

Gernot Hoffmann

Gaussian Filters



Carl Friedrich
Gauß
1777 - 1855

Contents

1. Introduction	2
2. Algorithm	3
3. Blur Control	3
4. Improvements	4
5. Binary Weighting	5
6. Arbitrary Filter Scaling	6
7. Standard n-order Sharpening Filter	7
8. General n-order Sharpening Filter	8
9. Contour Filters	10
10. Motion Blur	12
11. Laplacian Filters	13
12. Bode Plots for 2D Filters	17
13. References	18

Settings for Acrobat

Edit / Preferences / General / Page Display (since version 6)

Custom Resolution 72 dpi

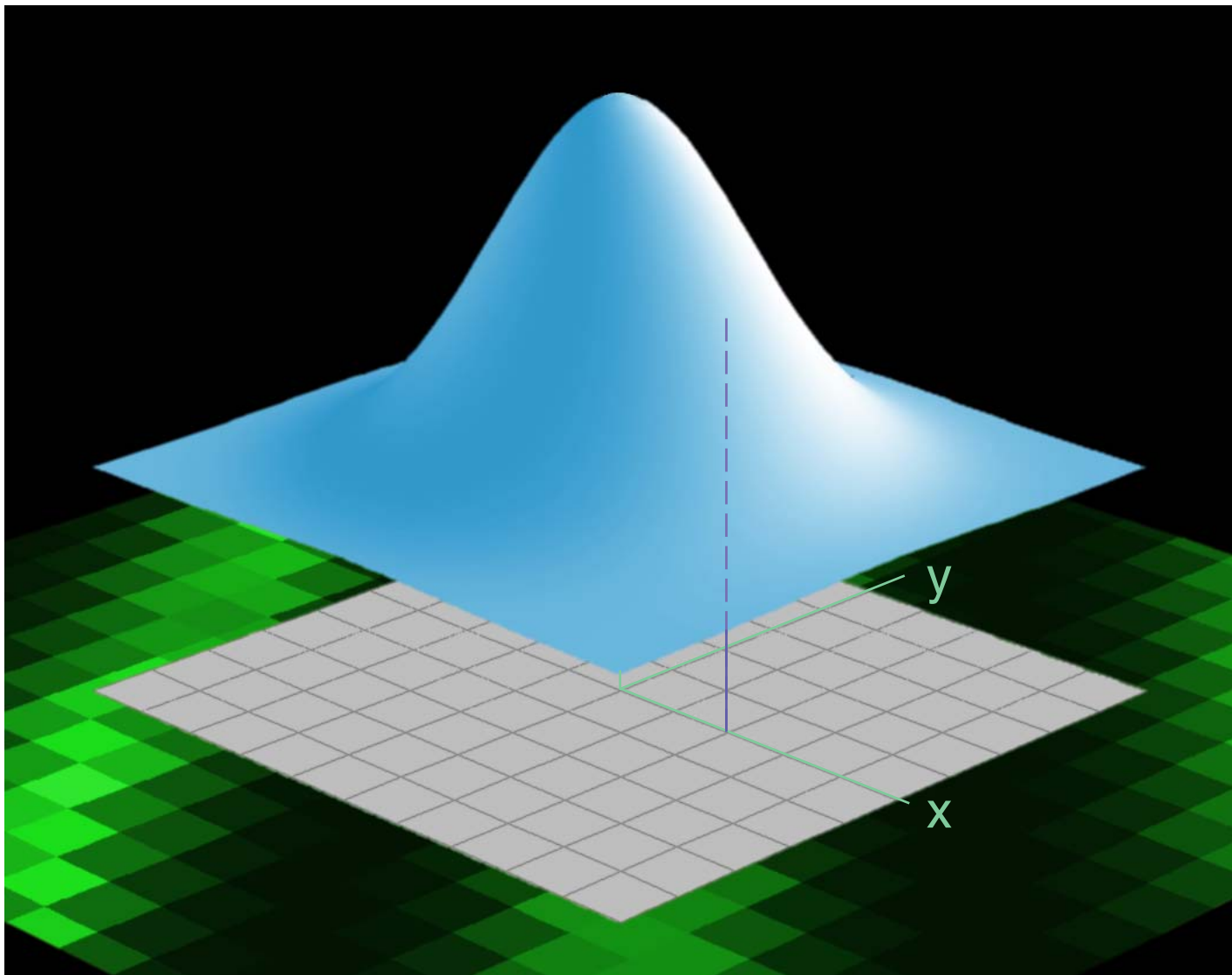
Edit / Preferences / General / Color Management (full version)

sRGB

EuroscaleCoated or ISOCoated or SWOP

GrayGamma 2.2

1. Introduction



A Gaussian filter smoothes an image by calculating weighted averages in a filter box.

Coordinates x_0, y_0 are arbitrary pixel positions in a bitmap image. x, y is a local coordinate system, centered in x_0, y_0 , as shown.

The gray area is a filter box with $m \cdot m$ knots. Box coordinates x and y reach from $-n$ to $+n$. The box width $m = (2n+1)$ is assumed odd.

Weight factors are calculated for a Gaussian bell by $w(x, y) = e^{-a}$ with $a = (x^2 + y^2) / (2r^2)$.

The filter radius r is in statistics the standard deviation sigma.

Choose $n = (2 \dots 3)r$ or $r = 0.465n$ for a reasonable reproduction without clipping.

E.g. for $x=r, y=0$ we find $w = e^{-0.5} = 0.6065$.

The image shows the function relative to the filter box vertically shifted. In the image the radius is $r=2$.

The algorithm on page 2 is not optimized. The algorithm can be made much faster by binary weight factors, because then the whole calculation is in Integer and the multiplications are merely shifts (as on page 5).

Note:

The image quality is optimal only for direct view by Acrobat. Browsers are sometimes not accurate. Please use Zoom 100%.

2. Algorithm

General Weight Factors

```
S=0
r2=2·Sqr(r)
For y=-n to +n Do
For x=-n to +n Do
Begin
  a=(Sqr(x)+Sqr(y))/r2
  w(x,y)=exp(-a)
  S=S+w(x,y)
End
```

General Image Filtering

```
For yo=n to ymax-n Do
For xo=n to xmax-n Do
Begin
  newred=0
  newgrn=0
  newblu=0
  For y=-n to n Do
  For x=-n to n Do
  Begin
    newred=newred+w(x,y)·red(x+xo,y+yo)
    newgrn=newgrn+w(x,y)·grn(x+xo,y+yo)
    newblu=newblu+w(x,y)·blu(x+xo,y+yo)
  End
  newred=newred/S
  newgrn=newgrn/S
  newblu=newblu/S
End
```

3. Blur Control

The blurring is controlled by two parameters:

- 1) The box width, described by $m=(2n+1)$ pixels in one direction
- 2) The radius r

The Gaussian bell in one direction delivers:

x/r	-3	-2	-1	0	1	2	3
w(x)	0.0111	0.1353	0.6065	1.0	0.6065	0.1353	0.0111

We can choose $r=0.465n$. This results in a weight factor 0.1 at the outermost pixel at $x=n$, which seems to be reasonable. Less than 0.1 does not make much sense. For pixels on the diagonal corners of the xy -box the value is anyway smaller.

Weight factors. Weak blur $n=1$, $r=0.465$. Strong blur $n=3$, $r=1.398$.

Weak blur:			0.100	1.0	0.100		
Strong blur:	0.100	0.358	0.773	1.0	0.773	0.358	0.100

4. Improvements

The Gauss formula can be separated. This will make the calculations faster.

$$w(x,y) = e^{-(xx+yy)} = w(x)w(y) = e^{-xx} \cdot e^{-yy}$$

Another method uses one source image, one array of the same size for the accumulation and a sequence of shifted images. This shifted image is made once for each position x,y for all pixels in the source image.

The author prefers the standard structure, because this is valid for any linear filter, like softening (blurring), sharpening and contour finding filters, also for some effects which use oscillations in the box.

A 5x5 binary Gaussian filter, programmed mainly in Intel Assembly Language, needs about one second for 1000x1000 pixels (PC 400MHz).

Note: the Gaussian filter in the straightforward kernel implementation cannot be executed inplace. One needs always a source framebuffer and a destination framebuffer.

5.1 Binary Weighting

This filter shows a crude approximation of the Gaussian bell function.

The weight factors are powers of 2, thus multiplications by weight factors can be replaced by binary shifting.

The sum of the weight factors is $S=80$. If all colors values are 255, then the weighted sum is 20400, which does not exceed the positive number space for Integer $2^{15}-1=32767$.

Radius $r=0.85$, approximately.

0	1	2	1	0
1	4	8	4	1
2	8	16	8	2
1	4	8	4	1
0	1	2	1	0

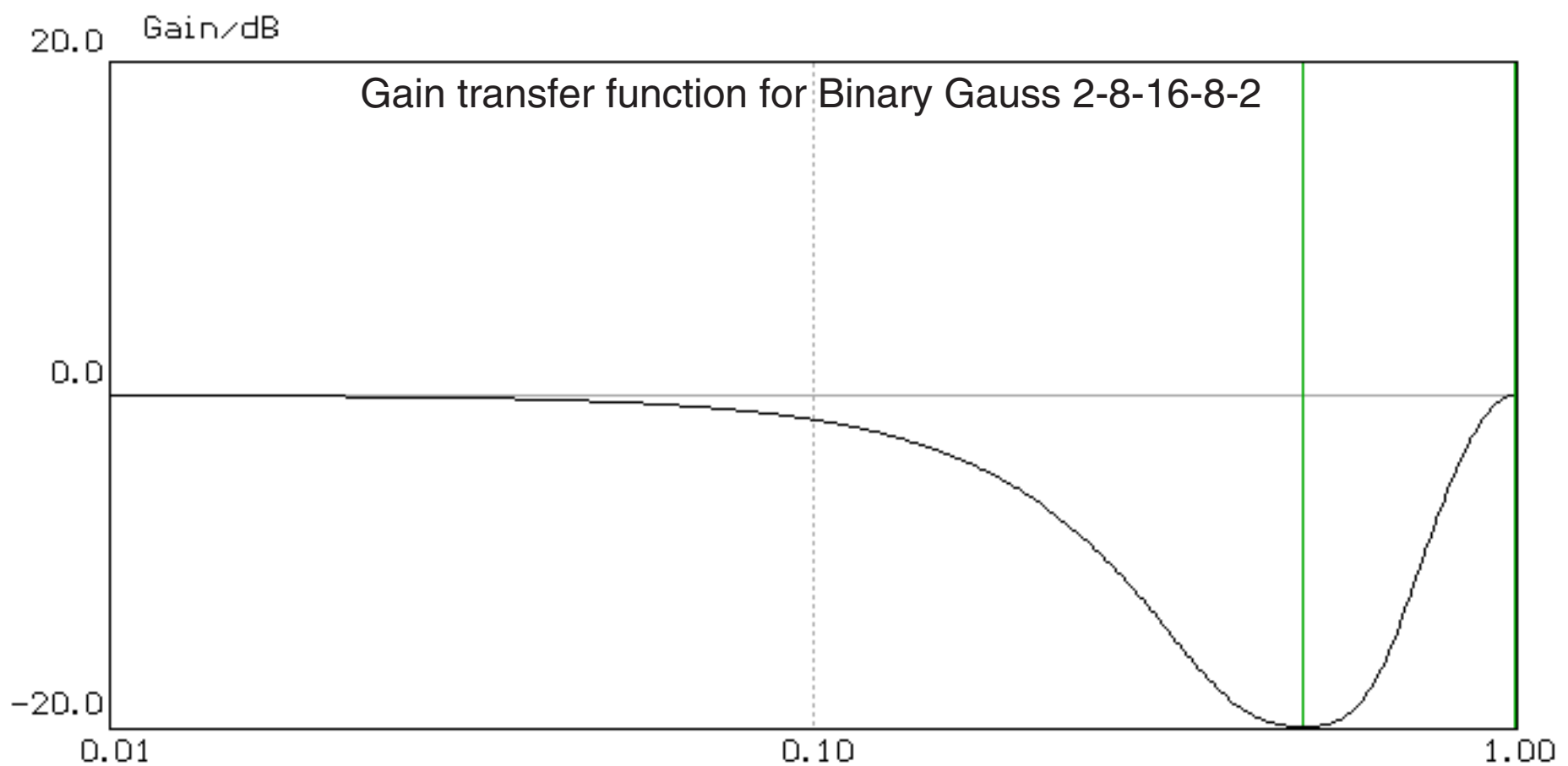
For more general applications binary weighting by LongInt (32bit) could be used. This is still much faster than floating point operations.

Part of a digital photo. Left not filtered. Right softened by Binary Weight Filter. Then both images were scaled down for 50% and placed into PDF by pixel synchronization.

The synchronization is perfect only for direct view by Acrobat but not necessarily by browser.



5.2 Binary Weighting Transfer Function



6. Arbitrary Filter Scaling

If the weight factors belong to a consistent set of data, like in the previously mentioned softening filters, then we have to divide all raw weight factors by the sum of the raw weight factors.

Not so for a general sharpening filter.

We start by a positive peak in the center and all negative values are taken from a Gaussian bell.

The binary weighting is not essential in this example. Generally spoken, we have raw positive weight factors P_i and raw negative weight factors N_i .

The actual filtering is done by scaled weight factors $p_i = 2P_i / S_p$ and $n_i = N_i / S_n$.

S_p = Sum of positive weight factors P_i

S_n = Sum of negative weight factors $|N_i|$

This scaling is based on the demand that a uniform color area should deliver the same color after filtering.

Example, as above:

$$S_p = 64$$

$$S_n = 4 \cdot 8 + 4 \cdot 4 + 4 \cdot 2 + 8 \cdot 1 = 64$$

The center peak in the scaled matrix is +2 and the other negative values are divided by 64.

For band pass and highpass filters we use equal sums and a contrast factor:

$$p_i = CP_i / S_p \text{ and } n_i = CN_i / S_n.$$

-0	-1	-2	-1	-0
-1	-4	-8	-4	-1
-2	-8	+64	-8	-2
-1	-4	-8	-4	-1
-0	-1	-2	-1	-0

7. Standard n-order Sharpening Filter

1	2	3	4	5	-0.0035	-0.0159	-0.0262	-0.0159	-0.0035
6	7	8	9	10	-0.0159	-0.0712	-0.1173	-0.0712	-0.0159
11	12	13	14	15	-0.0262	-0.1173	2.0	-0.1173	-0.0262
16	17	18	19	20	-0.0159	-0.0712	-0.1173	-0.0712	-0.0159
21	22	23	24	25	-0.0035	-0.0159	-0.0262	-0.0159	-0.0035

The drawing shows the numbering and the weight factors for the kernel for a sharpening filter, here with $n=2$. The algorithm works for any $n \geq 1$. The total number of elements is $N=(2n+1)^2$.

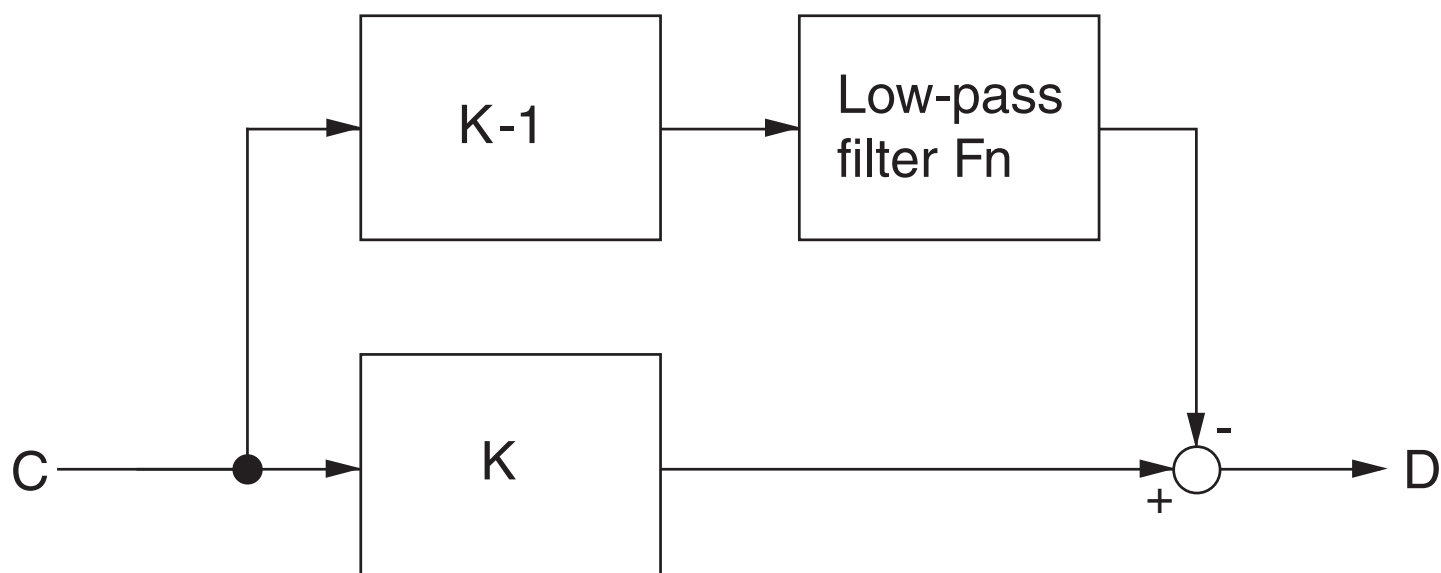
The center weight factor $fs[13]=2.0$ is a positive peak. The other weight factors $fs[k]$ are calculated by a negative Gaussian bell, according to the code below.

The sum of negative weight factors is -1.0 and the sum of all weight factors is $+1$, therefore a uniformly colored area remains unfiltered, as required.

```
Tutorial code, not optimized
sm:=0;
k :=1;
For j:=-n to n Do
For i:=-n to n Do
Begin
  ra:=Sqrt(Sqr(i)+Sqr(j))/n;
  ra:=exp(-2*Sqr(ra));
  fs[k]:=-ra;
  If (i<>0) Or (j<>0) Then sm:=sm+ra;
  Inc(k);
End;
k :=1;
For j:=-n to n Do
For i:=-n to n Do
Begin
  fs[k]:=fs[k]/sm;
  If (i=0) And (j=0) Then fs[k]:=2.0;
  Inc(k);
End;
```

8.1 General n-order Sharpening Filter / Konzept

This drawing shows the signal flow for a general sharpening filter. The low-pass filter F_n can be established by a Gaussian bell as explained in the chapters 1 to 3.



The input C is an unfiltered value $C=R, G$ or B at the center position. The output D is the sharpened value at the same center position. The low-pass filter executes the averaging as usual by a Gaussian bell. The filtered value is subtracted from the unfiltered value, but the two are multiplied by factors $(K-1)$ and K .

$$D = [K - (K-1) F_n] C$$

The factor K has this meaning:

- $K=1$ no sharpening
- $K=2$ sharpening as in previous chapter
- $K=1.5$ less sharpening than for $K=2$

$K=2$ is a very reasonable value but sometimes a little more control for the filtering is welcome. This filter has two degrees of freedom: the filtering order n and the factor K .

For a uniformly colored area the low-pass filter delivers just the input C and the total output D is the unchanged input C as well.

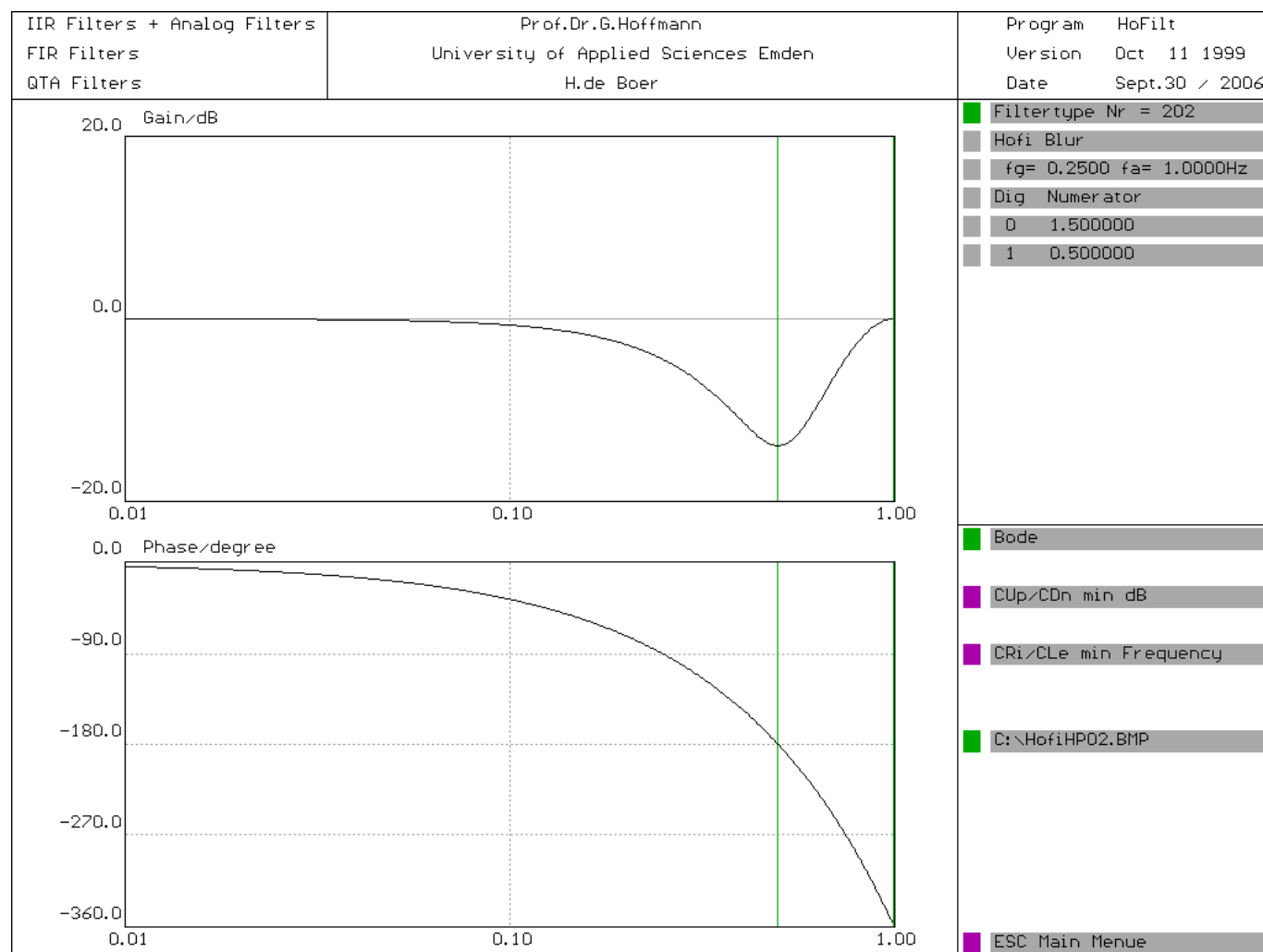
Note: the sharpening filter as above and the blurring filter (low-pass) are not complementary. This means that sequential applications do not cancel each other.

8.2 General n-order Sharpening Filter / Example

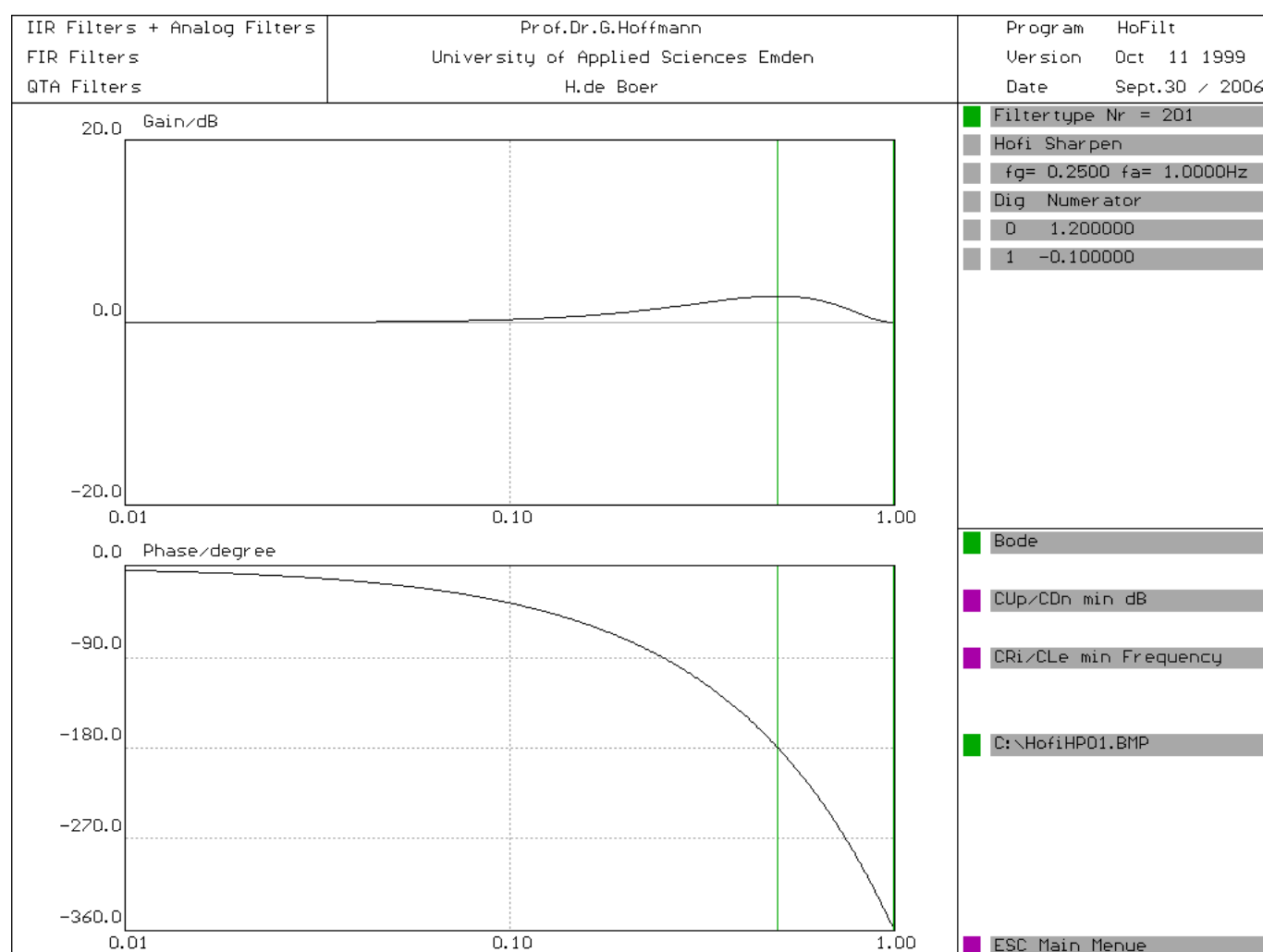
Example: a weak sharpening filter with n=1 and K=1.5. Mainly for eyes and hair in portraits.

$$F = K \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix} - \frac{K-1}{2.5} \begin{vmatrix} 1/8 & 1/4 & 1/8 \\ 1/4 & 1 & 1/4 \\ 1/8 & 1/4 & 1/8 \end{vmatrix}$$

The upper diagram shows the Bode plot for the low pass filter. The lower diagram shows the Bode plot for the sharpening filter. Coefficients were re-calculated according to chapter 12.



Use in PDF
72 dpi / zoom 200%



9.1 Contour Filters / Concept

The well-known contour filters by *Sobel*, *Roberts* and *Prewitt* [3] use 3x3 kernels, which have to be applied twice: in x- and y-direction.

They can be easily substituted by the nonlinear filter below, which finds contours in one pass. The sum of the absolute values of pixel differences in four directions is compared with an adjustable threshold.

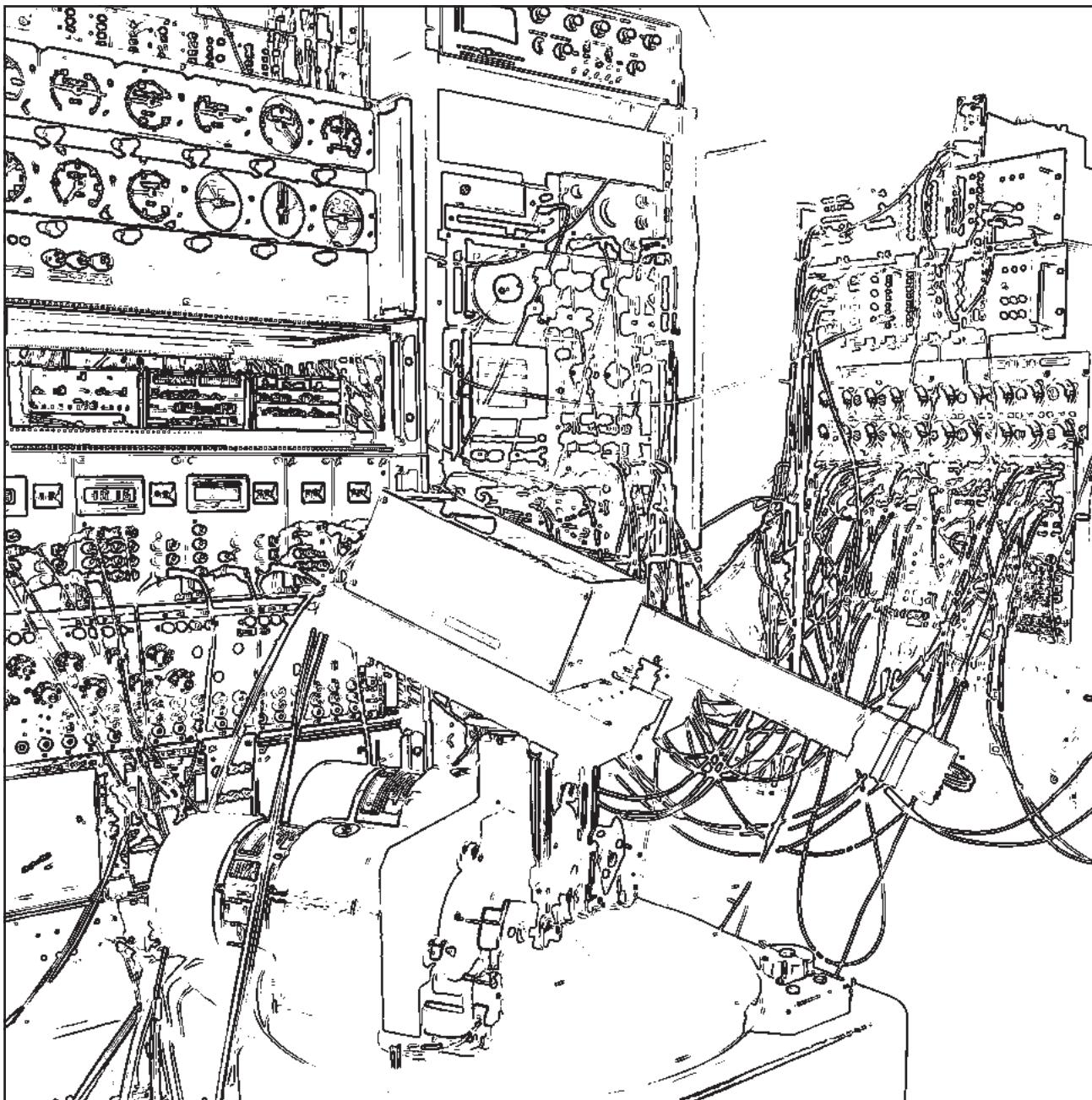
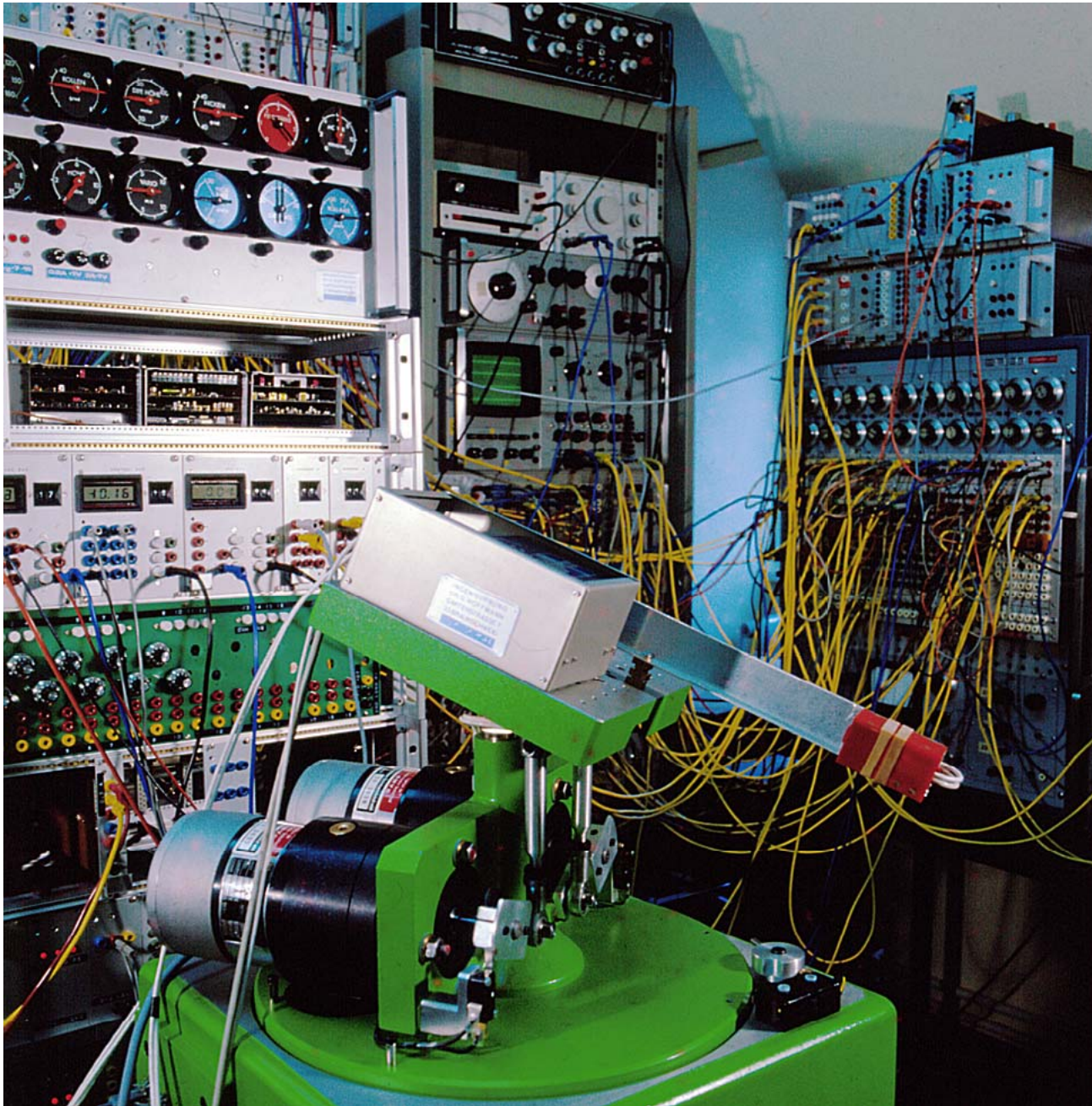
The example works for gray images. The actual implementation calculates contributions of three channels RGB and is programmed entirely in assembly code.

The contour filter creates a black-on-white mask, which is normally hidden. At black points, the original photo can be blurred (anti-aliasing) or sharpened (enhanced edges).

The famous *Canny Edge* filter is based on a multi-pass approach. Opposed to the filters above, the contour is delivered by single pixel lines or fragments of them.

```
Procedure FContour(xa,ya,xe,ye: Integer; thresh: Single);
{ Uses 3x3 points in area xa..xe, xa..ye }
{  1 2 3
  4 5 6
  7 8 9 }
Var cc,clim          : Single;
    x,y,c1,c2,c3,c4,c6,c7,c8,c9 : Integer;           { 16 bit }
    prgb1,prgb2,prgb3,prgb4,prgb6,prgb7,prgb8,prgb9: LongInt; { 32 bit }
Begin
clim:=50+250*(thresh-0.1);           { Threshold 0...1 }
FMemGrYIQ  (xa,ya,xe,ye,xa,ya,2);    { FMem gray }
FMemtoPMem (xa,ya,xe,ye,xa,ya);      { PMem copy }
ColToSx    (0,0,gmx,gmy,stan,whit);  { Screen white, global gmx,gmy }
For y:=ya+1 To ye-1 Do
  Begin
  For x:=xa+1 To xe-1 Do
  Begin
  prgb1:=GetPixel (x-1,y-1);         { PMem }
  prgb2:=GetPixel (x  ,y-1);
  prgb3:=GetPixel (x+1,y-1);
  prgb4:=GetPixel (x-1,y  );
  prgb6:=GetPixel (x+1,y  );
  prgb7:=GetPixel (x-1,y+1);
  prgb8:=GetPixel (x  ,y+1);
  prgb9:=GetPixel (x+1,y+1);
  c1:=prgb1 AND $000000FF;
  c2:=prgb2 AND $000000FF;
  c3:=prgb3 AND $000000FF;
  c4:=prgb4 AND $000000FF;
  c6:=prgb6 AND $000000FF;
  c7:=prgb7 AND $000000FF;
  c8:=prgb8 AND $000000FF;
  c9:=prgb9 AND $000000FF;
  cc:=Abs(c1-c9)+Abs(c2-c8)+Abs(c3-c7)+Abs(c4-c6);
  If cc>clim Then SetSixel(x,y,0); { Screen black }
  End; { x }
  End; { y }
  End;
End;
```

9.2 Contour Filters / Examples



Use in PDF
72 dpi / zoom 200%

Threshold 0.5

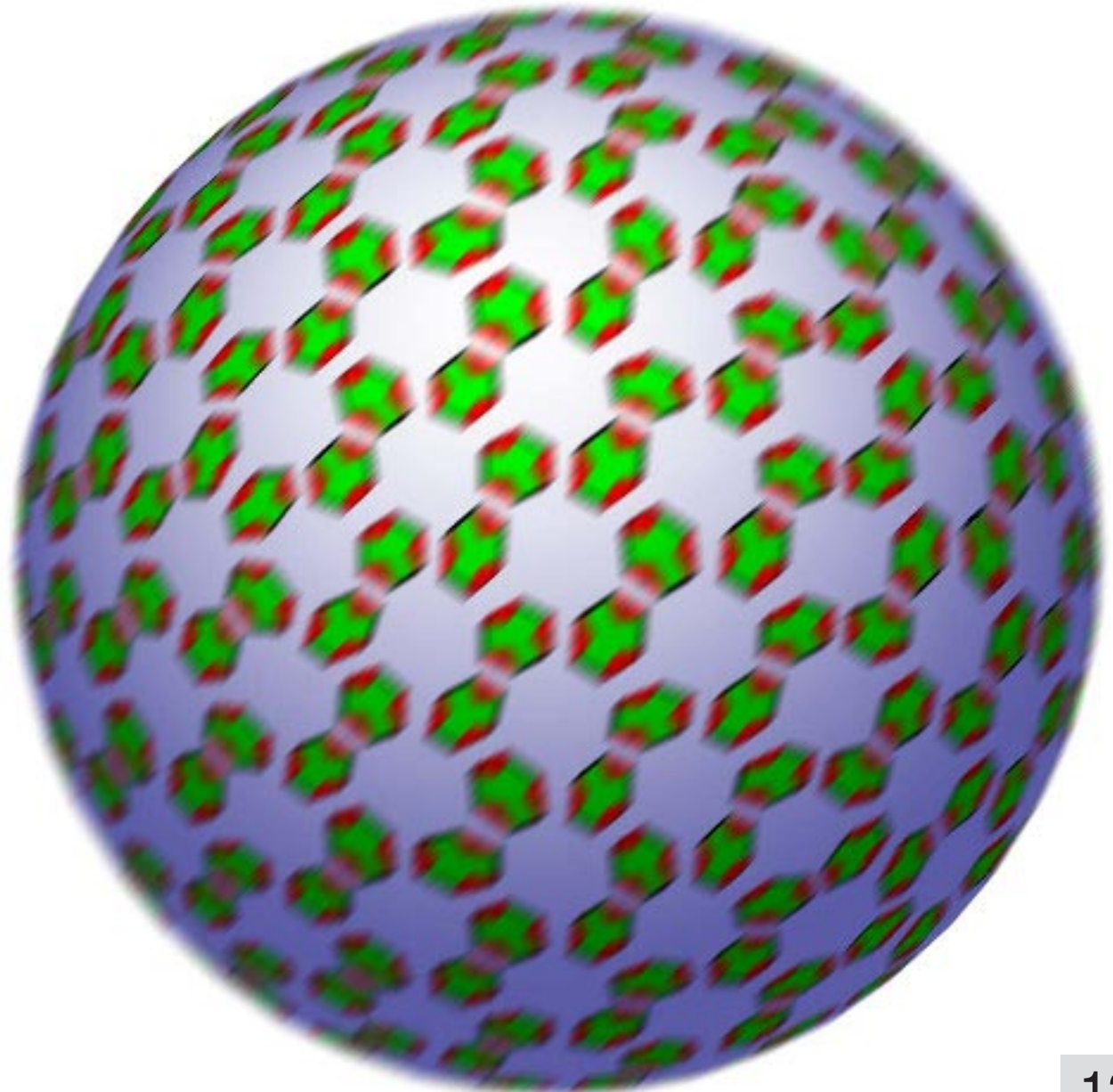
10.1 Motion Blur / Concept / Example 45°

Motion blur can be created by an elliptic Gaussian bell. The longer axis 'a' is aligned with the velocity vector. The elliptic Gaussian bell is created in coordinates u,v for an axis aligned ellipse and then transformed into a rotated ellipse in x,y.

The filter box contains many zeros and values near to zero. The speed can be increased by rounding near-zeros to zero and not executing filtering for these weight factors. A more tricky solution could use a list of valid weight factors.

The effect is very similar to motion blur by Photoshop®.

```
Va:=intens;           { Velocity 0...2 }
n:=Round(10*Va);
If n<1 Then n:=1;
a1:=0.2+0.5*n;
b1:=0.2;
a2:=Sqr(a1);
b2:=Sqr(b1);
siccoc(-angle*wrad,s,c); { Fast sine,cosine }
k:=1; fmax:=0;
For y:=-n to n Do
For x:=-n to n Do
Begin
u:=+c*x+s*y;
v:=-s*x+c*y;
fsk:=exp(-0.5*(u*u/a2+v*v/b2));
fmax:=fmax+fsk;
fsk[k]:=fsk;
Inc(k);
End;
k:=1;
For y:=-n to n Do
For x:=-n to n Do
Begin
fsk[k]:=fsk[k]/fmax;
Inc(k);
End;
```



11.1 Laplacian Filters / Concept / Preliminary

Laplacian filters are used e.g. in the context of the *Canny Edge* algorithm.

A general Laplacian filter is derived from a Gaussian bell as the sum of the second derivatives (Laplacian of Gaussian, LoG):

$$f(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$
$$w(x,y) = f_{xx} + f_{yy}$$
$$= \frac{2}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(\frac{x^2+y^2}{2\sigma^2} - 1 \right)$$

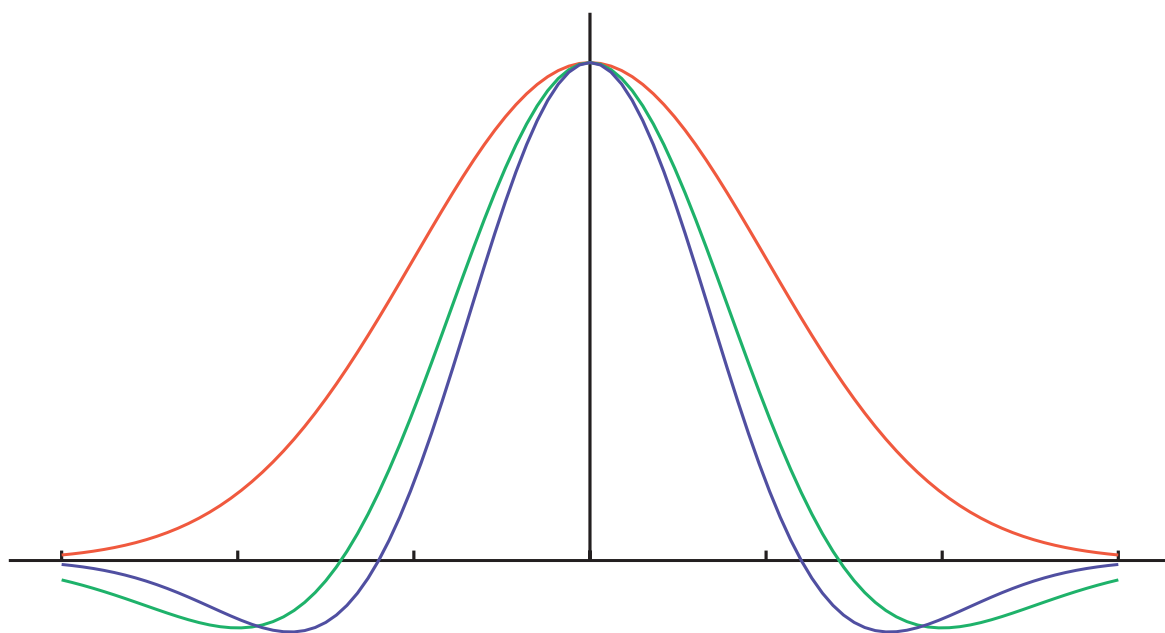
$$w(x,y) = e^{-r^2}(1-r_2)$$

For convenience, the leading factor was ignored and the sign reversed.

For cross sections of the functions $f(x,y)$ and $w(x,y)$ in one direction, we can set $y=0$. This is not the same as drawing the functions for the bell $f(x)$ and $w(x)$. Here we would have $w(\sigma)=0$. For the function $w(x,y)$ we have $w(\sigma,\sigma)=0$.

The graphic shows the bell (red), the Laplacian filter gain in one direction (green) and an equivalent Damped Cosine (blue), which is a kind of *Gabor* filter.

Tic marks are in sigma-distance.



For a sufficiently large kernel and properly normalized, this filter will work blurring with edge enhancement. This is agreeable for noisy images.

Opposed to the design as above, the simplest Laplacian filter for edge detection is a high pass filter (here with reversed sign, compared to text books, which does not matter):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The LoG filter and the Damped Cosine filter can be converted into true band pass filters (high pass for smallest kernel) by a special normalization:

Sum of positive values equal to +1, sum of negative values equal to -1.

The following examples are valid for this normalization.

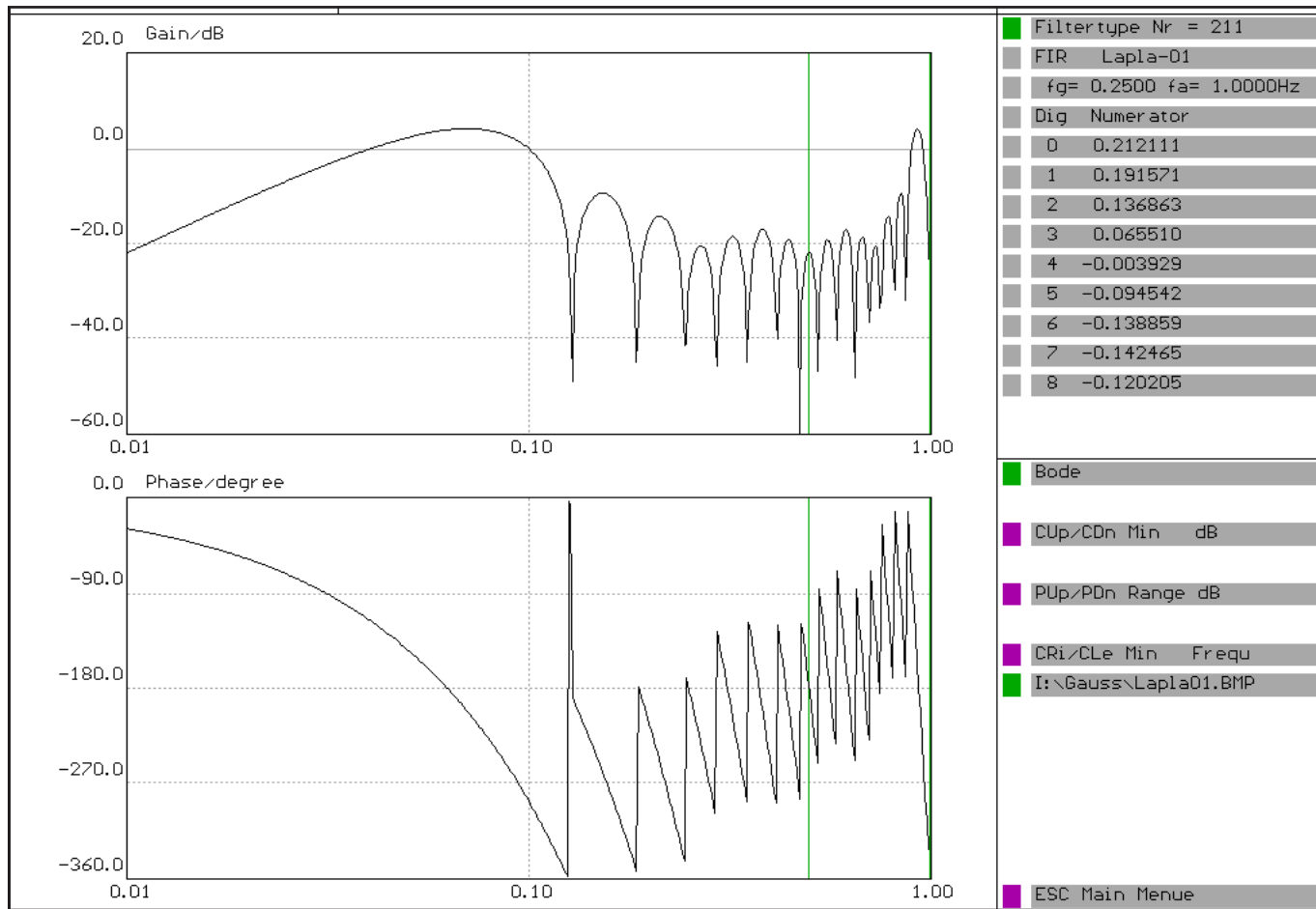
11.2 Laplacian Filters / Band Pass / Code

This code was tested:

```
n:=Round(radius);
If n<1 Then n:=1; If n>16 Then n:=16;
sig:=0.465*n; s2:=2*Sqr(sig);
k:=1; fp:=0; fm:=0; Pfil:=2;
Case Pfil Of
1: Begin
For y:=-n to n Do
For x:=-n to n do
Begin
r2:=(Sqr(x)+Sqr(y))/s2;
fsk:=exp(-r2)*(r2-1);
If fsk>0 Then fp:=fp+fsk Else fm:=fm-fsk;
fs[k]:=fsk;
Inc(k);
End;
End;
2: Begin
For y:=-n to n Do
For x:=-n to n do
Begin
r2:=Sqr(x)+Sqr(y);
r1:=1.25*pi*Sqrt(r2)/n;
r2:=r2/s2;
fsk:=exp(-r2)*cos(r1);
If fsk>0 Then fp:=fp+fsk Else fm:=fm-fsk;
fs[k]:=fsk;
Inc(k);
End;
End;
{ Alternatives for simple filters }
3: Begin
n:=1;
fs[1]:=-0.5; fs[2]:=-1; fs[3]:=-0.5;
fs[4]:=-1; fs[5]:=+8; fs[6]:=-1;
fs[7]:=-0.5; fs[8]:=-1; fs[9]:=-0.5;
fp:=8;
fm:=6;
End;
4: As on previous page

{ Contrast=0..10 }
k:=1;
For y:=-n to n do
For x:=-n to n do
Begin
fsk:=fs[k];
If fsk>0 Then fs[k]:=contr*fsk/fp Else fs[k]:=contr*fsk/fm;
Inc(k);
End;
```

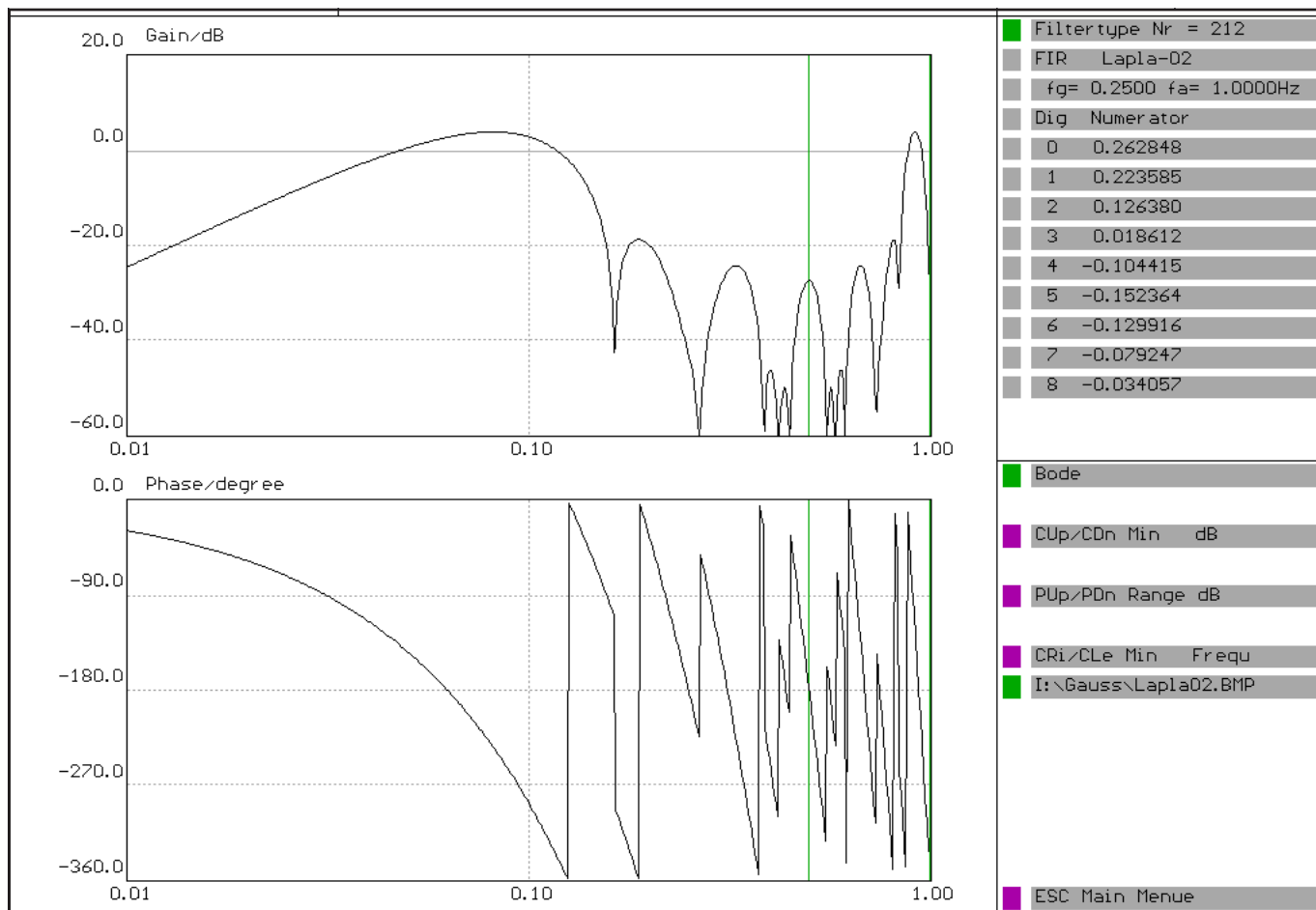
11.3 Laplacian Filters / Band Pass / Bode Plots



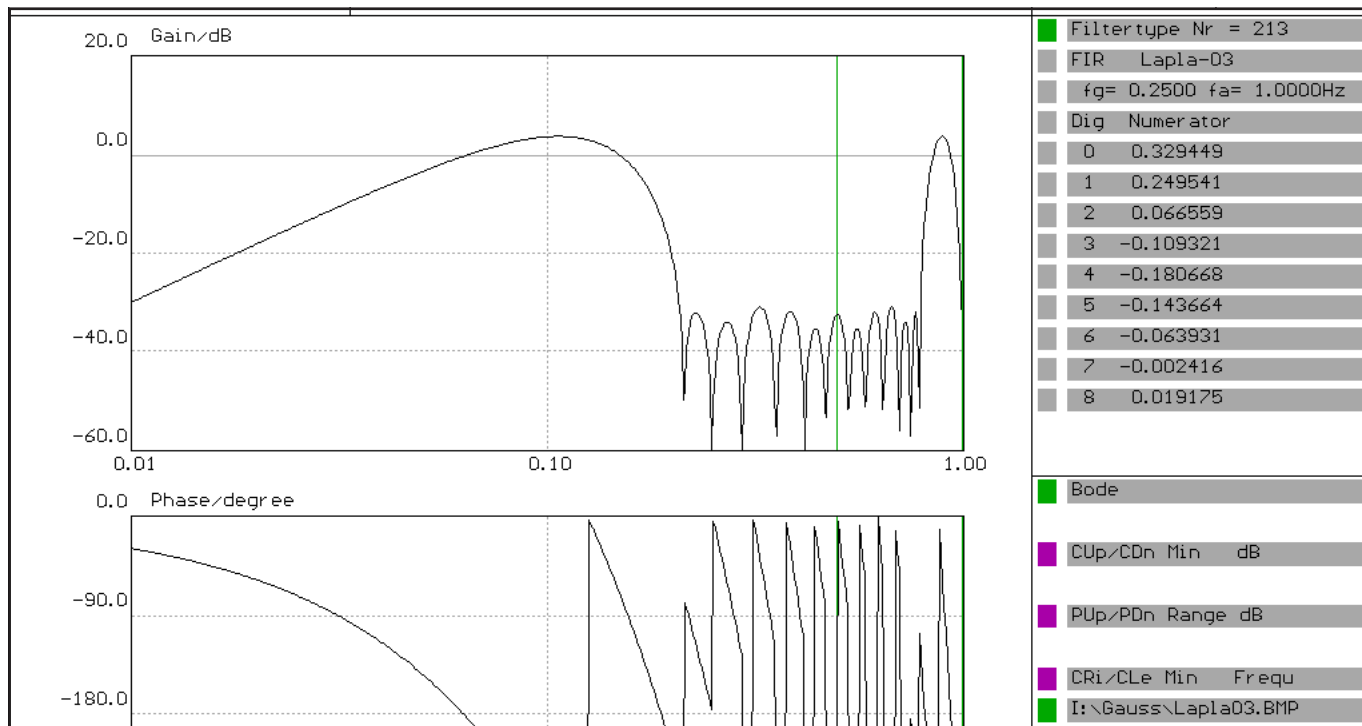
Bode plots for normalized band pass filters for n=8

Use in PDF
72 dpi / zoom 200%

Laplacian band pass filter



Damped Cosine
Better results for images



Optimized filter
This delivers bad results for images (double lines)

11.4 Laplacian Filters / Band Pass / Examples

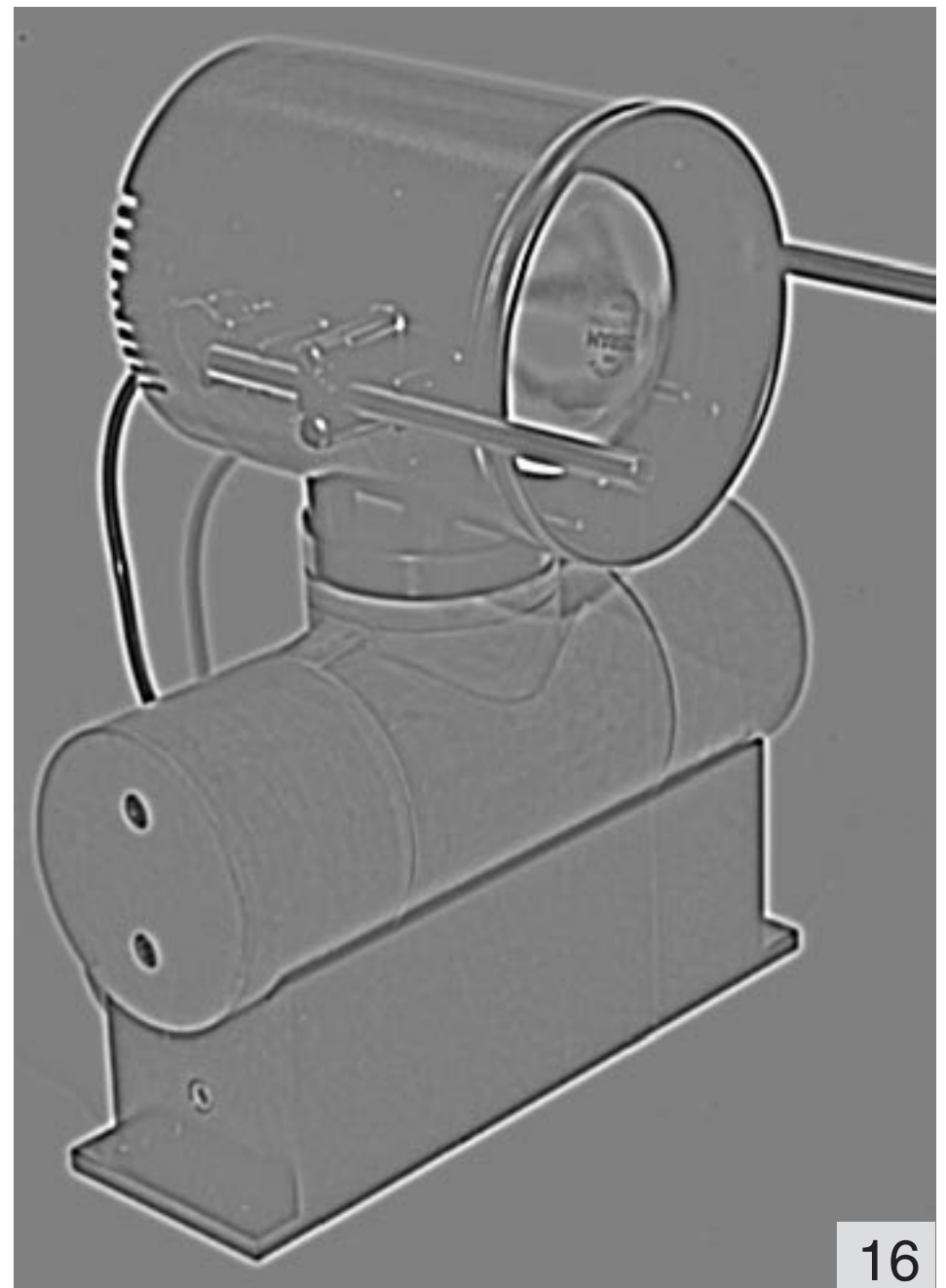
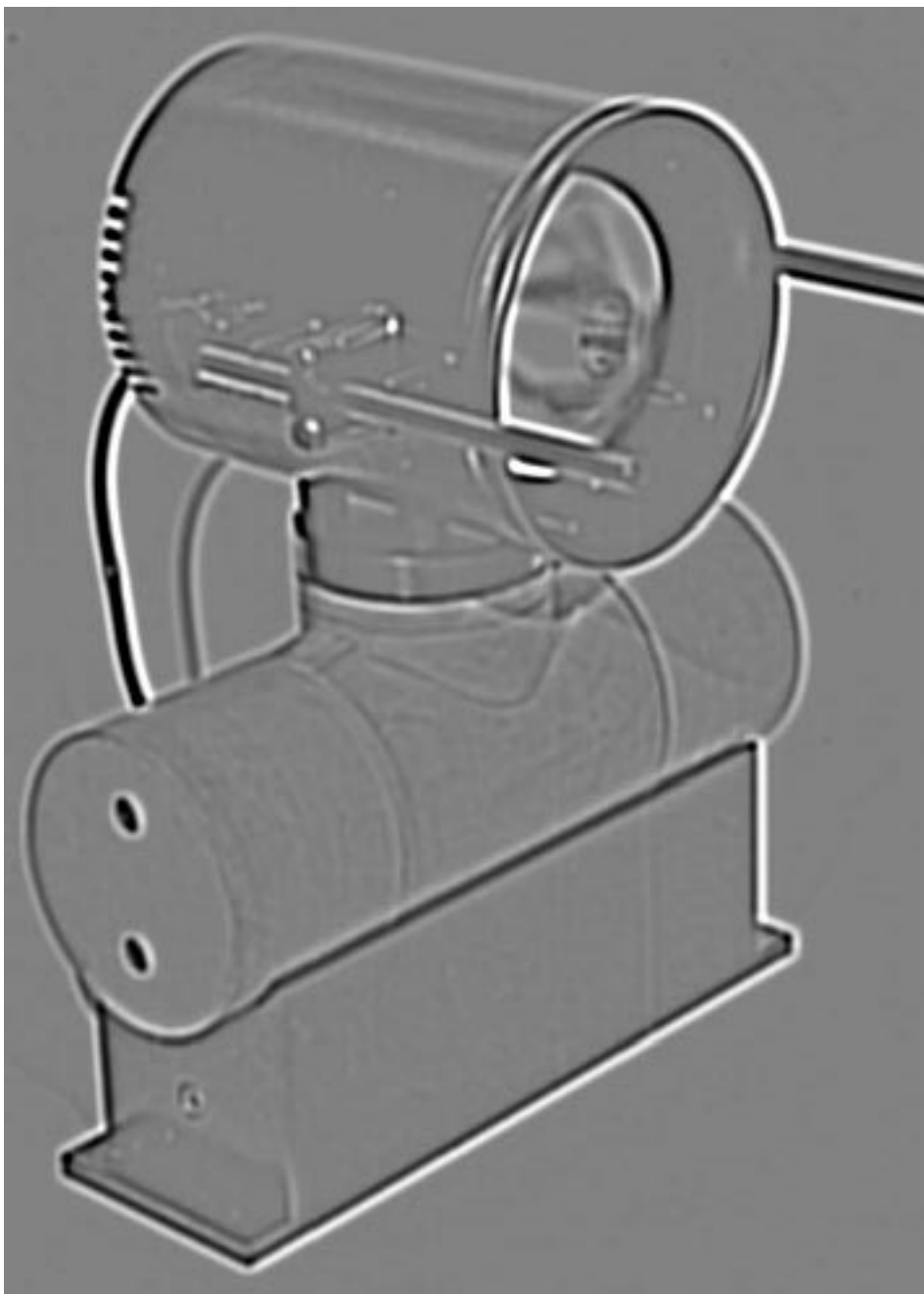


Application of bandpass filters for $n=8$, contrast=2 and background gray 128.

Noise can be removed by tri-level posterization.

Bottom left: Laplacian band pass

Bottom right: Damped Cosine band pass



12. Bode Plots for 2D Filters

Filters with weight factors $w(x,y)$ are called 2D filters. Bode plots are shown for 1D weight factors $w(x)$.

The 2D filters have (mostly) radial symmetry, but it is not correct to use just the center row values as weight factors $w(x)$ for the Bode plots.

An image may consist of a left dark gray half and a right light gray half. It has a vertical edge. Obviously it is possible to achieve by a one-row filter $w(x)$ the same effect as using the 2D filter - the edge can be blurred or sharpened.

The 1D filter $w(x)$ consists of the sum of the weight factors $w(x,y_i)$ in each column of the 2D filter.

The one-row filter $w(x)$ is normalized as described in chapter 6.

For low pass filters:

Sum of positive factors: +2

Sum of negative factors: -1

For band pass and high pass filters (eventually using a contrast factor):

Sum of positive factors: +1

Sum of negative factors: -1

Example for a Laplacian band pass:

```
n:=8;
sig:=0.465*n;
s2 :=2*Sqr(sig);
For x:=-n to n do
  Begin
    fsx:=0;
    For y:=-n to n do
      Begin
        r2 :=(Sqr(x)+Sqr(y))/s2;
        fsx:=fsx+exp(-r2)*(1-r2);
      End;
    fs[x]:=fsx;
  End;
fp:=0; fm:=0;
For x:=-n to n do
  Begin
    fsx:=fs[x];
    If fsx>0 Then fp:=fp+fsx Else fm:=fm-fsx;
  End;
For x:=-n to n do
  Begin
    fsx:=fs[x];
    If fsx>0 Then fs[x]:=fsx/fp Else fs[x]:=fsx/fm;
  End;
```

13. References

- [1] Elmar Schrüfer
Signalverarbeitung
Carl Hanser Verlag, München,Wien 1990

- [2] Samuel D.Stearns
Digitale Verarbeitung analoger Signale
R.Oldenbourg Verlag, München Wien 1988

- [3] Carsten Köhn
Bildanalyse und Bilddatenkompression
Carl Hanser Verlag München Wien 1996

- [4] Gernot Hoffmann
Fast Fourier Transform / Descreening
<http://www.fho-emden.de/~hoffmann/fft31052003.pdf>

Web docs about the filters will be added later.

This doc

<http://www.fho-emden.de/~hoffmann/gauss25092001.pdf>