

MMS-Photo-Edit

PROJEKTABSCHLUSSBERICHT

MULTIMEDIASYSTEME SS22

Team

| Name | Matr. Nr. |
|--------------------|------------------|
| Nico Layr | K11771407 |
| David Lefort | K01556644 |
| Andreas Froschauer | K12103798 |

Einleitung

Wie dem Projektantrag entnommen werden kann, war die Vision eine Online-Tool zu schaffen, dass einfache und performante Bildbearbeitung ermöglicht und das ganz unabhängig von der Rechenleistung der jeweiligen Clients. Dies wurde nur teilweise erreicht.

Problem 1: Internetgeschwindigkeit

Wie es im Web nunmal ist, müssen Daten erst von Server auf PC übertragen werden und vice versa. Vor allem das erste Hochladen des Bildes dauert je nach Größe am längsten. Bei der Implementierung wurde jedoch darauf geachtet, dass Bilder nicht erneut hochgeladen werden müssen um Filter anzuwenden, sondern diese am Server zwischengespeichert werden. Somit fällt bei der Anwendung von mehreren Filtern auf ein Bild ein Großteil der nötigen Übertragung weg. Der Download ist hingegen natürlich weiterhin von Nöten. Daher kommt es vor allem bei hochauflösenden Bildern zu einer merkbaren Wartezeit beim Download.

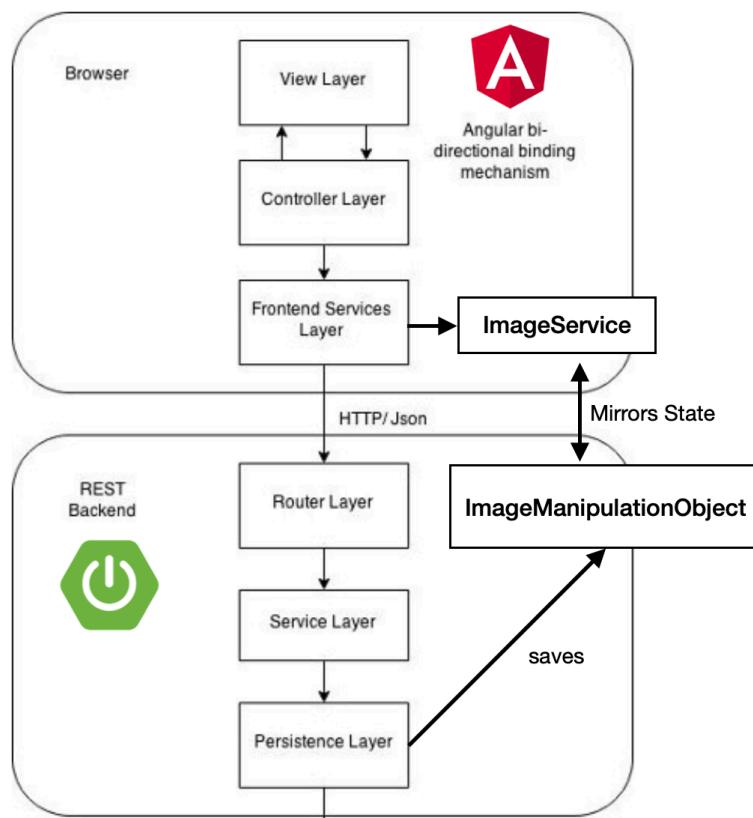
Problem 2: Bildbearbeitung mittels CPU

Die Bildbearbeitung Serverseitig konnte aufgrund von Mangel an Kenntnissen und Zeit nicht mittels Grafikkarte implementiert werden. So wird derzeit mittels CPU Pixel für Pixel abgegangen und der gewünschte Filter angewendet. Dies kostet natürlich Zeit und ist so nicht im Sinne der oben genannten “performante[n] Bildbearbeitung”. Auch kommt es auf einigen Testgeräten zu einem “HeapOutOfMemory”.

Problem 3: Abgang eines Teammitglieds

Ziemlich zu Beginn verließ uns ein Teammitglied, dies wirkte sich natürlich auf das Projekt aus. Es gibt derzeit sicherlich noch einige Bugs, vor allem im Bereich der gesiegelten Funktionen von “ImageService” im Frontend und “ImageManipulationObject” im Backend (dazu später mehr).

Architektur

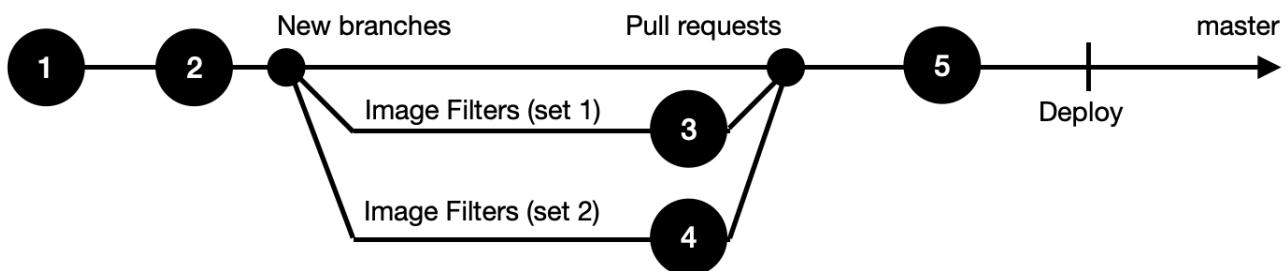


Kollaboration

Als Collaboration-Tool setzen wir auf git bzw. Github.

Unterhalb finden Sie eine Grafik, die abstrakt unseren Projektverlauf widerspiegelt.

1. Initial Commit (master): Angular und Spring Projekte erstellt
2. Core Commit (master): Frontend und Backend sind soweit fertig, um Filter zu implementieren und zu testen
3. Implementierung der Filter (set 1) *
4. Implementierung der Filter (set 2) *
5. Bufixes



Implementierung

Da es sich hierbei um ein Projekt für Multimediasysteme handelt, wird auf die Implementierung des Frontends und des generellen Backends nicht eingegangen. Viel mehr möchten wir die restlichen Seiten nutzen, die einzelnen Filter etc. zu beschreiben.

Selection

Man kann, wenn man nicht das ganze Bild durch Filter manipulieren will, sondern nur einen Teil davon, einen spezifischen Bereich auswählen. Auf technischer Seite heißt das, dass ein separates Raster aus Boolean Werten, welches genauso groß ist wie das Bild, zu jedem Filter mitgeliefert werden muss. Jeder Pixel hat also einen zugehörigen Boolean Wert, welcher True ist, wenn der Pixel ein Teil der Auswahl ist, False wenn nicht.

Es gibt zwei Selection-Modi: Rechteck- und Kreisauswahl:

Rechteck

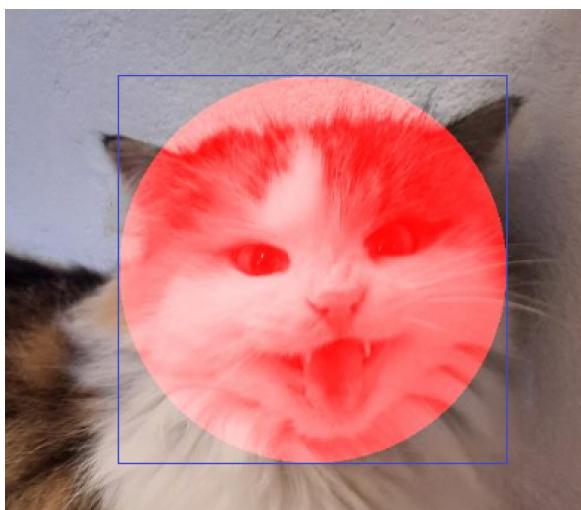
Für die Rechteck Auswahl bekommt man als Parameter den linken oberen Punkt des Rechtecks, sowie die Höhe bzw. Breite der Auswahl.

Der Code läuft mithilfe von Nested-Loops durch jeden Pixel des Rechtecks durch und setzt diese Werte auf True.

Die Rechtecks Auswahl ist bei weitem die Simpleste der drei, weil kaum Berechnungen ausgeführt werden müssen.

Kreis

Für die Kreis Auswahl bekommt man als Parameter den Mittelpunkt und den Radius des Kreises. Die Implementierung rennt wieder mithilfe von Nested-Loops durch das Bild und berechnet für jeden Pixel, ob die Entfernung zum Mittelpunkt des Kreises kleiner ist als der Radius. Wenn diese Bedingung zutrifft, ist der Pixel logischerweise Teil der Auswahl. Aus Optimierungsgründen werden nur die Pixel kontrolliert, die in dem Quadrat mit der Seitenlänge = $2 * \text{radius}$ und als Mittelpunkt den Kreismittelpunkt hat (siehe Bild).



Die Auswahl (rot) wird nur in dem Bereich innerhalb des blauen Quadrats kontrolliert.
Somit müssen weniger Pixel berechnet werden.

Die Entfernung zwischen Mittelpunkt(M) des Kreises und einem Pixel(P) wird berechnet, indem man zuerst den Vektor zwischen ihnen berechnet,

$$\vec{v} = P - M \text{ und dann die Länge dieses Vektors.}$$

$$\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \quad |\vec{v}| = \sqrt{x^2 + y^2}$$

Edge-Colorization

Der Edge-Colorization Filter besteht einerseits aus dem Erkennen der Kanten in einem Bild (Edge-Detection), andererseits dem Färben dieser Kanten.

Kantenerkennung

Das Bild wird in drei zweidimensionale Arrays getrennt, jeweils eines für eine Farbe (RGB). Danach wird auf diese drei einzelnen Arrays eine Convolution-Matrix (in diesem Fall „Sobel-Matrix“) angewandt. Die drei neuen Arrays werden nun wieder pixelweise zusammenaddiert zu einem Array. Schlussendlich werden noch Ausreißer, welche Werte über 255 oder unter 0 haben korrigiert. Pixel, die als Kanten erkannt wurden, werden nun eher höhere Werte haben, als Pixel, die als keine Kanten erkannt wurden.

Da Kantenerkennung mit dieser Matrix nur entweder horizontale oder vertikale Kanten erkennt, muss dieser Algorithmus zweimal ausgeführt werden, einmal mit einer horizontalen und einmal mit einer vertikalen Sobel-Matrix.

$$\begin{array}{ccc} \text{Horizontale Sobel-Matrix: } & \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} & \text{Vertikale Sobel-Matrix: } \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} \end{array}$$

Kantenzeichnung

Die Färbung der Kanten benutzt nun das Ergebnis der Kantenerkennung. Mit dem Parameter „Threshold“, der vom Nutzer bestimmt wird, werden die Kanten gefärbt. Das Threshold hat eine Reichweite von -100 bis 100. Dieser Wert wird auf die Reichweite 0 bis 255 skaliert.

Eine Nested-Loop, die über dem Array mit der Kantenerkennung iteriert, zeichnet die Edge-Farbe, wenn das Threshold von dem Wert des Pixels erreicht wird, sonst die Hintergrundfarbe. Da jeder der Pixel zwei Werte hat, einen für die horizontale und einen für die vertikale Erkennung, wird das Threshold mit dem Maximum der beiden Werte verglichen.

Edge-Farbe und Hintergrundfarbe werden beide vom Nutzer bestimmt.



Bildquelle: Pixabay.com

Greyscale

Für den Greyscale Filter muss man für jeden Pixel des Bildes den Durchschnitt des Rot-, Grün-



und Blauwerts berechnen und diesen Pixel dann mit der Farbe ersetzen, welche als Rot-, Grün und auch als Blauwert diesen Wert hat.

Beispiel:

Pixel im Bild hat die Farbe: (231, 200, 49) -> Durchschnitt ist $(231+200+49) / 3 = 160$

Der Pixel bekommt also die neue Farbe (160, 160, 160).

RGB-Manipulation

Der RGB-Manipulation Filter ändert die Teilwerte (Rot, Grün und Blau) des Bildes basierend auf vom Nutzer gegebenen Parametern. Diese Parameter sind im Wertebereich von -100% bis +100%.

Man braucht für die Berechnung der neuen Farbe zwei Formeln, eine für Prozentwerte über 0 und eine für Werte unter 0:

Für Prozentwerte über 0 (F_1 = neuer Farbwert, F_0 = alter Farbwert, P = Prozentwert):

$$F_1 = F_0 + (255 - F_0) * P / 100$$

Für Prozentwerte unter 0 (F_1 = neuer Farbwert, F_0 = alter Farbwert, P = Prozentwert):

$$F_1 = F_0 - F_0 * P / 100$$

Beispiel:

Ein Pixel hat die Werte (R: 200, G: 50, B: 100)

und der Nutzer gibt folgende Parameter: (R: -50%, G: 100%, B: 50%)

Der Rot Wert wird berechnet, indem man 50% seines Originalwerts subtrahiert:

$$R = 200 - 200 * 50 / 100 = 100$$

Der Grün Wert wird berechnet, indem man 100% seiner Differenz zu 255 zu 100 addiert:

$$G = 50 + (255-50) * 100 / 100 = 255$$

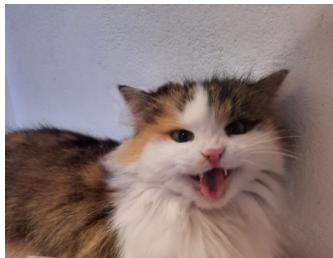
Der Blau Wert wird berechnet, indem man 50% seiner Differenz zu 255 zu 50 addiert:

$$B = 100 + (255-100) * 50 / 100 = 177.5$$

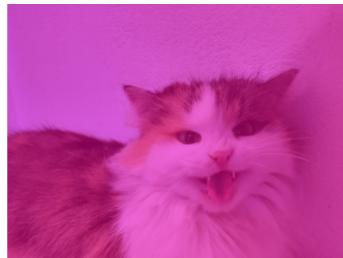
Der neue Pixel hat also die Werte (100, 255, 177).

Weitere Beispiele:

Original:



Parameter (50, -50, 30)



Parameter (-50,0,-30)



Parameter(40,40,-50)



Gaussian Blur

Als Beispiel für einen Blur Filter haben wir uns für den Gaussian-Blur entschieden. Dieser nimmt von jedem Pixel die in einem gewissen Radius umliegenden Pixel und errechnet einen Farbwert daraus. In der Regel werden weiter innen liegende Pixel stärker gewichtet und beeinflussen daher den neu berechneten Farbwert mehr. Die Gewichte der Matrix werden mit der Gaußschen Funktion für den zwei-dimensionalen Raum berechnet, wobei das Sigma die Varianz darstellt.

Bei der Implementierung musste beachtet werden, dass Koordinaten der Gewichtungsmatrix außerhalb des Bildes liegen können. Wenn eine Koordinate ungültig ist, wird die Koordinate auf einen passenden Wert gesetzt (siehe Implementierung BlurFilter). Es ist auch noch anzumerken, dass die Implementierung nicht auf Performanz optimiert ist. Weiterführend wäre es aber möglich, die Laufzeit des Filters beispielweise mithilfe eines C++ Scripts, das die GPU anstatt der CPU verwendet, zu verbessern.

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

Figure 1 Gaußsche Funktion
https://en.wikipedia.org/wiki/Gaussian_blur

Original:



Blur:



Color Inversion

Ein weiterer Filter in unserem Foto Editor ermöglicht es dem Benutzer die Farben umzukehren, d.h. das Inverse der Farbe wird berechnet.

Das Inverse lässt sich mit einer einfachen Differenz von dem RGB Wert für Weiß (255) und dem jeweiligen Rot-, Grün- und Blauwert des RGB Tripels eines Pixels berechnen. Beispielsweise wird Schwarz in Weiß umgewandelt oder Rot in Cyan.

Original:



Invertiert:



Bildrotation

Hier hatten wir verschiedene Möglichkeiten der Implementierung. Wir mussten entscheiden, ob wir eine volle 360 Grad Rotation unterstützen oder ob es dem Benutzer nur erlaubt ist, aus einer bestimmten Anzahl an Rotationsmöglichkeiten wählen zu können. Schlussendlich und auch aufgrund dessen, dass in unserer Gruppe ein Mitglied kurzfristig das Team verlassen hatte, haben wir uns dazu entschieden, nur eine 90 Grad Links- und Rechtsdrehung und eine 180 Grad Drehung zu erlauben. Da beliebige Werte für die Grad als Input möglich sind, muss man zuerst Modulo 360 rechnen und wie in der Implementierung (rotateManipulation) dann in die entsprechende Richtung rotieren.

Bei der Implementierung von Links- und Rechtsrotation war zu beachten, dass nicht nur die Bildinformation, sondern auch das Bildformat bei der Rotation des ganzen Bildes angepasst werden muss. Die Höhe des rotierten Bildes ist die Breite des Originalbilds und analog wird als neue Breite die Höhe des Originalbilds verwendet. Dann muss man nur noch die Pixel auf die richtige Position mappen. Im Falle der 180 Grad Rotation wenden wir einfach den Spiegelfilter (horizontal) an. Die Dimensionen des Bilds bleiben hier gleich.

Rechts 90 Grad:



Links 90 Grad:



Spiegelung

Hier kann zwischen horizontaler und vertikaler Spiegelung ausgewählt werden. Das Vorgehen bei der Implementierung dieser beiden Filter war sehr ähnlich. Man musste nur die Farbwerte links und rechts bzw. oben und unten vertauschen.

Horizontal bzw. Rotation 180 Grad:



Vertikal:



Weiterführende Links

- [Kurze Einführung in die Bildbearbeitung](#)
- [Angular](#)
- [Spring](#)
- [REST](#)
- [CORS](#)
- [Convolution Filter](#)
- [Gaussian Blur](#)
- [Edge Colorisation](#)