21 Bildschirmgrafik

Gliederung

21.1 Allgemeines zur Bildschirmgrafik

Für die komfortable Erstellung von Bildschirmgrafiken steht in Turbo-Pascal die Unit »GRAPH.TPU« zur Verfügung. Sie enthält eine Vielzahl von vordefinierten Grafikprozeduren, -funktionen und -konstanten.

Wichtig: Im Menüpunkt *Option/Verzeichnisse* muß im Feld *Unit-Verzeichnis* ein Zugriffspfad zur Unit »GRAPH.TPU« eingetragen sein. Bei Standard-Installation auf einer Festplatte: »C:\BP\UNITS«.

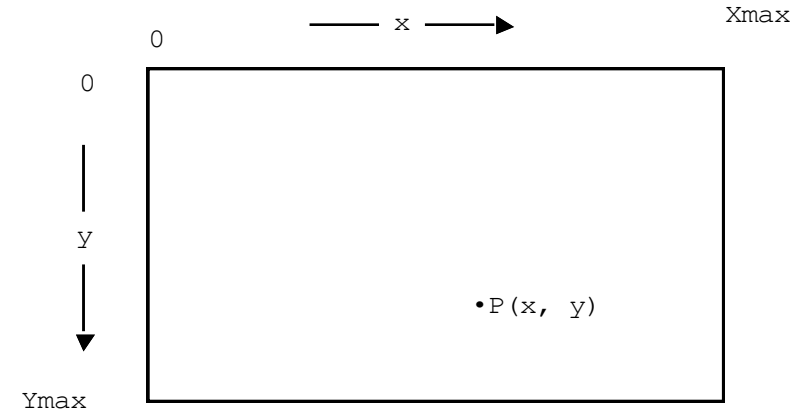
Außer dieser Unit benötigt man noch die Grafiktreiber für die Bildschirmkarte, z.B. "EGAVGA.BGI" für VGA-Grafikkarten mit einer Aufläsung von 640 * 480 und bei Verwendung von Vektorschriften die entsprechenden Schriftdateien, z.B. "Goth.CHR" oder "Trip.CHR".

Die Grafiktreiber und auch die Schriften stehen bei Standardinstallation im Verzeichnis "C:\BP\BGI". Der Zugriffspfad zum Grafiktreiber muß im Programm genannt werden.

»BGI« bedeutet ***B**orland **G**raphic **I**nterface*.

Es können nur Strings auf den Grafikbildschirm geschrieben werden. Numerische Daten sind deshalb vorher in Strings zu konvertieren.

Die Koordinaten des Grafik-Bildschirms:



Bei einer Grafikkarte von z.B. 640 * 480 laufen die x-Koordinaten von 0..639 und die y-Koordinaten von 0..479.

Zu Beginn eines Grafikprogramms muß die Grafik entsprechend der verwendeten Grafikkarte initialisiert werden. Dabei ist besonders darauf zu achten, daß der Zugriffspfad zum zutreffenden Grafiktreiber richtig gesetzt ist.

Das folgende kleine Programm zeigt die Initialisierung der Grafik, die Verwendung der Grafikprozedur *Circle* und auch das Beenden der Grafik.

```
program Pas21011; { Grafik }

uses
  CRT, GRAPH;

var
  Grafiktreiber,
  Grafikmodus:   Integer;

begin
  Grafiktreiber := Detect;
     { Mit "Detect" automatische Erkennung des Grafiktreibers, }
     { nicht bei allen Grafikkarten, wohl aber z.B. bei VGA    }

  InitGraph(Grafiktreiber, Grafikmodus, 'C:\BP\BGI');
     { Der Zugriffspfad ist ggf. anzupassen                    }
     { Wenn "Grafikmodus" nicht vorher spezifiziert, dann wird }
     { automatisch die höchste Auflösungsstufe eingestellt     }

  Circle(200, 100, 80);    { Kreis: xM = 200, yM = 100, r = 80 }

  repeat
  until ReadKey <> '';
  CloseGraph;   { Zurück zum Textmodus }
end.
```

21.2 Die Grafikprozedur PutPixel

Die Prozedur »PutPixel« ist die wichtigste aller Grafikprozeduren und dient zum Setzen eines Pixels (picture element) an der Position x/y. Durch Überschreiben mit der Hintergrundfarbe kann das Pixel auch wieder "gelöscht" werden.

Format: `PutPixel(`*x*`, `*y*`, `*farbe*`)`

x	x-Koordinate. Integer-Ausdruck
y	y-Koordinate. Integer-Ausdruck
farbe	Pixelfarbe. Word-Ausdruck (Werte 0 bis 15) oder vordefinierte Konstanten nach folgender Auflistung:

Die vordefinierten Farbkonstanten:

0 Black	4 Red	8 Darkgray	12 Lightred
1 Blue	5 Magenta	9 Lightblue	13 Lightmagenta
2 Green	6 Brown	10 Lightgreen	14 Yellow
3 Cyan	7 Lightgray	11 Lightcyan	15 White

21.3 Weitere wichtige Grafikprozeduren, -funktionen und -konstanten (Auswahl)

Abkürzungen:

K Konstante
P Prozedur
F Funktion

K `Detect`
Zweck: Automatische Erkennung des Grafiktreibers.
Die automatische Erkennung des VGA-Karten problemlos möglich.

P `InitGraph(`*`grafiktreiber, grafikmodus, pfad`*`)`
Zweck: Initialisierung der Grafik.
`grafiktreiber` Integervariable. Vorher initialisiert, z.B. mit `Detect`.
`grafikmodus` Integervariable. Braucht mit Ausnahme der beiden genannten Grafikkarten nicht initialisiert zu sein, wenn die höchste Auflösungsstufe gewünscht wird.
`pfad` Zugriffspfad zum Grafiktreiber, z.B. zum Treiber für die VGA Karte "EGAVGA.BGI".

P `CloseGraph`
Zweck: Beendet den Grafikmodus und wechselt zum Textmodus. Am Programmende nicht notwendig.

P `PutPixel(`*`x, y, farbe`*`)`
Zweck: Pixel setzen/löschen. Siehe 21.2.

P `Line(`*`x1, y1, x2, y2`*`)`
Zweck: Linie vom Punkt P1(*`x1, y1`*) zum Punkt P2(*`x2, y2`*) ziehen.
`x1, y1, x2, y2` Integer-Ausdrücke
Hinweis: Bei allen Linien-Operationen (Linien, Rechtecke, Kreise usw.) werden die Linien mit dem Standardlinienart (SolidLn = durchgezogene Linie) und in der Standarddicke (NormWidth = 1 Pixel) gezeichnet, wenn nicht mit der Grafikprozedur "SetLineStyle" andere Linienarten oder -dicken vereinbart werden. Details siehe späteres Demo-Programm und Handbuch.

P `MoveTo(`*`x, y`*`)`
Zweck: Versetzt den (unsichtbaren) Grafikcursor zum Punkt P(*`x, y`*).
`x, y` Integer-Ausdrücke

P `LineTo(`*`x, y`*`)`
Zweck: Zeichnet Linie vom aktuellen Cursorpunkt zum Punkt P(*`x, y`*),
`x, y` Integer-Ausdrücke

P `RectAngle(x1, y1, x2, y2)`
Zweck: Zeichnet achsparalleles Rechteck mit der linken oberen Ecke P1(*x1, y1*) und der rechten unteren Ecke P2(*x2, y2*).
x1, y1, x2, y2 Integer-Ausdrücke

F `GetMaxX`
Zweck: Liefert maximalen X-Wert (z.B. 639 bei VGA 640 * 480) im Ergebnis-Datentyp Integer.

F `GetMaxY`
Zweck: Liefert maximalen Y-Wert (z.B. 479 bei VGA 640 * 480) im Ergebnis-Datentyp Integer.

P `Circle(x, y, radius)`
Zweck: Zeichnet Kreis um Mittelpunkt M(*x, y*) mit *radius*.
x, y Integer-Ausdrücke
radius Word-Ausdruck

P `Arc(x, y, winkelStart, winkelEnd, radius)`
Zweck: Zeichnet Kreisbogen um Mittelpunkt M(*x, y*) mit *radius* zwischen Anfangswinkel *winkelStart* und Endwinkel *winkelEnd*.
x, y Integer-Ausdrücke
radius Word-Ausdruck
winkelStart, *winkelEnd* Word-Ausdrücke. Winkel im Gradmaß, ansonsten aber Drehrichtung und Ausgangspunkt im mathematischen Sinne.

P `OutText(s)`
Zweck: Gibt String *s* an der aktuellen Position des Grafikcursors aus. Numerische Daten müssen in Strings gewandelt werden.
s String-Ausdruck

P `OutTextXY(x, y, s)`
Zweck: Gibt String *s* an der Position P(*x, y*) aus. Numerische Daten müssen in Strings gewandelt werden.
x, y Integer-Ausdrücke
s String-Ausdruck

F `GetPixel(x, y)`
Zweck: Liefert die *farbe* des Pixels P(*x, y*) mit Ergebnis-Datentyp Word. Werte zwischen 0 (= Black) und 15 (= White). Zu *farbe* siehe 21.2.
x, y Integer-Ausdrücke

Allgemeiner Hinweis: Es ist auch im Grafik-Modus möglich, mit »Read« bzw »ReadLn« Daten von der Tastatur einzulesen. Die Benutzereingaben erscheinen am aktuellen Punkt des Grafikcursors (wenn noch Cursor nicht bewegt wurde, dann in der linken oberen Bildschirmecke) und können mit der Taste »Backstep« editiert werden.

Das folgende Demo-Programm enthält weitere Grafik-Prozeduren und Funktionen. Die Erklärung ist aus der Programmumgebung zu entnehmen. Weitere Details siehe Online-Hilfe.

21.4 Allgemeines Demo-Programm

```
program Pas21041;  { Kapitel 21: Bildschirmgrafik }
  { Dieses Demo-Programm enthält z.T. fixe Koordinaten für die VGA-
    Farb-Grafikkarte in der Auflösung 640 * 480. Für andere Grafik-
    Karten wird im allgemeinen eine Anpassung der Koordinaten
    notwendig sein.
    Dieses Demo-Programm verwendet nur einen Teil der in der Unit
    »GRAPH.TPU« definierten Funktionen, Prozeduren und Konstanten
    und zwar:

    Bezeichner     Bedeutung
    ------------------------
    GetMaxX        function
    GetMaxY        function
    GraphResult    function
    GraphErrorMsg  function
    InitGraph      procedure
    PutPixel       procedure    Die wichtigste Grafik-Prozedur
    RectAngle      procedure
    OutTextXY      procedure
    Line           procedure
    MoveTo         procedure
    SetLineStyle   procedure
    Line           procedure
    LineTo         procedure
    Bar            procedure
    Circle         procedure
    CloseGraph     procedure
    SetTextStyle   procedure
    SetViewPort    procedure
    ClearViewPort  procedure
    ClipON         const
    ClipOFF        const
    grOk           const
    Detect         const
    Black          const
    White          const
    DottedLn       const
    NormWidth      const
    ThickWidth     const
    SolidFill      const
    CloseDotFill   const
    XHatchFill     const
    ShlashFill     const
    BkSlashfill    const
    GothikFont     const
    HorizDir       const
    VertDir        const
    ------------------------------------------------------------------
  }

uses
  CRT, GRAPH;  { Die Turbo-Pascal-Unit »GRAPH.TPU« }

const
  Zugriffspfad = 'C:\BP\BGI'; { Zugriffspfad z. Treiber »EGAVGA.BGI« }
                              { Ggf. anpassen                        }
```

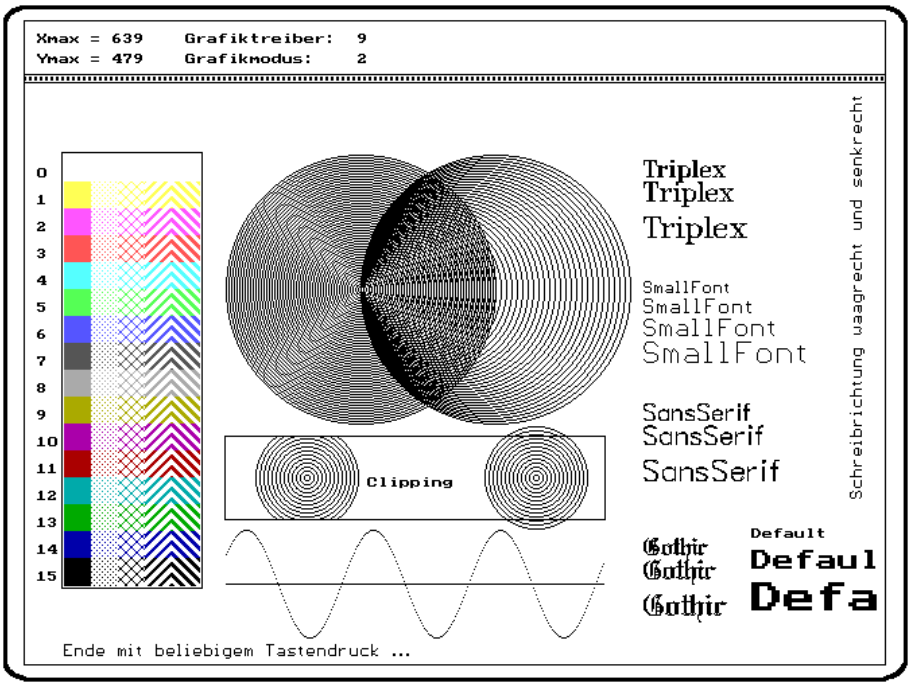
```
var
  Grafiktreiber,
  Grafikmodus,
  Grafikfehlercode:  Integer;

  Xmax:              Integer;
  Ymax:              Integer;
  Farbe,
  Schriftgroesse:    Word;
  x,  y,
  x1, y1,
  x2, y2,
  ye,
  DeltaX,
  DeltaY,
  Radius:            Integer;
  XmaxStr,
  YmaxStr:           string[3];
  GrafiktreiberStr,
  GrafikmodusStr,
  FarbeStr:          string[3];


begin
  Grafiktreiber := Detect;       { Konstante "Detect" aus Unit GRAPH }
         { Automatische Erkennung des installierten Grafiktreibers   }

  InitGraph(Grafiktreiber, Grafikmodus, Zugriffspfad);
           { Wenn mehrere Grafik-Modi existieren, wählt die Grafik-
             Prozedur »InitGraph« den mit der höchsten Auflösung aus }

  Grafikfehlercode := GraphResult;   { Funktion "GraphResult }
                                     { aus Unit GRAPH        }
  if Grafikfehlercode <> grOk then   { Konstante "grOK" aus  }
     begin                           { Unit GRAPH            }
       WriteLn('Grafik-Fehler: ', GraphErrorMsg(Grafikfehlercode));
       WriteLn('Programmabbruch notwendig ...');
       Halt;
     end;

  Xmax := GetMaxX;   { liefert maximale X-Koordinate }
  Ymax := GetMaxY;   { liefert maximale Y-Koordinate }

  Str(Xmax,           XmaxStr);          { Konvertierung in String }
  Str(Ymax,           YmaxStr);
  Str(Grafiktreiber, GrafiktreiberStr);
  Str(Grafikmodus,   GrafikmodusStr);

{ ------------ Rechteck zeichnen ---------------------------------- }
  x1 := 0;  x2 := Xmax;
  y1 := 0;  y2 := Ymax;
  RectAngle(x1, y1, x2, y2);

{ ------------ Text an der Stelle (X, Y) ausgeben ----------------- }
  OutTextXY(10,  10, 'Xmax = ' + XmaxStr);
  OutTextXY(10,  25, 'Ymax = ' + YmaxStr);
  OutTextXY(120, 10, 'Grafiktreiber:  ' + GrafiktreiberStr);
  OutTextXY(120, 25, 'Grafikmodus:    ' + GrafikmodusStr);

{ ------------ Linie zeichnen ------------------------------------- }
  x1 := 0;  x2 := Xmax;
  y1 := 40; y2 := y1;
  Line(x1, y1, x2, y2);

{ ------------ Linien zeichnen, Linientyp einstellen -------------- }
  MoveTo(0, 43);
  SetLineStyle(DottedLn, 0, ThickWidth);
  LineTo(Xmax, 43);   { Bewegung vom Grafik-Cursor aus }
```

```
  SetLineStyle(SolidLn, 0, NormWidth);
  Line(0, 46, Xmax, 46);

{ ------------ gefüllten Balken zeichnen, Füllmuster --------------- }
  DeltaX := 20;
  DeltaY := 20;
  x1 :=  30;  x2 := x1 + DeltaX;
  y1 := 100;  y2 := y1 + DeltaY;

  RectAngle(x1 - 2,            y1 - 2,
            x2 + 4*DeltaX + 2, y2 + 15*DeltaY + 2);

  for Farbe := Black to White do   { 16 Farben, von 0 bis 15 }
    begin
      Str(Farbe, FarbeStr);
      OutTextXY(x1 - 20, y1 + Farbe*DeltaY + DeltaY div 2, FarbeStr);

      SetFillStyle(SolidFill, Farbe);
      Bar(x1, y1 + Farbe*DeltaY,
          x2, y2 + Farbe*DeltaY); { zeichnet gefüllten Balken }

      SetFillStyle(CloseDotFill, Farbe);
      Bar(x1 + DeltaX, y1 + Farbe*DeltaY,
          x2 + DeltaX, y2 + Farbe*DeltaY);

      SetFillStyle(XHatchFill, Farbe);
      Bar(x1 + 2*DeltaX, y1 + Farbe*DeltaY,
          x2 + 2*DeltaX, y2 + Farbe*DeltaY);

      SetFillStyle(SlashFill, Farbe);
      Bar(x1 + 3*DeltaX, y1 + Farbe*DeltaY,
          x2 + 3*DeltaX, y2 + Farbe*DeltaY);

      SetFillStyle(BkSlashFill, Farbe);
      Bar(x1 + 4*DeltaX, y1 + Farbe*DeltaY,
          x2 + 4*DeltaX, y2 + Farbe*DeltaY);
    end;

{ ------------ Kreis zeichnen ------------------------------------- }
  Radius := 100;
  x1 := 250;
  y1 := 200;
  while Radius > 0 do
    begin
      Circle(x1,          y1, Radius);
      Circle(x1 + Radius, y1, Radius);
      Dec(Radius, 2);
    end;

{ ------------ Punkte setzen -------------------------------------- }
  x1 := 150;   x2 := 430;
  y1 := Ymax - 60;
  Line(x1, y1, x2, y1);
  Farbe := White; { »White« = 15 }
  for x := x1 to x2 do
    begin
      y :=  y1 + Round(40*Sin(x/15));
      PutPixel(x, y, Farbe); { Mit der Farbe »Black« = 0 wird
                               ein gesetzter Punkt gelöscht    }
    end;

{ ------------ Fenster setzen, mit/ohne Clipping ------------------- }
  x1 := 150;   y1 := 310;
  x2 := 430;   y2 := y1 + 60;
  RectAngle(Pred(x1), Pred(y1), Succ(x2), Succ(y2));

  SetViewPort(x1, y1, x2, y2, ClipON);    { Fenster setzen }
```

```
    { SetViewPort selbst arbeitet immer mit absoluten Koordinaten.   }
    { Ansonsten gelten alle anderen Grafikkoordinaten anschließend   }
    { relativ zur linken oberen Ecke des gesetzten Fensters.         }

    { Konstante ClipON  (= True):   mit  Clipping }
    { Konstante ClipOFF (= False ): ohne Clipping }

  ClearViewPort;      { Fenster löschen, hier nicht notwendig }

  OutTextXY(105, 30, 'Clipping');

  for Radius := 2 to 40 do
    begin
      Circle(60, 30, Radius);
      Radius := Radius + 2;
    end;

  SetViewPort(x1, y1, x2, y2, ClipOFF);   { ohne Clipping }
  for Radius := 2 to 40 do
    begin
      Circle(180 + 50, 30, Radius);
      Radius := Radius + 2;
    end;

  SetViewPort(0, 0, Xmax, yMax, ClipON);  { ganzer Bildschirm }

{ ------------ Schriften ------------------------------------------- }
  { Format für "SetTextStyle":                                       }
  {              SetTextStyle(font, richtung, groesse)               }
  {                                                                  }
  { font:       Word-Ausdruck, 0..4, oder:                           }
  {             DefaultFont   (= 0, Pixelschrift)                    }
  {             TriplexFont   (= 1, Vektorschrift, Datei Trip.CHR    }
  {             SmallFont     (= 2, VektorSchrift, Datei Litt.CHR    }
  {             SansSerifFont (= 3, Vektorschrift, Datei Sans.CHR    }
  {             GothicFont    (= 4, Vektorschrift, Datei Goth.CHR    }
  {             Die Vektorschriften können nur benutzt werden, wenn  }
  {             die angegebenen Schrift-Dateien des BGI existieren.  }
  {             Sie werden bei der Installation von Turbo-Pascal auf }
  {             einer Festplatte C im folgenden Verzeichnis abgelegt:}
  {             "C:\BP\BGI"                                          }
  { richtung:   Word-Ausdruck, aber nur 0 oder 1, bzw.:              }
  {             HorizDir (= 0, horizontale Richtung)                 }
  {             VertDir  (= 1, vertikale Richtung  )                 }
  { groesse:    Word-Ausdruck, Skalierungsfaktor. Standardeinstellung}
  {             beim DefaultFont 1, bei Vektorschriften 4. Skalierung}
  {             ist nur sinnvoll bei Vektorschriften, da DefaultFont }
  {             mit einer 8 * 8 - Pixeldarstellung arbeitet, die bei }
  {             Vergrößerung sehr rauh wirkt.                        }

  x1 := 460; y1 := 90;
  for Schriftgroesse := 1 to 3 do
    begin
      SetTextStyle(TriplexFont, HorizDir, Schriftgroesse);
      y1 := y1 + Schriftgroesse * 8;
      OutTextXY(x1, y1, 'Triplex');
    end;

  x1 := 460; y1 := 180;
  for Schriftgroesse := 5 to 8 do
    begin
      SetTextStyle(SmallFont, HorizDir, Schriftgroesse);
      y1 := y1 + Schriftgroesse * 2;
      OutTextXY(x1, y1, 'SmallFont');
    end;

  x1 := 460; y1 := 270;
```

```
  for Schriftgroesse := 1 to 3 do
    begin
      SetTextStyle(SansSerifFont, HorizDir, Schriftgroesse);
      y1 := y1 + Schriftgroesse * 8;
      OutTextXY(x1, y1, 'SansSerif');
    end;

  x1 := 460; y1 := 370;
  for Schriftgroesse := 1 to 3 do
    begin
      SetTextStyle(GothicFont, HorizDir, Schriftgroesse);
      y1 := y1 + Schriftgroesse * 8;
      OutTextXY(x1, y1, 'Gothic');
    end;

  x1 := 540; y1 := 370;
  for Schriftgroesse := 1 to 3 do
    begin
      SetTextStyle(DefaultFont, HorizDir, Schriftgroesse);
      y1 := y1 + Schriftgroesse * 8;
      OutTextXY(x1, y1, 'Default');
    end;

{ ------------ Schrifttyp und Schreibrichtung ändern --------------- }
  SetTextStyle(SmallFont, VertDir, 5);
  OutTextXY(Xmax - 30, 50, 'Schreibrichtung waagrecht und senkrecht');

{ ------------ Feierabend ------------------------------------------ }
  SetTextStyle(SmallFont, HorizDir, 5);
  OutTextXY(30, 460, 'Ende mit beliebigem Tastendruck ... ');
  repeat
  until KeyPressed;

  CloseGraph;  { Umschalten in den Textmodus }

end.
```

Die Ausgabe des Programms Pas21041.PAS:



21.5 Demo-Programm Moiré

Das folgende Programm erzeugt Moiré-ähnliche Grafiken, fast so schön wie Apfelmännchen. Der Grundgedanke:

Man setze in eine nahezu willkürliche Funktion die Koordinaten xk und yk ein. Dann betrachte den ganzzahligen Teil der Funktion: Ist dieser geradzahlig, dann setze man bei xk, yk einen Punkt, sonst nicht (Rechnung modula 2). Damit kann man z.B. mit Monochrom-Karten bereits s/w-Graphiken erstellen. Bei Farbkarten rechnet man nicht modulo 2, sondern z.B. modulo 16 (bei Standard-VGA). Die Funktion normiere man zweckmäßigerweise so, daß der Wertebereich 0 bis 65535 abgedeckt wird, was dem Pascal-Datentyp »Word« entspricht. Im vorliegenden Programm wird folgende Funktion benutzt:

$$F(x,y) = x^p + y^{2p}$$

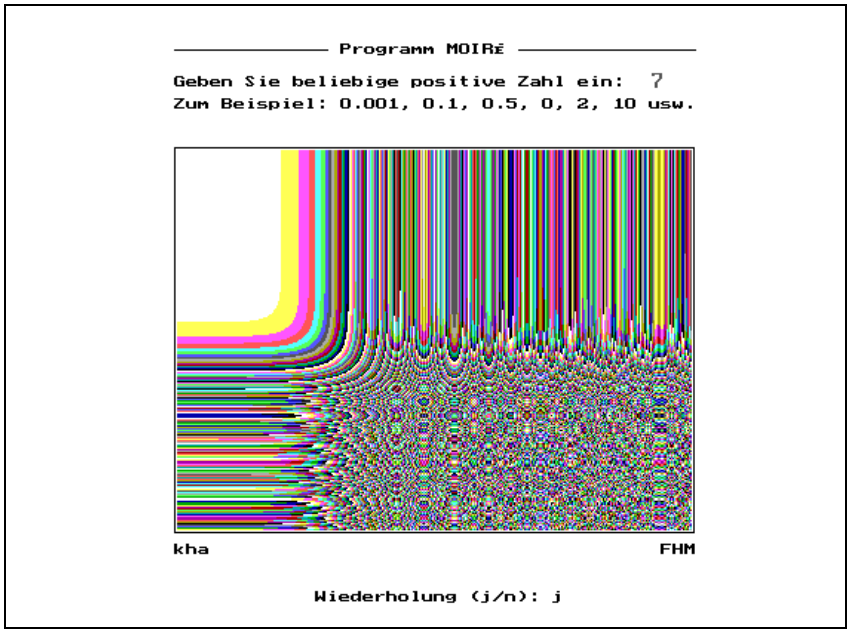
x und y sind normierte Koordinaten (x, y = 0 ... 1), p ist ein frei wählbarer (positiver) Parameter, mit dem höchst unterschiedliche Graphiken dargestellt werden können. Die Funktion liefert Werte zwischen 0 und 2, da jeder der beiden Terme Werte zwischen 0 und 1 liefert. Durch Multiplikation mit »MaxInt« (Pascal-Konstante mit dem Wert 32767) wird die gewünschte Normierung auf den Datentyp »Word« erreicht.

In Pascal ist bekanntlich die Potenzfunktion z^n nicht implementiert. Man muß sie mit e^(n * ln z) darstellen. In Pascal-Schreibweise: z^n = Exp(n * Ln(z))

```
program Pas21051; { Moiré, Turbo-Pascal, VGA 640 * 480, K. Haller }

uses
  CRT, GRAPH;

const
  xMin = 150;  xMax = 500;  { Zeichenfläche x-Richtung waagrecht     }
  yMin = 100;  yMax = 360;  { Zeichenfläche y-Richtung senkrecht     }
  Grafikpfad = 'C:\BP\BGI'; { In diesem Verzeichnis »Graph.TPU«      }
  FarbenMax  = 16;          { Bei VGA »16« Farben                    }

type
  TReal = Extended;

var
  xk, yk:        Integer;
  Farbe:         Byte;
  p, x, y,
  yTerm,
  Funktion:      TReal;
  Taste:         Char;
  Fehlercode:    Integer;
  pStr:          string;
  Grafikmodus,
  Grafiktreiber: Integer;

begin
  Grafiktreiber := Detect;

  InitGraph(Grafiktreiber, Grafikmodus, Grafikpfad);

  repeat
    RectAngle(xMin, yMin, xMax, yMax); { zeichnet Rechteck }

    OutTextXY(xMin, 30, '------------- Programm MOIRÉ ----------' +
                        '-----');
    OutTextXY(xMin, 52, 'Geben Sie beliebige positive Zahl ein: ' +
                        '     ');
    OutTextXY(xMin, 67, 'Zum Beispiel: 0.001, 0.1, 0.5, 0, 2, '   +
                        '10 usw.');
    OutTextXY(xMin, yMax + 8, 'kha                              '   +
                        '          FHM');
    repeat
      GotoXY(10 + 50, 4);
      ReadLn(pStr);
      Val(pStr, p, Fehlercode);
    until (Fehlercode = 0) and (p >= 0.0);

    for yk := yMin + 2 to yMax - 2  do
      begin
        y     := (yk - yMin) / (yMax - yMin);
                 { y = 0..1      }
        yTerm := Exp(2*p*Ln(y));
                 {.. = y^(2*p)   }

        for xk := xMin + 2 to xMax - 2 do
          begin
            x        := (xk - xMin) / (xMax - xMin);
                        { x = 0..1      }
            Funktion := Exp(p*Ln(x)) + yTerm;
                        { Funktion = 0..2      }
            Funktion := MaxInt * Funktion;
                        { F = 0..65535  }
            Farbe    := Round(Funktion) mod FarbenMax;
```

```
                          { Farbe = 0..15 }
            PutPixel(xk, yk, Farbe);
          end;
      end;

    OutTextXY(xMin + 95, yMax + 40, 'Wiederholung (j/n): j');

    repeat
      Taste := ReadKey;
      if Taste = #13
         then Taste := 'j';
    until UpCase(Taste) in ['J', 'N'];

    ClearDevice;   { Löscht Grafik-Bildschirm }

  until UpCase(Taste)  = 'N';

  CloseGraph;
end.
```

Eine Ausgabe des Programms Pas21051.PAS:



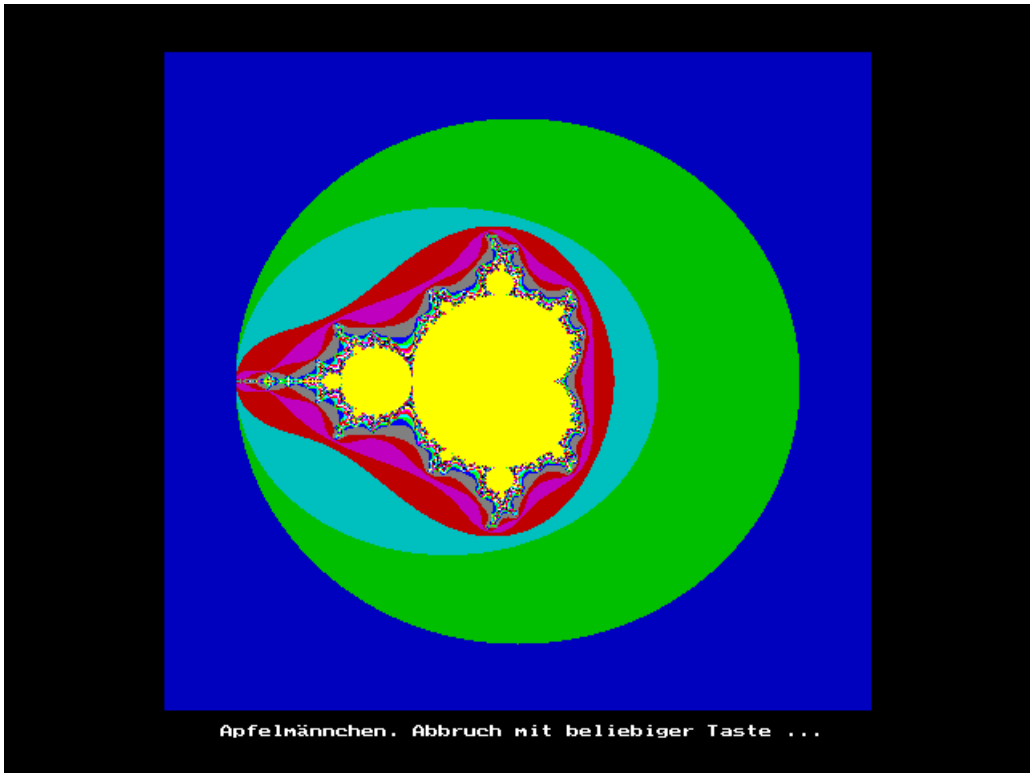
21.6 Demo-Programm Fraktale 1 (B. Mandelbrot)

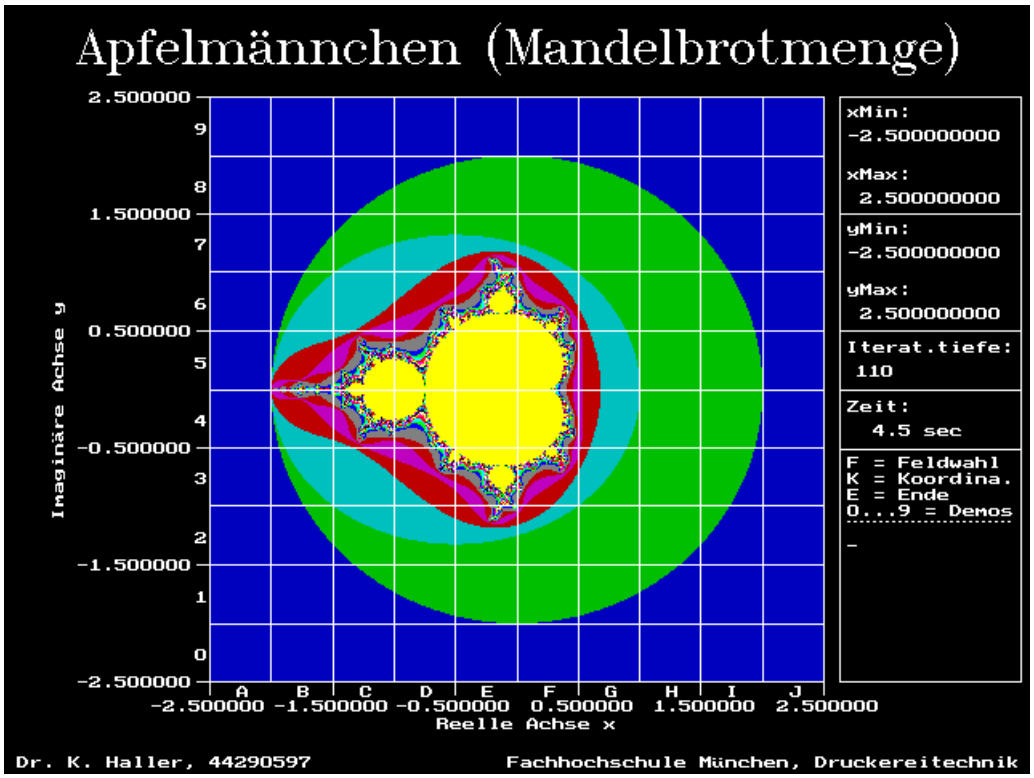
```
{$N+ Coprozessor benutzen }
program Pas21061; { Das Apfelmännchen der Mandelbrotmenge }
                  { Turbo-Pascal, VGA-Graphik,  K. Haller }
uses
  CRT, GRAPH;

const
  FarbenMax = 16;     { In Turbo-Pascal max. 16 Farben    }
  XK_Min = 100;  XK_Max = 540;   { Bildschirm-Koordinaten }
  YK_Min =  30;  YK_Max = 440;
```

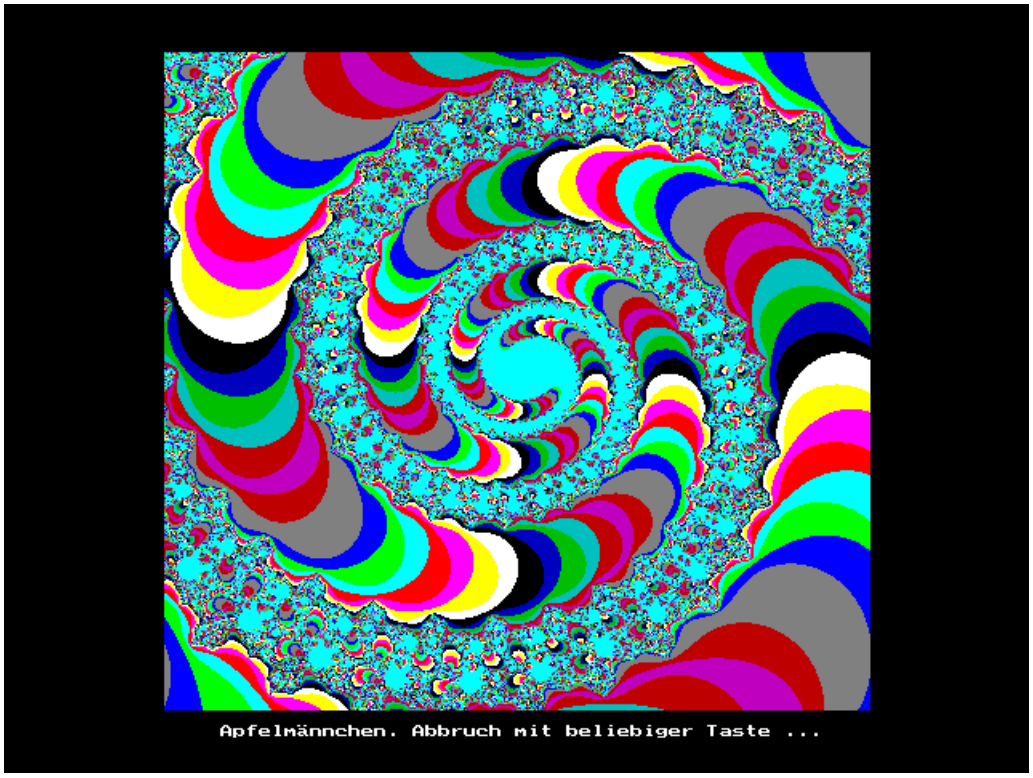
```
var
 Ende:            Boolean;
 xMin, xMax,
 yMin, yMax:      Double;
 Ch:              Char;
 Iterationstiefe: Word;
    { Zur Iterationstiefe ---------------------------------- }
    { Man teste auch mit: 0, 1, 2, 3, 4, ... 999, ....       }
    { Je höher, desto besser die Auflösung, dafür aber auch  }
    { längere Rechenzeiten. Mit "110" bei VGA, 16 Farben:    }
    { im Zentrum gelbe Farbe. Je kleiner der Auschnitt aus   }
    { dem Apfelmännchen, desto höher sollte die Iterations-  }
    { tiefe gewählt werden. }

procedure Apfelmaennchen(XK Min, XK Max,
                         YK Min, YK Max:  Integer;
                         xMin,   xMax,
                         yMin,   yMax:    Double;
                         Iterationstiefe,
                         FarbenMax:       Word);
  { Benoit Mandelbrot: Amerikanischer Mathematiker.
    Iterationsvorschrift für Mandelbrotmenge: zNeu = z*z + c,
    Komplexe Zahl z, komplexe Konstante c. Dabei ist c die
    Koordinate des aktuellen Punktes in der komplexen Zahlenebene.
    Wobei: z = x  + j * y,       x:  Realteil,  y:  Imaginärteil
           c = cx + j * cy,      cx: Realteil,  cy: Imaginärteil
           mit: j =  √(-1), Wurzel aus -1, imaginäre Einheit
    Die Rechnung ergibt mit Berücksichtigung von j*j = -1:
    zNeu = (x*x - y*y + cx)  + j * (2*x*y + cy)
    d.h: zNeuReal = xNeu = x*x - y*y + cx
         zNeuImag = yNeu = 2*x*y     + cy     }
const
  RadiusMax = 2.0; { Für Apfelmännchen: 2.0, ansonsten spiele man }
var
  x,   y,
  cx,  cy,
  cxDelta,
  cyDelta,
  xAlt:    Double;
  XK, YK:  Integer;  { Pixelkoordinaten }
  i:       Word;     { Iterationszähler }
  Ch:      Char;
begin
  cxDelta := (xMax - xMin) / (XK Max - XK Min);
  cyDelta := (yMax - yMin) / (YK Max - YK Min);
  cy := yMin;
  for YK := YK Max downto YK Min do   { senkrechte Pixel-Koordinaten }
    begin
      cx := xMin;
      for XK := XK Min to XK Max do   { waagrechte Pixel-Koordinaten }
        begin
          x := 0.0;  { Man teste auch mit |x|, |y|   2.0 }
          y := 0.0;  { Für Apfelmännchen: 0.0 und 0.0    }
          i := 0;    { Iterationszähler }
          repeat
            Inc(i);
            xAlt := x;          { xAlt auch für yNeu }
            x    := Sqr(x) - Sqr(y) + cx;     { xNeu }
            y    := 2 * xAlt * y   + cy;      { yNeu }
          until (Sqr(x) + Sqr(y) >= Sqr(RadiusMax)) or
```

```
                (i = Iterationstiefe);
           PutPixel(XK, YK, i mod FarbenMax);
           cx := cx + cxDelta;
         end; { Ende XK-for-Schleife }
       cy := cy + cyDelta;
       if KeyPressed then
          begin
            CloseGraph;
            Exit;
          end;
    end;     { Ende YK-for-Schleife }
end;  { ****** Ende Prozedur "Apfelmaennchen(...)" ********* }

procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
  Write(Meldung);
end;

procedure Grafik Initialisieren;
const
  Zugriffspfad = 'C:\BP\BGI';  { Gegebenenfalls anpassen }
var
  Grafiktreiber,
  Grafikmodus,
  Fehlercode:   Integer;
begin
  Grafiktreiber := Detect;
  InitGraph(Grafiktreiber, Grafikmodus, Zugriffspfad);
  Fehlercode := GraphResult;
  if FehlerCode <> GrOK then
     begin
       ClrScr;
       WriteXY(10, 10, 'Grafikfehler: ');
       WriteLn(GraphErrorMsg(Fehlercode));
       WriteXY(10, 14, 'Programmabbruch notwendig. ' +
                       'Drücke beliebige Taste ... ');
       repeat
       until ReadKey <> '';
       Halt(1); { >>>>>>>>>>>>>>>>>>>>>>>>> }
     end;
  ClearDevice;
  OutTextXY(135, 450, 'Apfelmännchen. ' +
            'Abbruch mit beliebiger Taste ... ');
end;

procedure Menue(var xMin, xMax, yMin, yMax: Double;
                var Iterationstiefe: Word;
                var Ende: Boolean);
var
  Ch: Char;
begin
  ClrScr;
  WriteXY(10, 6, 'Das Apfelmännchen der Mandelbrotmenge:');
  WriteXY(10, 8, 'Eingabe Demo 1...3, (0 für Ende): ');
  while KeyPressed do
    Ch := ReadKey;
  repeat
    Ch := ReadKey;
  until Ch in ['0'..'3'];
  Write(Ch);
```

```
  Ende := False;
  case Ch of
  '0': Ende := True;
  '1': begin
         xMin := -2.5;   yMin := -2.5;
         xMax := +2.5;   yMax := +2.5;
         Iterationstiefe := 110;
       end;
  '2': begin
         xMin :=  0.36067;     yMin :=  0.58583;
         xMax :=  0.36068;     yMax :=  0.58584;
         Iterationstiefe := 110;
       end;
  '3': begin
         xMin :=  -0.7667805;  yMin :=   0.1000325;
         xMax :=  -0.7667790;  yMax :=   0.1000340;
         Iterationstiefe := 299; { Bei sehr kleinen Ausschnitten }
       end;    { große Iterationstiefe notwendig. Bei Beispiel 3 }
  end;         { mit Tiefe 299 schönes Farbenspiel }
end;

begin { ----- Hauptprogramm ------- }
  repeat
    Menue(xMin, xMax, yMin, yMax, Iterationstiefe, Ende);
    if not Ende then
       begin
         Grafik Initialisieren;
         Apfelmaennchen(XK Min, XK Max,
                        YK Min, YK Max,
                        xMin,   xMax,
                        yMin,   yMax,
                        Iterationstiefe,
                        FarbenMax);
         repeat
         until ReadKey <> '';
         CloseGraph;
       end;
  until Ende;
end.
```


Apfelmännchen. Abbruch mit beliebiger Taste ...


Apfelmännchen (Mandelbrotmenge)
2.500000
1.500000
0.500000
-0.500000
-1.500000
-2.500000
Imaginäre Achse y
A B C D E F G H I J
-2.500000 -1.500000 -0.500000 0.500000 1.500000 2.500000
Reelle Achse x
xMin:
-2.500000000
xMax:
2.500000000
yMin:
-2.500000000
yMax:
2.500000000
Iterat.tiefe:
110
Zeit:
4.5 sec
F = Feldwahl
K = Koordina.
E = Ende
0...9 = Demos
Dr. K. Haller, 44290597
Fachhochschule München, Druckereitechnik



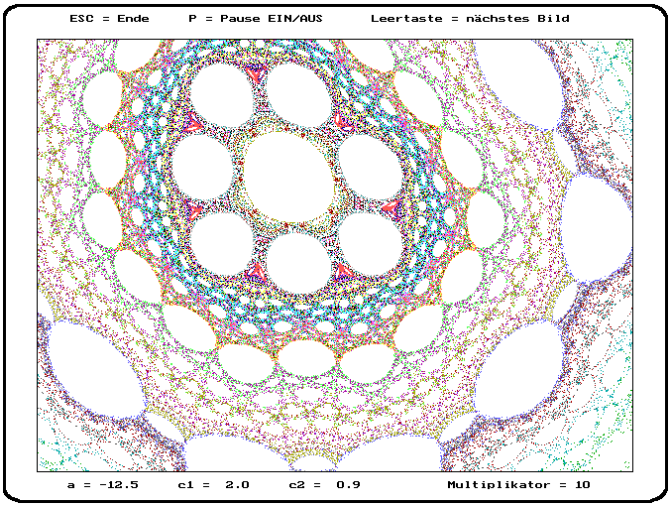
21.7 Demo-Programm Fraktale 2 (B. Martin)

```
{$N+ Coprozessor benutzen }
program Pas21071; { Fraktale mit reellen Zahlen nach Barry Martin  }
                  {    y - f(x) --> x                              }
                  {    a - x    --> y                              }
                  { Mit  f(x) = Sign(x) * Sqrt(Abs(c1*x - c2)      }
                  { Modifikation nach: MC, 10/92, S. 88, H. Scheid }
                  { Man teste auch mit anderen f(x)                }
uses
  CRT, GRAPH;

type
  TReal = Extended;

const
  FarbenNrMax = 16;
  Esc         = #27;
  ChDummy     = '?';
  Multiplikator     =    10;   { Man teste auch mit anderen Werten }
  FarbwechselModulo = 10000;   { Man teste auch mit anderen Werten }

var
  Grafiktreiber,
  Grafikmodus,
  xMin, xMax,
  yMin, yMax,
  MaxX Halbe,
  MaxY_Halbe:    Integer;
```

```
  Str1, Str2:    string;
  Ch:            Char;
  x, y, fx,
  a, c1, c2:     TReal;
  xP, yP:        Integer;   { Pixelkoordinate }
  FarbenNr:      Byte;
  Zaehler:       LongInt;

function Signum(x: TReal): ShortInt;
begin
  if x >= 0
     then Signum := +1
     else Signum := -1;
end;

begin  { ------------------------------------------------------------ }
  ClrScr;

  Grafiktreiber := Detect;
  InitGraph(Grafiktreiber, Grafikmodus, 'C:\BP\BGI');  { anpassen! }

  xMin := 20;   xMax := GetMaxX - 20;
  yMin := 25;   yMax := GetMaxY - 20;

  MaxX Halbe := (xMax - xMin) div 2;
  MaxY Halbe := (yMax - yMin) div 2;

  Randomize;

  repeat
    SetViewport(0, 0, GetMaxX, GetMaxY, ClipON);
      { Zur vordefinierten Grafikkonstanten "ClipON" = True,   }
      { automatische Clipping-Kontrolle. Gegenstück: "ClipOFF" }
      { In der momentanen Situation "ClipON" nicht wesentlich  }
    ClearViewport;

    { Zufallsparameter a, c1, c2 }
    a  := -250 + Random(2*250 + 1);   a  := a/10;
    c1 :=  -30 + Random(2*30  + 1);   c1 := c1/10;
    c2 :=  -10 + Random(2*10  + 1);   c2 := c2/10;

    { Es folgt String-Ausgabe aller Parameter }
    Str(a:4:1, Str1);   Str2 :=         '     a = '  + Str1;
    Str(c1:4:1, Str1);  Str2 := Str2 + '     c1 = ' + Str1;
    Str(c2:4:1, Str1);  Str2 := Str2 + '     c2 = ' + Str1;
    Str(Multiplikator, Str1);
    Str2 := Str2 + '           Multiplikator = ' + Str1;
    OutTextXY(10, 470, Str2);
    Str2 := '      ESC = Ende     P = Pause EIN/AUS      ' +
            'Leertaste = nächstes Bild';
    OuttextXY(5, 0, Str2);

    RectAngle(xMin - 1, yMin - 1, xMax + 1, yMax + 1);
    SetViewPort(xMin, yMin, xMax, yMax, ClipON);
                                      { "ClipON" hier sinnvoll }
    Ch       := ChDummy;
    x        := 0.0;
    y        := 0.0;
    FarbenNr := FarbenNrMax div 2;
    Zaehler  := 0;

    repeat
      Inc(Zaehler);
```

```
      fx := Signum(x) * Sqrt(Abs(c1*x - c2)); { Man teste auch }
       { andere f(x), z.B: fx := Sin(x) + Cos(x); mit a := 1.0 }

      fx := y - fx;
      y  := a - x;
      x  := fx;

      xP := Round(x * Multiplikator) + MaxX Halbe;
      yP := Round(y * Multiplikator) + MaxY Halbe;

      if Zaehler mod FarbwechselModulo = 0
         then FarbenNr := (FarbenNr + 1) mod FarbenNrMax;

      PutPixel(xP, yP, FarbenNr);

      if KeyPressed then
         begin
           Ch := UpCase(ReadKey);
           if Ch = 'P' then
              repeat                              { 'P' = Pause }
              until UpCase(ReadKey) = 'P';        { nochmal 'P' }
         end;
    until Ch in [Esc, ' '];

  until Ch = Esc;

  CloseGraph;
end.
```

Eine Ausgabe des Programms Pas25071.PAS:



21.8 Demo-Programm Fourier-Reihe

Nach Fourier kann man jede beliebige periodische Funktion, die auch Unstetigkeiten enthalten darf, durch eine trigonometrische Summe darstellen. Bei einer Summation bis ins Unendliche wird die Funktion exakt, in anderen Fällen näherungsweise dargestellt.

Die Zerlegung eines beliebigen Signals $f(t)$ in eine Fourier-Reihe ist Aufgabe der Fourier-Transformation. Bei technischen Anwendungen (Schwingungsanalyse: eindimensionale Transformation, Bildanalyse: zweidimensionale Transformation) beschränkt man sich aus praktischen Gründen (Rechenzeit!) auf diskrete Punkte des Signals. Man spricht dann von einer diskreten Fourier-Transformation (DFT). Nach dem Shannonschen Abtasttheorem läßt sich $f(t)$ exakt aus den Abtastwerten rekonstruieren, wenn die Abtastfrequenz größer ist als das Doppelte der höchsten in $f(t)$ vorkommenden Frequenz.

Mit der FFT (Fast Fourier Transformation) steht ein besonders schneller Algorithmus für die diskrete Fourier-Transformation zur Verfügung, der in vielen Fällen erst eine sinnvolle technische Anwendung ermöglicht. Im Rechenzeitverhältnis steht die "normale" DFT zur FFT wie etwa MinimumSort zu QuickSort bei den Sortieralgorithmen.

Im nachfolgenden Demo-Programm wird die Fourier-Reihe für drei einfache periodische Funktionen gezeigt:

- Sägezahn $y = x$ für: $-\pi < x < \pi$
  Fourier-Reihe: $$y = 2\left(\frac{\sin x}{1} - \frac{\sin 2x}{2} + \frac{\sin 3x}{3} - .....\right)$$
- Rechteck $y = 1$ für: $0 < x < \pi$
  $y = -1$ für: $\pi < x < 2\pi$
  Fourier-Reihe: $$y = \frac{4}{\pi}\left(\frac{\sin x}{1} + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + ...\right)$$
- Dreieck $y = x$ für: $-\frac{\pi}{2} \le x \le \frac{\pi}{2}$
  $y = \pi - x$ für: $\frac{\pi}{2} \le x \le \frac{3}{2}\pi$
  Fourier-Reihe: $$y = \frac{4}{\pi}\left(\frac{\sin x}{1^2} - \frac{\sin 3x}{3^2} + \frac{\sin 5x}{5^2} - .....\right)$$

Im Programm werden die y-Werte zu eins normiert, damit die Grafikausgaben alle die gleiche Höhe haben.

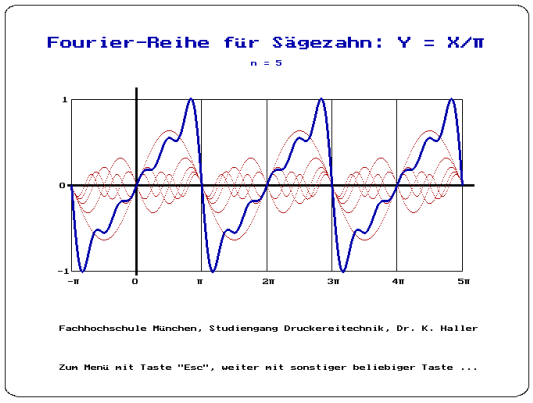
Man beachte das Überschwingen beim Sägezahn und beim Rechteck an den Unstetigkeitsstellen (Gibbsches Phänomen).

```
program Pas21081;  { Fourier-Reihen, 34080897, K. Haller }

uses
  CRT, GRAPH;

const
  Pfad  = 'C:\BP\BGI';   { Ggf. anpassen }
  x0    =   70;
  xKMin =   x0;  xKMax = x0 + 500;
  yKMin =  100;  yKMax = 400;
  y0    =  220;  dy    = 110;
```

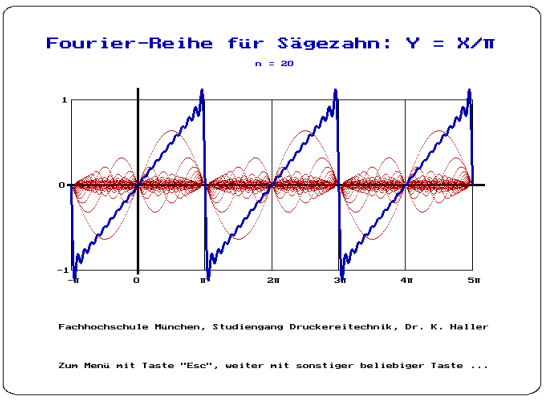
```
var
  Kurve:   (Saegezahn, Rechteck, Dreieck);
  xK, yK,
  k, Farbe:     Integer;
  x, y,
  yTemp,
  Normierung:   Real;
  n, i:         Word;
  nStr,
  PiStr,
  Legende:      string;
  Ch:           Char;
  Grafiktreiber,
  Grafikmodus:  Integer;

procedure Rahmen;
begin
  SetColor(Yellow);
  SetTextStyle(DefaultFont, HorizDir, 2);
  OutTextXY(40, 30, 'Fourier-Reihe f•r ' + Legende);
  SetTextStyle(DefaultFont, HorizDir, 1);
  SetTextStyle(DefaultFont, HorizDir, 1);
  Str(n, nStr);
  OutTextXY(xKMin + 230, y0 - dy - 50, 'n = ' + nStr);
  SetColor(White);
  RectAngle(xKmin, y0 - dy, xKMax, y0 + dy);
  for k := -1 to 5 do
    begin
      Line(xKmin + (k + 1)*(xKMax - xKMin) div 6, y0 - dy,
           xKmin + (k + 1)*(xKMax - xKMin) div 6, y0 + dy);

      Moveto(xKmin + (k + 1)*(xKMax - xKMin) div 6 - 5, y0 + dy + 10);

      Str(k, PiStr);
      if PiStr =  '1' then PiStr := '';
      if PiStr = '-1' then PiStr := '-';
      if PiStr <> '0'
         then OutText(PiStr + 'π')
         else OutText(PiStr);
    end;

  MoveTo(xKMin - 15, y0 - 3);       OutText('0');
  MoveTo(xKMin - 18, y0 + dy - 3);  OutText('-1');
  MoveTo(xKMin - 12, y0 - dy - 3);  OutText('1');

  SetLineStyle(0, 0, Thickwidth);
  Line(xKmin + (xKMax - xKMin) div 6, y0 - dy - 15,
       xKmin + (xKMax - xKMin) div 6, y0 + dy +  5);
  Line(xKmin -  5, y0,
       xKmin + 6*(xKMax - xKMin) div 6 + 15, y0);

  SetLineStyle(0, 0, NormWidth);

  OutTextXY(55, 450, 'Zum Menü mit Taste "Esc", ' +
                     'weiter mit sonstiger beliebiger Taste ... ');
  OutTextXY(55, 400, 'Fachhochschule München, Studiengang ' +
                     'Druck- und Medientechnik, Dr. K. Haller');
```

```
end;

begin
  Grafiktreiber := Detect;
  InitGraph(Grafiktreiber, Grafikmodus, Pfad);

  repeat
    ClearDevice;
    SetColor(Yellow);
    OutTextXY(250,  70, 'Fourier-Reihen');
    SetColor(White);
    OutTextXY(250, 100, '1   Sägezahn');
    OutTextXY(250, 115, '2   Rechteck');
    OutTextXY(250, 130, '3   Dreieck ');
    OutTextXY(250, 145, '0   Ende    ');
    OutTextXY(250, 160, '------------');

    repeat
      Ch := ReadKey;
    until Ch in ['0'..'3', #27];

    if (Ch = '0') or (Ch = #27)
       then Halt;

    case Ch of
    '1': begin
           Kurve      := Saegezahn;
           Legende    := 'Sägezahn: Y = X/π';
           Normierung := 2.0/Pi;
         end;
    '2': begin
           Kurve      := Rechteck;
           Legende    := 'Rechteck: Y = ±1';
           Normierung := 4.0/Pi;
         end;
    '3': begin
           Kurve      := Dreieck;
           Legende    := 'Dreieck: Y = X*2/π';
           Normierung := 8.0/Pi/Pi;
         end;
    end;

    n := 0;

    repeat
      Inc(n);
      ClearDevice;
      SetColor(White);
      Rahmen;
      SetColor(Yellow);

      for xK := xKMin to xKMax do
        begin
          x := ((xK - xKmin)/(xKMax - xKmin) * 6 - 1)*Pi;
          y := 0.0;

          case Kurve of
          Saegezahn: begin
```

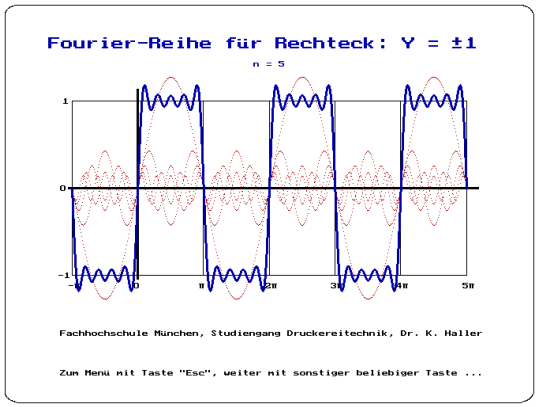
```
                        for i := 1 to n do
                          begin
                            yTemp := Sin(i*x)/i;
                            if not Odd(i)
                               then yTemp := -yTemp;
                            y  := y  + yTemp;
                            yK := y0 - Round(dy*yTemp*Normierung);
                            PutPixel(xK, yK, LightCyan);
                          end;
                        y := y*Normierung;
                      end;
          Rechteck:   begin
                        for i := 1 to n do
                          begin
                            yTemp := Sin((2*i - 1)*x)/(2*i - 1);
                            y  := y  + yTemp;
                            yK := y0 - Round(dy*yTemp*Normierung);
                            PutPixel(xK, yK, LightCyan);
                          end;
                        y := y*Normierung;
                      end;
          Dreieck:    begin
                        for i := 1 to n do
                          begin
                            yTemp := Sin((2*i - 1)*x)/Sqr(2*i - 1);
                            if not Odd(i)
                               then yTemp := -yTemp;
                            y  := y  + yTemp;
                            yK := y0 - Round(dy*yTemp*Normierung);
                            PutPixel(xK, yK, LightCyan);
                          end;
                        y := y*Normierung;
                      end;
          end;

          yK := y0 - Round(dy*y);
          SetLineStyle(SolidLn, 0, ThickWidth);
          if xK = xKMin
             then MoveTo(xK, yK)
             else LineTo(xK, yK);
          SetLineStyle(SolidLn, 0, NormWidth);
        end;

      repeat
      until KeyPressed;

    until ReadKey = #27;

  until Ch = '0';

  CloseGraph;
end.
```



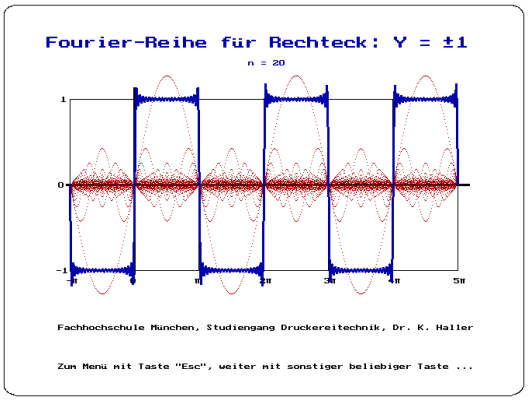
Sägezahn: $n = 5$



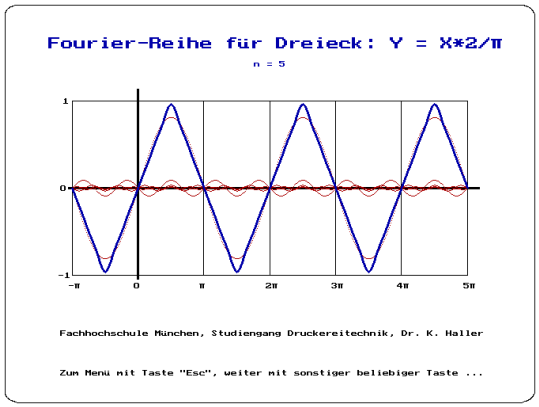
Sägezahn: $n = 20$



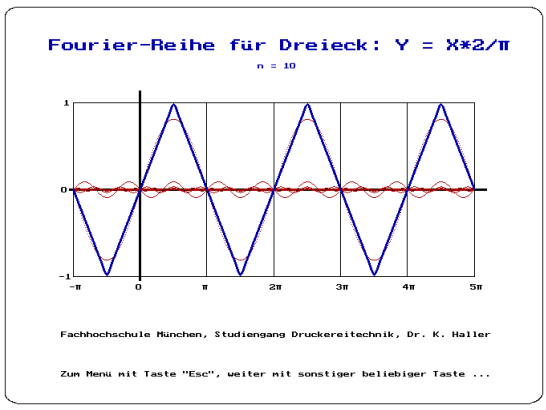
Rechteck: $n = 5$



Rechteck: $n = 20$



Dreieck: $n = 5$



Dreieck: $n = 10$

21.8 Demo-Programm Lissajous-Figuren

```
program Pas21091;   { "Pas21091.PAS", Dr. K. Haller, FHM }

uses                { Lissajous-Figuren entstehen durch Überlagerung }
  CRT, GRAPH;       { von zwei Sinusschwingungen mit verschiedenen   }
                    { Frequenzen, Phasenlagen und Amplituden.        }
                    { Zusammengesetzte Teile von Lissajous-Figuren   }
                    { könnten auch zur Umrißbeschreibung (Outline-   }
                    { Codierung) von Schriftzeichen benutzt werden.  }
                    { Siehe auch Kap. "Bézier-Funktionen".           }
                    { Einfachstes Beispiel für Lissajous-Figur:      }
                    { x = sin(t), y = sin(m*t + Phi)                 }

const
  k       = 150;    { Für Kantenlänge 0...1              }
  dt      = 0.01;   { "dt = Delta-t", steuert Auflösung  }
  Pause1  = 1.0;    { Bei Pentium 90 MHz z.B. "1.0"      }

var
  Grafiktreiber,
  Grafikmodus,
  x0, y0:        Integer;
  Pause:         Word;
  t, m, Phi,
  x, y:          Real;
  Ch:            Char;
  DemoNr:        Byte;
  DemoNrStr:     string[3];
  mStr, PhiStr:  string[10];

begin
  Grafiktreiber := Detect;
  InitGraph(Grafiktreiber, Grafikmodus, 'C:\BP\BGI');

  x0 := GetMaxX div 2;
  y0 := GetMaxY div 2 + 20;

  repeat
    SetColor(White);
    ClearViewport;
    OutTextXY(230, 100, 'Demo Lissajous-Figuren');
    OutTextXY(230, 110, 'Dr. K. Haller, FHM, DR');
    OutTextXY(230, 120, '----------------------');
    OutTextXY(230, 130, '1   Verzögerte Ausgabe');
    OutTextXY(230, 140, '2   Schnelle Ausgabe  ');
    OutTextXY(230, 150, 'Esc Beenden           ');
    OutTextXY(230, 160, '----------------------');
    OutTextXY(230, 170, '1');
    repeat
      Ch := ReadKey;
      if Ch = #13 then Ch := '1';
    until Ch in ['1'..'2', #27];
    case Ch of
    '1': Pause := Round(Pause1);  { langsam }
    '2': Pause := 0;              { schnell }
    #27: Halt;  { >>>>>>> Abbruch >>>>>>>>> }
    end;

    DemoNr := 0;
```

```
    repeat
      DemoNr := 1 + (DemoNr mod 11);
      case DemoNr of
        1: begin m := 1.0;          Phi :=  0.0;          end;
        2: begin m := 1.0;          Phi :=  0.4711;       end;
        3: begin m := 1.0;          Phi :=  Pi/2;         end;
        4: begin m := 2.0;          Phi :=  0.0;          end;
        5: begin m := 0.5;          Phi :=  0.0;          end;
        6: begin m := 0.5;          Phi :=  Pi/4;         end;
        7: begin m := 0.5;          Phi := -Pi/4;         end;
        8: begin m := 2.0;          Phi :=  Pi/4;         end;
        9: begin m := 1.5;          Phi :=  0.0;          end;
       10: begin m := Exp(0.4711); Phi :=  Ln( 0.4711); end;
       11: begin m := -Ln(0.4711); Phi :=  Exp(0.4711); end;
      end;

      SetColor(White);
      ClearViewport;
      OutTextXY(x0 -  65, y0 - k - 65, '  Lissajous-Figuren  ');
      OutTextXY(x0 -  65, y0 - k - 45, 'Einfachstes Beispiel:');
      OutTextXY(x0 - 115, y0 - k - 30,
                'x = sin(t),  y = sin(m*t + Phi)');
      OutTextXY(x0 - 190, y0 + k + 30,
                'Zum Menü mit »Esc«, ' +
                'weiter mit beliebiger Taste ... ');
      Str(DemoNr, DemoNrStr);
      OutTextXY(x0 + k + 15, y0 - k,      'Demo-Nr ' + DemoNrStr);
      Str(m:6:4, mStr);
      OutTextXY(x0 + k + 15, y0 - k + 18, 'm = ' + mStr);
      Str(Phi:6:4, PhiStr);
      OutTextXY(x0 + k + 15, y0 - k + 30, 'F = ' + PhiStr);
      MoveTo(x0 - k, y0 - k);
      LineRel(0,  2*k);  LineRel( 2*k,  0);
      LineRel(0, -2*k);  LineRel(-2*k,  0);
      Moveto(x0 - k - 10, y0);  LineRel(2*k + 20, 0);
      Moveto(x0, y0 - k - 10);  LineRel(0, 2*k + 20);
      OutTextXY(x0 - 3,     y0 - 3,  '0');
      OutTextXY(x0 + k + 2, y0 + 6,  '1');
      OutTextXY(x0 - 10,    y0 - k - 10, '1');
      SetColor(Yellow);
      Moveto(x0, y0 - Round(k*Sin(Phi)));  { (x, y) bei "t = 0.0" }
      t := 0.0;
      repeat            { Der Kern des Programms ... }
        t := t + dt;
        x := Sin(t);
        y := Sin(m*t + Phi);
        Lineto(x0 + Round(k*x), y0 - Round(k*y));
        Delay(Pause);   { Ggf. eine kleine Pause zum Betrachten }
      until KeyPressed;
      if DemoNr = 11 then Write(#7);
    until ReadKey = #27;
  until Ch = #27; { Schleife wird aber früher beendet }
end.
```