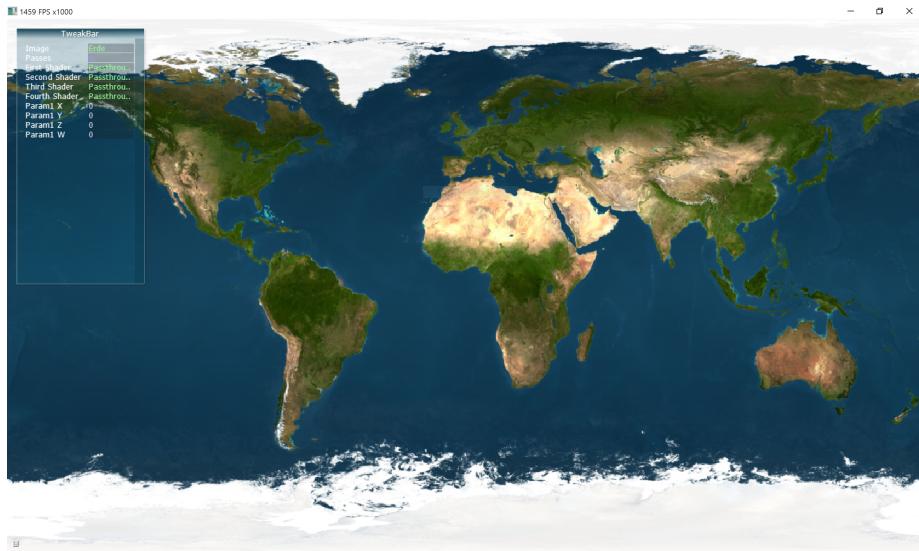


# Contents

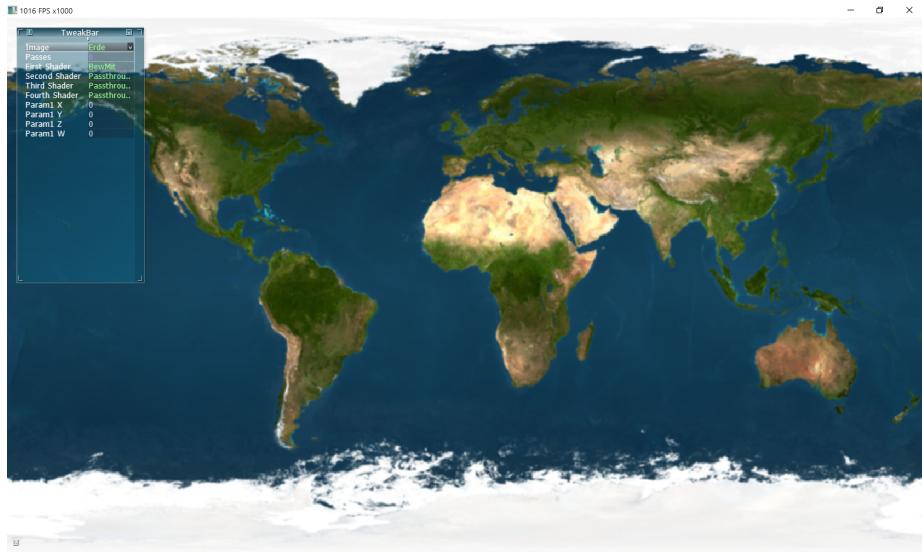
<b>Das Ursprungsbild der Aufgaben 1.1 und 1.2</b>	<b>2</b>
<b>Aufgabe 1.1</b>	<b>2</b>
<b>Aufgab1 1.2</b>	<b>3</b>
1. Brightness_Contrast . . . . .	3
2. Gauss3x3 . . . . .	4
3. Gauss5x5 . . . . .	5
4. Gauss7x7 . . . . .	6
Zeiteinsparung durch Aufteilung in Vertikal und Horizontal . . .	6
5. Laplace . . . . .	7
6. Sobel . . . . .	7
7. Sharpen . . . . .	8
8. Dilatation . . . . .	9
9. Erosion . . . . .	10
<b>Aufgabe 1.3</b>	<b>10</b>
1. Störungen entfernen . . . . .	10
2. Kanten extrahieren . . . . .	12
3. Binärbild erstellen . . . . .	14

## Das Ursprungsbild der Aufgaben 1.1 und 1.2



### Aufgabe 1.1

Bei Verwendung des Filters BewMit erscheint das Bild verschwommen. Der BewMit-Filter nimmt eine Einheitsmatrix, um alle Pixel im Umfeld des aktuellen Pixels mit der gleichen Gewichtung zusammen zu rechnen. Details verschwimmen somit aus dem Bildbereich. Eine Alternative zum erzielen des selben Ergebnis findet sich in Aufgabe 1.2, Teilaufgabe 2.



## Aufgabe 1.2

### 1. Brightness\_Contrast

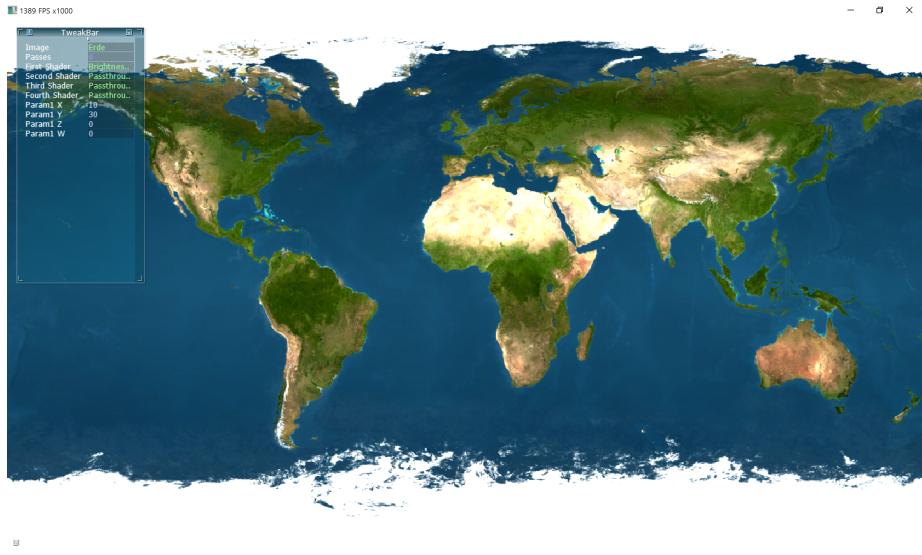
Der Brightness\_Contrast-Filter soll es möglich machen über den X- und Y-Parameter des Programms die Helligkeit(X) und den Kontrast(Y) des eingegebenen Bildes zu verändern. Zur Erhöhung der Helligkeit müssen alle Werte eines Pixel-Vektors um den gleichen Wert erhöht werden. Somit wird lediglich der vom Nutzer angegebene Wert x durch 100 geteilt und anschließend auf den Pixel-Vektor addiert:

```
fragColor += param1.x / 100.0;
```

Zur Erhöhung des Kontrastes muss der Pixel-Vektor mit einem Faktor skaliert werden. Außerdem wird nach der Faktorisierung noch eine Korrektur addiert, um die Helligkeit des Bildpixels nicht zu verändern:

```
fragColor = scale * fragColor + 0.5 * (1.0 - scale) * vec4(1);
```

Im letzten Schritt muss mithilfe von clamp verhindert werden, dass ein Pixel-Vektor nach der Anpassung von Helligkeit und Kontrast Werte > 1 enthält.

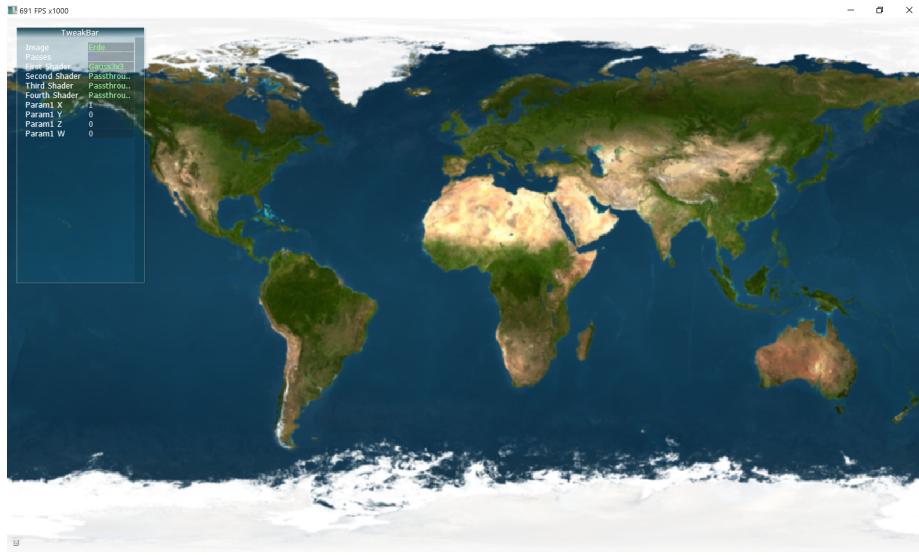


## 2. Gauss3x3

Im Gegensatz zum BewMit-Filter wird beim Gauss-Filter ein Algorithmus zur Generierung der Gewichtung der umgebenden Pixel verwendet. Diese Funktion ist abhangig von den gegebenen Positionen der Pixel und einer vom Nutzer konfigurierbaren Varianz. Die Funktion zur Berechnung der Impulsantwort findet sich [hier](#). Dies ist die entsprechende Umsetzung im Shader:

```
float varianz(vec2 cords){  
    float a = 1/(2.0 * 3.145 * pow(param1.x,2));  
    float ba = pow(cords.x,2)+pow(cords.y,2);  
    float bb = 2*pow(param1.x,2);  
  
    return a * exp((ba/bb)*(-1.0));  
}
```

Durch Erhöhung der Varianz über den Parameter X wird die Unschärfe erhöht.



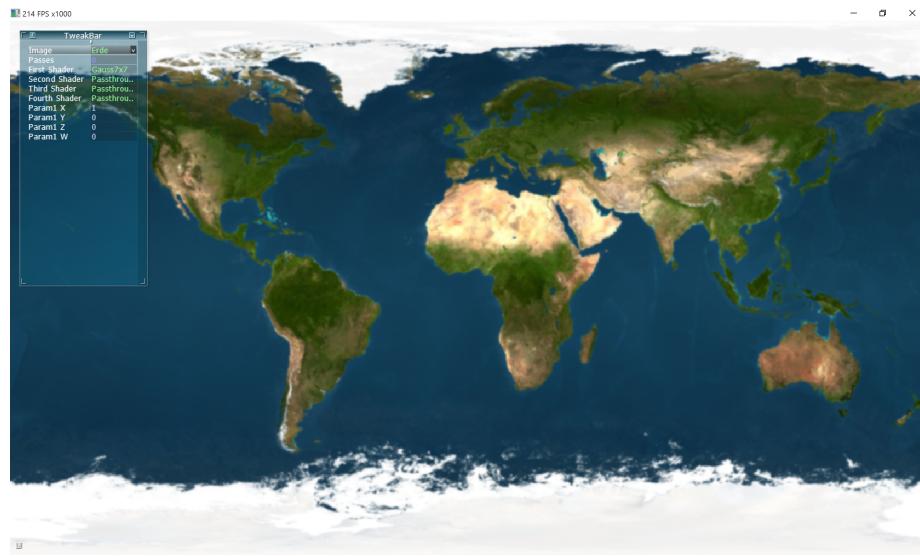
### 3. Gauss5x5

Um einen Gauss-Tiefpassfilter mit einer Größe von 5x5 Pixeln zu implementieren, wurde schlichtweg der Code des 3x3-Filters kopiert und die Anzahl an Schleifendurchläufen zur Berechnung des neuen Pixelwertes auf  $5 \times 5 = 25$  erhöht.



## 4. Gauss7x7

Wie bereits beim 5x5 Gauss-Filter wurde auch hier einfach der Code des 3x3-Filters kopiert und die Anzahl der Schleifendurchläufe in diesem Fall auf 49 erhöht.



### Zeiteinsparung durch Aufteilung in Vertikal und Horizontal

**Grundmessung 7x7 mit 2 Passes ca. 103 FPS**

**Messung mit 2 Passes nach Aufteilung in vertikalen und horizontalen Gaussfilter** Im Shader Gauss5x5 wurde die offsets-Matrix in einen horizontalen Vektor mit der Höhe 7 angepasst, in Gauss7x7 wurde ein einzeiliger Vektor mit der Breite 7 erstellt. Beide Shader in Kombination erzielen nun das gleiche Ergebnis wie die ursprüngliche Implementierung des Gauss7x7, allerdings wesentlich schneller.

Ergebniss der Messung: ca 440 FPS Dies entspricht einer Performance-Steigerung von ca. 338%.

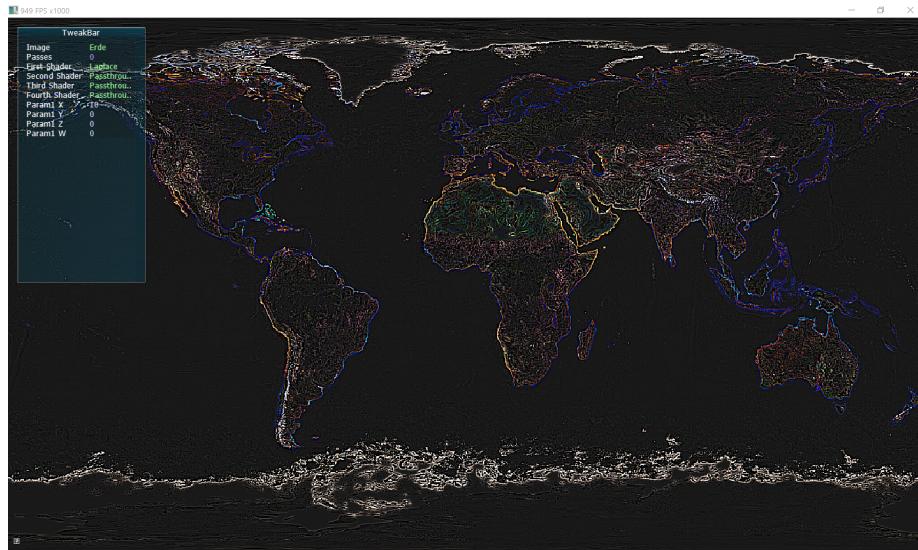
Die Einsparung kommt durch die wesentlich geringere Anzahl an Operationen je Pixel. Während beim regulären 7x7 Gaussfilter auch diagonale Werte für jeden Pixel mit in die Berechnung einbezogen werden, spart man sich dies bei der Aufteilung in eine vertikale und eine horizontale Berechnung. So spart man sich je Pixel knapp 35 Operationen.

## 5. Laplace

Der Laplace-Filter funktioniert, indem von allen benachbarten Pixeln im 3x3-Bereich vom 8-fachen des aktuellen Pixel-Vektors abgezogen. Der Kern für die Laplace-Filterung sieht also wie folgt aus:

```
const int weight[9] = { -1, -1, -1,
                      -1,  8, -1,
                      -1, -1, -1};
```

Anschließend wird auch hier wieder `clamp` verwendet um den Pixel-Vektor im gültigen Wertebereich zu halten.

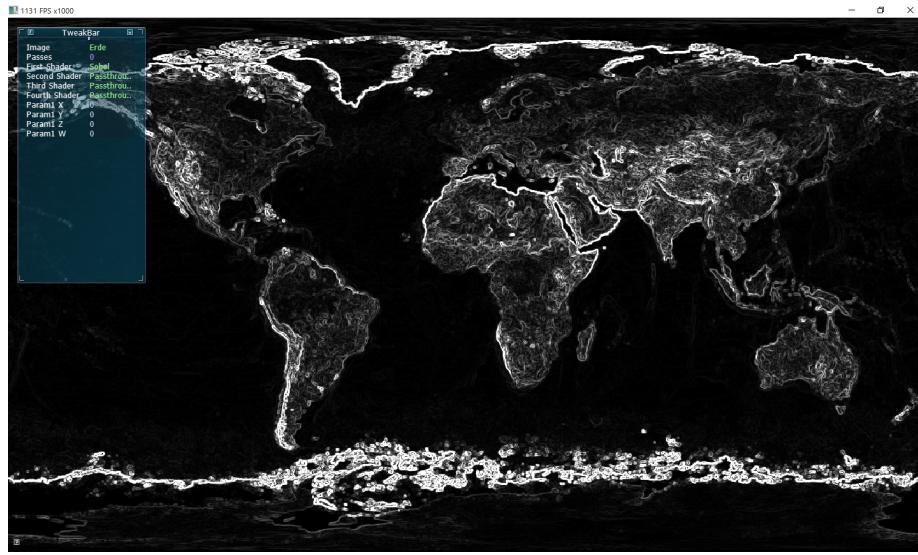


In diesem Filter ist auch der Brightness & Kontrast Filter integriert, womit man über die Parameter X und Y die Helligkeit bzw. den Kontrast verändern kann.

## 6. Sobel

Der Soebelbetragsooperator ist eine weitere Möglichkeit zur Kantendetektion neben dem Laplace-Filter. Für diesen werden zwei Filterkerne verwendet, mit denen man zwei Bilder als Ergebnis erhält. Das eine Bild zeigt die Kanten in vertikaler und das andere in horizontaler Richtung. Um nun die Ausprägung der Kante an einer bestimmten Position zu erhalten werden die beiden Bilder wie folgt miteinander kombiniert:

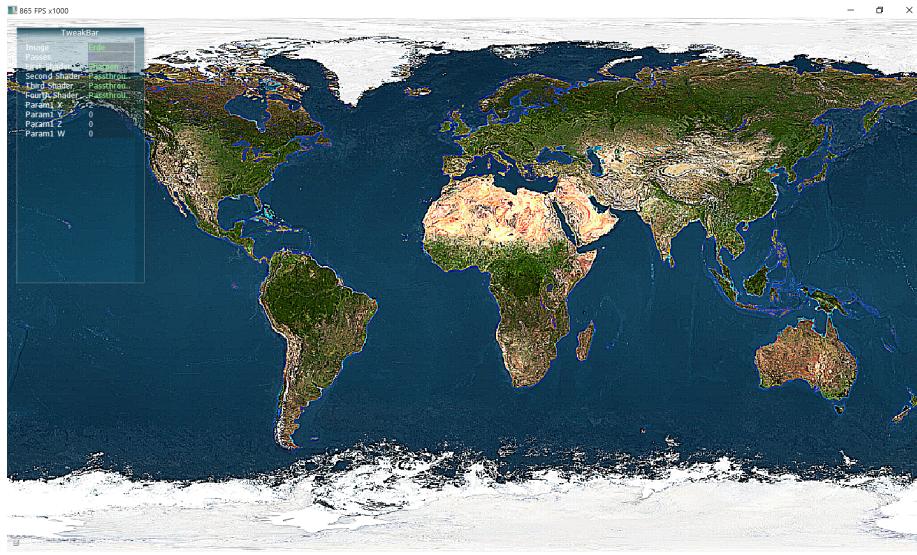
```
sqrt(pow(bild1.rgb, 2) + pow(bild2.rgb, 2))
```



Über den Parameter X kann der Schwellwert eingestellt werden, bei dem eine Kante als Kante detektiert wird. Je höher der Schwellwert ist, desto größer muss die Änderung zur Detektion einer Kante sein.

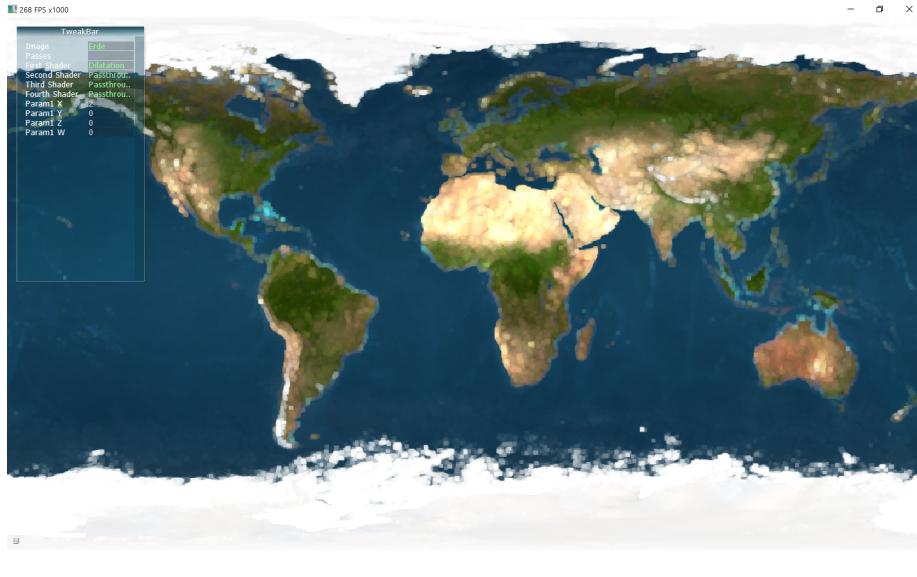
## 7. Sharpen

Zum schärfen eines Bildes wird auf dem ursprünglichen Bild ein Hochpassfilter angewendet und anschließend mit dem Original addiert. Um die Schärfung über einen Parameter zu steuern, wird dieser Hochpassfilter mit dem Parameter multipliziert. Der Hochpassfilter verwendet den gleichen Filterkern wie der Laplace-Filter.



## 8. Dilatation

Bei der Dilatation sollen helle Bereiche verstärkt werden. Dafür wird der aktuelle Pixel auf den maximalen Farbwert aus der Umgebung des aktuellen Pixels gesetzt.



Mit dem Parameter X kann die Größe der Umgebung angepasst werden. Je größer die Umgebung, desto größer dehnen sich die Bereiche aus. Dieser Filter könnte auch in horizontale und vertikale Anteile aufgeteilt werden, jedoch würde dies das Ergebnis negativ beeinflussen.

## 9. Erosion

Im Gegensatz zur Dilatation werden bei der Erosion die dunklen Bereiche verstärkt. Um dies zu erreichen wird hier, anstatt des Maximums, das Minimum aus der betrachteten Umgebung verwenden. Dieser Filter könnte auch in horizontale und vertikale Anteile aufgeteilt werden, jedoch würde dies das Ergebnis negativ beeinflussen.

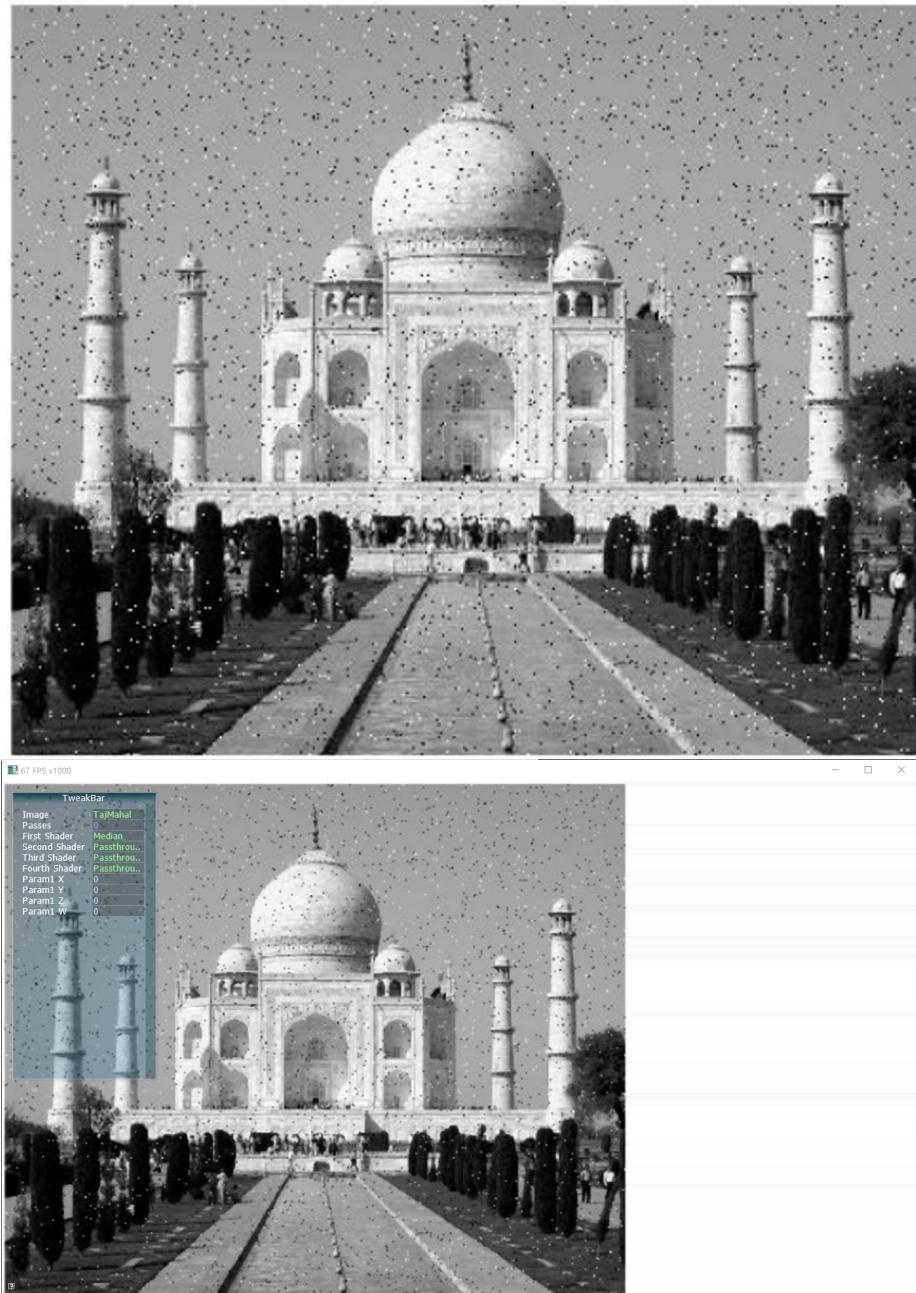


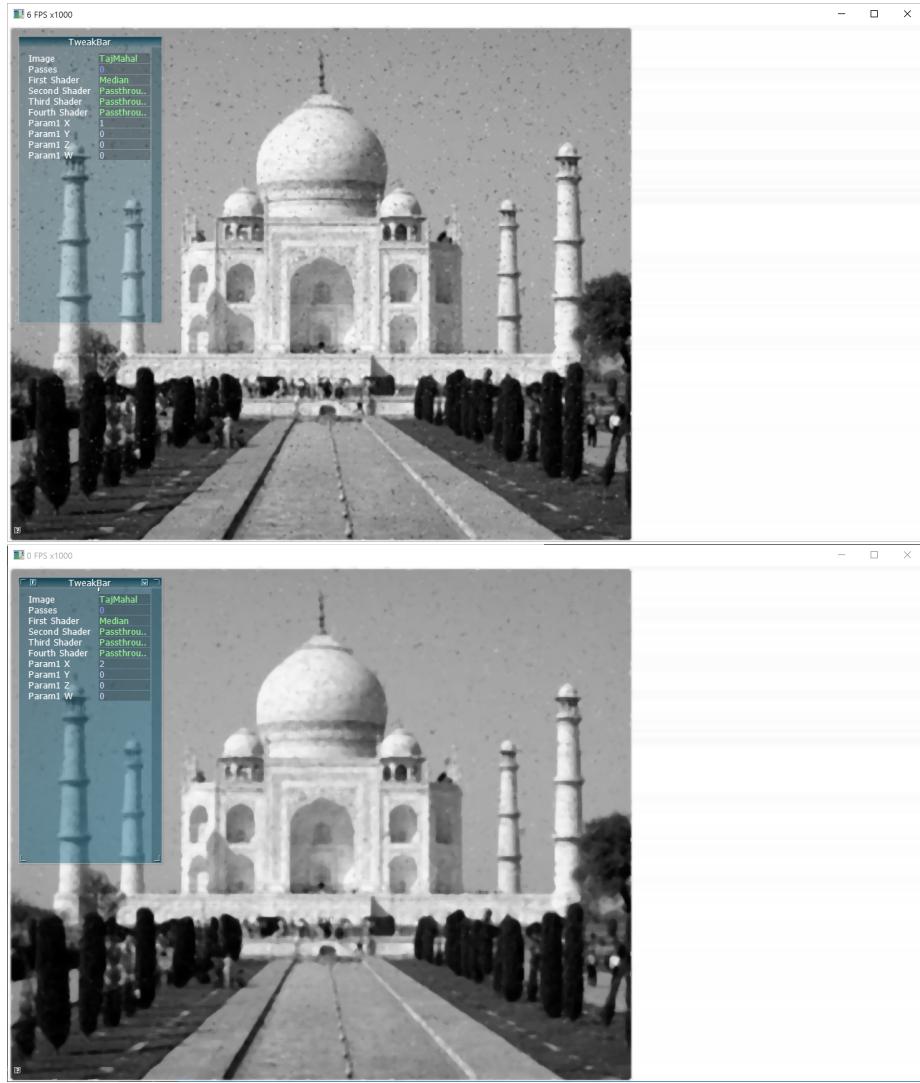
Mit dem Parameter X kann die Größe der Umgebung angepasst werden. Je größer die Umgebung, desto größer dehnen sich die Bereiche aus.

## Aufgabe 1.3

### 1. Störungen entfernen

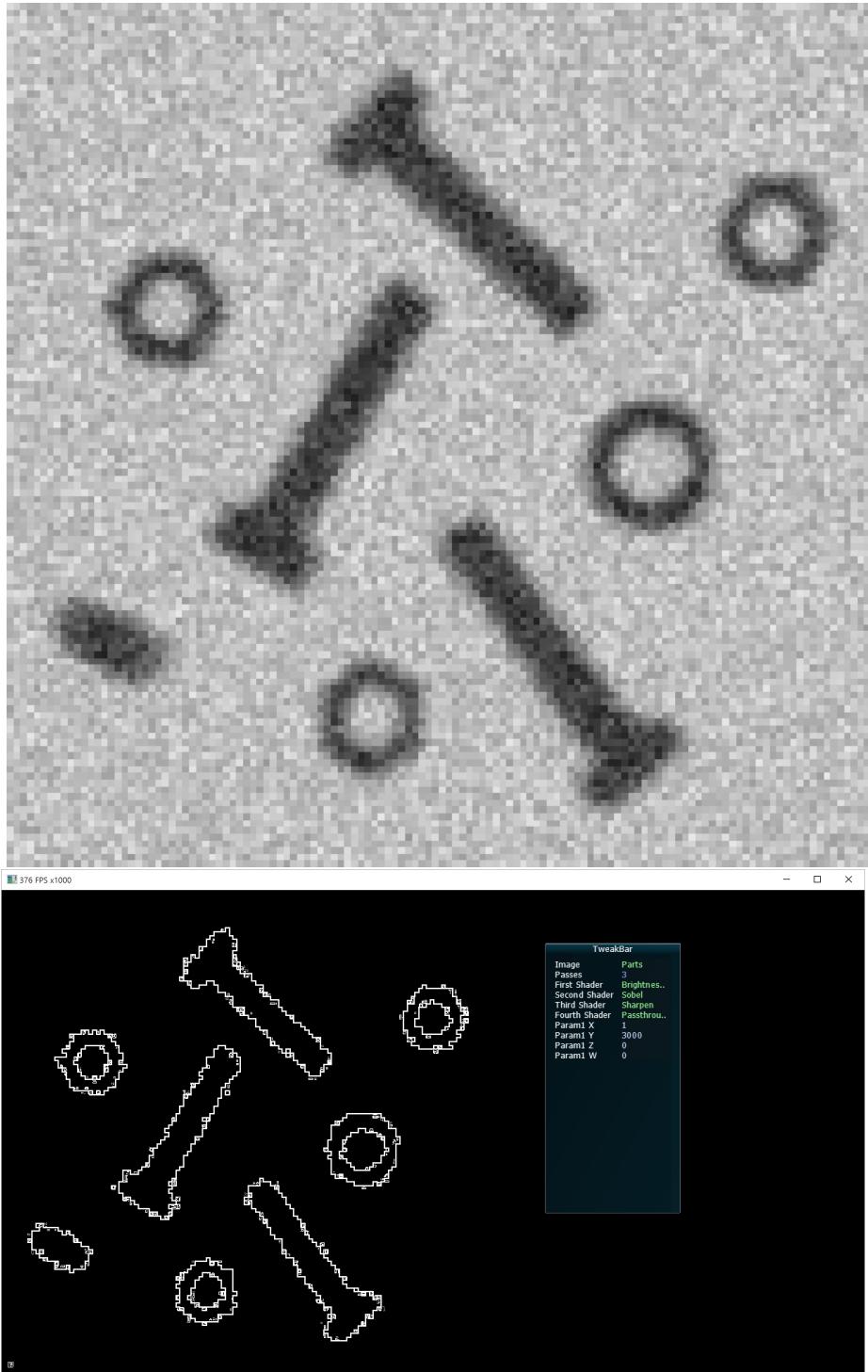
Um die Störungen aus dem Bild zu entfernen ist es nötig einen Median-Filter zu verwenden, da die Störungen gleichermaßen aus weißen und schwarzen Pixeln bestehen. Hier würde Dilatation und Erosion nicht helfen. Bei der Implementierung des Median-Filters kann über den Parameter x angegeben werden welche Größe dieser haben soll. 0 entspricht 3x3, 1 entspricht 5x5 und 2 entspricht 7x7. Dabei sollte unbedingt darauf geachtet werden, dass der 7x7 Median-Filter extrem Ressourcen verbraucht und sich das Programm eventuell aufhängt.





## 2. Kanten extrahieren

Um die Kanten zu extrahieren wird der Sobel-Filter verwendet. Um ein besseres Ergebnis zu erzielen wurde zunächst das Rauschen durch starke Erhöhung des Kontrastes (3000) so gut wie möglich entfernt. Abschließend wurde noch einmal der Sharpen-Filter angewendet, um die Kanten weicher zu machen.



### **3. Binärbild erstellen**

Zur Erstellung eines Binärbild wurde der Filter Binary erstellt, der anhand des in Parameter Z angegebenen Schwellwerts (zwischen 0 und 255) den jeweiligen Pixel auf 0 oder 1 setzt. Zur Vorbehandlung des Ursprungsbildes wurde zuvor noch zwei Erosionen und eine Dilatation durchgeführt um bessere Abgrenzungen zu erzeugen. Um anschließend die Reiskörner zählen zu können kann, durch die vorverarbeitung in ein Binärbild component labeling angewendet werden. Dabei wird die Komponente definiert als Menge aller miteinander verbundenen 1-Pixel (oder weißen Pixeln). Anschließend wird jeder gefundenen Komponente eine Nummer zugeordnet, die höchste dieser Nummern ist dann die Anzahl an Reiskörnern im gegebenen Bild.

