Praktikum 1

Lisa Obermaier, Simon Thum

1.1 FragmentShaderPassthough.glsl Vs. FragmentShaderBewMit.glsl

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

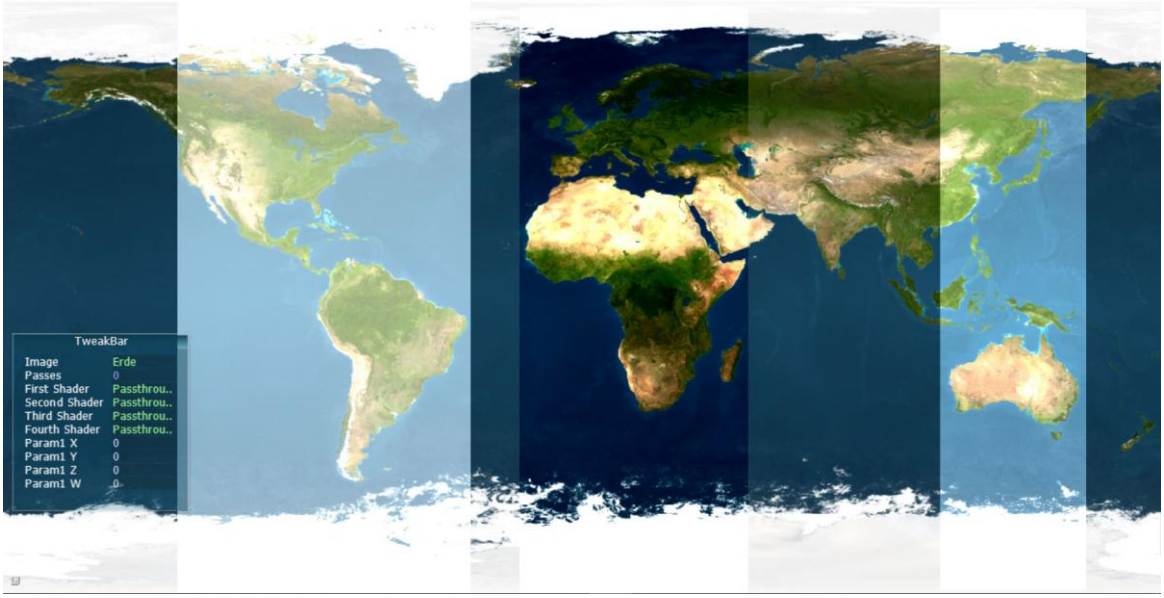
```
19  void main()
20  {
21
22      fragColor = texture(textureMap, texCoords);
23
24  }
```

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) FragmenShaderBrightness_Contrast.glsl



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) FragmentShaderGauß3x3.glsl

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) SegmentShaderGauß5x5.glsl

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) SegmentShaderGauß7x7.glsl

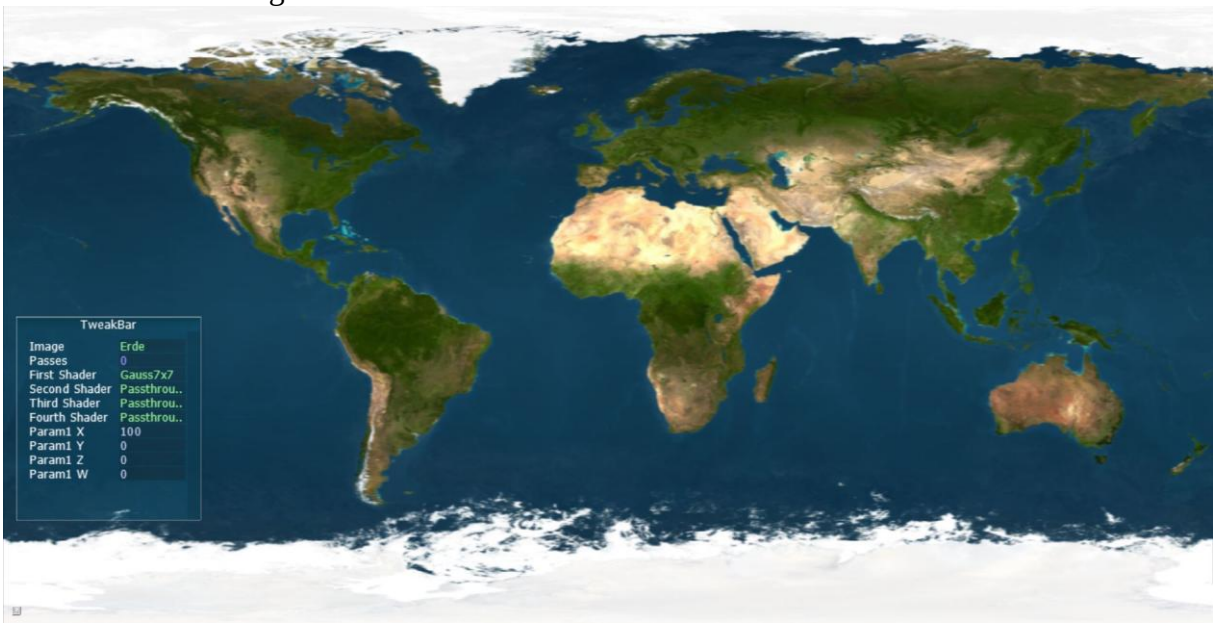
Mit einem 7x7-Fenster im Gauß-Tiefpass-Filter ist das Verschwimmen des Bildes nun gut wahrzunehmen. Ansonsten funktioniert er äquivalent zum Gauß-Tiefpass-Filter mit 3x3 oder 5x5-Fenster. Allerdings macht sich hier die Performance schon deutlich bemerkbar, denn nur die 49 Iterationen in der Schleife,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

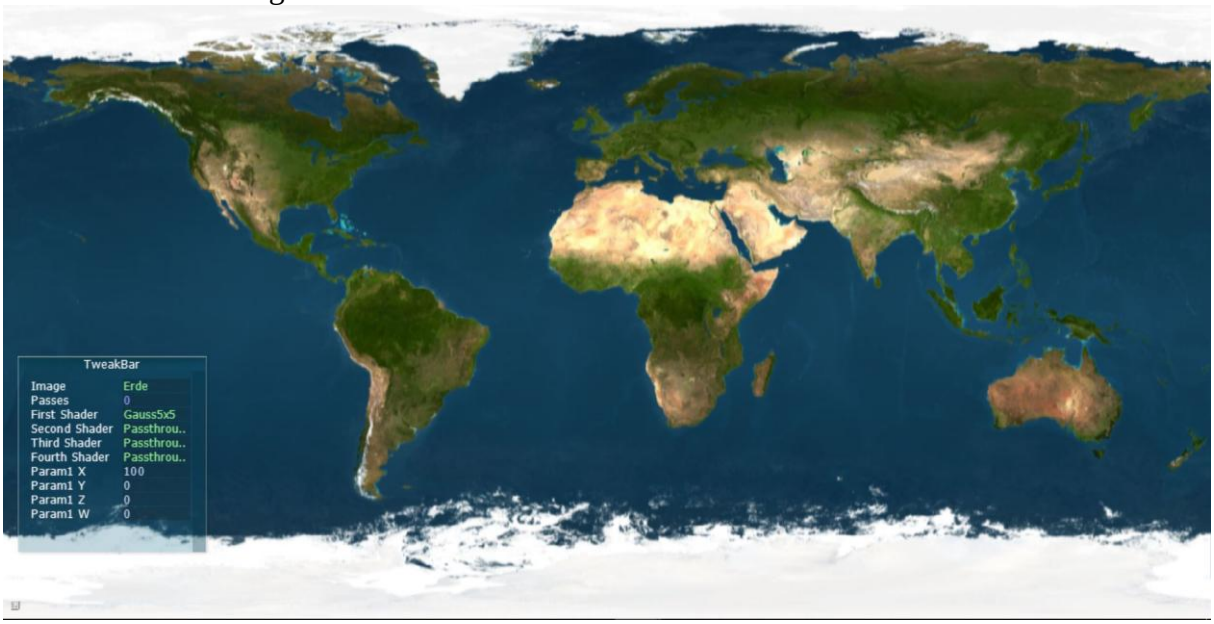
Shader in X-Richtung:



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      }
```

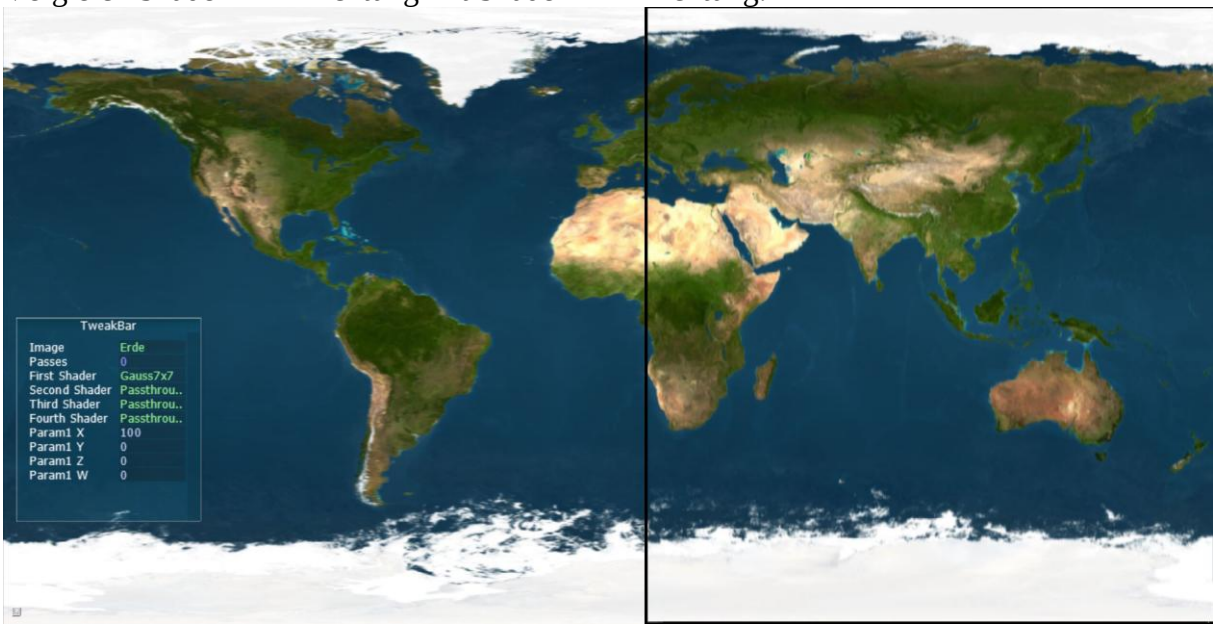
Shader in Y-Richtung:



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.

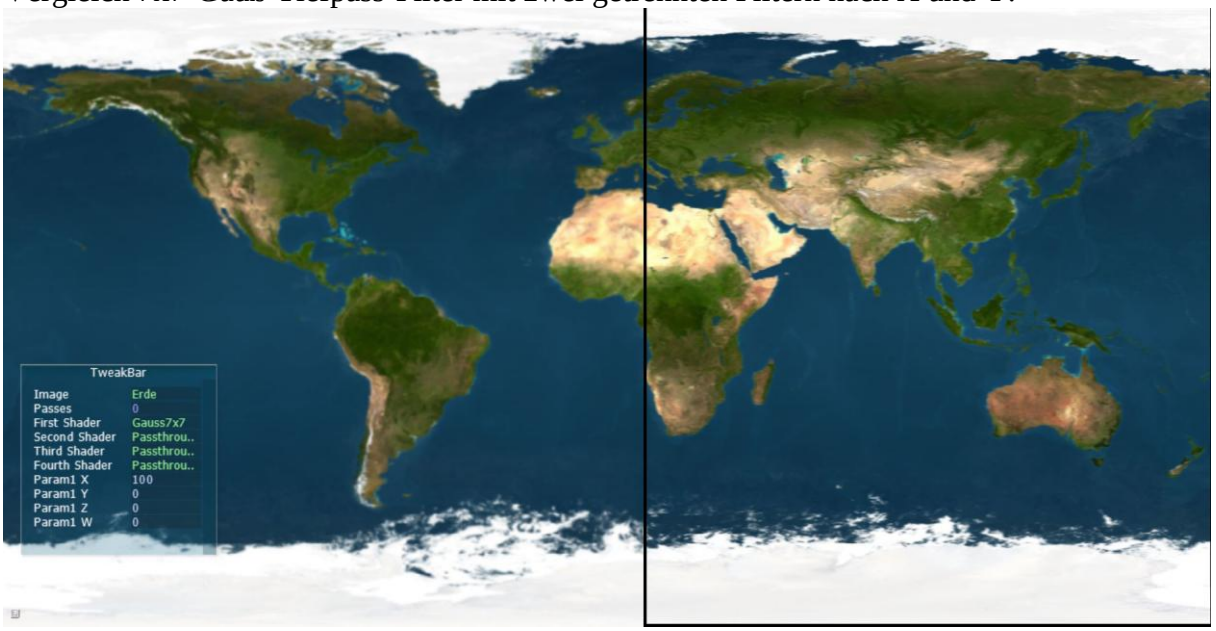
```
64          vec4 texel;
65          float pi = 3.14159265359;
66          float h;
67          float counter = 0.0 ;
68          float p1 = param1[0] / 100;
69
70          if (p1 != 0) {
71
72          for (int i = 0; i < 7; i++) {
73                  float y = offsets[i][1];
74
75                  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:



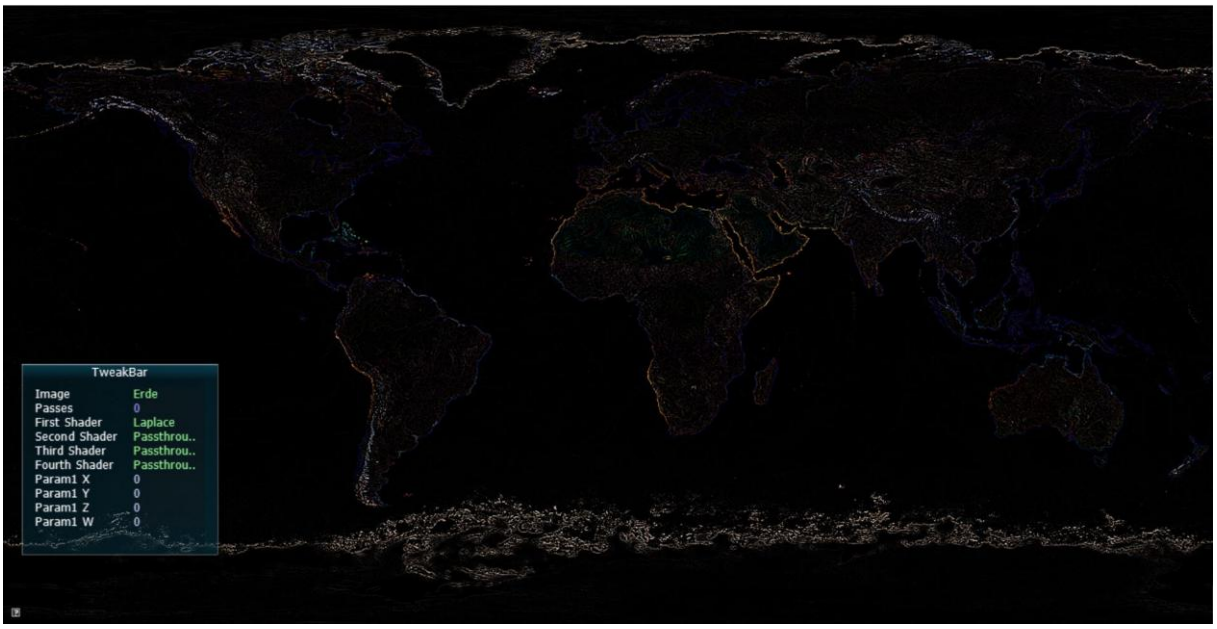
In diesem Bild oben sieht man nun den Unterschied zwischen der Verwischung auf der horizontalen und der Verwischung auf der vertikalen Linie.

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) FragmentShaderLaplace.glsl



Mit Hilfe des zweidimensionalen Laplace-Filters (siehe https://de.wikipedia.org/wiki/Laplace-Filter), 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.

```
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
41
42      fragColor = texel;
```