abbildung 4c).

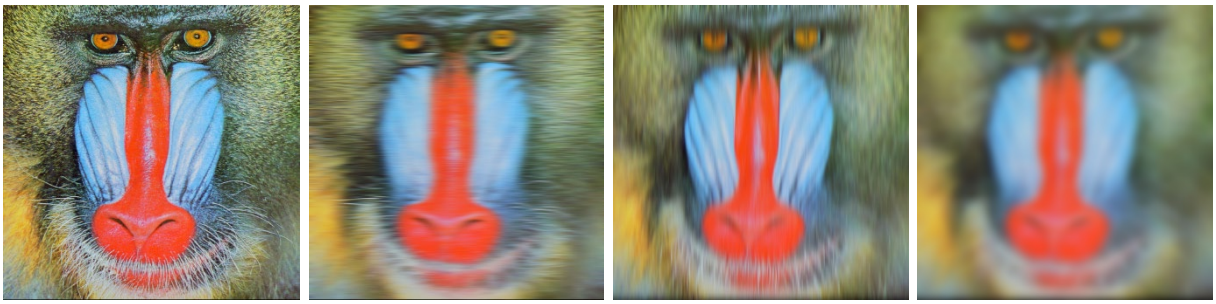
Aufgabe 9 2D Filter mit separierbaren Filtern

Implementieren Sie nun `filter2D_XY_Sep_Clamp`, welche die Filterung zunächst in horizontaler und dann in vertikaler Richtung ausführt (vgl. Abbildung 4d).



(a) (b) (c) (d)

Abbildung 4 Ein ungefiltertes Bild (a) wird box-gefiltert und zwar (b) in x Richtung, (c) in y Richtung sowie (d) in x und y Richtung.



(a) (b) (c) (d)

Abbildung 5 Ein ungefiltertes Bild (a) wird Gauß-gefiltert und zwar (b) in x Richtung, (c) in y Richtung sowie (d) in x und y Richtung.

Aufgabe 10 Tensorprodukt

Implementieren Sie `tensorProduct`, welche mit dem Tensor-Produkt aus zwei 1D Kernels einen 2D Kernel berechnet.

Aufgabe 11 2D Filter mit 2D Filter Stencils

Implementieren Sie `filter2D_XY_Clamp`, welche die Filterung eines Bildes mit einem 2D Filter-Stencil durchführt. Das Ergebnis sollte wie in Abb. 1d aussehen. Vergleichen Sie die Laufzeitunterschiede, welche im Ausgabefenster in Millisekunden ausgegeben werden.

Aufgabe 12 Gauß-Filter statt Box Filter

In `gaussianStencil` wird bisher ein Box-Filter konstruiert. Berechnen Sie stattdessen einen Gauß-Filter mit der passenden Anzahl an positiver Taps! Passen Sie dazu auch `nPosTaps` an!

Das Ergebnis sollte wie in Abbildung 5 aussehen.