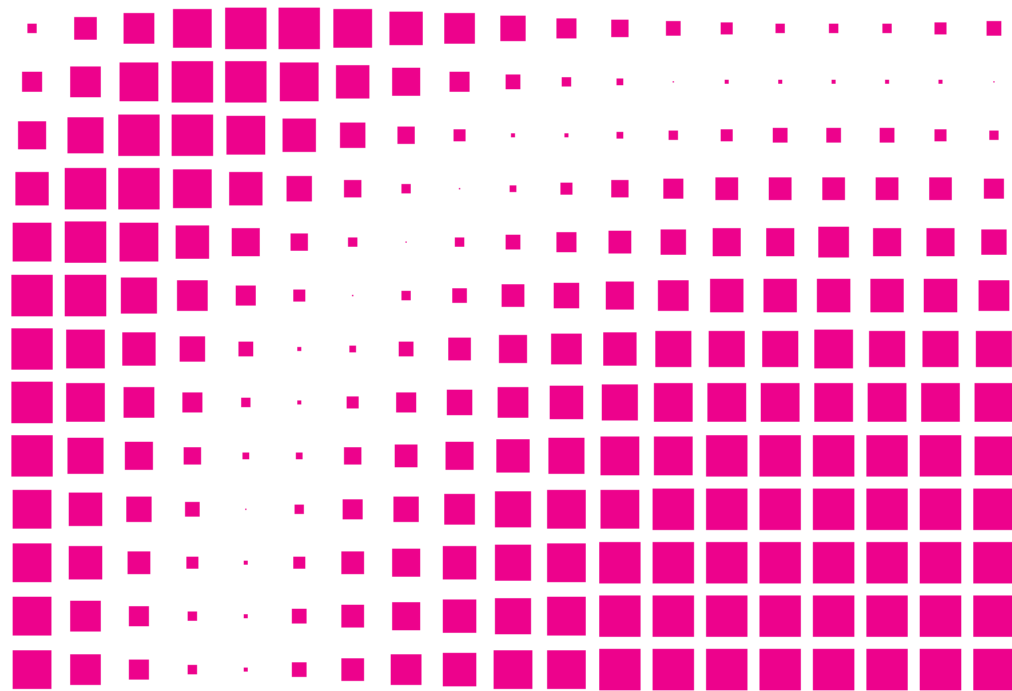


Gernot Hoffmann

Fast Fourier Transform

Descreening



Contents

1.	Introduction	2
2.	Mathematics	3
3.	Fourier Series	5
4.	FS Code	6
5.	DFT Example	8
6.	DFT Code	9
7.	FFT Example	13
8.	FFT Descreening	15
9.	Optimized Descreening	16
10.	FFT Code	18
12.	Gaussian Filters	24
12.	3D Relief	25
13.	References	26

1.1 Introduction

The discretization of the Fourier integrals leads to the Fourier Series. The first example shows the straightforward implementation of the Fourier Series.

The Discrete Fourier Transform should be considered as a special transformation which is loosely related to the Fourier Series. The DFT is always executed complex to complex in-place.

We are starting with a generic 1D- and 2D Discrete Fourier Transformation (DFT), including an example.

The Fast Fourier Transform FFT is an economically programmed DFT.

The algorithms are described in many text books, e.g. [1], [2]. The main subroutines FFT0 and FFT1 were taken from [1], with shorter names for the variables. Some helpful hints for the 2D-FFT and the complete C-code can be found in [3].

This documents contains only the generic version of the FFT. Image width and height in pixels are equal and always a power of two.

The essential procedures FFT0, FFT1 and FFT2 can be easily translated from Pascal into any other language. The program code is shown completely, but some parts depend on libraries in the background.

Further application examples demonstrate the descreening of rastered scans by Gaussian blur, by sharp edged low pass filters and by bundles of notch filters.

Settings for Acrobat

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

Custom Resolution 72dpi and use zoom 100% / 200%

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

sRGB

Euroscale Coated or ISO Coated or SWOP

Gray Gamma 2.2

2.1 Mathematics

Fourier Series

Time domain (t)

Sample interval $dt = T$ for $t=0$ to $(N-1)T$

Lowest frequency $f_1 = 1/(NT)$, $\omega_1 = 2\pi f_1$, $\omega_u = u\omega_1$

Even N $M = N/2$

Odd N $M = N \text{ div } 2$

$$a_u = 2/(NT) \sum_{n=0}^{N-1} f(nT) \cos(u\omega_1 nT)$$

$$b_u = 2/(NT) \sum_{n=0}^{N-1} f(nT) \sin(u\omega_1 nT)$$

$$f(nT) = a_0/2 + \sum_{u=1}^M [a_u \cos(u\omega_1 nT) + b_u \sin(u\omega_1 nT)]$$

Spatial domain (x)

Sample interval $dx = 1$ for $x=0$ to $N-1$

Lowest frequency $f_1 = 1/N$, $\omega_1 = 2\pi f_1$, $\omega_u = u\omega_1$

$$a_u = (2/N) \sum_{x=0}^{N-1} f(x) \cos(ux2\pi/N)$$

$$b_u = (2/N) \sum_{x=0}^{N-1} f(x) \sin(ux2\pi/N)$$

$$f(x) = a_0/2 + \sum_{u=1}^M [a_u \cos(ux2\pi/N) + b_u \sin(ux2\pi/N)]$$

Test example

$$f(x) = 1.0 + \sin(30 \cdot 2\pi x/N) + 0.2 \sin((N \text{ div } 2) 2\pi x/N) + \cos(80.5 \cdot 2\pi x/N)$$

The fourth component is added for $x > N/2$

The program uses (i) instead of (x)

2.2 Mathematics

1D-DFT Discrete Fourier Transform

Time domain (t)

Sample interval $dt = T$ for $t=0$ to $(N-1)T$

Lowest frequency $f_1 = 1/(NT)$, $\omega_1 = 2\pi f_1$, $\omega_u = u\omega_1$

$$F(u\omega_1) = (1/N) \sum_{n=0}^{N-1} f(nT) [\cos(u\omega_1 nT) - j\sin(u\omega_1 nT)]$$

$$f(nT) = \sum_{u=0}^{N-1} F(j\omega_u) [\cos(u\omega_1 nT) + j\sin(u\omega_1 nT)]$$

Spatial domain (x)

Sample interval $dx = 1$ for $x=0$ to $N-1$

Lowest frequency $f_1 = 1/N$, $\omega_1 = 2\pi/N$, $\omega_u = 2\pi u/N$

$$F(u) = (1/N) \sum_{x=0}^{N-1} f(x) [\cos(ux2\pi/N) - j\sin(ux2\pi/N)]$$

$$f(x) = \sum_{u=0}^{N-1} F(u) [\cos(ux2\pi/N) + j\sin(ux2\pi/N)]$$

$$\text{Re}_u = \text{Real}[F(j\omega_u)]$$

$$\text{Im}_u = \text{Imag}[F(j\omega_u)]$$

DC Value

$$A_0 = \sqrt{\text{Re}_0^2 + \text{Im}_0^2}$$

AC Values

$$A_u = 2 \sqrt{\text{Re}_u^2 + \text{Im}_u^2}$$

2D-DFT Discrete Fourier Transform

Spatial domain (x,y)

Sample interval $dx = dy = 1$ for $x=0$ to $N-1$ and $y = 0$ to $N-1$

Lowest frequency $f_1 = 1/N$, $\omega_1 = 2\pi/N$, $\omega_u = 2\pi u/N$

$\omega_u = u\omega_1$, $\omega_v = v\omega_1$, in x and y direction

The cosines and sines are replaced by the complex exponential

$F(u,v)$ is a complex number

DFT

$$F(u,v) = (1/N^2) \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} f(x,y) e^{-j(2\pi/N)(ux+vy)}$$

IDFT

$$f(x,y) = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} F(u,v) e^{+j(2\pi/N)(ux+vy)}$$

The factor $1/N^2$ can be assigned to the second equation or distributed as $1/N$ to both equations.

The above formulation defines F as an amplitude spectrum.

3.1 Fourier Series

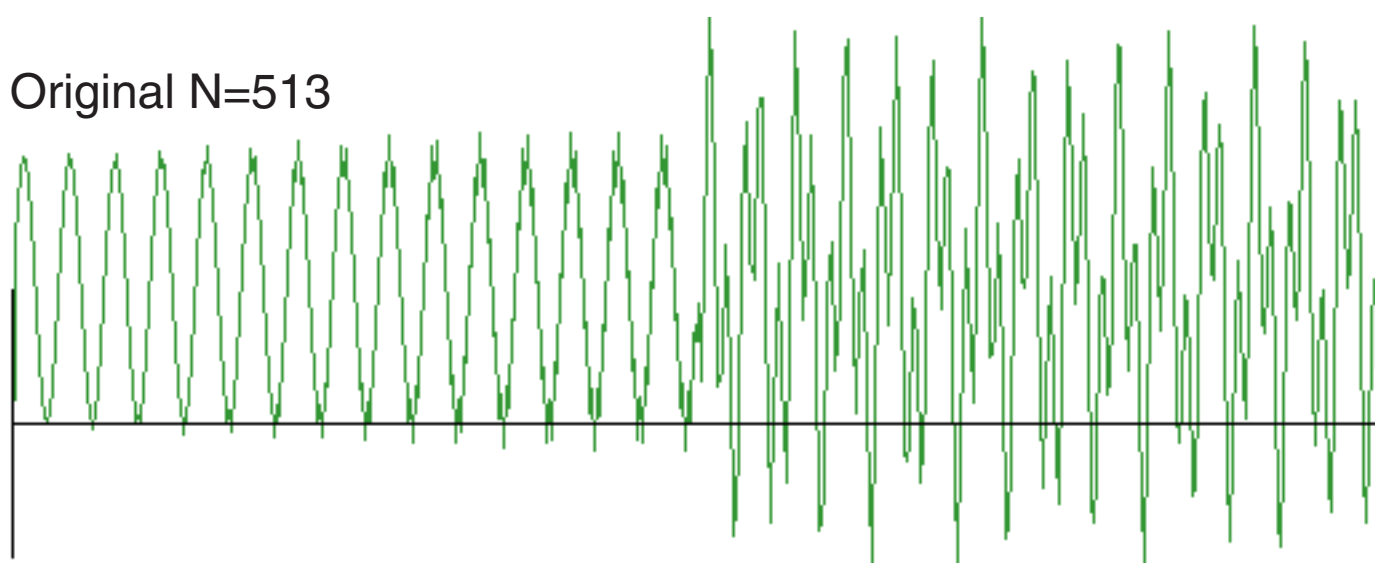
The example shows an ordinary onedimensional Fourier series FS.

Top is the original function, in the middle the amplitude spectrum and bottom the reconstruction by the inverse FS.

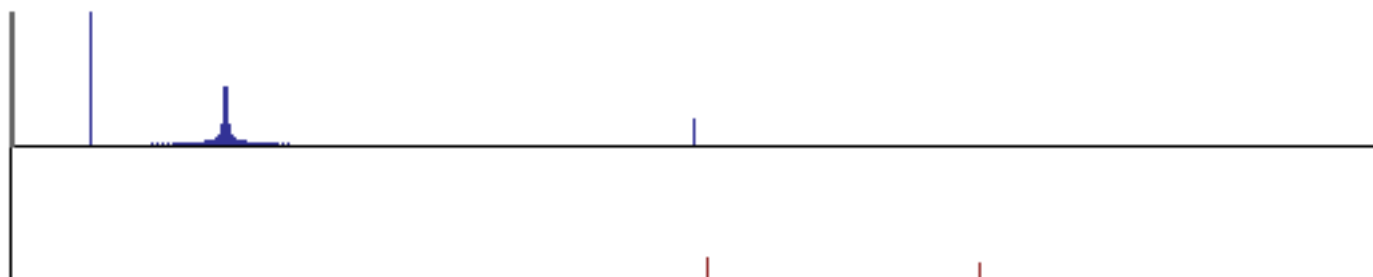
Lowest frequency	$f_1 = 1/N$
Nyquist frequency	$f_n = 1/2$
Frequency index	$u = 0 \dots (N \text{ div } 2)$
Period	$p = 1/f = N/u$
Frequency	$f = uf_1 = u/N = 1/p$

Nyquist frequency f_n is half the sampling frequency. N must be odd, because $N=2M+1$ parameters are used in the series. E.g. for $N=513$ the highest measurable frequency index is 256, but the Nyquist frequency is at 256.5. Leakage happens if (u) is not integer. For even N the highest measurable frequency is $N/2$, but sine components with this frequency are sampled as zeros. The peak in the middle is at $u=N \text{ div } 2=256$.

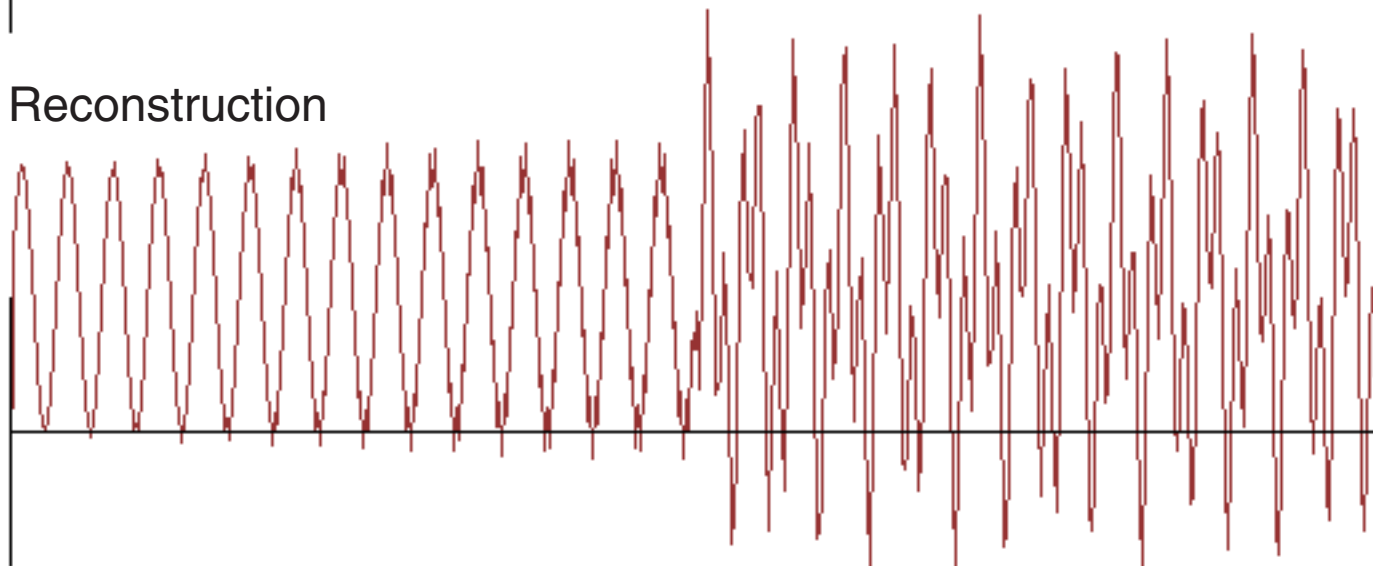
Best view zoom 100%



Amplitude Spectrum



Reconstruction



4.1 Fourier Code

```
Program XDFT02;
{   Project:   Real Fourier Series
  Author: G.Hoffmann
  Date:      June 27, 2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses      Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
          Zefir36,Zefir37,Zefir38,Zefir39;
Const     M=1025;
Type      TNarr=Array[0..M-1] of Single;
Var       Co,Si,Fi,Ak,Bk: TNarr;
Var       N,xa,ya,sc : Integer;
          Iml       : String;

Procedure TDFT (N: Integer; Var Co,Si: TNarr);
{ Sine+Cosine tables }
Var i,n1: Integer; pi2: Single;
Begin
n1:=N-1; pi2:=2*pi/N;
For i:=0 To n1 Do SicCoc(i*pi2,Si[i],Co[i]); { Fast Sine+Cosine }
End;

Procedure FDFT1 (N: Integer; Var Fi,Ak,Bk,Co,Si: TNarr);
{ Forward Fourier }
Var rt,it,dn,cc,ss: Single; i,k,ik,n1,n2 : LongInt;
Begin
n1:=N-1; n2:=N div 2; dn:=2/N;
For k:=0 to n2 Do
Begin
rt:=0; it:=0;
For i:=0 to n1 Do
Begin
ik:=(i*k) Mod N;
rt:=rt+Fi[i]*Co[ik];
it:=it+Fi[i]*Si[ik];
End; {i}
Ak[k]:=dn*rt; Bk[k]:=dn*it;
End; {k}
End;

Procedure IDFT1 (N: Integer; Var Fi,Ak,Bk,Co,Si: TNarr);
{ Inverse Fourier }
Var rt : Single; i,k,ik,n1,n2 : LongInt;
Begin
n1:=N-1; n2:=N div 2;
For i:=0 to n1 Do
Begin
rt:=0.5*Ak[0];
For k:=1 to n2 Do
Begin
ik:=(i*k) Mod N;
rt:=rt+Ak[k]*Co[ik]+Bk[k]*Si[ik];
End; {k}
Fi[i]:=rt;
End; {i}
End;

Procedure Fill (N:Integer; Var Fi: TNarr);
Var i,n2: Integer; pi2 : Single;
Begin
n2:=N div 2; pi2:=2*pi/N;
For i:=0 to N-1 Do
Begin
Fi[i]:=1.0+Sic(30*pi2*i) + 0.2*Sic(n2*pi2*i);
If i>n2 Then Fi[i]:=Fi[i]+Coc(80.5*pi2*i);
End;
End;
```

4.2 Fourier Code

```
Procedure DrawT (N: Integer; Fi,Ak,Bk: TNarr; xa,ya,sc,pal,col: Integer);
{ Draw function }
Var i,xo,yo,xn,yn,xe,ye: Integer;
Begin
xe:=xa+N; xo:=xa;
yo:=ya+Round(sc*Fi[0]);
For i:=1 to N-1 Do
  Begin
  xn:=xa+i;
  yn:=ya-Round(sc*Fi[i]);
  MakeSLine(xo,yo,xn,yn,pal,col);
  xo:=xn; yo:=yn;
  End;
MakeSLine(xa,ya,xe,ya, grsc,0);
MakeSLine(xa,ya+sc,xa,ya-sc,grsc,0);
End;

Procedure DrawS (N: Integer; Fi,Ak,Bk: TNarr; xa,ya,sc,pal,col: Integer);
{ Draw amplitude spectrum }
Var k,xn,yn,xe,ye,n2: Integer; s: Single;
Begin
xe:=xa+N;
n2:=N div 2;
For k:=1 to n2 Do
  Begin
  xn:=xa+k;
  s:=Sqrt(Sqr(Ak[k])+Sqr(Bk[k]));
  yn:=ya-Round(sc*s);
  MakeSLine(xn,ya,xn,yn,pal,col);
  End;
MakeSLine(xa,ya,xe,ya, grsc,0);
MakeSLine(xa,ya+sc,xa,ya-sc,grsc,0);
yn:=ya-Round(sc*0.5*Ak[0]);
MakeSLine(xa, ya,xa, yn,grsc,100);
MakeSLine(xa+1,ya,xa+1,yn,grsc,100);
End;

BEGIN
Configure
(ConfigMax,ZebraMax,Flag,Conf,ZebrArray,
ZebrPath,SmdrPath,
BuffDrNm,SnapDrNm,ZebrNm,
LastDriv,SourDriv,DestDriv,
SourceNr,GcardIdy,VesaMode,VesaCode,
SoundsOn,NoClkInt,AutoCatG,ShowIcon);
VesaStart(VesaMode);
Im1:='I:\fft__\fft__951.bmp';
ColToScr(grsc,255);
N:=513; { always odd }
TDFT (N,Co,Si);
Fill(N,Fi);
xa:=20; ya:=200; sc:=50;
DrawT(N,Fi,Ak,Bk,xa,ya,sc,60,100);
FDFT1 (N,Fi,Ak,Bk,Co,Si);
ya:=ya+170;
DrawS(N,Fi,Ak,Bk,xa,ya,sc,120,100);
IDFT1 (N,Fi,Ak,Bk,Co,Si);
ya:=ya+200;
DrawT(N,Fi,Ak,Bk,xa,ya,sc,0,100);
SaveImag(im1);
Stop;
VesaEnde;
ClrScr;
END.
```

5.1 DFT Example

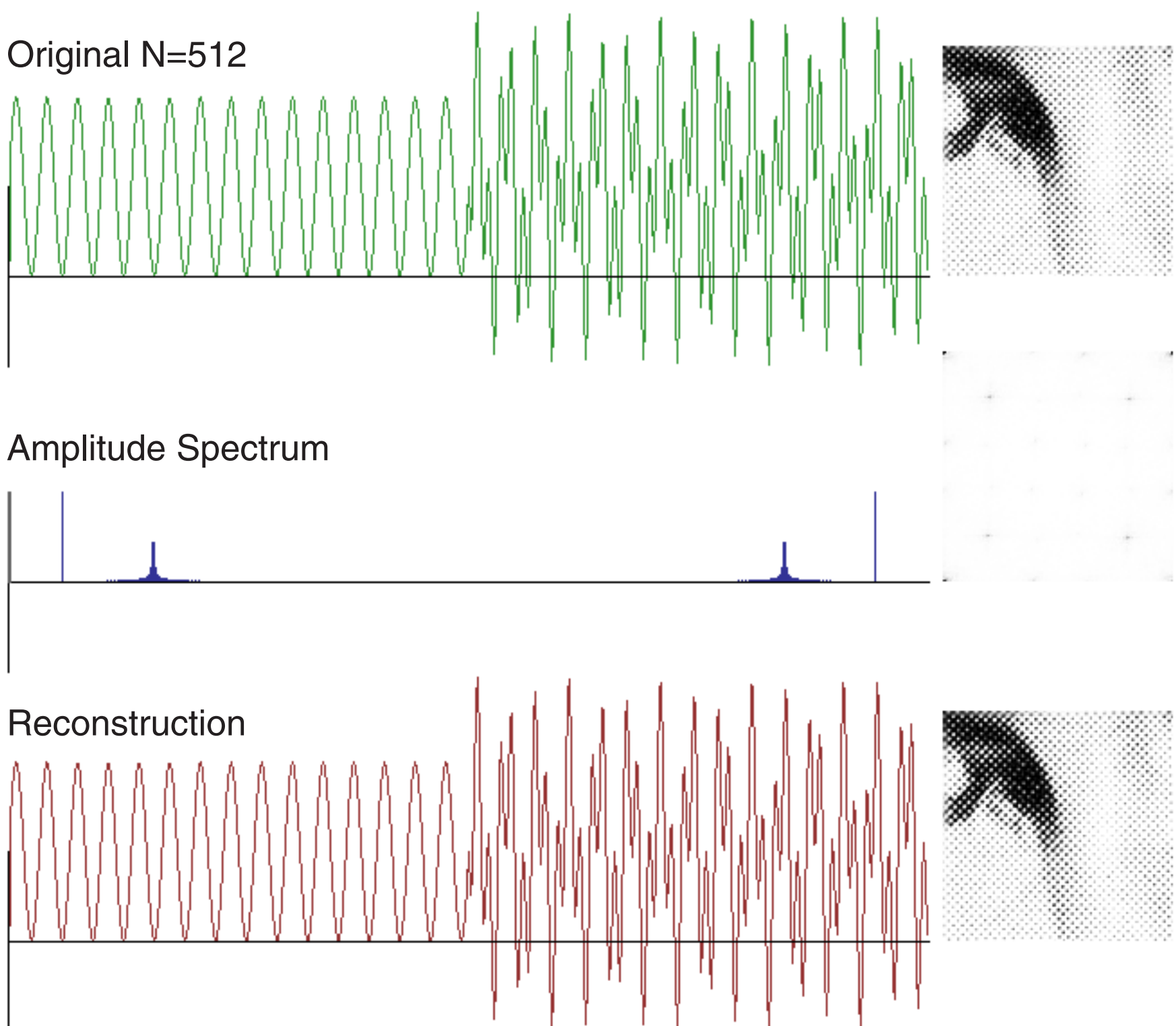
The example shows left an 1D-DFT and right a 2D-DFT.

Top is the original function, in the middle the amplitude spectrum and bottom the reconstruction by the inverse DFT.

Lowest frequency	$f_1 = 1/N$
Nyquist frequency	$f_n = 1/2$
Frequency index	$u = 0 \dots (N \text{ div } 2)$
Period	$p = 1/f = N/u$
Frequency	$f = uf_1 = u/N = 1/p$

Nyquist frequency f_n is half the sampling frequency. N is mostly even, because this is necessary for the FFT, though not for the DFT. E.g. for $N=512$ the highest measurable frequency index is 256, the same for the Nyquist frequency. But sine components with this frequency are sampled as zeros.

Best view zoom 100%



6.1 DFT Code

```
Program ZDFT02;
{   Project:   DFT 2D
  Author:     G.Hoffmann
  Date:      June 27, 2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses      Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
          Zefir36,Zefir37,Zefir38,Zefir39;
Const     M=1024;
Type      Cplx=Array[0..M-1] of Single;
Var       Co,Si,Re,Im,Am: Cplx;
Var       Dir,xa,ya,sc   : Integer;
          N,N1,P         : Integer;
          vis,flag,sel   : Integer;
          Im0,Im1        : String;
{ Complex Array KMem and Real Arrays PMem,QMem are global }

Procedure DFT0 (N: Integer; Var Co,Si: Cplx);
{ Sine+Cosine tables }
Var i,n1: Integer; pi2: Single;
Begin
n1:=N-1; pi2:=2*pi/N;
For i:=0 To n1 Do SicCoc(i*pi2,Si[i],Co[i]); { Fast Sine+Cosine }
End;

Procedure DFT1 (N: Integer; Var Re,Im,Co,Si: Cplx; Dir: Integer);
{ 1D-DFT, in-place }
Var rt,it,rp,ip,cc,ss: Single;
    i,j,ij,n1         : LongInt;
    Ar,Ai              : Cplx;
Begin
n1:=N-1;
For i:=0 to n1 Do
Begin
rt:=0; it:=0;
For j:=0 to n1 Do
Begin
ij:=(i*j) Mod N; { modulo=remainder }
cc:=Co[ij]; ss:=Si[ij]*Dir;
rp:=Re[j]; ip:=Im[j];
rt:=rt+rp*cc+ip*ss; { complex mult }
it:=it+ip*cc-rp*ss;
End; {j}
Ar[i]:=rt; Ai[i]:=it;
End; {i}
If Dir=+1 Then
Begin
For i:=0 to n1 Do
Begin
Re[i]:=Ar[i]/N; Im[i]:=Ai[i]/N;
{ 1D-Test only: } Am[i]:=2*Sqrt(Sqr(Re[i])+Sqr(Im[i]));
End;
{ 1D-Test only: } Am[0]:=0.5*Am[0];
End;
If Dir=-1 Then
For i:=0 to n1 Do
Begin
Re[i]:=Ar[i]; Im[i]:=Ai[i];
End;
End;
End;
```

6.2 DFT Code

```
Procedure DFT2 (N,Dir: Integer);
{ In-place 2D-DFT in complex KMem[i]^[j]=KMem[i,j] }
Var i,j,n1: Integer;
Begin
n1:=N-1;
{ DFT for all rows }
For i:=0 to n1 Do
Begin
For j:=0 To n1 Do
Begin
With KMem[i]^[j] Do Begin Re[j]:=Real; Im[j]:=Imag; End;
End;
DFT1(N,Re,Im,Co,Si,Dir);
For j:=0 to n1 Do
Begin
With KMem[i]^[j] Do Begin Real:=Re[j]; Imag:=Im[j]; End;
End;
End;
{ DFT for the columns, using the new rows }
For j:=0 to n1 Do
Begin
For i:=0 To n1 Do
Begin
With KMem[i]^[j] Do Begin Re[i]:=Real; Im[i]:=Imag; End;
End;
DFT1(N,Re,Im,Co,Si,Dir);
For i:=0 To n1 Do
Begin
With KMem[i]^[j] Do Begin Real:=Re[i]; Imag:=Im[i]; End;
End;
End;
End;
End;
```

```
Procedure Copy(Dir: Integer);
{ Copy Image P to Array or Array to Image }
Var i,j,n1,d: Integer; c: LongInt;
Begin
n1:=N-1;
If Dir=+1 Then
Begin
c:=255;
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
With KMem[i]^[j] Do
Begin Real:=PMem[i]^[j] And c; Imag:=0; End;
End;
End;
If Dir=-1 Then
Begin
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
With KMem[i]^[j] Do d:=Round(Sqrt(Sqr(Real)+Sqr(Imag)));
{If d<0 Then d:=0 Else If d>255 Then d:=255; }
c:=LimTab^[d];
QMem[i]^[j]:= c SHL 16 + c SHL 8 + c; { Format 0ccc }
End;
End;
End;
End;
```

6.3 DFT Code

```
Procedure MakeS;  
{ Make Image of Spectrum  
  Images are stored Longint argb  
  Gray images r=g=b=c Longint 0ccc }  
Var i,j,n0,n1,n2,ns: Integer;  
    c                : LongInt;  
    max,s,x,y        : Single;  
Begin  
n1:=N-1; n2:=N div 2; n0:=n2-1;  
{ Autoscale, exclude DC value }  
ns:=0;  
max:=0;  
With KMem[ns]^ [ns] Do  
Begin x:=Real; y:=Imag; Real:=0; Imag:=0; End;  
For i:=0 to n1 Do  
For j:=0 to n1 Do  
Begin  
  With KMem[i]^ [j] Do s:=Sqr(Real)+Sqr(Imag);  
  If s>max Then max:=s;  
End;  
max:=1/Sqrt(max);  
With KMem[ns]^ [ns] Do  
  Begin Real:=x; Imag:=y; End;  
For i:=0 to n1 Do  
For j:=0 to n1 Do  
Begin  
With KMem[i]^ [j] Do s:=max*Sqr(Sqr(Real)+Sqr(Imag));  
{c:=Round(255*exp(0.4545*ln(s))); Gamma }  
  c:=GamTab^[Round(255*(1-s))];  
  QMem[i]^ [j]:=c SHL 16 + c SHL 8 + c; { Format 0ccc }  
End;  
End;  
  
Procedure Fill (N:Integer);  
Var i,n2: Integer; pi2 : Single;  
Begin  
n2:=N div 2; pi2:=2*pi/N;  
For i:=0 to N-1 Do  
  Begin  
    Re[i]:=1.0+Sic(30*pi2*i) + 0.2*Sic(n2*pi2*i);  
    If i>n2 Then Re[i]:=Re[i]+Coc(80.5*pi2*i);  
    Im[i]:=0;  
  End;  
End;  
  
Procedure DrawT(N,pal,col: Integer);  
{ Draw function }  
Var i,xo,yo,xn,yn,xe,ye: Integer;  
Begin  
xe:=xa+N; xo:=xa;  
yo:=ya+Round(sc*Re[0]);  
For i:=1 to N-1 Do  
  Begin  
    xn:=xa+i;  
    yn:=ya-Round(sc*Re[i]);  
    MakeSLine(xo,yo,xn,yn,pal,col);  
    xo:=xn; yo:=yn;  
  End;  
MakeSLine(xa,ya,xe,ya, grsc,0);  
MakeSLine(xa,ya+sc,xa,ya-sc,grsc,0);  
End;
```

6.4 DFT Code

```

Procedure DrawS(N,pal,col: Integer);
{ Draw amplitude spectrum }
Var i,xn,yn,xe,ye: Integer;
Begin
xe:=xa+N;
For i:=0 to N-1 Do
  Begin
  xn:=xa+i;
  yn:=ya-Round(sc*Am[i]);
  MakeSLine(xn,ya,xn,yn,pal,col);
  End;
MakeSLine(xa,ya,xe,ya,grsc,0);
MakeSLine(xa,ya+sc,xa,ya-sc,grsc,0);
yn:=ya-Round(sc*Am[0]);
MakeSLine(xa,ya,xa,yn,grsc,100);
MakeSLine(xa+1,ya,xa+1,yn,grsc,100);
End;
{Pend}

BEGIN
Configure
(ConfigMax,ZebraMax,Flag,Conf,ZebrArray,
ZebrPath,SmdrPath,
BuffDrNm,SnapDrNm,ZebrNm,
LastDriv,SourDriv,DestDriv,
SourceNr,GcardIdy,VesaMode,VesaCode,
SoundsOn,NoClkInt,AutoCatG,ShowIcon);
MemKStart; { KMem,PMem,QMem }
VesaStart(VesaMode);
Im0:='I:\fft_\fft__900.bmp';
Im1:='I:\fft_\fft__921.bmp';
LimFill(LimTab); { Clipping Table }
GamFill(GamTab,1/2.2); { Gamma Tab.,clipping }
ColToScr(grsc,255); { Background }
N:=512; { 1D-DFT Test }
DFT0(N,Co,Si);
Fill(N);
xa:=20; ya:=200; sc:=50;
DrawT(N,60,100);
DFT1(N,Re,Im,Co,Si,+1);
ya:=ya+170;
DrawS(N,120,100);
DFT1(N,Re,Im,Co,Si,-1);
ya:=ya+200;
DrawT(N,0,100);
N:=128; { 2D-DFT Test }
N1:=N-1;
DFT0(N,Co,Si); { Sine/Cosine }
vis:=0;
ReadImag(Im0,'P',vis,flag); { Store Image in PMem }
PMemGrYIQ(0,0,N1,N1,0,0,2); { Convert to Gray }
PMemToSx(0,0,N1,N1,540,72); { Show }
Copy(+1); { Copy Image to Array }
DFT2(N,+1); { Array to Spectrum }
MakeS; { Draw Sectrum }
QMemToSx(0,0,N1,N1,540,242); { Show }
DFT2(N,-1); { Spectrum to Array }
Copy(-1); { Array to Image }
QMemToSx(0,0,N1,N1,540,442); { Show }
SaveImag(Im1); { Save page }
Stop;
VesaEnde;
MemKEnde;
ClrScr;
END.

```

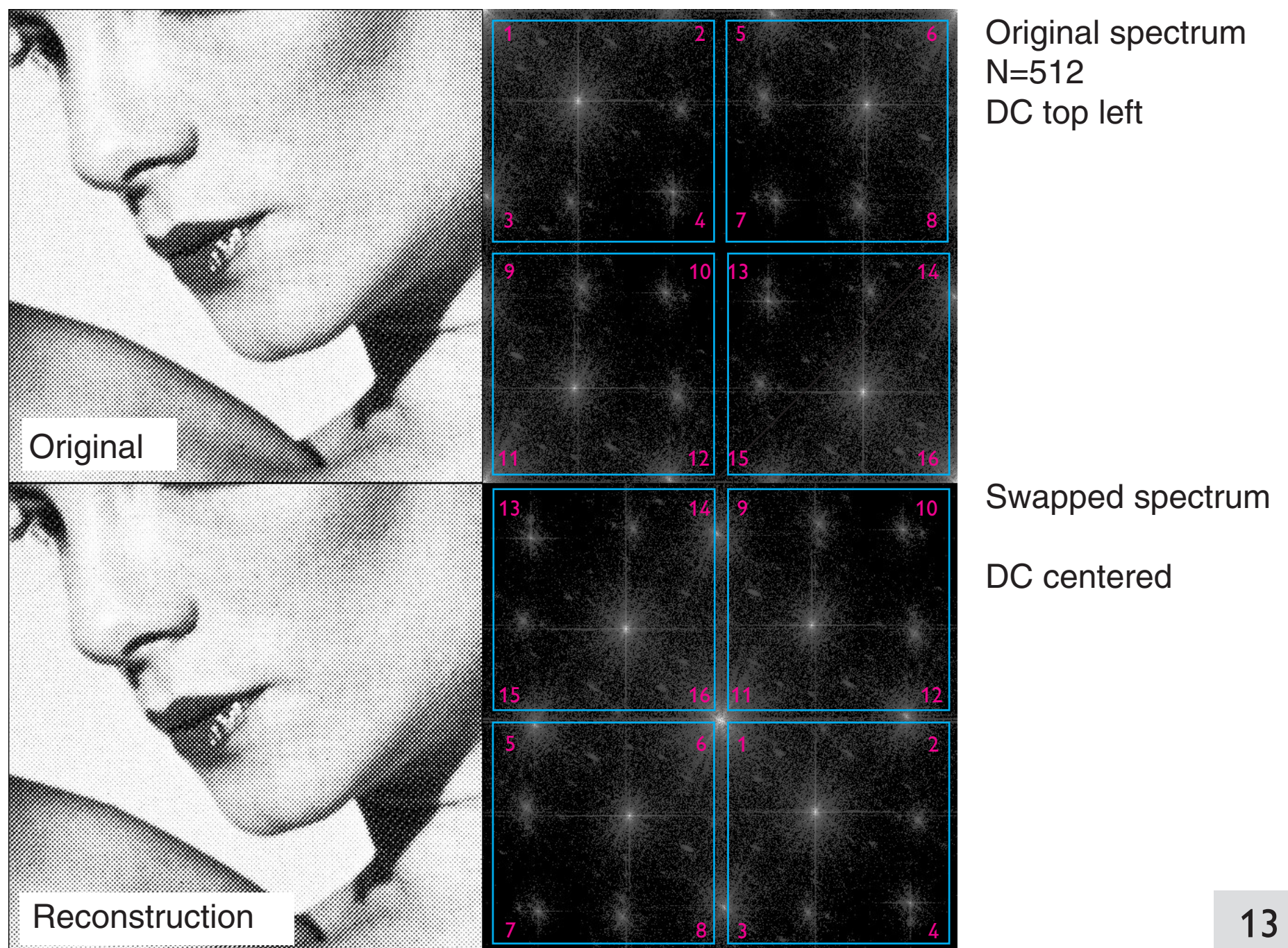
7.1 FFT Example

The gray values of an image are written into the real part of a complex array. The imaginary part is zero. The 2D-FFT starts by 1D-FFTs in rows. The new values in the rows are taken as inputs for 1D-FFTs in columns.

The spectrum is swapped for better visualization and for filters. The DC value is top left in the original spectrum and centered in the swapped spectrum.

Spectra gray values are enhanced by a strong inverse gamma correction, exponent 0.25.

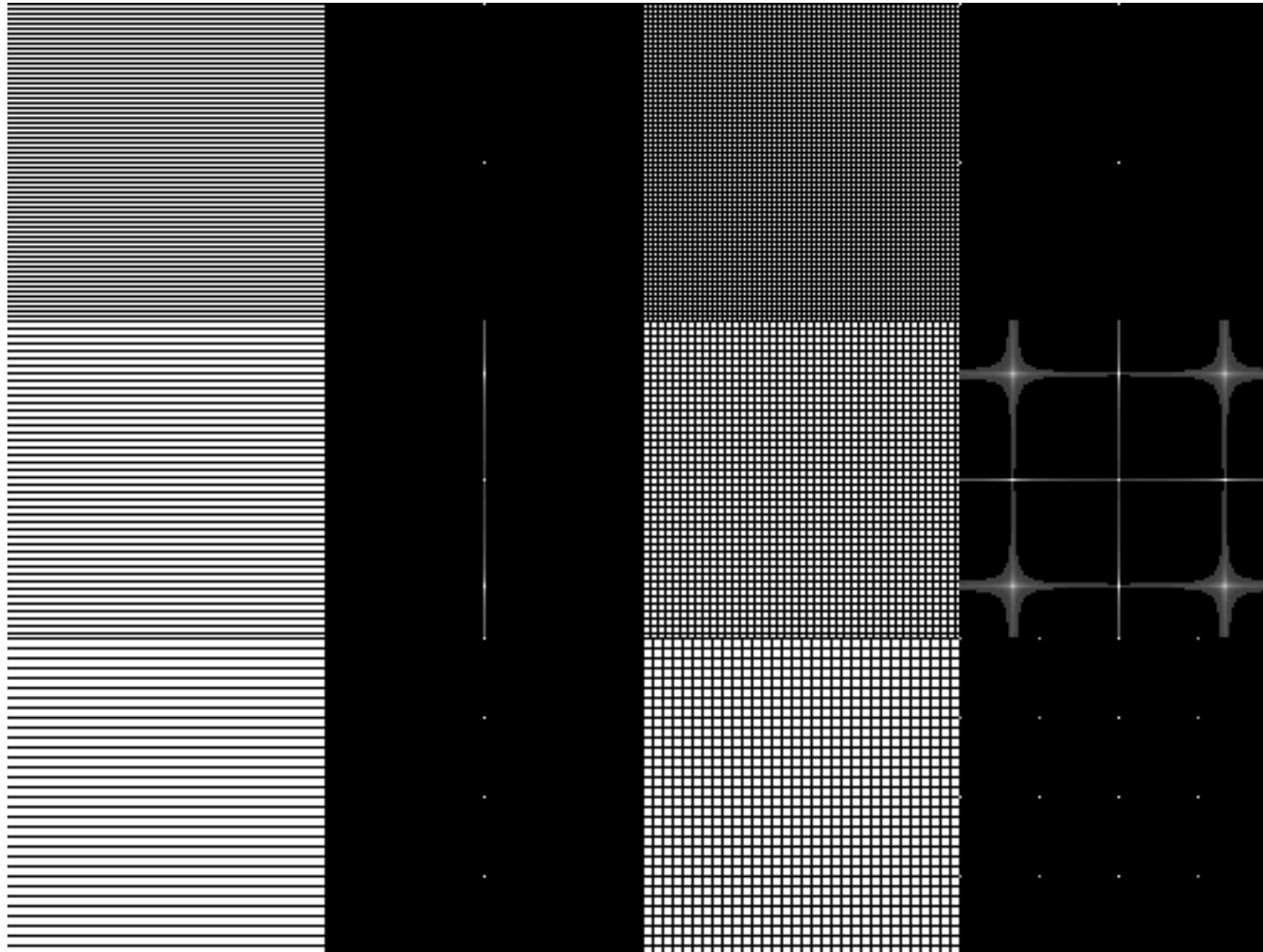
Best view zoom 200%



7.2 FFT Example

This page shows the swapped spectra of some regular line and grid patterns.

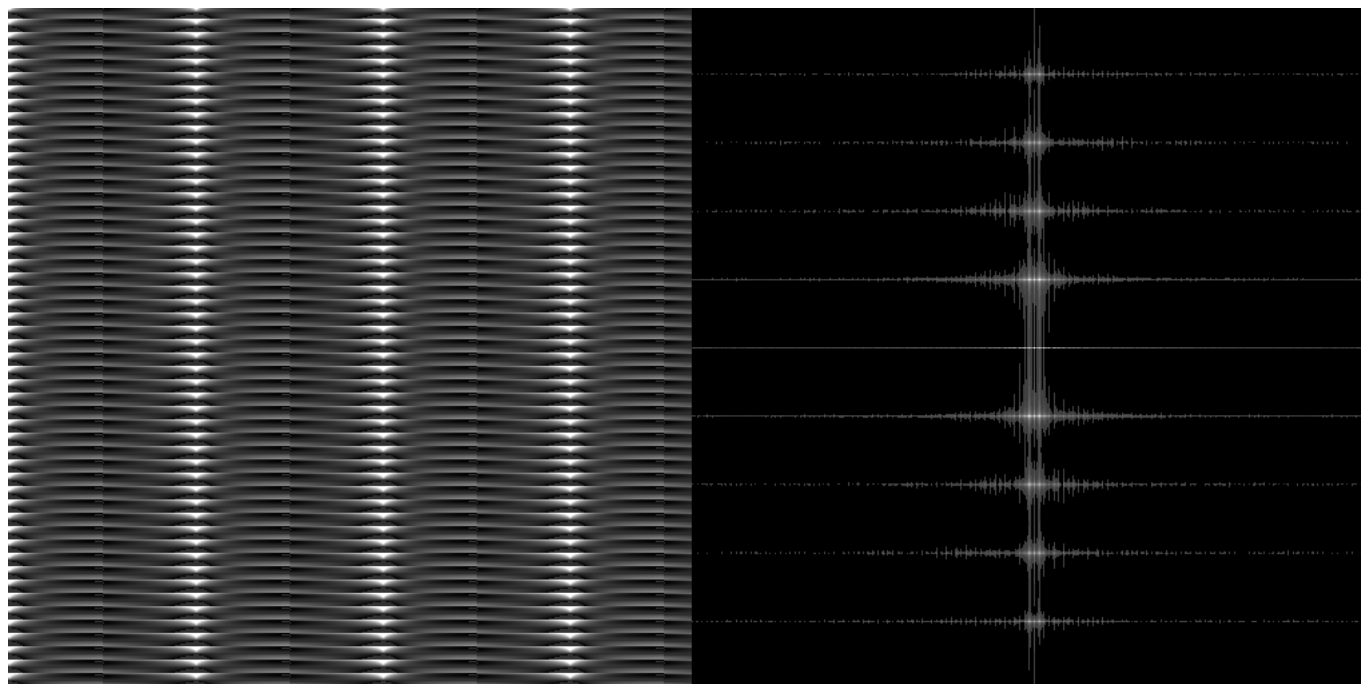
Best view zoom 200%



1 line,1 gap

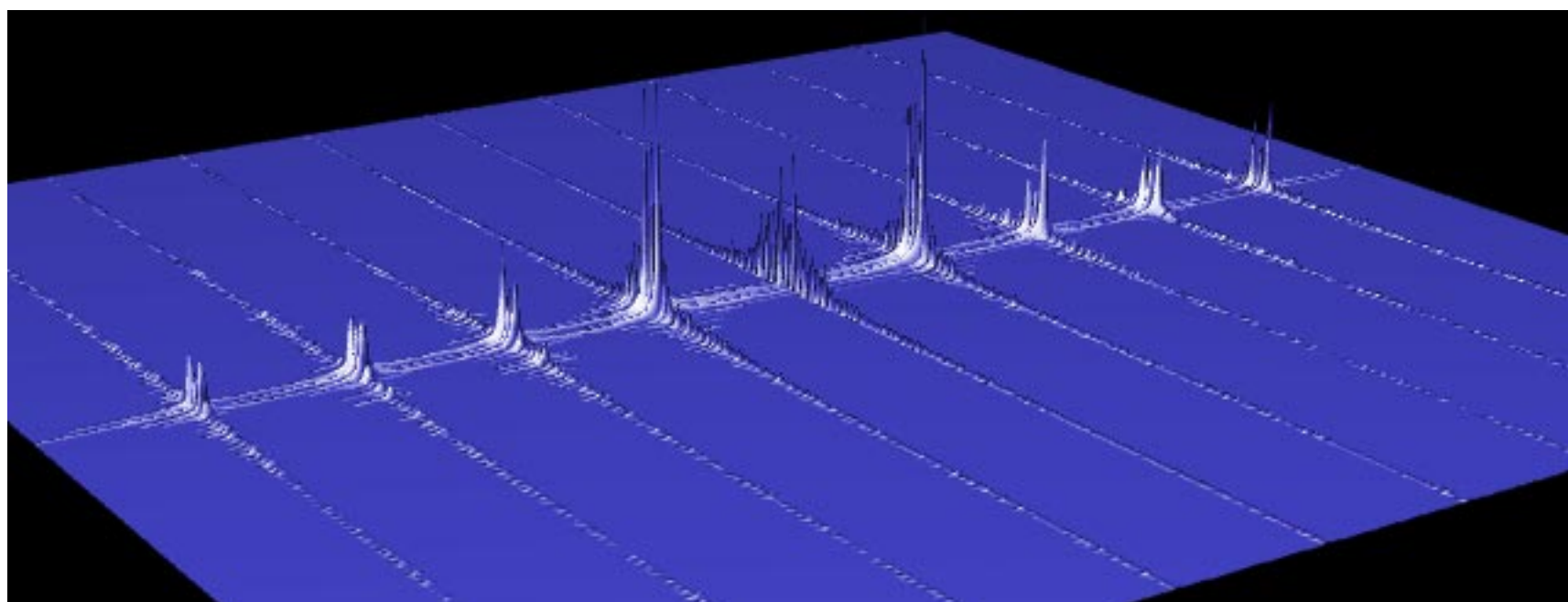
1 line,2 gaps
Leakage

1 line,3 gaps



Pattern

Shows
leakage



8.1 FFT Descreening

Scanning of already rastered prints can result in Moiré artifacts. Several solutions are available. Methods 2 to 5 are based on high resolution scanning.

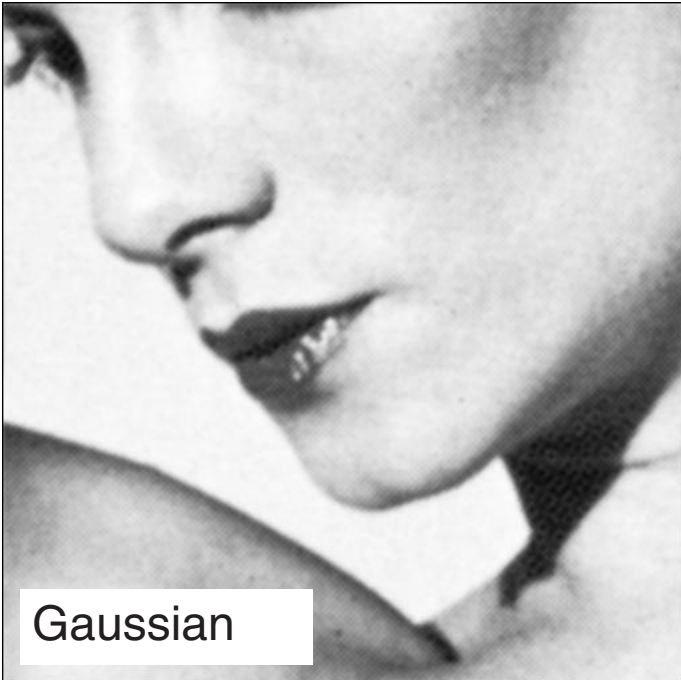
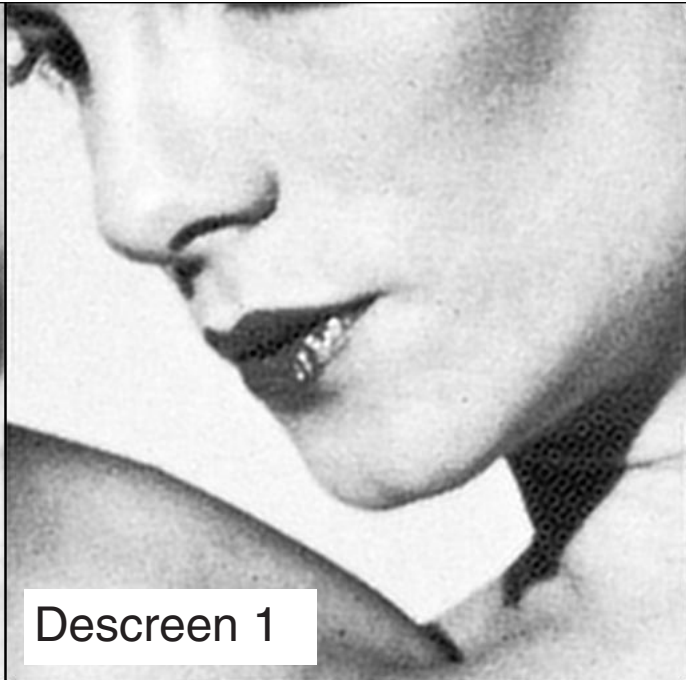
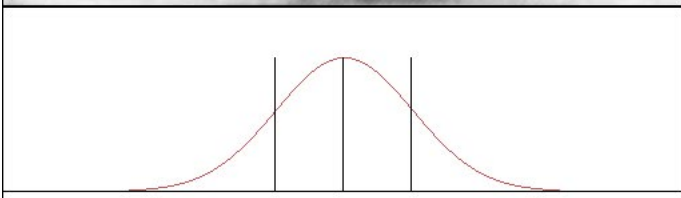

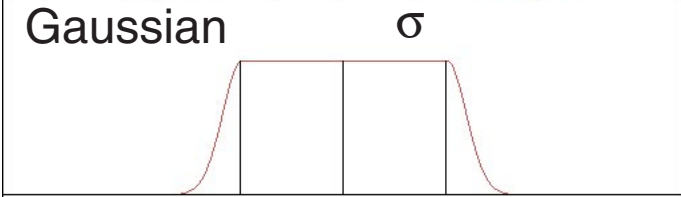
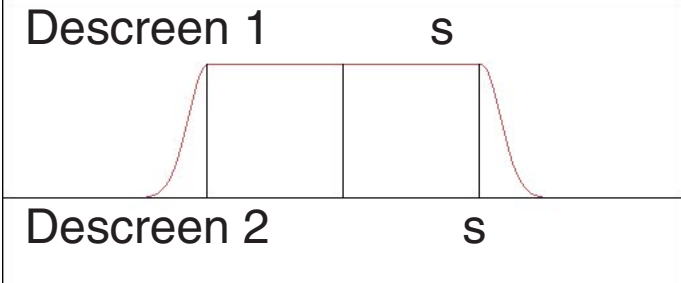
1. Synchronized scanning [4], applicable only to gray images.
2. Gaussian blur in the spatial domain.
3. Gaussian blur in the frequency domain (multiply by Gaussian weight factors).
4. Descreening by sharp edged low pass filter in the frequency domain.
5. Descreening by bundles of notch filters in the frequency domain.

Method 2 is quite common. The methods 3 and 4 are illustrated here.

Descreening by sharp edged low pass filters tends to sharper reconstructions, but there are most likely some artifacts.

An experimental version of Method 5 is shown in the next chapter.

Best view zoom 200%

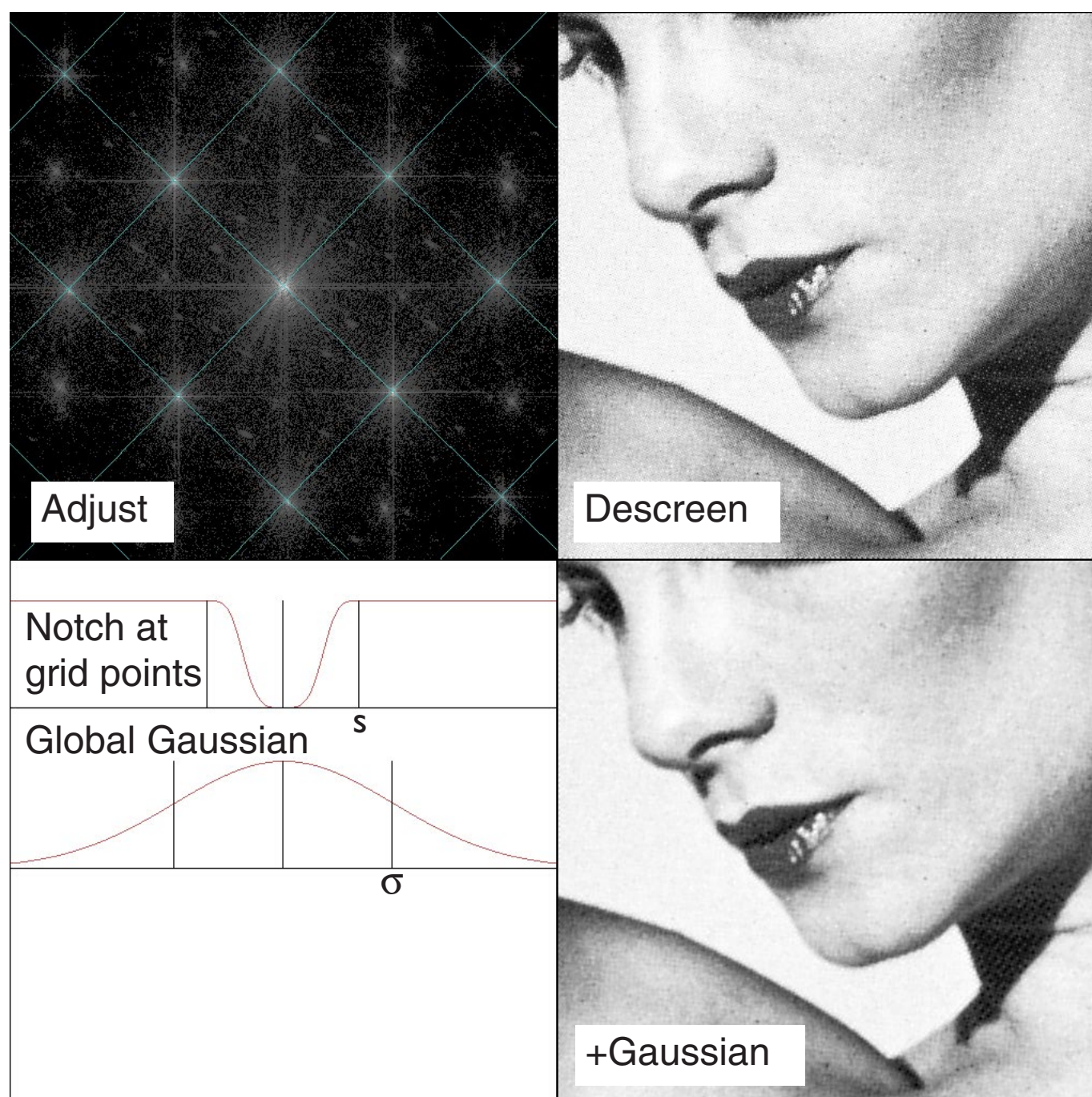
 <p>Gaussian</p>	 <p>Descreen 1</p>	<p>Width 512 pixels</p> <p>Gaussian Standard Deviation $s = 0.2 \cdot 256 = 51.2$</p>
 <p>Gaussian σ</p>	 <p>Descreen 2</p>	<p>Descreen 1-2 Spectrum zero outside radius s</p> <p>1 $s = 0.3 \cdot 256 = 77$ 2 $s = 0.4 \cdot 256 = 102$</p>
 <p>Descreen 1 s</p>		
 <p>Descreen 2 s</p>		

9.1 Optimized Descreening

A grid is shown in the frequency plot. The angle and the grid width can be adjusted manually until the grid points are at the high frequency peak positions.

Spectra gray values are enhanced by a strong inverse gamma correction, exponent 0.25. At each grid point (but not for the low frequency center point) a notch filter is applied. The real part and the imaginary part of the spectrum are multiplied by the filter weight factors. This doesn't remove all high frequency components. Because of leakage the peaks in the frequency domain are by no means simple shapes with round cross sections. Therefore a weak Gaussian filter is applied additionally.

Best view zoom 200%



Width 512 pixels

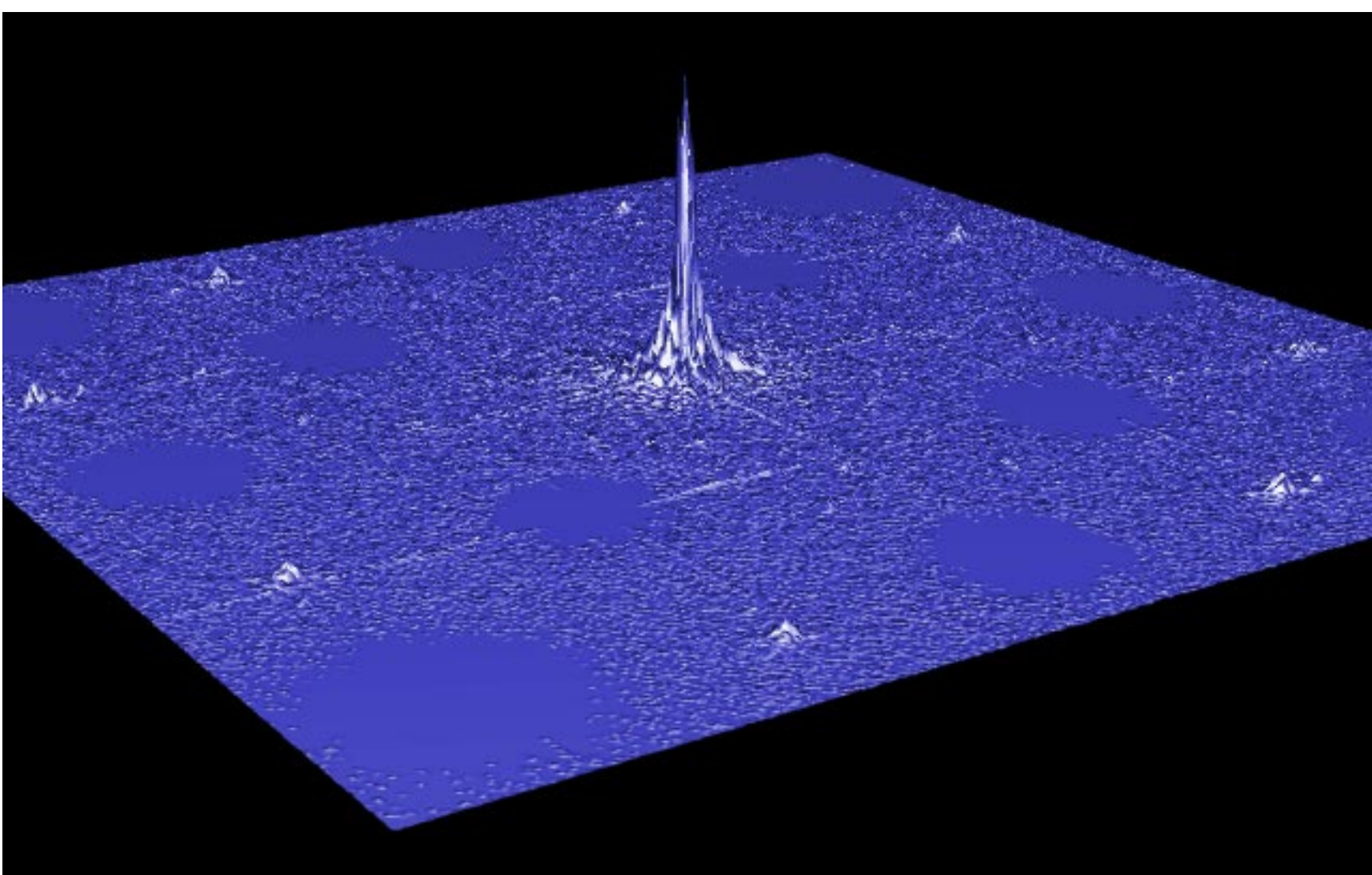
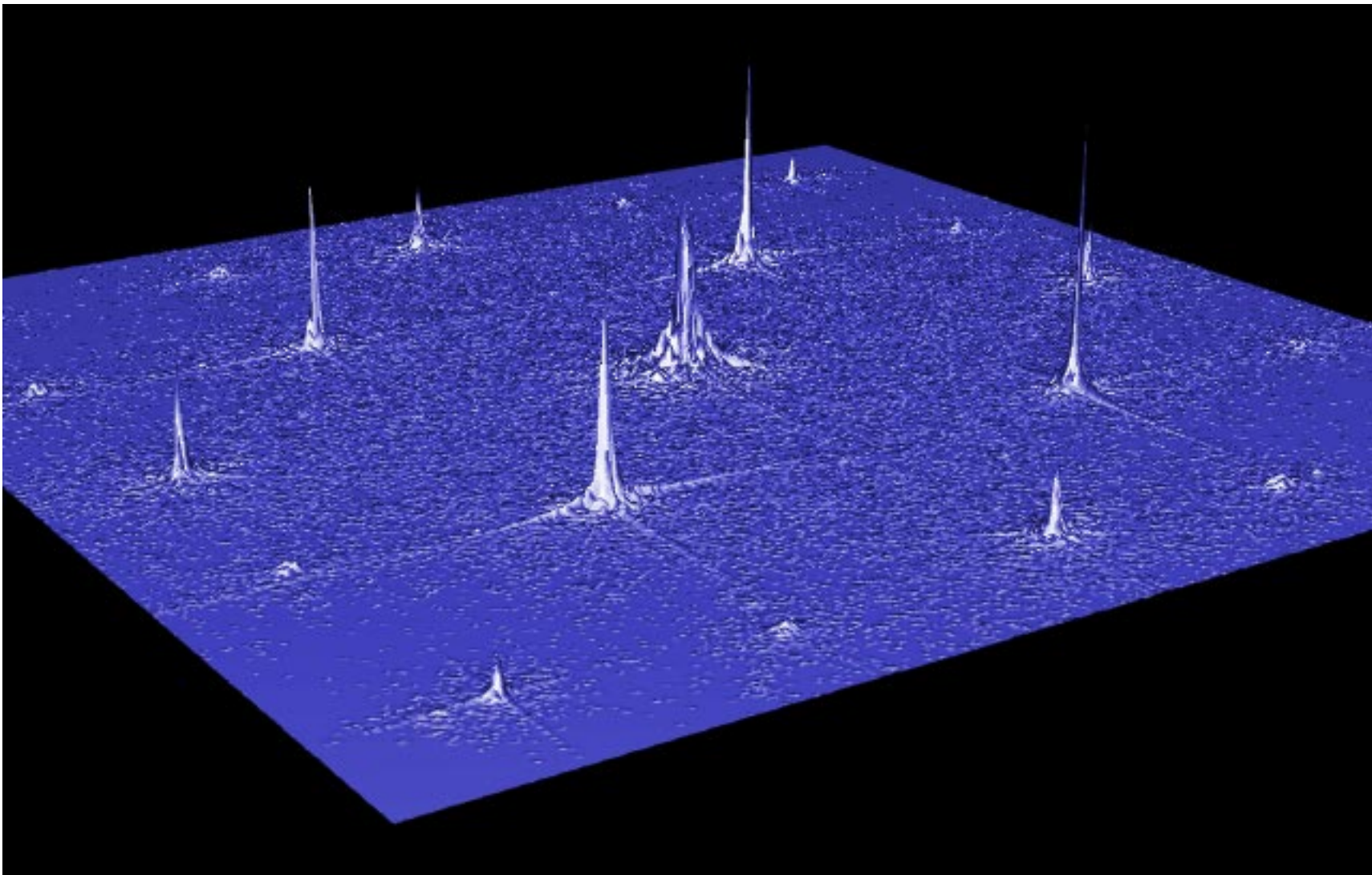
Adjustable angle and gridwidth
Notch filter at grid points
 $s = 0.5$ gridwidth

Additionally a global weak Gaussian filter

9.2 Optimized Descreening

Original amplitude spectrum and spectrum after notch filtering. Spectrum height maps are linear. Different scales because of autoscaling.

Best view zoom 100%



10.1 FFT Code

```
Program ZFFT02;
{   Project:      FFT 2D
  Author:        G.Hoffmann
  Date:          June 18, 2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses   Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
       Zefir36,Zefir37,Zefir38,Zefir39;
Const  M=1024;
Type   Cplx=Array[0..M-1] of Single;
Var     Co,Si,Re,Im : Cplx;
Var     Dir,xa,ya,sc : Integer;
       N,N1,P      : Integer;
       vis,flag,sel : Integer;
       Im0,Im1,Im2 : String;
       GauTab: Array [0..M div 2] Of Single;
{ Complex Array KMem.Real,KMem.Imag and Arrays PMem,QMem are global }
{Pini}
```

```
Procedure FFT0 (Var Co,Si: Cplx; Var N: Integer);
{ Twiddle Factors }
Var i,h,z,t: Integer; b: Single;
Begin
P:=Round(ln(N)/ln(2)); { 10 for 1024 }
z:=0;
For i:=0 To p-1 Do
  Begin
  t:=1 SHL i; b:=pi/t;
  For h:=1 To t Do
    Begin
    SicCoc(b*(1-h),Si[z],Co[z]); { Fast Sine+Cosine }
    Inc(z);
    End;
  End;
End;
```

```
Procedure FFT1 (Var Re,Im,Co,Si: Cplx; N: Integer; Dir: Integer);
Var Ret,Imt,Reh,Imh,Buf,ni: Single;
    P,e1,e2,i,h,z,s,bi,a,b: Integer;
Begin
P:=Round(ln(N)/ln(2));
For e1:=1 To N-1 Do
  Begin
  e2:=0;
  For bi:=0 To P-1 Do e2:=2*e2+((e1 SHR bi) AND 1);
  If e2>e1 Then
    Begin
    Buf:=Re[e1]; Re[e1]:=Re[e2]; Re[e2]:=Buf;
    Buf:=Im[e1]; Im[e1]:=Im[e2]; Im[e2]:=Buf;
    End;
  End;
z:=0;
For s:=0 To P-1 Do
  Begin
  a:=1 SHL s; b:=2*a;
  For h:=0 To a-1 Do
    Begin
    Ret:=Co[z]; Imt:=Si[z]*Dir;
    Inc(z); e1:=h;
    For i:=0 To (N div b) -1 Do
      Begin
      e2:=e1+a;
      Reh:=Ret*Re[e2]-Imt*Im[e2];
      Imh:=Ret*Im[e2]+Imt*Re[e2];
```

10.2 FFT Code

```
    Re[e2]:=Re[e1]-Reh; Im[e2]:=Im[e1]-Imh;
    Re[e1]:=Re[e1]+Reh; Im[e1]:=Im[e1]+Imh;
    Inc(e1,b);
    End; {i}
    End; {h}
End; {s}
If Dir=+1 Then
Begin
    ni:=1/N;
    For i:=0 to N-1 Do
        Begin
            Re[i]:=Re[i]*ni; Im[i]:=Im[i]*ni;
        End;
    End;
End;
```

```
Procedure FFT2(Dir: Integer);
{ In-place 2D-FFT in complex KMem[i]^[j]=KMem[i,j] }
Var i,j,n1: Integer;
Begin
    n1:=N-1;
    { FFT for all rows }
    For i:=0 to n1 Do
    Begin
        For j:=0 To n1 Do
        Begin
            With KMem[i]^[j] Do
                Begin Re[j]:=Real; Im[j]:=Imag; End;
            End;
        FFT1(Re,Im,Co,Si,N,Dir);
        For j:=0 to n1 Do
        Begin
            With KMem[i]^[j] Do
                Begin Real:=Re[j]; Imag:=Im[j]; End;
            End;
        End;
    End;
    { FFT for the columns, using the new rows }
    For j:=0 to n1 Do
    Begin
        For i:=0 To n1 Do
        Begin
            With KMem[i]^[j] Do
                Begin Re[i]:=Real; Im[i]:=Imag; End;
            End;
        FFT1(Re,Im,Co,Si,N,Dir);
        For i:=0 To n1 Do
        Begin
            With KMem[i]^[j] Do
                Begin Real:=Re[i]; Imag:=Im[i]; End;
            End;
        End;
    End;
End;
```

```
Procedure MakeS(swap: Boolean);
{ Make Image of Spectrum
  Images are stored Longint argb
  Gray images r=g=b=c Longint 0ccc }
Var i,j,n0,n1,n2,ns : Integer;
    c                : LongInt;
    max,s,x,y        : Single;
Begin
    n1:=N-1; n2:=N div 2; n0:=n2-1;
    { Autoscale, exclude DC value }
    If swap Then ns:=n2 Else ns:=0;
    max:=0;
    With KMem[ns]^[ns] Do
    Begin x:=Real; y:=Imag; Real:=0; Imag:=0; End;
```

10.3 FFT Code

```
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
  With KMem[i]^ [j] Do s:=Sqr(Real)+Sqr(Imag);
  If s>max Then max:=s;
End;
max:=1/Sqrt(max);
With KMem[ns]^ [ns] Do
  Begin Real:=x; Imag:=y; End;
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
With KMem[i]^ [j] Do s:=max*Sqrt(Sqr(Real)+Sqr(Imag));
{c:=Round(255*exp(0.4545*ln(s))); Gamma }
  c:=GamTab^[Round(255*s)];
  QMem[i]^ [j]:=c SHL 16 + c SHL 8 + c; { Format 0ccc }
End;
End;

Procedure Swap;
{ Swap Spectrum; i: row; j: column }
Var i,j,n0,n1,n2,ni,nj: Integer; s: Single;
Begin
n1:=N-1;
n2:=N div 2;
n0:=n2-1;
n0:=N div 2-1;
For i:=0 to n0 Do
Begin
ni:=n2+i;
  For j:=0 to n0 Do
  Begin
  nj:=n2+j;
    s:= KMem[i ]^ [j ].Real;
    KMem[i ]^ [j ].Real:=KMem[ni]^ [nj].Real;
    KMem[ni]^ [nj].Real:=s;
    s:= KMem[i ]^ [j ].Imag;
    KMem[i ]^ [j ].Imag:=KMem[ni]^ [nj].Imag;
    KMem[ni]^ [nj].Imag:=s;
    s:= KMem[i ]^ [nj].Real;
    KMem[i ]^ [nj].Real:=KMem[ni]^ [j].Real;
    KMem[ni]^ [j].Real:=s;
    s:= KMem[i ]^ [nj].Imag;
    KMem[i ]^ [nj].Imag:=KMem[ni]^ [j].Imag;
    KMem[ni]^ [j].Imag:=s;
  End;
End;
End;

Procedure Copy(Dir: Integer);
{ Copy Image P to Array or Array to Image Q }
Var i,j,n1,d: Integer; c: LongInt;
Begin
n1:=N-1;
If Dir=+1 Then
Begin
c:=255;
For i:=0 to n1 Do
For j:=0 to n1 Do
  Begin
  With KMem[i]^ [j] Do
  Begin Real:=PMem[i]^ [j] And c; Imag:=0; End;
  End;
End;
End;
```

10.4 FFT Code

```
If Dir=-1 Then
Begin
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
With KMem[i]^ [j] Do d:=Round(Sqrt(Sqr(Real)+Sqr(Imag)));
{If d<0 Then d:=0 Else If d>255 Then d:=255; }
c:=LimTab^ [d];
QMem[i]^ [j]:= c SHL 16 + c SHL 8 + c; { Format 0ccc }
End;
End;
End;
```

```
Procedure FilterD (sig: Single; ya: Integer);
{ Descreen Filter }
Var i,j,n0,n1,n2          : Integer;
    sd,fu                 : Single;
    x1,y1,x2,y2,si,xa,yh : Integer;
Function ga(x,y:Single)   : Single;
Begin
ga:=GauTab[Round(Sqrt(Sqr(x)+Sqr(y)))] ;
End;
Begin
n1:=N-1; n2:=N div 2; n0:=n2-1;
sd:=sig*n2; { Standard deviation }
si:=Round(sd);
For i:=0 to N Do
Begin
If i<si Then GauTab[i]:=1 Else
Begin
fu:=6*Sqr((i-sd)/50);
If fu<5 Then GauTab[i]:=exp(-fu) Else GauTab[i]:=0;
End;
End;
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
fu:=ga(i-n0,j-n0);
With KMem[i]^ [j] Do
Begin Real:=Real*fu; Imag:=Imag*fu; End;
End;
yh:=100;
x1:=0; y1:=Round(-yh*ga(-n0,0));
For i:=-n0 to n2 Do
Begin
x2:=x1+1; y2:=Round(-yh*ga(i,0));
MakeSline(x1,ya+y1,x2,ya+y2,0,140);
x1:=x2; y1:=y2;
End;
MakeSline(0,ya,N,ya,0,0);
MakeSline(n0,ya,n0,ya-yh,0,0);
MakeSline(n0-si,ya,n0-si,ya-yh,0,0);
MakeSline(n0+si,ya,n0+si,ya-yh,0,0);
End;
```

```
Procedure FilterG (sig: Single; ya: Integer);
{ Gaussian Blur Filter }
Var i,j,n0,n1,n2          : Integer;
    sd,fu                 : Single;
    x1,y1,x2,y2,si,xa,yh : Integer;
    dd                    : Double;
Function ga(x,y:Single)   : Single;
Begin
ga:=GauTab[Round(Sqrt(Sqr(x)+Sqr(y)))] ;
End;
Begin
n1:=N-1; n2:=N div 2; n0:=n2-1;
sd:=sig*n2; { Standard deviation }
si:=Round(sd);
```

10.5 FFT Code

```

For i:=0 to N Do
Begin
dd:=0.5*Sqr(i/sd);
If dd<5 Then GauTab[i]:=exp(-dd) Else GauTab[i]:=0;
End;
For i:=0 to n1 Do
For j:=0 to n1 Do
Begin
fu:=ga(i-n0,j-n0);
With KMem[i]^ [j] Do
Begin Real:=Real*fu; Imag:=Imag*fu; End;
End;
yh:=100;
x1:=0; y1:=Round(-yh*ga(-n0,0));
For i:=-n0 to n2 Do
Begin
x2:=x1+1; y2:=Round(-yh*ga(i,0));
MakeSline(x1,ya+y1,x2,ya+y2,0,140);
x1:=x2; y1:=y2;
End;
MakeSline(0,ya,N,ya,0,0);
MakeSline(n0,ya,n0,ya-yh,0,0);
MakeSline(n0-si,ya,n0-si,ya-yh,0,0);
MakeSline(n0+si,ya,n0+si,ya-yh,0,0);
End;

Procedure Box(x1,y1,N:Integer);
Var n1,n2: Integer;
Begin
n1:=N-1;
MakeSline(x1, y1 ,x1+n1,y1,0,0);
MakeSline(x1,y1+n1,x1+n1,y1+n1,0,0);
MakeSline(x1, y1,x1 ,y1+n1,0,0);
MakeSline(x1+n1,y1,x1+n1,y1+n1,0,0);
End;
{Pend}

BEGIN
Configure
(ConfigMax,ZebraMax,Flag,Conf,ZebrArray,
ZebrPath,SmdrPath,
BuffDrNm,SnapDrNm,ZebrNm,
LastDriv,SourDriv,DestDriv,
SourceNr,GcardIdy,VesaMode,VesaCode,
SoundsOn,NoClkInt,AutoCatG,ShowIcon);
MemKStart; { KMem, PMem, QMem }
VesaStart(VesaMode);
GamFill(GamTab,1/4); { Gamma Tab.,clipping }
LimFill(LimTab); { Clipping Table }
Im0:='I:\fft_\fft__900.bmp';
Im1:='I:\fft_\fft__911.bmp';
Im2:='I:\fft_\fft__912.bmp';
ColToScr(grsc,255); { Background }
N:=512; { Size }
N1:=N-1;
FFT0(Co,Si,N); { Twiddle }
vis:=0;
ReadImag(Im0,'P',vis,flag); { Store Image in PMem }
PMemGrYIQ(0,0,N1,N1,0,0,2); { Convert to Gray }
PMemToSx(0,0,N1,N1,0,0); { Show at 0,0 }
{ Original Spectrum }
Copy(+1); { Copy Image to Array }
FFT2(+1); { Array to Spectrum }
MakeS(false); { Spectrum Image }
QMemToSx(0,0,N1,N1,N,0); { Show at N,0 }
{ Swapped Spectrum }
Swap; { Swap Spectrum }
MakeS(true); { Spectrum Image }
QMemToSx(0,0,N1,N1,N,N); { Show at N,N }

```

10.6 FFT Code

```
{ Reconstruction }
Swap;
FFT2(-1);           { Spectrum to Array }
Copy(-1);          { Array to Image }
QMemToSx (0,0,N1,N1,0,N); { Show at 0,N }
Swap;
Box(0,0,N);
Box(0,N,N);
Stop;
SaveImag(Im1);     { Save page 1 }
ColToScr(grsc,255); { Background }
{ Gaussian Filter }
Copy(+1);          { Copy Image to Array }
FFT2(+1);          { Array to Spectrum }
Swap;
FilterG(0.20,650); { Gaussian Filter }
Swap;
FFT2(-1);
Copy(-1);
QMemToSx (0,0,N1,N1,0,0);
{ Descreen Filter 1 }
Copy(+1);          { Copy Image to Array }
FFT2(+1);          { Array to Spectrum }
Swap;
FilterD(0.3,800);  { Descreen Filter 1 }
Swap;
FFT2(-1);
Copy(-1);
QMemToSx (0,0,N1,N1,N,0);
{ Descreen Filter 2 }
Copy(+1);          { Copy Image to Array }
FFT2(+1);          { Array to Spectrum }
Swap;
FilterD(0.4,950);  { Descreen Filter 2 }
Swap;
FFT2(-1);
Copy(-1);
QMemToSx (0,0,N1,N1,N,N);
Box(0,0,N);
Box(N,0,N);
Box(0,N,N);
Box(N,N,N);
SaveImag(Im2);     { Save page 2 }
Stop;
VesaEnde;
MemKEnde;
ClrScr;
END.
```

11.1 Gaussian Filters

A Gaussian filter in the spatial domain has to be mapped to a Gaussian filter in the frequency domain.

The weighting factors for a filter box with width $-n$ to $+n$ in the spatial domain are calculated according to an investigation by the author [6]:

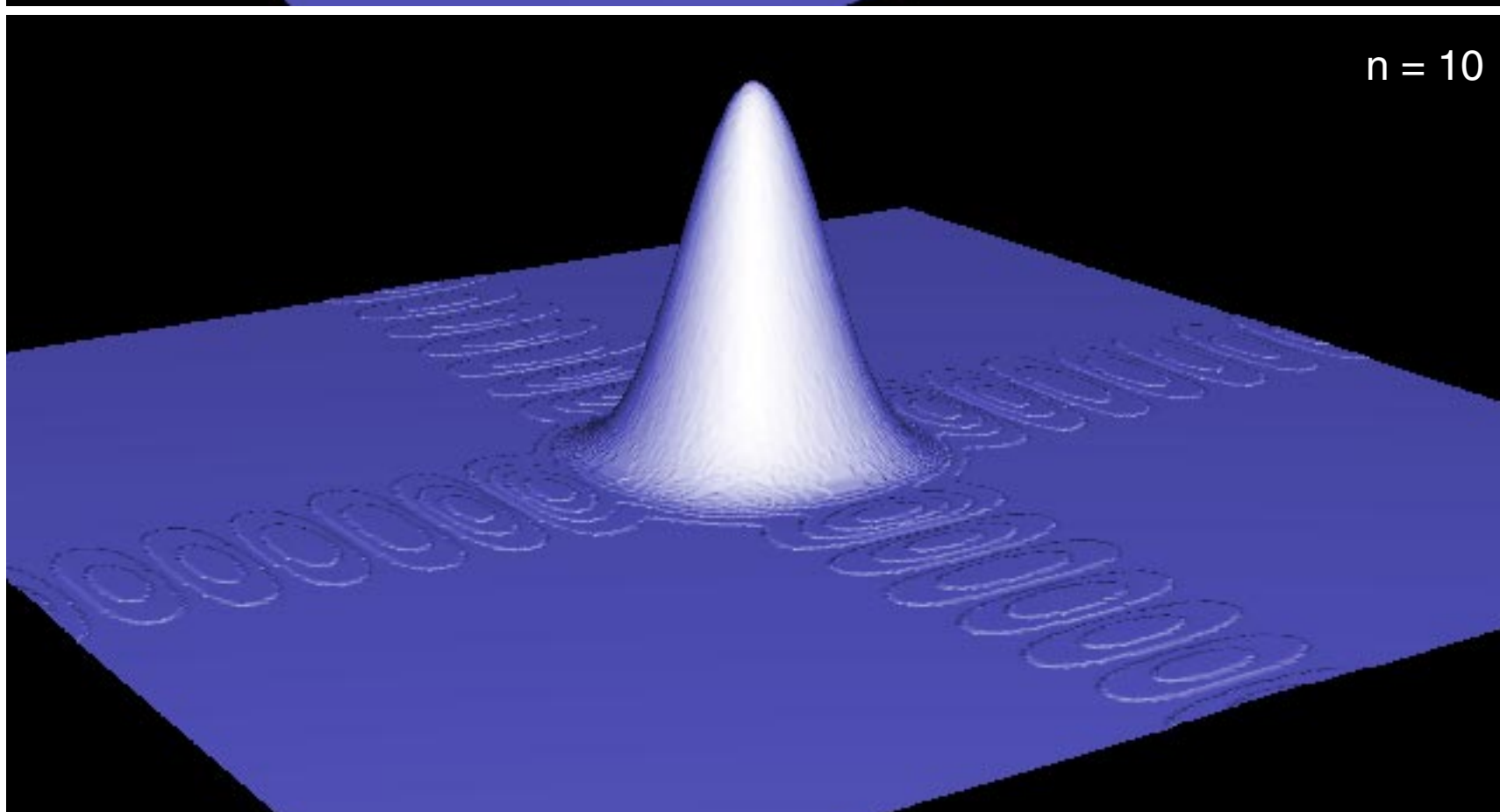
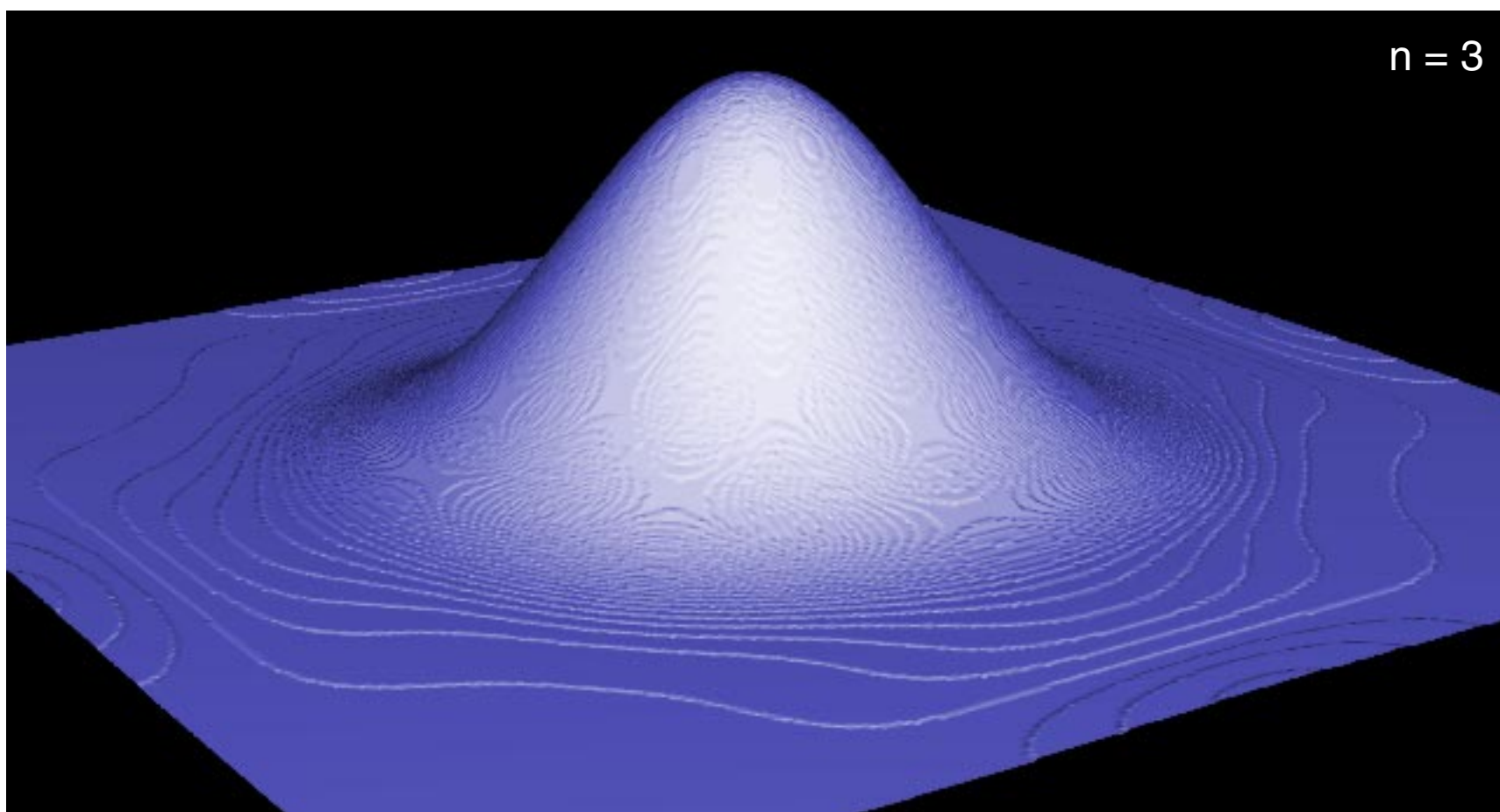
$$w(x,y) = \exp[-0.5 (x^2 + y^2)/(0.465n)^2]$$

The weighting factors are converted into an image with $(2n+1)^2$ pixels.

This small image is a part of a larger 512 x 512 pixel image.

The FFT of this image delivers immediately the equivalent filter function in the frequency domain, as shown here for $n=3$ and $n=10$.

Best view zoom 100%



12.1 3D Relief

The gray values of a filtered image are used as heights for a terrain. The image was filtered in the frequency domain (just an exercise).

Image width 512. The terrain is modeled with 400 x 400 grid points, using linear interpolation in the filtered image.

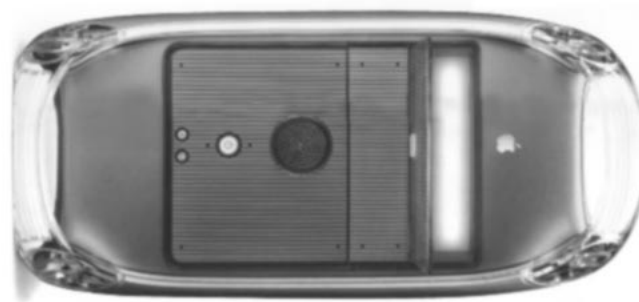
Spectra gray values are enhanced by a strong inverse gamma correction, exponent 0.25. The 3D height maps are linear.

Best view zoom 200%

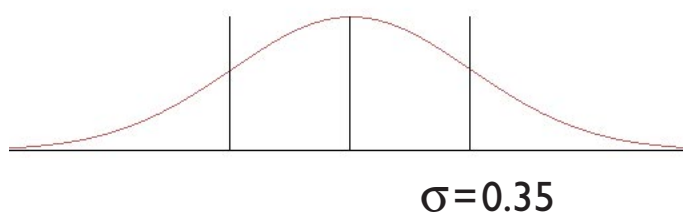
Original



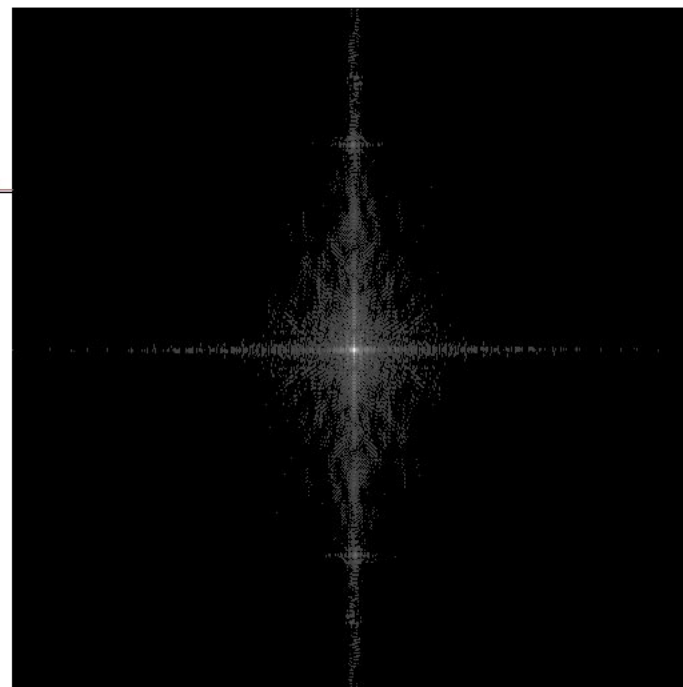
Filtered



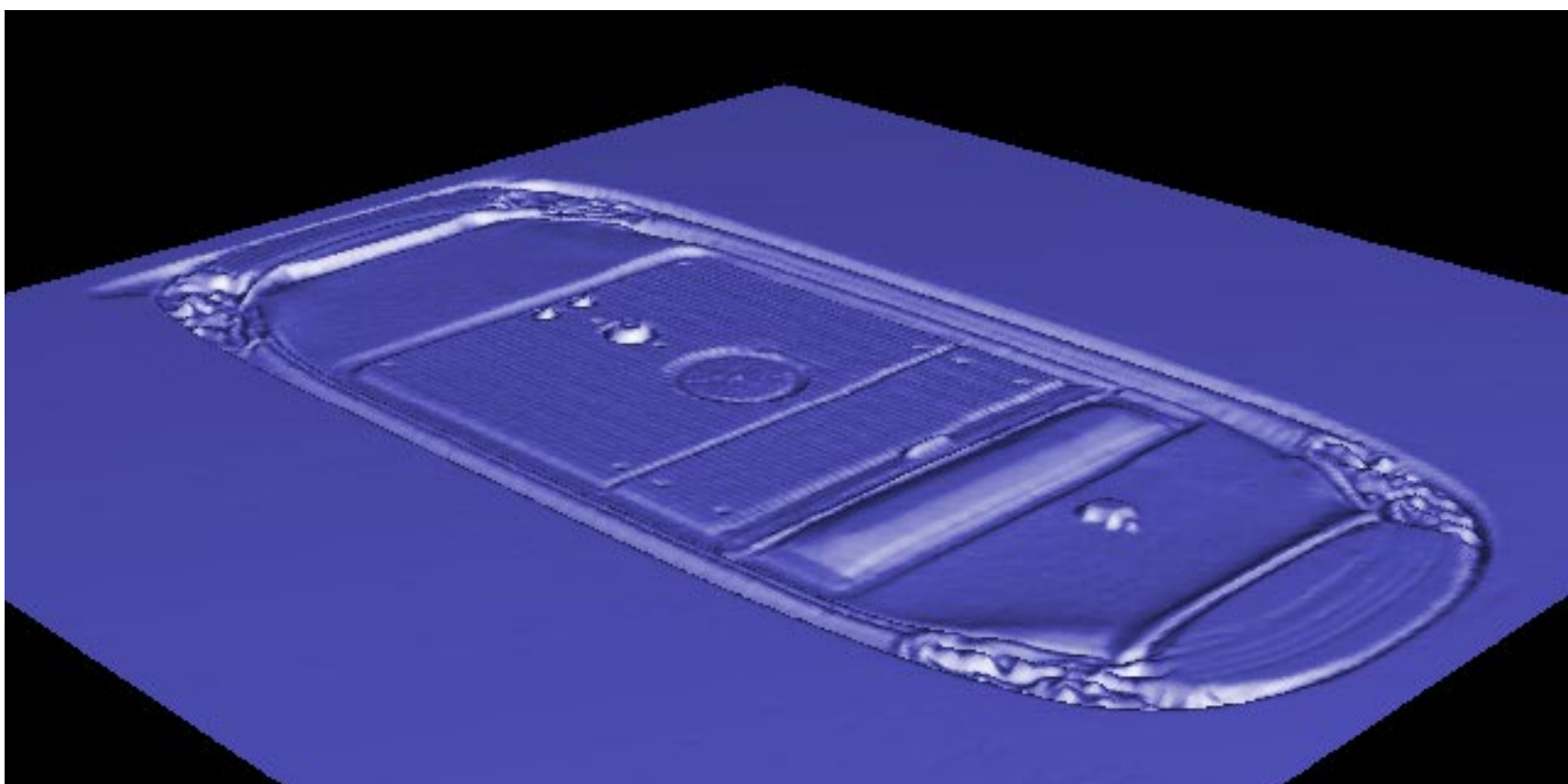
Filter



Spectrum



3D Relief



13.1 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] Paul Bourke
2-Dimensional FFT
<http://astronomy.swin.edu.au/~pbourke/analysis/fft2d>

 - [4] Gernot Hoffmann
Scanning Rastered Images
<http://www.fho-emden.de/~hoffmann/scan121200.pdf>

 - [5] Robert Mapplethorpe
Lady Lisa Lyon
Schirmer/Mosel, München 1983

 - [6] Gernot Hoffmann
Gaussian Filters
<http://www.fho-emden.de/~hoffmann/gauss25092001.pdf>
- This doc:
<http://www.fho-emden.de/~hoffmann/fft31052003.pdf>