- Das Format der Type-1-Schriften wurde von Adobe zusammen mit PostScript entwickelt und ist seit 1985 in Nutzung.
- Anders als bei PostScript wurde jedoch das Format der Type-1-Schriften erst 1990 veröffentlicht.
- 1994 wurde das Format als Teil 3 des ISO-Standards 9541 unter dem Titel *Glyph Shape Representation* als Standard akzeptiert.
- Trotz der Konkurrenz durch TrueType sind die Type-1-Schriften bis heute auf dem Markt präsent.
- OpenType ist die Nachfolge-Technologie, die sowohl von Adobe als auch Microsoft unterstützt wird. OpenType unterstützt dabei sowohl die Type-1- als auch die TrueType-Technologie und fügt noch einige Erweiterungen hinzu.
- Literatur: *Adobe Type 1 Format* und Yannis Haralambous: *Fonts & Encodings*, O'Reilly.

Grundsätzlich ähneln Type-1-Schriften weitgehend den Type-3-Schriften, abgesehen davon, dass

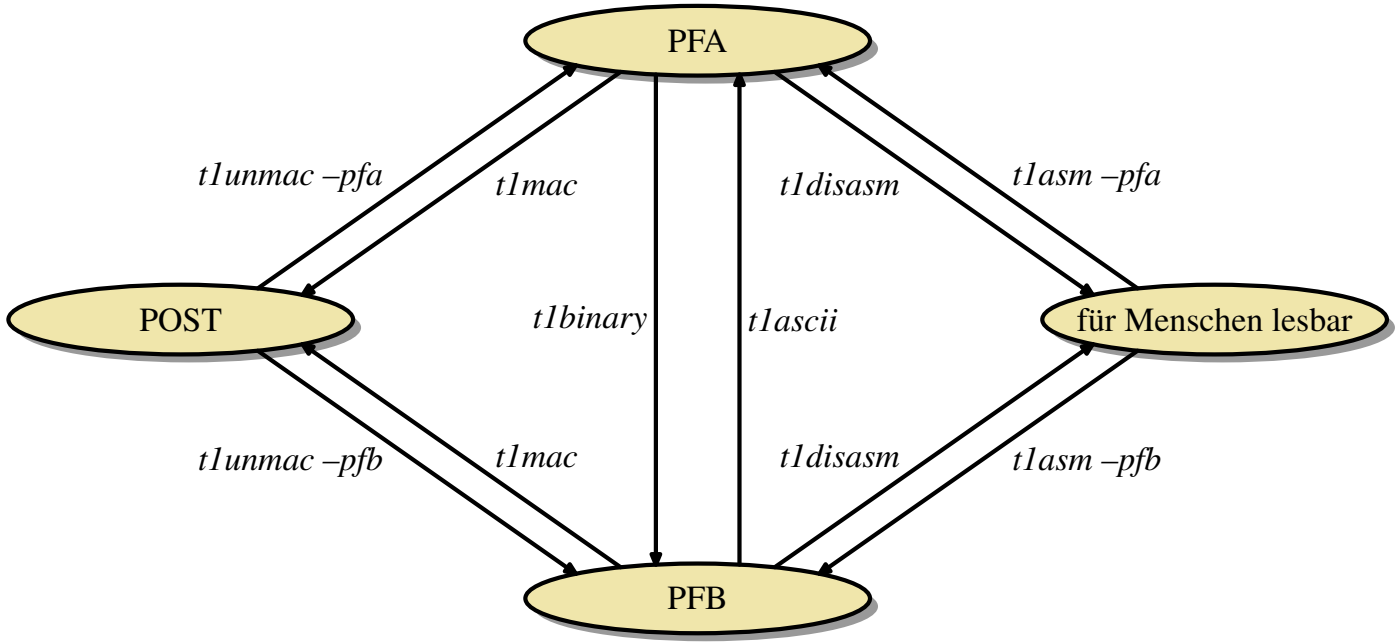
- nicht die Standard-Operatoren von PostScript zur Beschreibung der Kurven verwendet werden, dass
- zusätzlich Hinweise zur Optimierung der Rasterung integriert sind (*hints*) und dass
- das Format größtenteils binär und möglicherweise sogar verschlüsselt ist (auf Basis des *eexec*-Operators, der eine einfache Stromchiffre verwendet, deren Algorithmus und von Adobe verwendeter Schlüssel inzwischen öffentlich sind).

Zu einem Type-1-Schriftschnitt gehören folgende vier Komponenten:

- Ein öffentliches assoziatives Array, das dem eines Type-3-Schriftschnitts ähnelt und in jedem Falle als ASCII-Text abgelegt wird,
- ein privates assoziatives Array, das die Hinweise zur Rasterung aufnimmt und das nur in binärer und möglicherweise sogar nur in verschlüsselter Form vorliegt,
- Prozeduren und
- die Beschreibung der Kurven, die wiederum nur in binärer und möglicherweise sogar in verschlüsselter Form vorliegen.

- Unter Unix sind die Formate PFA (A wie ASCII) und PFB (B wie binär) üblich. Bei PFB werden die binären Teile hexadezimal dargestellt.
- Für den Macintosh gibt es das POST-Format.
- Alle drei Varianten lassen sich dank den Werkzeugen von Lee Hetherington und Claire Connelly ineinander überführen.

Konvertierungen zwischen den Formaten



- Konvertierungswerkzeuge von `http://www.lcdf.org/type/`

utmr8a.disasm

```
%!PS-AdobeFont-1.0: NimbusRomNo9L-Regu 1.05
% ...
12 dict begin
/FontName /NimbusRomNo9L-Regu def
% ... more contents of the public dictionary ...
currentdict end
currentfile eexec
dup /Private 15 dict dup begin
% ... private dictionary ...
end
readonly put
noaccess put
dup /FontName get exch definefont pop
mark currentfile closefile
cleartomark
```

- Hier und im folgenden werden Auszüge des Times-Roman-Schriftschnitts von URW++ präsentiert, der unter der GPL zur Verfügung steht. Betrachtet wird der Text, den *t1disasm* generiert hat.

Schlüssel	Typ	Bedeutung
FontType	integer	1
FontMatrix	array	Transformations-Matrix für die Kurven innerhalb der Definitionen für die Zeichen; typisch ist ein 1000 × 1000-Koordinatensystem
FontInfo	dict	assoziatives Array mit weiteren Feldern, die den Schriftschnitt beschreiben
Encoding	dict	bildet Werte aus dem Bereich $[0, 255]$ in Namen für die einzelnen Zeichen ab
FontBBox	array	Bounding-Box aller übereinander gezeichneter Zeichen
CharStrings	dict	Kurven der einzelnen Zeichen

Schlüssel	Typ	Bedeutung
FontName	name	Name des Schriftschnitts
PaintType	integer	1 (fill) oder 2 (stroke)
Private	dict	Hinweise zur Rasterung
UniqueID	integer	Eindeutige Zahl zwischen 0 und $2^{24} - 1$
XUID	array	Alternativ zu UniqueID: Eindeutige Folge von Zahlen, wobei die erste von Adobe vergeben wird

Einträge im öffentlichen assoziativen Array 417

utmr8a.disasm

```
/FontName /NimbusRomNo9L-Regu def
/PaintType 0 def
/WMode 0 def
/FontBBox {-168 -281 1000 924} readonly def
/FontType 1 def
/FontMatrix [0.001 0.0 0.0 0.001 0.0 0.0] readonly def
/Encoding StandardEncoding def
/UniqueID 5020931 def
```

- Die Einträge für `/CharStrings` und `/Private` kommen erst nachträglich hinzu.

Einträge im öffentlichen assoziativen Array 418

utmr8a.disasm

```
/FontInfo 10 dict dup begin
/version (1.05) readonly def
/Notice ((URW)++,Copyright 1999 by (URW)++ ...) readonly def
/Copyright (Copyright (URW)++ ...) readonly def
/FullName (Nimbus Roman No9 L Regular) readonly def
/FamilyName (Nimbus Roman No9 L) readonly def
/Weight (Regular) readonly def
/ItalicAngle 0.0 def
/isFixedPitch false def
/UnderlinePosition -100 def
/UnderlineThickness 50 def
end readonly def
```

- `/FontInfo` gehört mit zum öffentlichen assoziativen Array und ist selbst eines.

- Im Prinzip können die Namen der einzelnen Zeichen, die in `/Encoding` aufgenommen werden, beliebig gewählt werden.
- In Bezug auf PDF kann das ein Problem sein: Wenn interaktiv nach einem Text in einem Dokument gesucht wird, wie kann festgestellt werden, welches Zeichen im Schriftschnitt welchem Zeichen (etwa in Unicode) entspricht?
- Die Lösung besteht darin, dass die Namen der einzelnen Zeichen gewissen Konventionen entsprechen müssen.
- Das ist festgelegt über ISO 10036. Siehe `http://10036ra.org/wikka/HomePage`
- Alternativ finden sie sich auch in der Adobe Glyph List (AGL): `http://www.adobe.com/devnet/opentype/archives/glyph.html`

utmr8a.disasm

```
currentfile eexec
```

- `currentfile` ist ein Operator, der einen Verweis auf die aktuelle Eingabequelle (mit dem Type-1-Schriftschnitt) zurückliefert.
- `eexec` nimmt den Verweis auf eine Eingabedatei, um ihn die damit referenzierte Eingabe zu entschlüsseln und anschließend auszuführen.
- In einer PFA-Datei ist bis zu diesem Punkt (abgesehen von den ersten 6 Bytes) alles Klartext und erst danach beginnt das eigentliche binäre Format.

Zum assoziativen Array `/Private` gehören folgende Komponenten:

- Globale Hinweise zur Rasterung,
- mehrere historische Parameter,
- einige Operatoren, die für die Prozeduren benötigt werden,
- die Prozeduren selbst (`/Subrs`) und
- weitere Prozeduren (`/OtherSubrs`).

Zu den binären Daten gehört ferner noch `/CharStrings`, das jedoch erst nachträglich in das übergeordnete assoziative Array eingeordnet wird. Es enthält die Kurvendefinitionen der einzelnen Zeichen einschließlich individueller Hinweise zur Rasterung.

utmr8a.disasm

```
/BlueValues [-20 0 450 470 662 682] def
/BlueScale 0.039625 def
/StdHW [30] def
/StdVW [85] def
/StemSnapH [30 38 43 53 60 73] def
/StemSnapV [78 85 91 103 109 115] def
```

- `/BlueValues` spezifiziert die y-Koordinaten für die wichtigsten Höhenlinien.
- Die optionalen Parameter `/BlueScale` und `/BlueShift` spezifizieren, wie weit die Ausrichtung aufgrund der Höhenlinien erfolgt.
- `/StdHW` und `/StemSnapH` spezifizieren die wichtigsten horizontalen Strichstärken. Analog gibt es `/StdVW` und `/StemSnapV` für die vertikalen Strichstärken.

utmr8a.disasm

```
/password 5839 def
/MinFeature {16 16} def
```

- Diese Parameter sind entsprechend der Spezifikation von Adobe aus historischen Gründen notwendig.
- Eine weitere Erklärung gibt es nicht dazu. Entsprechend finden sie sich in dieser Form in jedem Type-1-Schriftschnitt.

utmr8a.disasm

```
/RD {string currentfile exch readstring pop} executeonly def
/ND {noaccess def} executeonly def
/NP {noaccess put} executeonly def
```

- Diese Operatoren müssen ebenfalls innerhalb des `/Private`-Arrays definiert werden.
- `/ND` bzw. `/NP` fügen eine Prozedur in ein assoziatives Array bzw. ein reguläres Array ein und beschränken den Zugriff.

utmr8a.disasm

```
/Subrs 251 array
dup 0 {
   3 0 callothersubr pop pop setcurrentpoint return
} NP
dup 1 {
   0 1 callothersubr return
} NP
dup 2 {
   0 2 callothersubr return
} NP
dup 3 {
   return
} NP
% ...
ND
```

- Die Prozeduren sind in einem regulären Array und somit nur über die jeweilige Nummer erreichbar.
- Die Prozeduren 0 bis 3 sind vordefiniert. Der entsprechende Text wird deswegen normalerweise ignoriert.
- Mit `callothersubr` erfolgt ein Aufruf der anderen Prozeduren. Mit `return` endet die Ausführung einer Prozedur. Ein `return` ist zwingend.

- Der Operator `hsbw` mit den beiden Parametern *sbx* und *wx* spezifiziert mit (*sbx*, 0) den am weitesten links liegenden Punkt der Kurve und den nachfolgenden Weiterbewegungsvektor mit (*wx*, 0).
- Ferner wird die aktuelle Position auf (0, *sbx*) gesetzt. Diese Position geht jedoch nicht in die Kurve selbst ein. Diese muss zwingend mit einer relativen Bewegung beginnen.
- Dies muss der erste Operator sein.
- Alternativ kann `sbw` verwendet werden, falls der Weiterbewegungsvektor eine von 0 abweichende y-Komponente hat.

Operator	Beschreibung
dx dy rmoveto	bewegt sich relativ zur aktuellen Position um (*dx*, *dy*)
dx hmoveto	ist äquivalent zu *dx* `0 rmoveto`
dy vmoveto	ist äquivalent zu `0` *dy* `rmoveto`
dx dy rlineto	Linie mit relativen Koordinaten
dx hlineto	horizontale Linie
dy vlineto	vertikale Linie
dx1 dy1 dx2 dy2 dx3 dy3 rcurveto	Bézier-Kurve, wobei alle Koordinaten relativ zum Ausgangspunkt genommen werden
dx1 dx2 dy2 dy3 hvcurveto	äquivalent zu *dx1* `0` *dx2 dy2* `0` *dy3* `rcurveto`
dy1 dx2 dy2 dx3 vhcurveto	äquivalent zu `0` *dy1 dx2 dy2 dx3* `0 rcurveto`
closepath	Ende einer Kurve; mehrere Kurvendefinitionen sind zulässig; der aktuelle Punkt bleibt bestehen
endchar	Ende einer Zeichendefinition

Operatoren für individuelle Hinweise zur Rasterung 428

Operator	Beschreibung
hstem	Spezifikation eines horizonalen Strichs mit der unteren y-Koordinate und der Höhe
vstem	Spezifikation eines vertikalen Schafts mit der niedrigeren x-Koordinate und der Breite
vstem3	adressiert die Problematik des »m«, indem mit insgesamt 6 Parametern die Angaben für drei einzelne `vstem`-Operatoren zusammengefasst werden.
hstem3	analog zu vstem3, jedoch horizontal

utmr8a.disasm

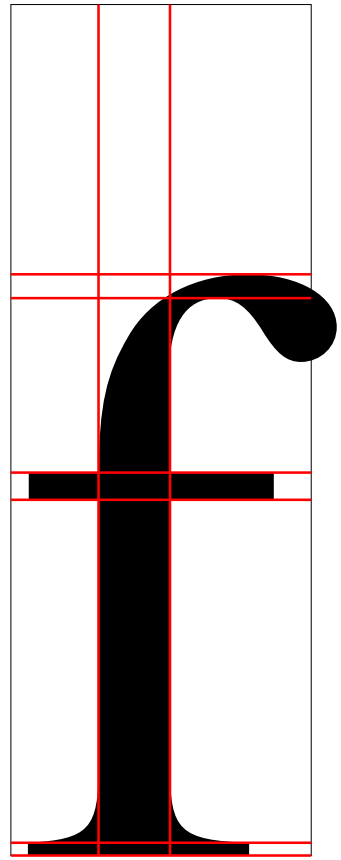
```
2 index /CharStrings 316 dict dup begin
% ...
/a {
   % ...
} ND
% ...
/.notdef {
   % ...
} ND
end
```

- Da die Definition textuell in der Deklaration von `/Private` eingebettet ist, muss mit `2 index` explizit das übergeordnete assoziative Array referenziert werden (das immer noch auf dem Stack an genannter Position liegt).
- Die einzelnen Prozeduren enthalten jeweils individuelle Hinweise zur Rasterung und die Kurvendefinition. Hierbei dürfen nur die speziellen Operatoren verwendet werden.
- Die Namen müssen mit denjenigen übereinstimmen, die in `/Encoding` referenziert werden.

Definition des Schnitts für den Buchstaben f 430

utmr8a.disasm

```
/f {
   20 333 hsbw
   0 15 hstem 418 32 hstem 655 28 hstem 83 84 vstem
   289 450 rmoveto
   -123 hlineto 116 vlineto
   58 19 31 38 vhcurveto
   21 0 14 -10 18 -29 rrcurveto
   16 -26 12 -10 17 0 rrcurveto
   23 19 18 23 hvcurveto
   36 -44 26 -60 vhcurveto
   -62 0 -53 -27 -26 -46 rrcurveto
   -26 -44 -8 -36 -1 -80 rrcurveto
   -82 hlineto -32 vlineto 82 hlineto
   -314 vlineto
   0 -73 -11 -12 -72 -4 rrcurveto
   -15 vlineto 260 hlineto 15 vlineto
   -82 3 -11 11 0 75 rrcurveto
   314 vlineto 122 hlineto
   closepath endchar
} ND
```



- Die Ursprünge der TrueType-Schriften liegen in den Bemühungen von Apple und Microsoft, unabhängig von der Schrifttechnologie von Adobe zu werden.
- Dies war insbesondere für die Rasterung von Schriften am Bildschirm relevant.
- Die ursprüngliche Fassung geht auf den finnischen Entwickler Sampo Kaasila zurück, der für Apple arbeitete.
- Apple integrierte TrueType-Schriften zuerst 1991 bei MacOS 6; Microsoft folgte 1992 bei Windows 3.1.
- Beide arbeiteten unabhängig voneinander weiter, was zu zahlreichen Inkompatibilitäten geführt hat.

- Das Dateiformat ist vollumfänglich binär, aber vollständig dokumentiert.
- Dokumentation gibt es von Apple und von Microsoft, die nicht deckungsgleich sind:
  - `http://developer.apple.com/fonts/TTRefMan/index.html`
  - `http://www.microsoft.com/typography/otspec/default.htm`

- Es gibt von Just van Rossum ein frei verfügbares Werkzeug, mit dessen Hilfe TTF-Dateien in ein XML-Format und wieder zurück überführt werden können:
  `http://sourceforge.net/projects/fonttools/`
- Unter Debian steht das Werkzeug über das Paket *fonttools* zur Verfügung.
- Dies wird im folgenden verwendet werden, um einzelne Teile einer TrueType-Datei zu betrachten und zu analysieren.

```
hochwanner$ ttx -l TimesNewRoman.ttf
Listing table info for "TimesNewRoman.ttf":
    tag      checksum   length   offset
    ----  ----------  -------  -------
    LTSH  0x814d6df0       663      316
    OS/2  0xd70a8dc7        86      980
    PCLT  0x59d381e3        54     1068
    VDMX  0x4e236882      4500     1124
    cmap  0x80a4ee32      1750     5624
    cvt   0xf3deda81      1990     7376
    fpgm  0x3775a72f      1395     9368
    gasp  0x00180009        16    10764
    glyf  0x784bf0ce    144434    10780
    hdmx  0x756f659c     15944   155216
    head  0xc17591b0        54   171160
    hhea  0x0f0308af        36   171216
    hmtx  0xb6f7a4a5      2636   171252
    kern  0xa677acd1      5220   173888
    loca  0x03143262      2640   179108
    maxp  0x08fa069b        32   181748
    name  0xf80cc1bf      4881   181780
    post  0x9141e4c3      4898   186664
    prep  0xc6afb762      3874   191564

hochwanner$
```

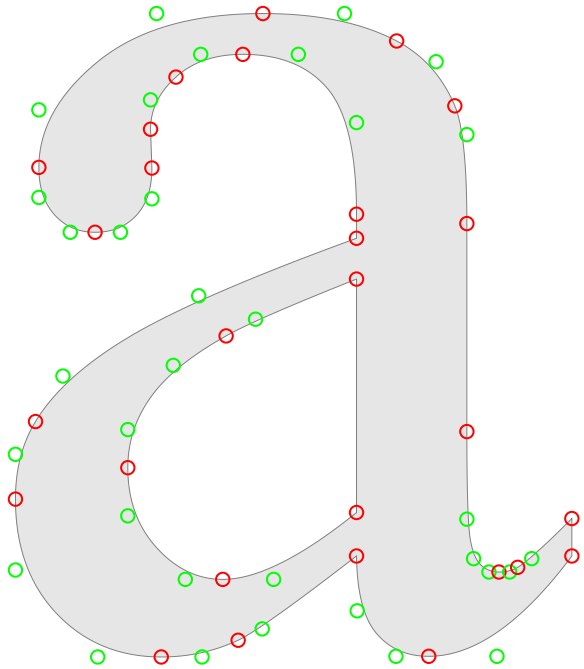
- Das TrueType-Dateiformat sieht auf der obersten Ebene ein Verzeichnis einzelner Tabellen vor. Die Tabellennamen haben dabei bis zu vier Zeichen und belegen einen zusammenhängenden Abschnitt der Datei.
- Die wichtigsten Tabellen sind *head* (globale Definitionen), *cmap* (Abbildung zwischen Zeichensätzen und den im Schriftschnitt enthaltenen Zeichen), *glyf* (Kurvendefinitionen der einzelnen Zeichen zusammen mit Instruktionen zur ihrer Rasterung), *hhea*, *OS/2* und *hmtx* (globale und individuelle Metriken für den horizontalen Satz) und *kern* (Kerning-Tabellen).

Kurvendefinition bei TrueType

TimesNewRoman.ttx

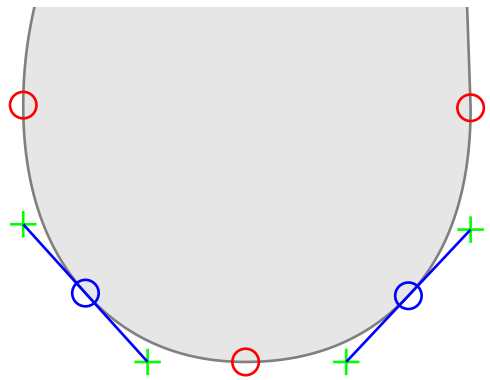
```
<TTGlyph name="a" xMin="73" yMin="-19" xMax="905" yMax="943">
  <contour>
    <pt x="583" y="132" on="1"/>
    <pt x="442" y="23" on="0"/>
    /* ... */
    <pt x="584" y="50" on="0"/>
  </contour>
  <contour>
    <pt x="583" y="197" on="1"/>
    <pt x="583" y="546" on="1"/>
    <pt x="432" y="486" on="0"/>
    <pt x="388" y="461" on="1"/>
    <pt x="309" y="417" on="0"/>
    <pt x="241" y="321" on="0"/>
    <pt x="241" y="264" on="1"/>
    <pt x="241" y="192" on="0"/>
    <pt x="327" y="97" on="0"/>
    <pt x="383" y="97" on="1"/>
    <pt x="459" y="97" on="0"/>
  </contour>
  <instructions><assembly>
      /* ... instructions for rasterization ... */
    </assembly></instructions>
</TTGlyph>
```

Kurvendefinition bei TrueType



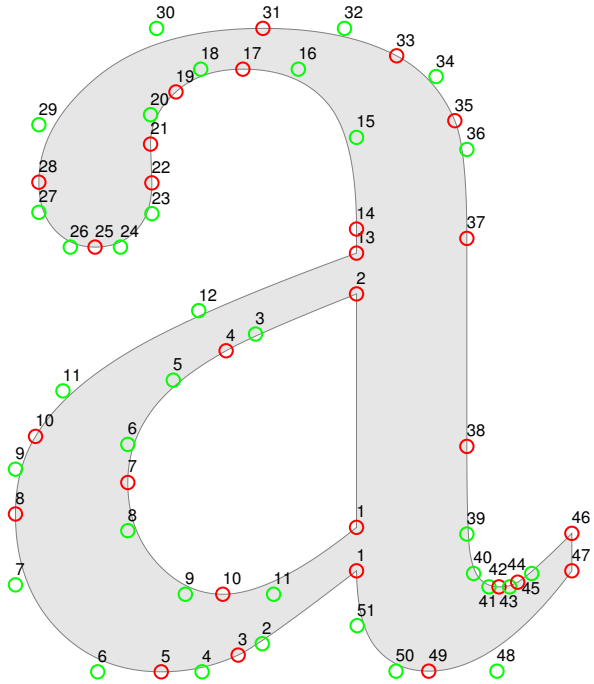
Buchstabe „a“ bei *TimesNewRoman*

- Die Kurvendefinition besteht aus beliebig vielen einzelnen jeweils geschlossenen Kurven (*contour*).
- Jede Kurvenspezifikation besteht aus beliebig vielen Punkten, die entweder auf der Kurve liegen ($on = 1$, hier rot) oder außerhalb ($on = 0$, hier grün).
- Daraus ergibt sich dann eine Folge quadratischer Bézier-Kurven und gerader Linien.

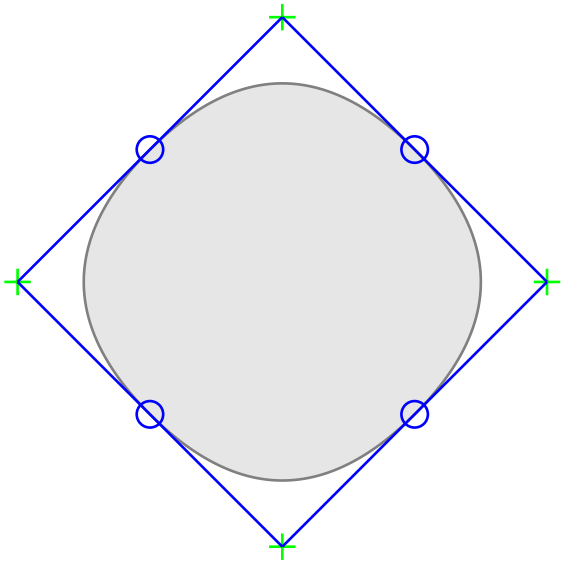


- Wenn mehrere Punkte außerhalb der zu zeichnenden Kurve aufeinanderfolgen ($on = 0$, in der Zeichnung grün), dann wird jeweils implizit genau dazwischen ein Punkt auf der Kurve gewählt (in der Zeichnung blau).
- Die jeweiligen Verbindungslinien (ebenfalls blau) zwischen zwei aufeinanderfolgenden Außenpunkten sind dann tangential zu der zu zeichnenden Kurve.

Kurvendefinition bei TrueType

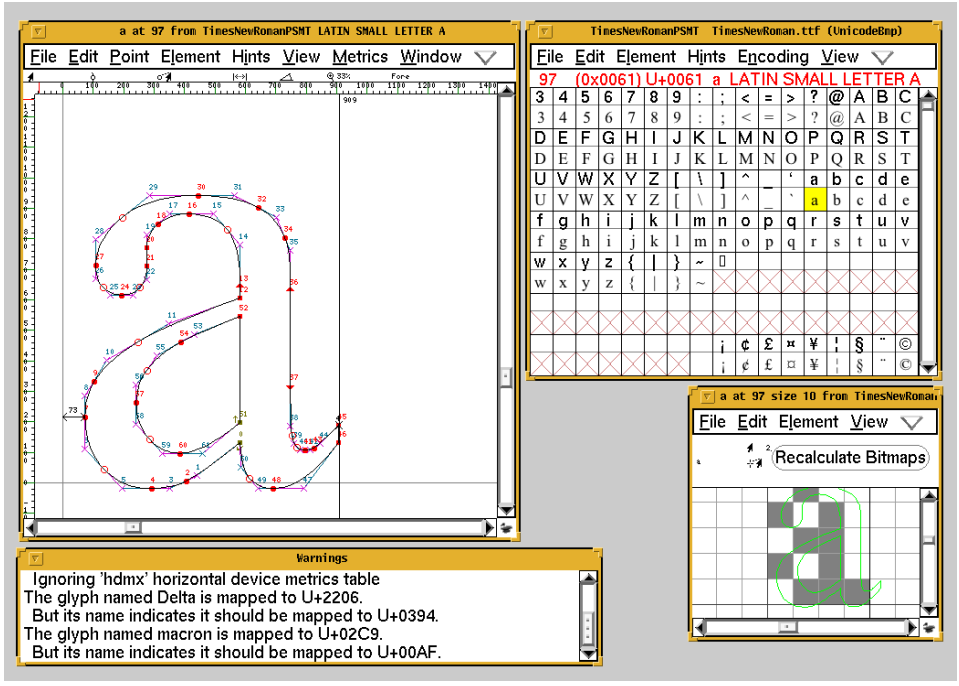


- Die einzelnen Punkte sind so aneinanderzureihen, dass die zu füllende Fläche entsprechend der gewählten Richtung immer rechts liegt.
- Außenlinien werden entsprechend im Uhrzeigersinn, Augen gegen den Uhrzeigersinn gezeichnet.
- Gefüllt wird anschließend entsprechend der *non-zero winding number rule*.

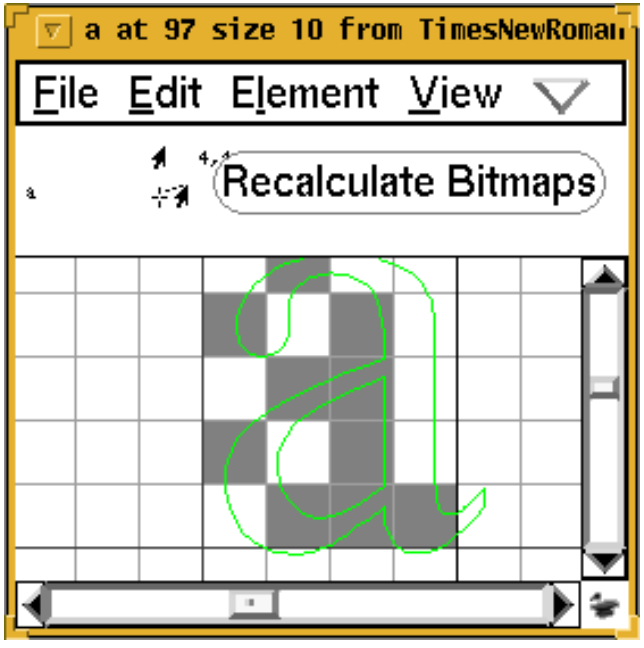


- Prinzipiell sind auch Kurvendefinitionen zulässig, bei denen alle Punkte außerhalb der Kurve liegen.
- Das Beispiel demonstriert dies an den Punkten $(200, 300)$, $(300, 400)$, $(400, 300)$ und $(300, 200)$, die ein Quadrat bilden.

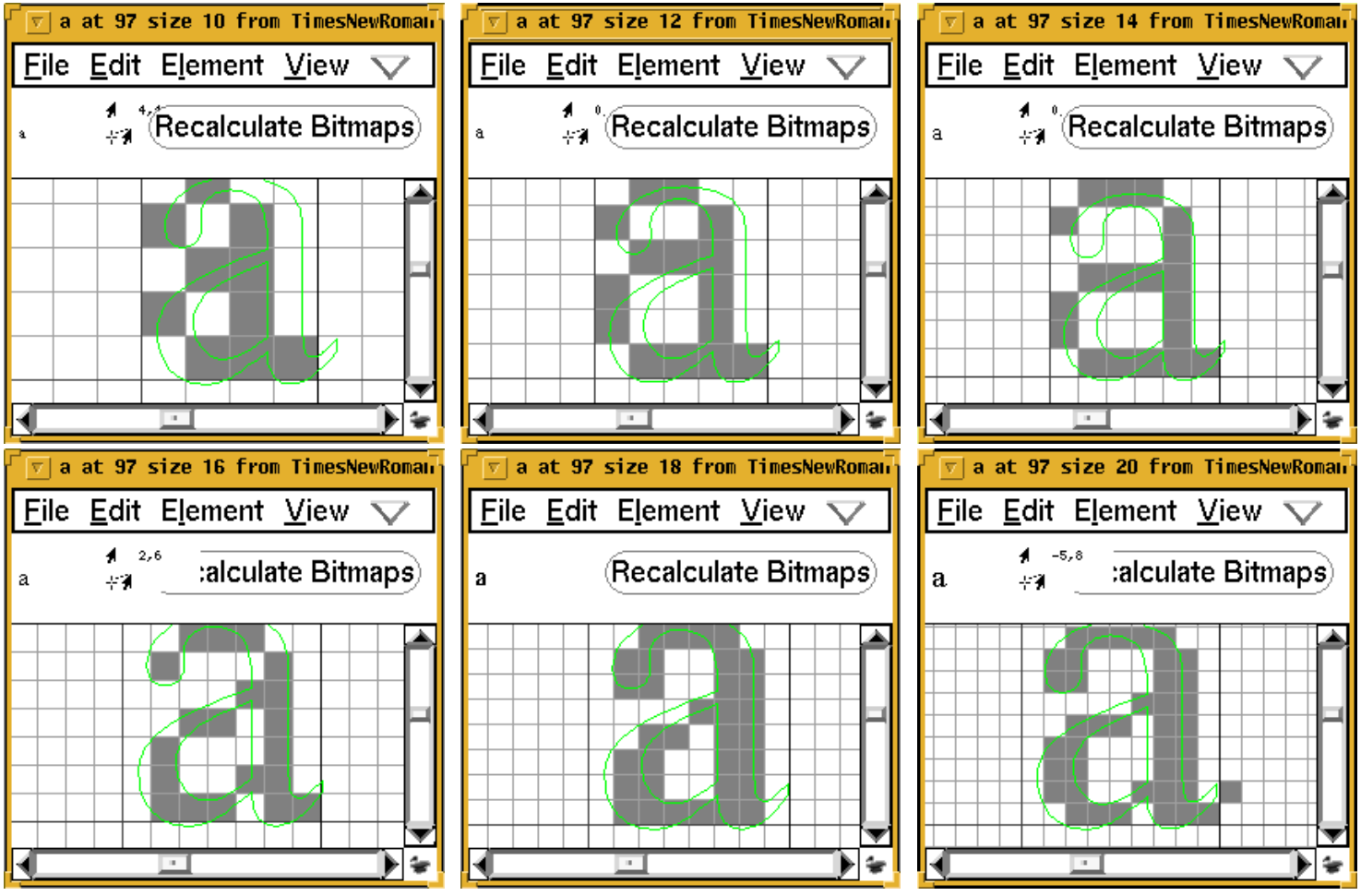
- Für die Modellierung erscheinen kubische Bézier-Kurven einfacher, da die beiden Tangenten auf sehr einfache Weise unabhängig voneinander gewählt werden können.
- Bei quadratischen Kurven sind daher mehr einzelne Bézier-Kurven notwendig.
- Die quadratischen Bézier-Kurven lassen sich jedoch sehr viel effizienter berechnen. Insbesondere vereinfacht sich die Rasterisierung.
- Auch werden unerwünschte Knicks und Scheitelpunkte bei quadratischen Kurven zuverlässig vermieden.



- Zur Visualisierung und insbesondere zum Testen der Rasterisierung empfiehlt sich der Einsatz des freien Software-Pakets *fontforge*.
- `http://fontforge.sourceforge.net/`
- Bei Debian steht es als Paket (*fontforge*) zur Verfügung und unter dem gleichen Namen gibt es das Paket auch bei den Macports.

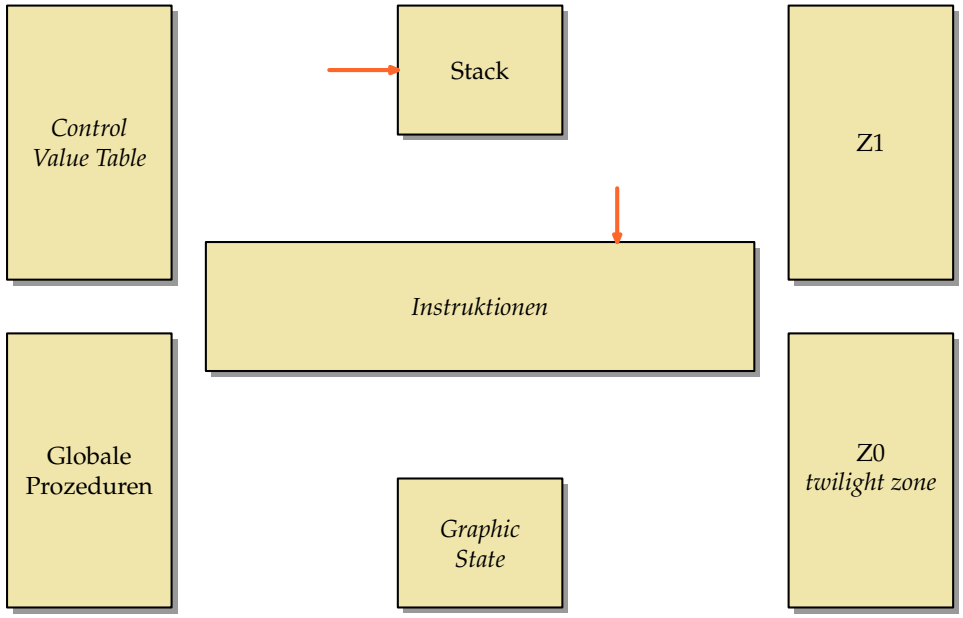


- Anders als bei Type1-Schriften ist die Rasterung bei TrueType-Schriftschnitten frei programmierbar.
- Das Programm kann in Abhängigkeit von der Rasterung sämtliche Punkte frei verschieben und auch ganze Teile (wie etwa Serifen) zum Verschwinden bringen, wenn die Auflösung zu gering ist.
- In dem gezeigten Beispiel ist das „a“ trotz der Rasterung auf 4×5 Felder noch klar zu erkennen.


a at 97 size 10 from TimesNewRoman
File Edit Element View
Recalculate Bitmaps
a at 97 size 12 from TimesNewRoman
File Edit Element View
Recalculate Bitmaps
a at 97 size 14 from TimesNewRoman
File Edit Element View
Recalculate Bitmaps
a at 97 size 16 from TimesNewRoman
File Edit Element View
a at 97 size 18 from TimesNewRoman
File Edit Element View
Recalculate Bitmaps
a at 97 size 20 from TimesNewRoman
File Edit Element View

- Zu den TrueType-Formatbeschreibungen gehört eine virtuelle Maschine, die frei programmiert werden kann, um die Kontrollpunkte der Kurve an eine Rasterung anzupassen.
- Jedem einzelnen Zeichen kann eine entsprechende Prozedur beigefügt werden. Darüber hinaus werden globale Prozeduren unterstützt, die von den für ein einzelnes Zeichen zuständigen Prozeduren aufgerufen werden können.
- Für die globalen Prozeduren gibt es die Tabellen *fpgm* und *prep*. In *cvt* können globale Variablen zusammen mit ihren Initialwerten spezifiziert werden.
- Das TrueType-Format enthält nur den Code der virtuellen Maschine, der in dieser Form nur schwer lesbar ist.

Virtuelle Maschine für die Instruktionen



- Die virtuelle Maschine führt die in einem separaten Speicher abgelegten Instruktionen aus.
- Zugänglich ist ein Stack für Berechnungen und Parameterübergabe, globale, bereits vorinitialisierte Variablen (CVT), der aktuelle Grafikzustand und die in Zonen Z1 und Z0 organisierten Kontrollpunkte der Kurven.

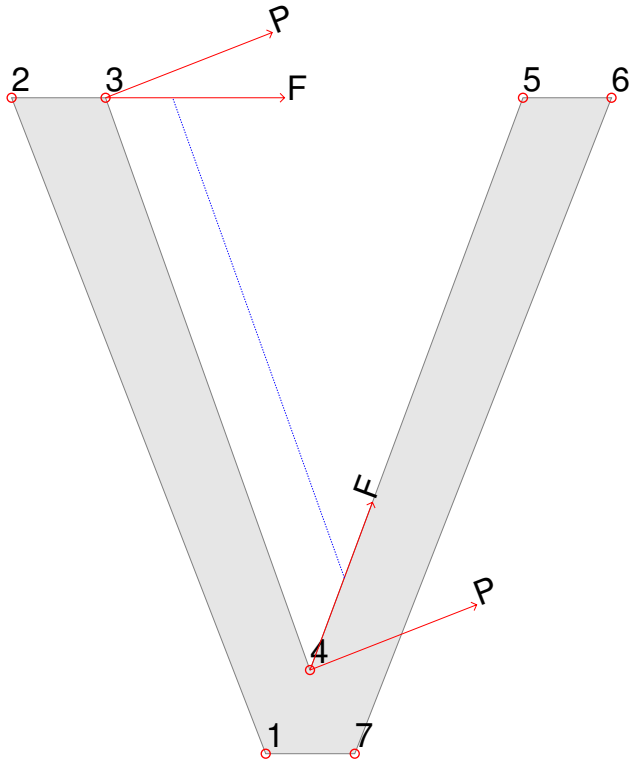
Folgende Datentypen werden unterstützt:

- *long* und *ulong*: ganze Zahlen, jeweils 32 Bit (für die Indizierung, insbesondere von Punkten, und für Boolean-Werte)
- *f26.6*: Zahl in Fixpunkt-Darstellung mit 26 binären Stellen vor dem Komma und 6 binären Stellen nach dem Komma (wird zur Repräsentierung der Koordinaten und von Distanzen verwendet)
- *f2.14*: Fixpunkt-Darstellung für Koordinaten normierter Richtungsvektoren (für Freiheits- und Projektionsvektor)

Der Stack besteht nur aus 32-Bit-Wörtern. Arithmetische Operationen gibt es nur für Werte des Typs *f26.6*.

- Alle Kontrollpunkte der einzelnen Kurven sind entsprechend der vorgegebenen Reihenfolge durchnummeriert (beginnend ab 1) und werden über eine Zone und einen Index ausgewählt.
- Für jeden Punkt gibt es die Koordinaten und die Information, ob dieser auf oder außerhalb der Kurve liegt.
- In der Zone *Z1* sind alle vorgegebenen Punkte zu finden.
- Die Zone *Z0* (auch *twilight zone* genannt) ist zu Beginn leer, kann aber frei verwendet werden.

- Ziel der Instruktionen ist es, die Punkte in *Z1* an das vorgegebene Raster anzupassen.
- Alle Punkte können individuell verschoben werden. Es ist auch möglich, die Eigenschaft zu verändern, ob ein Kontrollpunkt auf der Kurve oder außerhalb davon liegt.
- Das Verschieben der Punkte unterliegt dem Rastergitter und dem aktuellen graphischen Zustand.
- Wenn einige Punkte verschoben worden sind, können die dazwischenliegenden Punkte in dazu passender Weise nachträglich interpoliert werden.
- Wenn die Instruktionen abgearbeitet sind, kommt der klassische Rasterisierungs-Algorithmus zum Zuge, optional mit Erweiterungen, die einige Sonderfälle besser behandeln oder Anti-Aliasing unterstützen.



Nach einer Grafik von H. Schwarz
aus P. Karow: *Digitale Schriften*

- Zum graphischen Zustand gehört der Freiheits- und der Projektionsvektor.
- Distanzen werden immer nur relativ zum Projektionsvektor gemessen oder interpretiert. Die Distanzen haben daher auch ein Vorzeichen, das wahlweise berücksichtigt wird oder nicht.
- Punkteverschiebungen erfolgen immer nur in Richtung des Freiheitsvektors.
- Im Beispiel wird der Projektionsvektor orthogonal zu der Strecke $\overline{P_3P_4}$ gewählt, dann eine feste Distanz gewählt, dann P_3 entlang dem Freiheitsvektor $\overline{P_2P_3}$ verschoben, danach P_4 entlang von $\overline{P_4P_5}$.

Wenn Punkte entlang dem Freiheitsvektor verschoben werden, dann kann die tatsächlich gewählte Verschiebungsdistanz von den Rundungsmodi und dem Raster abhängig gemacht werden.
Konkret lassen sich bei den Rundungsmodi, die zum graphischen Zustand gehören, drei Parameter einstellen:

- Periode: bei 1 werden die Gitterpunkte angesprungen, bei 0.5 die Halbgitterpunkte, bei 2 nur jeder zweiter Gitterpunkt.
- Phase: bei 0 werden genau die Gitterpunkte genommen, bei größeren Werten werden die angesprungenen Punkte entsprechend zum Gitter verschoben.
- Schwelle: regelt, wohin gerundet wird: bei 1 immer zum kleineren Wert, bei 0.5 zum näher gelegenen Wert.