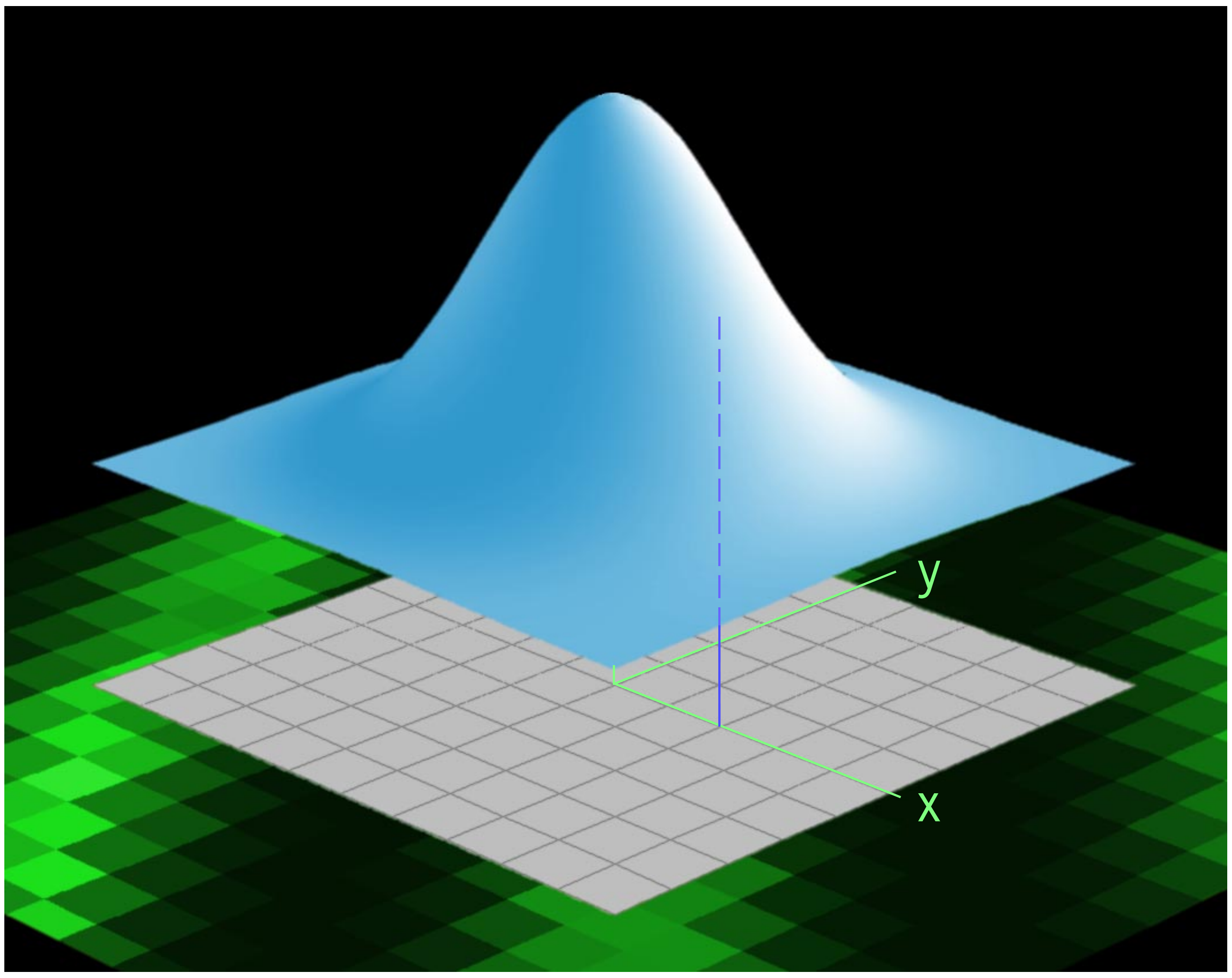
Gaussian Filter

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



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

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

Weight factors are calculated for a Gaussian bell by $w(x,y) = e^{-a}$
with $a = (x^2 + y^2) / (2 \cdot r^2)$.
The filter radius r is in statistics the standard deviation sigma.
Select $n = (2...3) \cdot r$ 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.
The radius is $r = 2$ in the image.

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 (page 4).

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

Gaussian Filter

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
```

Blur Control

The blurring is controlled by two parameters:

1) The box size, described by (2·n+1) pixels in one direction
2) The radius r

The Gaussian bell in one direction delivers:

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

We can choose r=0.465·n. This results in a weight factor 0.1 at the outermost pixel, at x=n , which seems to be reasonable. Less than 0.1 doesn´t make much sense. For pixels on the diagonal corners of the xy-box the value is smaller.

Weak blur:	small box	n=1	r=0.465					
Weight factors			0.1	1.0	0.1			

Strong blur:	large box	n=3	r=1.398				
Weight factors	0.1	0.358	0.773	1.0	0.773	0.358	0.1

Improvements

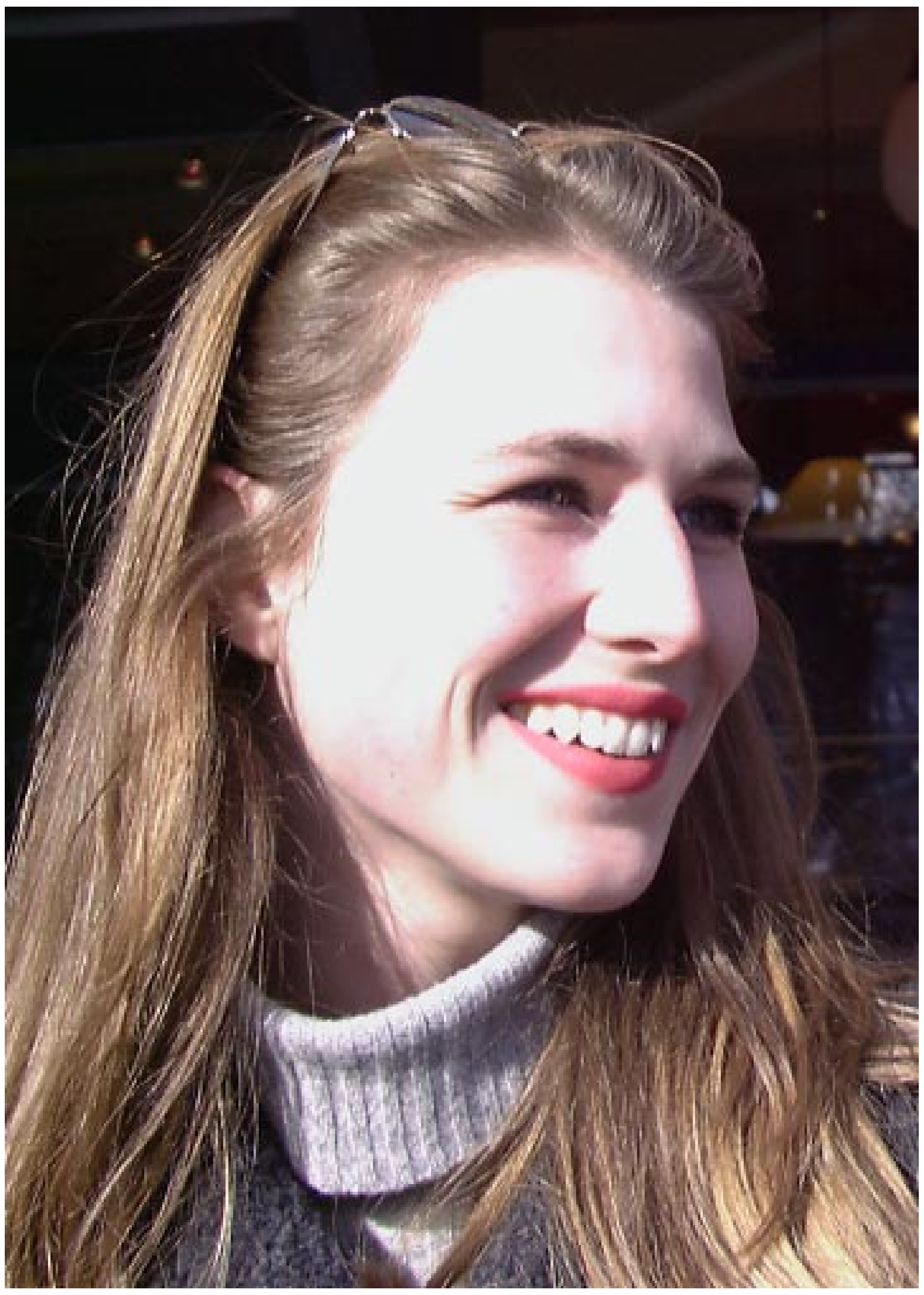
The Gauss formula can be separated. This will make the calculations faster.
$w(x,y) = e^{-(x \cdot x+y \cdot y)} = w(x) \cdot w(y) = e^{-x \cdot x} \cdot e^{-y \cdot y}$
Another methods uses one source image, one array of the same size for the accumulation and a sequence of shifted images. This shifted 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, sharpening and contour finding filters, also for some effects which use oszillations in the box.
A 5 x 5 binary Gaussian Filter, programmed mainly in Intel Assembly Language, needs about one second for 1000 x 1000 pixels.

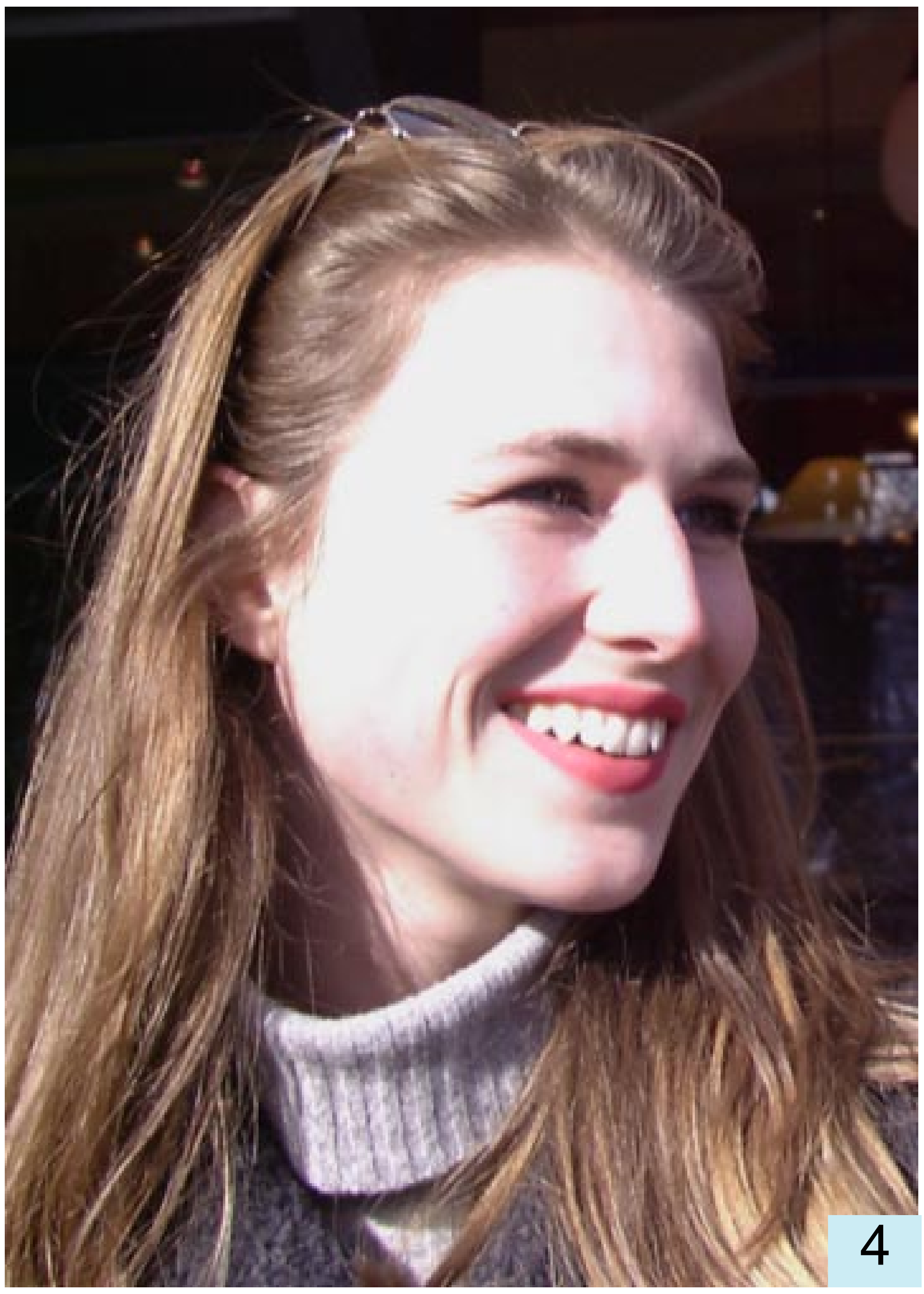
Binary Weighting

This filter shows a crude approximation of the Gauss 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.

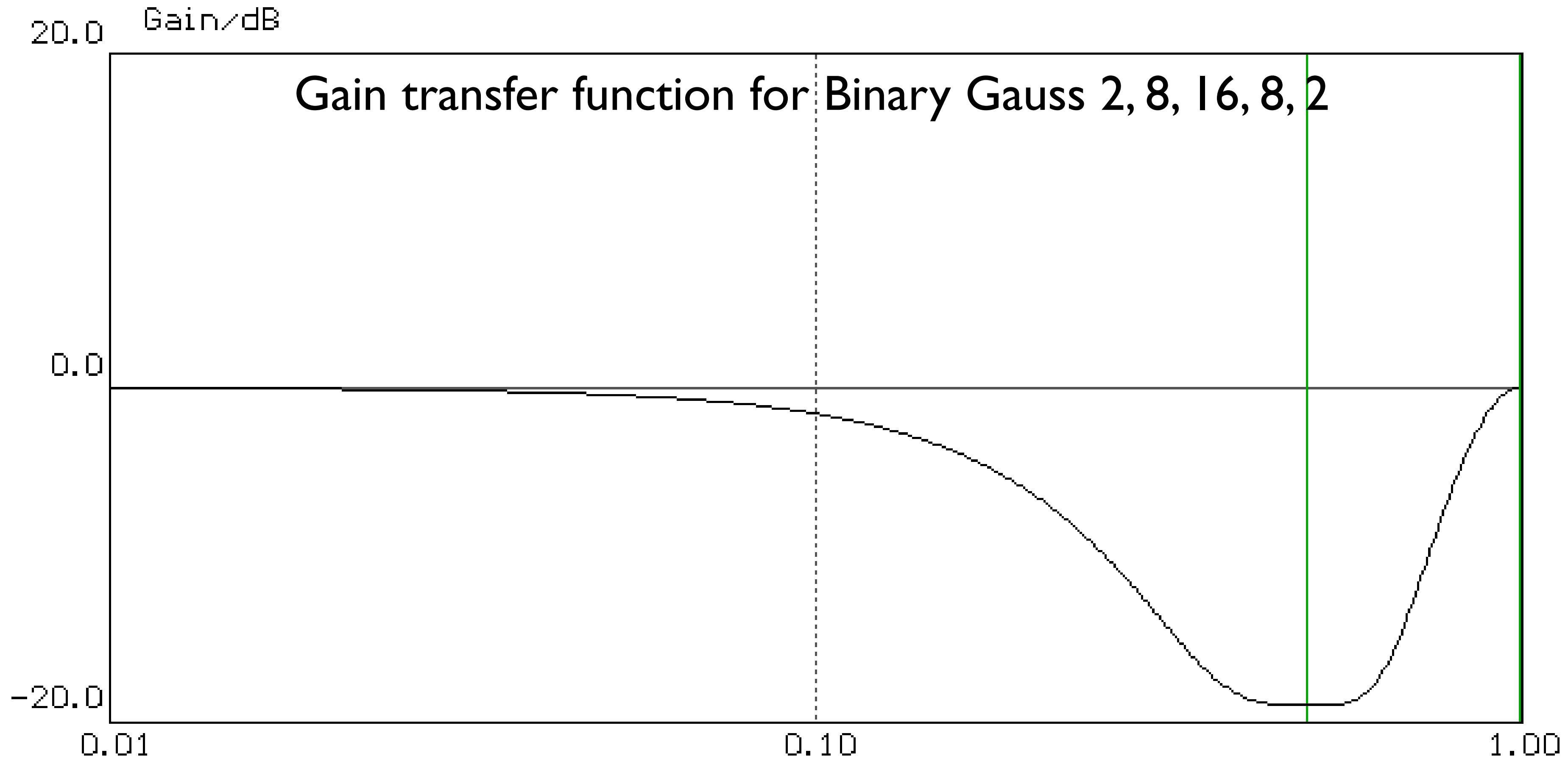
0	1	2	1	0
1	4	8	4	1
2	8	16	8	2
1	4	8	4	1
0	1	2	1	0

For more general applications binary weighting by LongInt (32bit) could be used. This is still much faster than floating point operations.
Part of digital photo. Left: not filtered. Right: softened by Binary Weight Filter. Then both images scaled down to 50% and placed into PDF by pixel synchronization. The synchronization is perfect only for direct view by Acrobat Reader, not by browser.





Binary Weighting Transfer Function



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 the raw weight factors.
Not so for a general sharpening filter.
We start by a positive peak in the center and negative values are taken from a Gauss bell. The binary weighting is not essential in this example.

-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

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 $p_i = P_i / S_p$ and $n_i = N_i / S_n$.

S_{ip} = Sum of positive weight factors P_i
S_{in} = Sum of negative weight factors $|N_i|$ (absolute values)
$S_p = 0.5 \cdot S_{ip}$
$S_n = S_{in}$

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

Example, as above :

$S_{ip} = 64$
$S_{in} = 4 \cdot 8 + 4 \cdot 4 + 4 \cdot 2 + 8 \cdot 1 = 64$

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

A General Sharpening Filter

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

-0.0035	-0.0159	-0.0262	-0.0159	-0.0035
-0.0159	-0.0712	-0.1173	-0.0712	-0.0159
-0.0262	-0.1173	2.0	-0.1173	-0.0262
-0.0159	-0.0712	-0.1173	-0.0712	-0.0159
-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 $m=(2 \cdot n+1)^2$. The center weight factor fs[13]=2.0 is a positive peak. The other weight factors fs[k] are calculated by a negative Gauss 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;
```

Gernot Hoffmann
December 20, 2002
Website: please load browser and click here