Praktikum 1

Lisa Obermaier, Simon Thum

1.1 Passthrough vs. Bewegter Mittelwert

Der Passthrough-Filter gibt lediglich den aktuellen Pixel wieder, er verursacht also keine Veränderung:

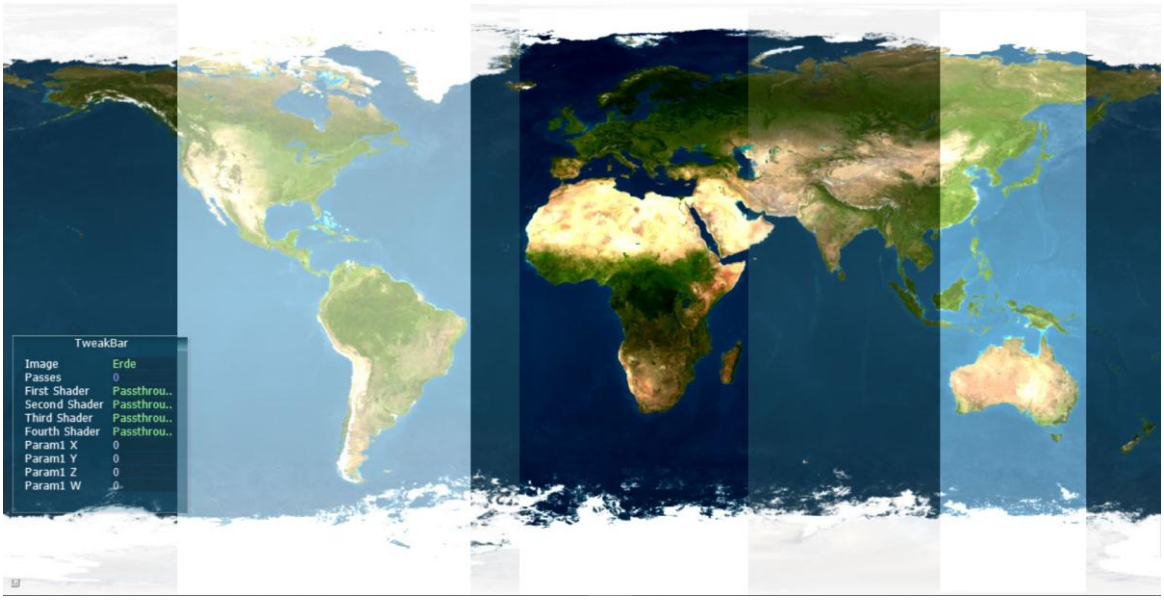
```
19  void main()
20  {
21
22      fragColor = texture(textureMap, texCoords);
23
24  }
```

Der Bewegte-Mittelwert-Filter dagegen durchläuft ein neun-elementiges Array, addiert die Farbwerte und gibt dem aktuellen Pixel den berechneten Mittelwert aller neun Pixel. Wie auf dem Bild oben zu sehen, ist das Bild folglich verschwommen.

```
30  void main()
31  {
32      vec4 texel = vec4(0.0, 0.0, 0.0, 1.0);
33
34      for (int i = 0; i < 9; i++)
35      {
36          texel += texture(textureMap, texCoords + offsets[i]);
37      }
38
39  //    1  1  1
40  //    1  1  1
41  //    1  1  1
42
43      fragColor =     texel/9.0 ;
44
45  }
```

1.2 Weitere Filter

1) Helligkeit und Kontrast



Um Brightness und Color zu modifizieren, müssen zwei Parameter veränderbar sein. param1[0] ist für die Helligkeit und param1[1] für den Kontrast verantwortlich. Zunächst wird die Darstellung des Pixels verschoben, anschließend der Kontrast multipliziert, dann die Helligkeit addiert und zuletzt die anfängliche Verschiebung wieder zurückgesetzt. Um den richtigen Alpha-Wert zu setzen, wird dieser vor den Operationen zwischengespeichert.
Im oben abgebildeten Bild ist Amerika mit modifizierter Helligkeit, Europa und Afrika mit modifiziertem Kontrast und Australien mit kombinierter Modifikation abgebildet.

```
30  void main()
31  {
32
33      vec4 texel = texture(textureMap, texCoords);
34
35      float alpha = texel[3];
36
37      texel = texel - 0.5;
38      texel = texel * ((param1[1] / 100) + 1.0);
39      texel = texel + (param1[0] / 100);
40      texel = texel + 0.5;
41      texel[3] = alpha;
42
43      fragColor = texel;
44
45  }
```

2) Gauß-Tiefpass-Filter mit 3x3-Fenster

Um den Gauß-Tiefpass-Filter zu realisieren, verwenden wir eine entsprechende Formel für den Filter, die mit x und y aus dem Vektor und einem modifizierbaren Sigma (in unserem Beispiel param1[0]) ein h berechnet, das mit dem aktuell betrachteten Pixel im texel für alle neun Iterationen addiert wird. Anschließend wird der texel-Wert durch die Summe der berechneten h's geteilt, um einen Mittelwert in Abhängigkeit des eingestellten Sigmas zu erhalten.

```
33      vec4 texel;
34      float pi = 3.14159265359;
35      float h;
36      float counter = 0.0 ;
37              float p1 = param1[0] / 100;
38
39
40      if (p1 != 0) {
41          for (int i = 0; i < 9; i++) {
42              float x = offsets[i][0];
43              float y = offsets[i][1];
44
45              h  = (1 / (2 * pi * p1 * p1)) * exp(-(x * x + y * y) / (2 * p1 * p1));
46
47              texel +=  texture(textureMap, texCoords + offsets[i]) * h;
48              counter += h;
49          }
50
51          fragColor = texel / counter;
52
53      } else
54          {
55          fragColor = texture(textureMap, texCoords);
56          }
```

3) Gauß-Tiefpass-Filter mit 5x5-Fenster

Der Gauß-Tiefpass-Filter mit einem 5x5-Fenster funktioniert äquivalent zu einem 3x3-Fenster, doch mit einem größeren Fenster wird die Veränderung im Bild deutlicher.

```
38      vec4 texel;
39      float pi = 3.14159265359;
40      float h;
41      float counter = 0.0 ;
42              float p1 = param1[0] / 100;
43
44
45      if (p1 != 0) {
46          for (int i = 0; i < 25; i++) {
47              float x = offsets[i][0];
48              float y = offsets[i][1];
49
50              h  = (1 / (2 * pi * p1 * p1)) * exp(-(x * x + y * y) / (2 * p1 * p1));
51
52              texel +=  texture(textureMap, texCoords + offsets[i]) * h;
53              counter += h;
54          }
55
56          fragColor = texel / counter;
57
58      } else
59          {
60          fragColor = texture(textureMap, texCoords);
61          }
```

4) Gauß-Tiefpass-Filter mit 7x7-Fenster

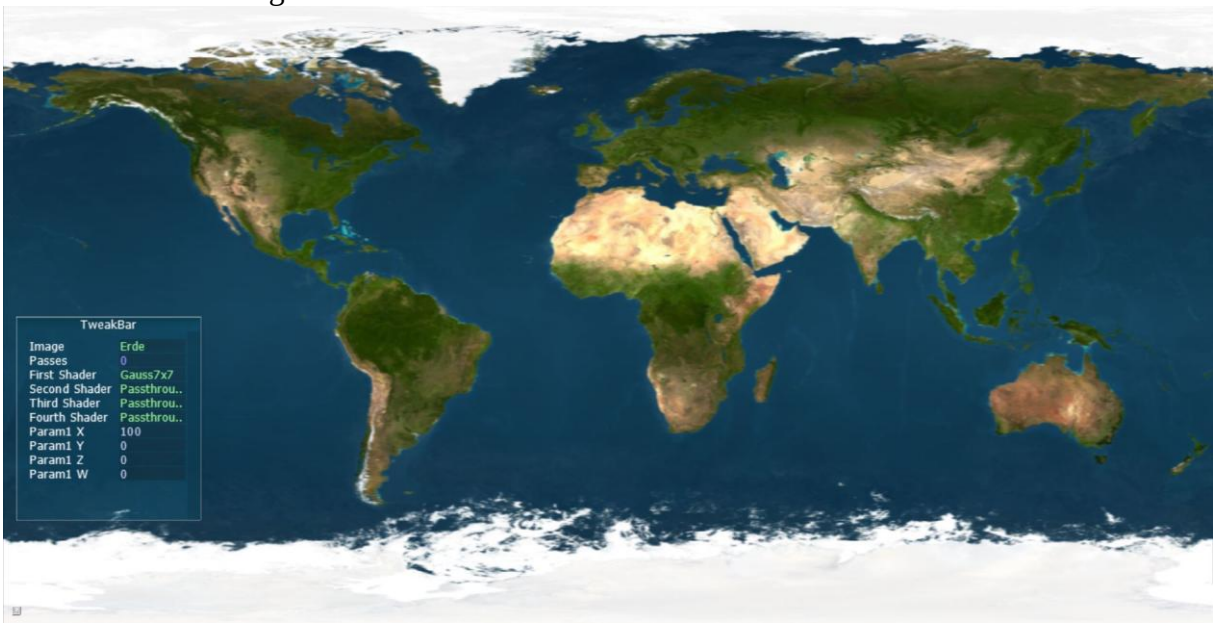
Mit einem 7x7-Fenster im Gauß-Tiefpass-Filter ist das Verschwimmen des Bildes nun gut wahrzunehmen. Ansonsten funktioniert er äquivalent zum Gauß-Tiefpass-Filter mit 3x3 oder 5x5-Fenster. Allerdings macht sich hier die Performance schon deutlich bemerkbar, denn nur die 49 Iterationen in der Schleife, werden nur noch etwa 100 FPS erreicht.

```
39      vec4 texel;
40      float pi = 3.14159265359;
41      float h;
42      float counter = 0.0 ;
43      float p1 = param1[0] / 100;
44
45
46      if (p1 != 0) {
47          for (int i = 0; i < 49; i++) {
48              float x = offsets[i][0];
49              float y = offsets[i][1];
50
51              h  = (1 / (2 * pi * p1 * p1)) * exp(-(x * x + y * y) / (2 * p1 * p1));
52
53              texel +=  texture(textureMap, texCoords + offsets[i]) * h;
54              counter += h;
55          }
56
57          fragColor = texel / counter;
58
59      } else
60          {
61          fragColor = texture(textureMap, texCoords);
62          }
```

Gauß Tiefpass-Filter realisiert mit zwei Shadern:

Um die Performance zu verbessern, sollen den oben beschriebenen Gauß-Tifpass-Filter mit 7x7-Fenster zwei Shader übernehmen, die jeweils nur über die X- oder die Y-Koordinate iterieren und so weniger als 49 Iterationen für die Darstellung der Pixel benötigen.

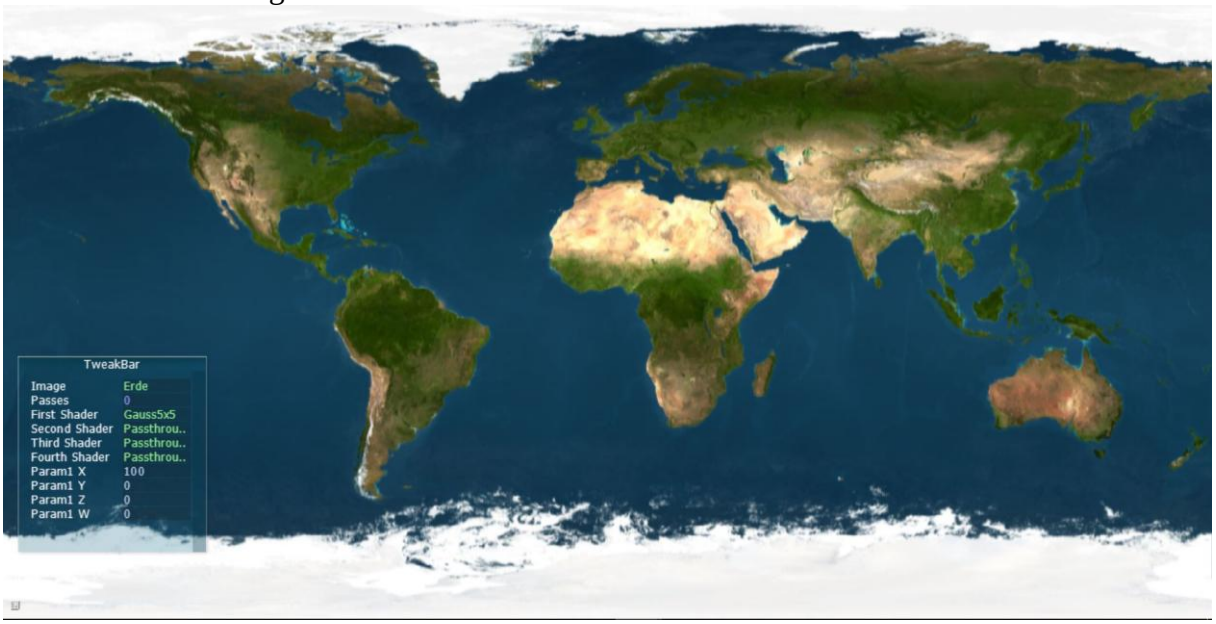
Shader in X-Richtung:



Mit einer Formel für den Gauß-Tiefpass-Filter, die lediglich eine Koordinate (in diesem Fall x) mit einem gegebenen Sigma berechnet, wird hier, analog zum 7x7-Gauß-Tiefpass-Filter von oben, das aktuelle Pixel mit dem errechneten h multipliziert, alles addiert und im Anschluss durch die Summe der errechneten h's geteilt. So kann ein, der modifizierbaren Gauß-Glocke entsprechender, Farbton über das 7x7-Fenster angenommen werden. Im Bild oben ist erkennbar, dass die Pixel horizontal verwischen.

```
65      vec4 texel;
66      float pi = 3.14159265359;
67      float h;
68      float counter = 0.0 ;
69      float p1 = param1[0] / 100;
70
71      if (p1 != 0) {
72
73          for (int i = 0; i < 7; i++) {
74                  float x = offsets[i][0];
75
76                  h  = (1 / (p1 * sqrt(2 * pi))) * exp(-(x * x) / (2 * p1 * p1));
77
78                  texel +=  texture(textureMap, texCoords + offsets[i]) * h;
79                  counter += h;
80              }
81
82              fragColor = texel / counter;
83
84
85      } else {
86          fragColor = texture(textureMap, texCoords);
87      }
```

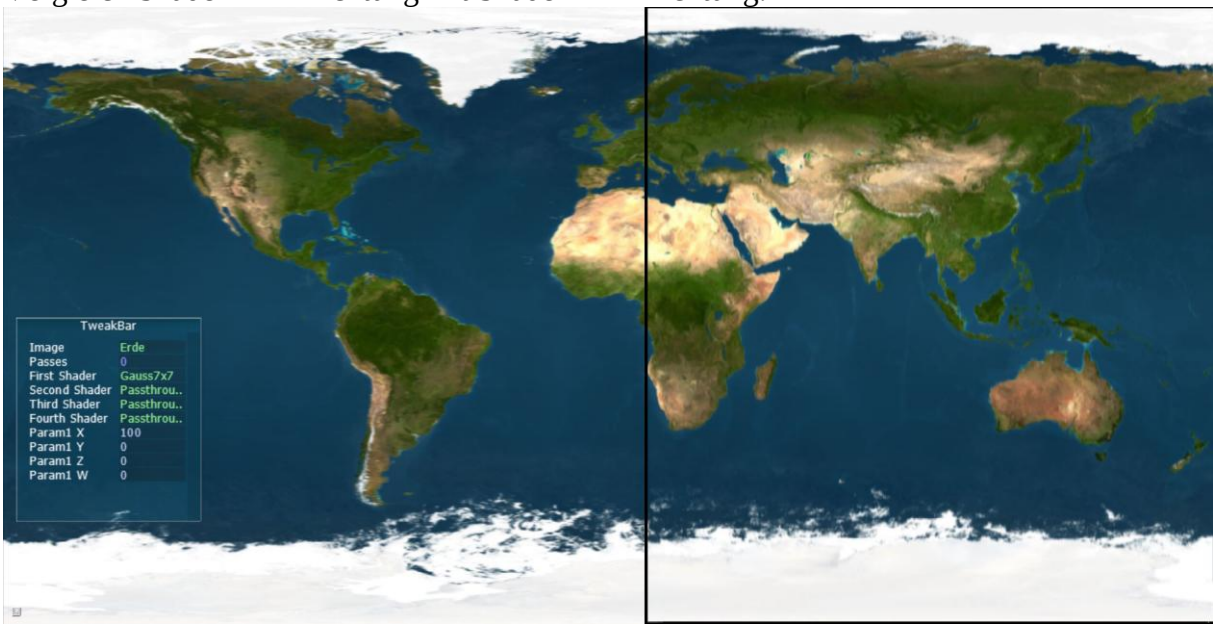
Shader in Y-Richtung:



Analog zur X-Richtung oben, verwischen die Pixel in diesem Bild horizontal. Die Berechnung bleibt gleich, mit dem Unterschied, dass die y-Koordinate zur Berechnung verwendet wird.

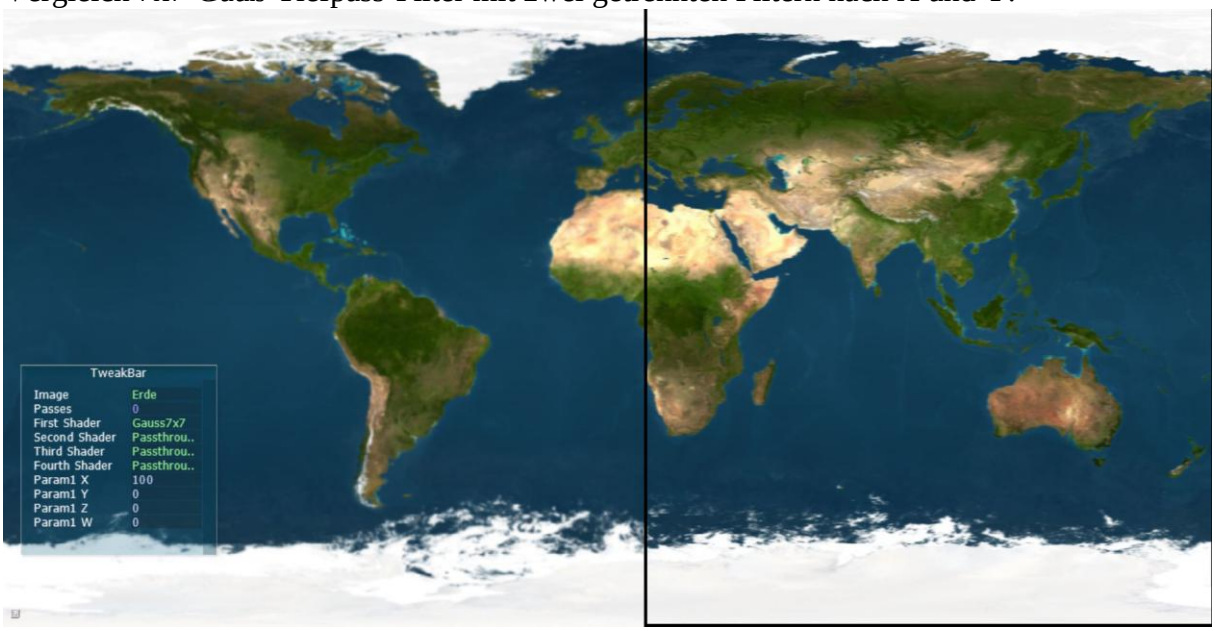
```
64          vec4 texel;
65          float pi = 3.14159265359;
66          float h;
67          float counter = 0.0 ;
68          float p1 = param1[0] / 100;
69
70          if (p1 != 0) {
71
72          for (int i = 0; i < 7; i++) {
73                  float y = offsets[i][1];
74
75                  h  = (1 / (p1 * sqrt(2 * pi))) * exp(-(y * y) / (2 * p1 * p1));
76
77                  texel +=  texture(textureMap, texCoords + offsets[i]) * h;
78                  counter += h;
79              }
80
81              fragColor = texel / counter;
82
83
84      } else {
85          fragColor = texture(textureMap, texCoords);
86      }
87  }
```

Vergleich Shader in X-Richtung mit Shader in Y-Richtung:



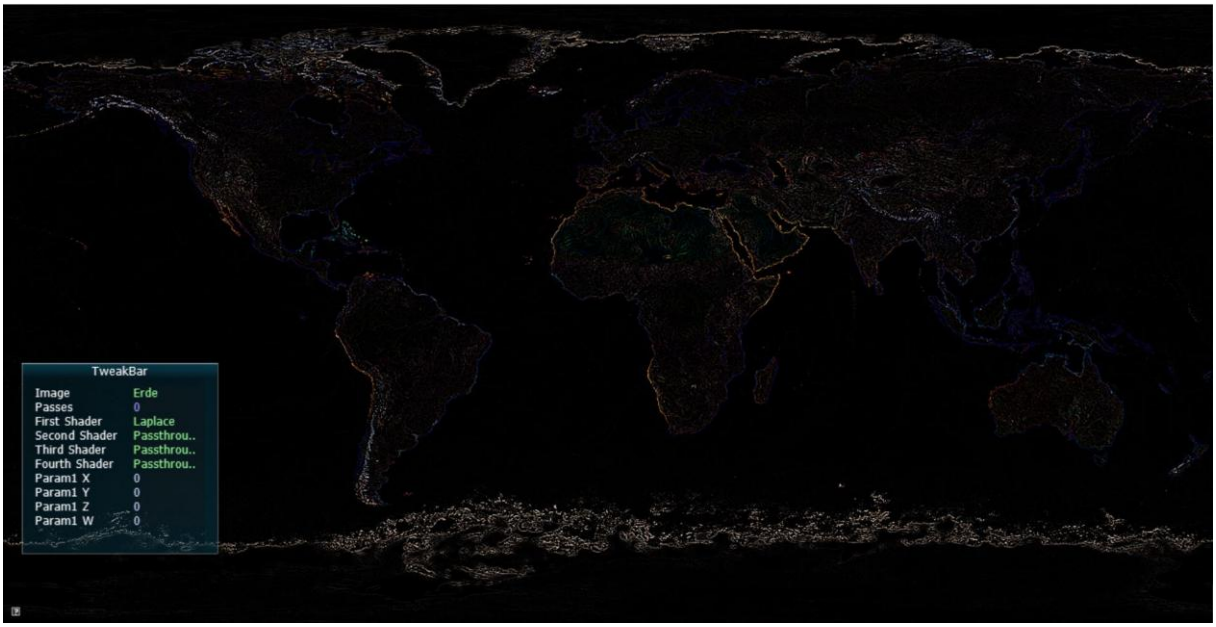
In diesem Bild oben sieht man nun den Unterschied zwischen der Verwischung auf der horizontalen und der Verwischung auf der vertikalen Linie.

Vergleich 7x7-Gauß-Tiefpass-Filter mit zwei getrennten Filtern nach X und Y:



Obiges Bild zeigt nun einen Vergleich der Filter über 49 Iterationen und zweimal über 7 Iterationen. Es ist kaum ein Unterschied erkennbar und mit Hilfe der zwei überlagernden Filter sind wieder FPS von 250 bis sogar 300 möglich, also eine Verbesserung um bis zu 200%.

5) Laplace



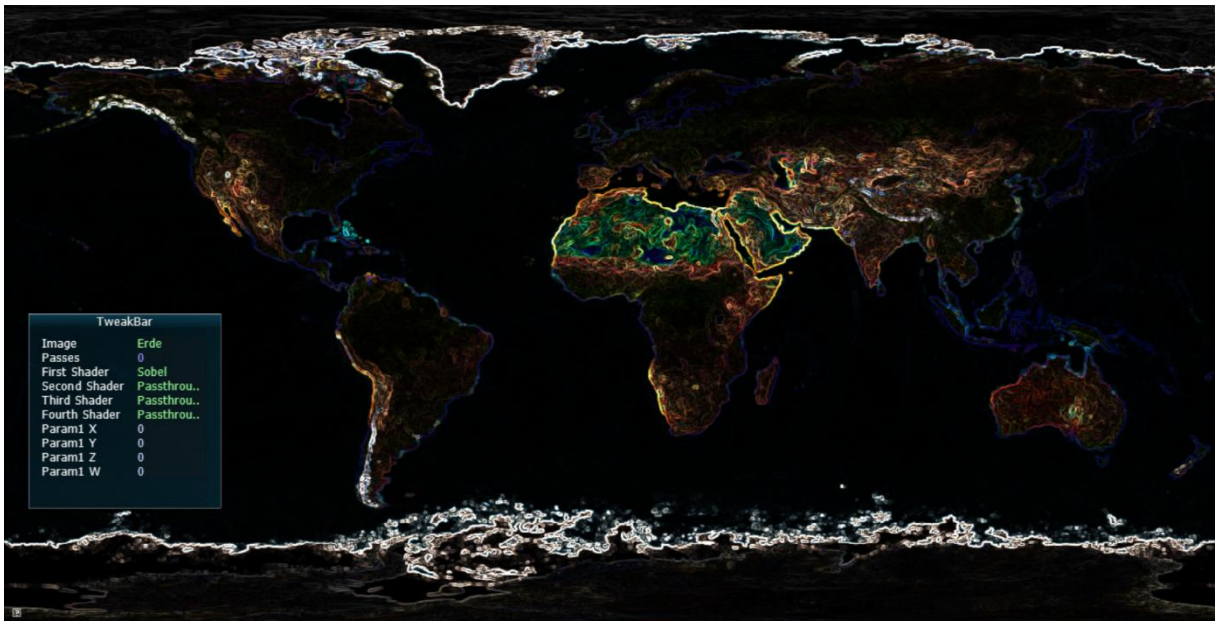
Mit Hilfe des zweidimensionalen Laplace-Filters können über eine Addition der texel die Kanten extrahiert werden.
Der Vektor sieht folgendermaßen aus:

0	-1	0
-1	4	-1
0	-1	0

und entsprechend werden die Pixel im Umfeld multipliziert. Das aktuelle texel wird mit 4 Multipliziert und die umliegenden texel oben, rechts, unten und links mit (-1). Diese Werte aufaddiert ergeben den neuen Farbwert des aktuellen Pixels.
Um Helligkeit und Kontrast anpassen zu können, muss die Veränderung der Paramterer param1[0] und param1[1] wie für den FragmenShaderBrightness_Contrast.glsl, abgefangen und entsprechen berechnet werden.

```
30  void main()
31  {
32
33          vec4 texel = texture(textureMap, texCoords);
34
35          texel = texel * 4 + texture(textureMap, texCoords + offsets[5]) * -1;
36          texel = texel + texture(textureMap, texCoords + offsets[3]) * -1;
37          texel = texel + texture(textureMap, texCoords + offsets[1]) * -1;
38          texel = texel + texture(textureMap, texCoords + offsets[7]) * -1;
39
40      texel += -0.5;
41      texel = texel * ((param1[1] / 100) + 1);
42      texel += (param1[0] / 100) + 0.5;
43      texel[3] = 1.0; //alpha
44
45
46
47      fragColor = texel;
```

6) Sobel



Der Sobel-Filter kann, ähnlich wie der Laplace-Filter, mit einer Matrix-Multiplikation gelöst werden. Die Berechnung ist folgende:

- $G(x) = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix} * A$

- $G(y) = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} * A$

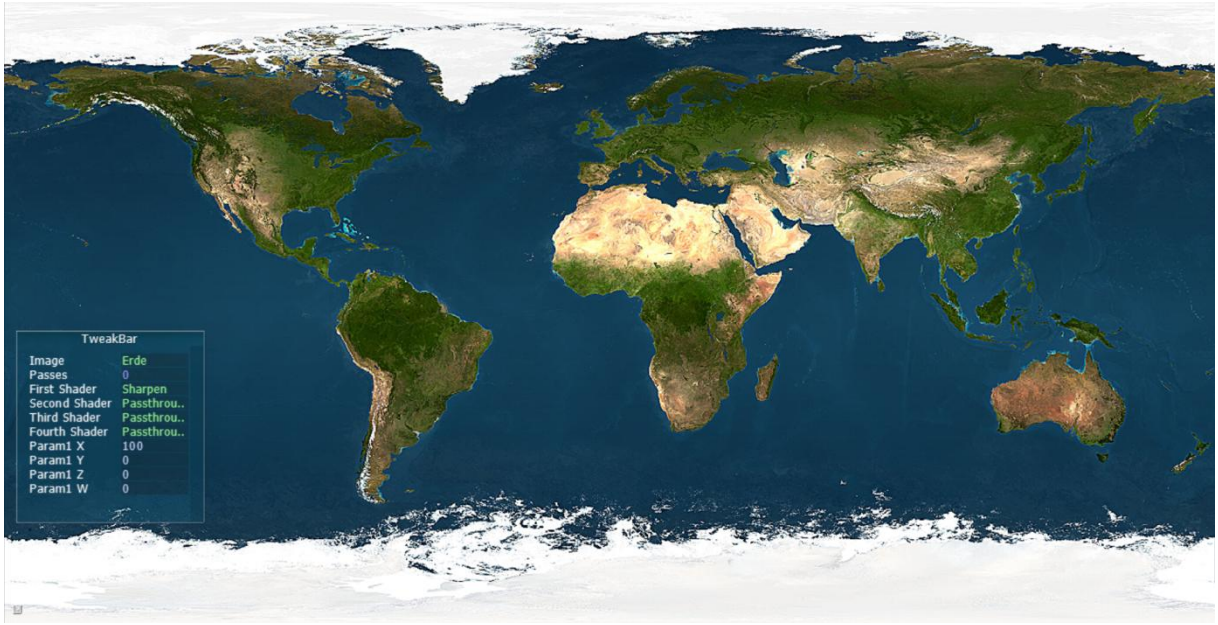
  wobei A immer das aktuelle Pixel darstellt.

- $G = \mathrm{sqrt}(G(x)^2 + G(y)^2)$

Wie beim Laplace-Filter werden die Helligkeit und der Kontrast über die verfügbaren Parameter berechnet.

```
33          vec4 texel = texture(textureMap, texCoords);
34          float alpha = texel[3];
35          vec4 x;
36          vec4 y;
37
38          x +=texture(textureMap, texCoords + offsets[2]) * 1;
39          x +=texture(textureMap, texCoords + offsets[1]) * 2;
40          x +=texture(textureMap, texCoords + offsets[0]) * 1;
41          x +=texture(textureMap, texCoords + offsets[8]) * -1;
42          x +=texture(textureMap, texCoords + offsets[7]) * -2;
43          x +=texture(textureMap, texCoords + offsets[6]) * -1;
44
45          x = x * texel;
46
47          y +=texture(textureMap, texCoords + offsets[2]) * 1;
48          y +=texture(textureMap, texCoords + offsets[5]) * 2;
49          y +=texture(textureMap, texCoords + offsets[8]) * 1;
50          y +=texture(textureMap, texCoords + offsets[0]) * -1;
51          y +=texture(textureMap, texCoords + offsets[3]) * -2;
52          y +=texture(textureMap, texCoords + offsets[6]) * -1;
53
54          y = y * texel;
55
56          texel = sqrt(x * x + y * y);
57
58      texel += -0.5;
59      texel = texel * ((param1[1] / 100) +1);
60      texel += (param1[0] / 100) + 0.5;
61      texel[3] = alpha;
62
63          fragColor = texel;
```

7) Sharpener

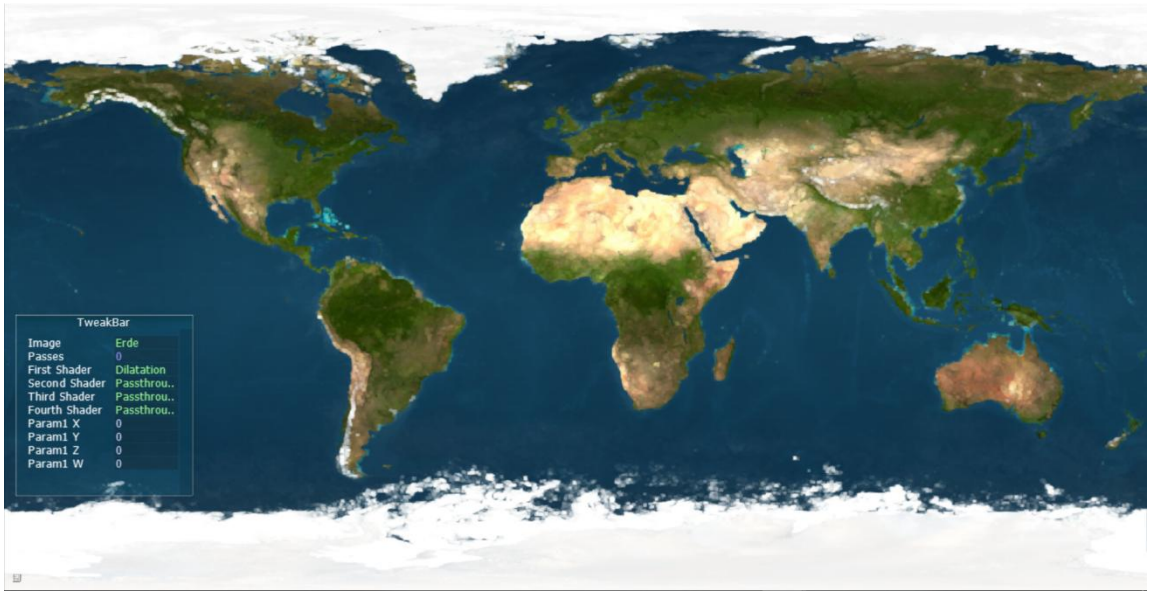


Das Bild wird schärfer, indem das Originalbild mit dem Ergebnis des Laplace-Filters addiert wird und die Stärke der Schärfung kann modifiziert werden durch den Parameter X (param1[0]). Dieser Parameter wird mit dem Laplace-Ergebnis modifiziert und so treten die Kanten bei höherem X stärker hervor.

```
30  void main()
31  {
32
33          vec4 original = texture(textureMap, texCoords);
34          float c = param1[0] / 100;
35
36          vec4 texel = texture(textureMap, texCoords);
37
38          texel = texel * 4 + texture(textureMap, texCoords + offsets[5]) * -1;
39          texel = texel + texture(textureMap, texCoords + offsets[3]) * -1;
40          texel = texel + texture(textureMap, texCoords + offsets[1]) * -1;
41          texel = texel + texture(textureMap, texCoords + offsets[7]) * -1;
42
43          original += texel * c;
44
45          fragColor = original;
```

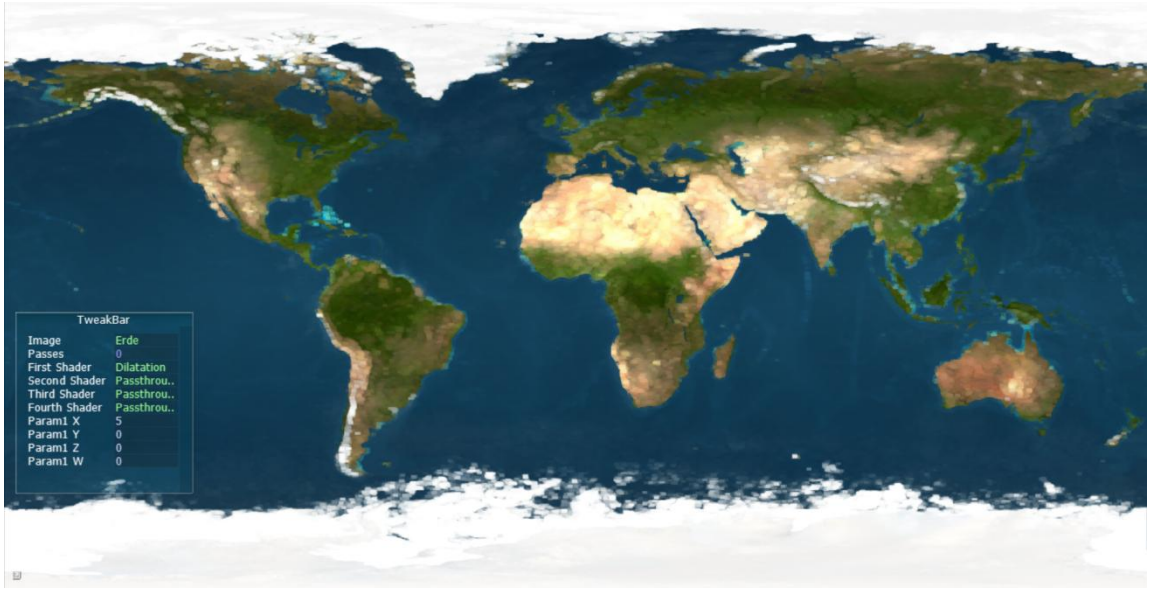
8) Dilatation

3x3-Fenster:



Bei der Dilatation werden bevorzugt hohe/weiße Werte ausgegeben und somit die hellen Teile des Bildes verstärkt dargestellt. Dies wird realisiert, indem im gegebenen Fenster (hier 3x3 Pixel) der höchste Wert ausgewählt und für das aktuelle Pixel gesetzt wird.

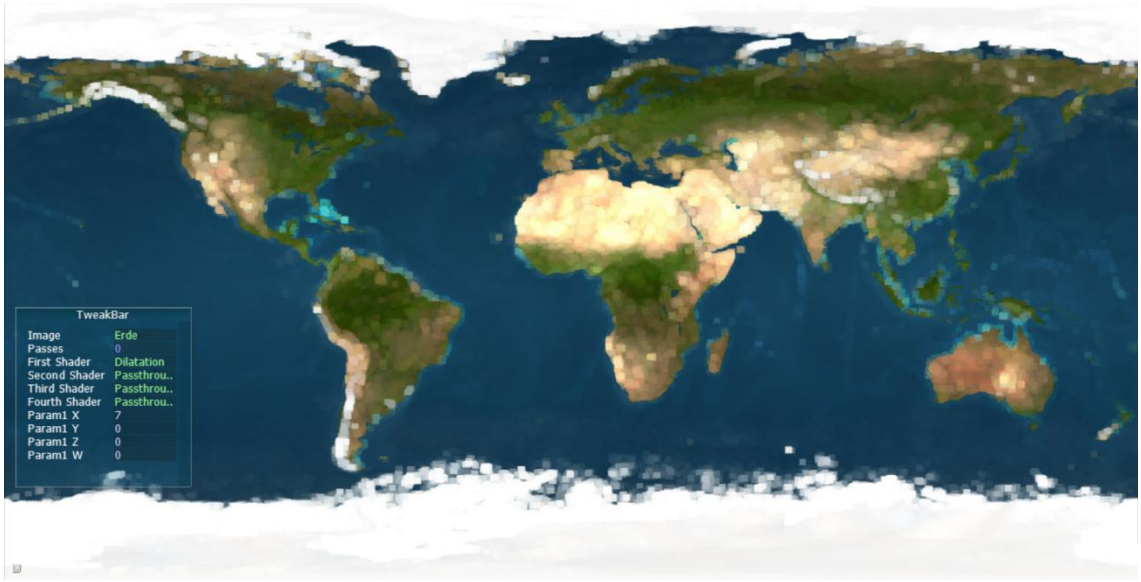
5x5-Fenster:



Das 5x5- und das 7x7-Fenster können ausgewählt werden, indem im X-Parameter 5 bzw. 7 eingegeben werden. Je größer das gewählte Fenster, umso eher treten die hellen Felder in den Vordergrund.

Im Code ist die unterschiedliche Fenstergröße über eine Fallunterscheidung nach param1[0] gelöst, nach der im Anschluss über unterschiedlich große Arrays iteriert wird.

7x7-Fenster:

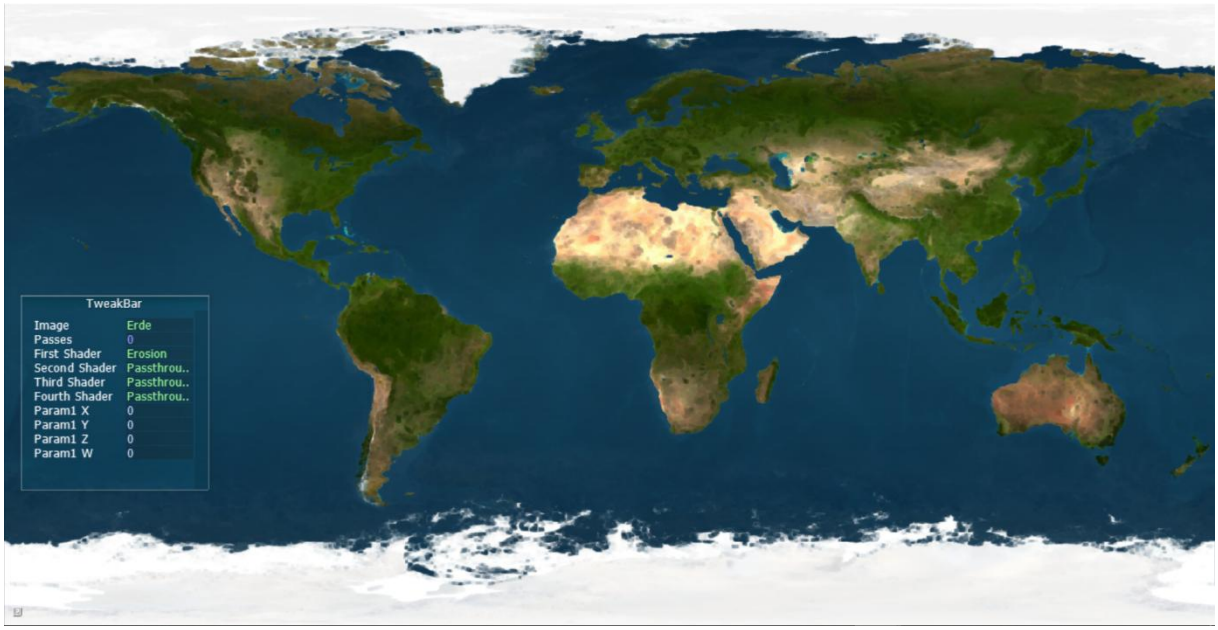


Der Filter kann in einen horizontalen und einen vertikalen Anteil aufgespalten werden, da sowohl horizontal als auch vertikal der höchste Farbwert angenommen wird und bei aufeinanderliegenden Filtern der höchste Wert angenommen werden wird.

```
48      float window = 3.0;
49
50      if (param1[0] == 5) {
51          window = 5.0;
52      }
53      else if (param1[0] == 7)  {
54          window = 7.0;
55      }
56
57      vec4 texel = texture(textureMap, texCoords);
58      vec4 max = texel;
59      vec4 tmp;
60
61      for (int i = 0; i < window*window; i++) {
62
63              if (window == 5.0) {
64                  tmp = texture(textureMap, texCoords + offsetsB[i]);
65              }
66              else if (window == 7.0) {
67                  tmp = texture(textureMap, texCoords + offsetsC[i]);
68              }
69              else {
70                  tmp = texture(textureMap, texCoords + offsets[i]);
71              }
72
73          if (max(tmp, max) == tmp) {
74              max = tmp;
75          }
76      }
77
78
79      fragColor = max;
```

9) Erosion

3x3-Fenster

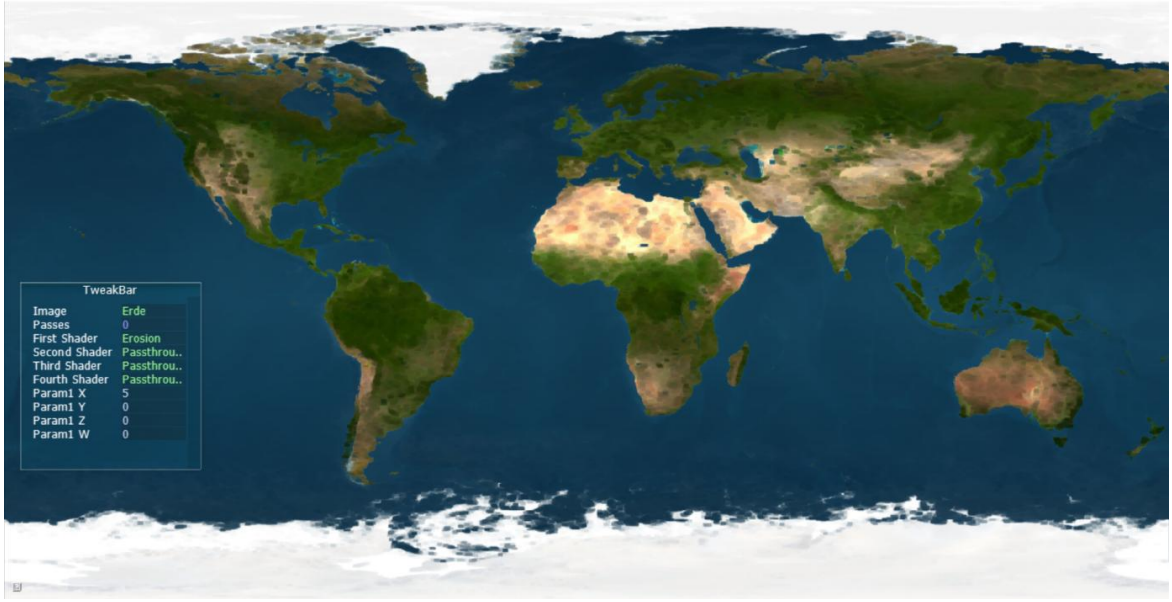


Die Erosion funktioniert analog zur Dilatation, nur dass statt dem höchsten Wert der niedrigste Wert aus dem gewählten Fenster für das aktuelle Pixel angenommen wird. Daher treten dunkle Felder des Bildes mehr in den Vordergrund. Je größer das Fenster wird, umso größer wird der Effekt.

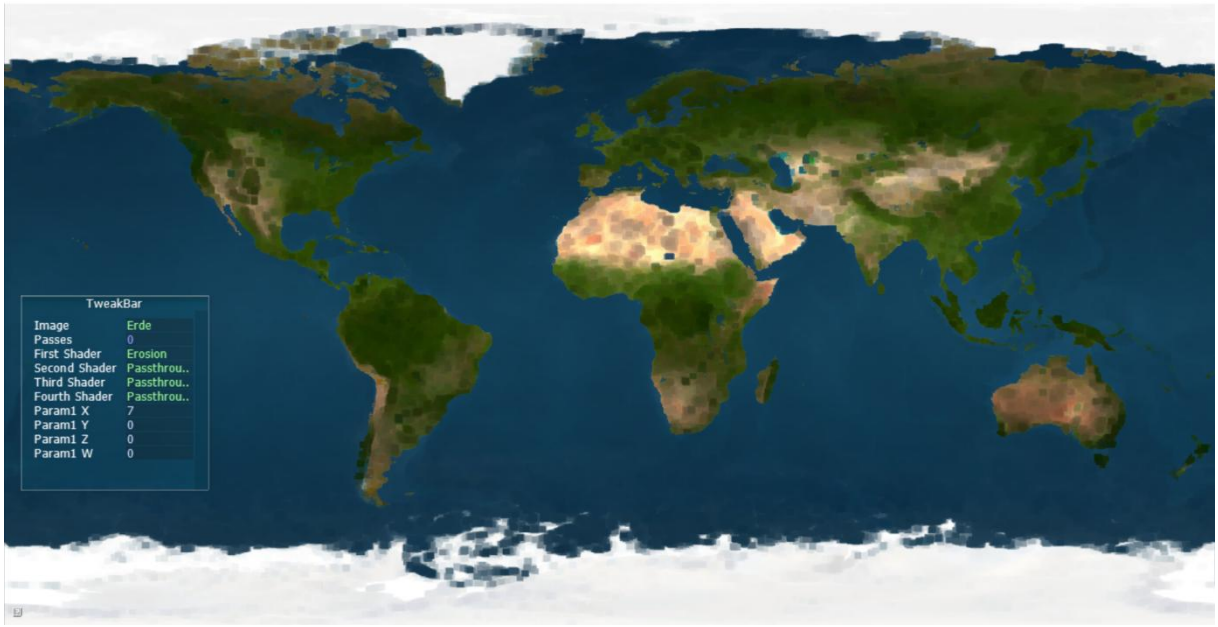
Die Fallunterscheidung wird hier ebenso über den param1[0] realisiert, wie bei der Dilatation.

Kann das 7x7-Fenster auch über zwei jeweils vertikale und horizontale Filter realisiert werden: Das Minimum einer gegebenen Matrix ergibt sich aus den Minima der Zeilen und der Spalten, entsprechend ist es durchaus möglich, die Bestimmung eines Minimums in zwei übereinanderliegende Filter aufzuteilen.

5x5-Fenster:



7x7-Fenster:

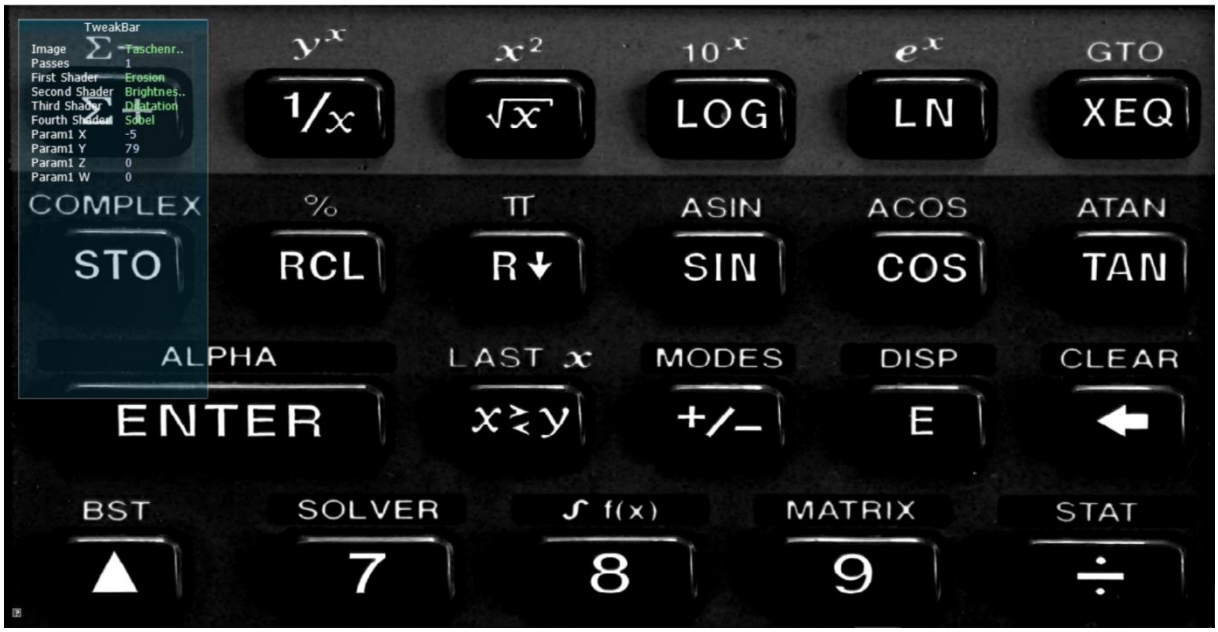



```
45  void main()
46  {
47
48      float window = 3.0;
49
50      if (param1[0] == 5) {
51          window = 5.0;
52      }
53      else if (param1[0] == 7)  {
54          window = 7.0;
55      }
56
57      vec4 texel = texture(textureMap, texCoords);
58      vec4 minimum = texel;
59      vec4 tmp;
60
61      for (int i = 0; i < window*window; i++) {
62
63              if (window == 5.0) {
64                  tmp = texture(textureMap, texCoords + offsetsB[i]);
65              }
66              else if (window == 7.0) {
67                  tmp = texture(textureMap, texCoords + offsetsC[i]);
68              }
69              else {
70                  tmp = texture(textureMap, texCoords + offsets[i]);
71              }
72
73          if (min(tmp, minimum) == tmp) {
74              minimum = tmp;
75          }
76      }
77
78      fragColor = minimum;
```

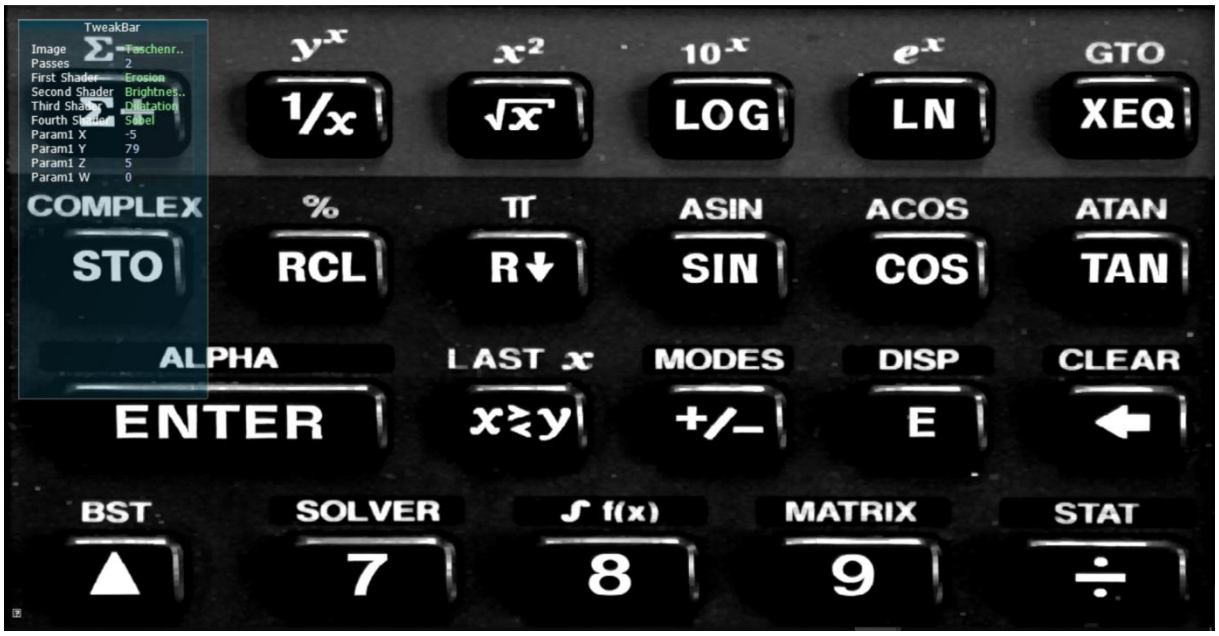
1.3 Taschenrechner

1. Schritt: Erosion 3x3-Fenster
   Zunächst wählen wir eine Erosion im 3x3 großen Fenster, um die dunklen Ränder um die Tasten herum zu betonen. Für diese Aufgabe wurde der Filter minimal angepasst, sodass das Fenster mit dem Parameter Z verstellt werden kann. Der Default liegt allerdings auf dem 3x3-Fenster, deshalb ist hier der Parameter Z = 0.

2. Schritt: Brightness_Contrast mit Brightness -5 und Contrast 79
   Mit dem zweiten Filter erhöhen wir den Kontrast (Y-Wert = 79) und lassen das Bild mehr ins Schwarze gehen, dies wird realisiert durch einen negativen Wert für die Helligkeit (X-Wert = -5)



3. Schritt: Dilatation mit 5x5-Fenster
   Die Dilatation sorgt dafür, dass die weißen Zahlen und Beschriftungen stärker hervortreten. Wir haben ein 5x5-Fenster gewählt, da hier der Effekt schön sichtbar wird, aber die Beschriftungen noch nicht verschwimmen, wie es ein 7x7-Fenster nach sich ziehen würde.



4. Schritt: Sobel-Filter
   Zuletzt sorgt der Sobel-Filter dafür, dass die Ränder der Beschriftungen auf dem Taschenrechner gut zu erkennen sind. Lediglich die Highlights auf den Tasten, die auf dem Originalbild eine ähnlich helle Farbe aufweisen, wie die Beschriftung selbst, können mit dieser Filterkombination nicht entfernt werden.