Seminar Assemblerprogrammierung
Interpretation von TrueType-Schriftarten

Matthias Köppe

November 1996

Zusammenfassung

TrueType-Schriftarten beschreiben die einzelnen Zeichen sind durch ihren Umriß und ein Interpreter-Programm. Durch Skalierung und Grid-Fitting erhält man für jedes Zeichen eine Folge von verallgemeinerten Polygonen. Es wird die Implementation eines Scan-Converters für diese Objekte in i386-Assemblercode beschrieben.

1 Von der TTF-Datei zur skalierten Outline

TrueType-Schriftarten liegen in einem systemunabhängigen Dateiformat (`.TTF`) vor. Es sind verschiedene Datenstrukturen definiert, die über symbolische Bezeichner (Tags) identifiziert werden. Die folgende Tabelle gibt einen Überblick über die wichtigsten Datenstrukturen.

Datenstrukturen in TTF-Dateien	
Bezeichner	Inhalt
`cmap`	Character to Glyph Index Mapping Table
`loca`	Index to Location Mapping Table
`glyf`	Glyph Data
`hhea`	Horizontal Header
`OS/2`	Windows Metrics
`name`	Naming Table

Durch Lookup in den Tabellen `cmap` und `loca` erhält man zu jedem Zeichen ein Byte-Offset in der `glyf`-Struktur. Dort befindet sich eine gepackte Beschreibung der Stützpunkte des Zeichenumrisses und ein Byte-Stream, der ein Interpreter-Programm darstellt. Die Stützpunkte liegen in einem 2.14-Festkommaformat vor und müssen auf die gewünschte Zeichengröße skaliert werden. Die Anweisungen in dem Interpreter-Programm dienen dem sogenannten Grid-Fitting, d. h. der Verzerrung des skalierten Umrisses, um bestimmte Merkmale, etwa gleichmäßige Strichdicken, sicherzustellen.

Nach Skalierung und Gridfitting wird der Umriß eines Zeichens durch eine Folge von verallgemeinerten Polygonen beschrieben. Jedes dieser Polygone liegt als `TPolygonHeader`-Datenstruktur, gefolgt von einer oder mehrerer `TTTPolyCurve`-Strukturen, vor.

`TPolygonHeader`-Datenstruktur		
Typ	Bezeichner	Inhalt
`DWORD`	`cb`	Größe der Struktur in Bytes
`DWORD`	`dwType`	Polygontyp (muß `tt_PolygonType` sein)
`TPOINTFX`	`pfxStart`	Ankerpunkt in 16.16-Festkommakoordinaten

Eine TTTPolyCurve-Datenstruktur repräsentiert dabei einen Linienzug oder eine quadratische B-Spline. Im Fall der quadratischen B-Spline sind die Stützpunkte abwechselnd sog. On-curve- und Off-curve-Punkte, die eine Kurve stückweise so definieren, daß sie die On-Curve-Punkte interpoliert.

TTTPolyCurve-Datenstruktur		
WORD	wType	tt_Prim_Line oder tt_Prim_QSpline
WORD	cpfx	Anzahl der Stützpunkte
cpfx DUP (TPOINTFX)	apfx	Stützpunkte in 16.16-Festkommakoordinaten

Diese Datenstrukturen sind im Handbuch zu Microsoft Windows beschrieben. Mit der Funktion GetGlyphOutline() läßt sich zu einem Zeichen ein Datenblock mit TPolygonHeader- und TTTPolyCurve-Strukturen ermitteln. Dadurch war es möglich gewesen, die Scan Conversion zu implementieren, bevor die aufwendige Interpretation der TTF-Dateien fertiggestellt war.

2 Von der skalierten Outline zur Schnittstellenstruktur

Die Scan Conversion eines ausgefüllten Polygons läßt sich am einfachsten durch die Bestimmung der Schnittstellen des Randes mit den einzelnen Scan-Zeilen realisieren. In unserem Fall liegt ein verallgemeinertes Polygon mit Festkommakoordinaten vor. Die quadratischen Spline-Kurven lassen sich aber (durch Berechnung geeignet vieler Zwischenpunkte) gleichmäßig durch Polygonzüge approximieren.

2.1 Implementation von Festkomma-Arithmetik

Festkommazahlen mit dem Layout 16.16 können in DWORD-Registern gespeichert werden. Bei Addition und Subtraktion können die gewöhnlichen Prozessorbefehle für DWORD-Zahlen genutzt werden. Multiplikation und Division (mit Runden) werden von den folgenden Makros ausgeführt.

```
        ; mulf returns eax,fixed * Factor,fixed in eax,fixed

mulf            MACRO Factor
                LOCAL   @noround
                imul    Factor
                shrd    eax, edx, 16
                jnc     SHORT @noround
                inc     eax
@noround:       ENDM

        ; divf returns eax,fixed / Divisor,fixed in eax, fixed

divf            MACRO Divisor
                mov     edx, eax
                sar     edx, 16
                sal     eax, 16
                idiv    Divisor
                ENDM
```

Das folgende Makro rundet eine 16.16-Festkommazahl.

```
        ; roundf rounds a fixed number.
```

```
roundf          MACRO   Value
                LOCAL   @NoRound
                sar     Value, 16
                jnc     SHORT @NoRound
                inc     Value
@NoRound:       ENDM
```

2.2 Implementation des Algorithmus von de Casteljau

Der Algorithmus von de Casteljau ermittelt einen Punkt auf der B-Spline. Der Wert $u \in [0,1]$ gibt die zu berechnende Stelle auf der Kurve zwischen zwei On-Curve-Punkten an. Mit `Buffer` wird ein Feld von `TPOINTFX`-Strukturen bezeichnet, das die drei lokalen Stützpunkte enthält:

```
;
; Casteljau: Apply the de Casteljau algorithm
;
; expects Buffer filled
;
; in    ebx     u, fixed
;       ebp     1-u, fixed
;       si      Buffer offset
;
; returns calculated point in QWORD Buffer[0]
;

PROC    Casteljau NEAR

        mov     cx, Degree
@@1:
        mov     di, si
        push    cx
@@2:
        mov     eax, [DWORD di]
        mulf    ebp
        mov     [DWORD di], eax
        mov     eax, [DWORD di+QWORD]
        mulf    ebx
        add     [DWORD di], eax
        add     di, QWORD
        loop    @@2

        pop     cx
        loop    @@1

        ret

ENDP
```

Die folgende Prozedur approximiert ein Segment einer B-Spline durch einen Polygonzug. Die Variable `SplineDense` gibt die Anzahl der Zwischenstellen an. Sie sollte der Skalierung angepaßt werden.

```
;
; uLoop: Loop u in the range [0, 1]
```

```
;
; expects Prep filled
;

PROC    uLoop   NEAR
        LOCAL   ru: DWORD, left: WORD, temp: DWORD = LocalSize
        enter   LocalSize, 0
        mov     ax, [SplineDense]
        or      ax, ax
        jz      SHORT @@2

        mov     [Left], ax
        mov     [ru], 0
@@1:

; Copy Prep to Buffer

        mov     ax, ds
        mov     es, ax
        mov     cx, (Degree + 1) * 2
        cld
        mov     si, OFFSET Prep
        mov     di, OFFSET Buffer
        rep     movsd

; Forward ru

        mov     eax, [SplineDeltaU]
        add     [ru], eax
        mov     ebx, [ru]

; Calc one point

        push    bp
        mov     ebp, 10000H
        sub     ebp, ebx
        mov     si, OFFSET Buffer
        call    Casteljau
        add     si, DWORD               ; -> y
        call    Casteljau
        sub     si, DWORD               ; -> x
        pop     bp

        mov     eax, [DWORD si]         ; EAX x2.fixed
        mov     ecx, [DWORD si + DWORD] ; ECX y2.fixed
        call    LineTo

        dec     [Left]
        jnz     @@1

@@2:
        leave
        ret
ENDP
```

```
;
; Fill the calculation buffer
;
; in    es:si   Current coordinates

PROC    Fill2   NEAR            ; es:si current coord

        movq    Prep+1*QWORD, es:si
        addq    es:si+1*QWORD
        sarq
        storq   Prep+2*QWORD
        push    edx eax
        push    es si
        call    uLoop
        pop     si es
        pop     [DWORD Prep]
        pop     [DWORD Prep + DWORD]
        ret

ENDP

PROC    Fill2L  NEAR

        movq    Prep+1*QWORD, es:si
        movq    Prep+2*QWORD, es:si+1*QWORD
        push    es si
        call    uLoop
        pop     si es
        ret

ENDP

;
; Public Spline Procedure
;

PUBLIC  Spline

PROC    Spline  NEAR
        ARG     Count: WORD = ArgSize RETURNS CoordPtr: DWORD, Last: QWORD

        push    bp
        mov     bp, sp

        les     si, [CoordPtr]
        movq    Prep, Last
        sub     [Count], 2
        jz      SHORT @@2
@@1:
        call    Fill2
        add     si, QWORD
        dec     [Count]
        jnz     @@1
@@2:
```

```
        call    Fill2L
        movq    Last, Prep+degree*QWORD
        add     si, 2*QWORD
        mov     [WORD CoordPtr], si
        pop     bp
        ret     ArgSize

ENDP
```

2.3 Behandlung der Polygonstrukturen

Um die bekannten Special Cases bei Polygonfüllalgorithmen auszuschalten, erlauben wir den y-Koordinaten nicht, die Scanzeilen genau zu treffen. Dies erledigt das folgende Makro.

```
; DeAlign macro de-aligns a fixed number, adding 1/65536 to integer values.
;

DeAlign         MACRO   Fract
                LOCAL   @@1
                or      Fract, Fract
                jnz     SHORT @@1
                add     Fract, 32
@@1:
                ENDM
```

Die Prozedur `OutPolygon` bearbeitet einen Block von Polygonstrukturen und verwendet die `Spline`- und `DrawLine`-Prozeduren, um die verallgemeinerten Polygone zu zeichnen.

```
; OutPolygon draws the t'type polygons. Dense should be chosen appropriately
; to the glyph size.

                ; Draws the polygon.

OutPolygon      PROC NEAR
                ARG     ScanBuf: DWORD, Dense: WORD, \
                        SizeS: WORD, Struct: DWORD = ArgSize
                LOCAL   StructEnd: WORD, PolyEnd: WORD, \
                        du: DWORD, temp: WORD, PointCount: WORD, \
                        FirstPoint: QWORD = LocalSize

                enter   LocalSize, 0

                mov     ax, dense
                mov     SplineDense, ax
                call    FixedIntRec             ; 1/dense
                mov     WORD PTR du, ax
                mov     WORD PTR du + 2, dx
                mov     WORD PTR SplineDeltaU, ax
                mov     WORD PTR SplineDeltaU + 2, dx

                mov     eax, ScanBuf
                mov     TTypeScanBuf, eax
```

```
                les     di, Struct

                mov     ax, SizeS               ; buffer size
                add     ax, di
                mov     StructEnd, ax           ; store ofs struct end

@@1:            mov     ax, es:[di]             ; polygon size
                add     ax, di
                mov     PolyEnd, ax             ; store ofs polygon end

                add     di, 8
                mov     edx, es:[di].xCoord
                mov     ecx, es:[di].yCoord
                DeAlign cx
                mov     TTypeLastPoint.xCoord, edx
                mov     TTypeLastPoint.yCoord, ecx
                mov     FirstPoint.xCoord, edx
                mov     FirstPoint.yCoord, ecx
                add     di, 8

@@2:            mov     ax, es:[di]
                cmp     ax, tt_prim_line
                jz      SHORT @@4
                cmp     ax, tt_prim_qspline
                jz      @@3
                mov     di, PolyEnd             ; unknown -- skip polygon
@@2a:
                push    es di
                mov     eax, FirstPoint.xCoord
                mov     ecx, FirstPoint.yCoord
                call    LineTo
                pop     di es
                cmp     di, StructEnd
                jnz     @@1                     ; next polygon
                jmp     @@6                     ; glyph drawn

        ; line

@@4:            mov     cx, es:[di+2]           ; point count
                add     di, 4

@@4a:           push    cx
                call    DrawLine
                pop     cx
                add     di, 8
                loop    @@4a

@@5:            cmp     di, PolyEnd
                jz      @@2a
                jmp     @@2

        ; b-spline

@@3:            mov     cx, es:[di+2]           ; coeff point count
```

```
                add     di, 4
                or      cx, cx
                jz      SHORT @@5               ; zero: no curve
                cmp     cx, 1
                jz      @@4a                    ; straight line

        ; curve (de Casteljau)

                pushq   TTypeLastPoint          ; QWORD Last
                push    es di                   ; DWORD CoordPtr
                push    cx                      ; WORD  Count
                call    Spline
                pop     di ax                   ; DWORD CoordPtr
                add     sp, 8

                jmp     @@5

@@6:            leave
                ret     ArgSize
OutPolygon      ENDP

        ; LineTo draws a line to (eax, ecx).

LineTo          PROC NEAR
                les     si, TTypeScanBuf                ; ES:SI scan buffer
                DeAlign cx
                mov     edx, ecx
                mov     ebx, eax
                xchg    edx, TTypeLastPoint.yCoord      ; EDX y1.fixed
                xchg    ebx, TTypeLastPoint.xCoord      ; EBX x1.fixed
                jmp     ScanDrawLine
LineTo          ENDP

        ; DrawLine draws a straight line to the point at es:di

DrawLine        PROC NEAR
                push    es di
                mov     eax, es:[di].xCoord
                mov     ecx, es:[di].yCoord
                call    LineTo
                pop     di es
                ret
DrawLine        ENDP
```

2.4 Erzeugung der Schnittstellenstruktur

Es wird die folgende Struktur verwendet, um die Schnittstellen mit den Scanzeilen zu verwalten.

TScanHeader-Struktur		
Typ	Bezeichner	Inhalt
WORD	FreeEntry	Offset auf die nächste freie Position
WORD	TopClip, ...	Metrische Daten
WORD	Bitmap	Generischer Zeiger auf eine Bitmap-Struktur
WORD	DoLine	NEAR-Zeiger auf eine Linienfunktion

Die `TScanHeader`-Struktur wird von einer Liste von `TScanEntry`-Strukturen gefolgt. Die ersten dieser Strukturen werden über die Nummer der Scanzeile indiziert und bilden die Köpfe von verketteten Listen von Schnittstellen.

`TScanEntry`-Struktur		
Typ	Bezeichner	Inhalt
`WORD`	`XValue`	X-Koordinate der Schnittstelle Bit 15: Typ der Schnittstelle (on/off)
`WORD`	`Offs`	`NEAR`-Zeiger auf die nächste Schnittstelle

Die folgende Prozedur fügt eine Schnittstelle in die passende verkettete Liste ein.

```
        ; Insert an entry
        ;
        ; AX    x value
        ; BX    y value                         -- is kept
        ; CX    Vertical direction +1 / -1      -- is kept
        ; ES:SI ScanHeader plus ScanEntry stream

scanInsert      PROC NEAR
                cmp     bx, es:[si].TScanHeader.TopClip
                jge     SHORT @@0
                sub     bx, es:[si].TScanHeader.BottomClip
                jl      SHORT @@0
                cld
                add     ax, es:[si].TScanHeader.LeftAdd
                mov     di, cx
                and     di, 8000H
                or      ax, di
                mov     di, es:[si].TScanHeader.FreeEntry
                mov     es:[di].TScanHeader.XValue, ax
                shl     bx, 2
                mov     ax, di
                xchg    ax, es:[bx][si+TYPE TScanHeader].TScanEntry.Offs
                mov     es:[di].TScanEntry.Offs, ax
                add     di, 4
                mov     es:[si].TScanHeader.FreeEntry, di
                shr     bx, 2
@@0:            retn
scanInsert      ENDP
```

Der folgende Code trägt eine Linie in die Schnittstellenstruktur ein, indem die Schnittstellen mit den Scanzeilen berechnet werden und `scanInsert` aufgerufen wird. Die Implementationsdetails sollen hier nicht erklärt werden.

```
        ; Draw a line of fixed-coord points and update ScanEntry stream
        ;
        ; ES:SI ScanHeader plus ScanEntry stream
        ; EBX   x1.fixed
        ; EAX   x2.fixed
        ; EDX   y1.fixed
        ; ECX   y2.fixed

scanDrawLine    PROC NEAR
                LOCAL   deltaX: DWORD, deltaY: DWORD, Delta: DWORD, \
                        Dir: WORD, CurX: DWORD, Count: WORD = LocalSize
```

```
                enter   LocalSize, 0

                call    SetupLine
                mov     eax, ebx                ; EAX x1
                push    dx ecx          ; +++
                call    NextLine
                mov     CurX, eax
                sar     edx, 16
                mov     Count, dx
                pop     edx             ;  --   y2
                call    NextLine
                sar     edx, 16
                mov     bx, Count               ; BX y start
                sub     dx, bx
                jz      SHORT @@1

        ; normal case: line count > 0

                iabs    dx
                mov     Count, dx
                pop     ax              ; -
                mov     eax, CurX
                call    AddRound
                mov     cx, Dir
@@2:
                mov     CurX, eax
                sar     eax, 16
                push    bx              ; +
                call    scanInsert
                pop     bx              ; -
                mov     eax, CurX
                add     eax, Delta
                add     bx, cx                  ; BX := BX + Dir
                dec     Count
                jnz     @@2

@@0:            leave
                ret

        ; special case: line count = 0

@@1:            pop     ax              ; -
                or      ax, ax
                jnz     @@0                     ; not on scan line

        ; on scan line

                cmp     DeltaX, 0
                mov     cx, -1                  ; on transition
                jl      SHORT @@3
                mov     cx, +1                  ; x2 < x1: off transition
@@3:
                mov     eax, CurX
                call    AddRound
```

```
                sar     eax, 16
                call    scanInsert

                leave
                ret
scanDrawLine    ENDP

        ; SetupLine calcs values on the stack frame
        ;
        ; IN    ebx     x1.fixed        OUT     deltaX  x2 - x1 (signed)
        ;       edx     y1.fixed                Dir     Y increment (integer)
        ;       eax     x2.fixed
        ;       ecx     y2.fixed                deltaY  y2 - y1 (signed)
        ;

SetupLine       PROC    NEAR

                cmp     ecx, edx
                mov     Dir, +1
                jge     SHORT @@3

                mov     Dir, -1
@@3:
                push    eax ecx
                sub     eax, ebx                ; EAX (delta x).fixed
                mov     deltaX, eax
                sub     ecx, edx                ; ECX (delta y).fixed
                mov     deltaY, ecx
                pop     ecx eax

                retn

SetupLine       ENDP

        ; AddRound adds 7FFFH (on) or 8000H (off) to EAX
        ;

AddRound        PROC NEAR

                cmp     Dir, +1                 ; on ?
                jnz     SHORT @@1
                add     eax, 7FFFh
                retn
@@1:
                add     eax, 8000h
                retn

AddRound        ENDP

        ; CalcDelta calcs the line Delta = deltaX / |deltaY|
        ;
        ; no calc if |deltaY| < 1.0, CF set
        ; OUT   ecx     |deltaY|
        ;       eax     Delta on CF clear, saved on CF set
```

```
        ;

CalcDelta       PROC NEAR
                mov     ecx, deltaY
                iabs    ecx
                cmp     ecx, 10000h
                jb      SHORT @@0

                mov     eax, deltaX
                divf    ecx
                mov     Delta, eax
                clc
@@0:
                retn
CalcDelta       ENDP

        ; MultDelta mults eax,fixed * (dx / |dy|)
        ;

MultDelta       PROC NEAR
                push    eax             ; ++
                call    CalcDelta
                jc      SHORT @@1

        ; Delta calculated: mult the former eax

                pop     ecx             ; --
                mulf    ecx
                retn

        ; Delta not calced: calc (eax * dx) / |dy|
@@1:
                pop     eax             ; --

        ; dy = 0: return 0
@@2:
                xor     eax, eax
                retn
MultDelta       ENDP

        ; NextLine skips to the next scan line, according to direction.
        ; if on scan line, no move.
        ;
        ; IN    eax     X,fixed         OUT     eax     X,fixed
        ;       edx     Y,fixed                 edx     Y,fixed

NextLine        PROC NEAR

                or      dx, dx                  ; frac(y) = 0 ?
                jz      SHORT @@0

                push    edx eax         ; ++++
                xor     eax, eax
                mov     ax, dx                  ; eax frac(y)
```

```
                cmp     Dir, +1
                jz      SHORT @@1

        ; down: add frac * Delta

                call    MultDelta
                pop     ecx             ;   --
                add     eax, ecx
                pop     edx             ; --
                xor     dx, dx
                retn

        ; up: add (1-frac) * Delta

@@1:
                neg     eax
                add     eax, 10000H
                call    MultDelta
                pop     ecx             ;   --
                add     eax, ecx
                pop     edx             ; --
                add     edx, 10000H
                xor     dx, dx
                retn

        ; no move: calc Delta

@@0:
                push    edx eax         ; ++++
                call    CalcDelta
                pop     eax edx         ; ----
                retn

NextLine        ENDP
```

3 Von der Schnittstellenstruktur zur Bitmap

Sobald die Schnittstellenstruktur für ein Zeichen vorliegt, genügt die Implementation einer Funktion, die eine horizontale Linie auf dem gewünschten Gerät zeichnet. Eine solche Funktion muß die folgende Argumentliste haben:

```
        ; DoLine specification

;               ARG     y: WORD, x2: WORD, x1: WORD, ABitmap: DWORD
```

Ein Zeiger auf diese Funktion wird in das `DoLine`-Feld der `TScanHeader`-Struktur eingetragen. Beim Aufruf durch die weiter unten vorgestellten Prozeduren wird der `Bitmap`-Zeiger aus `TScanHeader` als `ABitmap` übergeben.

Die folgende Prozedur findet die am weitesten links liegende Schnittstelle in einer vorgegebenen Scanzeile und entfernt sie aus der Schnittstellenstruktur.

```
        ; Find minimal entry. Remove this entry.
        ;
```

```
        ; In:   ES:SI   start of line
        ; Out:  CF      set on EOL
        ;       ZF      transition type (Z on, NZ off)
        ;       CX      minimum co-ordinate
        ;       Regs    destroyed

scanFindMin     PROC NEAR
                mov     cx, 7FFFh                       ; current minimum
                mov     bx, 0                           ; before min entry
                jmp     SHORT @@1
@@2:
                mov     ax, es:[si].TScanEntry.XValue
                push    ax cx
                and     ah, 7Fh
                and     ch, 7Fh
                cmp     ax, cx
                pop     cx ax
                jg      SHORT @@1

                jl      SHORT @@3

                test    ah, 80h
                jz      @@1
@@3:
                mov     bx, dx
                mov     cx, ax
@@1:
                mov     dx, si                          ; before entry
                mov     si, es:[si].TScanEntry.Offs
                or      si, si
                jne     @@2
                or      bx, bx
                stc
                jz      SHORT @@0
                and     cx, 7FFFH
                mov     si, es:[bx].TScanEntry.Offs     ; SI offset min entry
                test    es:[si].TScanEntry.XValue, 8000H
                mov     si, es:[si].TScanEntry.Offs     ; SI offset next entry
                mov     es:[bx].TScanEntry.Offs, si
@@0:
                ret
scanFindMin     ENDP
```

Durch wiederholte Anwendung dieser Prozedur wird das gesamte Zeichen mittels horizontaler Linienzüge ausgegeben.

```
scanConvBufP    PROC NEAR
                ARG     Buffer: DWORD = ArgSize
                LOCAL   Winding: WORD, Current: WORD, ABitmap: DWORD, \
                        LineProc: WORD, ATop: WORD = LocalSize
                enter   LocalSize, 0
                les     si, Buffer
                mov     eax, es:[si].TScanHeader.Bitmap
                mov     ABitmap, eax
                mov     ax, es:[si].TScanHeader.DoLine
```

```
                mov     LineProc, ax
                mov     ax, es:[si].TScanHeader.TopBearing
                mov     ATop, ax
                mov     ax, es:[si].TScanHeader.BottomClip
                mov     cx, es:[si].TScanHeader.TopClip
                sub     cx, ax                          ; count
                add     si, TYPE TScanHeader

@@1:
                push    si cx
                mov     Winding, 0

        ; Winding is zero > read entry

@@6:
                call    scanFindMin
                jc      SHORT @@5

                addz    Winding
                mov     Current, cx

                pop     cx si
                push    si cx

        ; Winding is nonzero > wait for zero

@@2:
                push    si
                call    scanFindMin
                pop     si
                jc      SHORT @@5

                addz    Winding
                jnz     @@2

                pop     ax si
                push    si ax
                dec     cx                      ; off position
                cmp     cx, Current
                jl      @@6

                push    es
                pusha

                add     ax, ATop
                push    ABitmap
                push    Current cx ax
                call    LineProc

                popa
                pop     es
                jmp     @@6
@@5:
                pop     cx si
```

```
                add     si, TYPE TScanEntry
                loop    @@1

                leave
                ret     ArgSize
scanConvBufP    ENDP
```

4 Vollständige Implementation

Die hier gezeigten Codeausschnitte sind ohne ein geeignetes Rahmenprogramm nicht lauffähig. Eine einfache Implementation in Pascal ist verfügbar in:

`http://www-hppool.cs.uni-magdeburg.de/~mkoeppe/mkm`

Es sind die folgenden Ausgabefunktionen implementiert:

- Direkte Bildschirmausgabe in 16- und 256-Farben-Modi (VESA)
- Ausgabe als Bitmap-Schriftartendatei
- Gefilterte Ausgabe („CrystalFont“)

Nicht implementiert ist dagegen der Interpreter für Gridfitting-Programme. Dadurch ist die Schriftausgabe in kleinen Skalierungen von geringer Qualität.