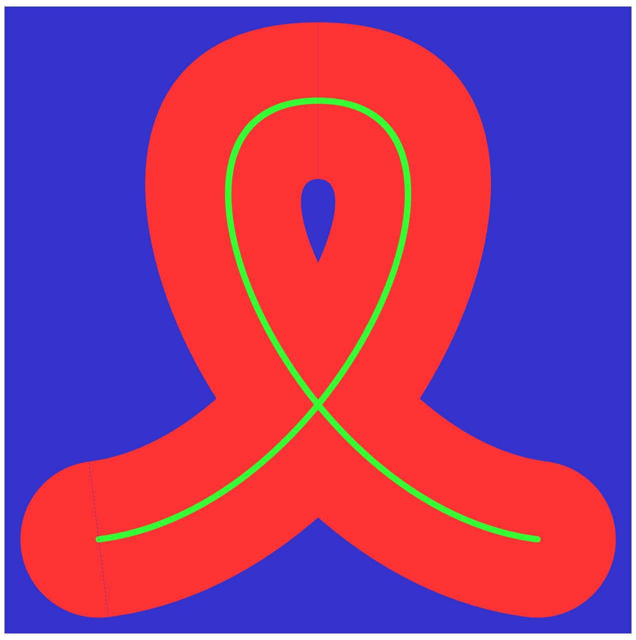
Gernot Hoffmann

Bézier Curves



Contents

Settings for Acrobat
Edit / Preferences / General / Page Display (since version 6)
Custom Resolution 72 dpi
Edit / Preferences / General / Color Management (full version only)
sRGB
EuroscaleCoated or ISOCoated or SWOP
GrayGamma 2.2

1. Definition of Bézier Polynomials

A graph segment is described in parameter form by two third-degree polynomials $x=x(t)$, $y=y(t)$ for $t=0$ to 1.
Each segment uses four control points P_0, P_1, P_2, P_3, where P_0 and P_3 are on the graph. The control points P_1, P_2 define the tangents in P_0, P_3. The longer the tangents the nearer is the curve to the tangents.

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_0$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + y_0$$

$$x(t) = ((a_x t + b_x)t + c_x)t + x_0$$
$$y(t) = ((a_y t + b_y)t + c_y)t + y_0$$

$$x_1 = x_0 + c_x/3$$
$$x_2 = x_1 + (c_x + b_x)/3$$
$$x_3 = x_0 + c_x + b_x + a_x$$

$$y_1 = y_0 + c_y/3$$
$$y_2 = y_1 + (c_y + b_y)/3$$
$$y_3 = y_0 + c_y + b_y + a_y$$

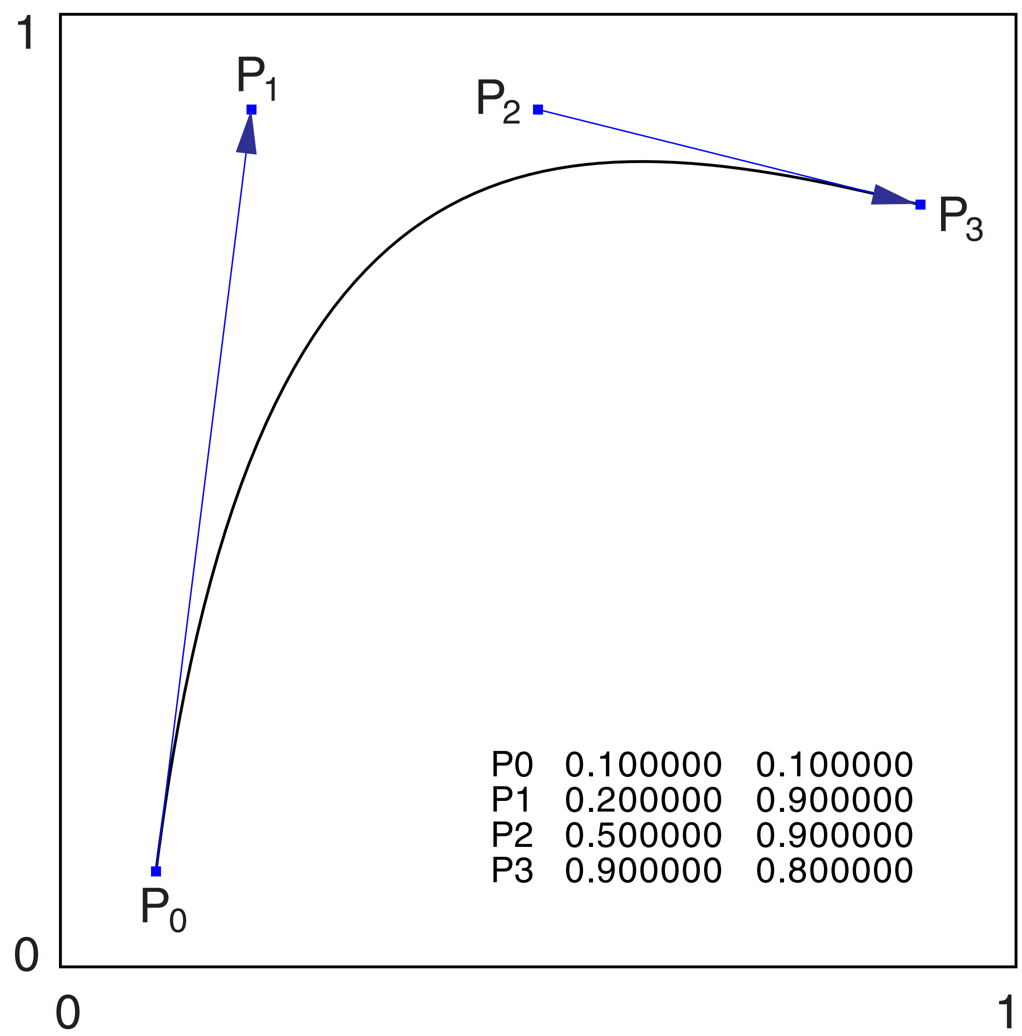
$$c_x = 3(x_1 - x_0)$$
$$b_x = 3(x_2 - x_1) - c_x$$
$$a_x = x_3 - x_0 - c_x - b_x$$

$$c_y = 3(y_1 - y_0)$$
$$b_y = 3(y_2 - y_1) - c_y$$
$$a_y = y_3 - y_0 - c_y - b_y$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}$$
$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$
$$\mathbf{b} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$
$$\mathbf{c} = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

$$\mathbf{a} = \mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0$$
$$\mathbf{b} = 3\mathbf{P}_2 - 6\mathbf{P}_1 + 3\mathbf{P}_0$$
$$\mathbf{c} = 3\mathbf{P}_1 - 3\mathbf{P}_0$$

$$\mathbf{a} = 3(\mathbf{P}_1 - \mathbf{P}_0) + 3(\mathbf{P}_3 - \mathbf{P}_2) - 2(\mathbf{P}_3 - \mathbf{P}_0)$$
$$\mathbf{b} = -6(\mathbf{P}_1 - \mathbf{P}_0) - 3(\mathbf{P}_3 - \mathbf{P}_2) + 3(\mathbf{P}_3 - \mathbf{P}_0)$$
$$\mathbf{c} = 3(\mathbf{P}_1 - \mathbf{P}_0)$$



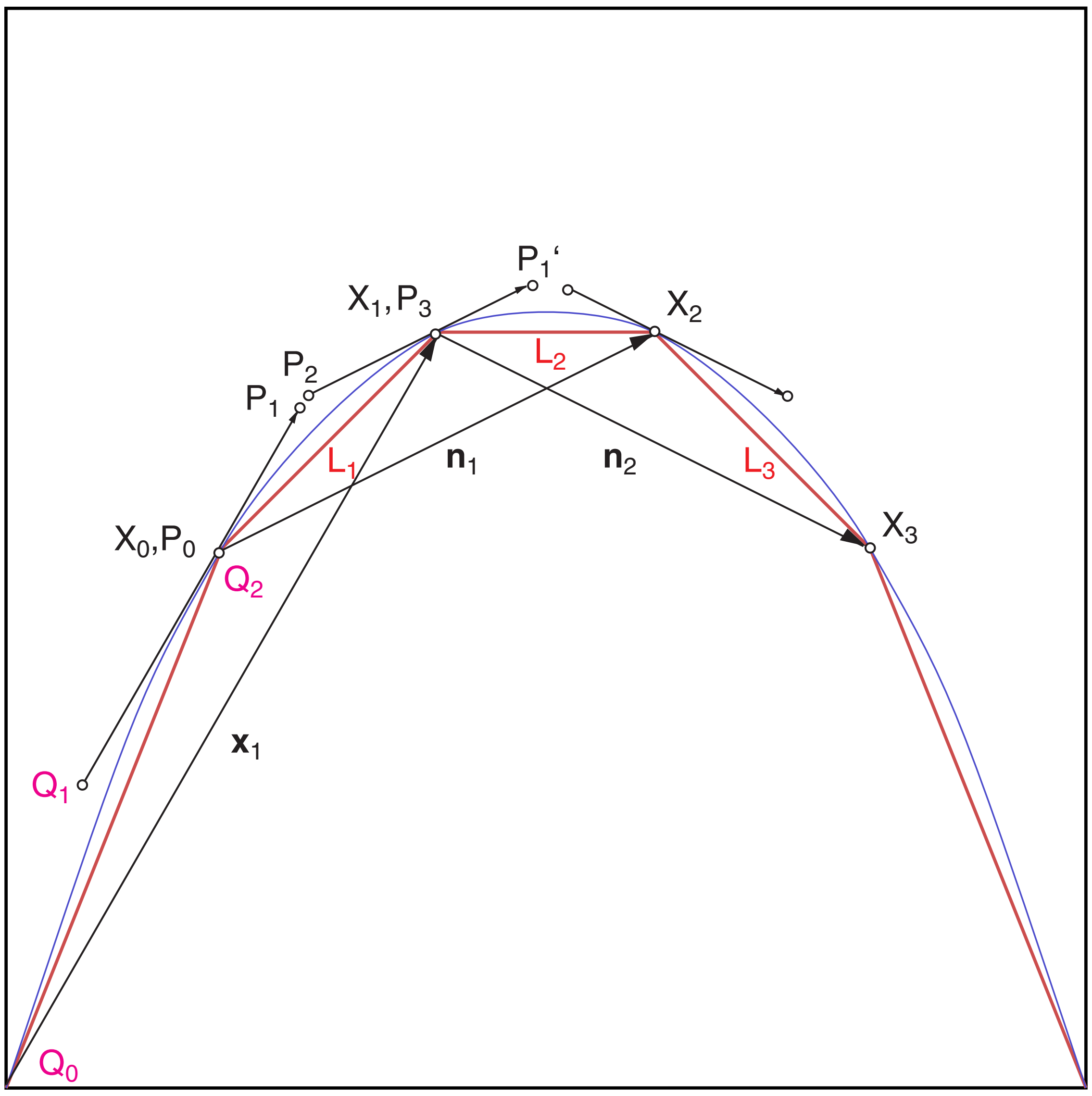
2.1 Bézier Polyline Interpolation

The task: replace a polyline (red) by a smooth function (blue), using PostScript Bézier interpolation.

M.Shemanarev [4] had described a very simple method for finding reasonable control points for the Bézier construction. The result is not as good as a spline interpolation but much easier to program.

We consider a sequence of points X_0, X_1, X_2, X_3 or vectors $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. X_0 is the first point of the *actual* sequence.

1. Connect X_0 and X_2 by $\mathbf{n}_1$.
2. Move $\mathbf{n}_1$ to X_1 and multiply by a factor, mostly $K_f = 0.3 \ldots 0.4$.
3. Shift the center of $\mathbf{n}_1$ along the tangent direction according to the proportion of the lengths L_1 and L_2.
   This delivers the control points P_2 and P_1‘ (this one for the next segment).
4. P_1 was already calculated in the previous step.
5. The control points for the actual Bézier segment are P_0, P_1, P_2 and P_3, from X_0 to X_1.



At the end segments one control point is missing. The point Q1 was already calculated by the mentioned method as P_2 (by local numbering).
Q_0, Q_1 and Q_2 are now considered as the control points of a *quadratic* Bézier polynomial.

It is possible to replace the quadratic polynomial $\mathbf{Q}(t)$ by a cubic polynomial $\mathbf{P}(t)$ which delivers the same curve. The conversion is necessary in order to draw the whole graph consistently, e.g. by Postscript *curveto*. Quadratic Bézier according to general laws [6].

$$\mathbf{Q}(t) = (\mathbf{Q}_0 - 2\mathbf{Q}_1 + \mathbf{Q}_2)t^2 + (2\mathbf{Q}_1 - 2\mathbf{Q}_0)t + \mathbf{Q}_0$$

$$\mathbf{P}(t) = (\mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0)t^3 + (3\mathbf{P}_2 - 6\mathbf{P}_1 + 3\mathbf{P}_0)t^2 + (3\mathbf{P}_1 - 3\mathbf{P}_0)t + \mathbf{P}_0$$

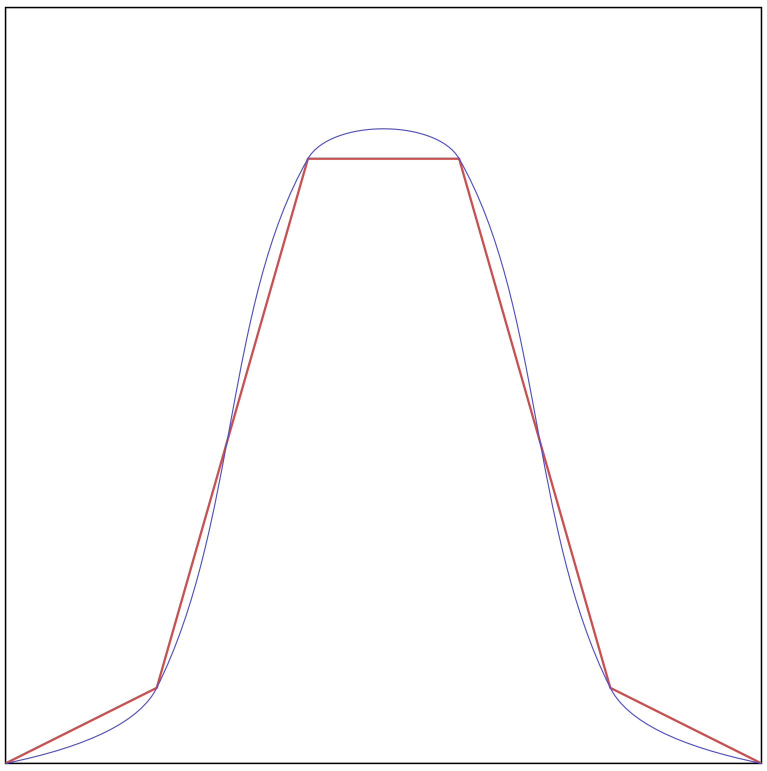
For $\mathbf{P}_0 = \mathbf{Q}_0$ and $\mathbf{P}_3 = \mathbf{Q}_2$ one can find $\mathbf{P}_1$ and $\mathbf{P}_2$ by matching the coefficients for t^2 and t^1 in both formulas:

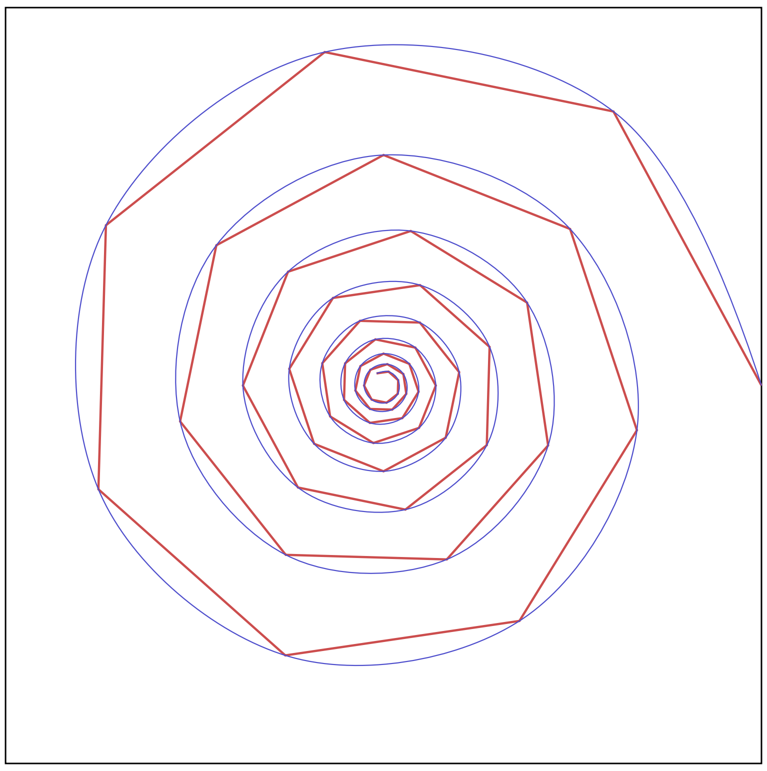
$$\mathbf{P}_1 = (\mathbf{Q}_0 + 2\mathbf{Q}_1)/3$$

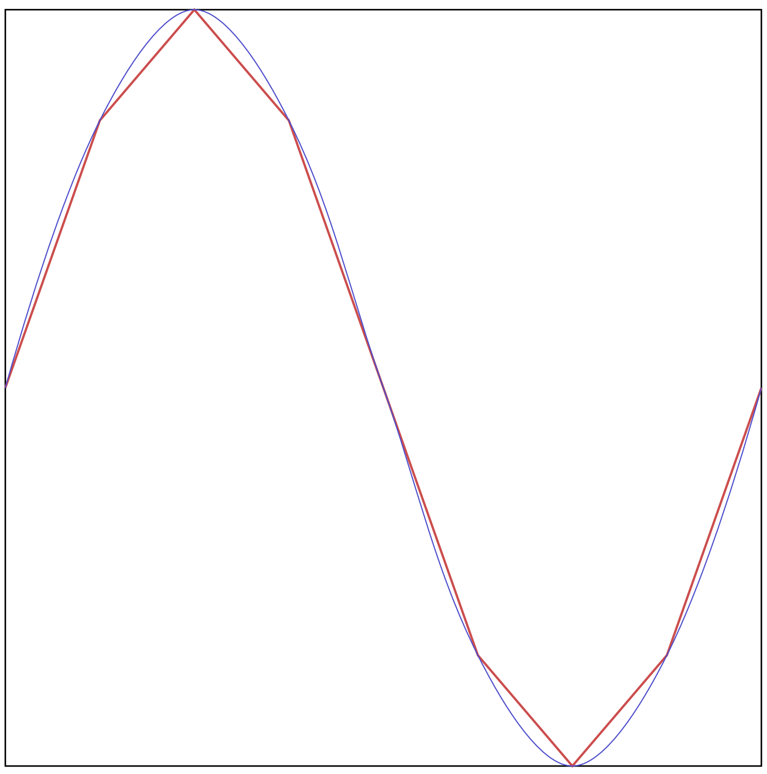
$$\mathbf{P}_2 = (\mathbf{Q}_2 + 2\mathbf{Q}_1)/3$$

2.2 Bézier Polyline Interpolation

```
%!PS-Adobe-3.0   EPSF-3.0
%%BoundingBox:   0 0 341 341
%%Creator:       Gernot Hoffmann
%%Title:         Inter-01
%%CreationDate:  Nov.06 / 2006

% Bezier through polyline

/mm { 2.834646 mul } def

/Typ 1 def

/Typ-01 % Bell
{
/ci [ 0 -1.0 -1.0
      1 -0.6 -0.8
      2 -0.2 +0.6
      3 +0.2 +0.6
      4 +0.6 -0.8
      5 +1.0 -1.0 ] def
} def

/Typ-02 % Spiral
{
/N 60 def
/a 0 def
/ci N 1 add 3 mul array def
/k 0 def
0 1 N
{pop
/R 3 a -0.001 mul exp def
/x R a cos mul def
/y R a sin mul def
/a a 50 add def
 ci k k put
 ci k 1 add x put
 ci k 2 add y put
/k k 3 add def
} for
} def

/Typ-03 % Sine
{
/N 8 def
/a 0 def
/ci N 1 add 3 mul array def
/k 0 def
0 1 N
{pop
/R 1 def
/x a 180 div 1 sub def
/y R a sin mul def
/a a 45 add def
 ci k k put
 ci k 1 add x put
 ci k 2 add y put
/k k 3 add def
} for
} def

Typ 1 eq {Typ-01} if
Typ 2 eq {Typ-02} if
Typ 3 eq {Typ-03} if

/sx  50 mm def
/xc  60 mm def
/yc  60 mm def

/Box
{ 0 setgray
  0.2 mm sx div setlinewidth
  newpath
  -1  -1 moveto 2 0 rlineto 0 2 rlineto -2 0 rlineto
  closepath
  stroke
} def
```







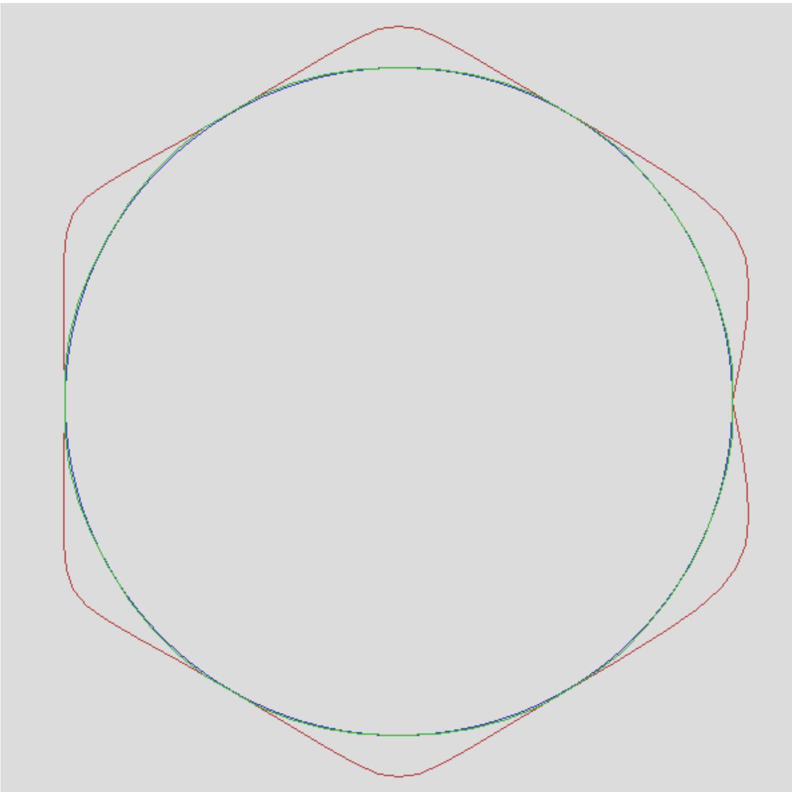
2.3 Bézier Polyline Interpolation

```
/Polyline
{ newpath
  0.8 0.3 0.3 setrgbcolor
  0.3 mm sx div setlinewidth
 /k 0 def
 /x0 ci 1 get def
 /y0 ci 2 get def
  x0 y0 moveto
  1 1 ci length 3 div 1 sub
  { /k  k 3 add def
    /x1 ci k 1 add get def
    /y1 ci k 2 add get def
     x1 y1 lineto
  } for
  stroke
} def

/Bezier
{ % Table ci: parameter,x,y
  % Reconstruction of two control points at the end segments
  % from one available control point at each end
  % Optimized for readabilty
  0.3 0.3 0.8 setrgbcolor
  0.15 mm sx div setlinewidth
 /kf 0.35 def  % 0.3 ..0.5; optimized for this example
 /N ci length 3 idiv 3 sub def
 /k 0 def
 /x0 ci 1 get def  /y0 ci 2 get def
 /x1 ci 4 get def  /y1 ci 5 get def
 /x2 ci 7 get def  /y2 ci 8 get def
 /n1x x2 x0 sub kf mul def
 /n1y y2 y0 sub kf mul def
 /L1  x1 x0 sub dup mul y1 y0 sub dup mul add sqrt def
 /L2  x2 x1 sub dup mul y2 y1 sub dup mul add sqrt def
 /q2x x1 L1 L1 L2 add div n1x mul sub def
 /q2y y1 L1 L1 L2 add div n1y mul sub def
 /p1x x0 q2x 2 mul add 3 div def % Reconstruction
 /p1y y0 q2y 2 mul add 3 div def
 /p2x x1 q2x 2 mul add 3 div def
 /p2y y1 q2y 2 mul add 3 div def
  newpath
  x0 y0 moveto
  p1x p1y p2x p2y x1 y1 curveto
  1 1 N
 {  pop
   /x3  ci k 10 add get def
   /y3  ci k 11 add get def
   /n2x x3 x1 sub kf mul def
   /n2y y3 y1 sub kf mul def
   /L3  x3 x2 sub dup mul y3 y2 sub dup mul add sqrt def
   /p1x x1 L2 L1 L2 add div n1x mul add def
   /p1y y1 L2 L1 L2 add div n1y mul add def
   /p2x x2 L2 L3 L2 add div n2x mul sub def
   /p2y y2 L2 L3 L2 add div n2y mul sub def
    p1x p1y p2x p2y x2 y2 curveto
   /k k 3 add def
   /x1  x2  def  /y1  y2  def /x2  x3  def  /y2  y3  def
   /n1x n2x def  /n1y n2y def /L1  L2  def  /L2  L3  def
  } for
/q1x x1 L2 L1 L2 add div n1x mul add def
/q1y y1 L2 L1 L2 add div n1y mul add def
/p2x x2 q1x 2 mul add 3 div def % Reconstruction
/p2y y2 q1y 2 mul add 3 div def
/p1x x1 q1x 2 mul add 3 div def
/p1y y1 q1y 2 mul add 3 div def
 p1x p1y p2x p2y x2 y2 curveto
 stroke
} def

xc yc translate
sx sx scale

Box
Polyline
Bezier
showpage
```

3.1 Bézier Optimization / Circle

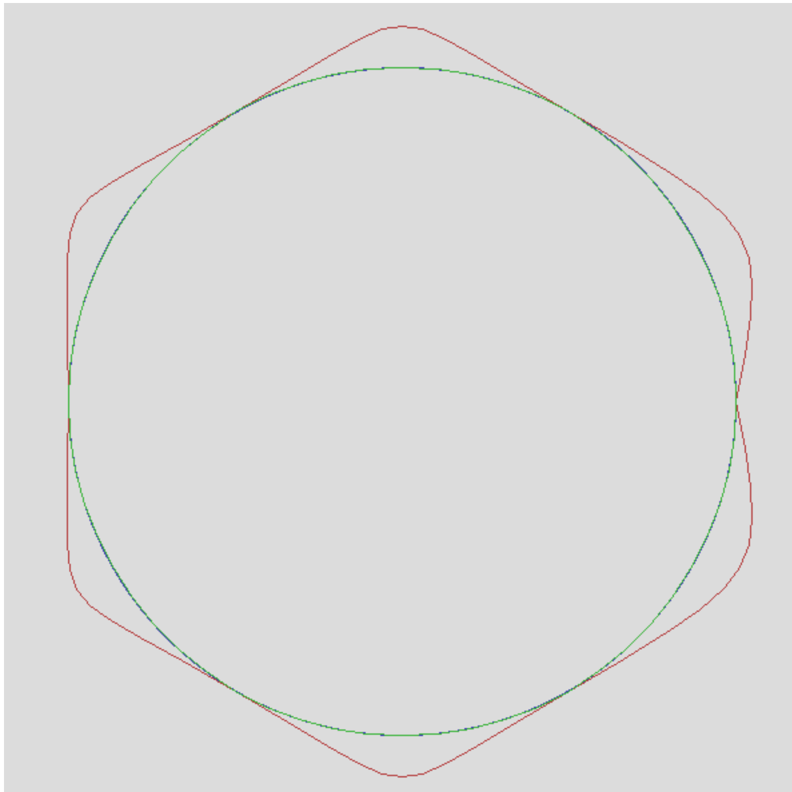
A graph F(x,y)=0 is described in parameter form by two analytical functions x=x(s), y=y(s) for s=0 to 1 . Alternatively a table with many entries could be used. The graph shall be replaced by a small number n of Bézier segments. Each segment uses four control points P_0, P_1, P_2, P_3, where P_0 and P_3 are on the graph.
The control points P_1, P_2 are found by a parameter optimization for 4·n unknowns x_i, y_i .

The blue curve is the original graph. The red curve shows the Bézier spline by initial control points. The green curve shows the result after 5 and after 12 iterations for final accuracy.



5 loops

Best view
Zoom
100%



12 loops

Circle by six segments

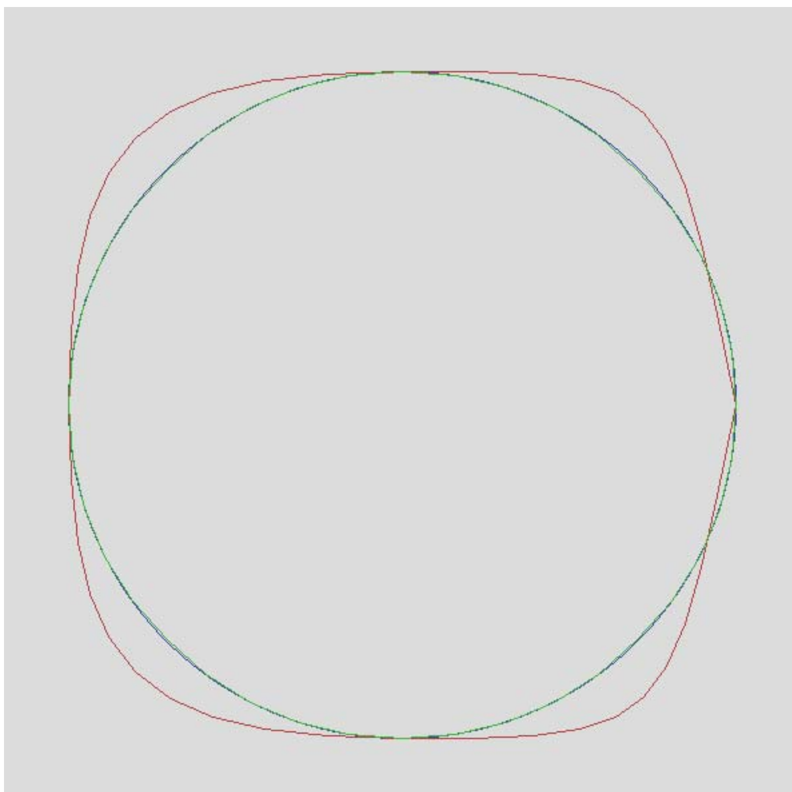
C:\Bezlist.txt   Data from ZBezier

i	x0	y0	x1	y1	x2	y2	x3	y3
0	1.0000	0.0000	1.0032	0.3505	0.8052	0.6935	0.5000	0.8660
1	0.5000	0.8660	0.1980	1.0441	-0.1980	1.0441	-0.5000	0.8660
2	-0.5000	0.8660	-0.8052	0.6935	-1.0032	0.3505	-1.0000	-0.0000
3	-1.0000	-0.0000	-1.0032	-0.3505	-0.8052	-0.6935	-0.5000	-0.8660
4	-0.5000	-0.8660	-0.1980	-1.0441	0.1980	-1.0441	0.5000	-0.8660
5	0.5000	-0.8660	0.8052	-0.6935	1.0032	-0.3505	1.0000	0.0000
6	1.0000	0.0000						

Circle by four segments (refer also to page 12)

C:\Bezlist.txt   Data from ZBezier

i	x0	y0	x1	y1	x2	y2	x3	y3
0	1.0000	0.0000	1.0139	0.5343	0.5343	1.0139	-0.0000	1.0000
1	-0.0000	1.0000	-0.5343	1.0139	-1.0139	0.5343	-1.0000	-0.0000
2	-1.0000	-0.0000	-1.0139	-0.5343	-0.5343	-1.0139	0.0000	-1.0000
3	0.0000	-1.0000	0.5343	-1.0139	1.0139	-0.5343	1.0000	0.0000
4	1.0000	0.0000						



12 loops

3.2 Bézier Optimization / Code

```
Program ZBezier;
{ Project    Find optimal Bezier points for an analytical
              function x(s), y(s) for s=0..1
  Author     Gernot Hoffmann
  Date       December 19, 2002 }

Uses     Crt,Dos,
         Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
         Zefir36,Zefir37,Zefir38,Zefir39,Zefir40;
Var      smin,smax,sd,s2,eps: Double;
         xm,ym,scale,flag      : Integer;
         xa,ya,xe,ye           : Integer;
         pal,col,lmax,sel      : Integer;
         txt                   : String;
Const    n=6;     { 0..n fixpoints                  }
         m=10;    { subdivisions in one line segment }
Type     Beztyp = Record x0,y0,x1,y1,x2,y2,x3,y3 : Double; End;
         BezArr = Array [0..n] of Beztyp;
Var      cpt: BezArr;
Type     ParArr = Array [1..4*n] Of Double;
Var      pn: Parrr;

Procedure    FuncVal  (sel: Integer; s: Double; Var xf,yf: Double);       Forward;
Procedure    Params;                                                      Forward;
Procedure    FuncDer  (sel: Integer; s: Double; Var Xs,Ys: Double);       Forward;
Procedure    DrawFunc (sel,pal,col: Integer);                             Forward;
Procedure    ValBez   (cpi: beztyp; t: Double; Var xb,yb: Double);        Forward;
Procedure    DrawBez  (n,m: Integer; cpt: BezArr; pal,col: Integer);      Forward;
Procedure    ContBez  (sel,n: Integer; Var cpt: bezarr);                  Forward;
Procedure    CptToPn  (n: Integer; cpt: BezArr; Var pn: Pararr);          Forward;
Procedure    PnToCpt  (n: Integer; pn: ParArr; Var cpt: BezArr);          Forward;
Function     ErrFun   (sel,sn: Integer; pn: ParArr): Double;              Forward;
Procedure    Steepest (sel,sn: Integer; Var pn:ParArr; Var lmax:Integer); Forward;


Procedure FuncVal(sel: Integer; s: Double; Var xf,yf: Double);
Const    pi2:Double=2*pi;
Begin
Case sel Of
0: Begin
    xf:=s;
    yf:=s;
   End;
1: Begin { Circle }
    xf:=coc(pi2*s);
    yf:=sic(pi2*s)
   End;
2: Begin { Spiral }
    xf:=(1-0.1*s)*coc(pi2*s);
    yf:=(1-0.1*s)*sic(pi2*s)
   End;
 End;
End;

Procedure Params;
Begin
  eps:=1E-4;
  xm:=xpx div 2;
  ym:=ypx div 2;
  scale:=250;                 { 1.0 -> scale pixels               }
  smin:= 0;                   { Parameter range                   }
  smax:=+1.00;
  sd:=(smax-smin)*eps;        { Increment for num. differentiation }
  s2:=2*sd/scale;
End;

Procedure FuncDer(sel: Integer; s: Double; Var Xs,Ys: Double);
Var x1,y1,x2,y2: Double;
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;
```

3.3 Bézier Optimization / Code

```
Procedure DrawFunc(sel,pal,col: Integer);
{    Draw analytical function by dense pixel sequence }
Var  xf,yf,s,ds,sx,sy,Xs,Ys,aXs,aYs     : Double;
     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,pal,col);
While (s<smax) And (k<kmax) Do
 Begin
 FuncDer(sel,s,Xs,Ys);
 aXs:=Abs(Xs);
 aYs:=Abs(Ys);
 If Abs(Xs)>=Abs(Ys) 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;
  If s>smax Then
  Begin
   FuncVal(sel,smax,xf,yf);
   px:=Round(scale*xf);
   py:=Round(scale*yf);
  End;
  SpcSixel(xm+px,ym-py,pal,col);
 Inc(k);
 End;
End;

Procedure ValBez (cpi: BezTyp; t: Double; Var xb,yb: Double);
{    4 control points p0,p1,p2,p3
     Calculate pb for t=0..1          }
Var  ax,bx,cx,ay,by,cy: Double;
Begin
 With cpi Do
 Begin
  cx:=(x1-x0)*3;
  bx:=(x2-x1)*3-cx;
  ax:= x3-x0-cx-bx;
  cy:=(y1-y0)*3;
  by:=(y2-y1)*3-cy;
  ay:= y3-y0-cy-by;
  xb:=((ax*t+bx)*t+cx)*t+x0;
  yb:=((ay*t+by)*t+cy)*t+y0;
 End;
End;

Procedure DrawBez(n,m: Integer; cpt: BezArr; pal,col: Integer);
{    Stroke path
     n subdivisions for control points
     m subdivisions per segment         }
Var  xb,yb,t,dt    : Double;
     p0,q0,p1,q1   : Integer;
     i,j           : Integer;
Begin
dt:=1/m;
For i:=0 to n-1 Do
 Begin
  With cpt[i] Do
  Begin
  p0:=xm+Round(scale*x0);
  q0:=ym-Round(scale*y0);
```

```
  End;
  t:=0;
  For j:=0 to m-1 Do
   Begin
   t:=t+dt;
   ValBez(cpt[i],t,xb,yb);
   p1:=xm+Round(scale*xb);
   q1:=ym-Round(scale*yb);
   MakeSline(p0,q0,p1,q1,pal,col);
   p0:=p1;
   q0:=q1;
   End;
 End;
End;

Procedure ContBez(sel,n: Integer; Var cpt: BezArr);
{    Function in parameter form xf(s),yf(s) for s=0..1
     Calculate fixpoints and approximations for other control points
     sel      Function selector
     n        Subdivisions for i=0..n fixpoints
     cpt      Array of Beztyp     }
Var  i,k              : Integer;
     dx,dy,s,ds,xf,yf : Double;
Begin
ds:=1/n;
s:=0;
With cpt[0] Do FuncVal(sel,s,x0,y0);
For i:=1 to n Do
 Begin
  s:=s+ds;
  FuncVal(sel,s,xf,yf);
  With cpt[i] Do
   Begin x0:=xf; y0:=yf;
   End;
  With cpt[i-1] Do
   Begin x3:=xf; y3:=yf;
   End;
 End;
For i:=1 to n-1 Do
 Begin
  dx:=0.4*(cpt[i+1].x0-cpt[i-1].x0);
  dy:=0.4*(cpt[i+1].y0-cpt[i-1].y0);
  With cpt[i] Do
  Begin
   cpt[i  ].x1:=x0+dx;
   cpt[i  ].y1:=y0+dy;
   cpt[i-1].x2:=x0-dx;
   cpt[i-1].y2:=y0-dy;
  End;
 End;
With cpt[0] Do
 Begin x1:=x2; y1:=y2;
 End;
With cpt[n-1] Do
 Begin x2:=x1; y2:=y1;
 End;
End;

Procedure CptToPn (n: Integer; cpt: BezArr; Var pn: ParArr);
{    Copy control points to parameter vector }
Var  i,k: Integer;
Begin
k:=1;
For i:=0 to n-1 Do
 Begin
  With cpt[i] Do
  Begin
   pn[k]:=x1; pn[k+1]:=y1; pn[k+2]:=x2; pn[k+3]:=y2;
  End;
  Inc(k,4);
 End;
End;
```

3.5 Bézier Optimization / Code

```
Procedure PnToCpt(n: Integer; pn: ParArr; Var cpt: BezArr);
{    Copy parameter vector to control points }
Var i,k: Integer;
Begin
k:=1;
For i:=0 to n-1 Do
 Begin
  With cpt[i] Do
  Begin
   x1:=pn[k]; y1:=pn[k+1]; x2:=pn[k+2]; y2:=pn[k+3];
  End;
  Inc(k,4);
 End;
End;

Function ErrFun(sel,sn: Integer; pn: ParArr): Double;
{    Sum of squares }
Var t,dt,s,ds,err,xb,yb,xf,yf : Double;
    i,j                       : Integer;
Begin
dt:=1/m; ds:=dt/n; s:=0; err:=0;
PntoCpt(n,pn,cpt);
For i:=0 to n-1 Do
 Begin
  t:=0;
  For j:=0 to m-1 Do
   Begin
   s:=s+ds; t:=t+dt;
   FuncVal(sel,s,xf,yf);
   ValBez (cpt[i],t,xb,yb);
   err:=err+Sqr(xb-xf)+Sqr(yb-yf);
   End;                                            No square root !
 End;
ErrFun:=err;
End;

Procedure Steepest (sel,sn: Integer; Var pn: ParArr; Var lmax:Integer);
{    Minimizes ErrFun for sn variables in parameter vector pn }
Const    eps=1E-16;   { Stop condition          }
         h  =1E-6;    { Differentiation step    }
Var      i,j,jopt,lup                         : Integer;
         z1,z2,z3,den,lam,lum,lem,dif,lim     : Double;
         qn,zx                                : ParArr;
Const jmax=10;
Begin
 lup:=0; { Loop counter }
 lam:=1; { Step control }
 z2:=ErrFun(sel,sn,pn);
 Repeat
  z1:=z2; Inc(lup);
  For i:=1 To sn Do
   Begin
   qn[i]:=pn[i]; pn[i]:=pn[i]+h;
   zx[i]:=(ErrFun(sel,sn,pn)-z1)/h; pn[i]:=qn[i];
   End;
  den:=1E-16; { Denominator offset }
  For i:=1 To sn Do den:=den+Sqr(zx[i]) ;
  lem:=z1/den; lum:=lam*lem;
  For i:=1 To sn Do pn[i]:=qn[i]-lum*zx[i];
  z2:=ErrFun(sel,sn,pn); dif:=z2-z1;
  If dif<0 Then
  Begin { OneDim }
    lim:=lum/jmax; jopt:=jmax;
    For j:=1 To jmax-1 Do
     Begin
     For i:=1 To sn Do pn[i]:=qn[i]-j*lim*zx[i];
     z3:=ErrFun(sel,sn,pn);
     If z3<z2 Then Begin z2:=z3; jopt:=j; End;
     End;
     For i:=1 To sn Do pn[i]:=qn[i]-jopt*lim*zx[i];
  lam:=7*lam;
  End
  Else
```

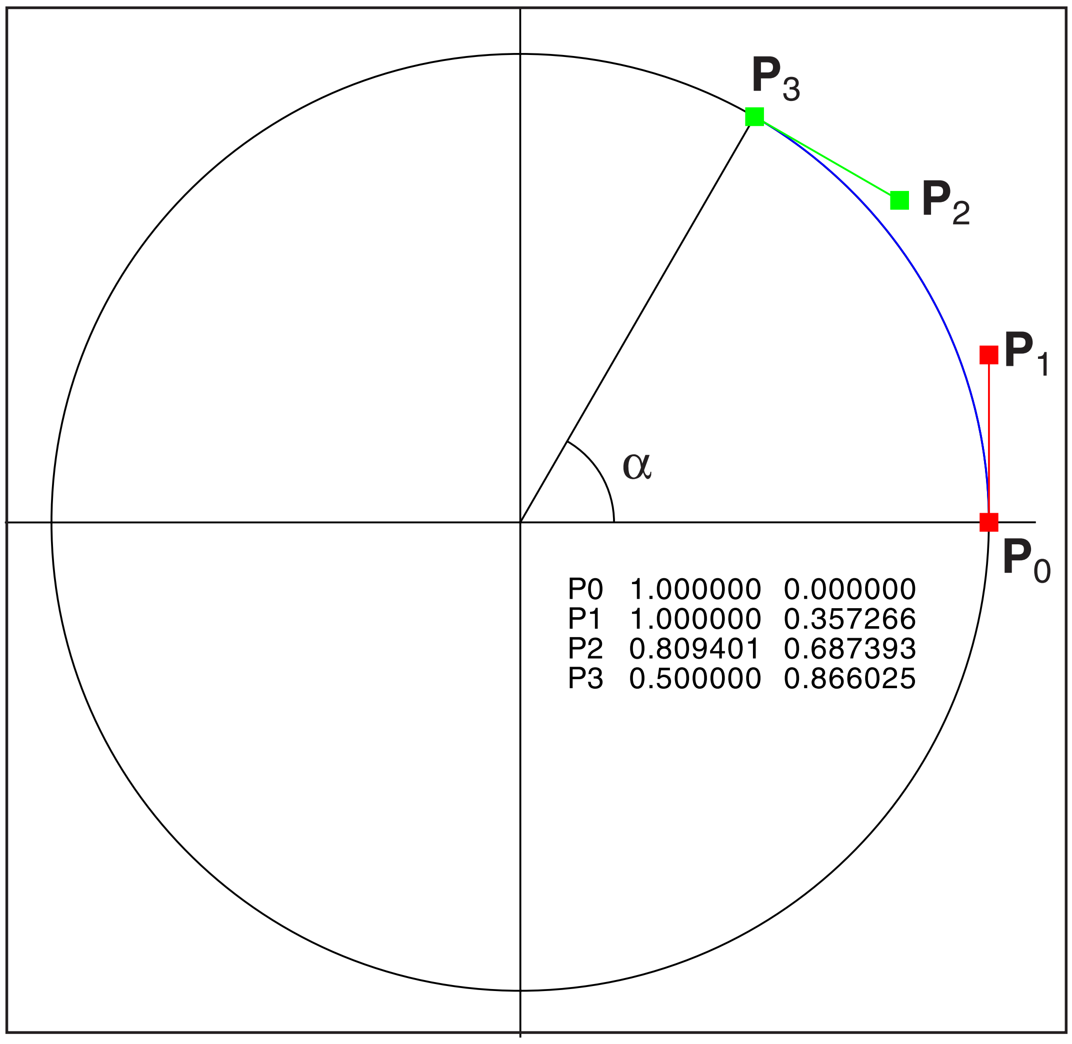
3.6 Bézier Optimization / Code

```
  Begin
   lam:=0.7*lam;
   While z2>=z1 Do
   Begin
    lum:=lam*lem;
    For i:=1 To sn Do pn[i]:=qn[i]-lum*zx[i];
    z2:=ErrFun(sel,sn,pn); lam:=0.7*lam;
    If lam<0.01 Then dif:=0; { Stop if step too small}
   End;
  End;
  Until (lup=lmax) Or (Abs(dif)<eps);
lmax:=lup;
End; { Steepest }

Procedure WriteCpt(n: Integer; cpt: BezArr);
Var      BezList  : Text;
         FileName : String;
         i,ioerr  : Integer;
Begin
FileName:='C:\Bezlist.dat';
Assign(BezList,FileName);
{$I-}ReWrite (BezList);{$I+}
 ioerr:=IoResult;
 If ioerr= 0 Then
  Begin
  WriteLn(BezList,FileName+' Data from ZBezier');
  WriteLn(BezList,'   i  x0      y0      x1      y1      x2      y2      x3      y3');
  For i:=0 to n-1 Do
   Begin
   With cpt[i] Do
    Begin
     WriteLn(BezList,i:4,x0:8:4,y0:8:4,x1:8:4,y1:8:4,x2:8:4,y2:8:4,x3:8:4,y3:8:4);
    End;
   End;
   With cpt[n] Do
    Begin
     WriteLn(BezList,n:4,x0:8:4,y0:8:4);
    End;
  Close(BezList);
  End;
End;

BEGIN
VesaMode:=Vmode42;
VesaCode:=$0115;
VesaStart(VesaMode);
MemGStart;
Params;
sel:=2;
ColToScr(181,220);
DrawFunc(sel,120,120);
ContBez (sel,n,cpt);
DrawBez (n,m,cpt,0,120);
CptToPn (n,cpt,pn);
lmax:=100;
Steepest(sel,4*n,pn,lmax);
PnToCpt (n,pn,cpt);
DrawBez (n,m,cpt,60,120);
Str(lmax:3,txt); txt:=txt+' loops';
WrTxtWxy(0,blac,10,580,txt);
SaveImag('H:\DrawF\DrawF602.BMP');
WriteCpt(n,cpt);
Stop;
MemGEnde;
VesaEnde;
END.
```

3.7.1 Bézier Optimization / Circle

A Bézier curve for a circle segment is found by adjusting the tangent lengths k so that the Bézier curve hits the circle at $\alpha/2$ for t=0.5. The algorithm is very accurate for angles up to 90°.

For α=90° we get k=0.552285.



Control points for a unit circle segment.

Tangent lengths k:

$$\mathbf{P}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{P}_1 = \begin{bmatrix} 0 \\ k \end{bmatrix} + \mathbf{P}_0 = \begin{bmatrix} 1 \\ k \end{bmatrix}$$

$$\mathbf{P}_3 = \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}$$

$$\mathbf{P}_2 = \begin{bmatrix} +k\sin(\alpha) \\ -k\cos(\alpha) \end{bmatrix} + \mathbf{P}_3$$

$$\mathbf{P}_1 - \mathbf{P}_0 = \begin{bmatrix} 0 \\ k \end{bmatrix}$$

$$\mathbf{P}_3 - \mathbf{P}_2 = \begin{bmatrix} -k\sin(\alpha) \\ +k\cos(\alpha) \end{bmatrix}$$

$$\mathbf{P}_3 - \mathbf{P}_0 = \begin{bmatrix} \cos(\alpha) - 1 \\ \sin(\alpha) \end{bmatrix}$$

Formulas in chapter 1.

$$\mathbf{a} = \begin{bmatrix} -3k\sin(\alpha) - 2\cos(\alpha) + 2 \\ +3k\cos(\alpha) - 2\sin(\alpha) + 3k \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} +3k\sin(\alpha) + 3\cos(\alpha) - 3 \\ -3k\cos(\alpha) + 3\sin(\alpha) - 6k \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ 3k \end{bmatrix}$$

Match at t=0.5 on the circle at $\alpha/2$:

$$\mathbf{x}(0.5) = \begin{bmatrix} +\frac{3}{8}k\sin(\alpha) + \frac{1}{2}\cos(\alpha) + \frac{1}{2} \\ -\frac{3}{8}k\cos(\alpha) + \frac{1}{2}\sin(\alpha) + \frac{3}{8}k \end{bmatrix} = \begin{bmatrix} \cos(\frac{\alpha}{2}) \\ \sin(\frac{\alpha}{2}) \end{bmatrix}$$

$$k = \frac{8\cos(\frac{\alpha}{2}) - 4(1+\cos(\alpha))}{3\sin(\alpha)}$$

$$k = \frac{8\sin(\frac{\alpha}{2}) - 4\sin(\alpha)}{3(1-\cos(\alpha))}$$

Division by zero for $\alpha \to 0$ can be avoided by a Taylor series approximation for small angles:

$\sin(\alpha) \approx \alpha$

$\cos(\alpha) \approx 1 - \frac{1}{2}\alpha^2$

This delivers

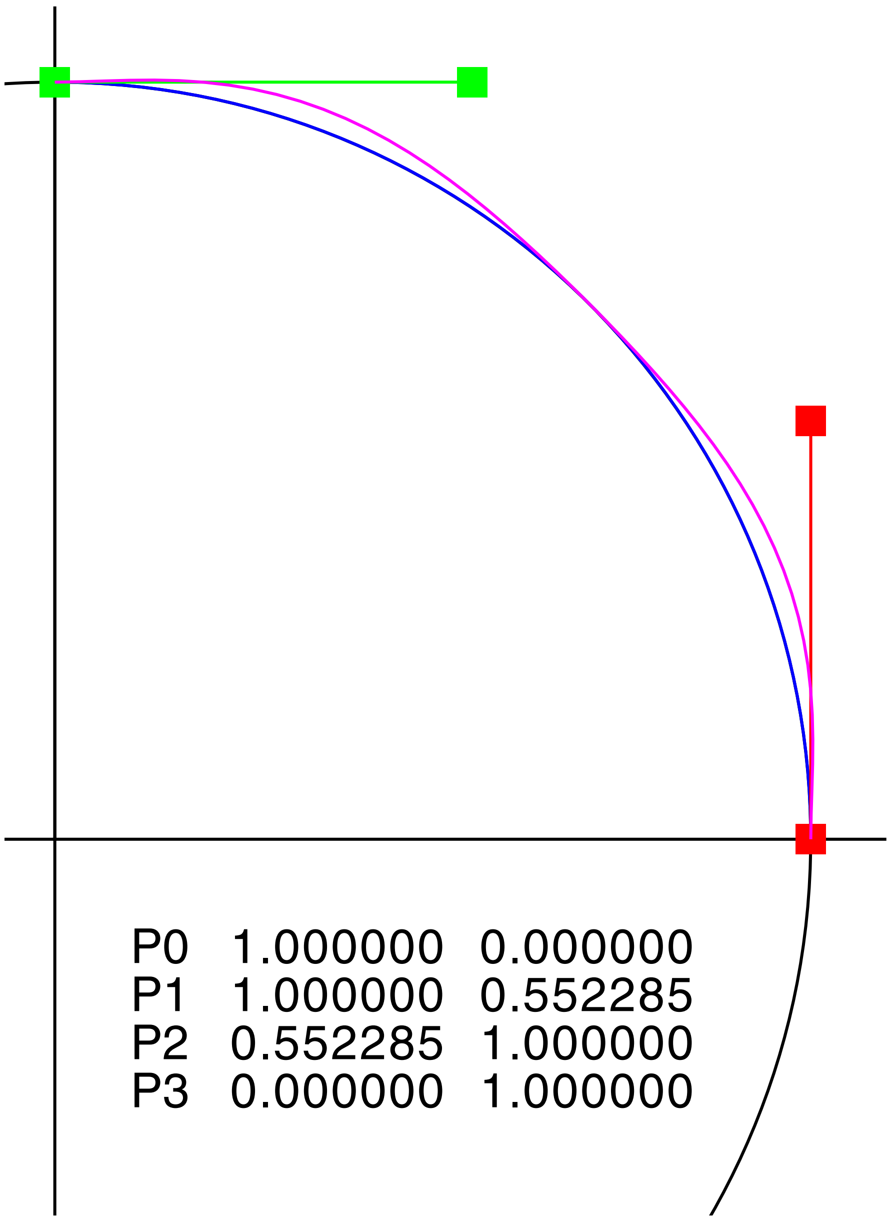
$k \approx \alpha/3$.

Each of the general formulas (left) can be simplified, thanks to *Hans Linders*:

$$k = \frac{4}{3}\tan(\frac{\alpha}{4})$$

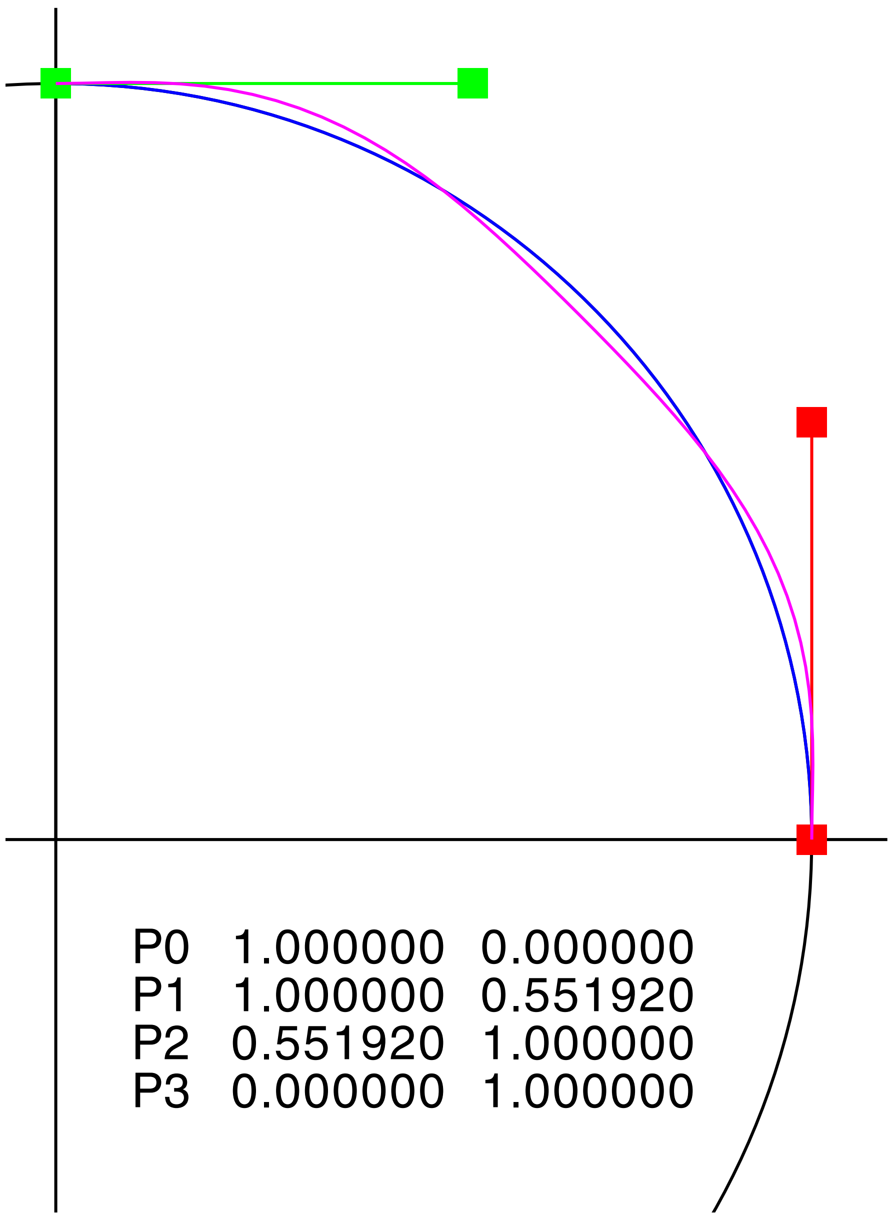
3.7.2 Bézier Optimization / Circle

Standard circle.
Magenta:
Radius error
multiplied by 100.



Param-t	Delta-R
0.000000	0.000000
0.020000	0.000010
0.040000	0.000037
0.060000	0.000072
0.080000	0.000112
0.100000	0.000153
0.120000	0.000190
0.140000	0.000221
0.160000	0.000246
0.180000	0.000263
0.200000	0.000271
0.220000	0.000272
0.240000	0.000265
0.260000	0.000251
0.280000	0.000232
0.300000	0.000208
0.320000	0.000181
0.340000	0.000152
0.360000	0.000123
0.380000	0.000094
0.400000	0.000068
0.420000	0.000045
0.440000	0.000026
0.460000	0.000012
0.480000	0.000003
0.500000	0.000000

Improved circle.
Magenta:
Radius error
multiplied by 100.

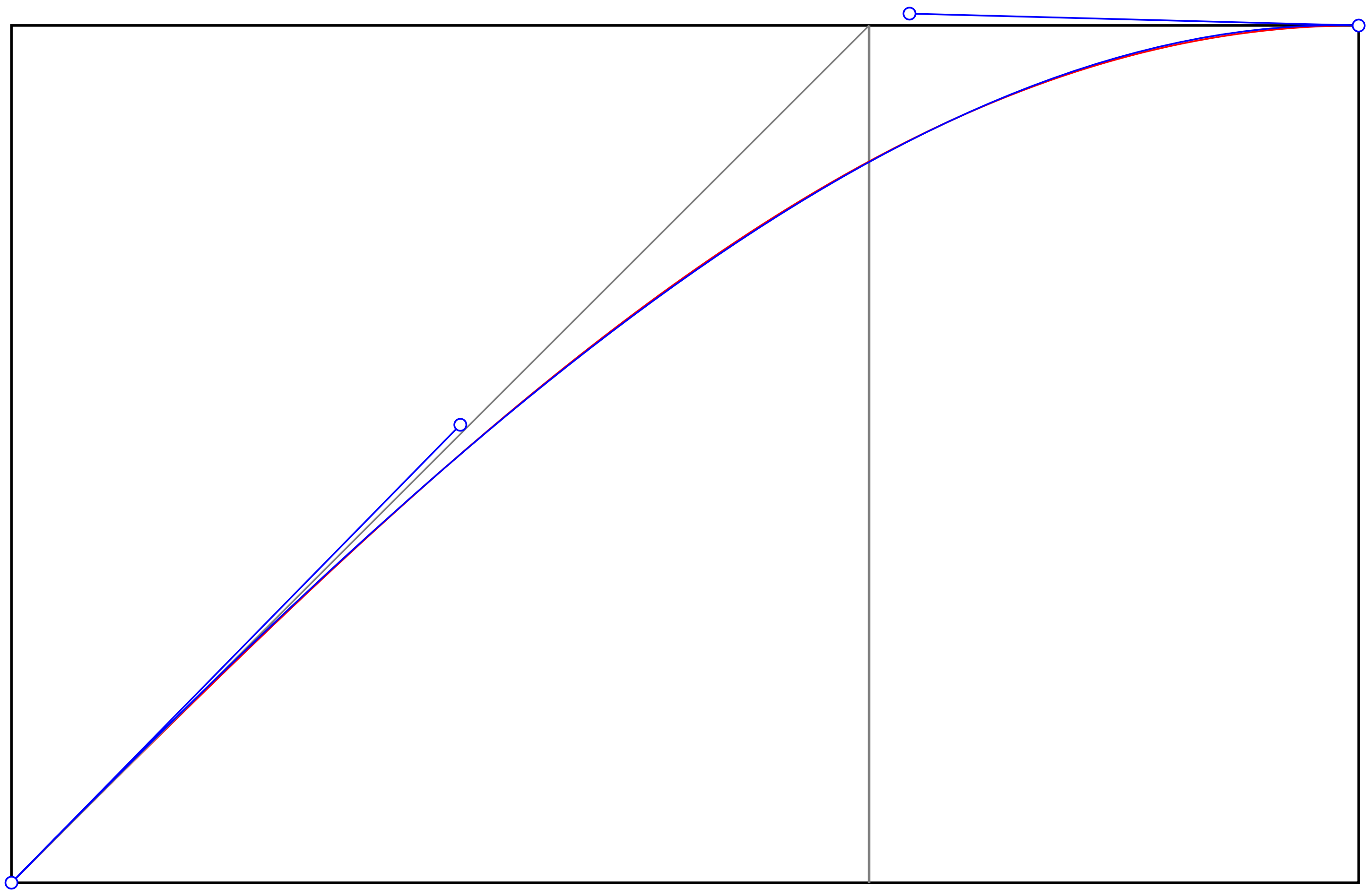


Param-t	Delta-R
0.000000	0.000000
0.020000	0.000009
0.040000	0.000032
0.060000	0.000063
0.080000	0.000096
0.100000	0.000129
0.120000	0.000156
0.140000	0.000178
0.160000	0.000191
0.180000	0.000197
0.200000	0.000194
0.220000	0.000182
0.240000	0.000164
0.260000	0.000138
0.280000	0.000107
0.300000	0.000072
0.320000	0.000035
0.340000	-0.000003
0.360000	-0.000041
0.380000	-0.000077
0.400000	-0.000110
0.420000	-0.000139
0.440000	-0.000162
0.460000	-0.000179
0.480000	-0.000190
0.500000	-0.000193

3.8 Bézier Optimization / Sine

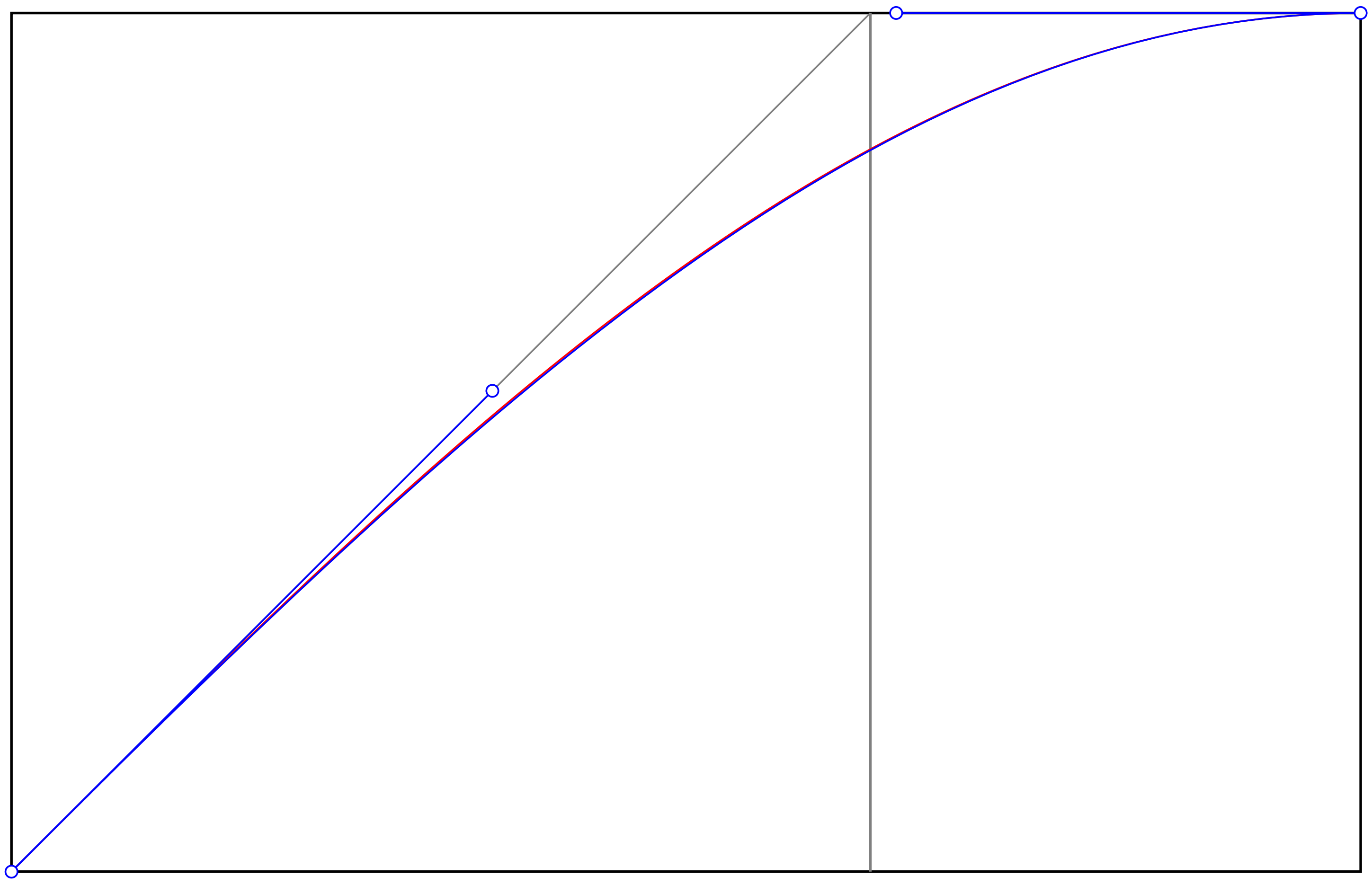
This is the best approximation for the function y = sin(x), the result of a numerical optimization. Input x in the range x = 0..π/2.

```
C:\Bezlist.txt Data from ZBezier
   i  x0      y0      x1      y1      x2      y2      x3      y3
   0  0.0000  0.0000  0.5234  0.5342  1.0472  1.0138  1.5708  1.0000
   1  1.5708  1.0000
```



This is a simplified version with tangents as expected:

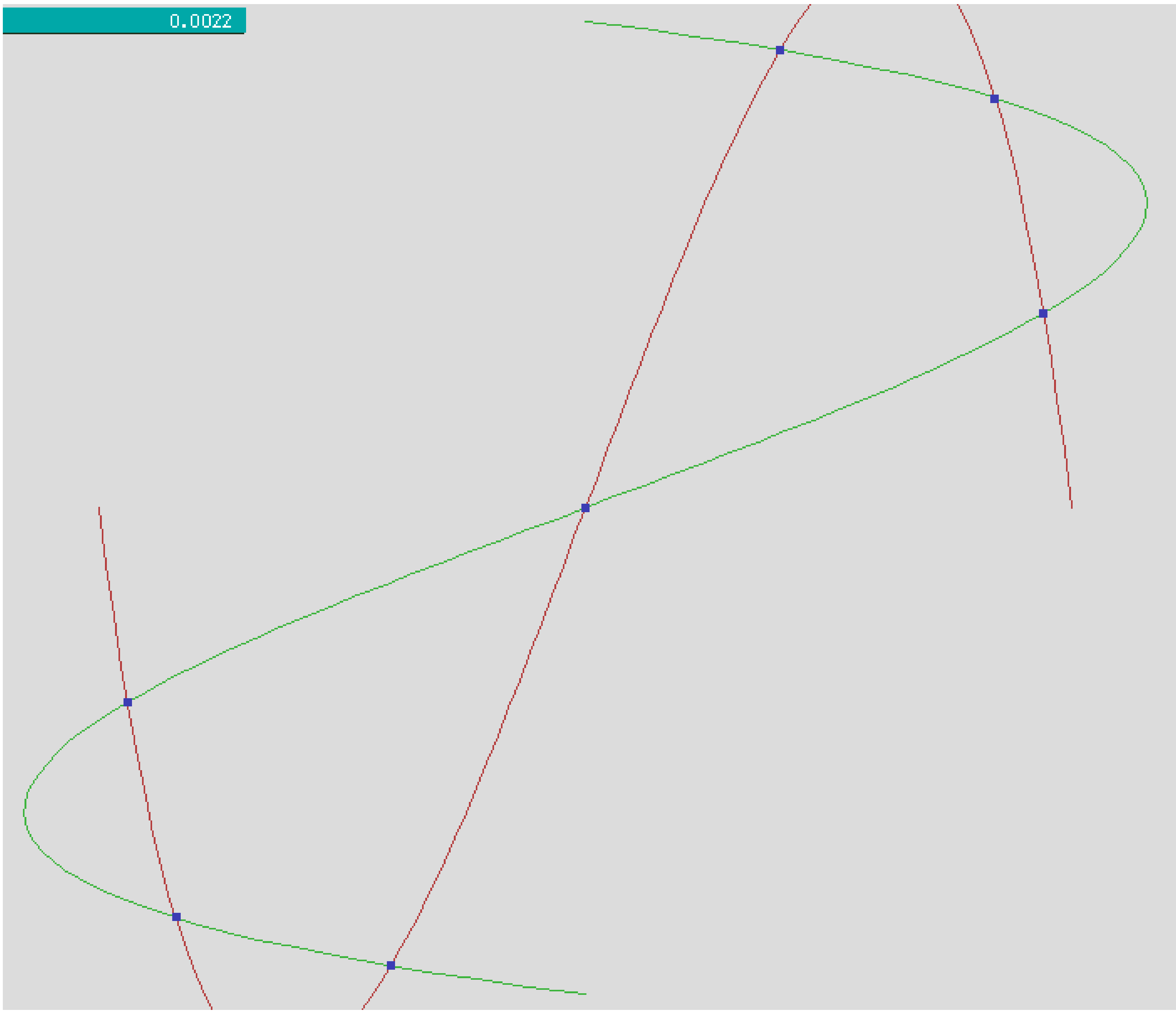
```
Modified
   i  x0      y0      x1      y1      x2      y2      x3      y3
   0  0.0000  0.0000  0.5600  0.5600  1.0300  1.0000  1.5708  1.0000
   1  1.5708  1.0000
```



4.1 Bézier Intersections

Two Bézier curves are replaced by polylines. Up to nine intersections can be found with reasonable accuracy.
An 'intersection' is defined by a true crossing of two line segments. Identical Béziers don´t have intersections. Overlapping line segments on the same line don´t have intersections. For m=100 subdivisions each, a Pentium P2 (400 MHz) needs (2.2 ... 5.0) ms for finding all intersections. The speed depends on the level of error checks and debug informations.

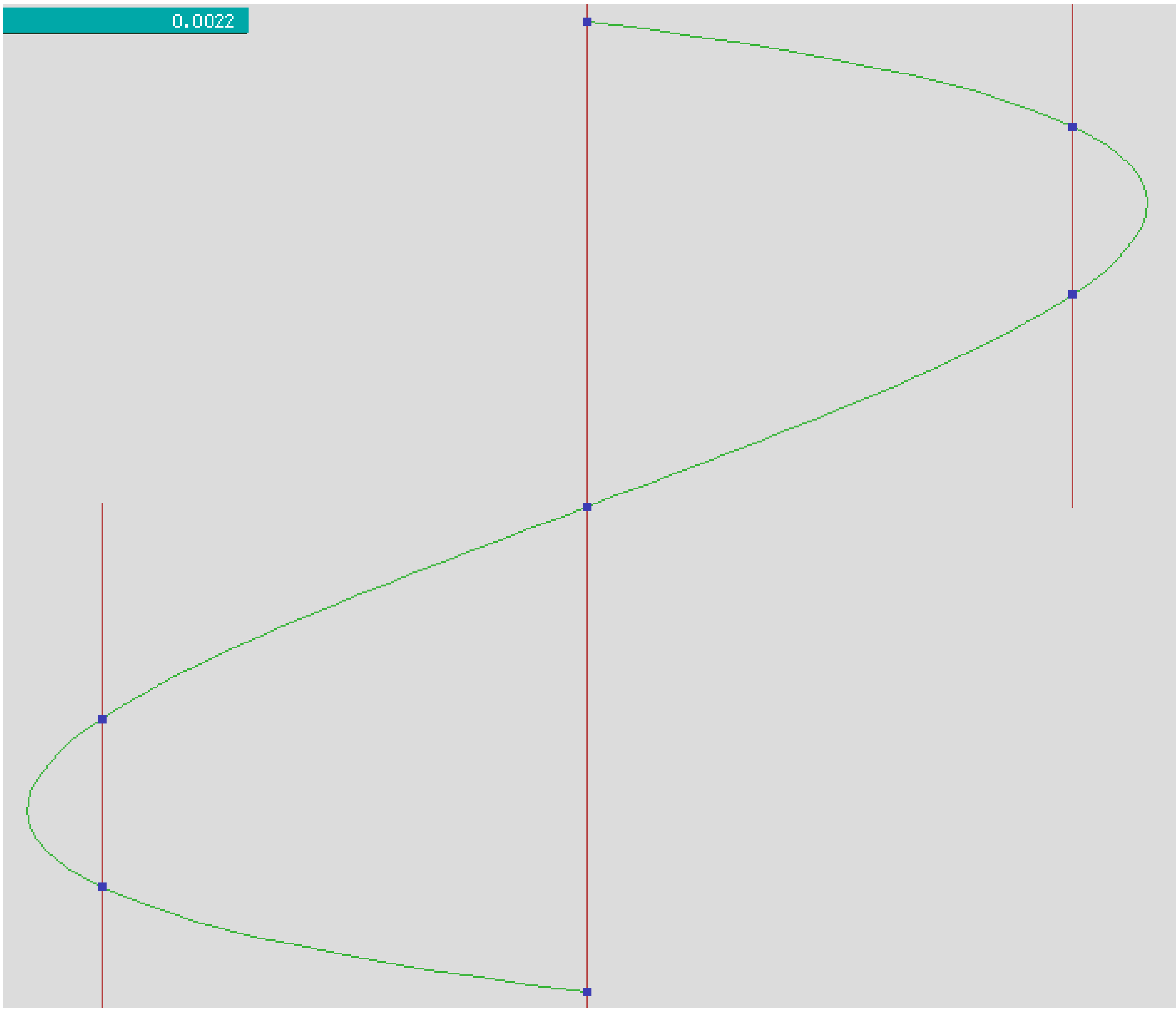
Best view zoom 100%



4.2 Bézier Intersections

Here we have shifted the control points P_1 and P_2 for the red curve in y-direction to y_1=+4E12 and y_2=-4E12. The solution is still accurate. Limits for accuracy and for the allowed range of values are shown in the source code.

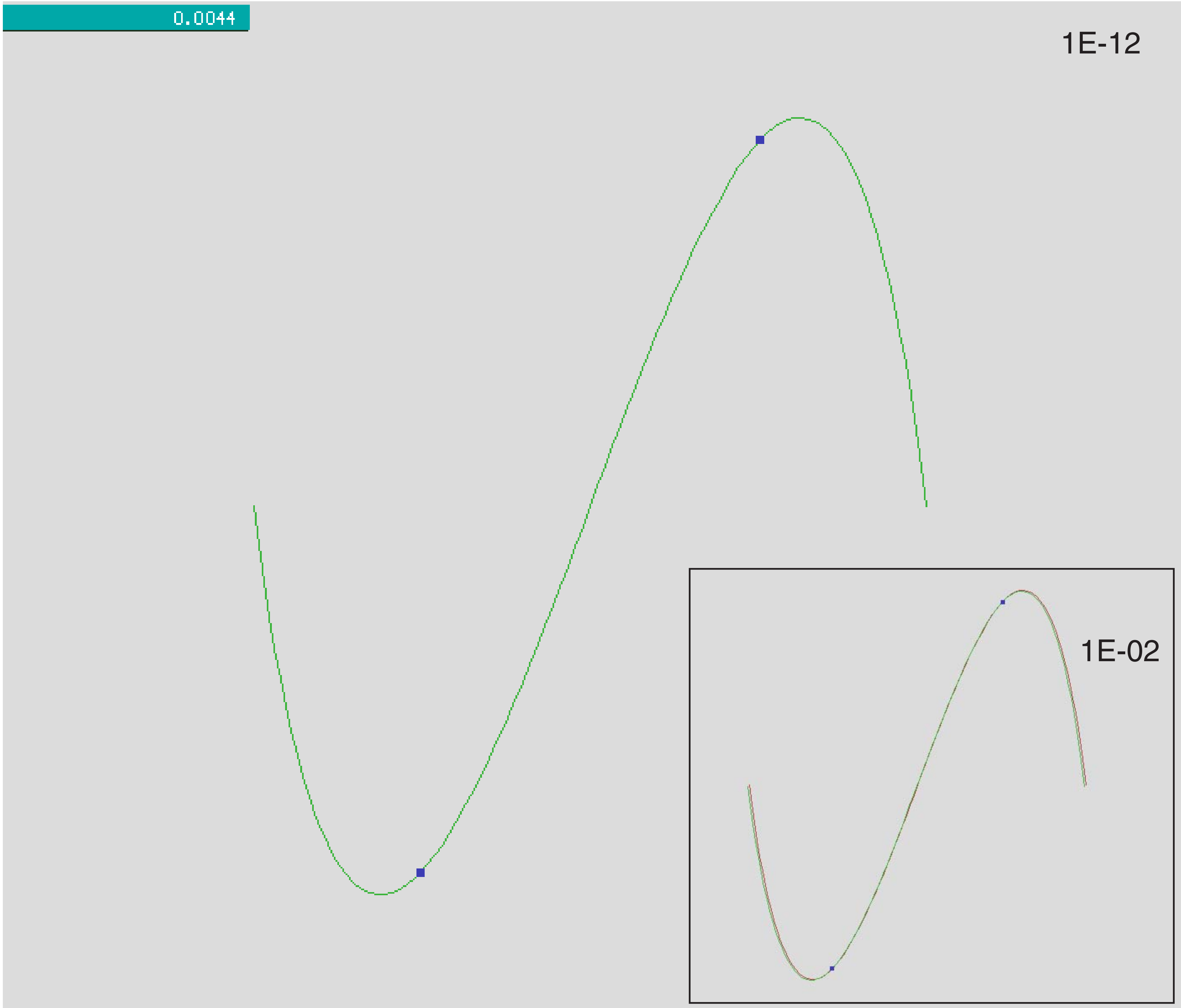
Best view zoom 100%



4.3 Bézier Intersections

Here we have shifted the endpoints of the red curve by 1E-12 symmetrically . The curves are nearly equal, therefore the red curve is not visible. The solution is still accurate. Limits for accuracy and for the allowed range of values are shown in the source code. The small image show as shift of 1E-2.

Best view zoom 100%




```
Procedure MakeBez(k:Integer; Var cpt: BezTyp);
Const C1: Double=1; C2: Double=1E-12;
{ Accurate for C2=1E-12, Safe for C2=1..1E-300   }
Begin
Case k Of
1: With cpt Do
   Begin
   x0:=1; y0:=0; x1:=0.50; y1:=4*C1; x2:=-0.5; y2:=-4*C1; x3:=-1; y3:=0;
   End;
2: With cpt Do
   Begin
   x0:=1-C2; y0:=-C2; x1:=0.50; y1:=4*C1; x2:=-0.5; y2:=-4*C1; x3:=-1-C2; y3:=-C2;
   End;
End;
End;
```

4.4 Bézier Intersections

```
Program ZBezier2;
{ Project    Find intersections of Bezier polynoms by polylines
  Author     Gernot Hoffmann
  Date       December 28, 2002 }

Uses     Crt,Dos,
         Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
         Zefir36,Zefir37,Zefir38,Zefir39,Zefir40;
Var      xm,ym,scale,flag : Integer;
         xa,ya,xe,ye      : Integer;
         pal,col,sel,k    : Integer;
         time1,time2      : Extended;
Const    m=100;  { subdivisions in one Bezier segment }
Type     BezTyp = Record x0,y0,x1,y1,x2,y2,x3,y3 : Double; End;
Var      cpt1,cpt2: BezTyp;
Type     TabTyp = Record x1,y1,x2,y2: Double; End;
         TabArr = Array [0..m] Of TabTyp;
Var      Tab1,Tab2: TabArr;

Procedure    Params;                                                    Forward;
Procedure    ValBez  (cpt: beztyp; t: Double; Var xb,yb: Double);        Forward;
Procedure    DrawBez (m: Integer; cpt: BezTyp; pal,col: Integer);        Forward;
Procedure    MakeBez (k:Integer; Var cpt: BezTyp);                       Forward;
Procedure    Filltab (cpt: BezTyp; Var Tab: TabArr);                     Forward;
Procedure    DrawTab (m: Integer; tab: TabArr; pal,col: Integer);        Forward;
Procedure    Mark    (p,q,pal,sel: Integer);                             Forward;
Procedure    Intsect (m: Integer; tab1,tab2: TabArr; pal,col: Integer);  Forward;
Procedure    MakeCLine(x1,y1,x2,y2,u1,u2,v1,v2:Double; pal,col: Integer);Forward;

Procedure Params;
Begin
  xm:=Round(0.50*gmx);
  ym:=Round(0.50*gmy);
  scale:=290;                { 1.0 —> scale pixels                  }
End;

Procedure MakeCLine(x1,y1,x2,y2,u1,u2,v1,v2: Double; pal,col: Integer);
{    Draw vector from x1,y1 to x2,y2
     clipped for window x=u1 to u2, y=v1 to v2 }
Var  xs,ys,xs1,ys1,xs2,ys2: Double;
     e,z: Integer;
     in1,in2,fnd: Boolean;
Label Ex;
 Procedure ClipSect;
 Var dx,dy,dp,dq,du,dv,A,B,D: Double;
 Begin
  dx:=x2-x1;
  dy:=y2-y1;
  Case e of
  1: Begin du:=u2-u1; dv:=0; dp:=u1-x1; dq:=v1-y1;
     End;
  2: Begin du:=0; dv:=v2-v1; dp:=u2-x1; dq:=v1-y1;
     End;
  3: Begin du:=u1-u2; dv:=0; dp:=u2-x1; dq:=v2-y1;
     End;
  4: Begin du:=0; dv:=v1-v2; dp:=u1-x1; dq:=v2-y1;
     End;
  End; { case }
 D:=dy*du-dx*dv;
 A:=dq*du-dp*dv;
 B:=dx*dq-dy*dp;
 fnd:=False;
 If D>0 Then
  If (0<A) And (A<D) And (0<B) And (B<D) Then fnd:=True;
 If Not fnd And (D<0) Then
  If (0>A) And (A>D) And (0>B) And (B>D) Then fnd:=True;
 If fnd Then
  Begin A:=A/D; xs:=x1+A*dx; ys:=y1+A*dy;
  End;
 End;
Begin
in1:=False; in2:=False;
If (x1>u1) And (x1<u2) And (y1>v1) And (y1<v2) Then in1:=True;
If (x2>u1) And (x2<u2) And (y2>v1) And (y2<v2) Then in2:=True;
```

4.5 Bézier Intersections

```
{ No clipping }
If in1 And in2 Then
Begin
 MakeSline(Round(x1),Round(y1),Round(x2),Round(y2),pal,col);
Goto Ex;
End;
{ No drawing }
If ((x1<u1) And (x2<u1)) Or ((x1>u2) And (x2>u2))
Or ((y1<v1) And (y2<v1)) Or ((y1>v2) And (y2>v2)) Then Goto Ex;
{ Check intersections }
z:=0; e:=1;
Repeat
ClipSect;
If fnd Then
 Begin
 Inc(z);
 If z=1 Then
  Begin xs1:=xs; ys1:=ys;
  End   Else
  Begin xs2:=xs; ys2:=ys;
  End;
 End;
Inc(e);
Until (e>4) Or (z=2);
Case z Of
0:  Goto Ex;
1:  If In1 Then
    MakeSline(Round(x1),Round(y1),Round(xs1),Round(ys1),pal,col) Else
    MakeSline(Round(x2),Round(y2),Round(xs1),Round(ys1),pal,col);
2:  MakeSline(Round(xs1),Round(ys1),Round(xs2),Round(ys2),pal,col);
End; { Case }
Ex:
End;

Procedure ValBez (cpt: BezTyp; t: Double; Var xb,yb: Double);
{    4 control points P0,P1,P2,P3
     Calculate Pb for t=0..1          }
Var ax,bx,cx,ay,by,cy: Double;
Begin
 With cpt Do
 Begin
  cx:=(x1-x0)*3;      bx:=(x2-x1)*3-cx;     ax:= x3-x0-cx-bx;
  cy:=(y1-y0)*3;      by:=(y2-y1)*3-cy;     ay:= y3-y0-cy-by;
  xb:=((ax*t+bx)*t+cx)*t+x0;
  yb:=((ay*t+by)*t+cy)*t+y0;
 End;
End;

Procedure DrawBez(m: Integer; cpt: BezTyp; pal,col: Integer);
{ Stroke path
  m subdivisions per segment         }
Var xb,yb,t,dt  : Double;
    p1,q1,p2,q2 : Double;
    i,j         : Integer;
Begin
dt:=1/m;
With cpt Do
 Begin
 p1:=xm+scale*x0;
 q1:=ym-scale*y0;
 End;
t:=0;
For j:=0 to m-1 Do
 Begin
  t:=t+dt;
  ValBez(cpt,t,xb,yb);
  p2:=xm+scale*xb;
  q2:=ym-scale*yb;
  MakeCline(p1,q1,p2,q2,0,gmx,0,gmy,pal,col);
  p1:=p2;
  q1:=q2;
 End;
End;
```

4.6 Bézier Intersections

```
Procedure Filltab (cpt: BezTyp; Var Tab: TabArr);
Var i                                 : Integer;
    t,dt,sw,xb,yb,xb1,yb1,xb2,yb2 : Double;
Begin
dt:=1/m; t:=0;
ValBez(cpt,t,xb1,yb1);
For i:=0 to m-1 Do
Begin
 t:=t+dt;
 ValBez(cpt,t,xb2,yb2); xb:=xb2; yb:=yb2;
 If xb2<xb1 Then
  Begin sw:=xb1; xb1:=xb2; xb2:=sw;
        sw:=yb1; yb1:=yb2; yb2:=sw;
  End;
 With Tab[i] Do
  Begin x1:=xb1; y1:=yb1; x2:=xb2; y2:=yb2;
  End;
 xb1:=xb; yb1:=yb;
End;
End;

Procedure Intsect (m: Integer; tab1,tab2: TabArr; pal,col: Integer);
Var i,j,p,q                       : Integer;
    p1x,p1y,g1x,g1y,g2x,g2y   : Double;
    D,A,B,px,py,min,max       : Double;
    fnd                       : Boolean;
Label Ex;
Begin
For i:=0 to m-1 Do
 Begin
  With tab1[i] Do
  Begin
   min:=x1; max:=x2;
   p1x:=x1;
   p1y:=y1;
   g1x:=x2-x1;
   g1y:=y2-y1;
  End;
  For j:=0 to m-1 Do
  Begin
   With tab2[j] Do
   Begin
    If (max<x1) Or (min>x2) Then Goto Ex;
    px :=x1-p1x;
    py :=y1-p1y;
    g2x:=x2-x1;
    g2y:=y2-y1;
    D:=g2x*g1y-g1x*g2y;
    A:=g2x*py -g2y*px;
    B:=g1x*py -g1y*px;
    fnd:=False;
   If D>0 Then
    If (0<A) And (A<D) And (0<B) And (B<D) Then fnd:=True;
   If Not fnd And (D<0) Then
    If (0>A) And (A>D) And (0>B) And (B>D) Then fnd:=True;
   If fnd Then
    Begin
     B:=B/D;
     px:=x1+B*g2x;
     py:=y1+B*g2y;
     p :=xm+Round(px*scale);
     q :=ym-Round(py*scale);
     Mark(p,q,pal,col);
    End;
   End; { with tab2[j] }
  Ex:
  End;
 End;
End;
```

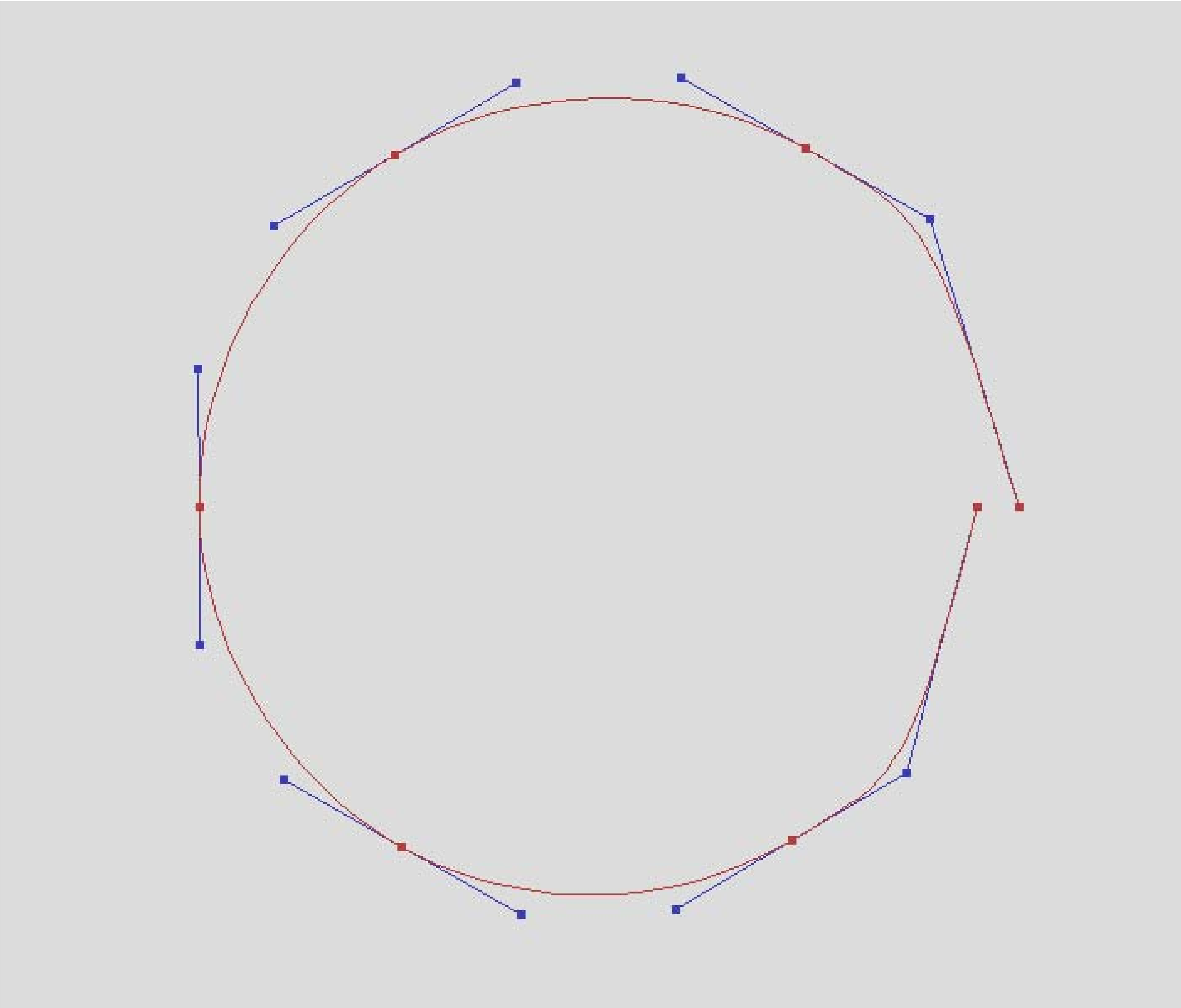
4.7 Bézier Intersections

```
Procedure DrawTab(m: Integer; tab: TabArr; pal,col: Integer);
Var i,p1,q1,p2,q2: Integer;
Begin
For i:=0 to m-1 Do
 With tab[i] Do
 Begin
 p1:=xm+Round(x1*scale);
 q1:=ym-Round(y1*scale);
 p2:=xm+Round(x2*scale);
 q2:=ym-Round(y2*scale);
 MakeSline(p1,q1,p2,q2,pal,col);
 End;
End;

Procedure Mark(p,q,pal,sel: Integer);
Var i,j: Integer;
Begin
For i:=-2 to 2 Do
For j:=-2 to 2 Do SpcSixel(p+i,q+j,pal,sel);
End;

Procedure MakeBez(k:Integer; Var cpt: BezTyp);
Const    C1: Double=1E12;
         C2: Double=1E0;
{ Accurate for   C1=1         C2=1..1E12
                 C1=1..1E12   C2=1
                 C1=1E12      C2=1E12
  Safe for       C1=1..1E300  C2=1..1E300 }
Begin
Case k Of
1: With cpt Do
   Begin
   x0:=1; y0:=0; x1:=0.50; y1:=4*C1; x2:=-0.5; y2:=-4*C1; x3:=-1; y3:=0;
   End;
2: With cpt Do
   Begin
   x0:=0; y0:=1; x1:=4*C2; y1:=0.5; x2:=-4*C2; y2:=-0.50; x3:=0;  y3:=-1;
   End;
End;
End;

BEGIN
VesaMode:=Vmode42;
VesaCode:=$0115;
VesaStart(VesaMode);
MemGStart;
Params;
ColToScr(181,220);
MakeBez (1,cpt1);
MakeBez (2,cpt2);
DrawBez (m,cpt1, 0,120);
DrawBez (m,cpt2,60,120);
time1:=time;
For k:=1 to 100 Do
Begin
FillTab (cpt1,tab1);
FillTab (cpt2,tab2);
Intsect (m,tab1,tab2,120,120);
End;
time2:=time;
WrNumwin(1,grel,1,'dt',(time2-time1)/100);
{ m=100:  2.2 ms for all intersections once }
SaveImag('I:\Bezier\Drawf611.BMP');
Stop;
MemGEnde;
VesaEnde;
END.
```

5.1 Bézier Fast Drawing

According to *Foley, van Dam et.al., Computer Graphics*[3] the Bézier curves can be drawn by straight line segments using the so-called forward differences for a fast calculation without multiplications. The example shows the brute force approach by Horner´s rule (incremented parameter t) and the improved method. Both methods deliver the same pixels.
Furtheron, the construction of initial control points (chapter 3) is demonstrated.

Best view zoom 100%



5.2 Bézier Fast Drawing / Code

```
Program ZBezier3;
{ Project    Draw Bezier by Forward Differences, Function x(s), y(s) for s=0..1
  Author     Gernot Hoffmann
  Date       December 23, 2002 }

Uses     Crt,Dos,
         Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
         Zefir36,Zefir37,Zefir38,Zefir39,Zefir40;
Var      xm,ym,scale : Integer;
         xa,ya,xe,ye : Integer;
         pal,col,sel : Integer;
Const    n=  6;  { 0..n fixpoints }
         m= 20;  { subdivisions in one line segment }
Type     BezTyp = Record x0,y0,x1,y1,x2,y2,x3,y3 : Double; End;
         BezArr = Array [0..n] of Beztyp;
Var      cpt: BezArr;

Procedure    Params;                                                   Forward;
Procedure    FuncVal (sel: Integer; s: Double; Var xf,yf: Double);    Forward;
Procedure    ContBez (sel,n: Integer; Var cpt: bezarr);                Forward;
Procedure    ValBez  (cpi: BezTyp; t: Double; Var xb,yb: Double);     Forward;
Procedure    DrawBez1(n,m: Integer; cpt: BezArr; pal,col: Integer);   Forward;
Procedure    DrawBez2(n,m: Integer; cpt: BezArr; pal,col: Integer);   Forward;
Procedure    Mark(p,q,pal,sel: Integer);                               Forward;

Procedure FuncVal(sel: Integer; s: Double; Var xf,yf: Double);
Const    pi2:Double=2*pi;
Begin
Case sel Of
0: Begin
    xf:=s;
    yf:=s;
   End;
1: Begin { Circle }
    xf:=coc(pi2*s);
    yf:=sic(pi2*s)
   End;
2: Begin { Spiral }
    xf:=(1-0.1*s)*coc(pi2*s);
    yf:=(1-0.1*s)*sic(pi2*s)
   End;
 End;
End;

Procedure Params;
Begin
  xm:=xpx div 2;
  ym:=ypx div 2;
  scale:=250;                { 1.0 -> scale pixels                  }
End;

Procedure Mark(p,q,pal,sel: Integer);
Var i,j: Integer;
Begin
For i:=-2 to 2 Do
For j:=-2 to 2 Do SpcSixel(p+i,q+j,pal,sel);
End;

Procedure ValBez (cpi: BezTyp; t: Double; Var xb,yb: Double);
{    4 control points p0,p1,p2,p3
     Calculate pb for t=0..1           }
Var ax,bx,cx,ay,by,cy: Double;
Begin
 With cpi Do
 Begin
  cx:=(x1-x0)*3;
  bx:=(x2-x1)*3-cx;
  ax:= x3-x0-cx-bx;
  cy:=(y1-y0)*3;
  by:=(y2-y1)*3-cy;
  ay:= y3-y0-cy-by;
  xb:=((ax*t+bx)*t+cx)*t+x0;
  yb:=((ay*t+by)*t+cy)*t+y0;
 End;
End;
```

5.3 Bézier Fast Drawing / Code

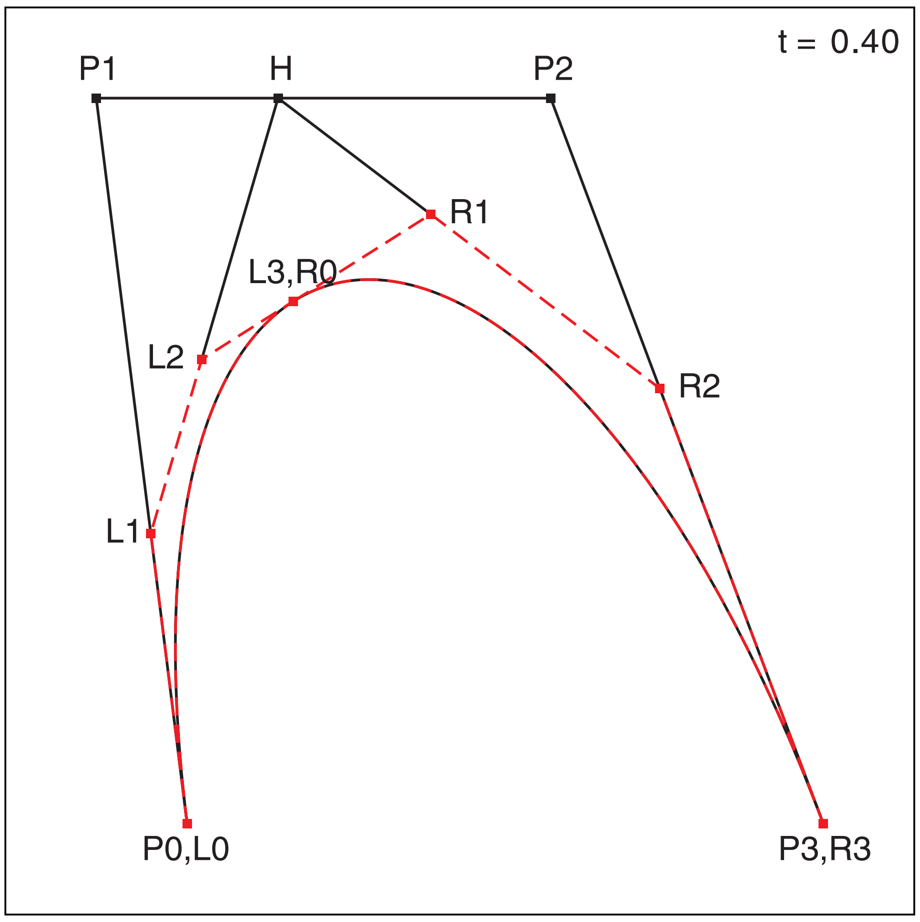
```
Procedure DrawBez1 (n,m: Integer; cpt: BezArr; pal,col: Integer);
{    Stroke path by brute force Horner in ValBez
     n subdivisions for control points
     m subdivisions per segment         }
Var  xb,yb,t,dt                  : Double;
     p0,q0,p1,q1,p2,q2,p3,q3     : Integer;
     i,j                         : Integer;
Begin
dt:=1/m;
For i:=0 to n-1 Do
 Begin
  With cpt[i] Do
  Begin
  p0:=xm+Round(scale*x0);
  q0:=ym-Round(scale*y0);
  p1:=xm+Round(scale*x1);
  q1:=ym-Round(scale*y1);
  p2:=xm+Round(scale*x2);
  q2:=ym-Round(scale*y2);
  p3:=xm+Round(scale*x3);
  q3:=ym-Round(scale*y3);
  Mark(p0,q0,0,120);
  MakeSline(p0,q0,p1,q1,120,120);
  Mark(p1,q1,120,120);
  Mark(p3,q3,0,120);
  MakeSline(p3,q3,p2,q2,120,120);
  Mark(p2,q2,120,120);
  End;
  t:=0;
  For j:=0 to m-1 Do
   Begin
   t:=t+dt;
   ValBez(cpt[i],t,xb,yb);
   p3:=xm+Round(scale*xb);
   q3:=ym-Round(scale*yb);
   MakeSline(p0,q0,p3,q3,pal,col);
   p0:=p3;
   q0:=q3;
   End;
 End;
End;

Procedure DrawBez2(n,m: Integer; cpt: BezArr; pal,col: Integer);
{    Stroke path by Forward Differences }
Var  ax,bx,cx,ay,by,cy           : Double;
     dt1,dt2,dt3,x,y             : Double;
     dx1,dx2,dx3,dy1,dy2,dy3     : Double;
     p0,q0,p1,q1,i,j             : Integer;
Begin
dt1:=1/m;
dt2:=Sqr(dt1);
dt3:=dt1*dt2;
For i:=0 to n-1 Do
 Begin
  With cpt[i] Do
  Begin
  cx :=(x1-x0)*3;              bx :=(x2-x1)*3-cx;        ax := x3-x0-cx-bx;
  cy :=(y1-y0)*3;              by :=(y2-y1)*3-cy;        ay := y3-y0-cy-by;
  dx1:=dt3*ax+dt2*bx+dt1*cx;  dx2:=6*dt3*ax+2*dt2*bx;   dx3:=6*dt3*ax;
  dy1:=dt3*ay+dt2*by+dt1*cy;  dy2:=6*dt3*ay+2*dt2*by;   dy3:=6*dt3*ay;
  x:=x0; y:=y0;
  p0:=xm+Round(scale*x); q0:=ym-Round(scale*y);
  End;
  For j:=0 to m-1 Do
   Begin
   x:=x+dx1; dx1:=dx1+dx2; dx2:=dx2+dx3;
   y:=y+dy1; dy1:=dy1+dy2; dy2:=dy2+dy3;
   p1:=xm+Round(scale*x);
   q1:=ym-Round(scale*y);
   MakeSline(p0,q0,p1,q1,pal,col);
   p0:=p1; q0:=q1;
   End;
 End;
End;
```

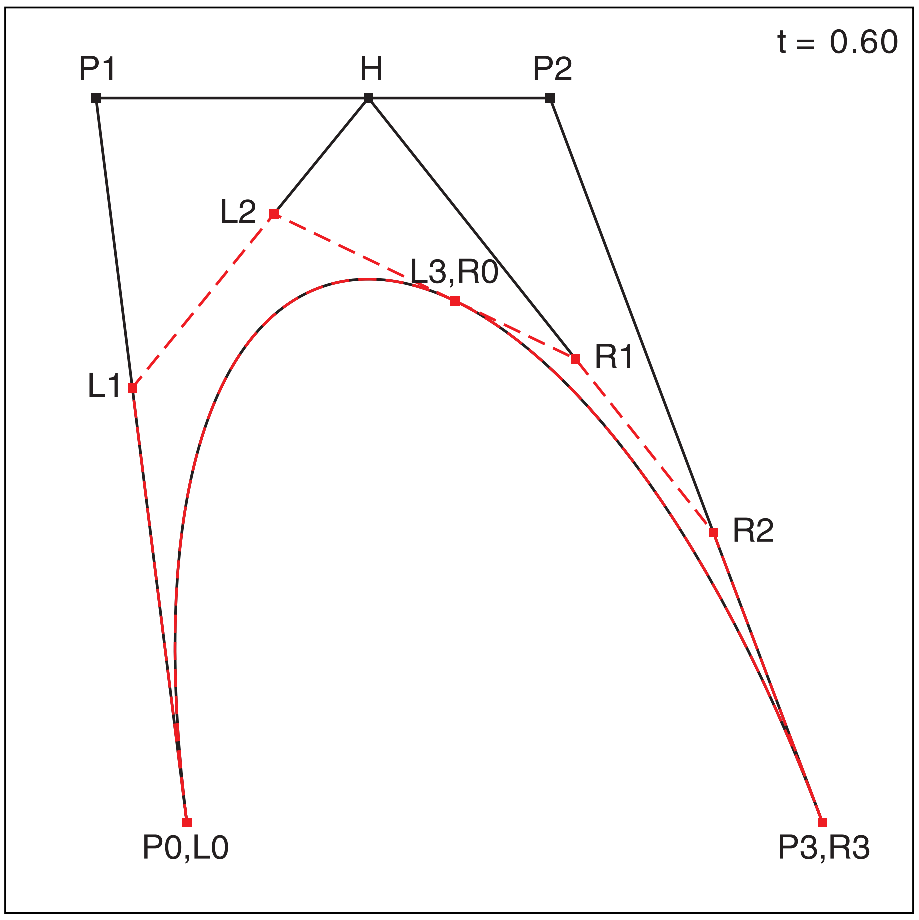
5.4 Bézier Fast Drawing / Code

```
Procedure ContBez(sel,n: Integer; Var cpt: BezArr);
{    Function in parametric representation xf(s),yf(s) for s=0..1
     Calculate fixpoints and approximations for other control points
     sel       Function selector
     n         Subdivisions for i=0..n fixpoints
     cpt       Array of Beztyp     }
Var i,k               : Integer;
     dx,dy,s,ds,xf,yf : Double;
Begin
ds:=1/n;
s:=0;
With cpt[0] Do FuncVal(sel,s,x0,y0);
For i:=1 to n Do
 Begin
  s:=s+ds;
  FuncVal(sel,s,xf,yf);
  With cpt[i] Do
   Begin x0:=xf; y0:=yf;
   End;
  With cpt[i-1] Do
   Begin x3:=xf; y3:=yf;
   End;
 End;
For i:=1 to n-1 Do
 Begin
  dx:=0.2*(cpt[i+1].x0-cpt[i-1].x0);
  dy:=0.2*(cpt[i+1].y0-cpt[i-1].y0);
  With cpt[i] Do
  Begin
   cpt[i  ].x1:=x0+dx;
   cpt[i  ].y1:=y0+dy;
   cpt[i-1].x2:=x0-dx;
   cpt[i-1].y2:=y0-dy;
  End;
 End;
With cpt[0] Do
 Begin x1:=x2; y1:=y2;
 End;
With cpt[n-1] Do
 Begin x2:=x1; y2:=y1;
 End;
End;

BEGIN
VesaMode:=Vmode42;
VesaCode:=$0115;
VesaStart(VesaMode);
MemGStart;
Params;
ColToScr(181,220);
sel:=2;
ContBez (sel,n,cpt);
DrawBez2(n,m,cpt,60,120);
DrawBez1(n,m,cpt, 0,120);
SaveImag('H:\DrawF\DrawF620.BMP');
Stop;
MemGEnde;
VesaEnde;
END.
```

6.1 De Casteljau Subdivision / Concept

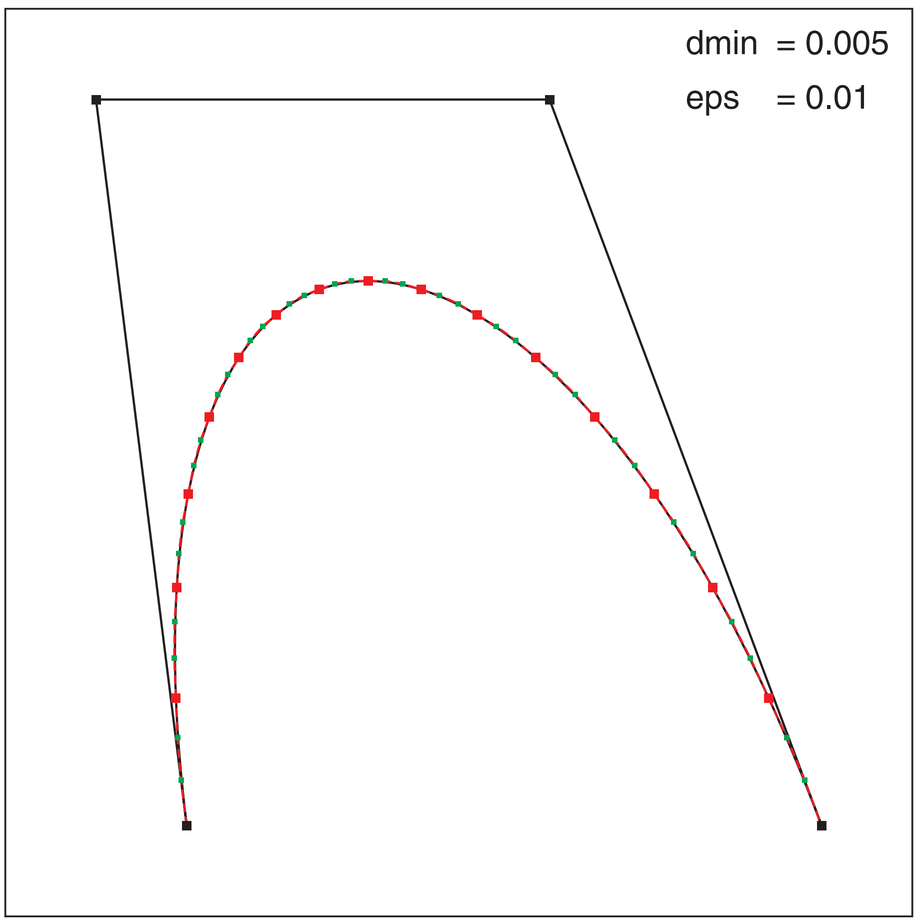
This chapter follows again *Foley, van Dam et.al., Computer Graphics* [3], with some improvements and further explanations. Any non-singular Bézier curve can be split at a parameter (t) into two curves. The examples show the cases for t=0.4 and t=0.6. Splitting is e.g. useful for adding more detail to the curves.

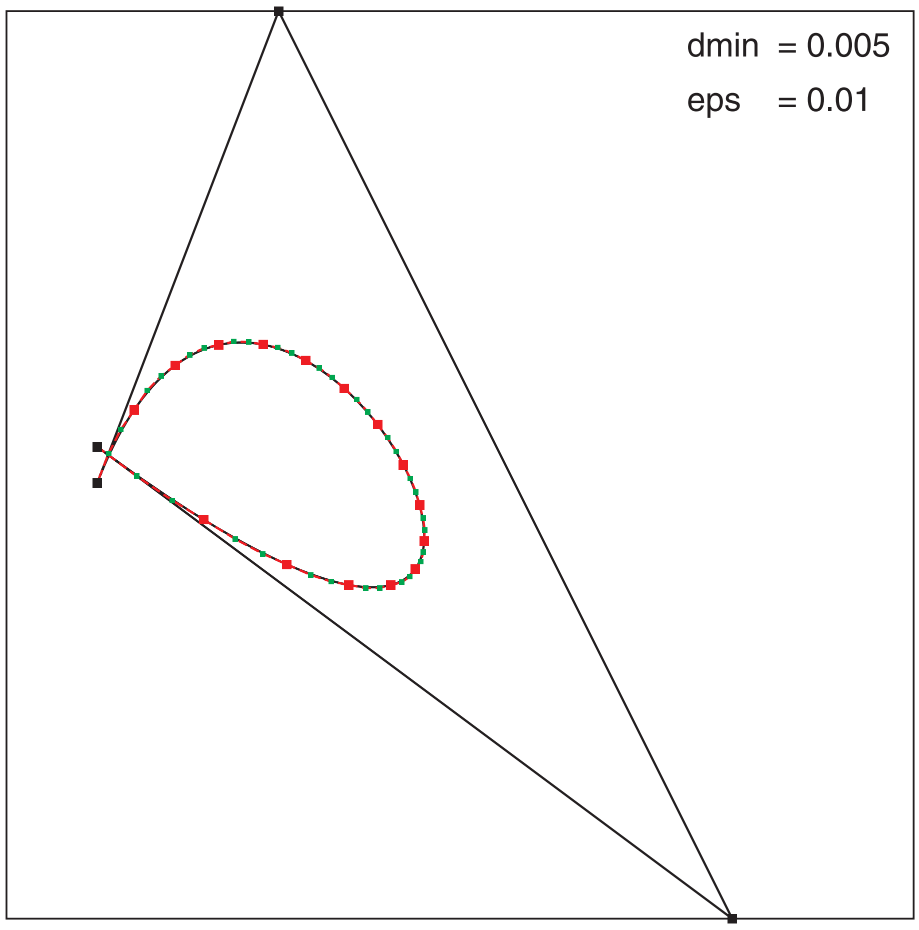




The lines P_0P_1, P_1P_2, P_2P_3 are divided at $P=P_n+t(P_{n+1}-P_n)$. This delivers L_1,H and R_2. A further similar subdivision creates the complete set of four control points for the left curve and for the right curve. For t=0.5 the calculation is simplified by $P=(P_n+P_{n+1})/2$. H is an auxiliary point which does not belong to the set of control points.

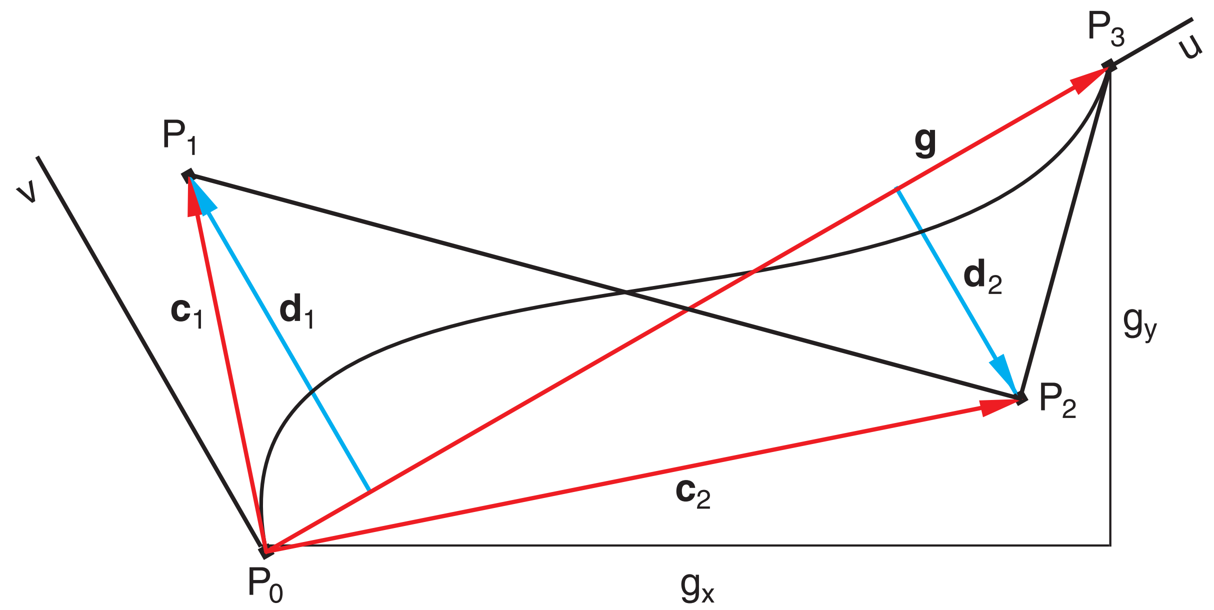
The two curves can be split again. If this process is continued, then the polyline through the control points is an approximation for the Bézier curve itself. Doing this economically delivers a second method for fast Bézier drawing, in addition to the method by forward differences, as explained in the previous chapter. Middle control points are shown green.





6.2 De Casteljau Subdivision / Flatness

The recursion has to be stopped if the approximation by polylines is suffciently accurate. The flatness can be tested by the distances of the control points P_1 and P_2 from the line P_0P_3, represented by vectors $\mathbf{d}_1$ and $\mathbf{d}_2$. The longer one has to be shorter than some flatness value d_{min}.



The two distance vectors can be calculated directly:

$$\mathbf{g} = \mathbf{P}_3 - \mathbf{P}_0$$

$$\mathbf{c}_i = \mathbf{P}_i - \mathbf{P}_0 \quad \text{for } i = 1,2$$

$$\mathbf{d}_i = \mathbf{c}_i - \frac{\mathbf{g}^T\mathbf{c}_i}{\mathbf{g}^T\mathbf{g}}\,\mathbf{g}$$

This algorithm is slow and suffers from the possibility of division by zero or near to zero. The Bézier curve can be interpreted in rotated coordinates u,v.
The v-coordinates of $\mathbf{d}_1$ and $\mathbf{d}_2$ are the lengths of the vectors.

$$\mathbf{g} = \mathbf{P}_3 - \mathbf{P}_0$$

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\sin(\alpha) = g_y / g$$

$$\cos(\alpha) = g_x / g$$

$$\mathbf{c}_i = \mathbf{P}_i - \mathbf{P}_0 \quad \text{for } i = 1,2$$

$$v_i = -c_{ix}\sin(\alpha) + c_{iy}\cos(\alpha)$$

Without denominator:

$$V_i = -c_{ix}\,g_y + c_{iy}\,g_x$$

$$V = \max(|V_1|,|V_2|)$$

If $V < d_{min}\,g$ Then Flat=true Else Flat=false

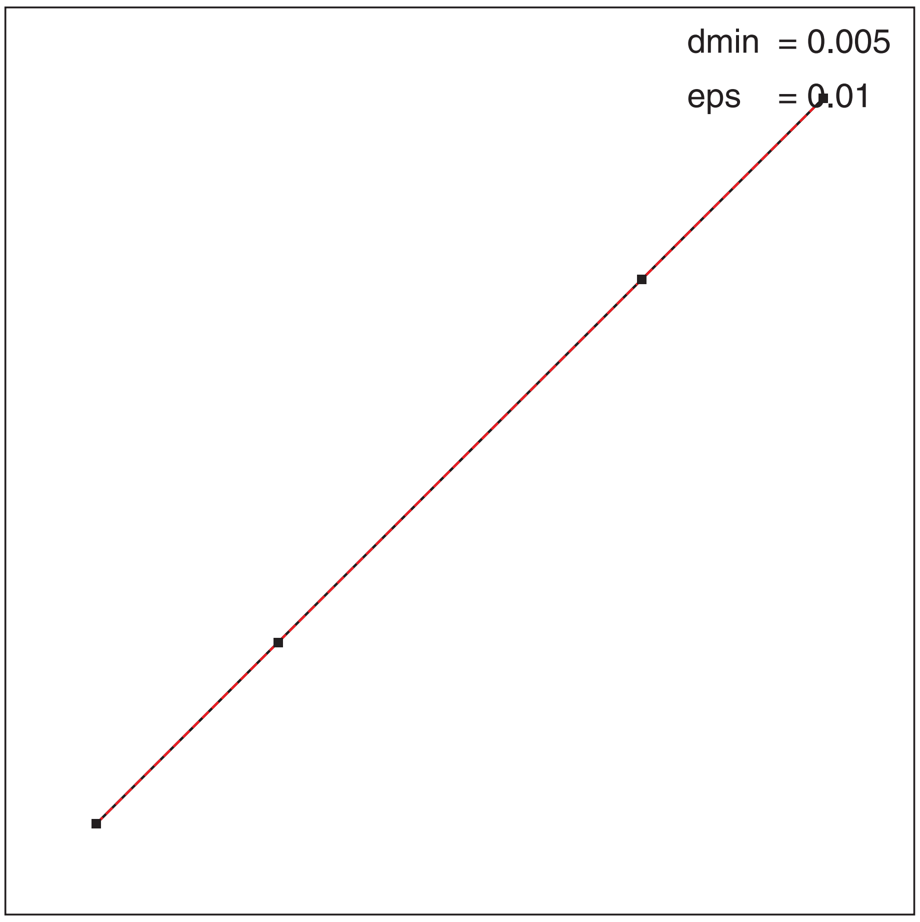
Improved by additional stop condition:

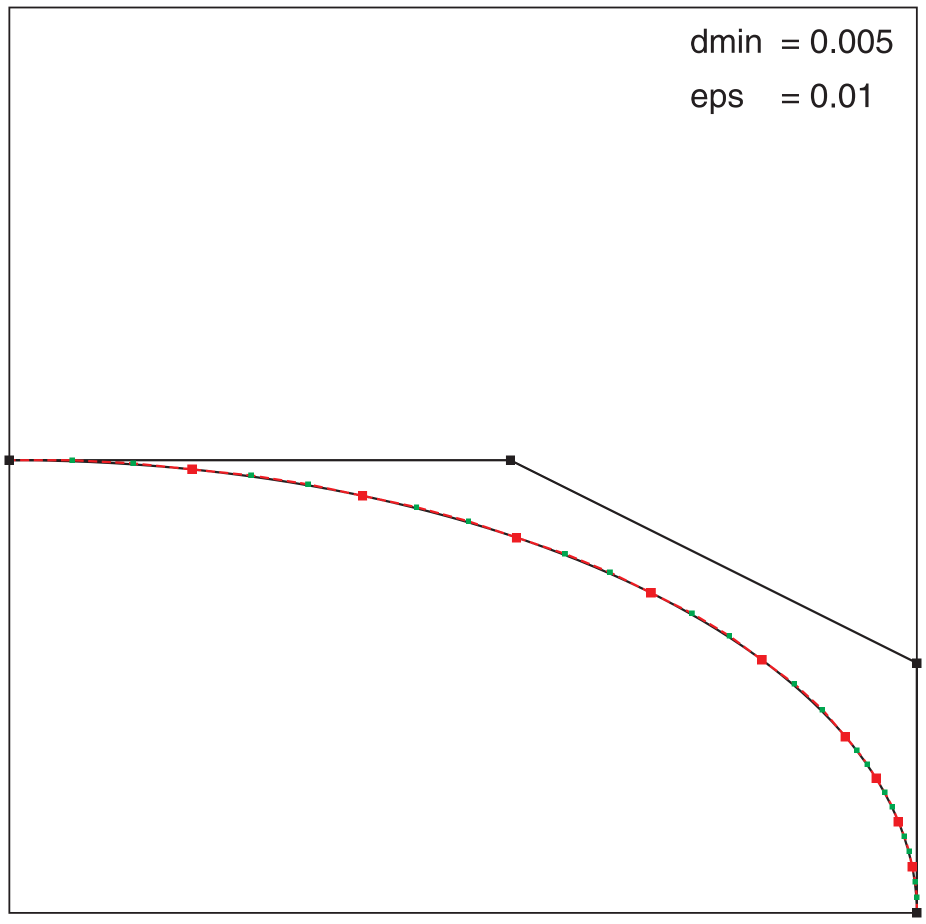
If $V < d_{min}\,(g+\varepsilon)$ Then Flat=true Else Flat=false

Division by g=0 could be avoided. Additionally, a small parameter ε was added, which plays the role of a minimal line segment length. The minimal deviation d_{min} should be replaced by $d_{min}=d_{min}/b$, where (b) is the edge lenght of an estimated bounding square for the actual class of Bézier curves, e.g. for glyphs. Singular Béziers may have g=0 and long tangents or g>0 but zero tangent lengths. These curves cannot be drawn at all.

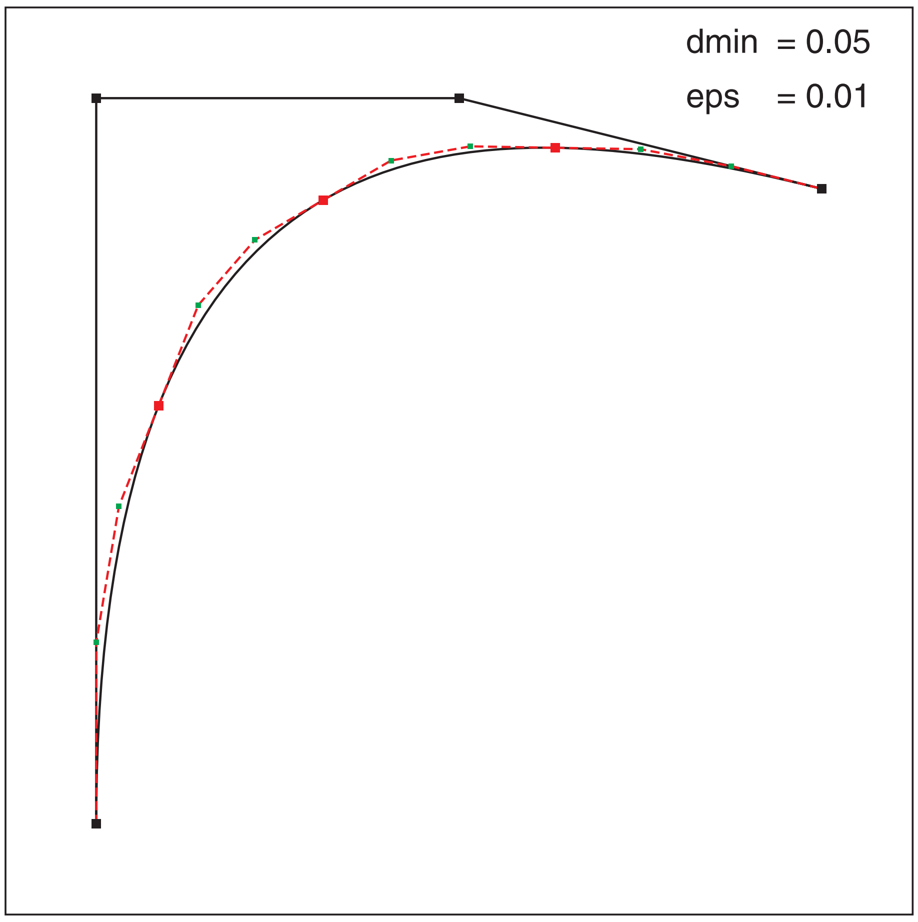
6.3 De Casteljau Subdivision / More Examples

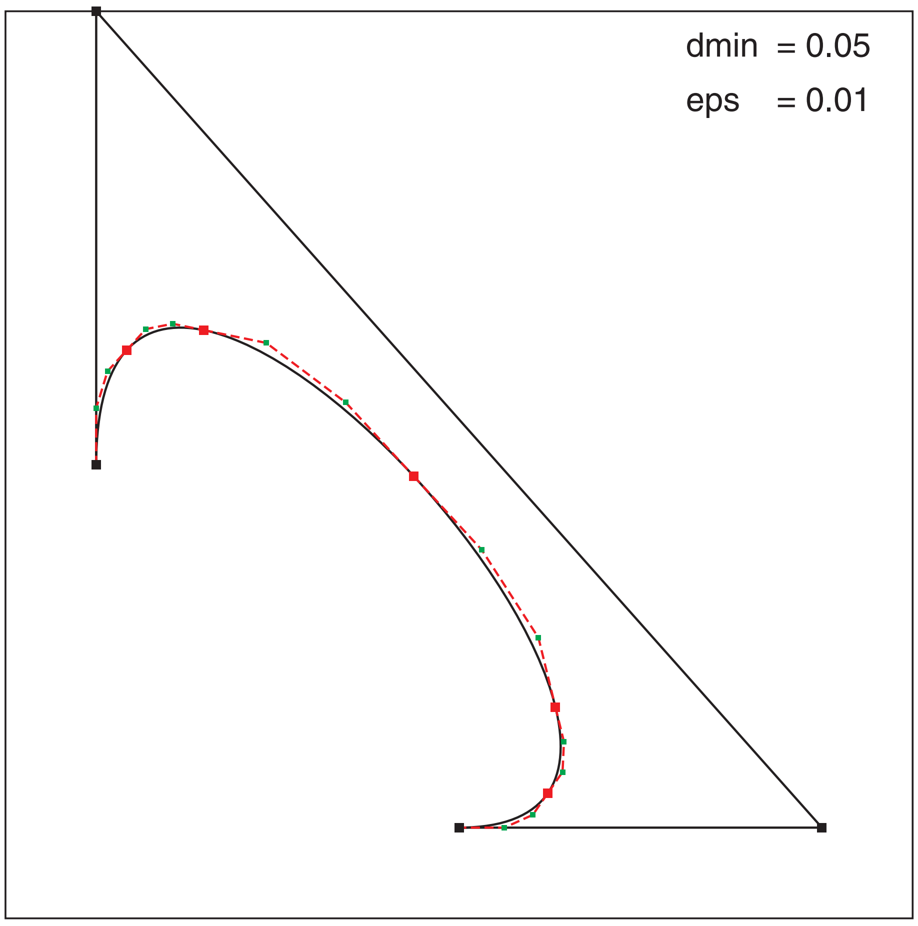
The graphic top left shows a straight line. The algorithm does not apply any subdivision.
The ellipse top right looks as expected.
The red squares are end points, the green squares are middle control points.





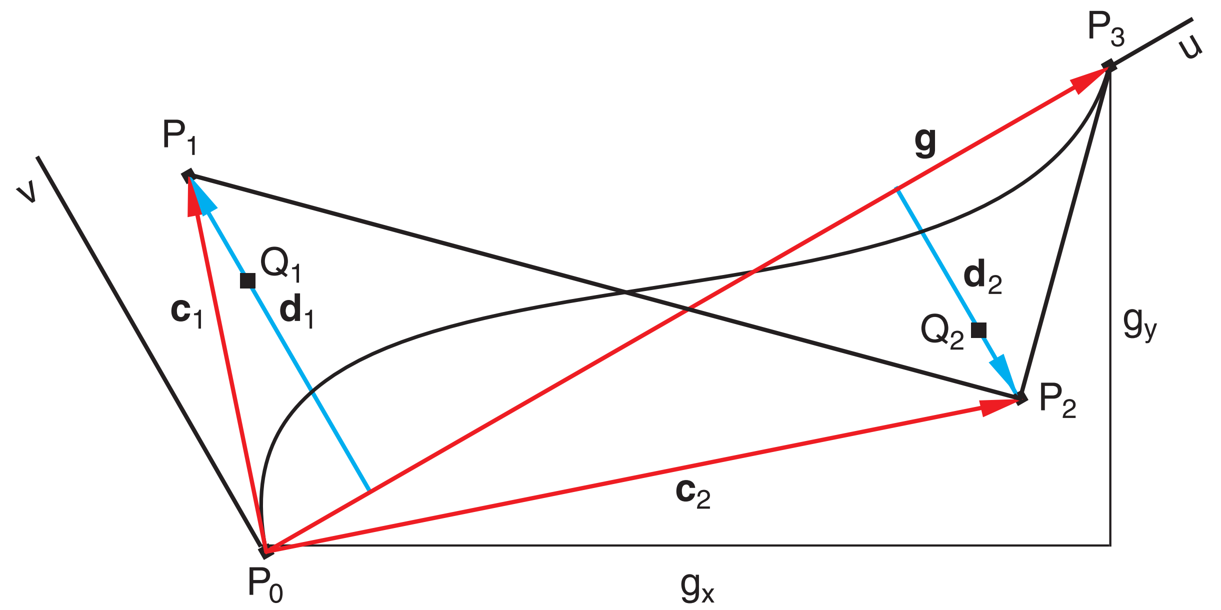
These graphics show coarse approximations by larger d_{min}.





6.4 De Casteljau Subdivision / Better Drawing

The two examples on the previous page, bottom, are showing clearly, that the polyline line through the control points $P_0P_1P_2P_3$ is by no means a good approximation.
Therefore the polyline is replaced by $P_0Q_1Q_2P_3$, where Q_i is shifted from P_i along $\mathbf{d}_i$ by an arbitrary factor K. K=0.5 delivers already reasonable results, but K=0.3 is better.



This is the modified algorithm:

$$\mathbf{g} = \mathbf{P}_3 - \mathbf{P}_0$$
$$g_s = g_x^2 + g_y^2$$
$$g = \sqrt{g_s}$$
$$\sin(\alpha) = g_y / g$$
$$\cos(\alpha) = g_x / g$$
$$\mathbf{c}_i = \mathbf{P}_i - \mathbf{P}_0 \quad \text{for } i = 1,2$$
$$v_i = -c_{ix}\sin(\alpha) + c_{iy}\cos(\alpha)$$

Without denominator:

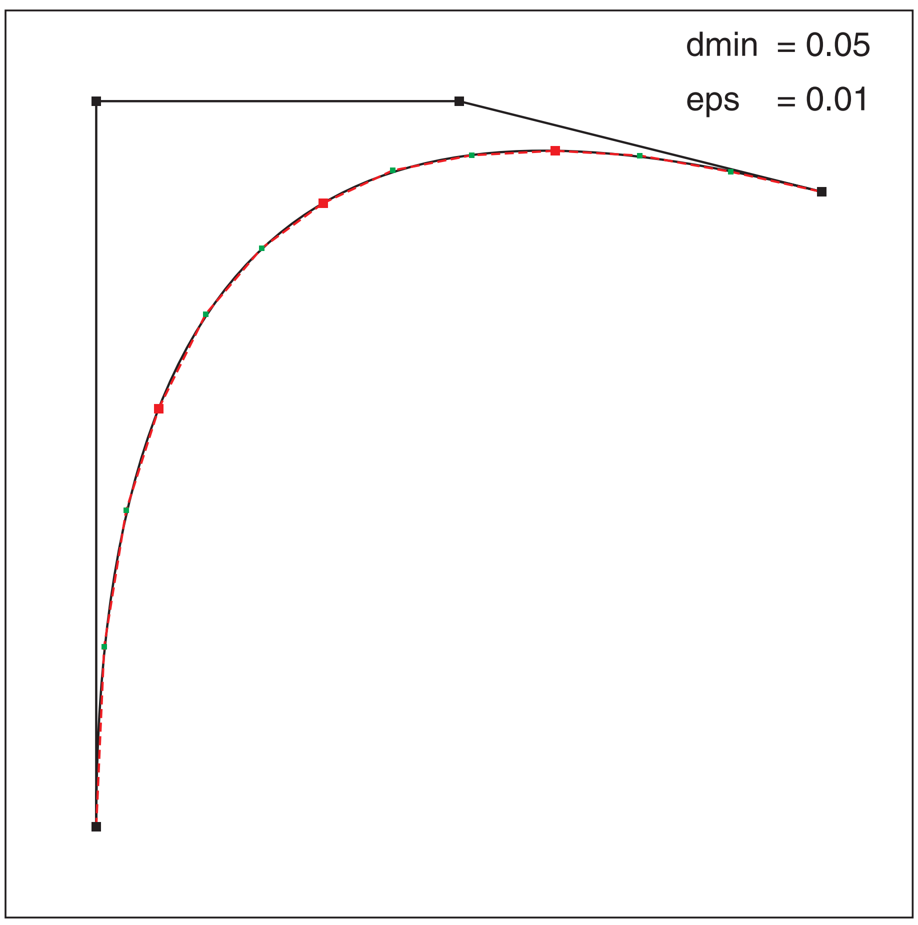
$$V_i = -c_{ix}\,g_y + c_{iy}\,g_x$$
$$V = \max(|V_1|,|V_2|)$$

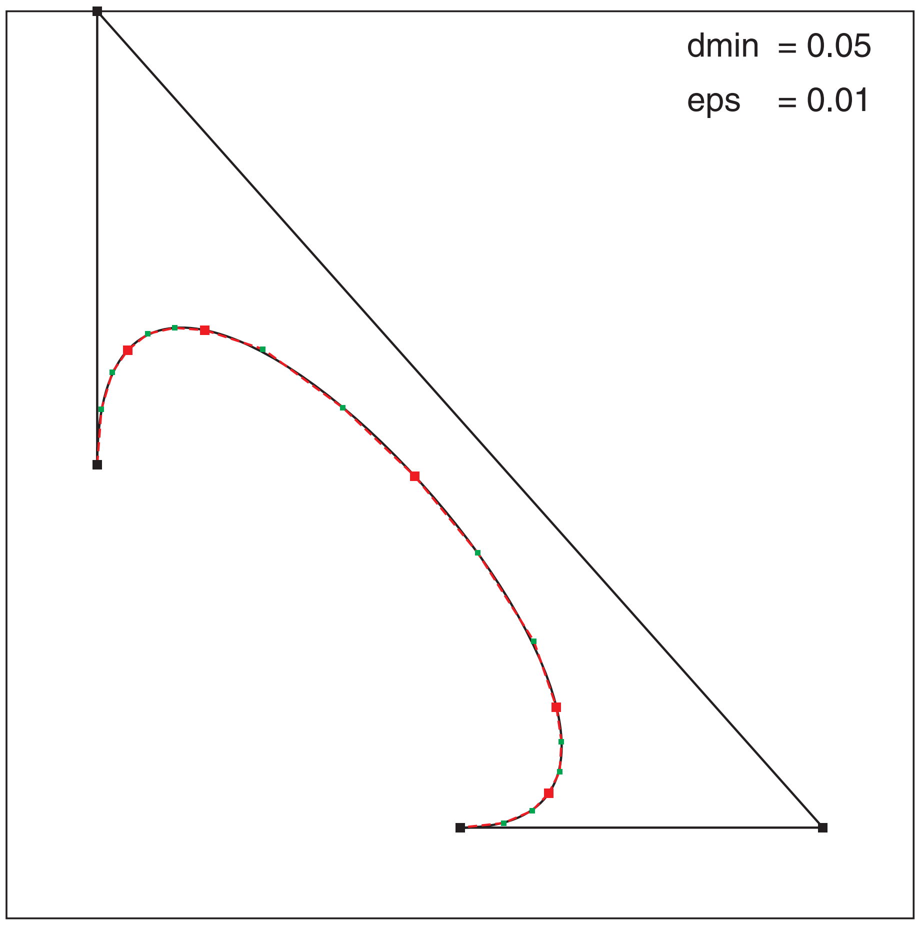
If $V < d_{min}\,g$ Then Flat=true Else Flat=false

Improved by additional stop condition:

If $V < d_{min}(g+\varepsilon)$ Then Flat=true Else Flat=false

$$K = 0.3$$
$$Q_{ix} = P_{ix} + K v_i \sin(\alpha) = P_{ix} + K V_i\, g_y / g_s$$
$$Q_{iy} = P_{iy} - K v_i \cos(\alpha) = P_{iy} - K V_i\, g_x / g_s$$





6.5 De Casteljau Subdivision / PostScript Code

```
%!PS-Adobe-3.0   EPSF-3.0
%%BoundingBox:   0 0 510 340
%%Creator:       Gernot Hoffmann
%%Title:         BezCastel13-Rec
%%CreationDate:  October 05/2006

% Disable setpagedevice
 /setpagedevice {pop} bind def

% Bezier de Casteljau Subdivision Recursion
% e is not used in this program

/Typ 3 def

Typ 1 eq {
/p0x 0.1 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.5 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.8 def
/e +0.03 def               } if
Typ 2 eq { % like sine full wave
/p0x 0.1 def /p0y  0.5 def
/p1x 0.1 def /p1y  1.0 def
/p2x 0.9 def /p2y  0.0 def
/p3x 0.9 def /p3y  0.5 def
/e +0.03 def               } if
Typ 3 eq { % like distorted sine halfwave
/p0x 0.2 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.6 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.1 def
/e +0.03 def               } if
Typ 4 eq {
/p0x 0.1 def /p0y  0.5 def
/p1x 0.1 def /p1y  1.0 def
/p2x 0.9 def /p2y  0.1 def
/p3x 0.5 def /p3y  0.1 def
/e +0.03 def               } if
Typ 5 eq { % loop
/p0x 0.1 def /p0y  0.52 def
/p1x 0.8 def /p1y  0.0 def
/p2x 0.3 def /p2y  1.0 def
/p3x 0.1 def /p3y  0.48 def
/e +0.03 def               } if
Typ 6 eq { % strictly linear
/p0x 0.1 def /p0y  0.1 def
/p1x 0.3 def /p1y  0.3 def
/p2x 0.7 def /p2y  0.7 def
/p3x 0.9 def /p3y  0.9 def
/e +0.03 def               } if
Typ 7 eq { % nearly linear
/p0x 0.1 def /p0y  0.1 def
/p1x 0.3 def /p1y  0.3 +1e-4 add def
/p2x 0.7 def /p2y  0.7 -1e-4 add def
/p3x 0.9 def /p3y  0.9 def
/e +0.03 def               } if
Typ 8 eq { % circle
/p0x 1.0 def   /p0y  0.0   def
/p1x 1.0 def   /p1y  0.552 def
/p2x 0.552 def /p2y  1.0 def
/p3x 0.0 def   /p3y  1.0   def
/e +0.03 def               } if
Typ 9 eq { % ellipse
/p0x 1.0 def   /p0y  0.0   0.5 mul def
/p1x 1.0 def   /p1y  0.552 0.5 mul def
/p2x 0.552 def /p2y  1.0   0.5 mul def
/p3x 0.0 def   /p3y  1.0   0.5 mul def
/e +0.03 def               } if
Typ 10 eq {
/p0x 0.1 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.5 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.8 def
/e +0.055 def              } if
```

6.6 De Casteljau Subdivision / PostScript Code

```
/mm {2.834646 mul} def

/sx  100 mm def  % Length 0..sx
/sy  100 mm def
/bx  510 def % Bounding box
/by  340 def

/x0  10 mm def   % Offset
/y0  10 mm def

/Bbox
{ 0.4 mm setlinewidth
  0 0 0 1 setcmykcolor
  0 0 moveto bx 0 rlineto 0 by rlineto bx neg 0 rlineto closepath stroke
} def

/Vbox
{ 0.2 mm sx div setlinewidth
  0 0 0 1 setcmykcolor
  0 0 moveto 1 0 rlineto 0 1 rlineto -1 0 rlineto closepath stroke
} def

/Dot1R
{/yd0 exch def   /xd0 exch def
 /d1 0.01 def    /d2 d1 0.5 mul def
  0 1 1 0 setcmykcolor
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto closepath fill
 } def

 /Dot2G
{/yd0 exch def   /xd0 exch def
 /d1 0.006 def   /d2 d1 0.5 mul def
  1 0 1 0 setcmykcolor
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto closepath fill
 } def

 /Dot1K
{/yd0 exch def   /xd0 exch def
 /d1 0.01 def    /d2 d1 0.5 mul def
  0 0 0 1 setcmykcolor
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto closepath fill
 } def

/PushP
{p0x p0y p1x p1y p2x p2y p3x p3y
} def

/PushL
{l0x l0y l1x l1y l2x l2y l3x l3y
} def

/PushR
{r0x r0y r1x r1y r2x r2y r3x r3y
} def

/PopP
{/p3y exch def /p3x exch def  /p2y exch def /p2x exch def
 /p1y exch def /p1x exch def  /p0y exch def /p0x exch def
} def

/PopL
{/l3y exch def /l3x exch def  /l2y exch def /l2x exch def
 /l1y exch def /l1x exch def  /l0y exch def /l0x exch def
} def

/PopR
{/r3y exch def /r3x exch def  /r2y exch def /r2x exch def
 /r1y exch def /r1x exch def  /r0y exch def /r0x exch def
} def
```

6.7 De Casteljau Subdivision / PostScript Code

```
/BezP
{ p0x p0y moveto p1x p1y p2x p2y p3x p3y curveto
  stroke
} def

/PolP
{ p0x p0y moveto p1x p1y lineto p2x p2y lineto p3x p3y lineto stroke
  p0x p0y Dot1R p1x p1y Dot2G p2x p2y Dot2G p3x p3y Dot1R
} def

/Subdiv
{/hhx p1x p2x p1x sub ts mul add def
 /l0x p0x                        def
 /l1x p0x p1x p0x sub ts mul add def
 /l2x l1x hhx l1x sub ts mul add def
 /r3x p3x                        def
 /r2x p2x p3x p2x sub ts mul add def
 /r1x hhx r2x hhx sub ts mul add def
 /l3x l2x r1x l2x sub ts mul add def
 /r0x l3x                        def
 /hhy p1y p2y p1y sub ts mul add def
 /l0y p0y                        def
 /l1y p0y p1y p0y sub ts mul add def
 /l2y l1y hhy l1y sub ts mul add def
 /r3y p3y                        def
 /r2y p2y p3y p2y sub ts mul add def
 /r1y hhy r2y hhy sub ts mul add def
 /l3y l2y r1y l2y sub ts mul add def
 /r0y l3y                        def
} def

/Dist % max distance of P1 or P2 from P0P3
{% gx = p3x - p0x
 % gy = p3y - p0y
 % g  = Sqrt(gx*gx+gy*gy)
 % c1x= p1x - p0x
 % c1y= p1y - p0y
 % c2x= p2x - p0x
 % c2y= p2y - p0y
 % sin(a) = gy/g
 % cos(a) = gx/g
 % v1 = -c1x*sin(a)+c1y*cos(a)
 % V1 = -c1x*gy + c1y*gx
 % V2 = -c2x*gy + c2y*gx
 % V = Max(abs(V1),abs(V2))
 % If V<dmin*(g+eps) Then flat
 /gx  p3x p0x sub def
 /gy  p3y p0y sub def
 /c1x p1x p0x sub def
 /c1y p1y p0y sub def
 /c2x p2x p0x sub def
 /c2y p2y p0y sub def
 /V1  c1y gx mul c1x gy mul sub abs def
 /V2  c2y gx mul c2x gy mul sub abs def
  V2 V1 gt {/V1 V2 def}if
 /g gx dup mul gy dup mul add sqrt def
  V1 dmin g eps add mul lt {true}{false}ifelse
} def

/Recur
{ PopP
  Dist
  {PolP  PolQ
  }
  {Subdiv
   PushR
   PushL
   Recur
   Recur
  } ifelse
} def
```

Better Drawing

```
/PolQ
{ p0x p0y moveto q1x q1y lineto q2x q2y lineto p3x p3y
  lineto stroke
  p0x p0y Dot1R q1x q1y Dot2G q2x q2y Dot2G p3x p3y Dot1R
} def

/Dist % max distance of P1 or P2 from P0P3
{% gx = p3x - p0x
 % gy = p3y - p0y
 % gs  =gx*gx+gy*gy
 % c1x= p1x - p0x
 % c1y= p1y - p0y
 % c2x= p2x - p0x
 % c2y= p2y - p0y
 % sin(a) = gy/g
 % cos(a) = gx/g
 % v1 = -c1x*sin(a)+c1y*cos(a)
 % V1 = -c1x*gy + c1y*gx
 % V2 = -c2x*gy + c2y*gx
 % W1 = 0.3*V1/gs
 % W2 = 0.3*V2/gs
 % qix= pix+Wi
 % qiy= piy-Wi
 % g = Sqrt(gs)
 % V = Max(abs(V1),abs(V2))
 % If V<dmin*(g+eps) Then flat
 /gx  p3x p0x sub def
 /gy  p3y p0y sub def
 /c1x p1x p0x sub def
 /c1y p1y p0y sub def
 /c2x p2x p0x sub def
 /c2y p2y p0y sub def
 /V1  c1y gx mul c1x gy mul sub def
 /V2  c2y gx mul c2x gy mul sub def
 /gs  gx dup mul gy dup mul add def
 /W1 V1 gs div 0.3 mul def
 /W2 V2 gs div 0.3 mul def
 /q1x p1x W1 gy mul add def
 /q1y p1y W1 gx mul sub def
 /q2x p2x W2 gy mul add def
 /q2y p2y W2 gx mul sub def
 /g  gs sqrt def
 /V1 V1 abs def
 /V2 V2 abs def
  V2 V1 gt {/V1 V2 def}if
  V1 dmin g eps add mul lt {true}{false}ifelse
} def
```

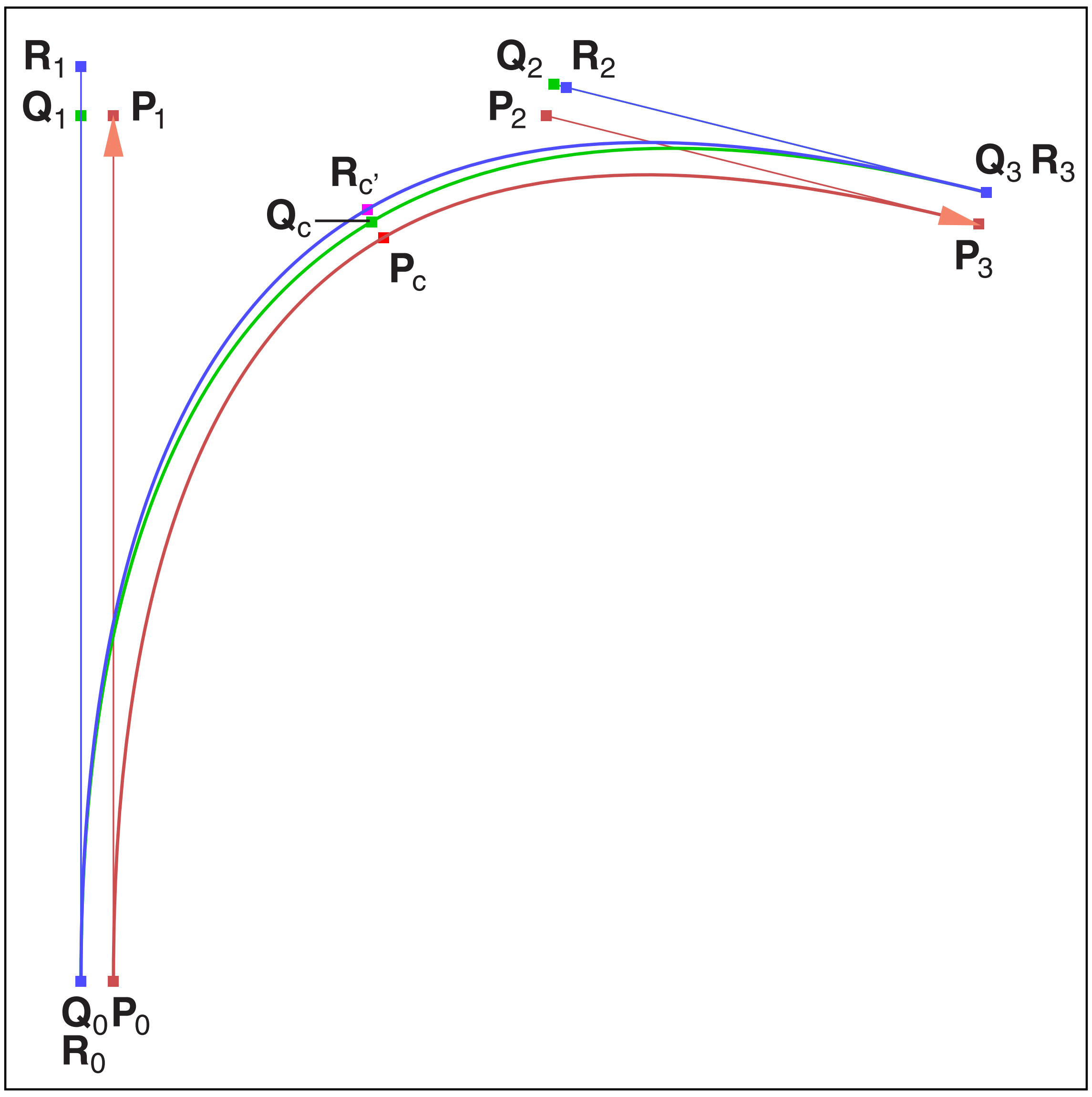
6.8 De Casteljau Subdivision / PostScript Code

```
% Bbox

x0 y0 translate
sx sy scale

Vbox

0.25 mm sx div setlinewidth

0 0 0 1 setcmykcolor
BezP
PolP

/ts   0.500 def
/eps  0.010 def
/dmin 0.005 def

0 1 1 0 setcmykcolor
[0.01 0.005] 0 setdash
PushP
PushP
Recur

0 0 0 1 setcmykcolor
[] 0 setdash
PopP
p0x p0y Dot1K p1x p1y Dot1K p2x p2y Dot1K p3x p3y Dot1K

/buf 20 string def
 0 0 0 1 setcmykcolor
/fh 10.5 sx div def
/Helvetica findfont fh scalefont setfont

0.75 0.95 moveto (dmin) show 0.85 0.95 moveto (= ) show
dmin buf cvs show
0.75 0.89 moveto (eps)  show 0.85 0.89 moveto (= ) show
eps  buf cvs show

showpage
```

7.1 Bézier Offset Curves / Concept

A true offset curve for a parametric polynomial curve is not a polynomial. The Bézier offset curve is an approximation. This approximation should be found without numerical optimization, here by solving a system of three linear equations.
The Bézier offset curves are constructed by shifting P_0 to Q_0 and P_3 to Q_3 in normal direction, by shifting the midpoint $P_c(t=0.5)$ to $R_{c'}$ and the demand for the same slope on the offset curve for a slightly modified parameter t ($R_{c'}$ is near to $t=0.5$).

Red Original curve **P**, tangents and midpoint $\mathbf{P}_c$.
Green Shifted curve **Q**, tangents and midpoint $\mathbf{Q}_c$.
Blue Offset curve **R** by varying the tangent lengths and the parameter t.
$\mathbf{Q}_c$ is moved to $\mathbf{R}_{c'}$. The slopes at $\mathbf{R}_{c'}$ and $\mathbf{P}_c$ are equal.



P0	0.100000	0.100000
P1	0.100000	0.900000
P2	0.500000	0.900000
P3	0.900000	0.800000
Pc	0.350000	0.787500
e	0.030000	
Q0	0.070000	0.100000
Q1	0.070000	0.900000
Q2	0.507276	0.929104
Q3	0.907276	0.829104
Qc	0.338638	0.802052
R0	0.070000	0.100000
R1	0.070000	0.945536
R2	0.518556	0.926284
R3	0.907276	0.829104
Rc	0.334884	0.813413
D0	0.147418431336	
D1	0.008391066282	
D2	-0.004157093856	
D3	-0.001307774271	
dk0	0.056920058416	
dk3	-0.028199281826	
dt	-0.008871170503	
Used equations	3	

D_0 is the main *Cramer* determinant. D_1, D_2, D_3 are the *Cramer* determinants for the unknowns $dk_0=\delta k_0$, $dk_3=\delta k_3$ and $dt=\delta t$ (in PostScript tables we do not use 'δ' for simplicity).

The source code shows the practical sequence of the calculation steps, even if the reader should not be familiar with PostScript.

PostScript float numbers are handled by Acrobat Distiller (and Photoshop and PageMaker) by fixed point integer. This is clearly in contradiction to the PS specifications [1].
One should not expect more than six to eight digits accuracy.

The title graphic of this doc shows a Bézier with loop, rendered by two Offset Béziers and circles for the end caps.

7.2 Bézier Offset Curves / Algorithm

The nomenclature is the same as in chapter1.

$$\mathbf{P} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{P}_0$$

$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

$$\begin{aligned}
\mathbf{a} &= 3(\mathbf{P}_1 - \mathbf{P}_0) + 3(\mathbf{P}_3 - \mathbf{P}_2) - 2(\mathbf{P}_3 - \mathbf{P}_0) \\
\mathbf{b} &= -6(\mathbf{P}_1 - \mathbf{P}_0) - 3(\mathbf{P}_3 - \mathbf{P}_2) + 3(\mathbf{P}_3 - \mathbf{P}_0) \\
\mathbf{c} &= 3(\mathbf{P}_1 - \mathbf{P}_0)
\end{aligned}$$

The tangent vectors point into curve direction $\mathbf{P}_0 \rightarrow \mathbf{P}_1$ and $\mathbf{P}_2 \rightarrow \mathbf{P}_3$.

$$\begin{aligned}
\mathbf{s}_0 &= \mathbf{P}_1 - \mathbf{P}_0 \\
\mathbf{s}_3 &= \mathbf{P}_3 - \mathbf{P}_2
\end{aligned}$$

$$\begin{aligned}
\mathbf{a} &= 3\mathbf{s}_0 + 3\mathbf{s}_3 - 2(\mathbf{P}_3 - \mathbf{P}_0) \\
\mathbf{b} &= -6\mathbf{s}_0 - 3\mathbf{s}_3 + 3(\mathbf{P}_3 - \mathbf{P}_0) \\
\mathbf{c} &= 3\mathbf{s}_0
\end{aligned}$$

Unit normal vectors:

$$\mathbf{n}_0 = \begin{bmatrix} -s_{0y} \\ +s_{0x} \end{bmatrix} \frac{1}{\sqrt{s_{0x}^2 + s_{0y}^2}}$$

$$\mathbf{n}_3 = \begin{bmatrix} -s_{3y} \\ +s_{3x} \end{bmatrix} \frac{1}{\sqrt{s_{3x}^2 + s_{3y}^2}}$$

Shift in normal direction with offset e:

$$\begin{aligned}
\mathbf{Q}_0 &= \mathbf{P}_0 + e\mathbf{n}_0 \\
\mathbf{Q}_3 &= \mathbf{P}_3 + e\mathbf{n}_3
\end{aligned}$$

Coefficients for curve $\mathbf{Q}$:

$$\begin{aligned}
\mathbf{A} &= 3\mathbf{s}_0 + 3\mathbf{s}_3 - 2(\mathbf{Q}_3 - \mathbf{Q}_0) \\
\mathbf{B} &= -6\mathbf{s}_0 - 3\mathbf{s}_3 + 3(\mathbf{Q}_3 - \mathbf{Q}_0) \\
\mathbf{C} &= 3\mathbf{s}_0
\end{aligned}$$

Midpoints at $t = 0.5$:

$$\mathbf{P}_c = \frac{1}{8}\mathbf{a} + \frac{1}{4}\mathbf{b} + \frac{1}{2}\mathbf{c} + \mathbf{P}_0$$

$$\mathbf{Q}_c = \frac{1}{8}\mathbf{A} + \frac{1}{4}\mathbf{B} + \frac{1}{2}\mathbf{C} + \mathbf{Q}_0$$

7.3 Bézier Offset Curves / Algorithm

Slopes:

$$\frac{dp_y}{dp_x} = \frac{dp_y/dt}{dp_x/dt} = \frac{3a_y t^2 + 2b_y t + c_y}{3a_x t^2 + 2b_x t + c_x}$$

$$\frac{dq_y}{dq_x} = \frac{dq_y/dt}{dq_x/dt} = \frac{3A_y t^2 + 2B_y t + C_y}{3A_x t^2 + 2B_x t + C_x}$$

Slopes at $t = 0.5$, described by finite pieces:

$$d\mathbf{P} = \begin{bmatrix} dp_x \\ dp_y \end{bmatrix} = \frac{3}{4}\mathbf{a} + \mathbf{b} + \mathbf{c}$$

$$d\mathbf{Q} = \begin{bmatrix} dq_x \\ dq_y \end{bmatrix} = \frac{3}{4}\mathbf{A} + \mathbf{B} + \mathbf{C}$$

Normal at midpoint of curve $\mathbf{P}$:

$$\mathbf{n}_c = \begin{bmatrix} -dp_y \\ +dp_x \end{bmatrix} \frac{1}{\sqrt{dp_x^2 + dp_y^2}}$$

Offset for midpoint (not exactly at $t=0.5$):

$$\mathbf{R}_{c'} = \mathbf{P}_c + e\mathbf{n}_c$$

Curve $\mathbf{R}$ is a variation of curve $\mathbf{Q}$:

$$\begin{aligned} \mathbf{R} &= \mathbf{Q} + \delta\mathbf{Q} \\ &= (\mathbf{A} + \delta\mathbf{A})t^3 + (\mathbf{B} + \delta\mathbf{B})t^2 + (\mathbf{C} + \delta\mathbf{C})t + \mathbf{Q}_0 + (3\mathbf{a}t^2 + 2\mathbf{b}t + \mathbf{c})\delta t \end{aligned}$$

The variation by δt has to use the slope of curve $\mathbf{P}$ because the slopes of $\mathbf{R}$ and $\mathbf{P}$ will be matched.
Variation at $t = 0.5$:

$$\delta\mathbf{Q} = \frac{1}{8}\delta\mathbf{A} + \frac{1}{4}\delta\mathbf{B} + \frac{1}{2}\delta\mathbf{C} + (\frac{3}{4}\mathbf{a} + \mathbf{b} + \mathbf{c})\delta t$$

$$\delta\mathbf{Q} = \frac{1}{8}\delta\mathbf{A} + \frac{1}{4}\delta\mathbf{B} + \frac{1}{2}\delta\mathbf{C} + d\mathbf{P}\,\delta t$$

Variations $\delta\mathbf{A}$, $\delta\mathbf{B}$, $\delta\mathbf{C}$ by modifying the tangent lengths:

$$\begin{aligned} \mathbf{s}_0 &\rightarrow (1 + \delta k_0)\mathbf{s}_0 \\ \mathbf{s}_3 &\rightarrow (1 + \delta k_3)\mathbf{s}_3 \\ \delta\mathbf{A} &= +3\mathbf{s}_0\,\delta k_0 + 3\mathbf{s}_3\,\delta k_3 \\ \delta\mathbf{B} &= -6\mathbf{s}_0\,\delta k_0 - 3\mathbf{s}_3\,\delta k_3 \\ \delta\mathbf{C} &= +3\mathbf{s}_0\,\delta k_0 \end{aligned}$$

The total variation moves $\mathbf{Q}_c$ to $\mathbf{R}_{c'}$:

$$\delta\mathbf{Q} = \frac{3}{8}\mathbf{s}_0\,\delta k_0 - \frac{3}{8}\mathbf{s}_3\,\delta k_3 + d\mathbf{P}\,\delta t = \mathbf{R}_{c'} - \mathbf{Q}_c$$

This vector equation delivers two linear equations for three unknowns $\delta k_0, \delta k_3$ and δt.

7.4 Bézier Offset Curves / Algorithm

Matching the slope at t=0.5:

$$\begin{bmatrix} dr_x \\ dr_y \end{bmatrix} = \frac{d\mathbf{R}}{dt} = 3(\mathbf{A}+\delta\mathbf{A})t^2 + 2(\mathbf{B}+\delta\mathbf{B})t + (\mathbf{C}+\delta\mathbf{C}) + (6\mathbf{a}t + 2\mathbf{b})\delta t$$

$$\begin{bmatrix} dr_x \\ dr_y \end{bmatrix} = \frac{3}{4}(\mathbf{A}+\delta\mathbf{A}) + (\mathbf{B}+\delta\mathbf{B}) + (\mathbf{C}+\delta\mathbf{C}) + (3\mathbf{a}+2\mathbf{b})\delta t$$

$$\begin{bmatrix} dr_x \\ dr_y \end{bmatrix} = [\frac{3}{4}\mathbf{A}+\mathbf{B}+\mathbf{C}] + [\frac{3}{4}\delta\mathbf{A}+\delta\mathbf{B}+\delta\mathbf{C}] + [(3\mathbf{a}+2\mathbf{b})\delta t]$$

Substitutions for each bracket content:

$$\begin{bmatrix} dr_x \\ dr_y \end{bmatrix} = [d\mathbf{Q}] + [-\frac{3}{4}\mathbf{s}_0\delta k_0 - \frac{3}{4}\mathbf{s}_3\delta k_3] + [3(-\mathbf{s}_0+\mathbf{s}_3)\delta t]$$

Equal slope at $\mathbf{P}_c$ and $\mathbf{R}_{c'}$ delivers the third equation:

$$\frac{dr_y}{dr_x} = \frac{dp_y}{dp_x}$$

$$-dp_x\, dr_y + dp_y\, dr_x = 0$$

$$\begin{aligned} &-dp_x(-\frac{3}{4}s_{0y}\,\delta k_0 \;\; -\frac{3}{4}s_{3y}\,\delta k_3 \;\; +3(-s_{0y}+s_{3y})\delta t) + \\ &+dp_y(-\frac{3}{4}s_{0x}\,\delta k_0 \;\; -\frac{3}{4}s_{3x}\,\delta k_3 \;\; +3(-s_{0x}+s_{3x})\delta t) = -dq_x\,dp_y + dq_y\,dp_x \end{aligned}$$

$$\begin{aligned} &\frac{3}{4}(-s_{0x}dp_y + s_{0y}dp_x)\delta k_0 + \frac{3}{4}(-s_{3x}dp_y + s_{3y}dp_x)\delta k_3 + \\ &+3\left[-(s_{0x}-s_{3x})dp_y + (s_{0y}-s_{3y})dp_x\right]\delta t \;\; = -dq_x\,dp_y + dq_y\,dp_x \end{aligned}$$

The complete system of equations for $\delta\mathbf{x} = (\delta k_0, \delta k_3, \delta t)^T$, after multiplying by some factors:

$$\mathbf{A}\,\delta\mathbf{x} = \mathbf{y}$$

$$\mathbf{A} = \begin{bmatrix} s_{0x} & -s_{3x} & \frac{8}{3}dp_x \\ s_{0y} & -s_{3y} & \frac{8}{3}dp_y \\ -s_{0x}\,dp_y + s_{0y}\,dp_x & -s_{3x}\,dp_y + s_{3y}\,dp_x & 4\left[-(s_{0x}-s_{3x})dp_y + (s_{0y}-s_{3y})dp_x\right] \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} \frac{8}{3}(r_{cx}-q_{cx}) \\ \frac{8}{3}(r_{cy}-q_{cy}) \\ \frac{4}{3}(-dq_x\,dp_y + dq_y\,dp_x) \end{bmatrix}$$

7.5 Bézier Offset Curves / Algorithm

In the third equation we can replace $(-dp_y, dp_x)^T$ by $\mathbf{n}_c$. The normal vector is a unit vector. This might be useful for a better normalization. More important is perhaps the geometrical interpretation: each component delivers the signed length of the first vector, projected onto the normal vector.

$$\mathbf{A} = \begin{bmatrix} s_{0x} & -s_{3x} & \frac{8}{3}dp_x \\ s_{0y} & -s_{3y} & \frac{8}{3}dp_y \\ \mathbf{s}_0^T\mathbf{n}_c & \mathbf{s}_3^T\mathbf{n}_c & 4(\mathbf{s}_0-\mathbf{s}_3)^T\mathbf{n}_c \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} \frac{8}{3}(r_{cx}-q_{cx}) \\ \frac{8}{3}(r_{cy}-q_{cy}) \\ \frac{4}{3}\,d\mathbf{Q}^T\mathbf{n}_c \end{bmatrix}$$

Solution by Cramer ($\mathbf{a}_k$ are columns of $\mathbf{A}$):

$$\begin{aligned} D_0 &= \det(\mathbf{A}) \\ D_1 &= \det(\mathbf{y}, \mathbf{a}_2, \mathbf{a}_3) \\ D_2 &= \det(\mathbf{a}_1, \mathbf{y}, \mathbf{a}_3) \\ D_3 &= \det(\mathbf{a}_1, \mathbf{a}_2, \mathbf{y}) \\ \delta x_i &= \frac{D_i}{D_0} \end{aligned}$$

Check for division overflow, using 'max' as largest allowed absolute value for any solution δx_i.

1. $|D_0| > 1$ Division is executable
2. $|D_0| \le 1$

   $|D_i| < |D_0| \cdot \max$ Division is executable

   $|D_i| \ge |D_0| \cdot \max$ Division is not executable

Only small solutions for the variations are expected.

Therefore $\max = 1$ is a reasonable limit. Actually 0.9 for $\delta k0, \delta k3$ and 0.45 for δt.

The offset curve has these control points:

$$\begin{aligned} \mathbf{R}_0 &= \mathbf{Q}_0 \\ \mathbf{R}_1 &= \mathbf{Q}_0 + (1+\delta k_0)\mathbf{s}_0 \\ \mathbf{R}_2 &= \mathbf{Q}_3 - (1+\delta k_3)\mathbf{s}_3 \\ \mathbf{R}_3 &= \mathbf{Q}_3 \end{aligned}$$

It can be shown that collinear control points lead to solutions of the type 0/0 which indicates multiple solutions. In this case (as found by the division check) one can set simply $\delta k0 = \delta k3 = \delta dt = 0$. For a straight line only the offset shift is applied, no further correction.

7.6 Bézier Offset Curves / Algorithm

The algorithm is based on several assumptions: the original Bézier curve is somewhat regular and the task is in a practical sense reasonable:

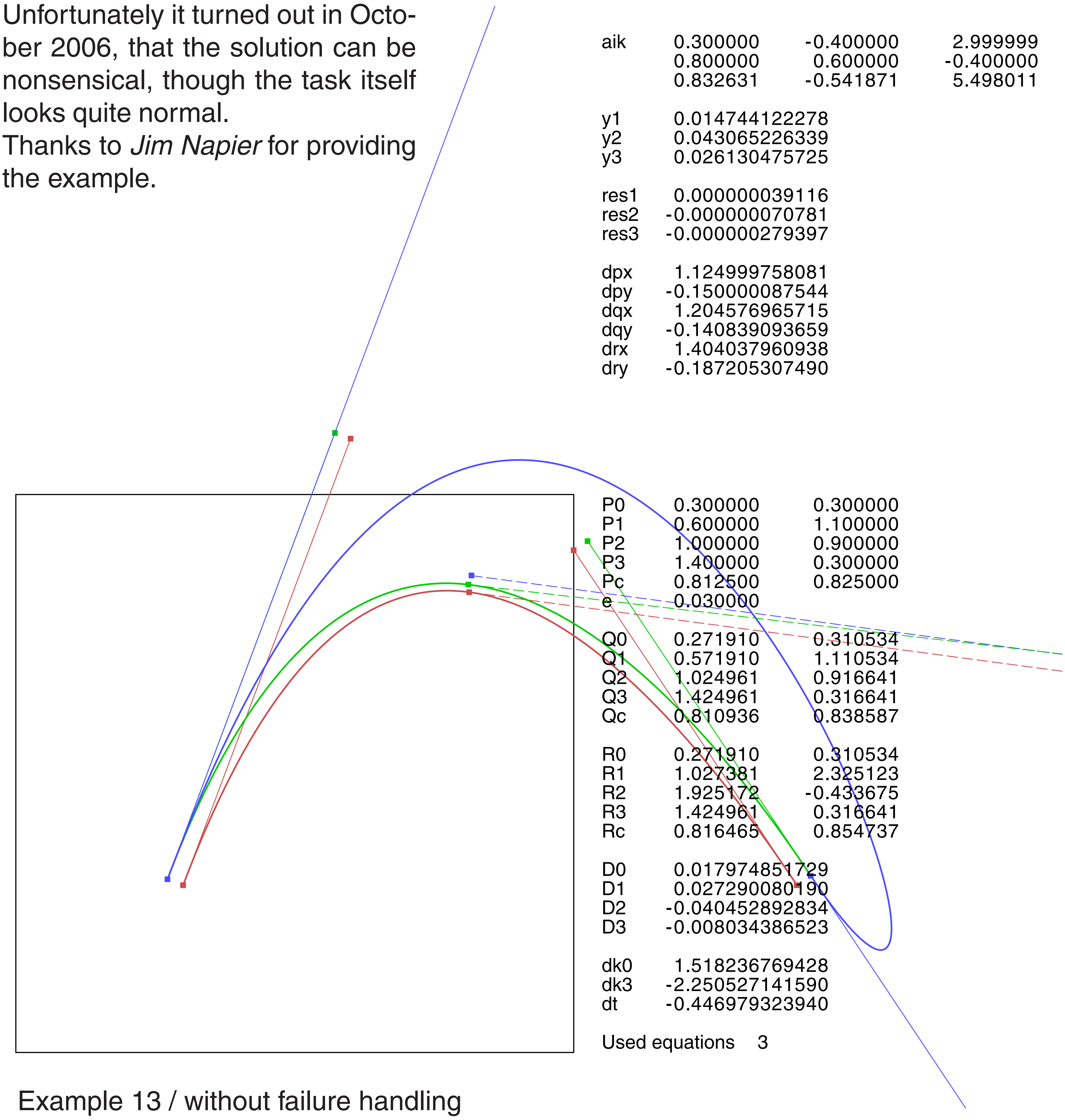
1. Tangent vectors have finite lengths.
2. The offset ’e’ is not too large. Moderately thick lines are useful applications.
3. The tangent directions as arrows point into the *same* direction if the control points are collinear. Opposite directions would deliver a highly unstable straight line.
4. The curves are roughly normalized for a unit bounding box.

The final algorithm was developed in August 2004 on Wasini Island, Kenya, using only paper and pencil and corrected in July 2005.
Thanks to *Rafael Latowicz* for improving the mathematical description and further contributions concerning the analysis of a subtle bug.

Unfortunately it turned out in October 2006, that the solution can be nonsensical, though the task itself looks quite normal.
Thanks to *Jim Napier* for providing the example.

aik	0.300000	-0.400000	2.999999
	0.800000	0.600000	-0.400000
	0.832631	-0.541871	5.498011

y1	0.014744122278
y2	0.043065226339
y3	0.026130475725
res1	0.000000039116
res2	-0.000000070781
res3	-0.000000279397
dpx	1.124999758081
dpy	-0.150000087544
dqx	1.204576965715
dqy	-0.140839093659
drx	1.404037960938
dry	-0.187205307490



P0	0.300000	0.300000
P1	0.600000	1.100000
P2	1.000000	0.900000
P3	1.400000	0.300000
Pc	0.812500	0.825000
e	0.030000	
Q0	0.271910	0.310534
Q1	0.571910	1.110534
Q2	1.024961	0.916641
Q3	1.424961	0.316641
Qc	0.810936	0.838587
R0	0.271910	0.310534
R1	1.027381	2.325123
R2	1.925172	-0.433675
R3	1.424961	0.316641
Rc	0.816465	0.854737

D0	0.017974851729
D1	0.027290080190
D2	-0.040452892834
D3	-0.008034386523
dk0	1.518236769428
dk3	-2.250527141590
dt	-0.446979323940

Used equations 3

Example 13 / without failure handling

7.7 Bézier Offset Curves / Algorithm

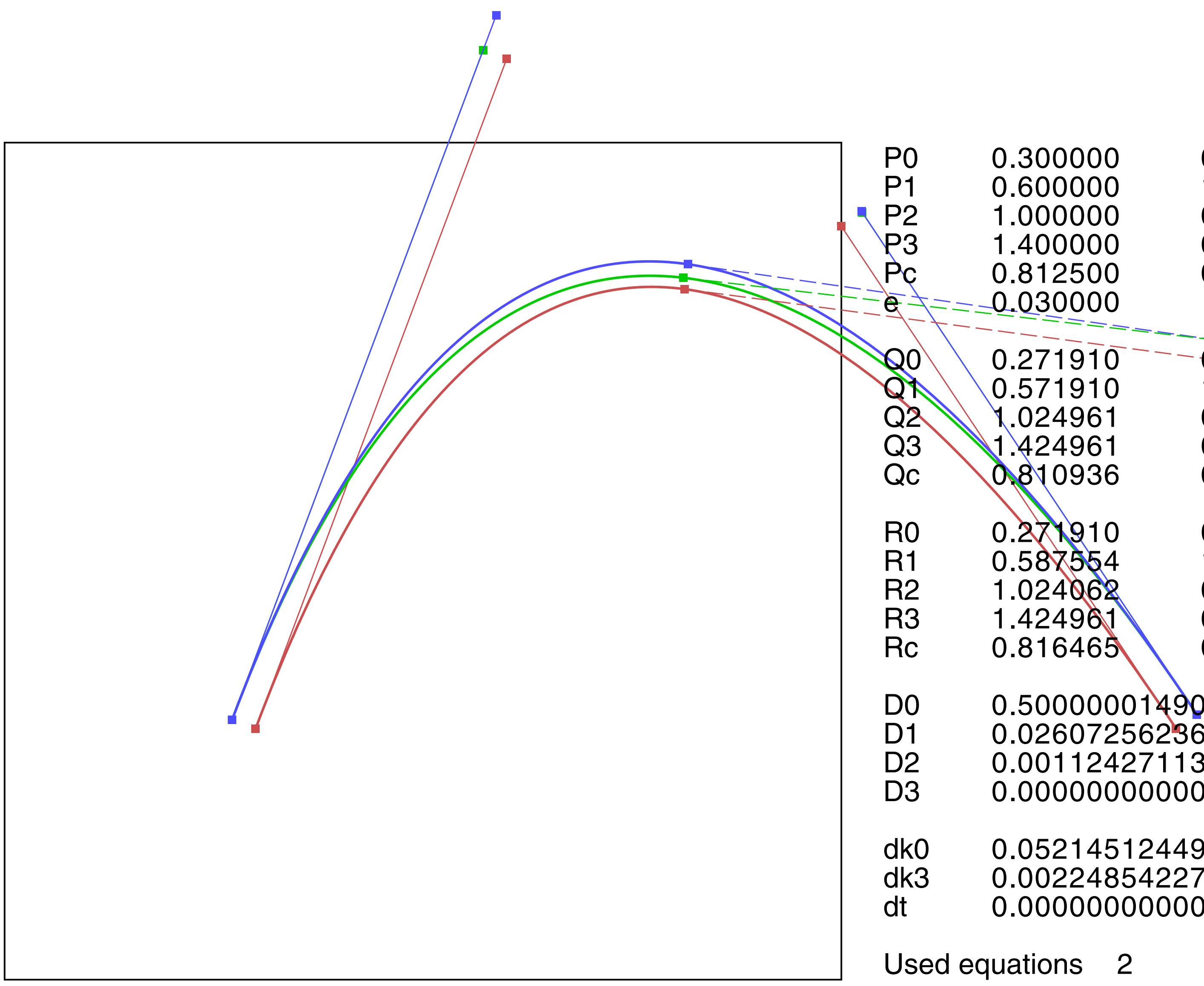
So far this phenomenon cannot be explained. The determinant of **A** is further evaluated in Appendix 1. It is still not clear which geometrical condition leads to a zero determinant. Failure handling is done at present as follows:
The third equation, the slope matching and the variation by δt are not used. The remaining two equations are written here still as three equations, for convenience:

$$\mathbf{A} = \begin{bmatrix} s_{0x} & -s_{3x} & 0 \\ s_{0y} & -s_{3y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} (8/3)(r_{cx} - q_{cx}) \\ (8/3)(r_{cy} - q_{cy}) \\ 0 \end{bmatrix}$$

aik	0.300000	-0.400000	0.000000
	0.800000	0.600000	0.000000
	0.000000	0.000000	1.000000

y1	0.014744122278
y2	0.043065226339
y3	0.000000000000
res1	0.000000000000
res2	0.000000000000
res3	0.000000000000
dpx	1.124999758081
dpy	-0.150000087544
dqx	1.204576965715
dqy	-0.140839093659
drx	1.192169778469
dry	-0.171114329269



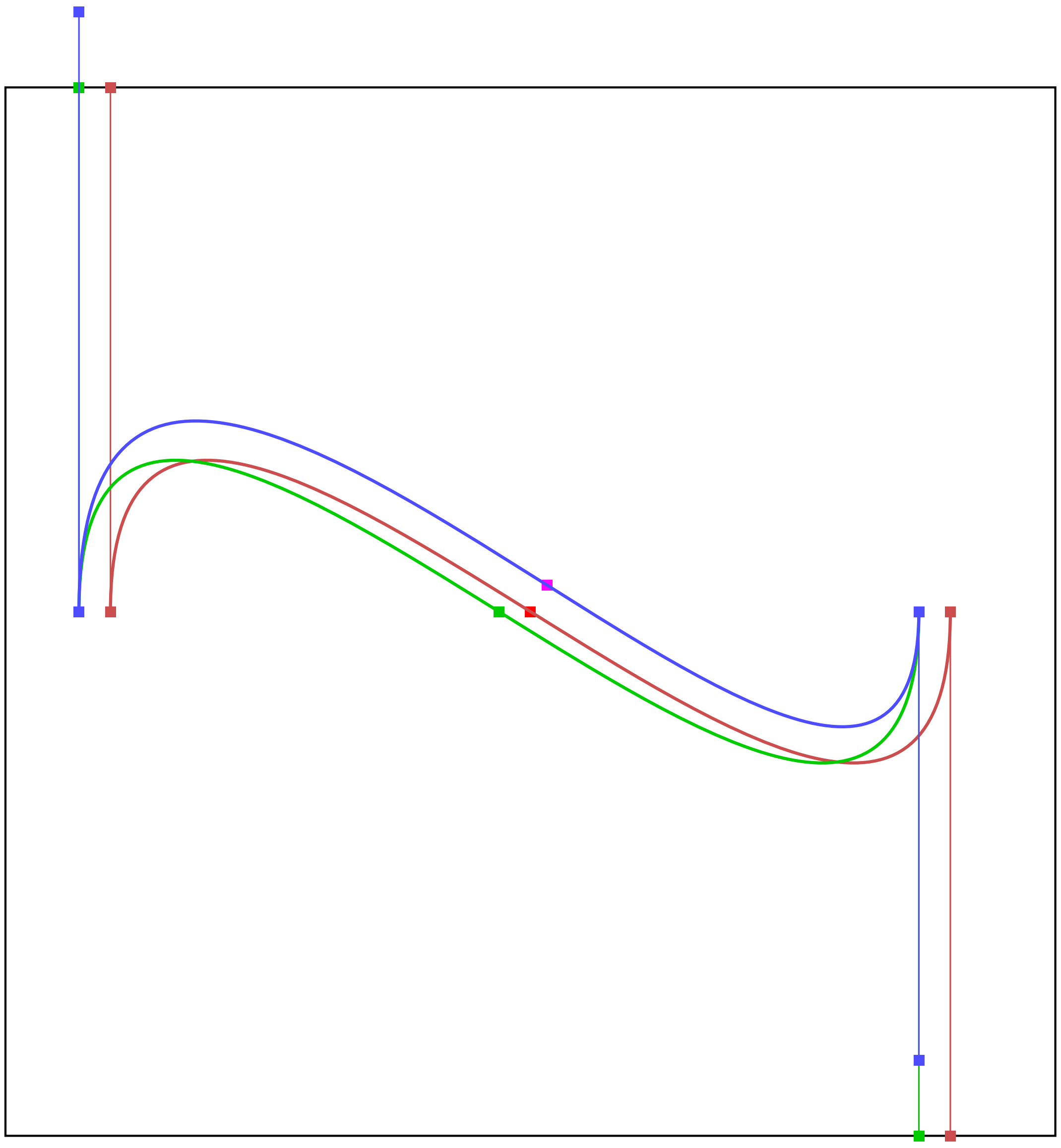
P0	0.300000	0.300000
P1	0.600000	1.100000
P2	1.000000	0.900000
P3	1.400000	0.300000
Pc	0.812500	0.825000
e	0.030000	
Q0	0.271910	0.310534
Q1	0.571910	1.110534
Q2	1.024961	0.916641
Q3	1.424961	0.316641
Qc	0.810936	0.838587
R0	0.271910	0.310534
R1	0.587554	1.152250
R2	1.024062	0.917990
R3	1.424961	0.316641
Rc	0.816465	0.854737

D0	0.500000014901
D1	0.026072562364
D2	0.001124271137
D3	0.000000000000
dk0	0.052145124497
dk3	0.002248542277
dt	0.000000000000

Used equations 2

Example 13 / with failure handling

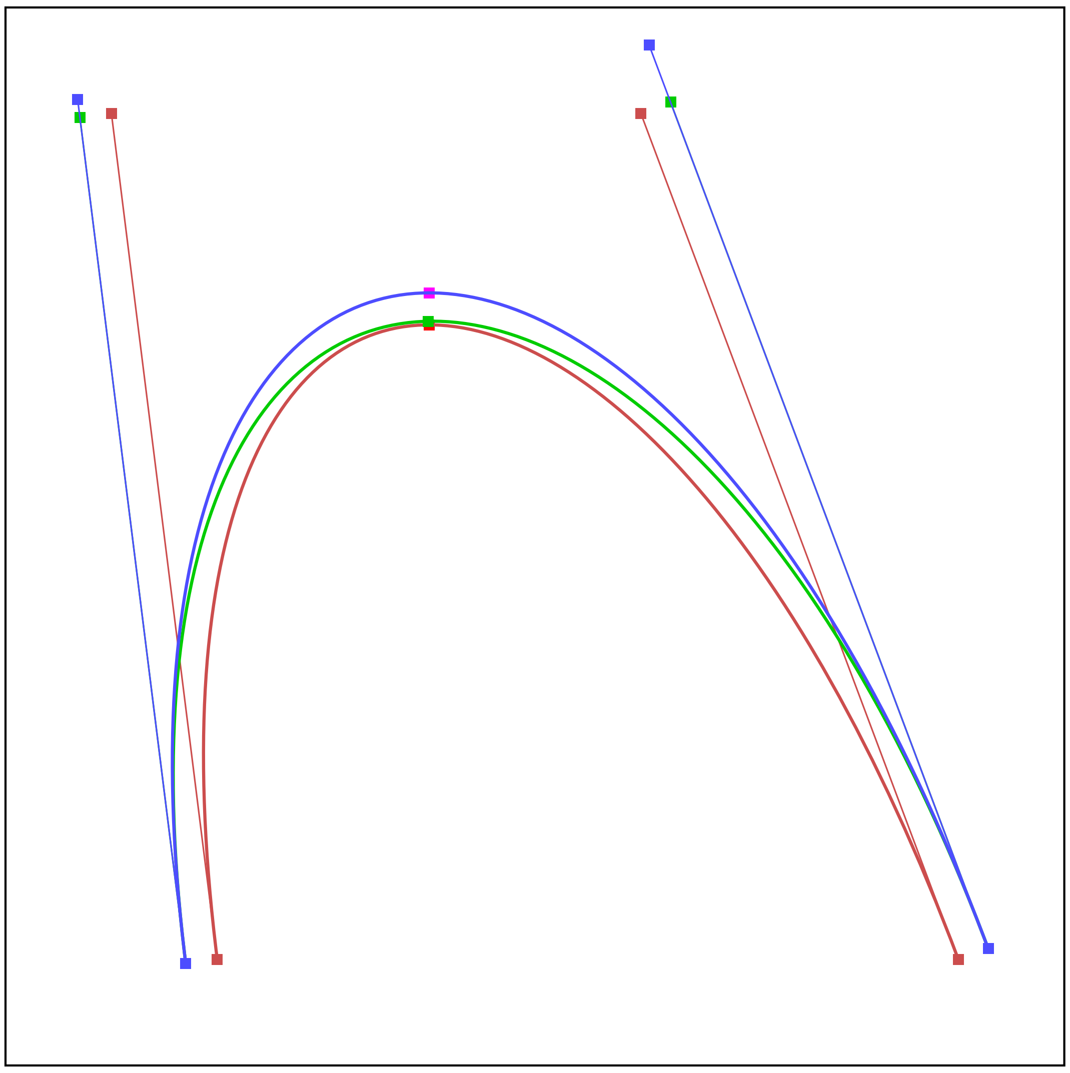
7.8 Bézier Offset Curves / Examples



P0	0.100000	0.500000
P1	0.100000	1.000000
P2	0.900000	0.000000
P3	0.900000	0.500000
Pc	0.500000	0.500000
e	0.030000	
Q0	0.070000	0.500000
Q1	0.070000	1.000000
Q2	0.870000	0.000000
Q3	0.870000	0.500000
Qc	0.470000	0.500000
R0	0.070000	0.500000
R1	0.070000	1.072170
R2	0.870000	0.072170
R3	0.870000	0.500000
Rc	0.515900	0.525440
D0	1.356797101485	
D1	0.195839981442	
D2	-0.195839713221	
D3	0.051897403873	
dk0	0.144339912466	
dk3	-0.144339718751	
dt	0.038249938328	

Used equations 3

Example 2

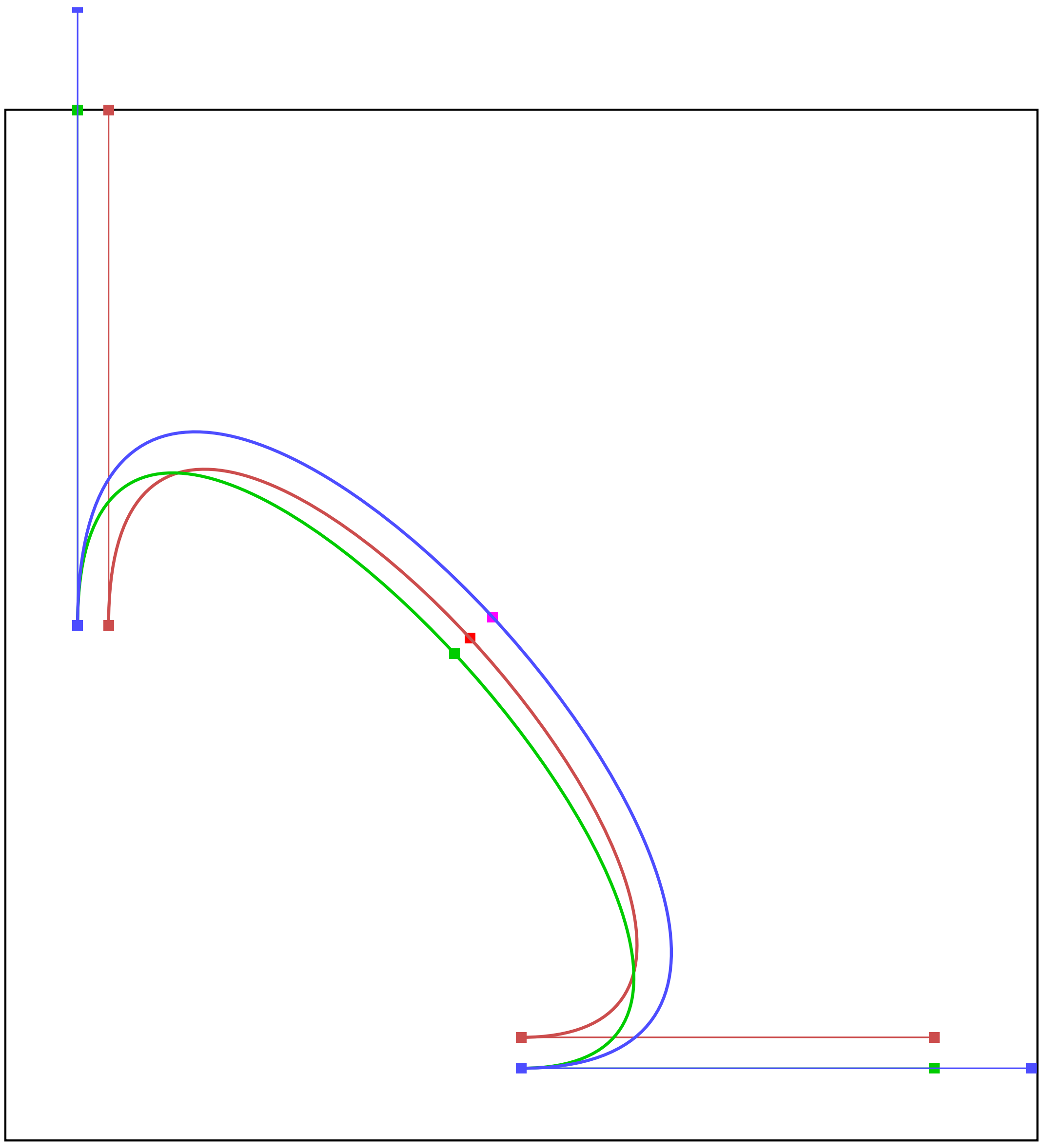


P0	0.200000	0.100000
P1	0.100000	0.900000
P2	0.600000	0.900000
P3	0.900000	0.100000
Pc	0.400000	0.700000
e	0.030000	
Q0	0.170232	0.096279
Q1	0.070232	0.896279
Q2	0.628090	0.910534
Q3	0.928090	0.110534
Qc	0.399161	0.703406
R0	0.170232	0.096279
R1	0.068118	0.913185
R2	0.607836	0.964544
R3	0.928090	0.110534
Rc	0.400000	0.730000
D0	-2.048000090571	
D1	-0.043279847750	
D2	-0.138266392938	
D3	-0.020996343737	
dk0	0.021132737791	
dk3	0.067512874893	
dt	0.010252120186	

Used equations 3

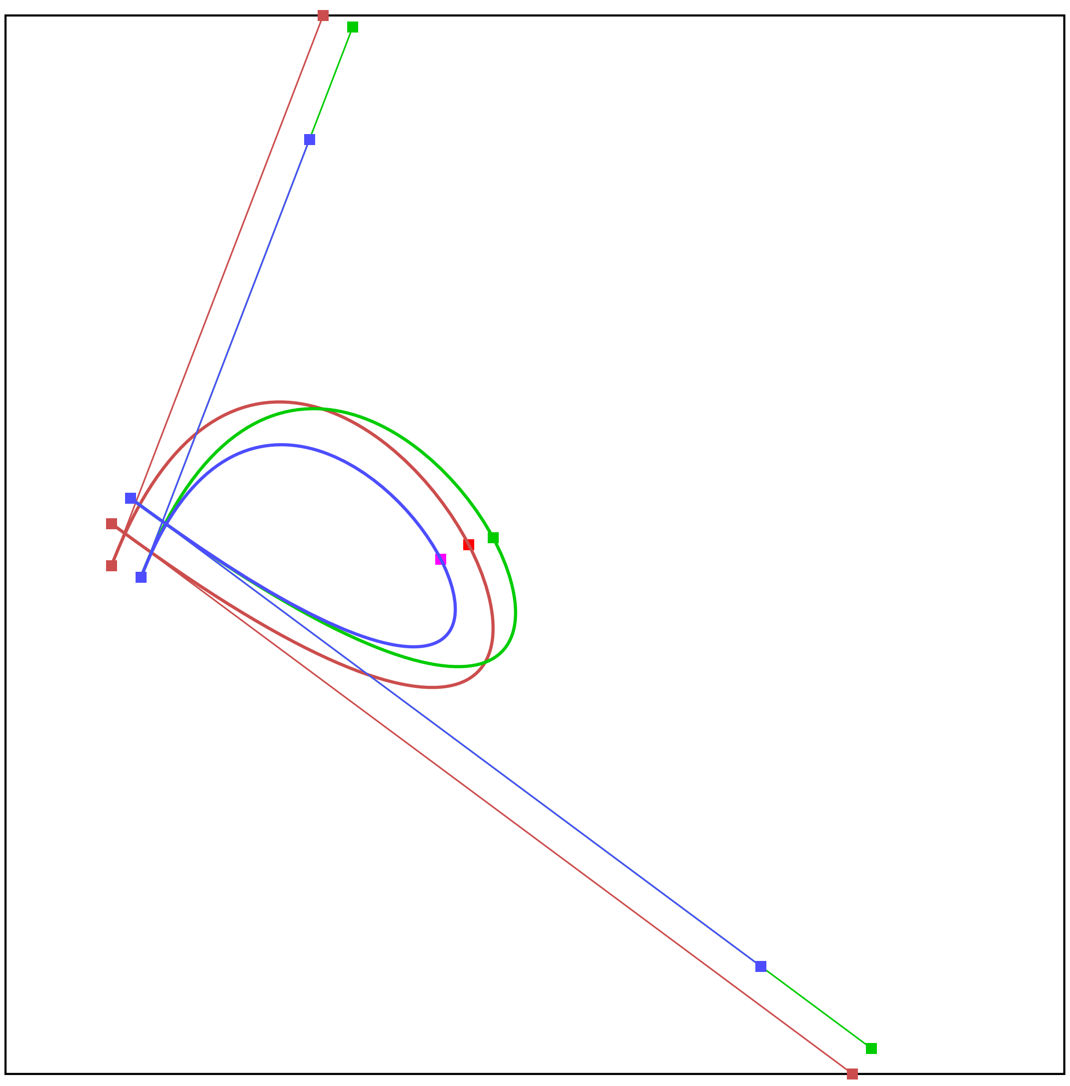
Example 3

7.9 Bézier Offset Curves / Examples



P0	0.100000	0.500000
P1	0.100000	1.000000
P2	0.900000	0.100000
P3	0.500000	0.100000
Pc	0.450000	0.487500
e	0.030000	
Q0	0.070000	0.500000
Q1	0.070000	1.000000
Q2	0.900000	0.070000
Q3	0.500000	0.070000
Qc	0.435000	0.472500
R0	0.070000	0.500000
R1	0.070000	1.099430
R2	0.994015	0.070000
R3	0.500000	0.070000
Rc	0.472044	0.507848
D0	-1.211860295336	
D1	-0.240989595376	
D2	-0.284832328766	
D3	-0.002408342785	
dk0	0.198859224430	
dk3	0.235037268035	
dt	0.001987310494	
Used equations	3	

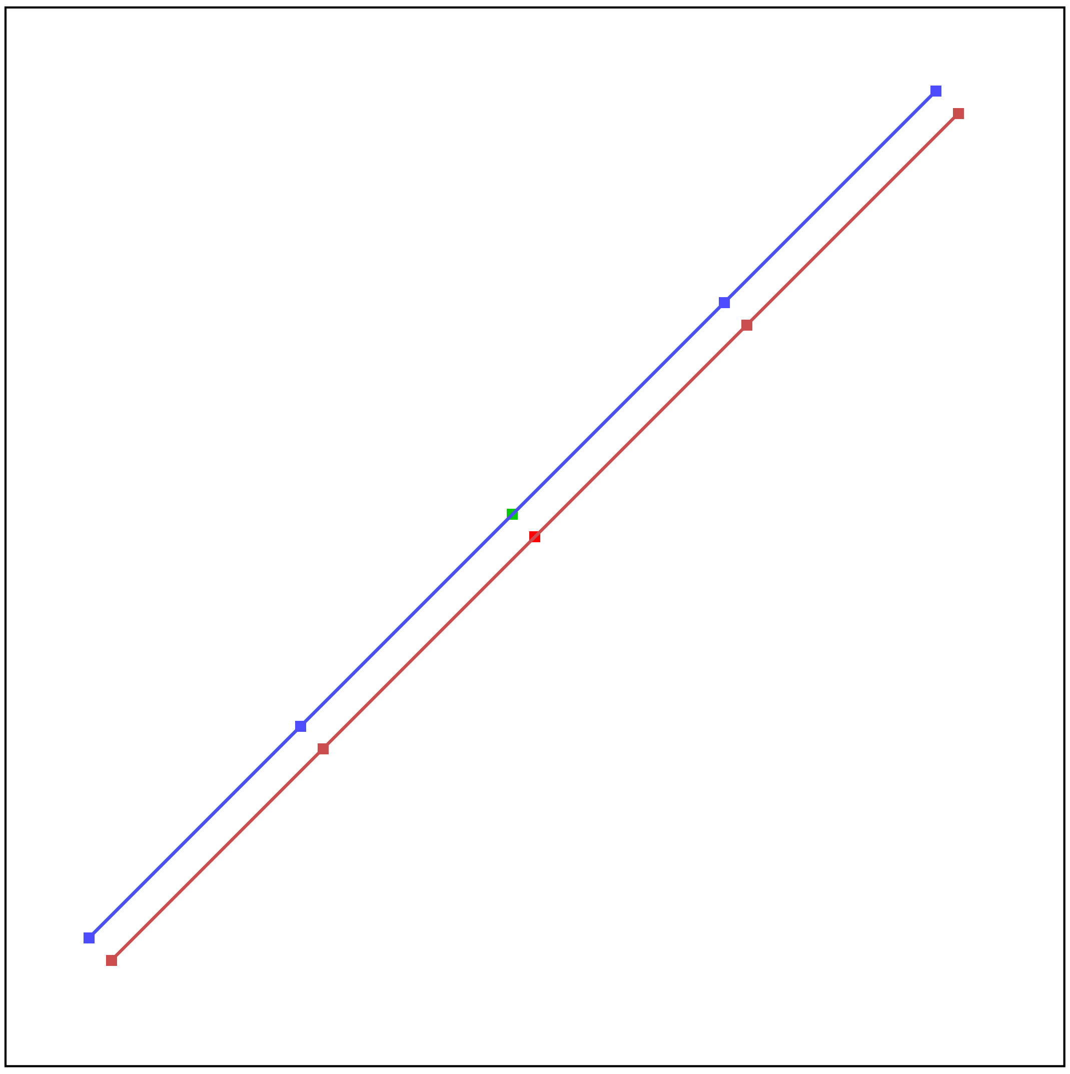
Example 4



P0	0.100000	0.520000
P1	0.800000	0.000000
P2	0.300000	1.000000
P3	0.100000	0.480000
Pc	0.437500	0.500000
e	0.030000	
Q0	0.117890	0.544082
Q1	0.817890	0.024082
Q2	0.328000	0.989231
Q3	0.128000	0.469231
Qc	0.460445	0.506656
R0	0.117890	0.544082
R1	0.713283	0.101790
R2	0.287010	0.882656
R3	0.128000	0.469231
Rc	0.410893	0.486142
D0	-2.182458868432	
D1	0.326143480714	
D2	0.447298626001	
D3	0.029370345296	
dk0	-0.149438536729	
dk3	-0.204951685200	
dt	-0.013457457479	
Used equations	3	

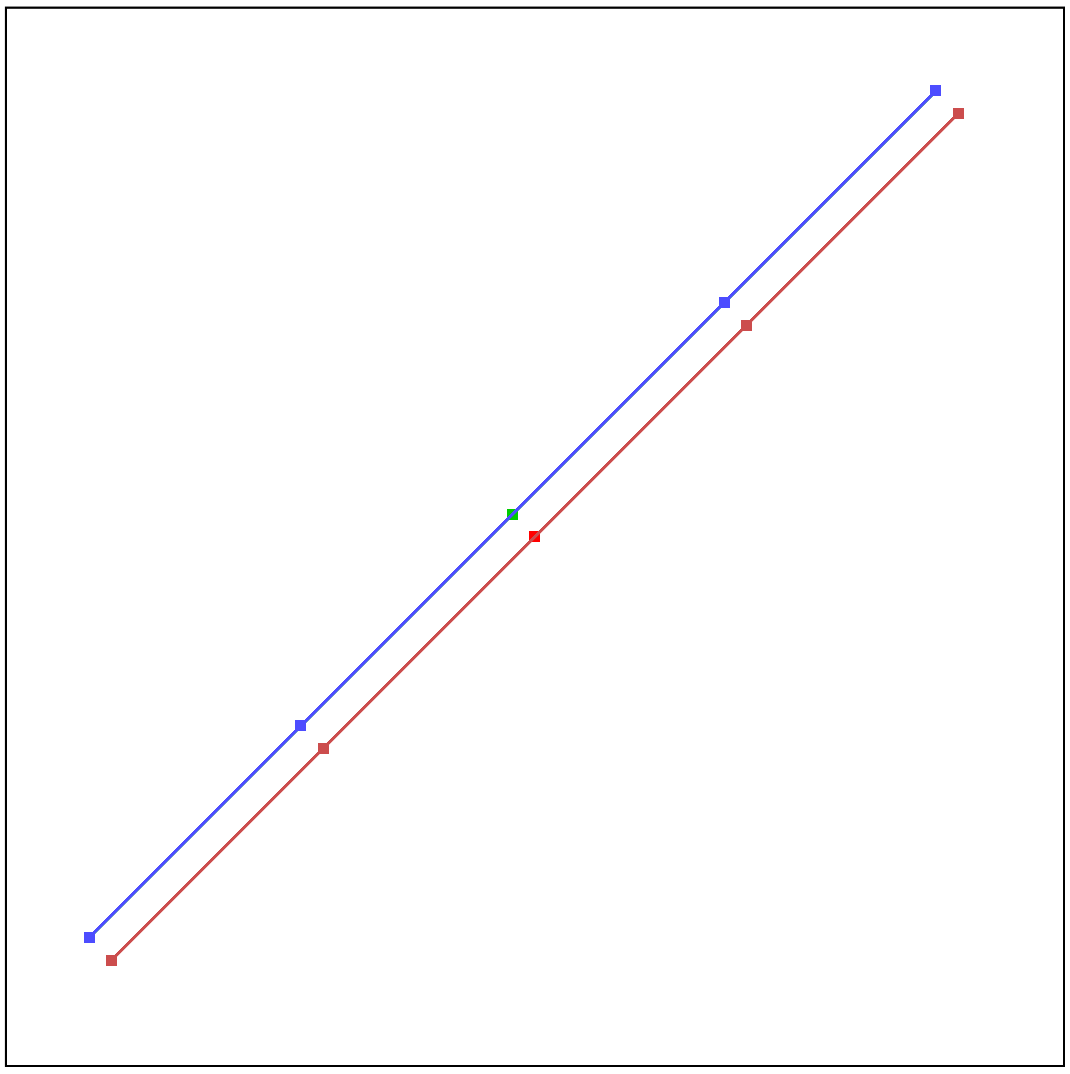
Example 5

7.10 Bézier Offset Curves / Examples



P0	0.100000	0.100000
P1	0.300000	0.300000
P2	0.700000	0.700000
P3	0.900000	0.900000
Pc	0.500000	0.500000
e	0.030000	
Q0	0.078787	0.121213
Q1	0.278787	0.321213
Q2	0.678787	0.721213
Q3	0.878787	0.921213
Qc	0.478787	0.521213
R0	0.078787	0.121213
R1	0.278787	0.321213
R2	0.678787	0.721213
R3	0.878787	0.921213
Rc	0.478787	0.521213
D0	0.000000000000	
D1	-0.000000015895	
D2	-0.000000015895	
D3	0.000000000000	
dk0	0.000000000000	
dk3	0.000000000000	
dt	0.000000000000	
Used equations	0	

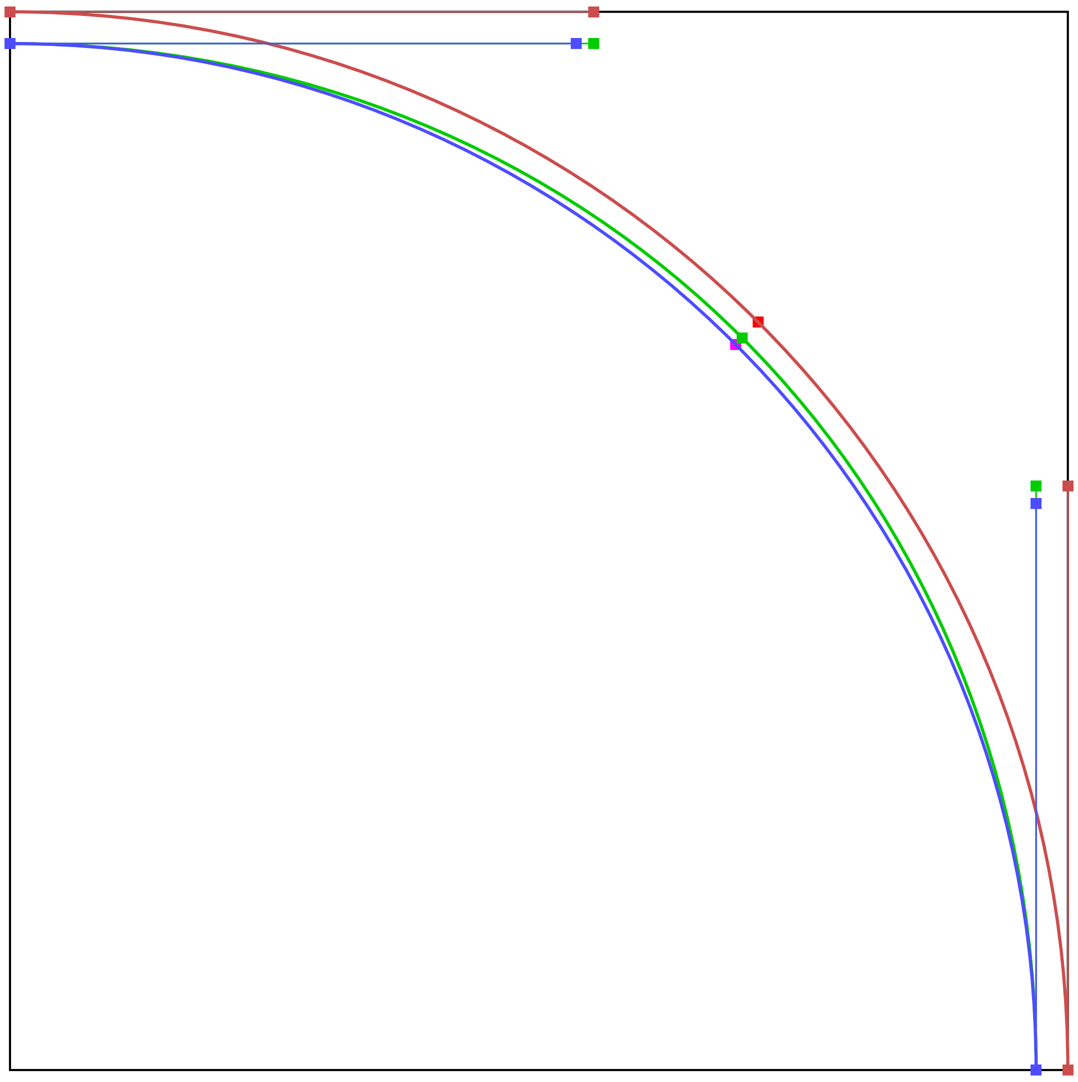
Example 6: straight line



P0	0.100000	0.100000
P1	0.300000	0.300100
P2	0.700000	0.699900
P3	0.900000	0.900000
Pc	0.500000	0.500000
e	0.030000	
Q0	0.078781	0.121208
Q1	0.278782	0.321308
Q2	0.678781	0.721108
Q3	0.878781	0.921208
Qc	0.478782	0.521208
R0	0.078781	0.121208
R1	0.278784	0.321310
R2	0.678784	0.721110
R3	0.878781	0.921208
Rc	0.478789	0.521215
D0	0.000000060355	
D1	0.000000000001	
D2	-0.000000000001	
D3	0.000000000000	
dk0	0.000011728651	
dk3	-0.000011721293	
dt	0.000005860646	
Used equations	3	

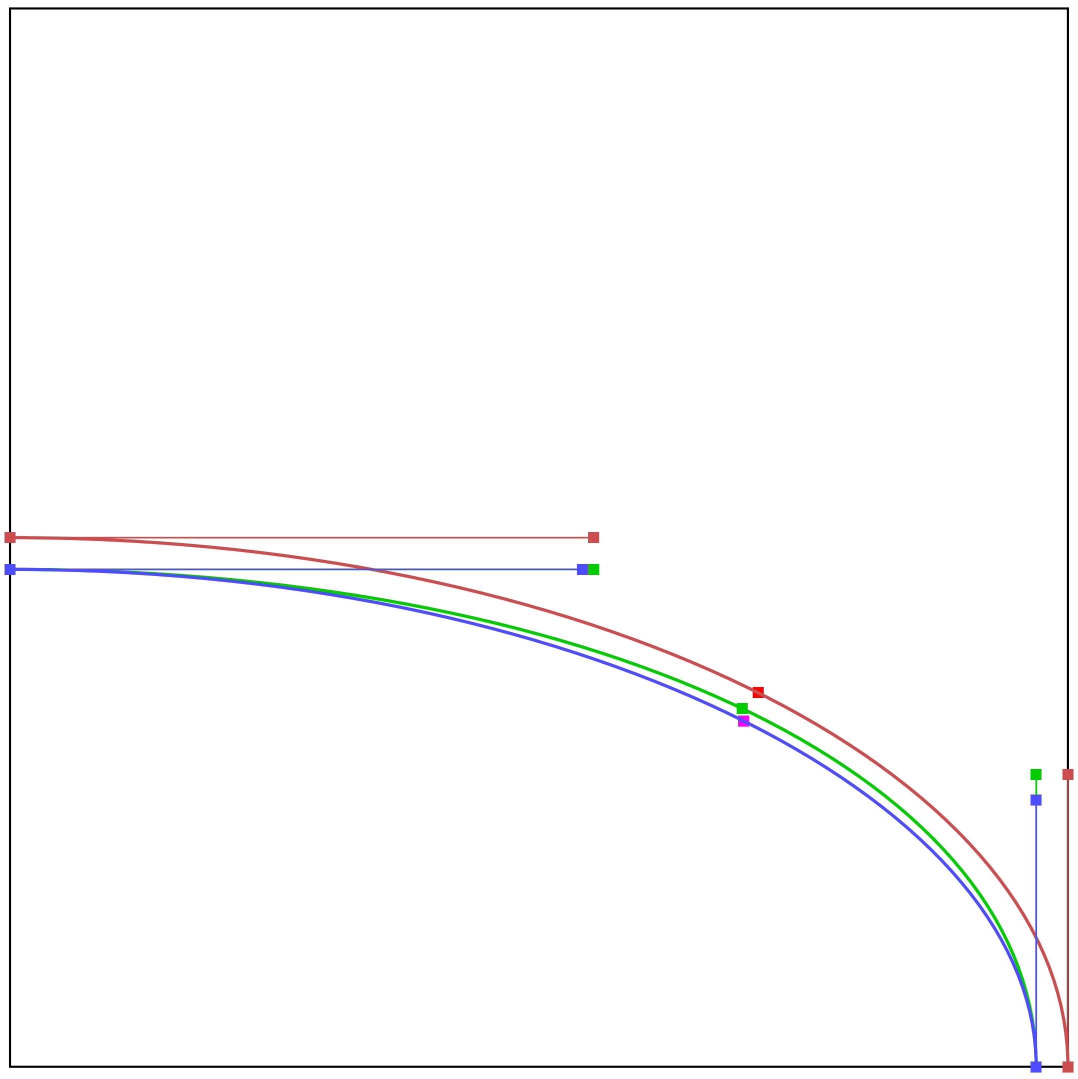
Example 7: almost straight line

7.11 Bézier Offset Curves / Examples



P0	1.000000	0.000000
P1	1.000000	0.552000
P2	0.552000	1.000000
P3	0.000000	1.000000
Pc	0.707000	0.707000
e	0.030000	
Q0	0.970000	0.000000
Q1	0.970000	0.552000
Q2	0.552000	0.970000
Q3	0.000000	0.970000
Qc	0.692000	0.692000
R0	0.970000	0.000000
R1	0.970000	0.535431
R2	0.535431	0.970000
R3	0.000000	0.970000
Rc	0.685787	0.685787
D0	-0.296470523936	
D1	0.008898681764	
D2	0.008898681764	
D3	0.000000000000	
dk0	-0.030015402211	
dk3	-0.030015402211	
dt	0.000000000000	
Used equations	3	

Example 8: circle segment



P0	1.000000	0.000000
P1	1.000000	0.276000
P2	0.552000	0.500000
P3	0.000000	0.500000
Pc	0.707000	0.353500
e	0.030000	
Q0	0.970000	0.000000
Q1	0.970000	0.276000
Q2	0.552000	0.470000
Q3	0.000000	0.470000
Qc	0.692000	0.338500
R0	0.970000	0.000000
R1	0.970000	0.252160
R2	0.540794	0.470000
R3	0.000000	0.470000
Rc	0.693584	0.326667
D0	-0.093752188464	
D1	0.008097901781	
D2	0.001903250722	
D3	0.000499481304	
dk0	-0.086375603540	
dk3	-0.020300865605	
dt	-0.005327675819	
Used equations	3	

Example 9: ellipse segment

7.12 Bézier Offset Curves / PostScript Code

```
%!PS-Adobe-3.0   EPSF-3.0
%%BoundingBox:   0 0 510 340
%%Creator:       Gernot Hoffmann
%%Title:         BezOffs11
%%CreationDate:  November 04 / 2006

% Bezier Offset Curve

% Disable setpagedevice
 /setpagedevice {pop} bind def

% Use extended bounding box for additional information

% Standard BBox 0 0 510 340
% Extended BBox 0 0 560 560

% Show midpoint slope: true or false
/DrawMid false def

/Typ 11 def

Typ 1 eq {
/p0x 0.1 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.5 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.8 def
/e +0.03 def               } if
Typ 2 eq { % like sine full wave
/p0x 0.1 def /p0y  0.5 def
/p1x 0.1 def /p1y  1.0 def
/p2x 0.9 def /p2y  0.0 def
/p3x 0.9 def /p3y  0.5 def
/e +0.03 def               } if
Typ 3 eq { % like distorted sine halfwave
/p0x 0.2 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.6 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.1 def
/e +0.03 def               } if
Typ 4 eq {
/p0x 0.1 def /p0y  0.5 def
/p1x 0.1 def /p1y  1.0 def
/p2x 0.9 def /p2y  0.1 def
/p3x 0.5 def /p3y  0.1 def
/e +0.03 def               } if
Typ 5 eq { % loop
/p0x 0.1 def /p0y  0.52 def
/p1x 0.8 def /p1y  0.0 def
/p2x 0.3 def /p2y  1.0 def
/p3x 0.1 def /p3y  0.48 def
/e +0.03 def               } if
Typ 6 eq { % strictly linear
/p0x 0.1 def /p0y  0.1 def
/p1x 0.3 def /p1y  0.3 def
/p2x 0.7 def /p2y  0.7 def
/p3x 0.9 def /p3y  0.9 def
/e +0.03 def               } if
Typ 7 eq { % almost linear
/p0x 0.1 def /p0y  0.1 def
/p1x 0.3 def /p1y  0.3 +1e-4 add def
/p2x 0.7 def /p2y  0.7 -1e-4 add def
/p3x 0.9 def /p3y  0.9 def
/e +0.03 def               } if
Typ 8 eq { % circle
/p0x 1.0 def   /p0y  0.0   def
/p1x 1.0 def   /p1y  0.552 def
/p2x 0.552 def /p2y  1.0 def
/p3x 0.0 def   /p3y  1.0   def
/e +0.03 def               } if
Typ 9 eq { % ellipse
/p0x 1.0 def   /p0y  0.0   0.5 mul def
/p1x 1.0 def   /p1y  0.552 0.5 mul def
/p2x 0.552 def /p2y  1.0   0.5 mul def
/p3x 0.0 def   /p3y  1.0   0.5 mul def
/e +0.03 def               } if
```

7.13 Bézier Offset Curves / PostScript Code

```
Typ 10 eq {
/p0x 0.1 def /p0y  0.1 def
/p1x 0.1 def /p1y  0.9 def
/p2x 0.5 def /p2y  0.9 def
/p3x 0.9 def /p3y  0.8 def
/e +0.03 def              } if
Typ 11 eq { % symmetrical parabola
/p0x 0 def /p0y  0 def
/p1x 0 def /p1y  1 def
/p2x 1 def /p2y  1 def
/p3x 1 def /p3y  0 def
/e -0.03 def              } if

Typ 12 eq { % near singularity
/p0x 0.3 def /p0y  0.3 def
/p1x 0.5 def /p1y  1.1 def % 0.596425
/p2x 1.0 def /p2y  0.9 def
/p3x 1.4 def /p3y  0.3 def
/e  0.03 def              } if

Typ 13 eq { % singularity
/p0x 0.3 def /p0y  0.3 def
/p1x 0.596425 def /p1y  1.1 def % 0.596425
/p2x 1.0 def /p2y  0.9 def
/p3x 1.4 def /p3y  0.3 def
/e  0.03 def              } if

/mm {2.834646 mul} def

/sx  100 mm def   % Length 0..sx
/sy  100 mm def
/bx  510 def % extended BBox 560 560
/by  340 def

/x0  10 mm def   % Offset
/y0  10 mm def

/Bbox
{ 0.4 mm setlinewidth
  0 setgray
  newpath
  0 0 moveto bx 0 rlineto 0 by rlineto bx neg 0 rlineto closepath stroke
} def

/Vbox
{ 0.2 mm sx div setlinewidth
  0 setgray
  newpath
  0 0 moveto 1 0 rlineto 0 1 rlineto -1 0 rlineto closepath stroke
} def

/Tang
{/yd1 exch def
 /xd1 exch def
 /yd0 exch def
 /xd0 exch def
  currentlinewidth 0.5 mul setlinewidth
 /d1 0.01 def
 /d2 d1 0.5 mul def
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
  newpath
  xd1 d2 sub yd1 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
  newpath
  xd0 yd0 moveto xd1 yd1 lineto
  stroke
  currentlinewidth 2 mul setlinewidth
} def
```

7.14 Bézier Offset Curves / PostScript Code

```
/Dot
{/yd0 exch def
 /xd0 exch def
 /d1 0.01 def
 /d2 d1 0.5 mul def
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
 } def

/Shownum
% Draw number by string
% Version May 15 2004 / uses rounding
% Full code in [5]
{} def

% Bbox

x0 y0 translate
sx sy scale

Vbox

0.3 mm sx div setlinewidth

/f83 8 3 div def
/f34 3 4 div def
/f43 4 3 div def
/f18 1 8 div def
/f14 1 4 div def
/f12 1 2 div def

/s0x p1x p0x sub def
/s0y p1y p0y sub def
/s3x p3x p2x sub def
/s3y p3y p2y sub def

/ax s0x  3 mul s3x 3 mul add p3x p0x sub 2 mul sub def
/ay s0y  3 mul s3y 3 mul add p3y p0y sub 2 mul sub def
/bx s0x -6 mul s3x 3 mul sub p3x p0x sub 3 mul add def
/by s0y -6 mul s3y 3 mul sub p3y p0y sub 3 mul add def
/cx s0x  3 mul def
/cy s0y  3 mul def

/dn  s0x dup mul s0y dup mul add sqrt def % dup mul = square
/n0x s0y dn div neg def
/n0y s0x dn div def

/dn  s3x dup mul s3y dup mul add sqrt def
/n3x s3y dn div neg def
/n3y s3x dn div     def

/pcx ax f18 mul bx f14 mul add cx f12 mul add p0x add def
/pcy ay f18 mul by f14 mul add cy f12 mul add p0y add def

 1 0 0 setrgbcolor
 pcx pcy Dot

/q0x p0x e n0x mul add def
/q0y p0y e n0y mul add def

/q3x p3x e n3x mul add def
/q3y p3y e n3y mul add def

/Ax s0x  3 mul s3x  3 mul add q3x q0x sub -2 mul add def
/Ay s0y  3 mul s3y  3 mul add q3y q0y sub -2 mul add def
/Bx s0x -6 mul s3x -3 mul add q3x q0x sub  3 mul add def
/By s0y -6 mul s3y -3 mul add q3y q0y sub  3 mul add def
/Cx s0x  3 mul def
/Cy s0y  3 mul def

/qcx Ax f18 mul Bx f14 mul add Cx f12 mul add q0x add def
/qcy Ay f18 mul By f14 mul add Cy f12 mul add q0y add def

/dpx ax f34 mul bx add cx add def
/dpy ay f34 mul by add cy add def
```

7.15 Bézier Offset Curves / PostScript Code

```
/dqx Ax f34 mul Bx add Cx add def
/dqy Ay f34 mul By add Cy add def

/dn  dpx dup mul dpy dup mul add sqrt def
/ncx dpy dn div neg def
/ncy dpx dn div     def

/rcx pcx ncx e mul add def
/rcy pcy ncy e mul add def

1 0 1 setrgbcolor
rcx rcy Dot

0.8 0.3 0.3 setrgbcolor
newpath
p0x p0y moveto
p1x p1y p2x p2y p3x p3y curveto
stroke
p0x p0y p1x p1y Tang
p2x p2y p3x p3y Tang

/k0 1.0 def /dk0 0.0 def
/k3 1.0 def /dk3 0.0 def
/t  0.5 def /dt  0.0 def

/q1x q0x k0 s0x mul add def
/q1y q0y k0 s0y mul add def

/q2x q3x k3 s3x mul sub def
/q2y q3y k3 s3y mul sub def

0 0.8 0 setrgbcolor
newpath
q0x q0y moveto
q1x q1y q2x q2y q3x q3y curveto
stroke

q0x q0y q1x q1y Tang
q2x q2y q3x q3y Tang
qcx qcy Dot

/a11 s0x def  /a12 s3x neg def  /a13 dpx f83 mul def
/a21 s0y def  /a22 s3y neg def  /a23 dpy f83 mul def

/a31 s0x ncx mul s0y ncy mul add def
/a32 s3x ncx mul s3y ncy mul add def
/a33 s0x s3x sub ncx mul s0y s3y sub ncy mul add 4 mul def

/y1 rcx qcx sub f83 mul def
/y2 rcy qcy sub f83 mul def
/y3 dqx ncx mul dqy ncy mul add f43 mul def

/A11 a22 a33 mul a23 a32 mul sub def
/A21 a12 a33 mul a13 a32 mul sub neg def
/A31 a12 a23 mul a13 a22 mul sub def
/A12 a21 a33 mul a23 a31 mul sub neg def
/A22 a11 a33 mul a13 a31 mul sub def
/A32 a11 a23 mul a13 a21 mul sub neg def
/A13 a21 a32 mul a22 a31 mul sub def
/A23 a11 a32 mul a12 a31 mul sub neg def
/A33 a11 a22 mul a12 a21 mul sub def
```

7.15 Bézier Offset Curves / PostScript Code

```
/Sys3
{
/De0 a11 A11 mul a21 A21 mul add a31 A31 mul add def
/De1 y1  A11 mul y2  A21 mul add y3  A31 mul add def
/De2 y1  A12 mul y2  A22 mul add y3  A32 mul add def
/De3 y1  A13 mul y2  A23 mul add y3  A33 mul add def
/aDe0 De0 abs def
aDe0 1 ge
{/flg 3 def }
{/flg 0 def
 /max1 0.90 def
 /max2 max1 0.5 mul def
% any absolute division result should be less than max
  De1 abs aDe0 max1 mul lt {/flg flg 1 add def} if
  De2 abs aDe0 max1 mul lt {/flg flg 1 add def} if
  De3 abs aDe0 max2 mul lt {/flg flg 1 add def} if } ifelse
% /flg 3 def % for test ONLY
  flg 3 eq % solutions for the three unknowns exist
{/dk0 De1 De0 div def
 /dk3 De2 De0 div def
 /dt  De3 De0 div def } if
} def

/Sys2
{
/a13 0 def /a23 0 def /a31 0 def /a32 0 def /a33 1 def /y3 0 def
/De0 a11 a22 mul a12 a21 mul sub def
/De1 y1  a22 mul y2  a12 mul sub def
/De2 a11  y2 mul y1  a21 mul sub def
/De3 0 def
/aDe0 De0 abs def
aDe0 1 ge
{/flg 2 def }
{/flg 0 def
 /max1 0.90 def
 /max2 max1 0.5 mul def
% any absolute division result should be less than max
  De1 abs aDe0 max1 mul lt {/flg flg 1 add def} if
  De2 abs aDe0 max1 mul lt {/flg flg 1 add def} if } ifelse
  flg 2 eq % solutions for the two unknowns exist
{/dk0 De1 De0 div def
 /dk3 De2 De0 div def } if
} def

Sys3

flg 3 lt
{ Sys2 } if

% Residuals should be zero, valid if flg=3
/res1 a11 dk0 mul a12 dk3 mul add a13 dt mul add y1 sub def
/res2 a21 dk0 mul a22 dk3 mul add a23 dt mul add y2 sub def
/res3 a31 dk0 mul a32 dk3 mul add a33 dt mul add y3 sub def

/k0 k0 dk0 add def
/k3 k3 dk3 add def
/t  t  dt  add def

/r0x q0x def
/r0y q0y def
/r1x q0x k0 s0x mul add def
/r1y q0y k0 s0y mul add def
/r2x q3x k3 s3x mul sub def
/r2y q3y k3 s3y mul sub def
/r3x q3x def
/r3y q3y def

0.3 0.3 1 setrgbcolor
newpath
r0x r0y moveto
r1x r1y r2x r2y r3x r3y curveto
stroke
r0x r0y r1x r1y Tang
r2x r2y r3x r3y Tang
```

7.16 Bézier Offset Curves / PostScript Code

```
% for MidSlope and TestList
/drx dqx s0x dk0 mul s3x dk3 mul add f34 mul sub s3x s0x sub dt mul 3 mul add def
/dry dqy s0y dk0 mul s3y dk3 mul add f34 mul sub s3y s0y sub dt mul 3 mul add def

/MidSlope
{
% Draw midpoint tangents
[0.02 0.007] 0 setdash
0.3 0.3 1 setrgbcolor
rcx rcy rcx drx add rcy dry add Tang
0.8 0.3 0.3 setrgbcolor
pcx pcy pcx dpx add pcy dpy add Tang
0 0.8 0 setrgbcolor
qcx qcy qcx dqx add qcy dqy add Tang
[] 0 setdash
} def

DrawMid {MidSlope} if

0 setgray
/fh 9.75 sx div def
/Helvetica findfont fh scalefont setfont

/tms 6 def
/txa 1.05 def
/txb 1.20 def
/txc 1.45 def
/txd 1.70 def

/LF
{/tya tya fh sub def
} def

/StanList
{/tya 0.97 def
txa tya moveto (P0) show txb tya p0x Shownum txc tya p0y Shownum LF
txa tya moveto (P1) show txb tya p1x Shownum txc tya p1y Shownum LF
txa tya moveto (P2) show txb tya p2x Shownum txc tya p2y Shownum LF
txa tya moveto (P3) show txb tya p3x Shownum txc tya p3y Shownum LF
txa tya moveto (Pc) show txb tya pcx Shownum txc tya pcy Shownum LF
txa tya moveto (e)  show txb tya e   Shownum                     LF
LF
txa tya moveto (Q0) show txb tya q0x Shownum txc tya q0y Shownum LF
txa tya moveto (Q1) show txb tya q1x Shownum txc tya q1y Shownum LF
txa tya moveto (Q2) show txb tya q2x Shownum txc tya q2y Shownum LF
txa tya moveto (Q3) show txb tya q3x Shownum txc tya q3y Shownum LF
txa tya moveto (Qc) show txb tya qcx Shownum txc tya qcy Shownum LF
LF
txa tya moveto (R0) show txb tya r0x Shownum txc tya r0y Shownum LF
txa tya moveto (R1) show txb tya r1x Shownum txc tya r1y Shownum LF
txa tya moveto (R2) show txb tya r2x Shownum txc tya r2y Shownum LF
txa tya moveto (R3) show txb tya r3x Shownum txc tya r3y Shownum LF
txa tya moveto (Rc) show txb tya rcx Shownum txc tya rcy Shownum LF
/tms 12 def
LF
txa tya moveto (D0) show txb tya De0  Shownum LF
txa tya moveto (D1) show txb tya De1  Shownum LF
txa tya moveto (D2) show txb tya De2  Shownum LF
txa tya moveto (D3) show txb tya De3  Shownum LF
LF
txa tya moveto (dk0) show txb tya dk0 Shownum LF
txa tya moveto (dk3) show txb tya dk3 Shownum LF
txa tya moveto (dt ) show txb tya dt  Shownum LF
/tms 0 def
LF
txa tya moveto (Used equations ) show 1.35 tya flg Shownum LF
} def
```

7.17 Bézier Offset Curves / PostScript Code

```
/TestList
{
/tya 1.8 def
/tms 6 def
txa tya moveto (aik) show
txb tya a11  Shownum  txc tya a12 Shownum txd tya a13  Shownum LF
txb tya a21  Shownum  txc tya a22 Shownum txd tya a23  Shownum LF
txb tya a31  Shownum  txc tya a32 Shownum txd tya a33  Shownum LF
LF
/tms 12 def
txa tya moveto (y1  ) show txb tya y1   Shownum LF
txa tya moveto (y2  ) show txb tya y2   Shownum LF
txa tya moveto (y3  ) show txb tya y3   Shownum LF
LF
/tms 12 def
txa tya moveto (res1) show txb tya res1 Shownum LF
txa tya moveto (res2) show txb tya res2 Shownum LF
txa tya moveto (res3) show txb tya res3 Shownum LF
LF

/tms 12 def
txa tya moveto (dpx ) show txb tya dpx Shownum LF
txa tya moveto (dpy ) show txb tya dpy Shownum LF
txa tya moveto (dqx ) show txb tya dqx Shownum LF
txa tya moveto (dqy ) show txb tya dqy Shownum LF
txa tya moveto (drx ) show txb tya drx Shownum LF
txa tya moveto (dry ) show txb tya dry Shownum LF
} def

StanList
TestList % use extended bounding box

showpage
```

8.1 Bézier Point Distance / Concepts

The distance between a point **Q** and a Bézier curve **P** cannot be found analytically.
Solving the problem by optimization requires the real roots of a fifth order polynomial and special treatment of endpoints.
Alternatively, the curve can be flattened (converted into line segments) which leads to the standard task 'distance between point and line segment ', again with endpoint investigations.

Recursion

The problem is solved by a straight search of the minimal squared Euclidian distance and recursive subdivision.

Step 1: the curve contains points 0 to 32 for t=0 to t=1 with dt=1/32.
The search starts at 0 and ends at 32.
t_{min} is the parameter for the minimal distance.

Step 2: dt = 2dt/16
Search from (t_{min}-1/32) to (t_{min}+1/32). t_{tmin} is replaced.
Values t are clipped for [0,1], thus no search happens beyond the end points.

Step 3: dt = 2dt/8

Step 4: dt = 2dt/4

Step 5: dt = 2dt/2

If two sequential parameters t_{min} differ by less than $\varepsilon=10^{-3}$ then the recursion stops earlier.
Endpoints are included.
The points on the curve are calculated by *Horner*, which requires basically six multiplications per step.

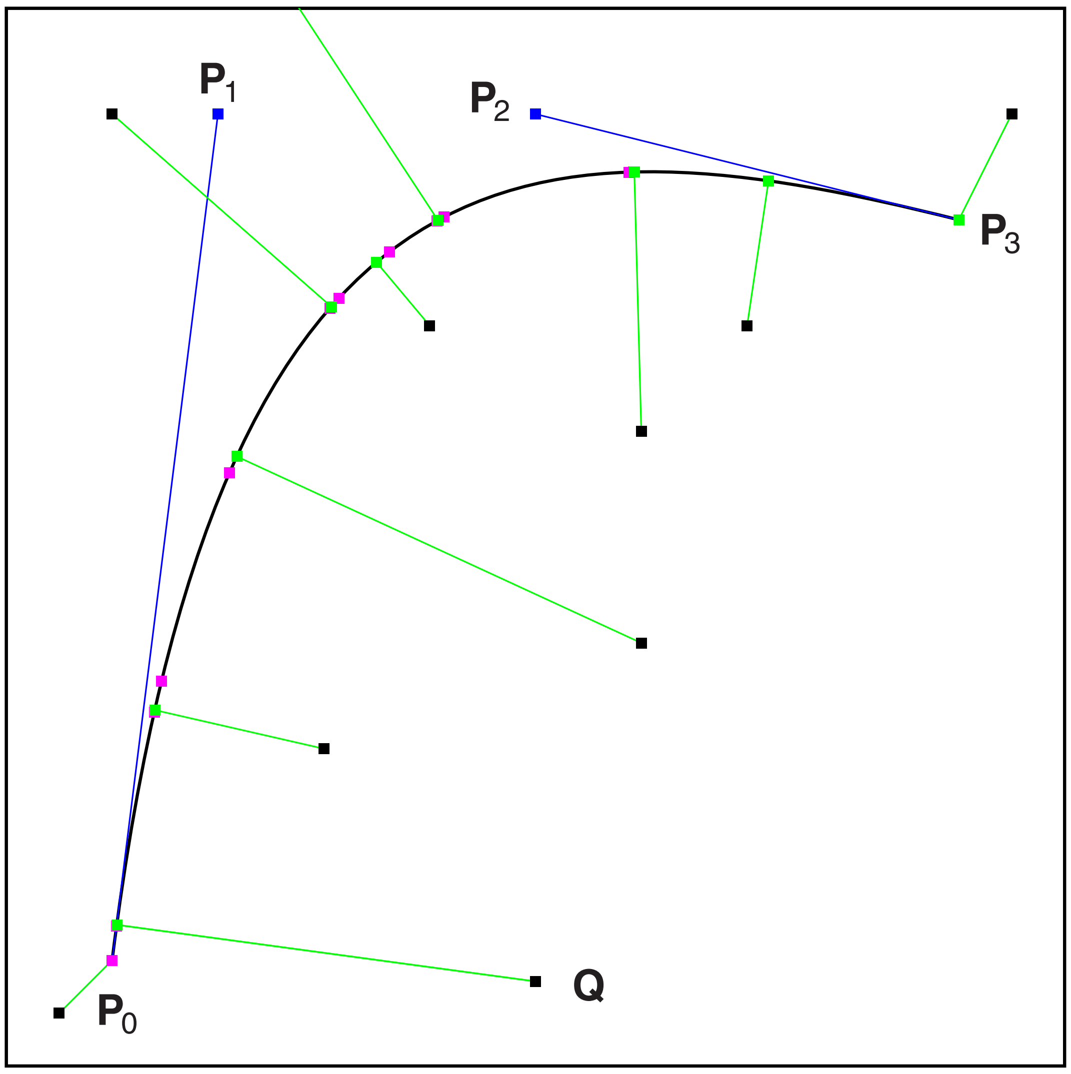
Straight Search

The *Horner* calculation is replaced by *Forward Differences*, as explained in chapter 5.
Besides the initialization one needs only additions, therefore the calculation per point should be much faster.
In the examples we have 101 points for dt=1/100. The number of points can be enlarged, depending on the required accuracy.
The squared Euclidian distances are compared as above, without any further subdivision.

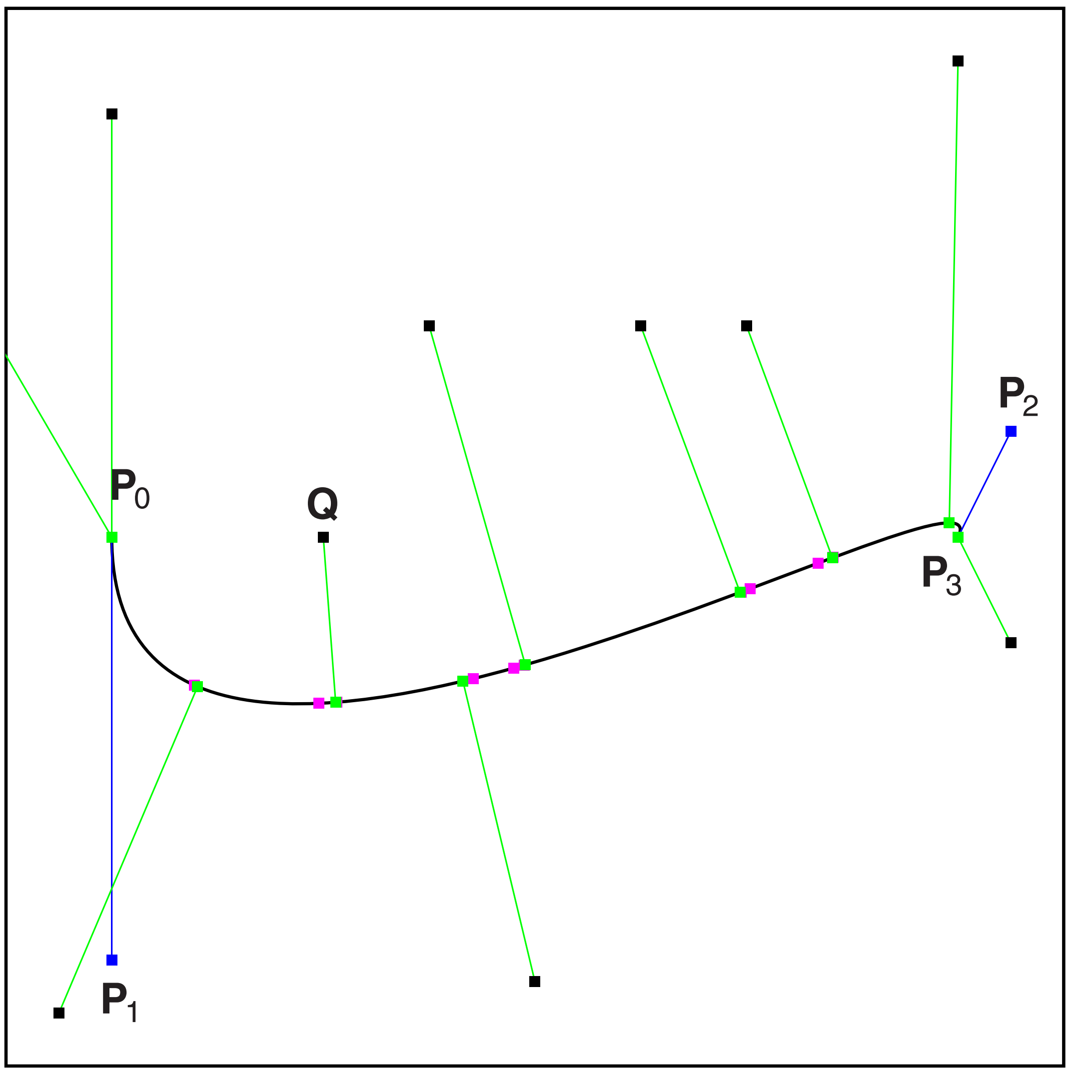
In either case the method is restricted to somewhat normalized Bézier curves for practical purposes.

8.2 Bézier Point Distance / Examples Recursion

Several test points are marked by a black square. Search points are shown magenta and final results green.



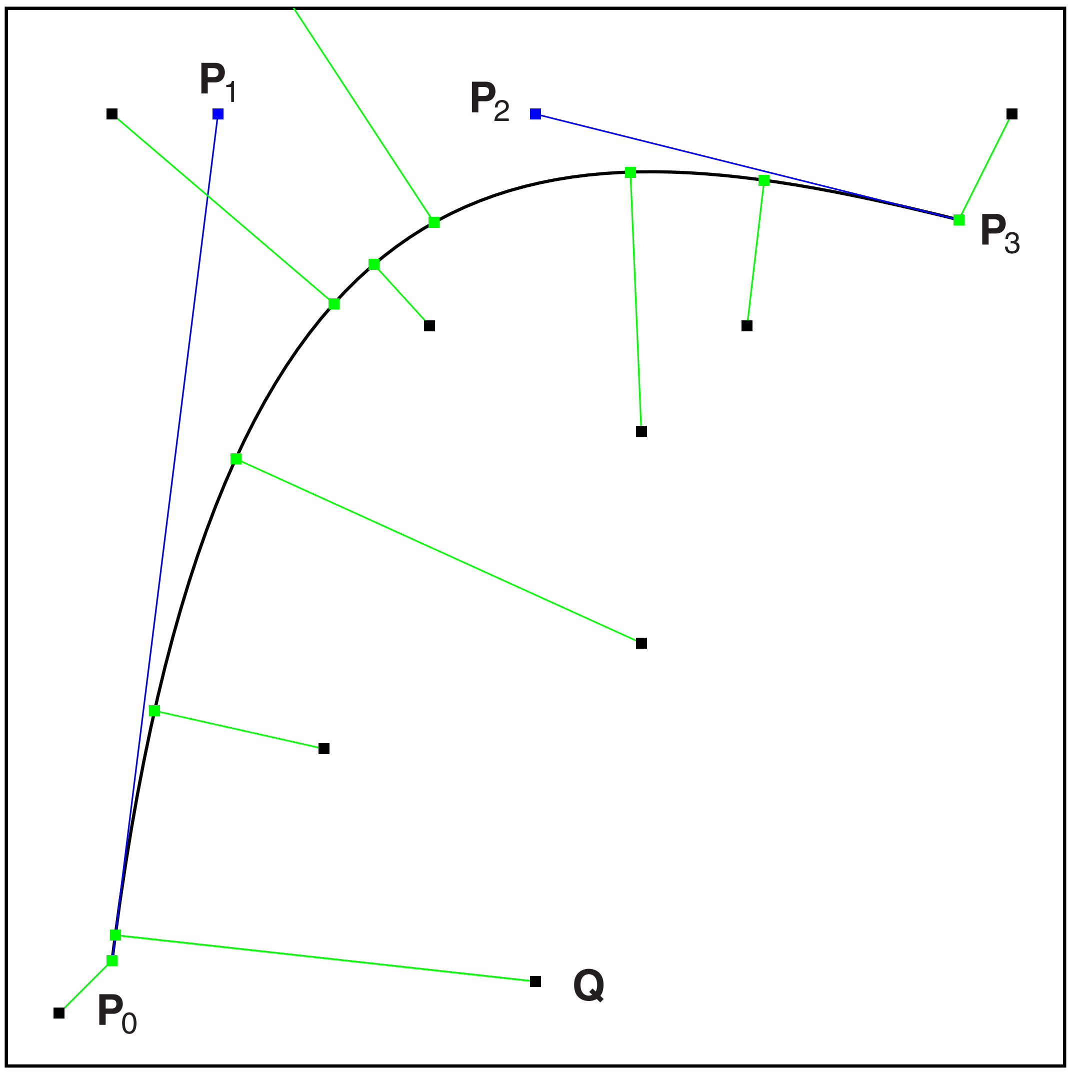
P0	0.100000	0.100000
P1	0.200000	0.900000
P2	0.500000	0.900000
P3	0.900000	0.800000
Q0	0.500000	0.080000
Magenta		Iteration
Green		Result
Delta-t		0.000488
Loops		3
Points		59



P0	0.100000	0.100000
P1	0.200000	0.900000
P2	0.500000	0.900000
P3	0.900000	0.800000
Q0	0.300000	0.500000
Magenta		Iteration
Green		Result
Delta-t		0.000977
Loops		3
Points		59

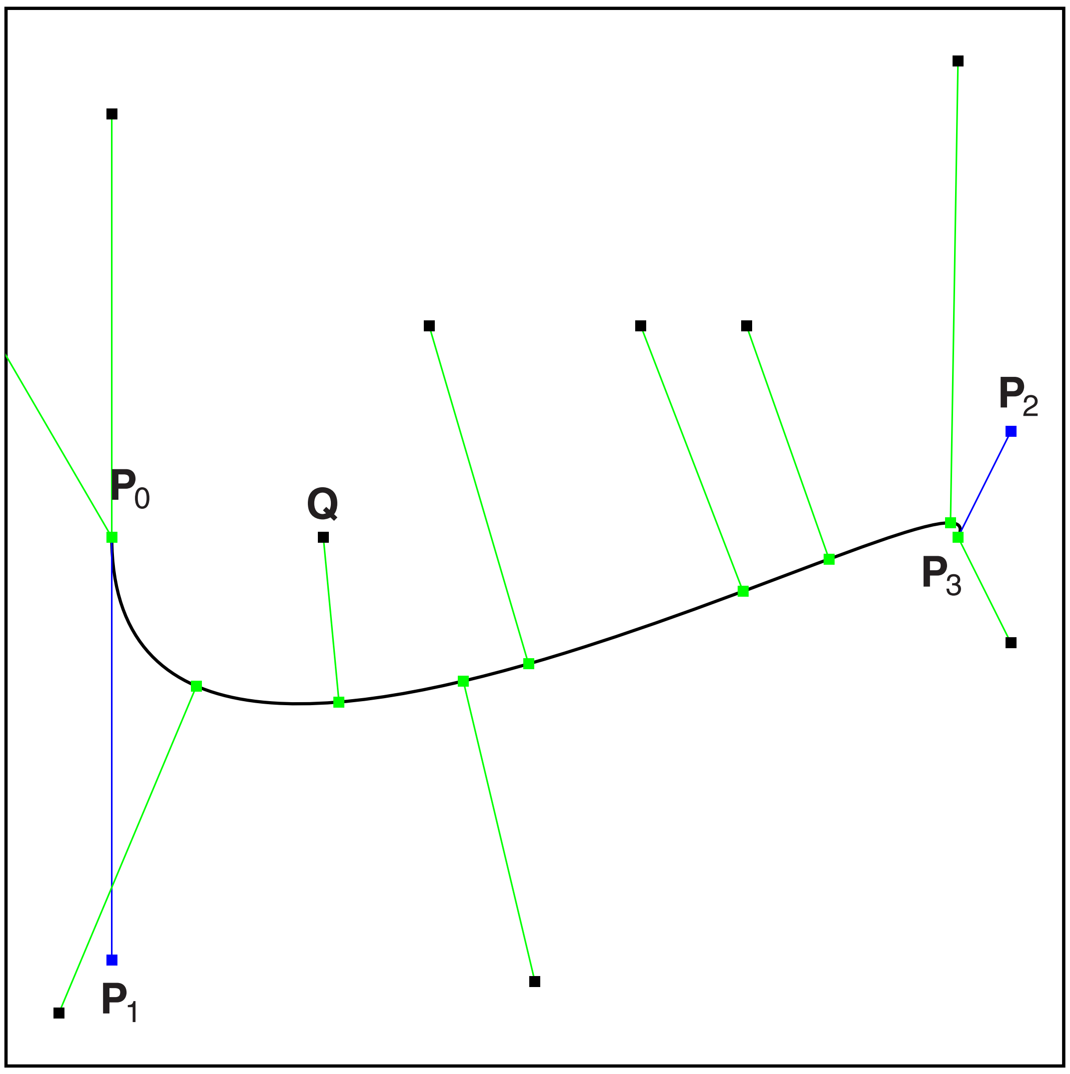
8.3 Bézier Point Distance / Examples Straight Search

Several test points are marked by a black square. Final results are shown green.



P0	0.100000	0.100000
P1	0.200000	0.900000
P2	0.500000	0.900000
P3	0.900000	0.800000
Q0	0.500000	0.080000

Points 101



P0	0.100000	0.100000
P1	0.200000	0.900000
P2	0.500000	0.900000
P3	0.900000	0.800000
Q0	0.300000	0.500000

Points 101

8.4 Bézier Point Distance / PS Code

```
%!PS-Adobe-3.0   EPSF-3.0
%%BoundingBox:   0 0 500 286
%%Creator:       Gernot Hoffmann
%%Title:         BezDist02-R
%%CreationDate:  July 30 / 2005

% Disable setpagedevice
 /setpagedevice {pop} bind def

% Bezier Curve. Min.distance of point

% PForw true : straight search by forward differences
%       false: recursive search, Horner

/PForw false def

/mm {2.834646 mul} def

/sx  100 mm def  % Length 1
/bx  500 def % Bounding box
/by  286 def

/x0  0.2 mm def  % Offset
/y0  0.2 mm def

/ax   0 def /bx   0 def /cx 0 def
/ay   0 def /by   0 def /cy 0 def
/px   0 def /py   0 def
/d1   0 def /d2   0 def
/xd0  0 def /yd0  0 def /xd1 0 def /yd1  0 def
/p0x  0 def /p0y  0 def /p1x 0 def /p1y  0 def
/p2x  0 def /p2y  0 def /p3x 0 def /p3y  0 def
/q0x  0 def /q0y  0 def
/eps  0 def /stp  0 def /di2 0 def /dmin 0 def
/t1   0 def /t2   0 def /tk  0 def /dt   0 def
/Delt 0 def
/tmin 0 def /told 0 def /Lp  0 def /Lpm 0 def
/dt1  0 def /dt2  0 def /dt3 0 def
/dx1  0 def /dx2  0 def /dx3 0 def
/dy1  0 def /dy2  0 def /dy3 0 def

/Bbox
{ 0.4 mm setlinewidth
  0 setgray
  newpath
  0 0 moveto bx 0 rlineto 0 by rlineto bx neg 0 rlineto closepath stroke
} def

/Vbox
{ 0.3 mm sx div setlinewidth
  0 setgray
  newpath 0 0 moveto 1 0 rlineto 0 1 rlineto -1 0 rlineto closepath stroke
  newpath 0 0 moveto 1 0 rlineto 0 1 rlineto -1 0 rlineto closepath clip
} def

/Tang
{/yd1 exch def
 /xd1 exch def
 /yd0 exch def
 /xd0 exch def
  currentlinewidth 0.5 mul setlinewidth
 /d1 0.01 def
 /d2 d1 0.5 mul def
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
  newpath
  xd1 d2 sub yd1 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
  newpath
  xd0 yd0 moveto xd1 yd1 lineto
  stroke
  currentlinewidth 2 mul setlinewidth
} bind def
```

8.5 Bézier Point Distance / PS Code

```
/Dot
{/yd0 exch def
 /xd0 exch def
 /d1 0.01 def
 /d2 d1 0.5 mul def
  newpath
  xd0 d2 sub yd0 d2 sub moveto d1 0 rlineto 0 d1 rlineto d1 neg 0 rlineto
  closepath fill
 } bind def

% /Shownum  %  as in previous chapter

/BezPointH
% Point by Horner
{/tp exch def
 /px ax tp mul bx add tp mul cx add tp mul p0x add def
 /py ay tp mul by add tp mul cy add tp mul p0y add def
 /dis px q0x sub dup mul py q0y sub dup mul add def    % squared distance
} def

/BezForw
% for forward differences only
{/dt1 dt def
 /dt2 dt1 dup mul def
 /dt3 dt2 dt1 mul def
 /dx1 ax dt3 mul bx dt2 mul add cx dt1 mul add def
 /dy1 ay dt3 mul by dt2 mul add cy dt1 mul add def
 /dx2 ax dt3 mul 6 mul bx dt2 mul 2 mul add def
 /dy2 ay dt3 mul 6 mul by dt2 mul 2 mul add def
 /dx3 ax dt3 mul 6 mul def
 /dy3 ay dt3 mul 6 mul def
} bind def

/BezPointF
% Point by forward differences
{/dis px q0x sub dup mul py q0y sub dup mul add def
 /px px dx1 add def /dx1 dx1 dx2 add def /dx2 dx2 dx3 add def
 /py py dy1 add def /dy1 dy1 dy2 add def /dy2 dy2 dy3 add def
} bind def

/BezDistR
{% Recursive subdivision
/eps  1e-3 def % limit for convergence of t
/dmin 1e+6 def
/tmin 0.5 def
/told 2 def
/t1   0 def
/t2   1 def
/Lpm  5 def   % max loops
/stp 32 def   % max points
/dt 1 stp div def
/Fp 0 def
1 1 Lpm
 {/Lp exch def
  /tk t1 def
     0 1 stp
   { pop
     tk BezPointH /Fp Fp 1 add def
     dis dmin lt {/dmin dis def /tmin tk def} if
    /tk tk dt add def
   } for
  tmin BezPointH 1 0 1 setrgbcolor px py Dot
 /Delt told tmin sub abs def
  Delt eps lt {exit} if % exit for
 /told tmin def
 /t1 tmin dt sub def t1 0 lt {/t1 0 def} if
 /t2 tmin dt add def t2 1 gt {/t2 1 def} if
 /stp stp 2 idiv def
 /dt t2 t1 sub stp div def
 } for
0 1 0 setrgbcolor px py q0x q0y Tang 0 0 0 setrgbcolor q0x q0y Dot
} bind def
```

8.6 Bézier Point Distance / PS Code

```
/BezDistF
{% Straight search by forward differences
/stp 100 def   % max points
/dt 1 stp div def
/Fp stp 1 add def
/Lp 1 def
/dmin 1e6 def
/px p0x def
/py p0y def
 BezForw
 0 1 stp
 {/k exch def
   BezPointF
   dis dmin lt {/dmin dis def /kmin k def} if
 } for
 /tmin kmin dt mul def
  tmin BezPointH
0 1 0 setrgbcolor px py q0x q0y Tang 0 0 0 setrgbcolor q0x q0y Dot
} bind def

/BezDist
{
 PForw {BezDistF}{BezDistR} ifelse
} def

% — Begin

% Bbox

x0 y0 translate
sx sx scale

gsave

Vbox

/p0x 0.1 def /p0y 0.5 def
/p1x 0.1 def /p1y 0.1 def
/p2x 0.95 def /p2y 0.6 def
/p3x 0.9 def /p3y 0.5 def

BezInit

0.3 mm sx div setlinewidth

/q0x 0.95 def /q0y 0.40 def BezDist
/q0x 0.70 def /q0y 0.70 def BezDist
/q0x 0.10 def /q0y 0.90 def BezDist
/q0x 0.90 def /q0y 0.95 def BezDist
/q0x 0.60 def /q0y 0.70 def BezDist
/q0x 0.40 def /q0y 0.70 def BezDist
/q0x 0.50 def /q0y 0.08 def BezDist
/q0x 0.05 def /q0y 0.05 def BezDist
/q0x  -10 def /q0y   20 def BezDist % larger values cause PS clipping errors
/q0x 0.30 def /q0y 0.50 def BezDist

/p0x 0.1 def /p0y 0.1 def
/p1x 0.2 def /p1y 0.9 def
/p2x 0.5 def /p2y 0.9 def
/p3x 0.9 def /p3y 0.8 def

grestore

 0 setgray
/fh 10.5 sx div def
/Helvetica findfont fh scalefont setfont

/tms 6 def
/txa 1.1 def
/txb txa 0.20 add def
/txc txb 0.20 add def
/txd txb 0.18 add def
```

8.7 Bézier Point Distance / PS Code

```
/LF
{/tya tya fh sub def
} def

/StanList
{/tya 0.34 def
txa tya moveto (P0) show txb tya p0x Shownum txc tya p0y Shownum LF
txa tya moveto (P1) show txb tya p1x Shownum txc tya p1y Shownum LF
txa tya moveto (P2) show txb tya p2x Shownum txc tya p2y Shownum LF
txa tya moveto (P3) show txb tya p3x Shownum txc tya p3y Shownum LF
txa tya moveto (Q0) show txb tya q0x Shownum txc tya q0y Shownum LF
txa tya moveto (Magenta) show txd tya moveto (Iteration) show LF
txa tya moveto (Green)   show txd tya moveto (Result)    show LF
txa tya moveto (Delta-t) show txc tya Delt Shownum LF
/tms 0 def
txa tya moveto (Loops)   show txc tya Lp Shownum LF
txa tya moveto (Points)  show txc tya Fp Shownum LF
} def

StanList

showpage
```

9. References

[1] PostScript Language Reference (PS3)
Addison-Wesley, Boston, San Francisco ...
2002

[2] H.McGilton + M.Campione
PostScript by Example
Addison-Wesley, Reading, Massachusetts ...
1998

[3] J.D.Foley et al.
Computer Graphics
Addison-Wesley, Reading, Massachusetts ...
1990

[4] M.Shemanarev
http://www.antigrain.com/agg_research/bezier_interpolation.html

[5] G.Hoffmann
PostScript file for Bézier Offset Curves
Rename txt by eps
http://www.fho-emden.de/~hoffmann/bezoffs11.txt

[6] A.Watt + M.Watt
Advanced Animation and Rendering Techniques
Addison-Wesley, Reading, Massachusetts ...
1994

This document
http://www.fho-emden.de/~hoffmann/bezier18122002.pdf

Gernot Hoffmann
November 27 - 2 / 2006
Website
Load browser / Click here

Appendix 1 (Bezier Offset Curves)

The analytical evaluation of the system determinant should help to identify singular cases by geometrical interpretations. So far this attempt was not successful.

Evalution of $\det(\mathbf{A})$:

$$\mathbf{A} = \begin{bmatrix} s_{0x} & -s_{3x} & \frac{8}{3}dp_x \\ s_{0y} & -s_{3y} & \frac{8}{3}dp_y \\ \mathbf{s}_0^T\mathbf{n}_c & \mathbf{s}_3^T\mathbf{n}_c & 4(\mathbf{s}_0 - \mathbf{s}_3)^T\mathbf{n}_c \end{bmatrix}$$

$$\begin{aligned} \det(\mathbf{A}) = & +\mathbf{s}_0^T\mathbf{n}_c(-s_{3x}\frac{8}{3}dp_y + s_{3y}\frac{8}{3}dp_x) \\ & -\mathbf{s}_3^T\mathbf{n}_c(+s_{0x}\frac{8}{3}dp_y - s_{0y}\frac{8}{3}dp_x) \\ & +4(\mathbf{s}_0 - \mathbf{s}_3)^T\mathbf{n}_c(-s_{0x}s_{3y} + s_{3x}s_{0y}) \end{aligned}$$

dp_x and dp_y can be replaced by

$$\begin{bmatrix} -dp_y \\ dp_x \end{bmatrix} = \mathbf{n}_c\sqrt{dp_x^2 + dp_y^2}\,.$$

The last bracket above can be written as cross product (actually the z-component of the cross product):

$$-s_{0x}s_{3y} + s_{3x}s_{0y} = \mathbf{s}_3 \times \mathbf{s}_0$$

$$\begin{aligned} \det(\mathbf{A}) = & \frac{16}{3}\sqrt{dp_x^2 + dp_y^2}\,(\mathbf{s}_0^T\mathbf{n}_c)(\mathbf{s}_3^T\mathbf{n}_c) \\ & +4(\mathbf{s}_3 \times \mathbf{s}_0)((\mathbf{s}_0 - \mathbf{s}_3)^T\mathbf{n}_c) \end{aligned}$$