

G. Hoffmann

Drawing Lines



Please download and read by Acrobat
Zoom 100%
Browsers are sometimes not accurate

Contents

1. Straight lines by MakeSline	2
2. Straight lines by MakeNLine	2
3. Speed test	4
4. Back to the roots - MakePLine	5
5. Stair lines by MakeMLine	6
6. Stair paths by MakeMPath	7
7. Drawing arbitrary functions	9

1. Straight lines by MakeSLine / Normal line style

This is a fast and reliable line drawing algorithm. It is not based on Bresenham. Bresenham's well known method (1965) needs endpoint swapping (at least for pointers) and the unpleasant octant sorting (in the original version).

The Sawtooth Method by the author (1996) is here published the first time.

```
Procedure MakeSLine (p0,q0,p1,q1: Integer; pal,col: Byte);
{ G.Hoffmann, September 30, 1996; Sawtooth Interpolation }
{ Revision December 22, 2001; minor improvements }
{ Tutorial Pascal version; an Assembler version is available }
{ Draws a line from (p0,q0) to (p1,q1) with color pal,col }
{ Uses an auxiliary sawtooth variable; true integer algorithm }
{ Perfect forward-reverse matching without endpoint swap }
{ By omitting the statement (2) there may be one more pixel x }
{ when the line is drawn forward and reverse }
{ p0*****x }
{ *****p1 }
{ The maximal deviation from the straight line is 0.5 pixel }
{ Versions for z-buffer and color interpolation are available }
Var dx,dy: Integer; saw,adx,ady,adm: Word; { or Integer }
Begin
dx:=p1-p0;
dy:=q1-q0;
adx:=Abs(dx); { absolute value }
ady:=Abs(dy); { absolute value }
If adx>=ady Then adm:=adx Else adm:=ady; { maximum }
SpcSixel (p0,q0,pal,col); { pixel No.0 of 0...adm explicit }
If adm>=2 Then { executed if at least 3 pixels }
Begin
If dx>=0 Then dx:=1 Else dx:=-1; { sign.increment }
If dy>=0 Then dy:=1 Else dy:=-1; { sign.increment }
{ 1 } saw:=adm SHR 1; { initial value ; shift right: saw:=adm div 2 }
{ 2 } If (dx=-1) And Not Odd(adm) Then Dec(saw);
If adx>=ady Then
For adx:=2 To adm Do
Begin
p0:=p0+dx; saw:=saw+ady;
If saw>=adm Then
Begin saw:=saw-adm; q0:=q0+dy;
End;
SpcSixel (p0,q0,pal,col);
End
Else
For ady:=2 To adm Do
Begin
q0:=q0+dy; saw:=saw+adx;
If saw>=adm Then
Begin saw:=saw-adm; p0:=p0+dx;
End;
SpcSixel (p0,q0,pal,col);
End;
End;
SpcSixel (p1,q1,pal,col);
End;
```



2. Straight lines by MakeNLine / New version of MakeSLine

This is an improved version of MakeSLine. The code is shorter and more elegant. MakeNLine is not faster than MakeSLine because now the first and the last pixel are drawn in the loop instead of explicitly.

```
Procedure MakeNLine (p0,q0,p1,q1: Integer; pal,col: Byte);  
{ G.Hoffmann, September 30, 1996; Sawtooth Interpolation }  
{ Revision December 24, 2001 }  
Var dx,dy: Integer; saw,adx,ady,adm: Word; { or Integer }  
Begin  
  dx:=p1-p0;  
  dy:=q1-q0;  
  adx:=Abs(dx);  
  ady:=Abs(dy);  
  If adx>=ady Then adm:=adx Else adm:=ady;  
  saw:=adm SHR 1;  
  If dx>=0 Then dx:=1 Else  
    Begin dx:=-1; If Not Odd(adm) Then Dec(saw);  
    End;  
  If dy>=0 Then dy:=1 Else dy:=-1;  
  If adx>=ady Then  
    For adx:=0 To adm Do  
      Begin  
        SpcSixel(p0,q0,pal,col);  
        p0:=p0+dx; saw:=saw+ady;  
        If saw>=adm Then  
          Begin saw:=saw-adm; q0:=q0+dy;  
          End;  
        End  
      Else  
        For ady:=0 To adm Do  
          Begin  
            SpcSixel(p0,q0,pal,col);  
            q0:=q0+dy; saw:=saw+adx;  
            If saw>=adm Then  
              Begin saw:=saw-adm; p0:=p0+dx;  
              End;  
            End;  
          End;  
        End;  
      End;  
End;
```

3. Speed test / MakeSLine versus Bresenham MakeBLine

MakeBLine uses the Bresenham line drawing algorithm by Kenny Hoff (1995). Unfortunately the retracing was not correct (code right side).

<http://www.cs.unc.edu/~hoff/projects/comp235/bresline/src/bresline.c>

Execution times were measured for 10 Mill. vectors *without pixel drawing*. Vectors can have any random length in a box with 10, 100 or 500 pixels width. The time for the empty loop including the random coordinate generation was subtracted. Pentium II, 350 MHz.

MakeBLine (corrected, left side) is faster than MakeSLine.

Pixel drawing may consume more than 6 times the execution time for the logical algorithm, depending on the mode (clipping, banked framebuffer).

```
Procedure MakeBLine (p0,q0,p1,q1: Integer; pal,col: Byte);
```

```
{Correction by G.Hoffmann }
Var x,y,dx,dy,adx,ady,ix,iy,du,dr,pp,cp,i:Integer;
Begin
dx:=p1-p0;    dy :=q1-q0;
adx:=Abs(dx);  ady :=Abs(dy);
x :=p0;      y :=q0;      cp:=0;
If dx>0 Then ix:=+1 Else Begin ix:=-1;cp:=-1; End;
If dy>0 Then iy:=+1 Else iy:=-1;
If adx>=ady Then
Begin
dr:=2*ady; du:=dr-2*adx; pp:=dr-adx;
For i:=0 to adx Do
Begin
SpcSixel(x,y,pal,col);
x:=x+ix;
If pp>cp Then
Begin y:=y+iy; pp:=pp+du;
End Else pp:=pp+dr;
End;
End Else
Begin
dr:=2*adx; du:=dr-2*ady; pp:=dr-ady;
For i:=0 To ady Do
Begin
SpcSixel(x,y,pal,col);
y:=y+iy;
If pp>cp Then
Begin x:=x+ix; pp:=pp+du;
End Else pp:=pp+dr;
End;
End;
End;
```

```
{ Algorithm by Kenny Hoff 09/02/95 }
Var x,y,dx,dy,adx,ady,ix,iy,du,dr,pp,i:Integer;
Begin
dx :=p1-p0;    dy :=q1-q0;
adx :=Abs(dx);  ady :=Abs(dy);
x :=p0;      y :=q0;
If dx<0 Then ix:=-1 Else ix:=+1;
If dy<0 Then iy:=-1 Else iy:=+1;
If adx>=ady Then
Begin
dr:=2*ady; du:=dr-2*adx; pp:=dr-adx;
For i:=0 to adx Do
Begin
SpcSixel(x,y,pal,col);
If pp>0 Then
Begin x:=x+ix; y:=y+iy; pp:=pp+du;
End Else
Begin x:=x+ix; pp:=pp+dr;
End;
End;
End Else
Begin
dr:=2*adx; du:=dr-2*ady; pp:=dr-ady;
For i:=0 To ady Do
Begin
SpcSixel(x,y,pal,col);
If pp>0 Then
Begin x:=x+ix; y:=y+iy; pp:=pp+du;
End Else
Begin y:=y+iy; pp:=pp+dr;
End;
End;
End;
```

Random Vector box	MakeBLine ms per Vector	MakeBLine ns per Pixel	MakeSLine ms per Vector	MakeSLine ns per Pixel
0 ... 10	0.884	171	1.290	250
0 ... 100	2.663	51	3.026	58
0 ... 500	10.259	40	10.255	40

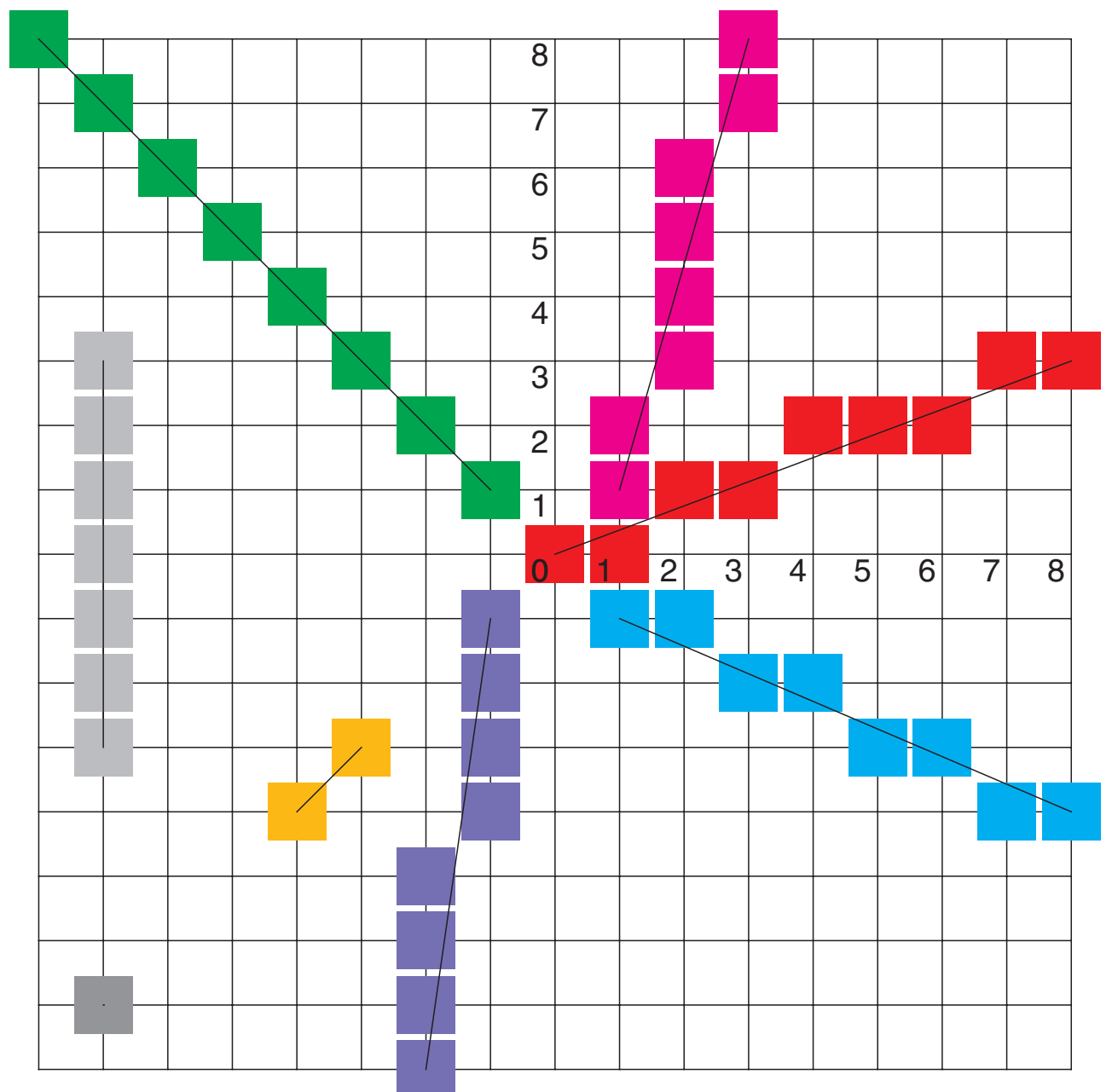
4. Back to the roots - MakePLine

MakePLine uses a straightforward incremental implementation. The graphic was created by PostScript. The speed of the algorithm is hardly of any importance because the rendering of the simulated pixels consumes much more time. Below pseudo code and the PostScript kernel.

```

Pixel(x0,y0)
dx =x1-x0  dy =y1-y0
adx=abs(dx)  ady=abs(dy)
If ((adx>0) or (ady>0)) Then
Begin
  If adx>ady Then
  Begin
    If dx>0 Then dnx=+1 Else dnx=-1
    dnf=dy/adx % float
    yf =y0      % float
    For i=1 To adx Do
    Begin
      x0=x0+dnx
      yf=yf+dnf % float
      Pixel(x0,Round(yf))
    End
  End Else
  Begin
    If dy>0 Then dny=+1 Else dny=-1
    dnf=dx/ady % float
    xf =x0      % float
    For i=1 To ady Do
    Begin
      xf=xf+dnf % float
      y0=y0+dny
      Pixel(Round(xf),y0)
    End
  End
End
End

```



```

/MakePLine
{x0 y0 Pixel
/dx x1 x0 sub def
/dy y1 y0 sub def
/adx dx abs def
/ady dy abs def
adx 0 gt ady 0 gt or
{
  adx ady gt
  { dx 0 gt {/dnx 1 def}
    {/dnx -1 def} ifelse
  /dny dy adx div def
  adx
  {/x0 x0 dnx add def
  /y0 y0 dny add def
  x0 y0 round Pixel
  } repeat
}
{
  /dnx dx ady div def
  dy 0 gt {/dny 1 def}
  {/dny -1 def} ifelse
  ady
  {/x0 x0 dnx add def
  /y0 y0 dny add def
  x0 round y0 Pixel
  } repeat
} ifelse
} if % adx>0 or ady>0
} def

```

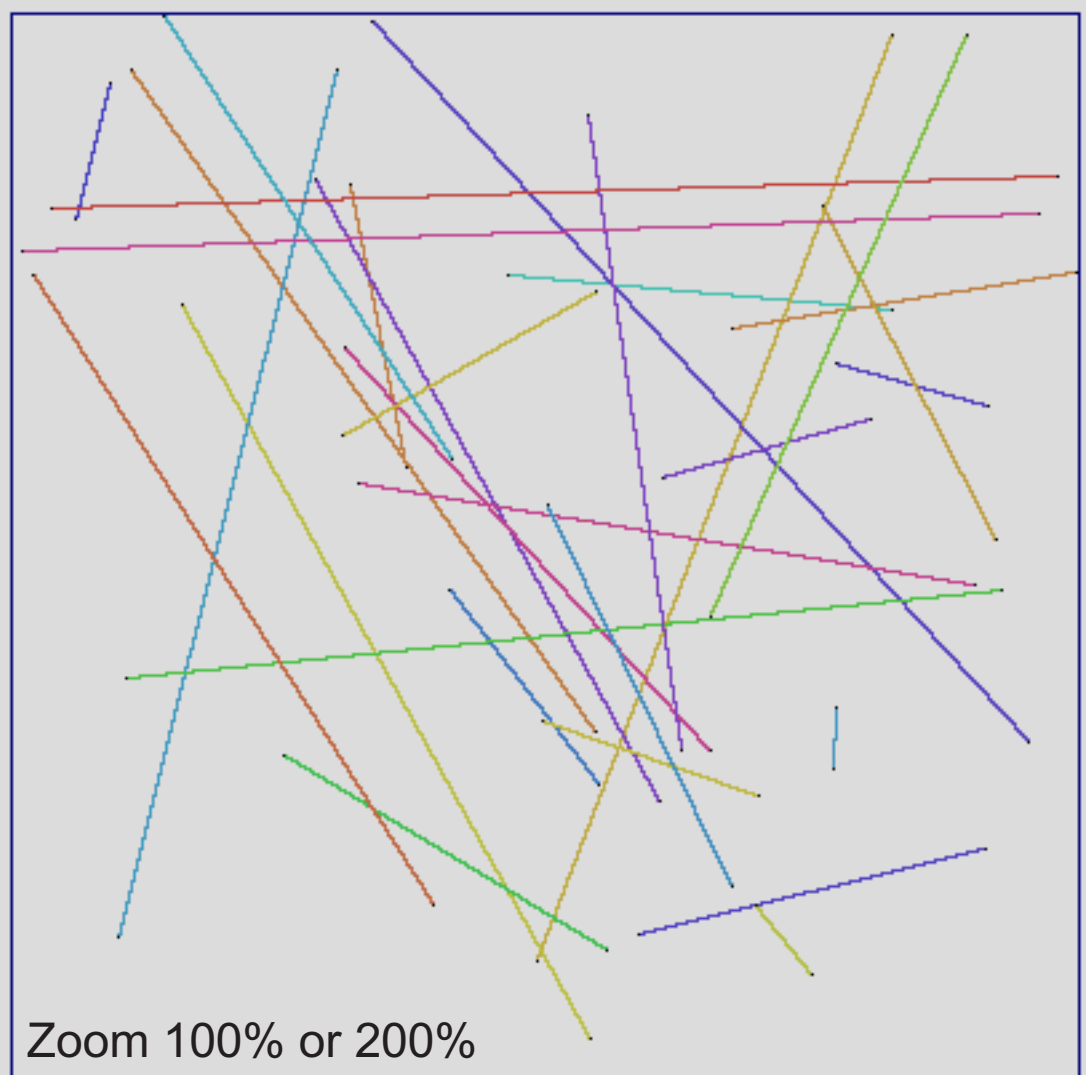
All lines were drawn in forward and reverse direction. The pixels are the same, but there is no guarantee that this will be always the case.

The red sequence uses at (4,2) correct rounding of the line coordinates (4, 1.500). Any tiny deviation may create the transition pixel (at 4, 1).

5. Stair lines by MakeMLine / Manhattan line style

Draws an optimized stair line in a grid. Grid width one pixel.

```
Procedure MakeMLine(p0,q0,p1,q1: Integer; Const pal,col: Byte);
{ Copyright G.Hoffmann, November 11, 2001;
  Pixel line from p0,q0 to p1,q1
  p0,q0,p1,q1   : Pixels
  pal,col       : both for color           }
Var i,dp,dq,p,q : Integer; saw,adp,adq,adm: Word;
Begin
dp:=p1-p0;      dq:=q1-q0;
p:=p0;          q:=q0;
adp:=Abs(dp);   adq:=Abs(dq);
SpcSixel(p,q,stan,blac);
If adm>=2 Then
Begin
If adp>=adq Then adm:=adp Else adm:=adq;
If adp>0 Then If dp>0 Then dp:=1 Else dp:=-1 Else dp:=0;
If adq>0 Then If dq>0 Then dq:=1 Else dq:=-1 Else dq:=0;
saw:=adm SHR 1;
If (dp=-1) And Not Odd(adm) Then Dec(saw);
If adp>=adq Then
  For i:=1 To adm Do
  Begin
  p:=p+dp; saw:=saw+adq;
  SpcSixel(p,q,pal,col);
  If saw>=adm Then
  Begin
  saw:=saw-adm; q:=q+dq;
  SpcSixel(p,q,pal,col);
  End;
  End
Else
  For i:=1 To adm Do
  Begin
  q:=q+dq; saw:=saw+adp;
  SpcSixel(p,q,pal,col);
  If saw>=adm Then
  Begin
  saw:=saw-adm; p:=p+dp;
  SpcSixel(p,q,pal,col);
  End;
  End;
End;
SpcSixel(p1,q1,stan,blac);
End;
```



6.1 Stair paths by MakeMPath / Manhattan path style

Draws an optimized stair path in a grid. Grid width $ds = 2$ to n pixels.

```
Procedure MakeMPath (p0,q0,p1,q1,ds: Integer; Const pal,col: Byte);
{ Copyright G.Hoffmann, November 13, 2001;
  Step line from p0,q0 to p1,q1
  p0,q0,p1,q1   : grid units
  ds            : grid width in pixels
  pal,col       : both for color          }
Var i,dp,dq,dx,dy,x,y,xs,ys: Integer;
    saw,adp,adq,adm      : Word;
    m,x0,y0,dx2,dy2     : Single;
Label EX;
Procedure Xline(xa,xe,y: Integer);
Var i: Integer;
Begin
If xa<=xe Then For i:=xa to xe Do SpcSixel(i,y,pal,col)
                Else For i:=xe to xa Do SpcSixel(i,y,pal,col);
End;
Procedure Yline(ya,ye,x: Integer);
Var i: Integer;
Begin
If ya<=ye Then For i:=ya to ye Do SpcSixel(x,i,pal,col)
                Else For i:=ye to ya Do SpcSixel(x,i,pal,col);
End;

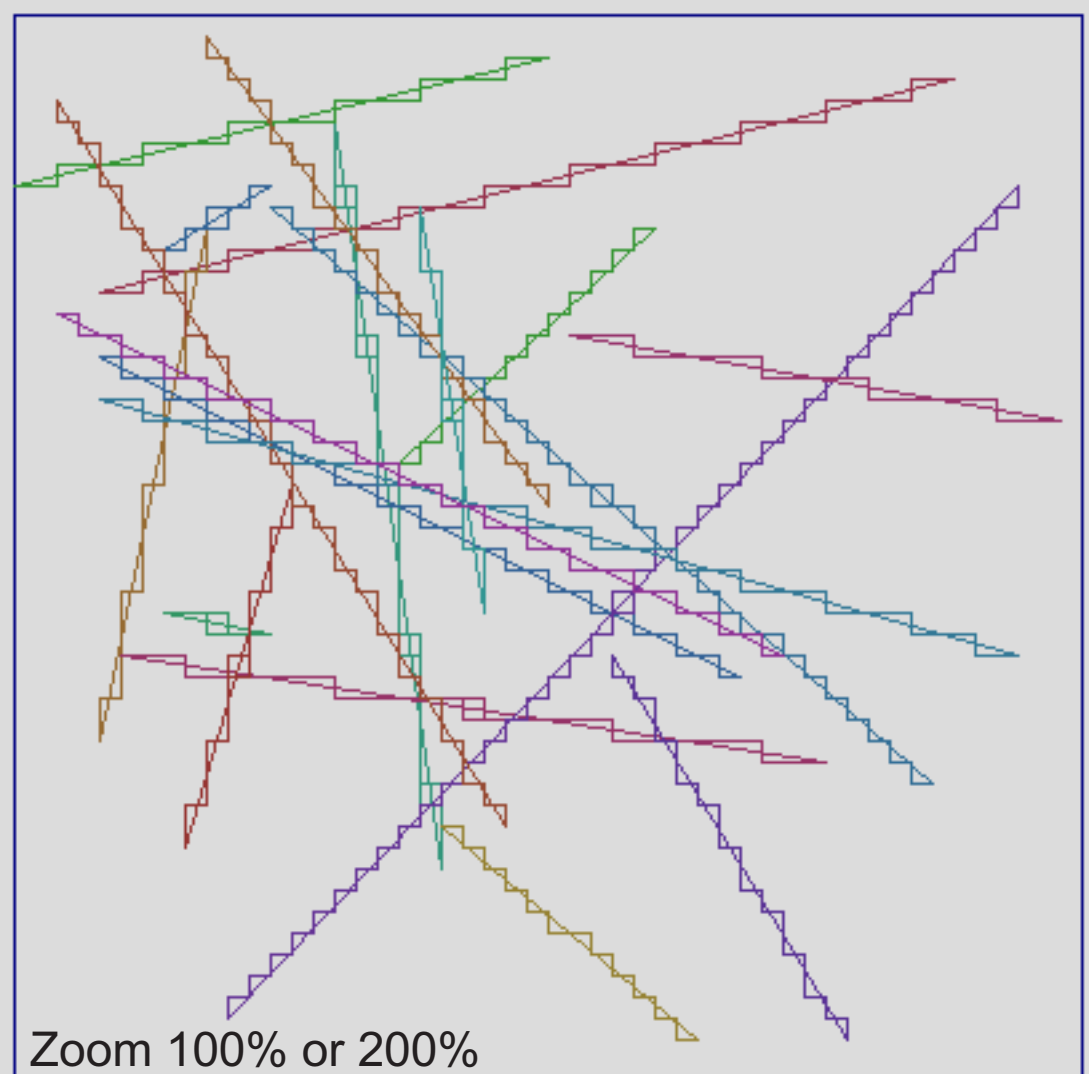
Begin
x :=p0*ds; x0:=x;
y :=q0*ds; y0:=y;
dp:=p1-p0;
dq:=q1-q0;
adp:=Abs(dp);
adq:=Abs(dq);
If adp>=adq Then adm:=adp
                Else adm:=adq;
If adm=0 Then Goto EX;
If adp>0 Then If dp>0 Then dx:=ds
                Else dx:=-ds;
                Else dx:=0;
If adq>0 Then If dq>0 Then dy:=ds
                Else dy:=-ds;
                Else dy:=0;
dx2:=0.5*dx-x0;
dy2:=0.5*dy-y0;
saw:=adm SHR 1;
If (dp<0) And Not Odd(adm)
Then Dec(saw);

( continued )
```

All paths are drawn in forward and reverse direction.

The matching is not absolutely perfect - a few lines differ occasionally.

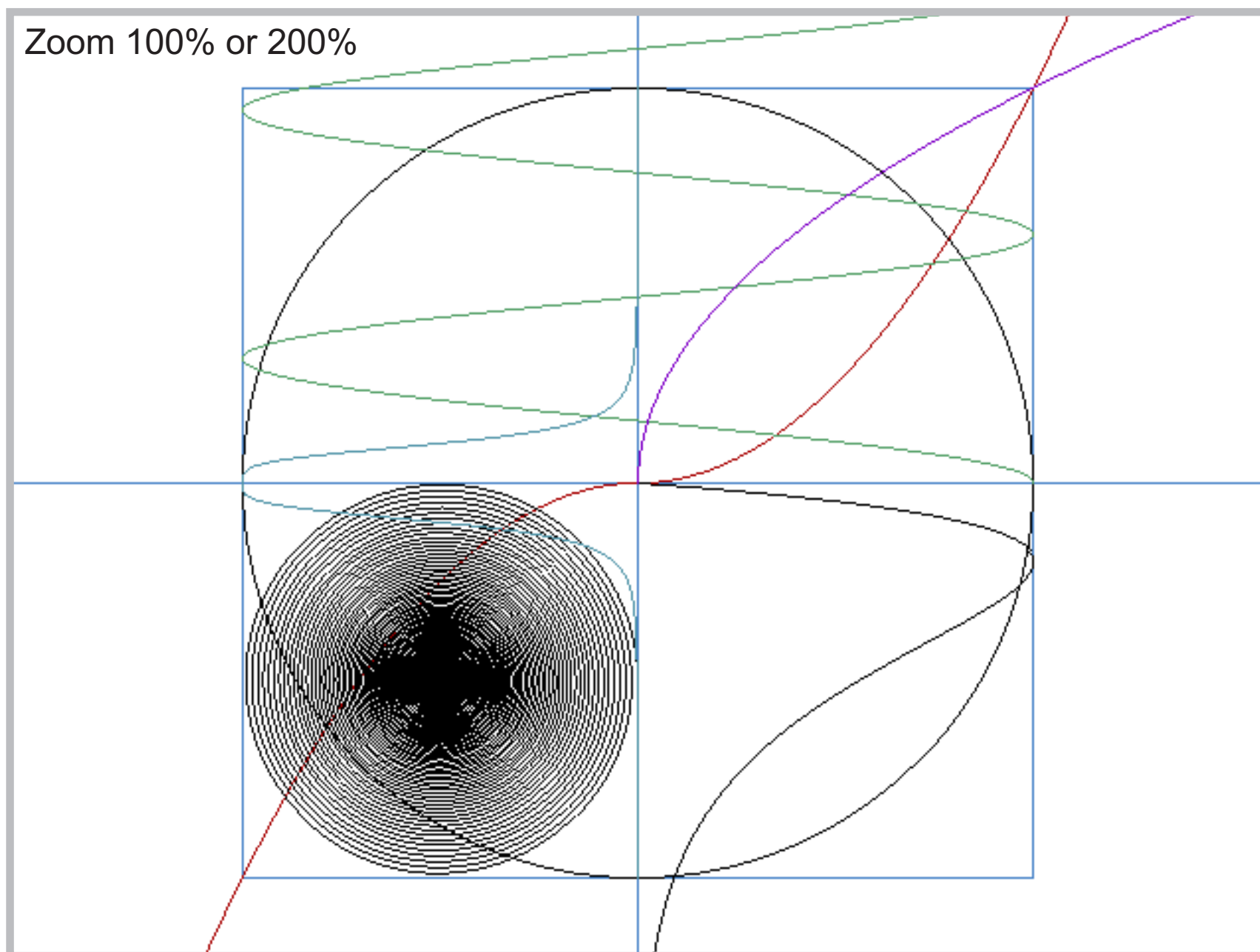
The steps are well balanced.



6.2 Straight lines by MakeMPath / Manhattan path style

```
If adp>=adq Then
Begin
  m:=dq/dp;
  For i:=1 To adm Do
  Begin
    xs:=x; x:=x+dx; saw:=saw+adq;
    If saw>=adm Then
    Begin
      saw:=saw-adm; ys:=y; y:=y+dy;
      If dq>0 Then
      Begin
        If m*(xs+dx2)<=ys+dy2 Then
        Begin
          XLine(xs,x,ys); YLine(ys,y,x);
        End Else
        Begin
          YLine(ys,y,xs); XLine(xs,x,y);
        End;
      End Else
      Begin
        If m*(xs+dx2)>ys+dy2 Then
        Begin
          XLine(xs,x,ys); YLine(ys,y,x);
        End Else
        Begin
          YLine(ys,y,xs); XLine(xs,x,y);
        End;
      End;
    End Else XLine(xs,x,y);
  End;
End
Else
Begin
  m:=dp/dq;
  For i:=1 To adm Do
  Begin
    ys:=y; y:=y+dy; saw:=saw+adp;
    If saw>=adm Then
    Begin
      saw:=saw-adm; xs:=x; x:=x+dx;
      If dp>0 Then
      Begin
        If m*(ys+dy2)<=(xs+dx2) Then
        Begin YLine(ys,y,xs); XLine(xs,x,y);
        End Else
        Begin XLine(xs,x,ys); YLine(ys,y,x);
        End;
      End Else
      Begin
        If m*(ys+dy2)>(xs+dx2) Then
        Begin YLine(ys,y,xs); XLine(xs,x,y);
        End Else
        Begin XLine(xs,x,ys); YLine(ys,y,x);
        End;
      End;
    End Else YLine(ys,y,x);
  End;
End;
EX:
End;
```

7.1 Drawing arbitrary functions



An arbitrary function is given analytically in parameter representation for $s=s_{\min}$ to s_{\max} :
 $x=x(s)$, $y=y(s)$.

The algorithm draws each pixel only once, takes the nearest pixel, does not cause gaps, does not use vectors or line line algorithms and is reasonably fast.

7.2 Drawing arbitrary functions / ZDrawFun

```
Program ZDrawFun;
{ Projekt:          Draw arbitrary function by pixels
  Author:          Gernot Hoffmann
                  http://www.fho-emen.de/~hoffmann

{ A function is given in parameter representation x=x(s) and y=y(s)
  Parameter range smin...smax
  Screen   range 0...gmx, 0...gmy pixels
  The function starts at s=smin
  The main direction is calculated by numerical differentiation Xs,Ys
  In the other direction one or no step is possible
  The parameter increment ds is calculated for one step in the main
  direction
  Then the function continues at s+ds until smax is reached
  Limitations:
  The computation requires that the function can be calculated in the
  range smin-ds to smax+ds, where ds is a small number
  It is assumed, that pixel coordinates are in limits +-32000      }

Uses      Crt,Dos,
          Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
          Zefir36,Zefir37,Zefir38,Zefir39,Zefir40;

Const     Rname='ZDrawFun';
          RDate='October 15, 2001';

Var       smin,smax,sd,s2,eps: Single;
          xm,ym,scale,i,flag : Integer;

Procedure FuncVal(sel: Integer; s: Single; Var xf,yf: Single);
Var ra: Single;
Begin
Case sel Of
1: Begin
    xf:=+s;          { Function x = x(s)          }
    yf:=+Sqr(s);    {           y = y(s)          }
  End;
2: Begin
    xf:=-s;
    yf:=-Sqr(s);
  End;
3: Begin
    xf:=cos(s);
    yf:=sin(s);
  End;
4: Begin
    xf:=cos(10*s);
    yf:=s;
  End;
5: Begin
    xf:=exp(-5*s)*13.6*s;
    yf:=-s;
  End;
6: Begin
    xf:=s;
    XpowerA(s,0.5,yf,flag); { y = x ^0.5   undefined for x<0   }
  End;
7: Begin
    ra:=0.5*exp(-s);
    xf:=ra*cos(200*s)-0.5;
    yf:=ra*sin(200*s)-0.5;
  End;
8: Begin
    yf:=(2*s-smax)/smax;
    xf:=Sqr(Sqr(yf))+eps;
    xf:=-eps/xf;
  End;
End; { Case }
End;
```

7.3 Drawing arbitrary functions / ZDrawFun

```
Procedure Params;
Begin
  eps:=1E-4;           { 1E-4 for Single; 1E-8 for double   }
  { gmx:=639;         Graphics card resolution             }
  { gmy:=479;         External                             }
  xm:=gmx Div 2;
  ym:=gmy Div 2;
  scale:=200;         { 1.0 -> 200 pixels                 }
  smin:= 0;          { Parameter range                   }
  smax:=+2*pi;
  sd:=(smax-smin)*eps; { Increment for num. differentiation }
  s2:=2*sd/scale;
End;

Procedure XpowerA (x,a: Single; Var y: Single; Var flag: Integer);
{ y=x^a for x>0 ; flag=0: OK, flag=1: no solution           }
Const eps=1E-30;
Begin flag:=0;
If x>eps Then y:=exp(a*ln(x)) Else Begin y:=0; flag:=1; End;
End;

Procedure FuncDer(sel: Integer; s: Single; Var Xs,Ys: Single);
Var x1,y1,x2,y2: Single;
Begin
  FuncVal(sel,s-sd,x1,y1);
  FuncVal(sel,s+sd,x2,y2);
  Xs:=x2-x1;           { Numerical differentiation         }
  Ys:=y2-y1;           { Originally Ys=(y2-y1)/(2*sd)   }
End;

Procedure DrawFunc(sel,col: Integer);
Var xf,yf,s,ds,sx,sy,Xs,Ys,aXs,aYs: Single;
    k,px,py           : Integer;
Const kmax=32000;
Begin
  s:=smin;
  k:=0;
  FuncVal(sel,s,xf,yf);
  px:=Round(scale*xf);
  py:=Round(scale*yf);
  SpcSixel(xm+px,ym-py,stan,col); { Includes Integer clipping }
  While (s<smax) And (k<kmax) Do
  Begin
    FuncDer(sel,s,Xs,Ys);
    aXs:=Abs(Xs);
    aYs:=Abs(Ys);
    If aXs>=aYs Then
    Begin
      s:=s+s2/aXs;
      FuncVal(sel,s,xf,yf);
      If Xs>0 Then Inc(px) Else Dec(px);
      py:=Round(scale*yf);
    End Else
    Begin
      s:=s+s2/aYs;
      FuncVal(sel,s,xf,yf);
      If Ys>0 Then Inc(py) Else Dec(py);
      px:=Round(scale*xf);
    End;
    SpcSixel(xm+px,ym-py,stan,col); { Includes Integer clipping }
    Inc(k);
  End;
End;
```

7.4 Drawing arbitrary functions / ZDrawFun

```
Procedure Axes; External;
Procedure Box; External;

BEGIN
VesaMode:=Vmode41; { VMode41: $0112 VBE 640 x 480 all 4-Byte TrueColor}
                  { VMode42: $0115 VBE 800 x 600 }
                  { VMode43: $0118 VBE 1024 x 768 }
                  { VMode44: $011B VBE 1280 x1024 }
VesaCode:=$0112; { Must match the above pixel Numbers; Format $####; }
VesaStart(VesaMode);
ColToScr(stan,whit);
Params;
Axes;
Box;
DrawFunc(7,yell);
DrawFunc(8,cyad);
DrawFunc(1,redd);
DrawFunc(2,redd);
DrawFunc(3,blac);
DrawFunc(4,gred);
DrawFunc(5,blac);
DrawFunc(6,magd);
SaveImag('H:\DrawF\DrawF100.BMP');
tcode:=Gettcode;
VesaEnde;
END.
```

Gernot Hoffmann

December 24 / 2001 + October 20 / 2004 + December 04 / 2005

Website

[Load browser / Click here](#)