System zur Erstellung von Schriften mittels Parameter und wiederverwendbaren Komponenten

Milo Telesko

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Oktober 2012

© Copyright 2012 Milo Telesko

Diese Arbeit wird unter den Bedingungen der *Creative Commons Lizenz Namensnennung–NichtKommerziell–KeineBearbeitung Österreich* (CC BY-NC-ND) veröffentlicht – siehe http://creativecommons.org/licenses/by-nc-nd/3.0/at/.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 7. Oktober 2012

Milo Telesko

Inhaltsverzeichnis

Erklärung iii

Kurzfassung vii

Abstract viii

1 Einleitung 1
- 1.1 Motivation 1
- 1.2 Zielsetzung 1
- 1.3 Aufbau der Arbeit 2

2 Schriftgestaltung 4
- 2.1 Typographie 4
  - 2.1.1 Definition 4
  - 2.1.2 Begriffe 5
  - 2.1.3 Liniensystem 5
- 2.2 Konstruktion von Buchstaben 6
- 2.3 Klassifizierung 8
  - 2.3.1 DIN 16518 8
  - 2.3.2 Matrix Beinert 9
- 2.4 Digitale Typographie 10
  - 2.4.1 Einteilung 10
  - 2.4.2 Zeichenkodierung 12
  - 2.4.3 Unicode 13
- 2.5 Formate 15
  - 2.5.1 Postscript 15
  - 2.5.2 TrueType 16
  - 2.5.3 OpenType 17

3 Automatisierte Schriftgestaltung 18
- 3.1 DTL Letter Modeller 19
- 3.2 Fontstruct 21
- 3.3 Kalliculator 21
- 3.4 Lettersoup 22

3.5 Multiple master 24
3.6 METAFONT 25
3.6.1 Zeichenfunktionen 25
3.6.2 Kurven 25
3.6.3 Pens 27
3.6.4 Parametrisierung und Makros 28
3.6.5 Derivate 28

4 Eigener Ansatz 29
4.1 Editor 29
4.2 Technische Einschränkungen 30
4.3 Beispielschrifen 30
4.4 Teilbereiche 32
4.5 Parameter 32
4.5.1 Ausdrücke 33
4.5.2 System Parameter 33
4.5.3 Parameter-Editor 34
4.6 Striche und Kurven 34
4.6.1 Punkte und Kurven 34
4.6.2 Stiche 35
4.6.3 Stricharten 35
4.6.4 Veränderungen der Dicke 36
4.6.5 Limitierungen 37
4.7 Wiederverwendung von Strukturen 38
4.7.1 Analyse 39
4.7.2 Folgerungen 43
4.7.3 Serifen-Editor 43
4.7.4 Limitierungen 49
4.8 Modifikationen 49
4.8.1 Aufbau 50
4.8.2 Punkte 50
4.8.3 Schnittpunkte 52
4.8.4 Skalierungen 54

5 Umsetzung 57
5.1 Aufbau 57
5.2 Parameter 59
5.3 Kurven und Striche 59
5.3.1 Kurven 60
5.3.2 Striche 62
5.3.3 Gerade Striche 63
5.3.4 Runde Striche 63
5.3.5 Änderung der Strichdicke 65
5.4 Serifen Editor 68

5.5 Modifikatoren 69
5.5.1 Aufbau 70
5.5.2 Punkte 71
5.5.3 Schnittpunkte 71
5.5.4 Skalierungen 72

6 Diskussion **73**
6.1 Aufbau von Schriften im Prototypen 73
6.2 Kritikpunkte 74
6.3 Einsatzbereich 75
6.4 Vergleich 75
6.4.1 METAFONT 75
6.4.2 Lettersoup 76
6.5 Fazit 76
6.5.1 Prototyp 77
6.5.2 Weiterverwendung einzelner Ansätze 77
6.5.3 Umfang der Arbeit 79
6.5.4 Schlussbemerkung 79

Quellenverzeichnis **80**
Literatur 80
Online-Quellen 81

Kurzfassung

Schriftgestaltung ist ein aufwendiger Prozess, der viel Zeit und Geschick braucht. Die Konstruktion von hochwertigen Schriften kann bis zu mehreren Jahren dauern. Viele der verfügbaren Lösungen zur Schriftgestaltung haben visuelle Editoren, die bei der Konstruktion der Zeichen helfen. Dies erlaubt ein sehr exaktes Arbeiten. Die Anpassung der Zeichen im Nachhinein ist jedoch mit einem gewissen Aufwand verbunden, da jedes Zeichen einzeln angepasst werden muss. Für die Schaffung von leicht veränderbaren Schriften muss dafür auf Beschreibungssprachen zurückgegriffen werden. Diese sind jedoch bei weitem nicht so intuitiv wie ein visueller Gestaltungsprozess.

In dieser Arbeit werden verschiedene Ansätze vorgestellt die versuchen, die Vorteile einer Beschreibungssprache mit denen eines visuellen Editors zu verbinden. Das dabei entstehende System erlaubt die Erstellung von einfachen Schriften, die über verschiedene Parameter veränderbar sind.

Abstract

Designing typefaces is a time-consuming process. It can take years for a professional font to be finished. Modern solutions for designing typefaces often offer a visual editor. It enables the designer to create very accurate letterforms. The drawback of these editors is that global changes are hard to apply. To adjust the appearance of the font, every single letter has to be changed manually. A possible solution for this problem is the use of special markup-languages which are capable of creating changeable fonts. The disadvantage of this approach is the missing visual editor.

In this work different approaches are made to combine the benefits of an markup-language and a visual editor which allows the creation of simple typefaces that are changeable via parameters.

Kapitel 1

Einleitung

1.1 Motivation

Bei der Gestaltung von Schriften muss meist sehr geplant vorgegangen werden. Die einzelnen Buchstaben müssen von ihrer Größe und dem optischen Erscheinungsbild zusammenpassen. Gängige Editoren verwenden bei der Konstruktion Konturen, die die einzelnen Buchstaben beschreiben [21, 23]. Diese Konturen können visuell gezeichnet werden und sind somit sehr einfach zu erstellen, jedoch sind nachträgliche Änderungen sehr mühsam. Jede Kontur muss einzeln angepasst werden, wenn das Aussehen der Schrift angepasst werden soll.

Alternative Systeme wie METAFONT bieten zwar die Möglichkeit die Schrift im Nachhinein komplett zu verändern, jedoch müssen diese mittels einer Beschreibungssprache erstellt werden. Dies ist ein aufwendiger Prozess, der außerdem ein sehr abstraktes Denken voraussetzt.

Eine Lösung würde ein Werkzeug darstellen, in dem es möglich ist, gewisse Vorteile von unterschiedlichen Technologien zu verbinden. Die in dieser Arbeit vorgestellten Ansätze versuchen eine mögliche Lösung zu zeigen. Es sollen dabei die Flexibilität einer Beschreibungssprache mit dem Komfort eines visuellen Editors verbunden werden.

1.2 Zielsetzung

Ziel dieser Arbeit und des dazugehörigen Projektes soll ein Prototyp sein. Dieser lotet die verschiedenen Möglichkeiten, wie eine Schrift einfach beschrieben werden kann und dabei veränderbar bleibt, aus. Die Beschreibung von Teilen einer Schrift soll nach einem logischen Gesichtspunkt erfolgen. Werte und Eigenschaften die zusammengehören oder in einer Abhängigkeit stehen, sollen über einzelne Parameter oder Modifikatoren verbunden werden können. Auf diese Weise soll eine Änderung, an einem dieser Parameter, eine Änderung der ganzen Schrift verursachen.

Es soll dabei kein fertiges Programm entstehen mit dem Schriften erstellt werden, sondern nur ein Werkzeug mit dem ein Vorlage entworfen werden kann. Diese Vorlage soll möglichst flexibel bleiben und schnell angepasst werden können. Im Gegensatz zu Editoren die es am Markt gibt, sollten Striche nicht mit einem geschlossenen Pfad erstellt, sondern nur dessen Verlauf und Dicke angegeben werden. Dieser Weg soll mehr an den kalligraphischen Vorgang angelehnt sein. Die Komplexität soll dabei nicht zu hoch sein, wie bei ähnlichen Technologien.

Der Prototype soll ein visueller Editor sein, der nur minimal mit Scriptsprachen in Berührung kommt. Es soll ein WYSIWIG[1] Editor entstehen der unmittelbar Rückmeldung über Veränderungen gibt.

Das System soll möglichst schnell Änderungen an der Vorlage vornehmen können um einen groben Entwurf für eine Schrift zu erstellen. Dieser soll dann in einem anderen Programm verfeinert werden können. Die Information über den Aufbau würde bei diesem Schritt verloren gehen.

Dieses System dient vor allem im ersten Schritt der Schriftgestaltung als Grundlage. Wie bei der Konstruktion von Zeichnungen, die aus Grundformen zusammengesetzt sind und anschließend verbunden werden, soll dieses System die Grundform für eine Schrift liefern. Die ist vielleicht nicht der Weg den professionelle Gestalter wählen würden, diese sind aber nicht das Zielpublikum. Vielmehr steht die Möglichkeit schnell viele Vorlagen zu erstellen im Vordergrund.

1.3 Aufbau der Arbeit

Die Arbeit teilt sich in zwei große Bereiche auf. Nach der Einleitung wird in den zwei Folgenden Kapiteln auf Schriftgestaltung eingegangen. Hier werden Themen wie Klassifizierung oder Gestaltungsprozess vorgestellt und wichtige Begriffe definiert.

Im zweiten Teil der Schriftgestaltung wird speziell auf die digitale Typographie eingegangen. Es soll Anfangs ein Überblick geschaffen werden, welche unterschiedlichen Arten es gibt, wie Schriften am Computer gespeichert werden können. In weiterer Folge werden wichtige Formate und Kodierungsverfahren vorgestellt. Vor allem deren Ansätze zur Wiederverwendung und Automatisierung werden ausführlicher besprochen.

Der zweite große Bereich dieser Arbeit widmet sich den eigenen Ansätzen. Hier wird beschrieben welche Verfahren gewählt wurden um ein System zu erstellen, das in der Lage ist Teile der Schriftgestaltung zu automatisieren. In diesem Kapitel finden sich unterschiedliche Ansätze wieder, die genau beschrieben werden.

Nach dem eigenen Ansatz wird die Umsetzung anhand eines Prototypen vorgestellt. Dieses Kapitel beschreibt den Aufbau und die Hintergründe. Es

[1] Abkürzung für *What you see is what you get*.

wird die technische Realisierbarkeit verschiedener Ansätze besprochen und die verwendeten Wege erklärt.

Zum Schluss wird das Kapitel Diskussion sich mit den Ergebnissen dieser Arbeit befassen und sich möglichen Problemen oder Erweiterungen widmen.

Kapitel 2

Schriftgestaltung

2.1 Typographie

2.1.1 Definition

Der Begriff Typographie ist nicht ganze eindeutig und umfasst je nach Kontext und zeitlicher Periode einen unterschiedlich großen Bereich der Schriftgestaltung. Für die bessere Unterscheidung werden hier eine historische und eine moderne Definition geliefert. In dieser Arbeit bezieht sich der Begriff Typographie meist auf die reine Erstellung einer Schrift.

Historische Definition

Historisch gesehen beschreibt es die Vervielfältigung von Werken mittels eines Druckverfahrens, bei dem bewegliche Letter verwendet werden. Der Unterschied zu Kalligraphie ist, dass durch die Verwendung dieser Lettern, das erzeugte Schriftbild einer Schriftart einheitlich ist. Derselbe Buchstabe wird im Text immer optisch gleich dargestellt. So sieht für den Leser zum Beispiel jedes „*e*“ im Text identisch aus, sofern dieselbe Schriftart und Größe verwendet wird. Diese Definition wird vor allem verwendet, wenn man Typographie im Unterschied zu Kalligraphie betrachtet.

Moderene Definition

Die Bedeutung des Begriffs hat sich inzwischen verändert und umfasst einen weitaus größeren Bereich. Demnach wird heute zwischen Mikro- und Makrotypographie unterschieden. Die Mikrotypographie umfasst das eigentliche Erstellen einer Schriftart, angefangen vom Entwurf bis zu dessen Umsetzung und ist damit näher an der ursprünglichen Bedeutung. Da es meistens um diesen Bereich geht wird von nun an, wenn von Typographie die Rede ist, Mikrotypographie gemeint sein. Den zweiten Bereich, den der neue Begriff der Typographie umfasst (Makrotypographie), beschreibt Bereiche wie Lay-

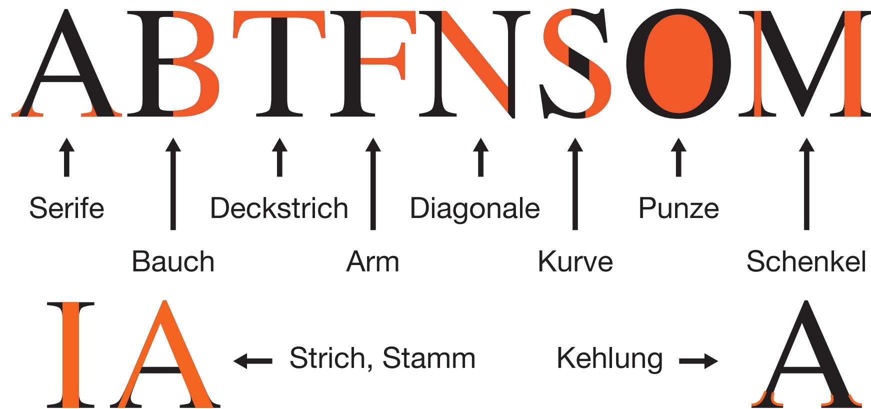


Abbildung 2.1: Bestandteile von Buchstaben.

out, Wahl der Schriftart und ähnliche Gestaltungen. Auf diese Themen wird nicht weiter eingegangen, da es keine weitere Rolle spielt [17].

Die Möglichkeiten, im Bereich der Typographie, haben sich mit der Entwicklung im digitalen Zeitalter schlagartig erweitert. Viele der verwendeten Begriffe stammen jedoch noch aus der analogen Zeit. Aus diesem Grund befassen sich die folgenden Abschnitte mit diesem Thema aus einer historischen Sicht.

2.1.2 Begriffe

Um die einzelnen Teile, aus denen Buchstaben aufgebaut werden, zu beschreiben, sind eigene Begriffe notwendig. Diese Begriffe sind nicht genormt, jedoch hat sich eine gewisse Terminologie durchgesetzt die weitgehend verwendet wird [16, 15]. Abbildung 2.1 führt die wichtigsten Bestandteile an. Diese Liste ist nicht vollständig, jedoch beinhaltet sie die meisten Begriffe die in weiterer Folge vorkommen werden.

Oft wird zwischen *Glyphe* und *Buchstaben* unterschieden. Buchstabe beschreibt allgemein ein Zeichen mit dessen Bedeutung. Glyphe hingegen meint ein konkretes Schriftzeichen in dessen graphischer Darstellung. Es kann in verschiedenen Schriftarten unterschiedliche Glyphen für ein *P* geben. Diese unterschiedlichen Glyphen beschrieben aber alle denselben Buchstaben. Ein weiterer wichtige Begriffe ist *Geviert*, welche dies Größe einer Glyphe samt Weißraum meint.

2.1.3 Liniensystem

Das Liniensystem in der Typographie wird dazu verwendet um Buchstaben auszurichten. Buchstaben teilen sich in verschiedene Gruppen, deren Höhe

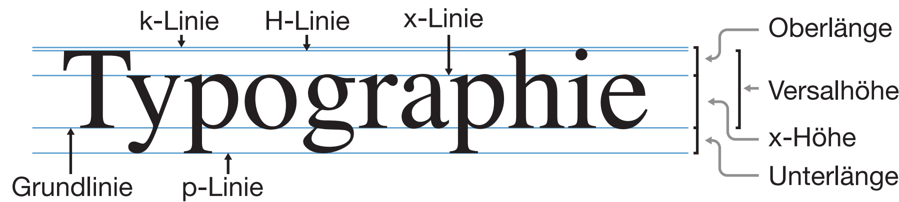


Abbildung 2.2: Typographisches Liniensystem.

bis zu einer gewissen Linie reicht. Abbildung 2.2 zeigt diese Linien anhand von mehreren Buchstaben [4, 6, 15]. Die Grundlinie stellt den Ausgangspunkt dar. Dies ist die Nulllinie. Die Namen der Linien leiten sich von einzelnen Buchstaben ab, die typisch sind für diese Höhe. So wird die Höhe der Kleinbuchstaben oft durch das x definiert. Diese Höhe wird x-Höhe genannt und ist der Abstand zwischen Grundlinie und x-Linie. Dasselbe gilt für die Großbuchstaben, wobei hier der Buchstabe H verwendet wird. Die p-Linie beschreibt die Ausdehnung nach unten und deren Abstand zu Grundlinie wird als Unterlänge bezeichnet. Die Oberklänge reicht meist etwas über die Versalhöhe. An dieser richten sich die Kleinbuchstaben aus die einen langen Strich haben, wie zum Beispiel das *h, l* oder *k*.

Diese Linien dienen als Referenzpunkte. In manchen Fällen müssen Buchstaben diese leicht übertreten, wie im Falle des *o*. Dies geschieht damit der Buchstabe optisch gleich groß wirkt wie andere Kleinbuchstaben. Dieser Teil der über die Linien reicht wird Überhang genannt.

2.2 Konstruktion von Buchstaben

Bei einer Schrift ist es wichtig, dass die Buchstaben im Zusammenspiel ein harmonisches Bild liefern. Sollten einzelne Buchstaben nicht passen, kann dies das gesamte Schriftbild zerstören. Es ist dadurch von Vorteil wenn bei der Gestaltung einer Schrift auf gewisse Hilfsmittel zurückgegriffen werden kann. Diese sollen helfen dieser Vorgang systematisch anzugehen. Es gibt kein festes System, auf welche Art und Weise dieser Gestaltungsprozess ablaufen soll. Je nach Vorlieben des Typographen kann dieses anders sein. Karen Chang beschreibt einen möglichen Ansatz [4]. Der Grund wieso sie eine bestimme Reihenfolge bei der Konstruktion gewählt hat, liegt darin, dass viele Buchstaben ähnliche Formen haben oder voneinander abgeleitet werden können. Einer der möglichen Wege dies zu nutzen ist, wenn man einen fertigen Buchstaben als Grundform für einen neuen verwendet.

Kleinbuchstaben eigen sich vor allem für dieses Verfahren, da diese aus einer Handschrift entstanden sind. Ursprünglich wurden diese Buchstaben mit einer kalligraphischen Feder konstruiert. Der Umstand, dass es bei der

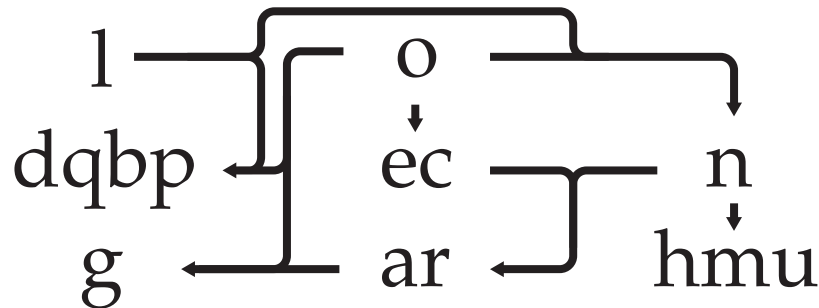


Abbildung 2.3: Konstruktionsweg von Buchstaben nach Karen Cheng [6].

Verwendung solch einer Feder einiges zu beachten gibt, ist sehr vorteilhaft. Es ist nicht möglich die Linien eines Buchstabens beliebig zu zeichnen. Es muss eine gewisse Richtung eingehalten werden, wie eine Feder gezogen wird. Da beim Schreiben die Feder in der Hand liegt und dadurch mit einem gewissen Winkel zum Papier steht, wird diese meistens nach unten oder horizontal gezogen. Wenn die Feder nach oben gezogen wird kann es passieren, dass diese im Papier stecken bleibt oder daran kratzt. Dadurch entsteht eine unsaubere Linie. Durch diese Limitierung ist es nötig eine komplexere Formen in kleine einfachere Teile zu zerlegen. Diese Zerlegung hat zur Folge, dass oft die Striche aus denen ein Buchstabe zusammengesetzt wird, bei anderen Buchstaben in gleicher Form verwendet werden.

Eine Ausschnitt aus ihrer Überlegung (Abb. 2.3) zeigt die Zusammenhänge einer Gruppe von Buchstaben. Als Ausgangsform wird als Erstes ein kleines *o* und ein kleines *l* gewählt. Durch ihren geometrisch sehr einfachen Aufbau, eignen sie sich besonders als Grundform. Diese zwei Buchstaben legen gleichzeitig auch viele Eigenschaften der Schrift fest, wie zum Beispiel Dicke der Striche und den Winkel der Feder.

Im ersten Schritt werden aus dem *o* und dem *l* zwei weitere Buchstabengruppen abgeleitet. Es entstehen dadurch einerseits die Kombination *d, q, b* und *p*. Das *o* stellt dabei die Rundung da und wird mit einem *l* an der Seite kombiniert.

Die zweite neue Gruppe umfasst den Buchstaben *n*. Dieser Schritt ist nicht so offensichtlich wie der vorangegangene. Hierbei geht es vor allem um die Größe des Punzens. Optisch sollte der Raum im *o* und im *n* gleich groß sein.

Ein *u* kann durch Rotation aus einem *n* erstellt werden. Dabei sind leichte Anpassungen an der Stelle an dem sich die beiden Striche treffen nötig. Zur selben Gruppe gehören das *m* und das *h*. Bei dem *h* wird der Strich aus dem *n* verlängert und bei dem *m* muss der rechte Teil verdoppelt werden.

Die Ableitung des *e* und *c* von *o* ist wiederum sehr einfach. Alle drei

Buchstaben haben eine sehr ähnliche runde Form.

Das *a* und *r* bestehen aus der Gruppe *e,c* und |textitn. Die Form des *r* findet sich im linken Teil des *n* wieder. Bei dem *a* in dieser Form ist der Zusammenhang kaum erkennbar. In vielen Schriften ist jedoch der Buchstabe dem *c* sehr ähnlich.

Als letzter Buchstaben dieses Beispiels wird hier das *g* angeführt. Dieser Zusammenhang ist nicht sehr offensichtlich. Es können nur Teile des *r* und die Rundung des *o* erkannt werden.

Dieses *copy & paste* Verfahren ist bei weitem nicht so einfach wie es den Anschein hat [14, S. 64]. Obwohl die Form der Striche sehr ähnlich ist, bedeutete dies nicht, dass diese einfach unverändert übernommen werden können. Es ist oft notwendig diese zu Stauchen oder ihre Dicke zu ändern. Es ist nicht weiter möglich ein *m*, wie im vorherigen Absatz, einfach aus einem *n*, mit zusätzlichen Bogen, zu konstruieren. Optisch wäre dieser Buchstabe viel zu breit und würde im Schriftbild durch seinen Weißraum im Punzen auffallen.

2.3 Klassifizierung

Um die Vielfalt der Schriften, die in den letzten Jahrhunderten erstanden sind, in sinnvolle Gruppen einzuteilen, gibt es mehrere Klassifikationsmodell. Diese Modelle schaffen gleichzeitig eine Terminologie um Schriften anhand ihrer Eigenschaften zu beschreiben.

Bei Klassifikationsmodellen werden meinst die unterschiedliche Ausprägung einzelner Elemente der Buchstaben verwendet, um diese einzuteilen. Zu den wichtigsten Eigenschaften gehören Strichstärke, Ausrichtung der Achsen, Rundungen oder auch Form der Serifen. Epochen wie Renaissance, Barock oder Klassizismus haben dabei jeweils für sie markante Eigenschaften hervorgebracht und sind meist auch Namensgeber dafür.

2.3.1 DIN 16518

Eine Einteilung wurde unter anderem vom Deutschen Institut für Normierung vorgenommen. Diese Klassifizierung nach DIN 16518 legt dabei elf verschiedene Gruppen fest [19]:

- Venezianische Renaissance-Antiqua
- Französische Renaissance-Antiqua
- Barock-Antiqua
- Klassizistische Antiqua
- Serifenbetonte Linear-Antiqua
- Serifenlose Linear-Antiqua
- Antiqua-Varianten

- Schreibschriften
- Handschriftliche-Antiqua
- Gebrochene Schriften
- Fremde Schriften

Die meisten Gruppen beinhalten Schriften aus der Antiqua-Familie. Es handelt sich dabei um Schriften mit runden Bögen. Diese Gruppe umfasst die meisten heute in Westeuropa verwendeten Schriften. Diese stellen eine Mischung von handschriftlichen Kleinbuchstaben und römischen Großbuchstaben da. Das erste Aufkommen dieser Schriftfamilie war in der Renaissance im 15. Jahrhundert. Sie wurde von den damals aufkommenden Humanisten entwickelt und werden deswegen oft als humanistisch bezeichnet.

Neben den Antiqua-Schriften werden unter anderem noch gebrochene Schriften angeführt, zu denen unter anderem die Fraktur und gotische Schriften zählen. Der Name dieser Gruppe leitet sich von dem Umstand ab, dass die Rundungen dieser Schriften gebrochen sind.

Durch das inzwischen hohe Alter dieser Norm, die im Jahre 1964 das letzte Mal überarbeitet wurde, werden viele digitale Schriftentwicklungen nicht berücksichtigt. Wolfgang Beinert schreibt zu dieser Problematik: *„Die DIN 16518 aus dem Jahre 1964 ist ausschließlich für Bleisatz-Werdruckschriften bis in die 1970er Jahre anwendbar. Sie weicht stark von internationalen Standards und wissenschaftlichen Betrachtungsweisen ab. Sie ist als Klassifikationsmodell für die digitale Typographie nicht mehr geeignet.“* [17].

2.3.2 Matrix Beinert

Wolfgang Beinert versucht mit seiner Klassifizierung, der sogenannten *Matrix Beinert*, eine modernere Einteilung zu schaffen, die auch neue digitale Schriften berücksichtigt. Diese Matrix verfügt über eine mehrstufige Unterteilung. Es wird dabei in der ersten Stufe nur sehr grob zwischen den unterschiedlichen Typen unterschieden [17]:

- Antiqua
- Egyptienne
- Grotesk
- Corporate Fonts
- Zierschriften
- Bildschirmschriften
- Gebrochene Schriften
- Nichtrömische Schriften
- Bildzeichen

Schriften der ersten drei Gruppen sind leicht zu trennen. Groteske Schriften sind einfach von Egyptienne und Antiqua Schriften zu unterscheiden, da

diese keine Serifen haben. Egyptienne und Antiqua unterscheiden sich durch die Serifen. Bei Egyptienne sind diese weitaus stärker betont. Selbst für Laien auf diesem Gebiet sollte es sehr schnell möglich sein, Schriften in eine dieser Kategorien einzuordnen. Erste eine weitere Einteilung dieser Kategorien zeigt eine feinere Unterteilung, die ähnlich ist wie bei der DIN-Norm.

Bei Corporate Fonts und Bildschirmschriften werden die Schriften nach ihrer Verwendung eingeteilt. Diese können auch Schriften aus anderen Kategorien beinhalten. Ein Corporate Font kann die Schrift des ORF-Logos sein. Diese würde auch als groteske Schrift eingeordnet werden. Bildschirmschriften sind speziell für den Bildschirm angepasste Schriften. Diese sollen eine besonders gute Darstellung am Bildschirm liefern.

Die letzten Kategorien beinhalten gebrochene Schriften, wie schon in der DIN-Norm beziehungsweise nichtrömische (Arabisch usw.) und Bildzeichen (Chinesisch, Japanisch).

2.4 Digitale Typographie

2.4.1 Einteilung

In der digitalen Typographie wird meist zwischen zwei oder drei verschiedenen Typen von Schriften unterschieden. Diese Unterteilung bezieht sich darauf, in welcher Form die einzelnen Glyphen gespeichert werden und hat nichts mit einer Klassifizierung oder mit einem Dateiformat zu tun. Es ist eine rein technische Einteilung.

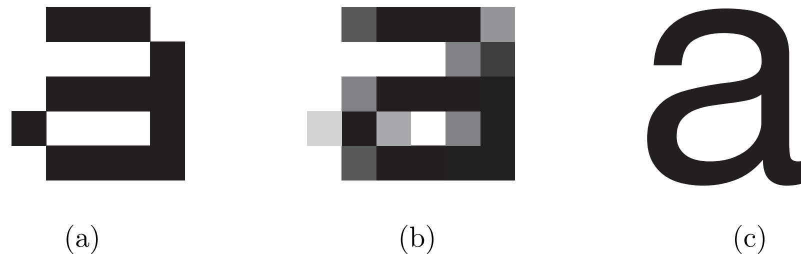
Es wird meistens zwischen Rasterfonts und Vektorfonts unterschieden. Oft wird noch eine dritte Gruppe von Schriften angeführt, die sogenannten *Stroke Fonts*[1]. Diese werden meist in die Gruppe der Vektorfonts eingeteilt. Sie weisen aber Eigenschaften von beiden Typen auf.

Rasterfonts

Da in den Anfängen des Desktop-Computers die Auflösung der Bildschirme sehr gering und wenig Rechenleistung vorhanden war, musste ein sehr einfaches Verfahren zur Darstellung von Schrift verwendet werden. Zu diesem Zweck wurden die Rasterfont, oder oft auch Bitmap-Fonts genannt, entwickelt.

Rasterfonts stellen eine der einfachsten Möglichkeiten zur Speicherung von Glyphen da. Damit keine Umwandlung beim darstellen nötig ist, wird die Schrift als vorgefertigte Grafik abgespeichert. Abbildung 2.4 (a) zeigt einen Buchstaben eines Rasterfont. In dieser Grafik wird die Glyphe über Pixel gezeichnet. Die Grafik kann entweder monochrom oder wie in den meisten Fällen binär sein. Für jede Glyphe wird ein Raster in einer bestimmten Größe

[1] Es gibt für diese Art von Schriften kaum eine einheitliche Bezeichnung bzw. Übersetzung im Deutschen.



(a) (b) (c)

Abbildung 2.4: Raster und Vektorschriften.

festgelegt. Die Dimension des Rasters kann sich dabei von Glyphe zu Glyphe unterscheiden, da diese nicht zwingend monospaced[2] sind.

Die Anforderungen an einen modernen Font beinhalten auch die Verwendung für mehrere Schriftgrößen. Dieser Punkt stellt für Rasterfonts ein Problem dar. Um ein optimales Ergebnis zu erzielen, müsste für jede Schriftgröße die komplette Schrift neu erstellt werden. Theoretisch könnte mittels Interpolation eine Schrift auf eine gewünschte Größe gebracht werden, dies würde aber meist nicht ohne Qualitätsverlust möglich sein. Für professionelle Druckerzeugnisse sind Rasterfonts somit kaum geeignet.

Der Speicherplatz, den ein Rasterfont braucht, hängt von der Größe der Buchstaben ab. Er ist leicht berechenbar, da dieser sich aus der Anzahl der Pixel pro Glyphe mal der Farbtiefe zusammensetzt. Bei kleiner Dimension von nur wenigen Pixeln (zum Beispiel 5x7 oder 8x16) ist der verbrauchte Gesamtspeicher sehr gering. Für größere Schriften springt der Speicherbedarf quadratisch an.

Vektorfonts

Im Gegensatz zu diesen Rasterfonts gibt es unter anderem noch Vektorfonts. Bei dieser Variante werden die Glyphen durch Konturen beschrieben. Diese Konturen sind mathematische Beschreibungen von Kurven. Erst beim Rendern einer Glyphe wird diese Kontur in eine Pixelmaske umgewandelt. Der Vorteil dieser mathematischen Beschreibung, liegt in der Möglichkeit die Glyphen beliebig skalieren zu können. Abbildung 2.4 (c) zeigt einen Buchstaben eine Vektorfonts. Dies bedeutet im Gegensatz zu den Rasterfonts, dass ein und dieselbe Glyphe für fast jede Auflösung verwendet werden kann.

Jedoch kann es bei Vektorfonts zu Problemen kommen, sollten diese für sehr geringe Auflösungen zwischen 8 und 12 Pixel Größe eingesetzt werden. Da die erzeugten Pixelraster meist monochrom sind, bedeutet dies, dass stark gerundet werden muss. Abbildung 2.4 (b) zeigt dieses Problem. Das Ergebnis mag zwar mathematisch korrekt sein, jedoch kann die Lesbarkeit stark dar-

[2]Schriften bei denen jeder Buchstaben gleich breit sind.

unter leiden. Zu diesem Zweck sind manche Schriften extra noch aufbereitet um auch optisch gute Ergebnisse für kleine Schriftgröße zu bieten.

Die Konturen eines Vektorfonts werden mittels Koordinaten beschrieben. Dies ermöglicht einen sehr geringen Platzverbrauch, da für eine Linie nur wenige Werte nötig sind. Gegenüber Rasterfonts ist der Speicherbedarf meist geringer, sofern es sich um größere Schriftgrößen handelt.

Stroke Fonts

Eine dritte Art von Fonts stellen die sogenannten *Stroke Fonts* dar. Diese speziellen Schriften werden wie Vektorschriften über einen Pfad definiert, jedoch wird die Dicke des Striches mit einer *Federspitze* erzeugt. Diese fährt mit einer bestimmten Form am Pfad entlang und zeichnet die Glyphe. Die erzeugten Glyphen werden als Pixel ausgegeben und können nicht skaliert werden. Für diese Art von Schriften gibt es nur wenige Vertreter, die zu einem verwendbaren System weiterentwickelt wurden. METAFONT ist einer der wenigen dieser Gattung. Der Abschnitt 3.6 geht mit METAFONT genau auf den Aufbau und Funktionalität von *Stroke Fonts* ein.

2.4.2 Zeichenkodierung

Eine Zeichenkodierung (englisch. *character encoding*) ist eine Tabelle, mithilfe der ein Bitmuster als Zeichen interpretiert werden kann. Es wird dabei jedem Zeichen ein eindeutiger Code zugewiesen. Dieses Verfahren ermöglicht es Zeichenketten zu speichern oder zu übertragen.

In den Anfängen der digitalen Datenverarbeitung wurden simple Standards verwendet, wie zum Beispiel ASCII (anglisch *American Standard Code for Information Interchange*). Mit 95 sichtbaren Zeichen (A-Z, a-z, 1-9 und den gängigsten Satzeichen) ist es ein recht überschaubarer Zeichensatz. Die darauf folgenden Standards wie ISO-8859 waren mit ca. 220 sichtbaren Zeichen ein wenig umfangreicher. Da es ca. 15 sprachspezifische Teilstandards in dieser ISO-Norm gibt, beläuft sich die Anzahl der Zeichen, die nötig sind um Font zu erstellen um vielfaches. Eine weitere Entwicklung stellt Unicode da, welches mit einer maximalen Zeichenanzahl von über vier Millionen (UTF-32) weit mehr Platz bietet als bisherige Standards.

Ein Font muss nicht zwangsläufig sämtlichen Zeichen eines Zeichensatzes unterstützen (gerade im Fall von Unicode ist dies kaum möglich), jedoch sollte dieser die gängigsten Buchstaben, Satzeichen und Zahlen beinhalten.

Grundsätzlich ist die Kodierung von Zeichen wichtig, besonders wenn es um die richtige Darstellung geht,. Es hat jedoch keine offensichtlich Bedeutung im Bezug auf die reine Gestaltung einer Schrift. In diesem Punkt unterscheidet sich Unicode von den meisten anderen Systemen. Bei der Entwicklung von Unicode wurde darauf geachtet, dass dieser nicht nur eine simple Tabelle ist, die sämtliche Zeichen beinhaltet, sondern dass auch gewisse

Eigenschaften der einzelnen Buchstaben oder Schriftsysteme ausgenutzt werden können, um diese besonders optimal zu kodieren.

2.4.3 Unicode

Unicode spielt als eines der umfangreichsten Systeme seiner Art eine wichtige Rolle in der digitalen Typographie. Es wurde sich bei der Entwicklung von Unicode zur Aufgabe gemacht, sämtliche Schriftsysteme und Symbole, die im Laufe der Menschheitsgeschichte entwickelt wurden, zu erfassen. Neben Schriftsystemen werden auch mathematische und technische Symbole gesammelt. Im Endeffekt sollte Unicode eine Lösung darstellen, die sämtliche im Umlauf befindlichen Zeichenkodierungen ablösen kann. Es wurden bis heute über 110.000 Zeichen in Unicode definiert (Stand Juli 2012) [36]. Dies stellt einen enormen Zuwachs gegenüber den bis dato verwenden Standards da.

Wie Eingangs schon erwähnt, wurde Unicode dabei nicht nur als einfache Tabelle konzipiert, in der jedem Code ein Zeichen zugeordnet wird. Gerade durch die große Anzahl an Zeichen im Unicode Zeichensatz macht es möglich, dass gewisse Umstände ausgenutzt werden können, um gewisse Gruppen von Zeichen besonders effizient zu kodieren.

Unicode basiert auf mehreren Prinzipien. Diese sollten unter anderem sicherstellen, dass Unicode als Standard akzeptiert wird und auch auf zukünftige Änderungen hin flexibel ist. Das Unicode Konsortium nennt folgende Prinzipien als die Grundlage auf der die Entwicklung basiert: *Universal repertoire, Logical order, Efficiency, Unification, Characters, not glyphs, Dynamic composition, Semantics, Stability, Plain Text, Convertibility*. Besonders zwei (bzw. drei) dieser Prinzipien sind besonders interessant. Erstens *Dynamic composition*, welches das zusammenfügen mehrere Einzelteile zu einem neuen Zeichen erlaubt (Siehe Abschnitt 2.4.3). Zweitens ist *Characters, not glyphs* wichtig, dass es eine logische Trennung zwischen Glyphen und Zeichen ermöglicht, wie im folgenden Abschnitt erklärt wird.

Zeichen und Glyphen

Unicode unterscheidet im Gegensatz zu vielen anderen Zeichensystemen zwischen Glyphen und Zeichen. Eine Glyphe stellt dabei eine visuelle Repräsentation eines Zeichens dar, im Gegensatz zum einem Zeichen, welches eine abstrakte Definition ist. So kann es zum Beispiel möglich sein, dass ein Zeichen auf Grund seiner Position oder im Zusammenhang mit anderen Zeichen (wie bei Ligaturen), anders dargestellt wird, wie man in Abbildung 2.5 sehen kann. Besonders interessant ist dies auch für den Abschnitt 2.4.3. Weiters kann es vorkommen, dass es für ein und das selbe Zeichen mehrere unterschiedliche Glyphen gibt. Andererseits kann auch dieselbe Glyphe für Zeichen mit unterschiedlicher Bedeutung verwendet werden, dessen Erscheinungsbild aber gleich ist. Dies passiert bei der sogenannten *Unification*. Es

Abbildung 2.5: Buchstabenkombination *ffi* ohne Ligaturen (links) und mit Ligaturen (rechts).

wurde dabei versucht alle chinesischen Schriftzeichen, die auch in anderen Sprachen (wie u.a. im Koreanischen und Japanischen) verwendet werden, zusammenzufassen und mit zusätzlicher Meta-Information zu versehen. Dies ermöglich es die Zeichen richtig zu verwenden und Platz im Unicode Raum zu sparen.

Dynamic composition

Eines weiteres Grundprinzip auf denen Unicode basiert ist *dynamic composition*. Dieses Prinzip wird mit einem Feature namens *ccmp* realisiert. Dies ermöglicht es verschiedene Glyphen zu kombinieren und damit ein neues Zeichen zu schaffen.

Verwendung findet dies unter anderem wenn mehrere Zeichen auf ein und das selben Grundzeichen basieren. Zum Beispiel unterscheidet sich der Buchstabe *„a“* von dem Buchstaben *„å“* nur durch das diakritische Zeichen. Es würde sich somit aus der Kombination der Glyphen für *„a“* und *„ ˚“* eine Glyphe für das Zeichen *å* ergeben. Die Schaffung von solchen Zeichen wird somit stark vereinfacht. Für die Umlaute im Deutschen ist es nicht mehr nötig eine eigene Glyphe zu erstellen. Es muss nur noch ein Vermerk erstellt werden, an welcher Position die Glyphe für die zwei Punkte an ein *a, u* oder *o* gelegt werden sollen. Aus diesem Grund wird ein eigener Bereich in Unicode zu Verfügung gestellt, indem diese diakritische Zeichen gesondert abgespeichert werden können. In manchen Sprachen wie zum Beispiel dem altgriechischem kann es vorkommen, dass mehrere diakritische Zeichen mit einem Buchstaben verbunden werden. Dies stellt kein Problem dar, da durch dieses System theoretisch beliebig viele Glyphen miteinander kombiniert werden können. Ein Unicode Renderer ist somit selbständig in der Lage neue Zeichen zu erzeugen.

Der Vorteil dieses Features ist, dass es nicht nötig ist für jedes Kombinationszeichen eine eigene Glyphe anzulegen, dies spart einerseits Speicherplatz und andererseits vereinfacht es den Designprozess und ermöglicht schnelle Änderungen am Font.

Der Einsatzbereich dieses Features beschränkt sich nicht nur auf diakritische Zeichen, was anhand des koreanischen Schriftsystems erkennbar wird.

Diese Schrift, die auch Hangul genannt wird, weist einige Besonderheiten auf. Es handelt sich hierbei nicht wie man vermuten könnte um Ideogramme wie in der chinesischen Sprache, sondern um eine Art Silbenschrift wobei diese im eigentlichen Sinn keine ist. Grundsätzlich besteht die Schrift aus einen Alphabet mit 24 Buchstaben. Diese dürfen jedoch nie nacheinander, wie beim lateinischen Alphabeten, geschrieben werden. Um ein gültiges Schriftzeichen zu erzeugen müssen zwei bis vier Buchstaben erst zu einem Silbenblock zusammengefasst werden. Die Verwendung dieser Silbenblöcke führt dazu, dass es nicht nur 24 Buchstaben als Zeichen gibt, sondern viele Tausend. *Dynamic composition* stellt nun eine Möglichkeit da, wie diese Kombination zu den Blöcken automatisch ablaufen kann.

2.5 Formate

Die folgenden Abschnitte sollten einen Einblick geben, welchen Aufbau und Funktionsumfang Schriftformate haben. Es wird TrueType, PostScript und OpenType behandelt. Letzteres ist wichtig, da es sich um das am verbreiteteste Format handelt und über interessante Features verfügt. Als weiteres Format wird METAFONT untersucht. Dieses Format ist vor allem wegen der Art wie Glyphen aufgebaut werden und dessen Ansätze zur Parametrisierung interessant. Das Augenmerk soll vor allem auf Bereiche der Technologien gelegt werden die für diese Arbeit wichtig Ansätze oder Einschränkungen bieten.

2.5.1 Postscript

PostScript ist einen von Adobe entwickelte Beschreibungssprache für Dokumente [9]. Die von PostScript verwendete Beschreibungssprache stellt wichtige Funktionen zum Zeichnen von Kurven zu Verfügung. Ein Beispiel für einen Postscript Code wird in Programm 2.1 gezeigt. Typisch ist die invertierte Notation für den Aufruf von Funktionen. Glyphen in einer PostScript Schrift werden mittels dieser Sprache gezeichnet.

PostScript verwendet Bézierkurven des dritten Grades für die Darstellung von Kurven [6]. Abbildung 2.6 (a) zeigt einen Buchstaben in einer PostScript Kurve. Diese Kurven haben je zwei Ankerpunkte an den Enden und zwei Kontrollpunkte dazwischen.

Das Format umfasst neben den Glyphen noch mehr Information, wie zum Beispiel für *Hints* oder *Kerning*. *Hints* werden benötigt um den Rendere beim Rastern der Schrift zu helfen ein möglichst klares Schriftbild am Bildschirm zu erzeugen. *Kerning* bestimmt den Abstand der Buchstaben zueinander. Buchstaben wie *A* und *V* müssen näher beieinander sein als *W* und *V*.

PostScript Schriften können über visuelle Editoren entworfen werden. Es besteht somit kein Grund diese Sprache zu lernen um eine PostScript Schrift zu erzeugen. Für PostScript gibt zahlreiche Versionen deren Funktionsum-

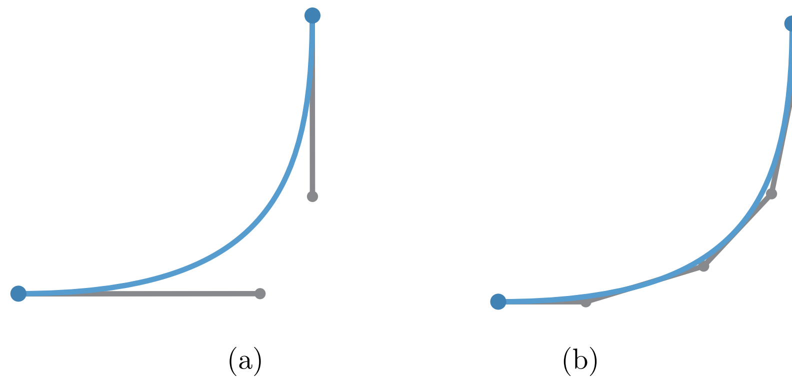


Abbildung 2.6: Vergleich von PostScript (links) und TrueType (rechts) Kurven.

```
1 newpath
2 100 100 moveto
3 100 0 lineto
4 0 0 lineto
5 closepath
6 1 0 0 setrgbcolor
7 fill
```

Programm 2.1: PostScript Code für ein rotes Quadrat von einer Seitenlänge 100 Einheiten

fang unterschiedlich groß ist und eine Reihe von Erweiterungen wie das in Abschnitt 3.5 erklärte Multiple master.

2.5.2 TrueType

TrueType ist ein von Apple entwickeltes Schriftformat. Sowie PostScript Schriften basieren auch TrueType Schriften auf Bezierkurven. Der große Unterschied liegt lediglich in der Höhe des verwendeten Grades. TrueType setzt auf Bézierkurven des 2-Grades im Gegensatz zu PostScript, welches Kurven des 3-Grades verwendet [6]. Abbildung 2.6 (b) zeigt eine TrueType Glyphe. Diese Kurve besteht aus einer Reihe von quadratischen Bézierkurven.

Als Vorteil der TrueType bevorzugten Variante wird oft die einfachere Berechnung angeführt. Kurven des 2-Grades lassen sich schneller Berechnen als solche des 3-Grades. Der Nachteil liegt wiederum in der Anzahl der Punkte die nötig sind um eine Form zu beschreiben. Diese ist bei TrueType meist höher als bei PostScript Schriften.

TrueType verfügt ebenso wie PostScript über *Kerning* und *Hints*. Die Implementierung des *Hintings* unterscheidet sich jedoch von dem Verfahren

von PostScript.

2.5.3 OpenType

OpenType ist ein von Microsoft und Adobe entwickeltes Font-Format. Es handelt sich dabei um eine Zusammenführung und Weiterentwicklung von TrueType und PostScript Fonts. OpenType ist eine Art Kontainerformat, da es möglich ist die Kurveninformation sowohl als TrueType als auch PostScript formatiert einzubetten. Man unterscheidet diese dann als TrueType-flavored und PostScript-flavored [30].

OpenType besitzt eine Reihe von Features die es von seinen Vorläufern abhebt. Features wie Ligaturen ermöglichen es Zeichenketten durch Glyphen ersetzten zu lassen. So ist es möglich für die Kombination von f und i eine eigene Glyphe zu erzeugen die beide Buchstaben verbindet. Diese Ligaturen können für beliebige Zeichenketten verwendet werden.

OpenType ist in dem Sinn relevant, da es sich um das heute üblich Format handelt in dem Schriften verwendet werden. Es limitiert durch den Funktionsumfang und die gängigen Implementierungen die Möglichkeit inwiefern eigene Features mit in die Schrift eingebettet werden können, oder ob diese vorher in OpenType brauchbare Information umgewandelt werden muss. Gerade die Features die es möglich machen einzelne Buchstabengruppen zusammenzufassen, deren Aufbau ähnlich ist, macht OpenType interessant.

Kapitel 3

Automatisierte Schriftgestaltung

Ansätze zur mathematischen Beschreibung von Schrift gibt es seit dem 15. Jahrhundert. Damals wurde versucht die einzelnen Teile von Buchstaben mittels geometrischer Formen zu beschreiben. Abbildung 3.1 zeigt zwei Buchstaben von Luca Pacioli aus dem 16. Jahrhundert. Diese Beschreibungen wurden im Laufe der Zeit immer wieder aufgenommen und verändert, unter anderem auch von Albrecht Dürrer.

In Paris des 19. Jahrhunderts wurden die textitPlaque Découpée Universelle entwickelt. Abbildung 3.2 zeigt diese Maske. Die Buchstaben werden mittels einen Stiftes konstruiert, indem die Form in den Schlitzen nachgezogen wird. Die Maske enthält dabei alle Buchstaben.

Es gibt viele weiter analoge Beispiele für Versuche Schrift in ein vorgefertigtes Muster zu bringen. Mit dem Aufkommen der digitalen Typographie haben sich die Möglichkeiten stark erweitert. Es gibt einige Projekte die diese Ansätze zur Automatisierung von Typographie verfolgen. METAFONT ist wohl das bekannteste und wird im Abschnitt 3.6 sehr ausführlich behandelt. Eine weitere wichtige Technologie stellt MultipleMaster dar. Diese von Ad-

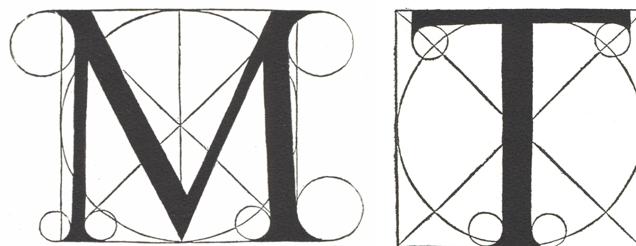


Abbildung 3.1: Beschreibung des Buchstaben *M* und *T* von Luca Pacioli Venedig 1509 [31, 32].

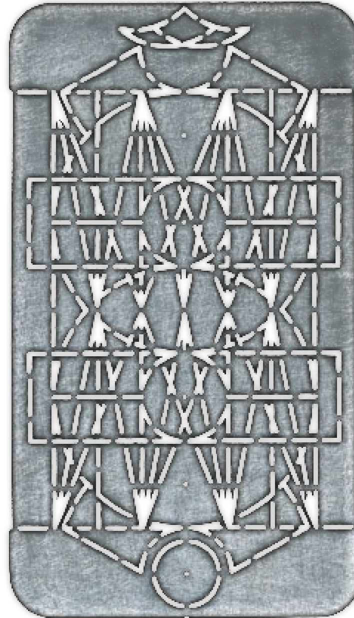


Abbildung 3.2: Maske zum Erstellen von Buchstaben *„Plaque Découpée Universelle“ Paris 1870* [33].

obe entwickelte Technologie wird nicht mehr weiterentwickelt, ist jedoch für Schriftdesigner nach wie vor ein wichtiges Werkzeug.

Viele der Projekte sind weder veröffentlicht noch fertiggestellt worden. In manchen Fällen gibt es nur Videos die deren Verwendung zeigen. Deren Ansätze sind jedoch interessant genug um Erwähnung zu finden, auch wenn über deren tatsächliche Verwendung nichts gesagt werden kann.

3.1 DTL Letter Modeller

Der DTL LetterModller (kurz: LeMo) ist ein von Frank E. Blokland entwickeltes Programm. Es handelt sich dabei nicht um einen reinen Editor zur Erstellung von Schriftarten, sondern viel mehr um ein Werkzeug, das es erlaubt die Form von Buchstaben zu definieren. Es ist in Begleitung einer PhD. Arbeit entstandenes Werkzeug und ist somit sehr experimentell. LeMo selbst wird als *. . . first step towards the automation of type design processes. . .* bezeichnet [20].

LeMo verwendet zwei verschiedene umfangreiche Systeme um Buchstabenformen zu erzeugen. Dabei wird zwischen den Groß- und Kleinbuchstaben getrennt. Diese Unterscheidung findet auf Grund ihrer unterschiedlichen Herkunft statt.

Für die Generierung der Glyphen stellt LeMo einige Parametern zu Ver-

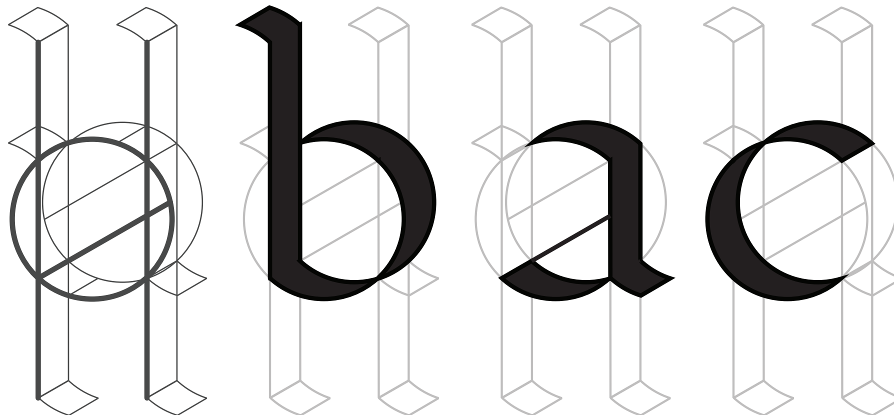


Abbildung 3.3: Aufbau der Glyphen für Kleinbuchstaben in Letter Modeller.

fügung. Diese können in einem numerischen Wertebereich verändert werden. Die ersten Parameter beziehen sich auf die Form des Pinsels der verwendet wird um die Glyphen zu zeichnen. Dies funktioniert ähnlich wie in METAFONT (Abschnitt 3.6), wobei es sich hier um einen rechteckigen Pinsel handelt bei dem die Höhe, Breite und die Neigung verändert werden kann. Weiter kann auf das Liniensystem Einfluss genommen werden. Es können jedoch nur die x-Höhe und die Ober- und Unterlänge verändert werden. Die letzten Parameter dienen dazu um die Glyphen in der Vertikalen zu skalieren und um die Schrift kursiv zu setzten.

Als Grundlage für Großbuchstaben wird ein Skelett verwendet. Dieses beschreit, ähnlich wie bei *Stroke Fonts*, den Verlauf der Striche und nicht dessen Umrisse. Dieses Skelett muss in einem separaten Programm erstellt und anschließend in LeMo importiert werden.

Für Kleinbuchstaben verwendet LeMo ein sogenanntes harmonisches Model (Abb. 3.3 links) [2, 3]. Diese Model beschreibt die Proportionen der Glyphen untereinander. Um eine Glyphe zu erzeugen, werden einzelne Flächen oder Linien in diesem Model ausgefüllt (Abb. 3.3 rechts). Dabei können diese beliebig miteinander verbunden werden. Auf diesem Weg werden sämtliche Kleinbuchstaben generiert. Die daraus entstehende Schriftart wirkt am Anfang wie eine gebrochene Schrift. Dies ist aber nur der erste Schritt in diesem Prozess.

Das harmonische Model bietet die Möglichkeit sämtliche Parameter zu nutzen. Der Durchmesser des Kreises in der Mitte, stellt dabei die x-Höhe da. Die Ausbreitung der vertikalen Balken definiert nach oben hin die Oberlänge

und nach unten hin die Unterlänge. Die vertikale Skalierung und die Neigung können ebenfalls angewendet werden. Ein Unterschied zu den Großbuchstaben ist jedoch wie die Parameter der Pinselform sich auswirken. Neigung und Breite können in dem Modell über die Extrusion und dessen Winkel dargestellt werden. Da sich die Form aus Flächen zusammensetzt ist die Pinseldicke nicht ohne weiteres darstellbar.

3.2 Fontstruct

Fontstruct ist ein webbasierter Schrifteditor aus dem Hause FontShop, der 2008 veröffentlicht wurde. Dieses, auf den Vertrieb von Schriften spezialisierte Unternehmen, wurde 1989 vom Ehepaar Spikermann und Naville Brody gegründet. Fontstruct unterscheidet sich stark von herkömmlichen Schrifteditoren. Die Glyphen werden dabei wie in einem Rasterfont gezeichnet. Jede Glyphe hat dabei einen Raster in dem einzelne Felder belegt werden können. Der Unterschied zu den Rasterfonts besteht darin, dass der Raster keine binäres Feld ist, sondern ein Muster aufnehmen kann. Diese Muster werden von dem Editor zu Verfügung gestellt [22]. Abbildung 3.4 zeigt den Aufbau einer Glyphe. Auf der linken Seite ist eine Liste mit Kacheln die in den Raster gesetzt werden. Auf diese Art können Glyphen konstruiert werden.

Fontstruct ist ein sehr einfacher Ansatz. Die dabei entstehenden Fonts sind von ihrer Form her sehr konstruiert, da über den Raster kaum anderen Formen zu gelassen werden. Einzelne Teile können einfach per Verdopplung an andere Stelle wiederverwendet werden.

3.3 Kalliculator

Kallicular ist ein Programm, das versucht kalligrafische Ansätze in digitaler Form umzusetzen [24]. Die Informationen über dieses Projekt sind eher spärlich, da es nie veröffentlicht wurde. Es wird wie in einem *stroke Font* ein Skelett verwendet, auf den ein Pinselstrich angewendet wird. Dieses Skelett wird mittels Bézierkurven beschrieben. Der Pinsel der darauf angewendet wird, kann über mehrere Parameter verändert werden [25]. Wie in METAFONT ist es möglich den Winkel des Striches bei den Punkten zu verändern. Die Glyphen werden offensichtlich wie in METAFONT als Bild ausgegeben.

Vor allem das Verändern der Eigenschaften der Punkte ist ein interessanter Punkt in diesem Projekt. Es wird versucht die kalligraphischen Eigenschaften eines Pinsels nachzuahmen und diesen möglichst genau zu definieren.

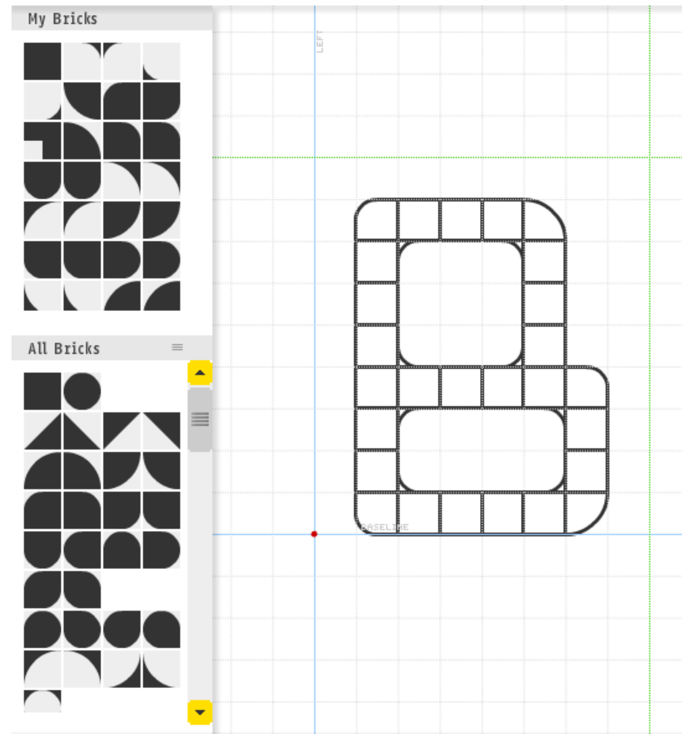


Abbildung 3.4: Glyphen einer Fontstruct Schrift.

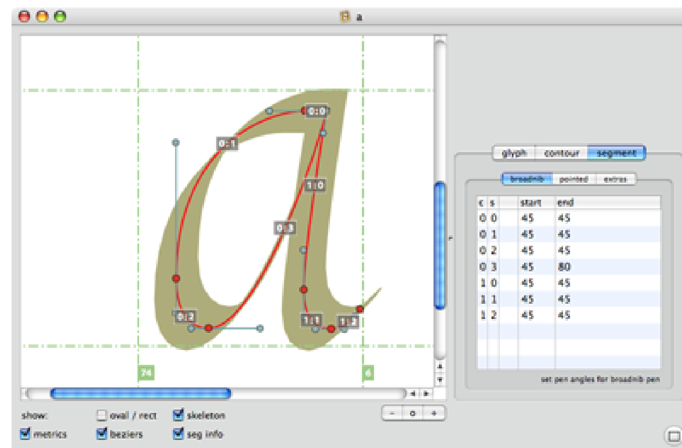


Abbildung 3.5: Lettersoup Editor mit verstellbaren Parametern.

3.4 Lettersoup

Lettersoup ist ein Projekt, das sehr stark ähnliche Ansätze wie diese Arbeit verfolgt. Ricardo Lafuente präsentierte 2008 einen in Phython geschrieben

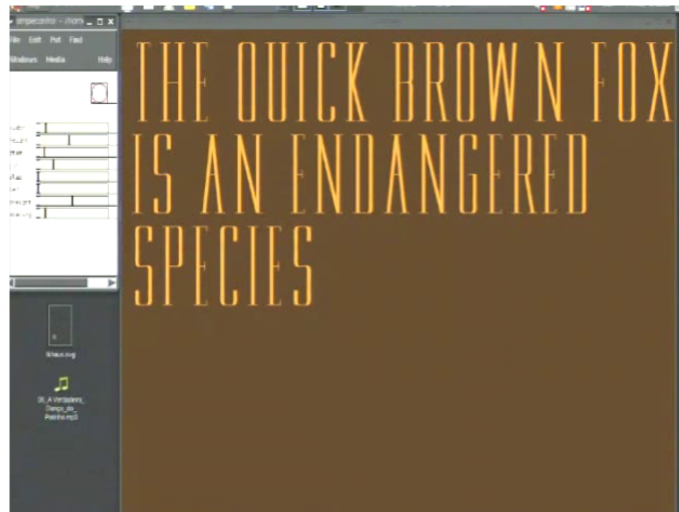


Abbildung 3.6: Lettersoup Editor mit verstellbaren Parametern.

Editor, mit dem es möglich ist, über mehrere Parameter eine Schrift zu generieren [37]. Dabei ist der Editor in der Lage aus mehreren Teilen eine Schrift zusammensetzten.

Es gibt eine Reihe von vorraffinierten Strukturen und Variablen die als Eingabe definiert werden müssen. Es werden zum Beispiel Striche, Querstriche und Diagonalen als Grundstrukturen verwendet und Variablen wie die Stärke des Striches und die Dicke in verschiedenen Ausrichtungen [26].

Mit Hilfe einer *Blaupause*, die Informationen über die Zusammensetzung dieser Teile hat, kann der Editor die Schrift erstellen. Diese Blaupause ist eine Beschreibungssprache, mit der angegeben wird an welche Seite eines Striches sich einen Serife befindet oder wie große einzelne Elemente sind. Diese Sprache ist von seiner Art her an METAFONT angelehnt aber besitzt keine Zeichenfunktionen. Sie dient rein zur Beschreibung der Zusammenhänge. Der Editor kann im Nachhinein diese Strukturen über Parameter verändern und den Font anzupassen.

Das Projekt selbst ist als Download verfügbar und wurde das letzte mal 2009 aktualisiert. Es gibt keine Informationen ob die Entwicklung weitergeführt wird oder nicht.

Vor allem die Möglichkeit einzelne Teile zu definieren und dann zusammenzufügen ist ein spannender Punkt. Einzig der sehr einfach gehalten Aufbau von Buchstaben wirkt sehr unausgereift.

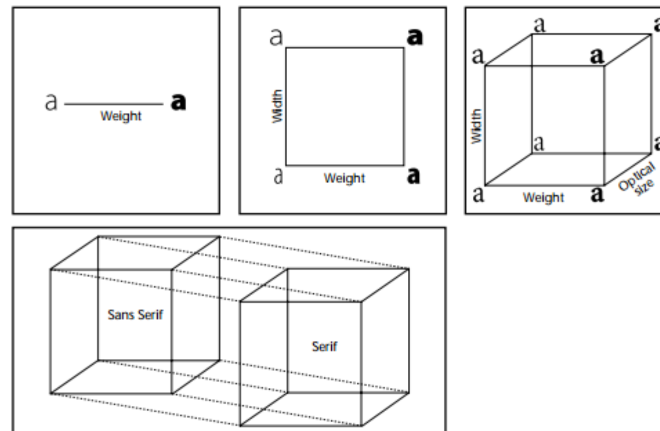


Abbildung 3.7: Beispiel für Multiple master Achsen [8, S. 10].

3.5 Multiple master

Multiple master Fonts sind eine Entwicklung von Adobe die dazu dienen sollte, dass Schriften flexibel aufgebaut werden können. Eine Schrift kann über mehrere sogenannte Achsen verändert werden. Diese Achsen stellen im Normalfall Eigenschaften wie Dicke oder Breite dar. Bei einer Achse wird zwischen zwei Schriften interpoliert die jeweils den Extrempunkt darstellen [8]. Mit Multiple master Schriften ist es möglich verschiedene Schnitte selbst zu erzeugen. Abbildung 3.7 zeigt Multiple master Achsen. In diesem Beispiel kann zwischen Breite, Strichstärke, optischer Dicke und Serifen interpoliert werden.

Die Technologie wurde von Adobe eingestellt und wird heute nicht mehr weiterentwickelt. In vielen Programmen fehlt die Unterstützung um diese Achsen zu nutzen. Für Schriftgestallter spielt es nach wie vor eine Rolle, um verschiedene Schnitte aus einer Schrift zu errechnen. Diese werden dann ohne die Multiple master Achsen gespeichert und in einem normalen Schriftformat verteilt.

Ansätze von Multiple master sind besonders wegen der Interpolation sehr interessant. Die Dokumentation beschreibt wie Kurven am besten Angelegt werden sollten, damit diese möglichst keine Fehler erzeugen.

3.6 METAFONT

Ein weiterer interessante Technologien im Bereich digitale Typographie ist METAFONT. Es handelt sich dabei um eine von Donald Knuth entwickelte Beschreibungssprache für Schriften. METAFONT wurde 1979 erstmals veröffentlicht und wenige Jahre später, 1983, in einer neuen komplett überarbeiteten Version zu Verfügung gestellt. METAFONT wurde als Ergänzung zum Textsatzsystem TeX entwickelt. Wie auch schon bei TeX versuchte Donald Knuth ein möglichst perfektes System zu schaffen. Dabei arbeitetet er mit bekannten Schriftgestaltern wie Matthew Carter (Verdana, Tahoma) oder Hermann Zapf (Optima, Palatino) zusammen. METAFONT unterscheidet sich in einigen Punkten Grundlegenden von anderen Ansätzen wie PostScript, sowohl von der Art wie einzelne Glyphen gezeichnet werden, als auch von der Art wie diese an das Ausgabegerät übergeben werden.

Der folgende Abschnitt soll dabei einen Einblick in METAFONT geben. Viele Ideen werden im Laufe dieser Arbeit in ähnlicher Form wieder vorkommen. Aus diesem Grund wird METAFONT besonders genau beschrieben. Den gesamten Umfang der Sprache hier wieder zu geben ist unmöglich. Für eine genaue Referenz zu einzelnen Funktionen ist das *„The METAFONT Book“* von Donald Knuth zu empfehlen [11].

3.6.1 Zeichenfunktionen

Einer der größten Unterschiede von METAFONT zu anderen vergleichbaren Sprachen sind die Zeichenfunktionen. Donald Knuth hat für METAFONT eine sehr mathematische Form der Beschreibung von Glyphen gewählt. Jeder Buchstabe besteht aus einer Reihe von Gleichungen. Diese erlauben eine sehr exakte Beschreibung von Formen. Der Nachteil besteht darin, dass es bis heute keinen visuellen Editor für METAFONT gibt. Somit muss zwangsläufig alles per Code erzeugt werden. Dies ist besonders für Schriftgestalter eine große Umstellung, was auch ein Grund für mangelnde Akzeptanz von METAFONT sein könnte. Jedoch mindert dies nicht dem Umstand, dass METAFONT über sehr umfangreiche und ausgeklügelte Zeichenmethoden verfügt.

3.6.2 Kurven

METAFONT verwendet nicht Anker- und Kontrollpunkte von Bézierkurven, die sonst üblicherweise verwendet werden, um Kurven zu beschreiben. Vielmehr hat sich Donald Knuth ein System überlegt, das erlaubt, mit möglichst wenig Punkten, das gewünschte Ergebnis zu erzielen. Dazu muss METAFONT einiges an *Intelligenz* zu Verfügung stellen. Es werden dabei MPC (*most pleasing curve*) Prinzipien verwendet [6]. Diese von Donald Knuth entwickelten Prinzipien legen fest, wie der optisch beste Pfad einer Kurve, die durch eine gegebene Anzahl von Punkten verläuft, auszusehen hat. Es können dadurch

Programm 3.1: Quadrat mit der Seitenlänge von 9pt

```
 1 beginchar("Square",9pt#,9pt#,0);
 2   z1 = (0, 0);
 3   z2 = (9pt, 0);
 4   z3 = (9pt, 9pt);
 5   z4 = (0, 9pt);
 6   pickup pencircle scaled 1pt;
 7
 8   // Curve A
 9   draw z1 -- z2 -- z3 -- z4 - cycle;
10
11   // Curve B
12   draw z1 .. z2 .. z3 .. z4 .. cycle;
13
14   // Curve C
15   draw z1 .. z2 .. z3 --- z4 .. cycle;
16
17   // Curve D
18   draw z1{dir45} .. {dir315}z2 .. z3 --- z4 .. cycle;
19 endchar;
```

sehr komplexe Kurven mit sehr wenig Information erzeugt werden. Dies lässt sich am besten anhand eines kleinen Beispiels zeigen, indem dieselben vier Punkte jeweils mit einer unterschiedlichen Anweisung verbunden werden.

Im Programmcode 3.1 werden in Zeile 2 bis 5 die Punkte $z1$ bis $z4$ definiert. In Zeile 9 wird METAFONT, mittels der *draw*-Funktion, nun mitgeteilt, dass diese Punkte mittels einer geraden Linie verbunden werden sollen (*Curve A*). Dies geschieht über die Anweisung „–“ zwischen den einzelnen Punkten. Die doppelten Bindestriche drücken dabei explizit aus, dass es sich um eine gerade Linie handeln soll und METAFONT dies nicht abrunden darf. Zusätzlich steht am Schluss des Befehls noch ein *cycle*. Dies drückt aus, dass der letzte mit dem ersten Punkt verbunden werden soll um eine geschlossene Kurve zu erzeugen. Die Ausgabe entspricht, wie zu erwarten ist, einer Box (Abb. 3.8 (a)).

Diese simple Ausgabe bietet offensichtlich noch keinen besonderen Vorteil gegenüber anderen Sprachen. Der große Unterschied wird erst dann ersichtlich, wenn es darum geht dem METAFONT ein bisschen *Intelligenz* abzuverlangen.

Betrachtet man nun die Ausgabe der Zeichenanweisung, die von *Curve B* erzeugt wird, sieht man, dass diese einen Kreis gezeichnet hat. Der Unterschied zu *Curve A* ist, dass nun die Punkte mittels der Anweisung („..“) verbunden werden. Dies teilt METAFONT mit, dass alle Punkte nach dem *MPC*-Verfahren verbunden werden sollen. Wie in Abbildung 3.8 (b) ersichtlich ist, zeigt die Ausgabe einen Kreis, bei dem alle vier Punkte Kreispunkte

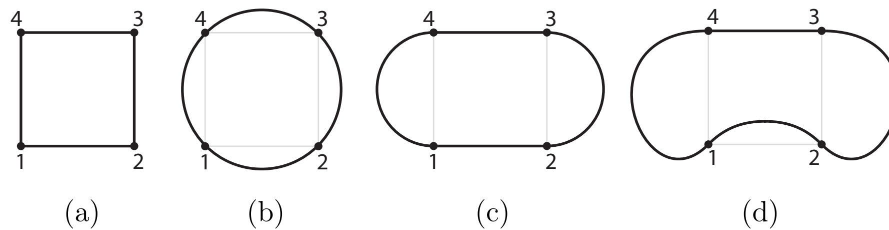


Abbildung 3.8: Skalierungs-Modifikator für die Rundung des Buchstaben *P*.

sind.

Es können auch komplexere Formen erzeugt werden, wie *Curve C*. Hier wird METAFONT mitgeteilt, dass Punkt *z3* und *z4* mittels einer gerader Linie verbunden wird und dass die Enden dieser Linie unter unendlicher *Spannung* stehen. Diese *Spannung* verhindert, dass METAFONT eine Kante an den Endpunkten erzeugt. Somit werden die Punkte *z2* und *z3*, gemäß der MPC Prinzipien, mittels einer Rundung verbunden (Abb. 3.8 (a)). Das Selbe gilt für Punkt *z4* und *z1*. Der Verlauf durch die restlichen Punkte soll dabei weiterhin automatisch erfolgen. Die letzte Kurve (*Curve D*) erweitert Anweisung (c) um den Zusatz, dass die Kurve Punkt *z1* im Winkel von 45° und Punkt *z2* im Winkel von 315° passieren soll (Abbildung 3.8 (d)).

Wie man an diesen vier Beispielen erkennen kann, ist METAFONT in der Lage sehr unterschiedliche Kurven aus denselben Punkten zu erzeugen unter Hilfestellung von Zusatzinformation, wie diese verbunden werden sollen. Yannis Haralambous beschreibt seine Erfahrung mit METAFONT so, dass die Ergebnisse oft besser sind, je weniger Information zur Verfügung gestellt werden [6].

METAFONT verfolgt bei der Definition von Formen den Ansatz, dass diese so angelegt werden sollten, wie es auch per Hand geschehen würde. So ist aus METAFONT-Schriften immer ersichtlich, dass es sich bei der Glyphe eines „X“ um zwei gekreuzte Linien handelt. Bei PostScript würde diese Form in eine einzige Kontur verwandelt werden, aus der der logische Aufbau nicht mehr ersichtlich ist. In den meisten Fällen ist dies zu vernachlässigen, doch hat METAFONT den Vorteil über wesentlich mehr logische Informationen über die Glyphen zu verfügen als PostScript.

3.6.3 Pens

Um einer Kurve eine bestimmte Dicke oder Fülle zugeben, werden in METAFONT sogenannte *pens* verwendet. Dabei handelt es sich um eine Art Pinsel, der mit einer bestimmten Form entlang der Kurve geführt wird. Es wird dabei jeder Pixel eingefärbt, welcher sich am Weg unter der Pinselform befindet.

METAFONT stellt ein Set von vorgefertigten *pen*s zur Verfügung, die sich oft an die Form von kalligrafischen Federspitzen orientieren. Der grundlegende Unterschied zu Outline-Systemen ist, dass keine Pfade den äußeren Rand beschreiben. Dies bedeutet wiederum, dass die daraus entstehende Form nicht einfach durch Bézierpfade beschrieben werden kann, da es keine mathematischen Formeln dafür gibt. Über Umwege ist dies zwar zum Teil möglich, wie im Abschnitt 3.6.5 gezeigt wird, jedoch existiert keine direkte Umwandlung die verlustfrei ist.

3.6.4 Parametrisierung und Makros

Da es sich um eine mathematische Beschreibung handelt, ist es auch möglich Schnittpunkte von Kurven oder Linien als Kurvenpunkte zu definieren. Auf diese Art lassen sich alle möglichen Kurven und Linien erstellen. Es ist somit möglich über mehrere Parameter eine Schrift komplett zu verändern. Werte wie Dicke von Strichen oder Skalierung von Buchstaben kann über Parameter definiert werden. Dies führt zu sehr komplexen Systemen. Donlad Knuth meint zu dieser Tatsache: *„Asking an artist to become enough of a mathematician to understand how to write a font with 60 parameters is too much. Computer scientists understand parameters, the rest of the world doesn't"* [1, S. 361].

Wie in viele Programmiersprachen ist es auch in METAFONT möglich, dass Teile eines Programmcode in einer eigenen Struktur zusammenzufassen werden können. *Definitions* funktionieren wie Makros, werden daher auch in einigen Fällen so bezeichnet. Formen wie Serifen können einfach in ein Makro zusammengefasst werden, das an mehreren Stellen mit den richtigen Parametern eingesetzt wird. Auf diesen Weg können gleiche Strukturen wiederverwendet werden.

3.6.5 Derivate

Im Laufe der Zeit sind rund um METAFONT einige neue Projekte entstanden. Diese versuchen die ursprüngliche Sprache zu verändern oder die erzeugten Schriften in einem anderen Format anzubieten. METATYPE und MetaFog sind zwei Projekte die versuchen METAFONT-Schriften umzuwandeln. Dabei wird von METATYPE das TrueType Format bevorzugt. MetaFog hingegen kann sowohl TrueType als auch PostScript Type 1 Kurven erzeugen [10]. Bei dieser Umrechnung muss die erzeugte Kurve interpoliert werden, da es keine mathematische Formel für diese Umwandlung gibt. Dies bedeutet wiederum, dass es zu Rechenfehler kommen könnte und der erzeugte Font nicht genau identisch mit seinem METAFONT Pendant ist.

Kapitel 4

Eigener Ansatz

Der folgende Abschnitt soll einen Einblick in die grundlegenden Überlegungen dieser Arbeit geben. Einige der angesprochenen Ideen sind nicht gänzlich neu, sie werden jedoch in einer erweiterten oder abgeänderten Form abgehandelt. Der Unterschied zu vielen gängigen Ansätze ist, dass kein vollkommenes Werkzeug zur Gestaltung von Schrift entstehen soll.

Das Ziel ist es auf Grund einer Analyse bestehender Schriften und bereits vorgestellter Technologien einen Ansatz zu finden, wie einzelne Teile einer Schrift wiederverwertet bzw. automatisiert werden können. Es wird immer versucht ein möglichst flexibles System zu schaffen. Eine technische Kompatibilität mit bestehenden Werkzeugen einzuhalten ist einer der wichtigsten Punkte.

4.1 Editor

Im Zuge dieser Arbeit wurde ein Prototyp entwickelt, der viele der vorgestellten Ansätze implementiert und diese auf ihre Realisierbarkeit überprüft. Der Prototyp ist ein visueller Editor in dem mehrere Glyphen angelegt werden können. Diese werden mit Pfaden gezeichnet und mittels den, in diesem Kapitel vorgestellten, Techniken bearbeitet oder verändert.

Abbildung 4.1 zeigt den schematischer Aufbau des Editors. Er ist in verschieden Bereiche unterteilt, die für sich je ein eigenes System darstellen. Der Hauptbereich ist dabei der Glyphen-Editor, der über eine Zeichenfläche verfügt. Für jeden Buchstaben kann eine eigne Glyphe angelegt werden. Es können Pfad angelegt und bearbeitet werden. Weiters verfügt dieser Glyphen-Editor über Linien die das Liniensystem repräsentieren.

Die genaue Beschreibung der anderen Einzelteile des Editors erfolgt in den folgenden Abschnitten. Dort wird deren Funktionalität und Verwendungszweck beschrieben.

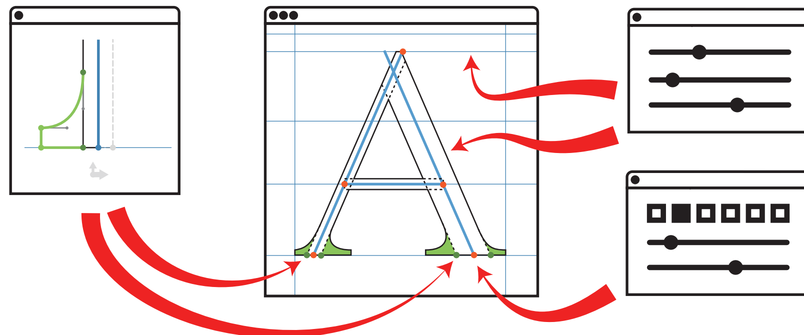


Abbildung 4.1: Schematischer Aufbau des Editors.

4.2 Technische Einschränkungen

Der Ansatz umfasst nicht alle Themen die zur Erstellung einer digitalen Schrift gehören. Der Focus dieser Arbeit liegt rein auf der Erstellung von Glyphen. Viele Teilbereiche wurden bewusst ausgelassen um den Umfang dieser Arbeit in Grenzen zu halten. Themen wie *Hinting* und *Kerning* sind zwar wichtig damit eine Schrift einen guten optischen Eindruck macht, jedoch haben diese nichts dem eigentlichen Ziel dieser Arbeit zu tun. Die entstehenden Lösungen sollen dabei möglichst einfach zu verwenden sein. Es soll ein Mittelweg gefunden werden zwischen leichter Bedienbarkeit und einem flexiblen System.

Eine Einschränkung dieser Arbeit ist technischer Natur. Die Resultate dieser Arbeit sollten in einem gängigen Schrifteditor implementiert werden können. Da so gut wie alle größeren Lösungen in dem Bereich mit Vektor-Schriften arbeiten, ist es notwendig, dass die verwendeten Kurven eine bestimmte Eigenschaft haben. Damit diese verlustfrei weiterverarbeitet werden könnten ist zwingen nötig, dass alle entstehenden Strukturen aus Bézierkurven bestehen. Diese sollte auch nicht höher als des dritten Grades sein, damit eine Konvertierung zu PostScript-Kurven möglich ist.

4.3 Beispielschrifen

Viele der folgenden Überlegungen verwenden eine Analyse bestehender Schriftarten als Grundlage. Dieser Weg wird auf Grund mehrerer Tatsachen gewählt. Grundsätzlich gibt es keine fixen Regeln oder Gesetze, die beschreiben wie eine Schrift auszusehen hat. Viele Werke über Schriftgestaltung gehen hier ähnliche Wege wie zum Beispiel Karen Cheng [4]. Die Verwendung von bestehenden Schriften sollte außerdem sicherstellen, dass die getroffenen Annahmen auch in der Praxis vorkommen.

Tabelle 4.1: Beispielschriften nach Klassen.

Klasse	Schrift
Egyptienne	Rockwell Std, Serifa Std, Egyptienne F
Antiqua	Adobe Garamond Pro, Palatino LT Std, Times
Groteske	Helvetica Neue LT Std, Univers LT Std, Frutiger LT Std

ABC abc ABC abc ABC abc
ABC abc ABC abc ABC abc
ABC abc ABC abc ABC abc

Abbildung 4.2: Verwendete Beispielschriften.

Die Auswahl, der Beispielschriften für die Analyse, erfolgt dabei nach mehreren Gesichtspunkten. Die Schriftarten sollten von namhaften Personen erstellt worden sein, die für ihre Typographien bekannt sind. Dies soll sicherstellen, dass die Schriften nach typographischen Regeln und mit einer gewissen Sorgfalt erstellt wurden. Weiteres Kriterium für die Auswahl ist, dass die Schriftart eine gewisse Verbreitung oder Relevanz hat. Die ausgewählten Schriften sind in Tabelle 4.1 angeführt. Die dabei verwendete Klassifizierung ist nach der Matrix Beinert (Abschnitt 2.3.2) gewählt worden.

Die Unterteilung findet in erste Linie nach den Kategorien statt. Hier wird zwischen den textitGrotesken, textitEgyptienne und den textitAntiqua unterschieden. Die drei Gruppen umfassen die am meisten verwendeten Schriftfamilien und sind optisch voneinander leicht trennbar. In jeder dieser Gruppen befinden sich jeweils drei Beispielschriften die typische Merkmale für diese aufweisen. Innerhalb der Gruppen sollten die einzelnen Schriften voneinander eindeutig unterscheidbar sein, vor allem im Bezug auf die Form der Serifen.

Die gezeigten Schriftbeispiele (Abb. 4.2) korrespondieren dabei mit der Liste (Tab. 4.1). Die vorkommenden Schriftarten gehören Teils zu den bekanntesten Typographien und sind von Schriftgestaltern wie Adrian Frutiger (*Univers, Frutiger*) oder Max Miedinger (*Helvetica*) entworfen.

Die Anzahl der Schriften ist bewusst sehr gering gehalten da viele der auftretenden Merkmale sich innerhalb einer der Schriften gut ablesen lassen können. Die Schriftart gibt dabei nur generelle die Form vor. Der Vergleich der einzelnen Glyphen untereinander liefert in weiterer Folge die nötigen Erkenntnisse.

4.4 Teilbereiche

Die Teilbereiche werden in drei verschiedenen Gruppen von Ansätzen eingeteilt. Es wird zwischen der Konstruktion von Glyphen, der Wiederverwendung von Elementen und Möglichkeiten zur Automatisierung unterschieden. Jeder dieser Bereiche kann dabei mehrere verschiedene Ansätze beinhalten. An manchen Stellen kann es dabei zu Überschneidungen kommen, da auf kommende Abschnitte vorgegriffen werden muss.

4.5 Parameter

Innerhalb einer Schriftart gibt es Abhängigkeiten zwischen einzelnen Elementen. Karen Chang zeigt ihn ihrem Buch *desgning type* einige Beispiele [4]. Eine dieser offensichtlichsten Zusammenhänge hierbei wäre die Größe der Buchstaben, die nach dem Liniensystem ausgerichtet werden, oder dass Strichstärken in einem gewissen Verhältnis zueinander stehen. Der Gedanke ist nun, ob es möglich ist, in einem visuellen Editor ein System zu implementieren, das mathematische oder geometrische Zusammenhänge zwischen Elementen beschreibt. Es muss dabei klar sein, dass hier keine Komplexität wie in METAFONT geschaffen werden kann, aber vielleicht eines in den Grundzügen ähnlich funktionierendes System.

Um diese Abhängigkeit zwischen Elementen zu beschreiben werden zwei verschieden System verwendet. Eines davon sind Parameter. Ein Parameter besteht dabei immer aus einem Namen und einem Wert. Dieser kann verwendet werden um den Eingangswert eines numerischen Attributs zu verändern. Diese Attribute sind zum Beispiel die Dicke eines Striches oder die vertikale Position eines Punktes.

Ein Parameter wird an einer Stelle mit einem bestimmten Namen definiert und mit einem Wert initialisiert. Dieser Name kann an beliebig vielen numerischen Attributen verwendet werden. So kann die Dicke der Striche über die ganze Schrift hinweg geändert werden. Es muss nur ein Parameter erstellt werden, der diesen Wert beinhaltet und dann bei jedem Strich bei dem Attribut für die Dicke angegeben wird. Für die weiteren unterschiedlichen Strichstärken können einfach zusätzliche Parameter erzeugt werden. Die Parameter für die verschiedenen Strichstärken einer Schrift könnten wie in Tabelle 4.2 aussehen.

Dies würde eine einfache Lösung darstellen, um global Änderungen vorzunehmen. Jedoch ändert sich die Strichstärke leicht bei unterschiedlichen Buchstaben, auch wenn es sich um denselben Strichtyp handelt [4, S. 41]. Für diese Anpassung würde jeweils ein neuer Parameter erzeugt werden müssen. Dies wiederum würde zu einer Zunahme von Parametern führen, die eigentlich in einer Abhängigkeit zueinander stehen sollten.

Tabelle 4.2: Beispiele für die Definition von Parametern für Strichstärken.

Name	Wert
Grundstrich	50
Haarstrich	22
Querstriche	20

Tabelle 4.3: Beispiele für die Definition von Parametern mit Ausdrücken für Strichstärken.

Name	Wert
Grundstrich	50
Haarstrich	Grundstrich · 44%
Querstriche	Haarstrich · 90%

4.5.1 Ausdrücke

Um die Funktionalität der Parameter zu erweitern, werden sogenannte Ausdrücke verwendet. Diese bieten die Möglichkeit einfache mathematische Ausdrücke zu formulieren um Zusammenhänge zu beschreiben. Diese Ausdrücke werden dabei in Form von einer Texteingabe realisiert. Diese Eingabe wird evaluiert und ein numerischer Wert erzeugt. Diese Ausdrücke können auch andere Parameter als Wert beinhalten.

Die zusätzliche Möglichkeit, die sich dadurch ergibt, ist, dass eine Anpassung einzelner Attribute möglich wird, ohne dass ein zusätzlicher Parameter dafür erzeugt werden muss. Die Parameter für die Strichstärken könnten wie in Tabelle 4.2 abgeändert werden, ohne dass dass die Veränderung des /textitGrundstrich die anderen Parameter beeinflusst.

Bedeutender ist aber die Möglichkeit die individuelle Anpassung für einzelne Glyphen vorzunehmen. Ist der Strich des *I* in der Schrift immer etwas dünner als der *Grundstrich*, könnte dem Strich des *I* als Attribut für die Dicke ein angepasster Ausdruck zugeteilt werden. Anstatt nur den Parameter *Grundstrich* zu verwenden würde womöglich der Ausdruck: „*Grundstrich* · 98%“ eine brauchbare Lösung darstellen. Dies würde verhindert, dass ein neuer Parameter erstellt werden muss, um einen einzelnen Wert anzupassen.

4.5.2 System Parameter

Neben den selbst definierten Parametern, werden vom System selbst welche zu Verfügung gestellt. Diese Parameter bestimmen die Umgebung. Werte für diese Parameter wären zum Beispiel die Position der Linien, oder die Größe des Gevierts. Diese Parameter werden für jede Glyphe extra berechnet und zur Verfügung gestellt.

4.5.3 Parameter-Editor

Der Prototyp stellt einen Parameter-Editor zu Verfügung. In diesem Editor können neue Parameter angelegt und verändert werden. Jeder dieser Parameter besteht auf zwei Texteingaben. Der Erste stellt den Namen des Parameters dar. Die zweite Eingabe ist der Wert des Parameters. Dieser kann numerische Werte beinhalten oder Ausdrücke. Die im Parameter-Editor erstellten Parameter sind global verfügbar und können überall bei Attributen eingesetzt werden.

Wie bei einer Definition einer Variablen, darf der Wert eines Parameters nicht den Parameter selbst beinhalten. Parameter dürfen jedoch andere Parameter beinhalten. Der Komplexität der Ausdrücke ist im Grunde keine Grenze gesetzt und hängt nur von der Implementierung ab.

4.6 Striche und Kurven

Striche stellen den Grundbaustein dar, aus dem alle Glyphen erzeugt werden. Bei dem Aufbau der Striche wird versucht sich an kalligraphische Ansätze zu halten [7, 12]. Ziel ist es die Konstruktion einfach zu gestalten und für Veränderungen offen zu lassen. Es soll so viel Information wie möglich über den Aufbau erhalten bleiben.

4.6.1 Punkte und Kurven

Die Konstruktion der Striche erfolgt über Bézierkurven. Dies geschieht wie in den gängigsten Grafikprogrammen[1] und Schriftgestaltungssoftware[2] [21, 23]. Die Bézierkurven sind dabei vom dritten Grad, das heißt sie verfügen über zwei Anker- und zwei Kontrollpunkte. Diese Punkte verfügen über eine Position im Raum, die den vertikalen und horizontalen Abstand zum Ursprung des Gevierts angibt. Diese zwei Werte sind numerisch und können als Parameter gesetzt werden. Die Ankerpunkte verfügen, gegenüber den Kontrollpunkten, noch über weitere Eigenschaften. Diese stellen den Anfang und das Ende eines Segments einer Kurve dar. In der Verbindung von zwei Segmenten teilen sich diese je einen Ankerpunkt. Damit der Verlauf der Kurve flüssig ist, muss der letzte Kontrollpunkt eines Segments mit dem Ankerpunkt am Ende, der gleichzeitig der Ankerpunkt am Anfang des nächsten Segments ist, und dem ersten Kontrollpunk des nächsten Segments auf einer Achse liegen. Der Kontrollpunk hat zwei zusätzliche Einstellungsmöglichkeiten um dies zu berücksichtigen, wie in Tabelle 4.4 angeführt wird.

[1] *Adobe Illustrator, Photoshop, InkScape, CorelDRAW.*
[2] *Glyphs, FontLab.*

Tabelle 4.4: Eigenschaften eines Kontrollpunkts

Wert	Bedeutung
x	vertikale Abstand zur Ursprungsposition
y	horizontaler Abstand zur Ursprungsposition
kontinuierlich	Kontrollpunkte liegen mit Ankerpunkt auf einer Achse
symmetrisch	Kontrollpunkte liegen mit Ankerpunkt auf einer Achse und haben den selben Abstand von diesem

4.6.2 Stiche

Im Unterschied zu Vektorschriften, soll die Dicke des Striches nicht über die Erstellung von zwei verbunden Kurven erfolgen, sondern automatisch berechnet werden. Um einen Strich mit Dicke zu erzeugen, muss ein Pfad angelegt werden. Dieser verläuft in der Mitte des Striches. Nachdem der Pfad angelegt wurde, kann dieser mit einer Dicke versehen werden. Diese Dicke stellt einen numerischen Wert dar, der per Parameter gesetzt werden kann. Die Form des Striches wird automatisch berechnet und als Form ausgegeben. Diese Form besteht dabei aus Bézierkurven. Dies bedeutet, dass der entstehende Strich kompatibel zu PostScript ist und könnte somit in einem anderen Programm weiterverarbeitet oder als PostScript-kompatibler Font abgespeichert werden.

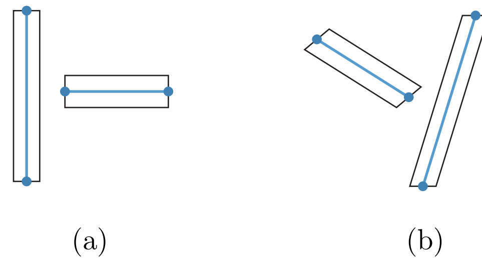
Ein Pfad der mit einer Dicke versehen wird, wird zu einem Strich. Die aus dem Pfad erzeugte Form des Striches kann nicht direkt bearbeitet werden. Nur die Verschiebung eines Pfadpunktes oder die Änderung der Dicke führt zu einer Veränderung.

4.6.3 Stricharten

Für die Konstruktion der Buchstaben werden verschieden Arten von Strichen zur Verfügung gestellt. Jeder dieser unterschiedlichen Stricharten verfügt über gewisse Eigenschaften die benötigt werden um einen bestimmten Teil einer Glyphe zu erzeugen. Es gibt dabei verschiedene Anforderungen. So müssen die Striche in gewissen Fällen eine bestimmte Breite und Winkel der Feder haben. Es werden zwei verschiede Arten von Strichen vorgestellt mit denen man sich eine Reihe von Schriften erstellen lassen kann.

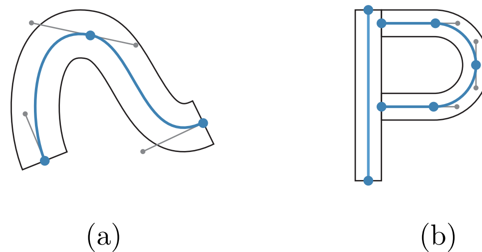
Gerade Striche

Gerade Striche sind jene Strich die über einen fixen Winkel der Feder verfügen. Der Winkel der Feder ist im Pfadverlauf immer der Selbe. Diese Striche werden vor allem für die Konstruktion der Großbuchstaben benötigt. Abbil-



(a) (b)

Abbildung 4.3: Striche mit einer fixen Feder.



(a) (b)

Abbildung 4.4: Striche mit einem runden Verlauf.

dung 4.4 (a) und (b) zeigen je ein Beispiele für diese Art. Bei dieser Art von Strichen regelt ein Parameter den Winkel der Feder. Ein zweiter Parameter wird verwendet um die Federbreite zu bestimmen.

Runde Striche

Runde Strich werden als jene bezeichnet, deren Feder immer normal zu dem innen liegenden Pfad steht. Die Außenkanten haben immer denselben Abstand zum Pfad.

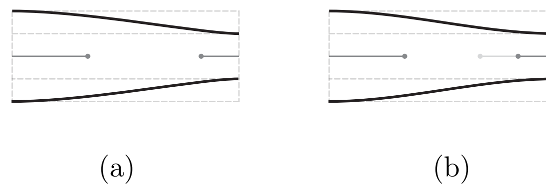
Abbildung 4.4 (a) zeigt einen runden Strich. Diese Art von Strichen wird für Bögen benötigt, wie in Grafik (b) bezeigt wird. Runde Striche haben nur einen Parameter, dieser regelt die Federbreite. Der Winkel wird durch den Pfad bestimmt und kann nicht verändert werden.

4.6.4 Veränderungen der Dicke

Die Veränderung der Dicke eines Striches ist ein wesentlicher Punkt, der in den meisten runden Glyphen zu finden ist [4]. Oft sind diese Unterschiede kaum zu erkennen, doch bei näherer Betrachtung können diese sichtbar gemacht werden. Abbildung 4.5 (a) zeigt drei Beispiele in denen eine Bogen vorkommt. Der Strich ist oben und unten dicker als auf den Seiten. Ein Vergleich in Grafik (b) zeigt dies anhand eines *O*. Dieselbe Glyphe wird einmal

(a) (b)

Abbildung 4.5: Vergleich von unterschiedlichen Strichdicken.



(a) (b)

Abbildung 4.6: Verlauf der Strichdicke im Vergleich.

normal und einmal um 90° gekippt übereinander gelegt. Der Unterschied ist deutlich sichtbar.

Um dieses Verhalten nachzuahmen, müssen runde Striche um eine Fähigkeit erweitert werden, diese erlaubt die Dicke des Striches zu ändern. Diese Änderung der Strichstärke findet dabei zwischen den Ankerpunkten statt. Jeder Punkt der Kurve verfügt über eine eigene Dicke die verändert werden kann. Zwischen diesen einzelnen Punkten wird die Dicke angepasst. Der Verlauf dieser Anpassung kann entweder von der Position der Kontrollpunkte abhängen, oder auch kontinuierlich sein. Abbildung 4.6 (a) zeigt den Verlauf unter Berücksichtigung des Abstandes der Kontrollpunkte. Dieses Verhalten kann zu einem Problem führen, wenn Kontrollpunkte zu nah an den Ankerpunkten sind. Grafik (b) verwendet einen kontinuierlichen Verlauf. Bei diesem Beispiel wird angenommen, dass die Kontrollpunkte den selben Abstand zueinander haben wie zu den Ankerpunkten.

Bei der Verbindung von mehreren Segmenten, bei dem die Punkte eine unterschiedliche Dicke haben, wird kein kontinuierlicher Verlauf des Pfades erzeugt. Abbildung 4.7 zeigt dieses Verhalten anhand von zwei Segmenten. Dieses Verhalten wurde bewusst gewählt um die Komplexität zu vermindern. Viele Bögen lassen sich trotz dieser Einschränkung erstellen.

4.6.5 Limitierungen

Die Möglichkeit mit dieser Art der von Striche Glyphen zu zeichnen ist sehr limitiert. Es reicht in vielen Fällen für groteske oder sehr konstruierte Schriften wie *Rockwell, Helvetica, Frutiger*. Für mehr handschriftliche Formen, die

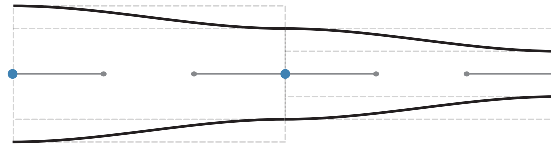


Abbildung 4.7: Verlauf der Strichdicke im Vergleich.

eine Veränderung des Federwinkels beinhalten, ist es schwer eine passende Strichform zu berechnen, da es keine direkte mathematische Formel gibt. Die individuelle Drehung der Achse in einem Strich erlaubt es nicht eine einfache Kurve mit einem Abstand zu berechnen. Die Bestimmung dieser Kurve ist eine komplexe Aufgabe die den Rahmen dieser Arbeit sprengen würde und wird nicht behandelt. Es ist auch nicht wichtig für die Funktionalität des Prototyps, da es hier um die Erstellung von veränderbaren Schriftprototypen geht. Das Vorhanden sein von Kurven, die veränderbare Federwinkel zulassen, würde die Funktionalität erweitern aber betrifft nicht die Grundidee hinter dieser Arbeit.

4.7 Wiederverwendung von Strukturen

Dieses Kapitel dreht sich um die Frage, wieweit es möglich ist, einzelne Strukturen die in einer Schrift vorkommen, wiederverwendbar gemacht werden können. Dieser Ansatz ist bei weitem nicht neu und findet sich in der einen oder anderen Form immer wieder. Viele dieser bisherigen Ansätze sind jedoch sehr statisch wie zum Beispiel bei der Verwendung von Mustern, die als Vorlage immer wieder eingesetzt werden. Dies erleichtert zwar den Arbeitsprozess jedoch bedarf es meistens einer zusätzlich manuellen Anpassung. Bei METAFONT-Schriften ist es dank der umfangreichen Beschreibungssprache möglich, diese Strukturen zu automatisieren. Dafür sind jedoch umfangreichen Programmierkenntnissen nötig.

Der Unterschied der hier gewählten Ansätze zu bereits bestehenden ist, dass es eine einfach Möglichkeit gibt eine Vorlage für wiederkehrende Strukturen zu definieren. In dieser Vorlage ist es möglich, die Struktur an verschiedenen Umgebungsparameter anzupassen. Aus der Vorlage können Instanzen erstellt werden, die an einer Glyphe platziert werden. Diese Instanz versucht sich automatisch an die Umgebung anzupassen indem es die vordefinierten Zustände aus der Vorlage verwendet. Sollten die Umgebungsparameter nicht exakt in der Vorlage vorkommen, wird versucht zwischen den ähnlichen Zuständen eine passende Form zu errechnen.

Die Vorteile die dadurch entstehen würden, sind einerseits ein wesentlich schnellere Arbeitsprozess, da das Kopieren und manuelle Anpassen dieser Formen wegfällt und anderseits, dass Veränderungen im Nachhinein einfach

realisierbar sind, da sich Änderungen der zentralen Vorlage sofort auf sämtlichen Instanzen auswirken.

Das Thema Serifen wird besonders beachtet. Diese gehören zu den am häufigsten wiederkehrenden Strukturen vieler Schriften. Sie prägen sehr stark das Erscheinungsbild, was sie zu einem wichtigen Gestaltungsfaktor machen.

4.7.1 Analyse

Als erstes soll eine Analyse bestehender Schriftarten einen Aufschluss geben welche Veränderungen überhaupt möglich sind. Zu diesem Zweck werden einzelne Glyphen einer Schrift miteinander verglichen. Es wird analysiert in welcher Form Serifen und andere Abschlüsse vorkommen und welche Umgebungsparameter einen Einfluss auf ihre Form haben.

Um sicherzustellen, dass bei den verwendeten Positionsangaben keine Verwirrung entsteht wird eine einheitliche Terminologie verwendet. Wenn also in den folgenden Abschnitten von Innen- und Außenkante die Rede ist, ist dabei die Oberseite und Unterseite von Serifen gemeint, die an einem vertikalen Strich liegen und sich an dessen Unterseite befinden. Serifen die im 90° Winkel vorkommen müssen somit gedreht betrachtet werden.

Serifen werden auf Grund ihrer Form, in eine Reihe von Kategorien eingeteilt. Im englischsprachigen Raum gibt es eine sehr genaue Definition für diese Einteilung, die teils auch im Deutschen verwendet wird. Diese wird jedoch nicht herangezogen. Für diese Analyse sind andere Eigenschaften wichtig. Zwischen Serifenformen, deren optischer Unteschied hier keine Rolle spielt, wird nicht differenziert.

Die hier vorgenommene Unterteilung unterscheidet zwischen vier verschiedene Gruppen. In dieser Einteilung werden unter *geraden Serifen* vor allem jene bei Egyptienne oder klassizistischer Linear-Antiqua vorkommenden Serifen zusammengefasst, deren Form nicht mit einer Kehlung in den Strich übergeht. Die Innen- und Außenkante dieser Serifen sind dabei meist parallel oder an der Oberkante leicht geneigt. Als zweite Gruppe werden Serifen mit einer Kehlung behandelt. Diese Form der Serifen ist sehr häufig bei Schriften für den Druck zu finden. Sie zeichnen sich dadurch aus, dass die Serifenform flüssig in den Strich übergeht. Die dritte Gruppe beinhaltet Serifen deren Außenkante geschwungen ist. Dies ist vor allem bei Schriften wie der Palatino sehr auffällig (Abb. 4.2 Mitte). Diese werden hier als *Geschwungene Serifen* eingeordnet. Als letztes werden Serifen die sich meist an der Oberseite einen Striches von Kleinbuchstaben befinden behandelt. Diese Serifen unterscheiden sich deutlich von jenen an der Unterseite oder denen bei Großbuchstaben. Die Form ist meistens unsymmetrische und geht über die ganze Breite des Striches hinweg. Sie sind also im Gegensatz zu den anderen Serifen nicht seitlich angebracht, sondern werden eher aufgesetzt.

Serifen dieser Gruppen können sich von den Merkmalen überschneiden, so können geschwungene Serifen auch eine Kehlung haben. Feststellungen die

über gekehlte Serifen getroffen wird, lassen sich somit auch auf geschwungene Serifen anwenden, sollten diese über eine Kehlung verfügen.

Gerade Serifen

Zu den einfachsten Serifen gehören jene, welche nicht flüssig in den Strich übergehen. Anders gesagt: Die Stelle, an der Innenkante, an der sich Strich und Serife treffen kantig ist. Schriftarten wie zum Beispiel *Rockwell* haben diese Art der Serifen (Abb. 4.2 links oben). Weiters gehören Schriften mit sehr dünnen Haarlinien-Serifen in diese Kategorie. Es gibt einige Fälle in denen die innenliegende Kante der Serife eine leichte Neigung besitzt. Diese Serifen werden ebenfalls als *gerade Serifen* behandelt.

Rein subjektiv wirken Serifen dieser Kategorie so, als wären sie immer gleich groß. Sie weisen für das menschliche Auge keine offensichtlichen Unterschiede in Form oder Größe auf. Dies würde bedeuten, dass eine statische Grundform für alle Serifen in der Schrift genügen würde. Ein näherer Vergleich zeigt jedoch, dass diese bei weiterem nicht so homogen sind, wie es den Anschein hat.

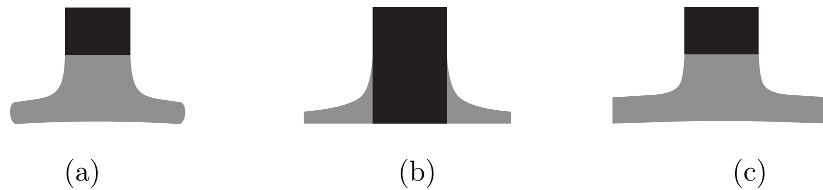
So könnte zum Beispiel vermutet werden, dass die Serifen an dem *H* und großen *I* Identisch sind. Aus Gründen der ähnlichen Form und der Tatsache, dass im Konstruktionsprozess das *H* oft aus dem *I* entsteht, würde diese Aussage untermauern. Eine Gegenüberstellung dieser Glyphen zeigt aber, dass diese Art von Serifen kaum identisch zueinander sind. Zwar weisen alle Serifen dieselbe Höhe auf, doch in der Breite tritt eine hohe Varianz auf.

Die Frage ist nun, welche Faktoren eine Rolle spielen könnten, die solche Unterschiede erklären würden. Bei dem Beispiel mit *I* und *H* sind zwei Unterschiede sichtbar. Erstens weisen beide Glyphen eine unterschiedliche Dimension bezüglich ihrer Dickte auf und zweitens ist ihr Strich unterschiedlich dick. Welche Abhängigkeit hier die wichtigere Rolle spielt oder ob die Breite der Glyphe Auswirkung auf Dicke des Striches hat, kann nicht genau abgelesen werden. Wichtig sind hier aber die Feststellungen, dass es einen dieser Zusammenhänge geben kann. Weiters ist erkennbar, dass die Breite der Serifen von der Richtung abhängt in die sie gerichtet sind. Über den ganzen Font hinweg sind die Serifen in Leserichtung länger als ihre Gegenüber. Dies würde einen weiteren Faktor darstellen, der zu berücksichtigen ist.

Gekehlte Serifen

Neben den *geraden Serifen* gibt es Serifen die eine Kehlung aufweisen. Diese Kehlung bildet einen runden Übergang zwischen der Serife und dem angrenzenden Strich. Diese Rundung ist ursprünglich ein Resultat, das durch die das Schnitzen der Glyphen entstanden ist [13].

Eine große Eigenheit der Serifen betrifft ihr Verhalten bei Strichen mit einer unterschiedlichen Neigung. Je nach Winkel des angrenzenden Striches



(a) (b) (c)

Abbildung 4.8: Unterschiedliche Serifenformen.

hat die Kehlung eine unterschiedliche Rundung, da diese immer einen flüssigen Übergang haben sollte. Aus diesem Grund muss der Auslauf der Kehlung immer mit dem Winkel des Stammes übereinstimmen. Bei einer Gegenüberstellung von Serifen mit unterschiedlich geneigtem Strich wird dies sichtbar.

Der Punkt an dem sich die Kehlung und der Strich berühren ist nicht immer zwingend auf derselben Höhe. Dieser Kontaktpunkt kann je nach Winkel höher oder tiefer (Abb. 4.11) liegen. Anhand der verschiedenen Strichwinkel ist zu erkennen, dass der Punkt seine Position sehr unterschiedlich verändert. Bei 90° sind beide Kontaktpunkte auf derselben Höhe. Bei einer Zunahme des Winkels kann man an der rechten Kante erst eine Ansteigen des Punktes erkennen, bis dieser wieder bei 135° nach unten absinkt. Ähnliches Verhalten zeigt sich an der linken Kante. Bei einer Zunahme steigt der Punkt anfangs, sinkt jedoch sofort nach unten ab bei weiterem Anstieg.

Ein trivialer Zusammenhang zwischen dem Winkel und dem Kontaktpunkt der Kehlung kann, anhand der Beispielschriften, nicht gefunden werden. Es besteht jedoch eine gewisse Relation die berücksichtigt werden kann.

Geschwungene Serifen

Die dritte Gruppe von Serifen die hier behandelt wird, sind sogenannte *geschwungene Serifen*. Die Bezeichnung *geschwungen* bezieht sich dabei auf die Form der Unterkante. Im Unterschied zu den meisten Serifen, deren Unterkante gerade ist und auf einer der Linien im Liniensystem aufliegt, ist diese geschwungen und berührt die Linie nur in wenigen Punkten.

In allen bisherigen Fällen wurden die Serifen nur an einer Seite an den Strich hinzugefügt. Serifen dieser Art haben eine weitere besondere Eigenschaft. Wenn ein Strich auf beiden Seiten eine Serife hat, kann vorkommen, dass beide durch die geschwungene Außenkante verbunden sind und somit eine gemeinsame Form ergeben. Abbildungen 4.8 (a) und (b) zeigen den Vergleich.

Ein weiteres Merkmal zeigt sich bei *Adobe Garamond Pro*. Die Serifen dieser Schrift verfügen über eine Kehlung und eine geschwungene Unterkante. Zusätzlich ist aber auch die Verbindung zwischen Kehlung und Unterkante leicht geschwungen wie in Abbildung 4.8 (c) zu erkennen ist. Zusätzlich

ijrunmdbhk

Abbildung 4.9: Unterschiedliche Serifen an der Ober- und Unterseite bei der Schrift Palatino

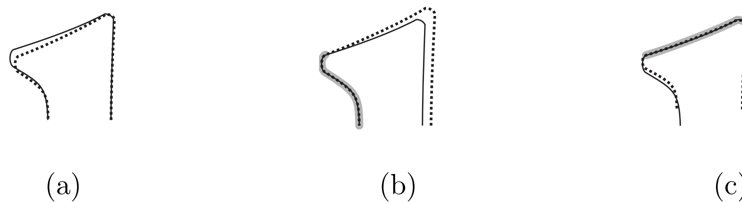


Abbildung 4.10: Veränderung der Form bei Serifen.

werden in Abbildung 4.10 (a) zwei Serifen mit unterschiedlicher Breite verglichen. Der Rückschluss der daraus gezogen werden kann ist, dass es sich wohl nicht um eine einfache Skalierung der Serifen handelt. Gewisse Bereiche der Serife scheinen von ihrer allgemeinen Breite nicht beeinflusst zu werden, wohingegen dies bei anderen sehr wohl der Fall ist.

Unsymmetrisch Serifen

Eine weiter Form von Serifen werden als *unsymmetrische Serifen* eingeordnet. Der Begriff unsymmetrisch wurde deswegen gewählt, da sich diese Serifen dadurch auszeichnen, dass ihre Form immer über die Hälfte des Striches reicht. Es kann somit nur eine Serife an das Ende des Striches gesetzt werden und nicht wie sonst zwei.

Diese Serifen werden hauptsächlich bei Kleinbuchstaben an der Oberseite von Strichen platziert. In manchen Fällen kann es auch vorkommen, dass diese auch auf beiden Seiten angebracht werden. In Abbildung 4.9 können diese Serifen deutlich erkannt werden.

Der Vergleich in Abbildung 4.10 (a) zeigt, dass sich die Form bei gleicher Strichstärke deutlich in der Form unterscheidet. Ein genauerer Vergleich (b) und (c) zeigt, das sich die Form durch Skalierung an den grau unterlegten Stellen anpassen lässt aber nur für einzelne Segmente. Diese Unterschiede lassen sich bei *i*, *l* und *n* ausmachen. Obwohl *i* und *l* ein fast gleiches Geviert haben ist ihre Form unterschiedlich. Weiters ist *i* und n mit derselben Höhe auch unterschiedlich. Dieses Beispiel zeigt, dass es Veränderungen gibt die nicht direkt in den Zusammenhang mit der Umgebung gebracht werden

können. Dies würde bedeuten, dass es zusätzliche Parameter gibt die eine Rolle spielen.

4.7.2 Folgerungen

Aus der Analyse können einige Rückschlüsse gezogen werden. Wichtig ist welche Möglichkeiten, die Vorlage für eine Serife, zu Verfügung stellen muss, damit diese möglichst universal einsetzbar ist. Eine Zusammenfassung der Analyse liefert folgende Liste:

- Die Form der Serifen kann abhängig sein von dem Winkel des Striches, an dem sie sich befinden.
- Die Dicke des Striches oder die Breite der Glyphe kann Auswirkung auf die Form der Serifen haben.
- Auf Grund der Seite an der sich die Serifen am Strich befinden können diese unterschiedlich breit sein.
- Zwei Serifen an entgegengesetzten Seiten eines Striches können eine zusammenhängende Form bilden.
- Eine Serife kann eine unsymmetrische Form haben, die beide Seiten eines Striches verbindet.
- Teile einer Serife können sich mit zunehmender Breite verändern, während manche dies nicht tun.
- Es gibt eine Reihe von Veränderungen deren Ursprung sehr komplex oder rein optischer Natur ist.

4.7.3 Serifen-Editor

Ein sogenannter Serifen-Editor soll die Möglichkeit bieten eine Serife zu definieren, die als Vorlage verwendet werden könnte. Diese Vorlage soll möglichst universell sein. Der Editor soll in der Lage sein, die verschiedenen Umgebung in der die Serife vorkommt, zu simulieren. Damit sich die Serife in späterer Folge anpassen kann, muss diese für die unterschiedliche Umgebungen definiert werden.

Die unterschiedlichen Formen werden in sogenannten Schlüsselpositionen festgelegt. Diese Positionen beschreibt die Form bei genau definierten Parametern. Zwischen diesen Schlüsselpositionen werden alle Formen interpoliert. Es würde somit ausreichen, neben der Ausgangsform, die Form an den Randpositionen zu definieren, da sämtliche andere Formen aus diesen errechnet werden könnten.

Aus der Analyse geht hervor, dass die Veränderungen zwischen den Randpositionen nicht linear sind. Dies wird zum Beispiel bei der Neigung der Serifen sichtbar (Abb. 4.11). Eine lineare Interpolation zwischen der Ausgangsform links und der Randposition rechts würde andere Zwischenformen

Abbildung 4.11: Veränderung der Form bei Serifen.

erzeugen als in der Abbildung gezeigt werden. Anhand der Position der Kontaktpunkte kann dies erkannt werden. Aus diesem Grund muss es möglich sein eine beliebige Anzahl an Schlüsselpositionen zu definieren.

Umgebung

Um die verschiedenen Zustände einer Serife beschreiben zu können, muss festgelegt werden in welcher Umgebung diese vorkommen und wie sich diese verändern kann. Zu diesem Zweck wird eine Reihe von Einstellungsmöglichkeiten bestimmt, die diese Veränderung im Editor simulieren. Die in der Analyse gewonnenen Folgerungen werden hier angewandt und ergeben folgende Liste:

- Neigung des Striches
- Dicke des Striches
- Dicke der Glyphe
- Ausrichtung des Striches
- Ausrichtung am Striche

Damit diese Einstellungen keine beliebigen Werte annehmen, ist es nötig diese einzuschränken. Es sollte dabei ein sinnvoller Wertebereich gefunden werden, der in der Praxis nicht überschritten wird.

Unter der Neigung des Striches wird die Differenz zwischen Strichwinkel und dessen Federwinkel am Platzierungspunkt verstanden. Dieser Einstellung ist ein numerischer Wert. Ein vertikaler Strich mit einer Horizontal ausgerichteten Feder würde somit 90° Neigung aufweisen. Diese 90° würden die Ausgangswert beschreiben. Von dieser Position aus ist eine Neigung in beide Richtungen möglich. Bei einer Veränderung um 90° würden die Kanten eines geraden Striches aufeinander liegen und somit nicht sichtbar sein. Dies würde somit die maximale Schwankungsbreite darstellen. Sinnvollerweise würde aber hier eine Wertebereich von 30 bis 150° ausreichen. Serifen in einem Bereich außerhalb dieser Winkel sind kaum zu finden und würden sehr extreme Formen aufweisen.

Die Dicke des Striches beschreibt die Breite des Striches am Platzierungspunkt. Da theoretisch die Breite über die Gesamtlänge des Striches nicht gleich sein muss wird dieser Punkt gewählt. Dieser Wert ist numerisch und sollte in einem positiven Wertebereich von mehr als 0 Einheiten liegen. Der Maximalwert sollte hier nicht mehr als die Dickte der Glyphe sein. In der Regel sollten hier keine Werte die größer als 2048 Einheiten sind vorkommen.

Die Dickte der Glyphe ist die Breite der Glyphe mit der Vor- und Nachbreite. Diese Einstellung ist numerisch und positiv sowie größer 0. In der Regel werden bei TrueType Schriften 1024 und bei OpenType 2048 Einheiten nicht überschritten. Die ist nur gängige Praxis, da bei OpenType in der Theorie beliebige Werte angenommen werden können. Hier würde es Sinn machen den Maximalwert auf die Größe des Gevierts (Abschnitt. 2.1.2) zu setzen.

Mit der Ausrichtung des Striches ist mehr ein Hilfsparameter gemeint. Generell findet man Serifen deren Unterkante parallel zum Liniensystem ist oder 90° zu diesem geneigt sind. Da diese sehr oft unterschiedliche Formen haben wäre es eine Hilfe wenn der Editor den Strich in der richtigen Ausrichtung darstellen könnte. Sinnvolle Werte wären hierbei „horizontal“ und „vertikal“.

Die Ausrichtung am Strich ist eine Einstellung die angibt, ob sich die Serife links oder rechts vom Strich befindet. Im Falle eines horizontalen Strichs wäre dies gleichwertig mit oberhalb oder unterhalb. Sinnvollerweise sollten diese Einstellung nur zwei Werte annehmen können. Der Einfachheit halber wird diese Ausrichtung als „rückwärts“ und „vorwärts“ bzw. „aufwärts“ und „abwärts“ bezeichnet

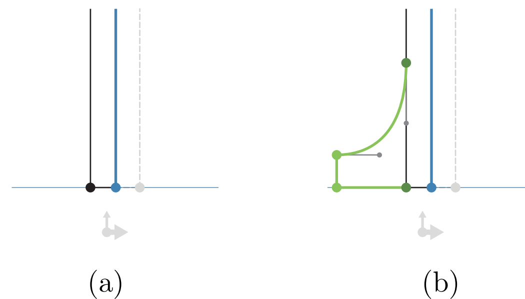
Aufbau des Editors

Der Serifen-Editor stellt neben dem Glyphen-Editor eine eigene visuelle Umgebung dar, die speziell für die Erstellung von Serifen gedacht ist. Es können mehrere verschiedene Serifen angelegt werden die unabhängig voneinander bearbeitet werden können.

Der Editor stellt eine visuelle Vorlage eines Striches zu Verfügung, der mittels Einstellungen, wie in Abschnitt 4.7.3 beschrieben, verändert werden kann. Abbildung 4.12 (a) zeigt die Grundeinstellung des Editors. Der Strich wird in der Mitte dargestellt. Am unteren Ende befindet sich eine Linie die zur Ausrichtung dient und in dieser Einstellung die Grundlinie darstellt. Ein Indikator an der Unterseite gibt die Richtung an in die der Strich ausgerichtet ist.

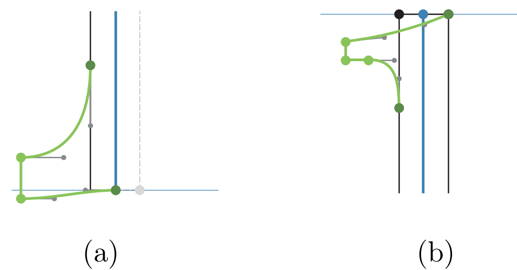
Konstruktion von Serifen

Im Gegensatz zum Glyphen-Editor werden hier nur Pfade und keine Striche verwendet um eine Form zu beschreiben. Eine mögliche Form einer Serife



(a) (b)

Abbildung 4.12: Aufbau des Serifen-Editors.



(a) (b)

Abbildung 4.13: einseitige und symmetrische Serifen.

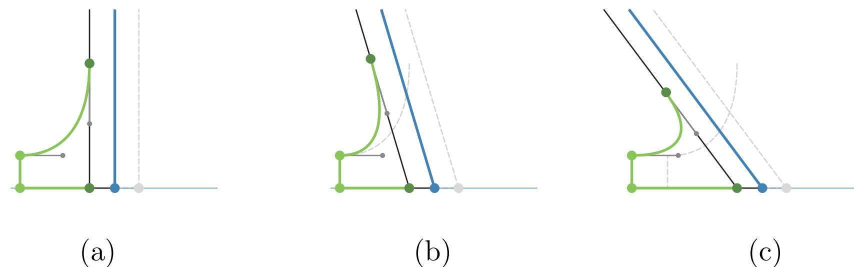
wird in Abbildung 4.12 (b) gezeigt.

Damit eine Serife aus dem Editor verwendet werden kann, muss eine Reihe von Bedingungen erfüllt sein. So darf eine Glyphe keine offenen Pfade haben. Beim Zusammensetzten mit dem Strich würde keine gültige Form entstehen. Die zweite Bedingung die erfüllt sein muss ist, dass die Serife mit dem Strich verbunden ist. Der Anfangs- und Endpunkt der Serife muss auf dem Pfad des Striches liegen. Wäre dies nicht der Fall, könnte erstens nicht festgestellt werden an welcher Position die Verbindung stattfindet und zweitens würde wieder ein offener Pfad entstehen. Diese zwei Punkte sind in Abbildung 4.12 (b) mit dunkelgrünen Punkten markiert.

Bei der Serife in Grafik (b) ist die eine Hälfte des Striches ausgegraut. Dies deutet an, dass es sich um eine Serife handelt, die an der Seite eines Striches angebracht wird. Diese wird automatisch gespiegelt wenn sie an der anderen Seite verwendet wird.

In Abbildung 4.13 sind zwei weiter Serifen abgebildet. In Grafik (a) ist eine Serife deren Endpunkt bis zur Mitte des Strichs geht. Es ist dabei wichtig, dass der Endpunkt an der Unterkante liegt damit die Form geschlossen wird.

Würden sich die Endpunkte jeweils auf unterschiedlichen Seiten der Mittelsachse befinden, wird angenommen, dass die Serife nicht symmetrisch ist,



(a) (b) (c)

Abbildung 4.14: Veränderung des Winkels des Striches.

wie in Grafik (b) abgebildet wird. In diesem Fall wird auch der Strich nicht mehr auf beiden Seiten ausgegraut. Die Serife hat in diesem Fall nur noch eine Richtung.

Veränderung der Umgebungseinstellungen

Die Anpassung an verschiedene Umgebungen ist der wichtigste Bestandteil des Serifen-Editors. Per Schlüsselpositionen sollen die einzelnen Formen der Serifen definiert werden und je nach Umgebung interpoliert werden. Durch die Anzahl der verschiedenen Einstellungen entsteht eine hohe Anzahl von Kombinationen.

Um die Anzahl der Schlüsselpositionen gering zu halten, wird versucht mit so wenigen Kombinationen wie möglich auszukommen. Aus diesem Grund werden die Veränderungen unabhängig voneinander betrachtet. Hierbei werden eine Winkel- und eine Dickenänderung unterschieden. Eine Kombination aus beiden Einstellungen kann nicht manuell erzeugt werden, sondern wird nur vom Serifen-Editor generiert.

Abbildung 4.14 zeigt eine Veränderung des Winkels. Für die jeweiligen Winkel von (a) 90, (b) 107 und (c) 127 Grad sind in diesem Fall die Schlüsselpositionen manuell definiert worden. Zwischen diesen wird vom System linear interpoliert um eine passende Serifenform zu erstellen. Es können in der Theorie beliebig vieler dieser Positionen definiert werden. Wichtig ist, dass für die extremen Randpositionen definierte Formen zu Verfügung stehen, da sonst der Spielraum bei Veränderungen sehr gering ist.

Für die Änderung der Dicke sind in Abbildung 4.15 (a) und (b) zwei Schlüsselpositionen definiert. Für die Änderung der Breite sollten weniger Schlüsselpositionen nötig sein als bei der Änderung des Winkels. Hier ist es wichtig, dass das System erkennen kann, welche Punkte sich bei einer Änderung der Breite bewegen und welche statisch sind.

In Abbildung 4.16 sind beide vorangegangen Veränderungen vermischt. Diese Form errechnet sich aus der Form für den Winkel von 4.15 (c) und der Form aus 4.15 (b).

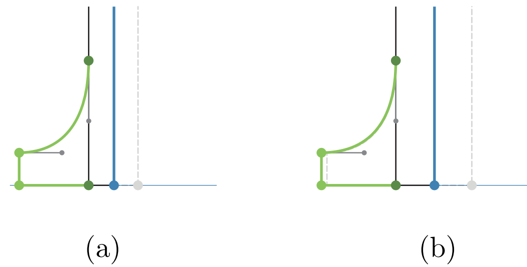


Abbildung 4.15: Veränderung der Breite des Striches.

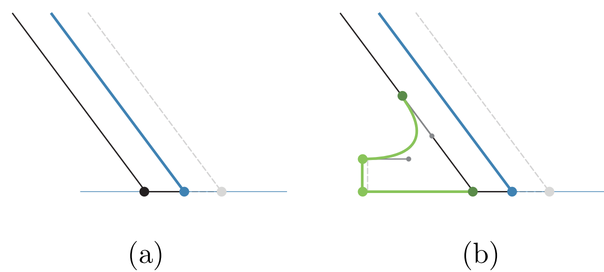


Abbildung 4.16: Veränderung des Winkels und der Breite des Striches.

Die Dicke des Gevierts oder des Striches sind Umgebungseigenschaften, die auf die Breite der Serife einen Einfluss haben. Um nicht für jeden dieser Eingeschalten eine neue Serifenform erstellen zu müssen wird hier mit einem Faktor gearbeitet. Dieser legt prozentuell fest, wie sich die Breite in Abhängigkeit der Ausgangsform verändert. Für die Breite des Gevierts und die des Striches gibt es jeweils einen eigenen Faktor. Um die Ausgangsform zu bestimmen, muss ein Referenzwert festgelegt werden.

Platzierung

Der Strich stellt für die Platzierung eigene Punkte zu Verfügung. Nur an diesen können die Serifen andocken. Serifen können grundsätzlich nur an den Enden von Strichen platziert werden. Es kommt dabei darauf an ob die Serife nur auf der Seite platziert wird oder über beide Seiten verläuft. Je nach Art muss unterschieden werden, wie viele dieser Serifenpunkte es gibt. Für Serifen die nur an einer Seite des Striches platziert werden gibt es am jeweils zwei dieser Punkte an jedem Strichende. Diese befinden sich jeweils an der linken und rechten Ecke eines Endes. Für Serifen die über beide Seiten gehen, gibt es nur einen Punkt pro Ende an dem sie platziert werden können. Dieser Punkt befindet sich in der Mitte des Striches. Diese Serifen haben von dem Editor eine vordefinierte Richtung an der sie sich orientieren.

Bei der Platzierung werden die Umgebungseigenschaften des Striches ausgelesen und an die Serife weitergegeben. Auf diese Art kann sich die Serife in ihrer Form anpassen. Auf diese Art wird bei einer Änderung des Striches automatisch die Serife mit geändert.

4.7.4 Limitierungen

Der Serifen-Editor kann nicht alle Umgebungseigenschaften berücksichtigen. Vor allem Serifen an dem Enden eines Großen *C* sind schwer zu realisieren. Diese legen sich direkt in die Rundung des Striches hinein. Das Ende ihres Striches ist nicht parallel zu der Achse auf der die Serife aufliegt.

Ein weitere Punkt der hier nicht berücksichtigt wird, sind Serifen die über mehrere Striche gehen. An dem obersten Punkt eines großen *A* gibt es Serifen, die an der Kante eines Striches beginnen und an der Kante eines anderen Striches auslaufen. Für diese Serifen müsste es zusätzlich eine Umgebungseinstellung geben, die diesen Umstand berücksichtigt. Diese Serifen kommen nicht so häufig vor und wurden aus diesem Grund nicht berücksichtigt.

4.8 Modifikationen

Einen weiteren Ansatz stellen sogenannte Modifikatoren dar. Diese werden verwendet um Zusammenhänge zwischen einzelnen Strichen einer Glyphe zu beschreiben. Ziel ist es, dass eine Glyphe mit diesen Modifikatoren in der Form beschrieben werden kann, dass diese sich bei Veränderungen automatisch anpasst und das beabsichtigte Aussehen behält. Eine Beschreibung in Prosa, für die Modifikatoren die ein große *A* beschrieben, könnte wie folgt aussehen:

- Die vertikalen Striche reichen von der Grundlinie zur Versalhöhe
- Der obere Endpunkt des rechten Striches ist am Ende der rechten Kante des linken Strichs
- Der Querstrich verläuft zwischen dem linken und rechten Strich
- Die Höhe de Querstrichs ist drei Prozent der Gesamthöhe unterhalb der x-Höhe

Eine Veränderung des Liniensystems würde zur Folge haben, dass sich die Glyphe automatisch anpasst und ihre Form behält. Das fertige Grundgerüst einer Schrift wäre mit diesem Modifikatoren mittels Parameter einfach änderbar.

Ähnliche Ansätze in diese Richtung gibt es bereits bei anderen Projekten wie zum Beispiel METAFONT oder *Lettersoup*. Es wird hier jedoch versucht ein einfaches System zu schaffen. Aus diesem Grund ist die Anzahl der Modifikationen möglichst gering gehalten. Weiters ist beim Entwurf darauf geachtet worden, dass diese Modifikatoren möglichst nicht aufeinander angewendet

werden müssen. Dies soll verhindern, dass die entstehenden Zusammenhänge zu komplex werden oder zirkuläre Abhängigkeiten entstehen.

4.8.1 Aufbau

Ein Modifikator ist aus mehreren Teilen aufgebaut. Als erstes Verfügt jeder Modifikator über ein Eingabe-Element und ein oder mehrere Ausgabe-Elemente. Aus dem Eingabe-Element werden die Werte ausgelesen und über den Modifikator verändert an die Ausgabe-Elemente weitergegeben. Bei einem Element kann es sich dabei um Punkte, Linien oder um Striche handeln. Je nach Modifikator kann es Unterschiede geben, welche Elemente akzeptiert werden.

Jeder dieser Modifikatoren verfügt wie elementare Elemente über verschiedene Einstellungen. Diese Einstellungen verändern das Verhalten des Modifikators. Je nach Kombination von Ein- und Ausgabe-Element kann sich die Anzahl der Einstellungsmöglichkeiten verändern.

Modifikatoren und Parameter sind sehr ähnlich von ihrer Funktionalität. Im Grunde ist ein Parameter ein spezieller Modifikator der andere oder keine Parameter als Eingabe-Element hat und diese verändert zurückgibt. Die Ausgabe eines Parameters kann auch nur einer skalaren Eigenschaft oder einem anderen Parameter zugewiesen werden.

4.8.2 Punkte

Der *Punkt-Modifikator* stellt dabei die Möglichkeit dar, zwei Elemente aneinander zu binden. Als mögliche Ein- und Ausgabe-Element des Punkt-Modifikators werden Punkte und Linien akzeptiert. Es gibt für diese Modifikation insgesamt drei verschiedene Konstellationen, aus denen eine sinnvolle Ausgabe erzeugt wird. Der Punkt-Modifikator ist einer der wichtigsten Modifikatoren da er am öftesten verwendet wird.

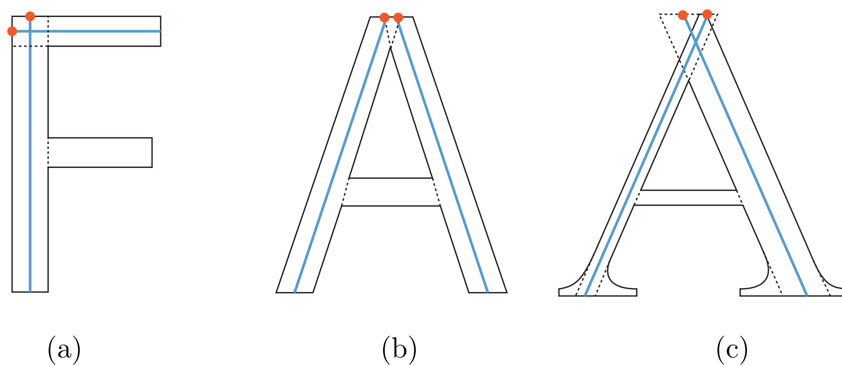
Punkt mit Punkt

Bei der Verbindung zwischen zwei Punkten hat der Punkt-Modifikator die Funktion, dass die Positionen der Ausgabe-Punkte an die des Eingabe-Punkt verschoben werden. Die Punkte befinden sich somit an derselben Stelle. Diese Funktionalität ist aber nicht ausreichend, um den Zusammenhang zwischen Punkten in einer Glyphe zu beschreiben. Der Punkt-Modifikator bietet aus diesem Grund eine Reihe von zusätzlichen Optionen. Diese unterscheiden sich je nach Art der Punkte. Es spielt dabei eine Rolle ob die Punkte einen Pinselstrich besitzen oder Teil einer Kurve sind.

Würde es sich bei der Ein- und Ausgabe nur um zwei Punkte handeln, die nicht Teil eines Pfades sind, würde es keine Optionen geben, da die Position das einzige ist, was verändert werden kann.

Tabelle 4.5: Einstellungen für die Punkt-mit-Punkt Verbindung

Einstellung	Auswirkung
<	Entgegengesetzte Außenkante liegen aufeinander
+	Eingabe-Punkt liegt auf Außenkante
>	Gleiche Außenkanten liegen aufeinander
.	Punkte liegen aufeinander



(a) (b) (c)

Abbildung 4.17: Skalierungs-Modifikator für die Rundung des Buchstaben *P*.

Sollten es sich bei dem Ein und Ausgabe-Element um einen Punkt handeln, der einen Pinselstrich besitzt, nutzt der Modifikator die zusätzliche Information des Pinselstriches an dieser Stelle. Die Verbindung wird nun nicht mehr nur auf einen Punkt beschränkt, sondern kann eine ganze Reihe von verschiedenen Möglichkeiten wählen. In Tabelle 4.5 sind die verfügbaren Optionen aufgelistet.

Die ersten drei Einstellungen gibt es jeweils für die linke und rechte Seite. Die Gesamtanzahl der Optionen für diese Einstellung wären somit sieben. Dies bedeutet zum Beispiel, dass die entgegengesetzten Außenkanten auf der linken oder rechten Außenkante des Eingabe-Elements liegen. Eine Beispiel für eine dieser Einstellungen wird in Abbildung 4.17 (a) gezeigt. Der Modifikator verbindet Punkt a mit b. Der vertikale Strich des F wird dabei mit dem Arm verbunden. Der vertikale Strich stellt dabei das Eingabe-Element dar. Die Verbindungseinstellung wird so gewählt, dass die gleichen Außenkanten aufeinander liegen (Unter der Voraussetzung das der Arm von links nach rechts verläuft). Die beiden Kanten Treffen sich im Punkt x.

Eine weitere Einstellung die dieser Modifikator in diesem Fall bietet, ist eine Verschiebung. Wie Abbildung 4.17 (b) zeigt, treffen sich die beiden Striche nicht im selben Punkt, sondern es besteht ein Abstand zwischen diesen

Punkten. Die Verschiebe-Option ermöglicht in diesem Fall den Abstand herzustellen. Mittels Parameter kann dieser Wert gesetzt werden. Die Achse auf der diese Verschiebung stattfindet ist identisch mit dem Winkel der Feder im Eingabe-Punkt.

Neben der Position der Verbindung und einer Verschiebung ist es in einigen Fällen nötig, dass die Striche abgeschnitten werden. Anhand des Beispiels in Abbildung 4.17 (c) wird deutlich, dass die Striche des *A* sich überschneiden. Die gestrichelte Linie zeigt dabei den ursprünglichen Verlauf. Damit die Striche des Punkt-Modifikators nicht wieder auseinander laufen gibt es eine Option die dies verhindert. Bei Buchstaben mit *Spitzen*, wie dem *M, N, W*, kommen solche Überschneidungen besonders oft vor.

Punkt mit Linie

Der Punkt-Modifikator kann auch dazu verwendet werden, um Punkte mit Linien zu verbinden. Dies ist zum Beispiel nötig, wenn Striche vom Liniensystem abhängen. Beispiele hierfür wäre der obere Arm eines *F* oder die beiden Arme eines *T*.

Das Verhalten bei dieser Konstellation von Eingabeelementen ist leicht unterschiedlich zu einer Punkt-mit-Punkt-Verbindung. Der wesentliche Unterschied hier ist, dass eine Linie eine horizontale oder vertikale Ausrichtung hat und somit nur den Wert für eine Achse liefern kann. Bei einer vertikalen Ausrichtung würde somit nur die x-Achse zu Verfügung stehen. Der Wert für y kann nicht bestimmt werden. Aus diesem Grund wird nur der eine verfügbare Wert an den Punkt weitergegeben. Dies bedeutet, dass der Punkt auf der anderen Achse weiter frei bewegbar bleibt.

Das Verhalten der Einstellung für die Position der Verbindung ist eingeschränkt durch den Umstand, dass eine Linie keine Außenkanten hat. In diesem Fall gibt es nur drei statt der sieben Möglichkeiten. Diese wären linke oder rechte Kante oder in der Mitte.

Die Einstellung für die Verschiebung wird durch die fehlende Achse auch beeinflusst. Da es nur einen Positionswert gibt für diesen Fall, kann die Verschiebung auch nur in diese Richtung stattfinden. Diese Verschiebung erlaubt einen Überlauf zu definieren.

Überschneidungen sind auf Grund der fehlenden Dicke der Linie nicht möglich. Aus diesem Grund wird diese Einstellung nicht verwendet.

4.8.3 Schnittpunkte

Es kann vorkommen, dass ein Strich andere Striche verbindet. Ein Beispiel hierfür wären die Querstriche des *A* und *H*. Je nach der Position der beiden Striche, ist die Länge des Querstriches anders. Dessen Endpunkte müssen sich an den anderen Strichen orientieren. Zusätzlich befindet sich dieser Querstrich noch auf einer gewissen Höhe.

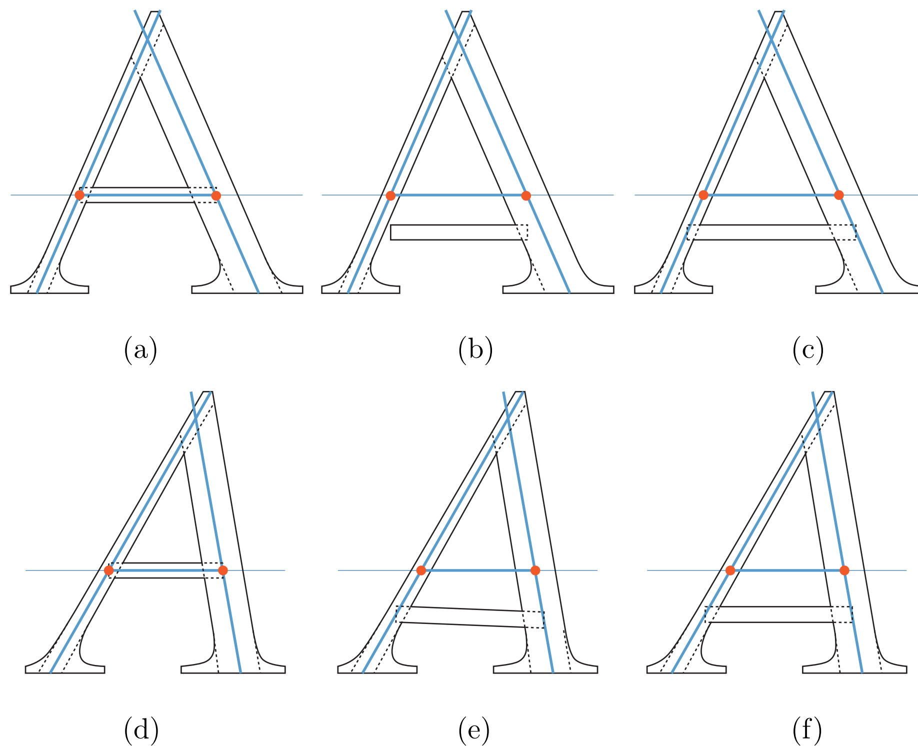


Abbildung 4.18: Schnittpunkt-Modifikatoren für Querstriche.

Der Modifikator nimmt zwei Elemente als Eingabe und gibt eine Position zurück. Bei den Elementen kann es sich nur um Linien oder Striche handeln.

Eine Einstellung ermöglicht die Verschiebung des Schnittpunktes. Dies wird untere anderem dazu benötigt, falls sich die Höhe der Querstriche zwischen den Glyphen unterscheidet. Bei der Bestimmung der Richtung kann eines der beiden Eingabe-Elemente gewählt werden. Bei dem Abstand der Verschiebung muss jedoch auf ein paar Besonderheiten Rücksicht genommen werden. In Abbildung 4.18 wird eine Verschiebung anhand der Achse der Linie gezeigt. Wie schon bei dem Punkt-Modifikator wird die fehlende Achse der Linie als Verschiebungsrichung verwendet. Grafik (a) stellt die Ausgangssituation dar. In Grafik (b) wird die Verschiebung gezeigt. Der Querbalken ist nicht mehr mit beiden Seiten verbunden. Damit verhindert wird, dass der Strich und der Querbalken die Verbindung verlieren, wird der Pfad des Striches verfolgt. Es wird nicht die Neigungsachse im Punkt der Verbindung genommen, sondern der Pfad in der Länge der Distanz verfolgt. In Grafik (c) ist mit dieser Einstellung die Verbindung wie gewünscht geschlossen. In diesem Fall wird das richtige Ergebnis geliefert, da die Winkel der beiden Striche von der Vertikalen aus gleich groß sind. Grafik (d) zeigt dasselbe Beispiel mit einem kursiven Schnitt. Die Verschiebungsachse ist wiederum der Strich. In

Grafik (e) wird deutlich, dass durch die unterschiedliche Neigung der Striche die Schnittpunkte verschiedene vertikale Abstände zu der Ursprungsposition haben. Der Querbalken ist nicht mehr waagrecht. Um einen Ausgleich zu schaffen, muss es eine Möglichkeit geben, bei der der Pfad verfolgt wird, aber der Abstand waagrecht oder senkrecht von der Ursprungsposition bestimmt wird. In Grafik (f) werden die Punkte auf dem Strich verfolgt bis diese denselben vertikalen Abstand von ihrer Ursprungsposition haben. Der Querbalken bleibt dadurch waagrecht.

Im Falle von sehr dünnen Strichen kann es wie schon beim Punkt-Modifikator zu Überschneidungen kommen. Aus diesem Grund muss es auch bei diesem Modifikator eine Einstellungsmöglichkeiten geben um dies zu verhindern.

Ein weiter Punkt der berücksichtigt werden muss, ist dass sich mehrere Schnittpunkte aus der Eingabe ergeben, sollte einer der verwendeten Striche geschwungen sein. Eine anfängliche Festlegung um den wievielten dieser Schnittpunkte es sich handelt ist nötig. Bei einer Veränderung kann es vorkommen, dass sich diese Anzahl der Schnittpunkte ändert. Aus diesem Grund kann es vorkommen, dass sich die Position des Schnittpunktes sprunghaft ändert.

4.8.4 Skalierungen

Neben Punkt- und Schnittpunkt-Verbindungen gibt es eine dritte Art von Modifikation die für Skalierungen zuständig sind. Die werden unter anderem für Rundungen benötigt.

Der Skalierungs-Modifikator verwendet zwei Punkte als Eingabe und kann mehrere Punkte als Ausgabe-Elemente haben. Die Eingabe-Punkte werden dabei als Referenz für die Skalierung verwendet. Verändert sich der Abstand zwischen diesen, werden alle Ausgabe-Punkte, anhand eines berechneten Faktors, verschoben.

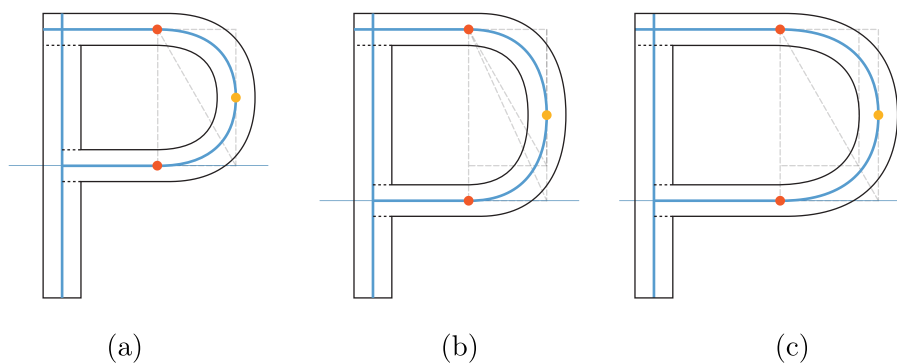
Der Modifikator besitzt eine Einstellung für das Verhalten der Skalierung. Es wird zwischen proportionaler und nicht proportionaler Skalierungen unterschieden. In Abbildung 4.19 sind diese zwei unterschiedlichen Einstellungen abgebildet. Grafik (a) stellt dabei die Ausgangssituation dar. Die Eingabe-Punkte sind dabei am oberen und unteren Ende des Bogens angebracht. Als Ausgabe-Element wird der der Punkt am äußersten Rand des Bogens gewählt. Der untere Teil des Stammes nach dem Bogen ist an die Linie fixiert.

Die Skalierungseinstellungen sind in Tabelle 4.6 aufgelistet. Diese listet das unterschiedliche Verhalten der Ausgabe-Punkte auf, wenn die Eingabe-Punkte verändert werden.

In Grafik (b) wird die Linie abgesenkt. Die Skalierung soll dabei nicht proportional stattfinden. Der Bogen des P beginnt sich zu strecken. Da sich die beiden Eingabe-Punkte auf einer Achse befinden, würde die *nicht proportionale* und *die proportionale vertikal* Einstellung dieses Ergebnis erzeugen.

Tabelle 4.6: Auswirkung der Skalierung bei Veränderung der Eingabe-Punkte bei verschiedenen Einstellungen

Einstellung	Auswirkung
Proportional	Seitenverhältnis bleibt bestehen
Nicht proportional	Veränderungen wirken sich nicht proportional aus
Nicht proportional horiz.	Nur horizontale Veränderungen wirken sich aus
Nicht proportional vert.	Nur vertikale Veränderungen wirken sich aus



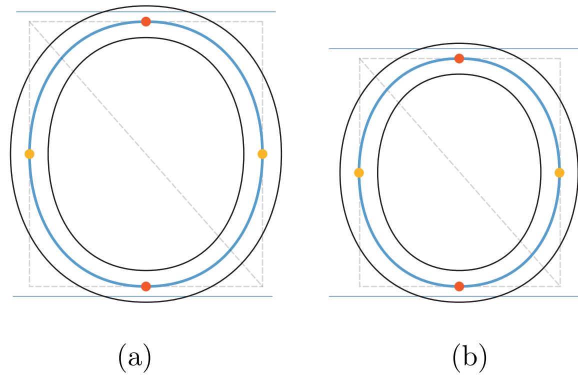
(a) (b) (c)

Abbildung 4.19: Skalierungs-Modifikator für die Bogen des Buchstaben P.

Die Skalierung für denselben Fall, nur mit einer proportionalen Veränderung, ist in Grafik (c) abgebildet. Obwohl die Eingabe-Punkt sich nur vertikal auseinander bewegen, wird der Ausgabe-Punkt auch nach rechts verschoben, damit diese denselben relativen Abstand zueinander behalten.

Ein zweites Beispiel wird in Abbildung 4.20 gezeigt. Hier wird der Strich für ein *O* mittels eines Skalierungs-Modifikators an die Grundlinie und die Versalhöhe fixiert. Die Einstellung ist so gewählt, dass die Seiten proportional skaliert werden. In Grafik (a) ist das *O* in der Ausgangsform abgebildet. Der obere und untere Punkt ist dabei mit einem Abstand an die jeweilige Linie gebunden und mit einem Abstand versehen. Dieser Abstand soll den Überhang erzeugen. In Grafik (b) wurde die Versalhöhe verringert. Die Form des *O* verändert sich nur gering. Die Kurvenform bleibt von der Form her gleich, wird allerdings enger im Verlauf. Stabil hingegen ist die Dicke des Striches. Dies ist unabhängig von der Skalierung, da nur der Pfad betroffen ist.

Der Modifikator verwendet bei der Berechnung des Mittelpunkts für Skalierung im Normalfall einen Standardwert. Dieser befindet sich in der Mitte zwischen den Eingabe-Punkten. In Abbildung 4.19 (b) und (c) befindet sich



(a) (b)

Abbildung 4.20: Skalierungs-Modifikator für den Buchstaben *O*.

der Mittelpunkt auf der Achse zwischen den zwei Eingabe-Punkt auf Höhe des Ausgabe-Punktes. Das eingezeichnete Quadrat, welches die beteiligten Punkte umschließt, zeigt wie die Skalierung sich nur nach rechts ausbreitet. In Abbildung 4.20 befindet sich der Mittelpunkt der Skalierung wiederum zwischen den Eingabe-Punkten auf Höhe der Ausgabe-Punkte.

Kapitel 5

Umsetzung

Dieses Kapitel befasst sich mit den technischen Hintergründen und Problemen der vorgeschlagenen Ansätze. Es wird beschrieben wir die einzelnen Ansätze realisiert werden können und welcher Weg der Umsetzung gewählt wurde.

Um in Abschnitt 4 beschrieben Ansätze auf ihre Realisierbarkeit zu überprüfen, wurde ein Prototyp erstellt. Dieser ist einer in Objective-C programmierte Applikation, die wesentliche Teile dieser Arbeit umsetzt. Es werden dabei die von Apple zu Verfügung gestellten Bibliotheken verwendet. Es handelt sich um eine Fensteranwendung die für Mac OS X entwickelt wurde.

Für den Aufbau der einzelnen Klassen werden zum Teil UML-Diagramme verwendet die die wichtigsten Zusammenhänge beschreiben sollen. Sonst wird eine Beschreibung der Funktionalität mittels Bildern verwendet.

Für die Namenskonvention werden zwei verschiedene Präfixe verwendet. Klassen mit `BZ` stellen elementare Funktionen für Linien, Kurven oder Punkte zu Verfügung. Bei dem Präfix `MT` handelt es sich um Klassen die für den Prototypen konkrete Implementierungen enthalten.

5.1 Aufbau

Der Prototyp besteht aus einer Cocoa-Applikation die mit einem dreispaltigem Layout aufgebaut ist, wie in Abbildung 5.1 gezeigt wird [29]. Jeder dieser Spalten stellt einen eigenen Bereich dar, indem verschiedenen Interaktionen möglich sind.

Die mittlere Spalte wird als Zeichenfläche verwendet. Hier wird für die Darstellung der Glyphen die `MTGlyphView` verwendet und für den Serifen-Editor die `MTSerifView`. Der Benutzer kann mittels Maus und Tastensteuerung neue Punkte und Kurven zeichnen und Modifikatoren anlegen. Zusätzlich werden die Linien des Liniensystems und eigene Linien dargestellt. Für den Serifen-Editor werden spezielle Hilfslinien verwendet die einen Strich simulieren.

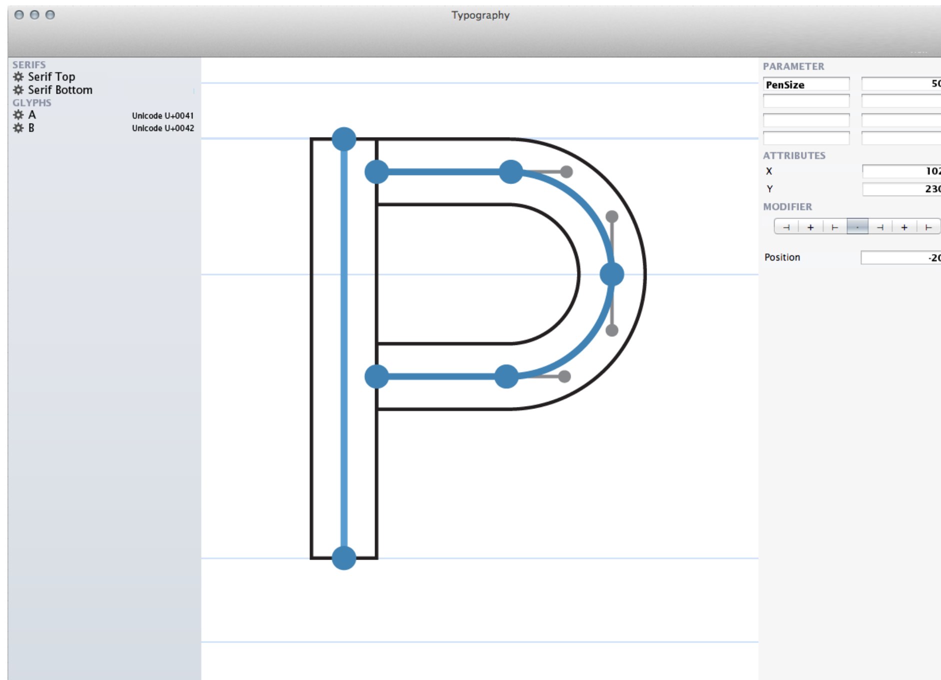


Abbildung 5.1: Aufbau des Prototyps.

Die rechte Spalte dient zur Auflistung der einzelnen Glyphen und Serifen. Diese werden getrennt voneinander in zwei Listen gehalten. Bei dem Anwählen einer Glyphe oder Serife wird diese in der Zeichenfläche dargestellt.

Der Parameter-Editor befindet sich in der rechten Spalte an der obersten Stelle. Dieser stellt eine Liste dar, bei dem immer zwei Textfelder in einer Reihe sind. Diese können verwendet werden um den Namen und den Wert eines Parameters zu bestimmen. Es können theoretisch beliebig viele Parameter in dieser Liste angelegt werden.

Für die Bearbeitung von einzelnen Elementen müssen diese mit der Maus angeklickt werden. Bei der Selektion eines einzelnen Elements, werden dessen Attribute in einer Liste in der rechten Spalte namens *Attributes* dargestellt. Dort können die Werte über ein Textfeld manipuliert werden. Je nach Element stehen andere Optionen zur Bearbeitung zur Verfügung. Bei der Selektion von mehreren Elementen werden keine Attribute dargestellt.

Unter der Liste für Attribute gibt es den *Modifier*-Editor. Wird ein Element mit einem Modifikator ausgewählt, werden die Einstellungsmöglichkeiten dieses Modifikators dort dargestellt. Wie in der Attributeliste können hier die unterschiedlichen Einstellungen für den oder die Modifikatoren gewählte werden.

Bei einer Veränderung eines Wertes in einem Editor wird die Zeichenfläche sofort aktualisiert. Diese stellt somit immer die aktuelle Version einer

Glyphe oder Serife dar.

5.2 Parameter

Parameter verhalten sich sehr ähnlich wie Variablen. Aus diesem Grund würde sich eine Implementierung anbieten die die eingegebenen Ausdrücke direkt evaluiert, wie dies etwa mit der `eval()`-Funktion in JavaScript möglich ist. ObjectiveC bietet keine ähnlichen Funktionen an, da es sich um keine interpretierte Sprache handelt. Aus diesem Grund würde sich die Verwendung von LUA-Script [27] oder Python [34] anbieten. Diese könnten als in den ObjectiveC Code eingebunden werden und als Brücke fungieren. Systemvariablen würden an den Interpreter weitergegeben werden der diese nutzt um die Parameter und Ausdrücke zu berechnen und einen Wert an das System zurückgeben.

In dem Prototyp wurde diese Parameter nur testhalber implementiert. Es werden nur einfache Ausdrücke mit einer Operation verarbeitet. Es soll nur die Grundfunktionalität dieses Systems demontiert werden.

Die Klasse `MTExpression` ist eine Erweiterung der Klasse `NSString`. In dieser Klasse wird eine Instanz auf einen Singelton `MTExpressions` gehalten. In diesem werden alle Parameter angelegt. Für die Umwandlung des Ausdrucks in einen Wert, ruft die Klasse `MTExpression` die Funktion `evaluateExpression` aus der `MTExpressions` Klasse auf und übergibt sich selbst als Parameter. Der Ausdruck wird evaluiert und als Double zurückgegeben.

Die Klasse `MTExpressions` verwendet in der `evaluateExpression` zerlegt den übergebene Ausdruck anhand der mathematischen Operationen *+, -, ** und */*. Die zerlegten Teile werden darauf überprüft, ob es sich um Zeichenketten handelt und gegebenenfalls mit dem in der Parameterliste vorhandenen Werten ausgetauscht. Sollten alle Zeichenketten durch Werte ausgetauscht worden sein, werden diese mit den einzelnen Operationen wieder zusammengerechnet. Das Endergebnis wird als Rückgabewert zurückgegeben.

5.3 Kurven und Striche

Für die grafische Darstellung von Kurven und Strichen werden Bézierpfade verwendet. *Cocoa* stellt mit seinem *AppKit-Framework* Zeichenmethoden zur Verfügung, mit denen diese gezeichnet werden können [28]. Für die Berechnung von Strichformen und ähnlichem wurden eigene Klassen definiert, die über die nötigen Funktionen verfügen.

5.3.1 Kurven

Kurven werden in einer eigenen Klasse namens `BZCurve` gekapselt. Diese verfügt über Funktionen für die Darstellung, zur Kollisionserkennung und für Manipulationen. Eine Kurve besteht aus einer Reihe von Punkten, deren Reihenfolge den Verlauf darstellen. Diese Kurven können in der `MTGlyphView` gezeichnet, verschoben und dessen Attribute verändert werden. Wie alle Elemente in der `MTGlyphView` haben Punkte eine visuelle Repräsentation und können bei einer Selektion über den Attribut-Editor bearbeitet werden.

Aufbau

Punkte werden über die Klasse `BZPoint` realisiert. Sie Besitz zwei Werte, die die Positionen in der *MTGlyphView* darstellen. Der Datentyp dieser Punkte ist `MTExpression` und ist somit in der Lage Ausdrücke als Positionen zu verarbeiten. Diese Klasse vererbt seine Eigenschafen an den `BZExtendedPoint` der eine *Delegate* hat, welches über eine Veränderung der Position informiert. Von diesem Punkt werden zwei weiter Punktarten abgeleitet: `BZAnchorPoint` und `BZControlPoint`. Der *BZAnchorPoint* wird verwendet um Ankerpunkte zu definieren. Mit ihm sind jeweils der vorhergehende und nachfolgende Kontrollpunk über eine Referenz, das *Delegate* verbunden. Wird ein Kontrollpunkt verschoben wird der Ankerpunkt informiert und kann den anderen Kontrollpunkt verschieben. Die Ankerpunkte werden in der `BZCurve` als sortierte Liste gehalten.

Der Klasse `BZCurveElement` wird verwendet um einzelne Segmente einer Kurve zu beschreiben. Sie hält eine Referenz auf die Ankerpunkte und die zwei dazwischen liegenden Kontrollpunkte. In dieser Klasse werden Funktionen zur Unterteilung und Berechnung von Schnittpunkten zur Verfügung gestellt. Abbildung 5.2 zeigt ein Diagramm der Aufbaus.

Für die grafische Darstellung wird die Funktion `NSBezierPath` verwendet. Diese durchläuft die Liste mit `BZCurveElement` und setzt dessen Punktwerte in die Funktion ein und malt diese in die `MTGLyphView`.

Abtastung

Für manche Berechnungen, die mit der Bézierkurve durchgeführt werden, ist es nötig diese abzutasten und in eine Reihe von Punkten umwandelt. Für die grafische Darstellung wird dies vom *AppKit-Framework* übernommen, jedoch kann nicht auf dessen verwendete Werte zurückgegriffen werden.

Für die Abtastung der Kurve wird der *De-Casteljau-Algorithmus* verwendet [6]. Dieser kann eine Kurve in zwei neue Bézierkurven teilen. Dieser Algorithmus wird rekursiv auf ein Segment die Linie angewendet. Die Unterteilung findet in der Mitte ($t = 0.5$) statt. Als Rekursionsanker wird die Flachheit der neuen Kurven genommen. Unterscheidet diese einen gewissen Toleranzwert, wird abgebrochen. Für ein exakteres Ergebnis könnte diese

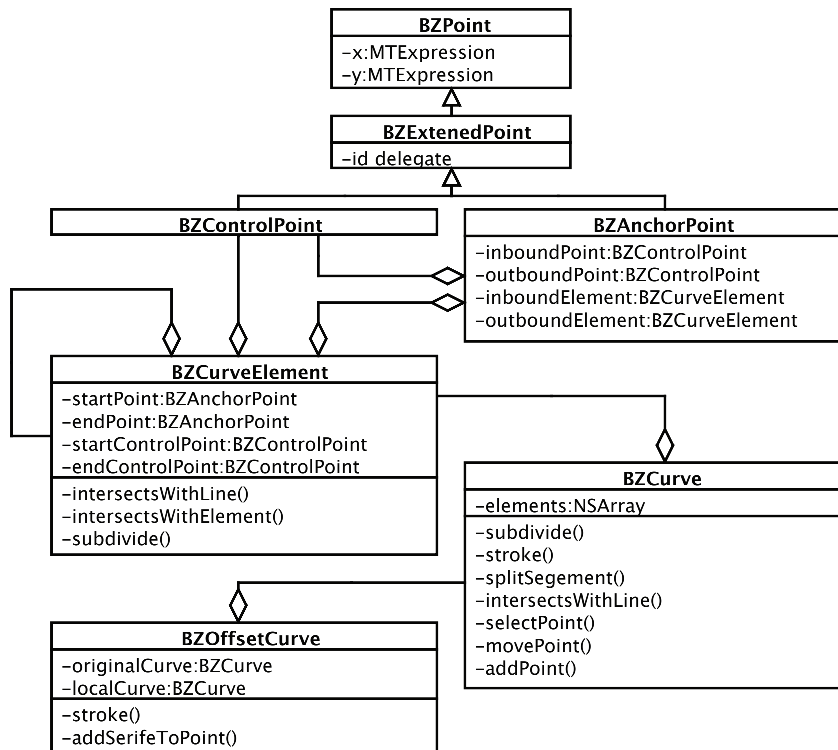


Abbildung 5.2: Referenzform für Breitenänderungen.

Unterteilung bis auf Pixelbreite durchgeführt werden [35].

Die abgetasteten Werte werden in einer sortierten Liste in der Klasse `BZCurveElement` gehalten. Die Berechnung wird dabei jedes Mal neu durchgeführt, wenn sich ein Punkt des Segments verändert.

Kollisionserkennung

Die Kollisionserkennung dient dazu, um festzustellen an welchen Punkt sich die Kurve mit einer Linie oder einer anderen Kurve überschneidet. Dies wird für die Modifikatoren und die Platzierung der Serifen benötigt.

Für die Berechnung von Schnittpunkten wird die Abtastung eines Kurvensegments verwendet. Zwischen den Abtastpunkten werden Linien erstellt. Um den Punkt zu ermitteln, an dem sich eine Kurve mit einer Linie oder einer anderen Kurve überschneidet, müssen diese abgetasteten Linien oder zwei Kurven/Linien gegeneinander, Stück für Stück, auf eine Überschneidung überprüft werden. Für diese Überprüfung gibt es einfache mathematische Formeln.

Der errechnete Schnittpunkt gibt einen Positionswert zurück. Um den entsprechenden Wert für t zu erhalten wird eine binäre Suche verwendet.

Diese Verändert den t Wert solange bist der Positonswert für dieses t mit dem Schnittpunkt übereinstimmt, oder eine gewisse Anzahl an Iterationen erreicht wird.

Unterteilung

Für die Erstellung von Strichen, kann es von Nöten sein, dass die Kurve an bestimmten Punkten unterteilt wird. Diese Unterteilung muss an Stellen stattfinden, an denen sich die Rundung der Kurve ändert oder die Ausbreitung auf der x-Achse oder y-Achse ein Maximum erreicht. Eine genaue Erklärung wieso dies nötig ist findet sich in Abschnitt 5.3.5.

Um die Punkte zu finden an denen sich der Wert für x oder y einem Maximum nähert, kann über eine Ableitung des *De-Casteljau-Algorithmus.* Die Gleichung

$$t = \frac{-\sqrt{-a \cdot b + a \cdot d + b^2 - b \cdot c - b \cdot d + c^2} \pm a - 2 \cdot b + c}{a - 3 \cdot b + 3 \cdot c - d} \tag{5.1}$$

stellt die erste Ableitung dar. Die vier variablen $a,\ b,\ c,\ d$ sind die vier Punkte der Kurve. In dieser Gleichung werden einmal die x und einmal die y-Werte der Punkte eingesetzt, um das jeweilige t zu bekommen. Es können jeweils zwei gültige Werte für t, deren Wert zwischen $0 < t < 1$ liegen, entstehen. Insgesamt gibt es somit zwischen null und vier Punkten an denen die Kurve unterteilt werden muss, damit diese an den Extrempunkten unterteilt wird.

Für eine weitere Unterteilung an den Wendepunkten wird die zweite Ableitung des textitDe-Casteljau-Algorithmus verwendet. Diese Ableitung

$$t = \frac{a - 2 \cdot b + c}{a - 3 \cdot b + 3 \cdot c - d} \tag{5.2}$$

kann für x und y jeweils nur einen Wert liefern und kann nur dann eingesetzt werden, wenn die Bedingung

$$a - 3 \cdot b + 3 \cdot c - d \neq 0 \tag{5.3}$$

erfüllt ist.

Für beide Ableitungen zusammen ergibt sich ein Maximum von sechs Punkten an denen die Kurve unterteilt werden muss. Eine *saubere* Kurve hat im Normalfall bereits Punkte an den Extremwerten und an den Wendepunkten. Diese Funktion kann somit auch dazu verwendet werden eine Kurve, hinsichtlich der Punktsetzung, zu optimieren.

5.3.2 Striche

Die Funktionen zum Erstellen und Zeichnen von Strichen werden in der Klasse `BZOffsetCurve` gesammelt. Der Verlauf der Striche ist von eine Kurve abhängig, aus diesem Grund wird eine Instanz eines Striches mit einer

BZCurve initialisiert. In der MTGlyphView können neue Striche nur auf bestehende Kurven gelegt werden. Um die original Kurve nicht zu verändern, wenn für die Berechnung der Striche Unterteilungen gemacht werden müssen, wird eine Kopie dieser Kurve angelegt. Je nach Art der Kurve gibt es verschiedene Attribute, die über den Attribute-Editor verändert werden können.

Bei der Erstellung von Strichen werden die Eckpunkte an den Enden als Serifenpunkte definiert. Diese können Instanzen der Klasse MTSerif aufnehmen. Diese erlauben es Serifen aus dem Serifen-Editor zu platzieren. Wenn eine Serife an einen Eckpunkt platziert wird, wird eine Referenz auf dieser Serife hinterlassen. Beim Zeichnen der Glyphe ließt die Klasse BZOffsetCurve den Winkel und Breite an dieser Position aus und fordert vom Serifen-Editor einen Pfad mit diesen Parametern an. Der Serifen-Editor erzeugt eine Form die vom Strich an die Position des Eckpunktes verschoben wird. Der Eckpunkt wird in der Zeichenanweisung durch den Pfad der Serife ausgetauscht. Zu diesem Zweck wird dem Pfad der Serife die NSBezierPath-KLasse übergeben, mit der der Strich gezeichnet wird. Es entsteht dadurch ein geschlossener Pfad.

5.3.3 Gerade Striche

Gerade Striche verfügen nur über eine Attribute vom Typ MTExpression, dass ihre Breite beschreibt. Um den Strich zu berechnen wird der Ausgangspfad zweimal dupliziert im festgelegten Winkel um die halbe Breite in beide Richtungen verschoben. Abbildung 4.3 zeigt Beispiel für gerade Striche. Stiche dieser Art bestehen nur aus geraden Linien. Rundungen sind in geraden Strichen nicht vorgesehen da diese nur für einfache Formen verwendet werden sollen.

Da nur gerade Linien verwendet werden um diese Striche zu zeichnen, ist keine Abtastung oder Unterteilung nötig. Schnittpunkte können einfach über Linienüberschneidungen festgestellt werden.

5.3.4 Runde Striche

Runde Striche werden als jene bezeichnet bei denen der Pfad Rundungen aufweisen kann. Die Feder bei dieser Art des Striches steht immer normal auf den Pfad. Abbildung 4.5 zeigt Beispiele für diese Art von Strichen. Die Außenkante zu berechnen ist im unterschied zu den geraden Strichen aufwendiger. Eine einfache Verschiebung des Pfade liefert kein brauchbares Ergebnis.

Für bestimmte Pfade gibt es eine Möglichkeit, eine Kurve zu berechnen die einen gewissen Abstand zur Ausgangskurve hat [18]. Abbildung 5.3 zeigt die den Vorgang der Konstruktion. Als erste wird der Punkt ermittelt von dem die Verschiebung ausgeht. Wie in Grafik (a) werden zwei Geraden konstruiert die durch die Ankerpunkte verlaufen und normal auf den Pfad stehen. Durch diesen Punkt werden zwei weitere Geraden konstruiert die

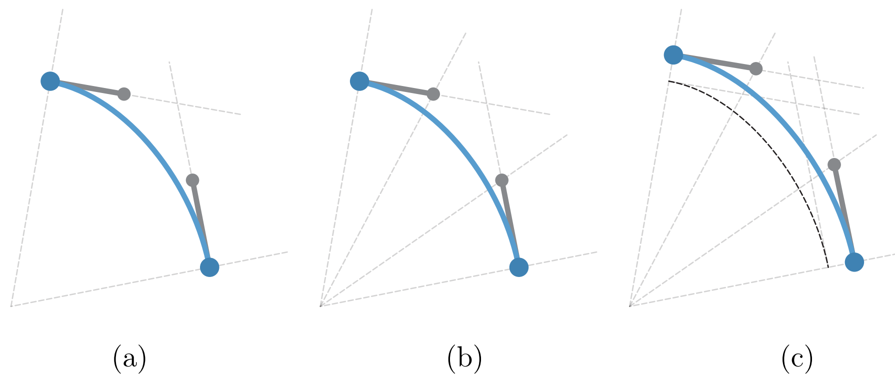


Abbildung 5.3: Verschiebung eines Pfades für eine Außenkante.

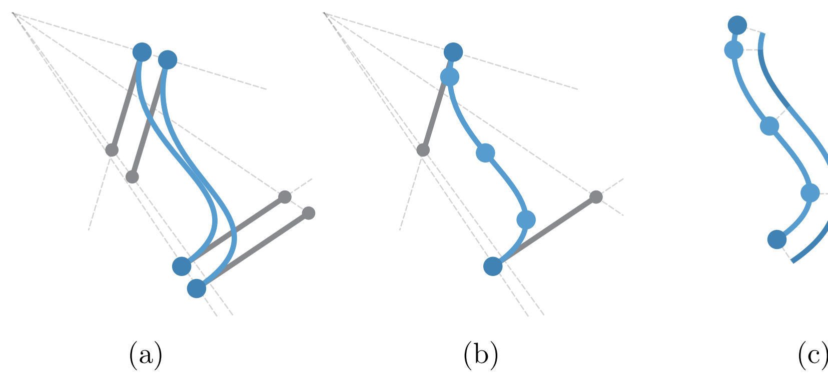


Abbildung 5.4: Unterteilung und Verschiebung einer Kurve bei stark Wendungen.

durch die Kontrollpunkte verlaufen, wie in Grafik (b). Die neue Kurve entsteht indem je der Kontrollpunkt und der dazugehörige Ankerpunkt parallel verschoben wird. Die Position der Punkt bleibt auf der Graden die durch diese Konstruiert wurde.

Abbildung 5.4 (a) zeigt den selben Vorgang bei einer Kurve die starke Wendungen hat. Die durch dieses Verfahren berechnete Kurve ist verläuft nicht wie gewünscht im selben Abstand. Diese Art der Verschiebung funktioniert nur unter gewissen Voraussetzungen. Es dürfen keine Wendepunkte in der Kurve vorkommen und der Winkel der die Geraden zwischen Anker- und Wendepunkte einschließt darf nicht 90 Grad überschreiten. Zu diesem Zweck wird der Pfad unterteilt. Dies geschieht nach dem in Abschnitt 5.3.1 beschriebenen Verfahren. Abbildung 5.4 (b) und (c) zeigen die Unterteilung der Kurve und die anschließende erzeugte Außenkante. Für diesen Fall besteht diese Kante aus vier Einzelkurven.

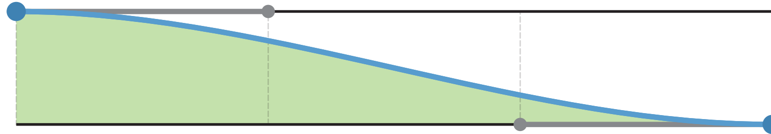


Abbildung 5.5: Referenzform für Breitenänderungen.

Für die Berechnung der Kanten durchläuft die Klasse `BZOffsetCurve` jedes `BZCurveElement` und lässt jeweils eine Außenkante über beide Seiten berechnen. Die Unterteilung wird jeweils auf eine Kopie des `BZCurve` angewendet.

5.3.5 Änderung der Strichdicke

Die Änderung der Strichdicke bei runden Strichen erfordert zusätzliche Anpassungen an den `BZOffsetCurve`. Diese Art von Kurve verfügt über keinen Federwinkel, sondern nur über eine Dicke in jedem Punkt. Aus diesem Grund muss die Klasse `BZOffsetCurve` die eine zusätzliches Attribut für jeden Punkt halten, das dessen Dicke beschriebt.

Die entstehende Änderung der Dicke soll zwischen zwei Werten verlaufen. Dieser Verlauf soll dabei Gleichmäßig sein. Die Position der Kontrollpunkte spielt keine Rolle. Eine Referenzkurve wie in Abbildung 5.5 soll den Verlauf zwischen den zwei Enden eines Segments definieren. Diese Referenz wird je nach Länge oder Differenz der Bereiten gleichförmig skaliert.

In Abbildung 5.6 (a) werden zwei Kurven in Abstand der beiden Randpunkte erstellt. Zwischen diesen soll die Dickenveränderung stattfinden. Punkte aus der Referenzkurve in 5.5 werden übertragen. Die grüne Form stellt dabei das erwünschte Ergebnis dar. Wie in Abbildung 5.6 (a) ersichtlich ist, sind die Tangenten nicht mehr horizontal oder vertikal. Dies würde dazu führen, dass Knicke im Strich entstehen. Um dies auszugleichen werden diese Tangenten wie in Grafik (b) gedreht, so dass diese mit dem Winkel wieder mit den Ursprungstangenten übereinstimmen. Die dabei entstehende Kurve zeigt deutliche Unterschiede zu der gewünschten Form.

Eine weiter Möglichkeit die Verschiebung anzupassen, wäre dien Verschiebung die wie bei der Berechnung der Außenkante funktioniert. Der Unterschied hier wäre, dass die Verschiebung an beiden Enden unterschiedlich weit durchgeführt wird. Abbildung 5.6 (c) zeigt die entstehende Kurve. Mit einer geringen Abweichung verläuft diese wie gewünscht.

Das Problem stellt eine mögliche Unterteilung der Kurve dar. Die vorgestellte Art der Verschiebung würde die Referenzkurve nur zwischen den einzelnen Punkten erzeugen. Wird eine Kurve jedoch Unterteilt um einen Strich zu erzeugen würde dies mehrere Wellen in der Kante erzeugen.

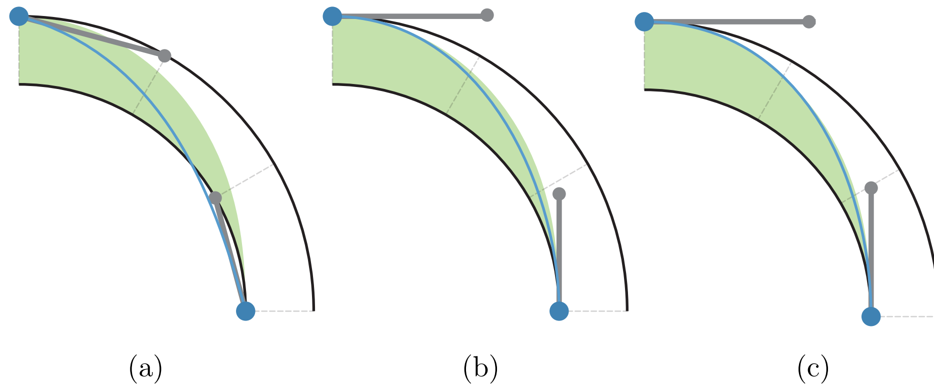


Abbildung 5.6: Anpassung der Kurve an unterschiedliche Breiten.

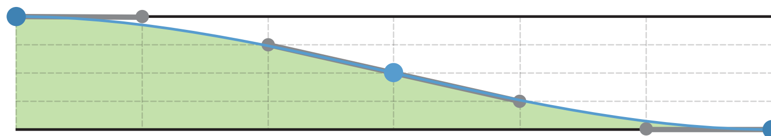


Abbildung 5.7: Unterteilte Referenzform für Breitenänderungen.

Um dieses Problem zu beseitigen gibt kann die Referenzkurve an den mehreren Punkten Unterteilt werden. Abbildung 5.7 zeigt eine veränderte Kurve die um einen weiteren Punkt in der Mitte verfügt. In Abbildung 5.8 wird diese Form wird wie in Abbildung 5.6 (a) in die Kurve gelegt. In die Tangenten der Randpunkte werden begradigt und die Tangenten im Mittelpunkt werden so verändert das diese auf einer Geraden liegen. Grafik (b) zeigt diese Anpassung. Die entstehende Kurve liegt sehr liegt nahe an der gewünschten Form.

Das Unterteilen der Referenzkurve erzeugt genauere Ergebnisse. Diese kann dazu verwendet werden um eine exaktere Kurve zu erzeugen. Dies kann bei sehr runden Kurven die über keine Extrem- oder Wendepunkte verfügt zu einem besseren Ergebnis führen.

Bei einer Unterteilung die nicht direkt in der Mitte der Referenzkurve stattfindet muss die entstehende Kurve im nachhinein weiter angepasst werden. Die Referenzkurve aus Abbildung 5.9 ergibt eine Kurve wie in Abbildung 5.10 (a) und in weitere Folge (b) gezeigt wird. Die Kurve entspricht nicht der gewünschten Form. Die zwei Kontrollpunkte um den mittleren Punkt müssen gedreht werden damit sich die Kurve anpasst. Die Funktion die verwendet wird um die Tangenten der mittleren Kontrollpunkte begradigt muss angepasst werden um diesen Fehler auszugleichen.

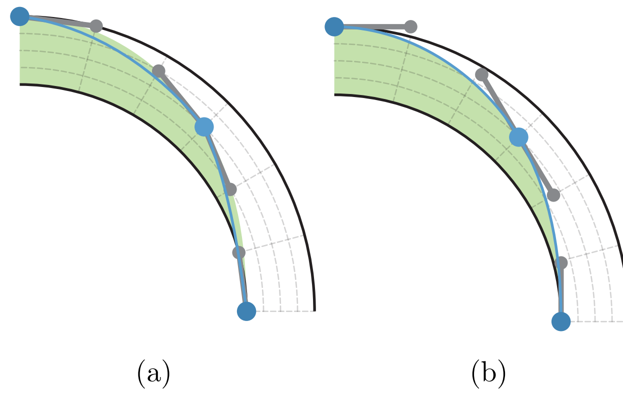

(a) (b)

Abbildung 5.8: Verbesserte Anpassung der Kurve an unterschiedliche Breiten.

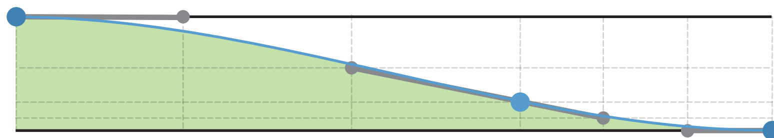


Abbildung 5.9: Unterteilte Referenzform für Breitenänderungen.

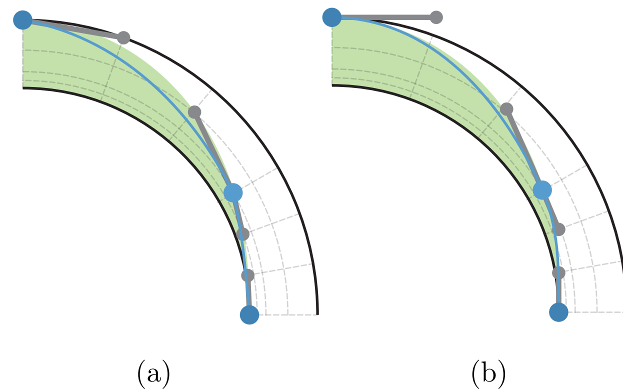

(a) (b)

Abbildung 5.10: Verbesserte Anpassung der Kurve an unterschiedliche Breiten.

Probleme

Die Berechnung der runden Kurven führt zu einigen Ungenauigkeiten. Vor allem bei der Änderung der Dicke können Rechenfehler entstehen. Diese

könnten durch das weitere Hinzufügen von Punkten ausgeglichen werden, jedoch würde dies bedeuten, das die Kurve aus sehr vielen Punkten besteht die unter umständen nicht nötig werden. Aus diesem Grund wird hier eine Ungenauigkeit in Kauf genommen um die Anzahl der Kurvenpunkte gering zu halten. Es kann bei kleinen Veränderungen von Kurvenpunkten dadurch zu Sprüngen in der Form des Striches kommen. Ein Algorithmus der die Kurve versucht zu optimieren und unnötige Punkte zu entfernen könnte eine Verbesserung lieferen.

5.4 Serifen Editor

Der Serifen-Editor wird wie in Abschnitt 4.7.3 aufgebaut. Im Serifen-Editor wird die Kurve für die Serife angelegt. Es ist nur eine einzige Kurve erlaubt. Die Klasse `MTSerifView` stellt die selben Zeichenfunktionen wie in der `MTGlyphView` zu Verfügung nur, das keine Striche gezeichnet werden können.

Als Datenstruktur dient eine Liste die für festgelegt Umgebungsparameter eine Kopie der Kurve enthält. Für jeden neu angelegten Pfad für eine bestimmten Umgebung wird eine neue Kopie erstellt und dessen Umgebungsparameter gespeichert.

Für verschiedene Winkel und Strichdicken ist je eine eigene Liste vorgesehen. Bei der Interpolation zwischen den den Serifen wird werden aus der Liste für jeden Umgebungsparameter die zwei Pfade gesucht deren Parameter am nächsten zu der gewünschten Position liegen. Sollte der angefoderte Wert den maximalen definierten Parameter überschreiten, ist keine Interpolation möglich. In desem Fall wird der Randwert genommen.

Die Berechnung erfolgt in der Reinefolge, das erst die Serife für einen bestimmten Winkel erzeugt wird und dann dessen Breitenänderung hinzugefügt wird. Bei der Änderung der Breite wird festgestellt welche Punkt sich zwischen den festgelegten Formen ändern. Der Abstand um den sie sich ändern wird auf die selben Punkte, der Serife für den Winkel, hinzugefügt. Dies bedeutet, dass zwischen der Breite und dem Winkel keine vollständige Interpolation durchgeführt wird. Es wird nur die relative Verschiebung der Punkte berücksichtigt.

Für die Inerpolation gibt es einen speziellen Punkt der anders behandelt werden muss als die anderne. Dieser Punkt ist jener, der an Seite des Striches liegt. Dieser Punkt muss immer mit dem Strich verbunden sein, damit die Form geschlossen ist. Bei einer linearen Interpolation kann es jedoch zu einem Problem kommen. In Abbildung 5.11 wird das Probelm veranschaulicht. Zwischen den zwei Formen am Rand wird die Serife linear Inerpoliert. Durch die unterschiedliche Höhe des obersten Punktes ergibt sich aus der Berechnung ein Punkt der nicht immer auf dem Strich liegt. Dieser Fehler würde dazu führen, dass einen Sprung im der Außenkante des Pfades entsteht. Je weiter die Punkte auf der Kante in den unterschiedlichen Positionen

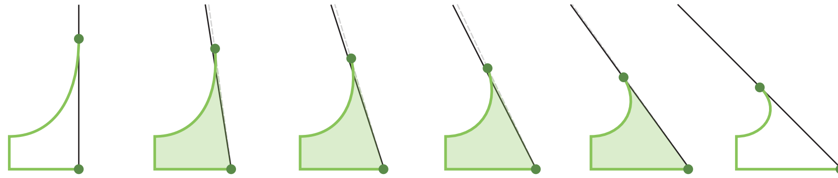


Abbildung 5.11: Fehler bei der Interpolation von Serifen.

auseinanderliegen, desto größer wird dieser Rechenfehler. Um dem Entgegen zu wirken wird für diesen einen Punkt einen Korrektur vorgenommen. Die Position bei jeder Interpolation wird so angepasst, das der Punkt auf den nähestesten Punkt der Kante verschoben wird. Dies gilt auf für Kontrollpunkte die auf der Außenkante des Striches aufligen.

Diese Korrektur ist in den meisten Fällen sehr minimal, wie man in Abbildung 5.11 dargestellt ist. Für die restlichen Punkte kann die normale lineare Interpolation angewendet werden, da diese die gewünschten Ergebnisse liefert.

Für die Platzierung an einem Strich kann von der `MTSerifView` eine Instanz von `MTSerif` angefordert werden. Diese wird an einem Eckpunkt eines Striches hinterlegt. Sie ist dafür zuständig die Umgebungsparameter des Striches auszulesen und vom Serifen-Editor einen passenden Pfad anzufordern. Dieser wird in der Klasse `MTSerif` wenn nötig gespiegelt und gedreht und an den Strich weitergegeben.

5.5 Modifikatoren

Modifikatoren sind im Grunde den Ausdrücken bei Parametern sehr ähnlich. Sie nehmen Werte von Elementen entgegen, verändern diese und geben diese an andere weiter. Die Sprache der Ausdrücke könnte durchaus so gewählt werden, das diese Modifikatoren ersetzten, jedoch würde dies den Anspruch widersprechen einen möglichst visuellen Editor zu schaffen. Modifikatoren können somit als eine speziell angepasste Ausdrücke verstanden werden die über ein eine eigene Darstellung und vordefinierte Attribute verfügen.

Bei der Erstellung von Modifikatoren wird eine Selektion von Elementen an diesen übergeben. Die Reihenfolge spielt dabei eine Rolle welche Elemente als Ein- und Ausgabe gesehen werden. Modifikatoren haben eine fixe Anzahl an Eingabe-Elementen, aus diesem Grund werden die Elemente am Anfang einer Selektion als Eingabe angesehen und der Rest als Ausgabe.

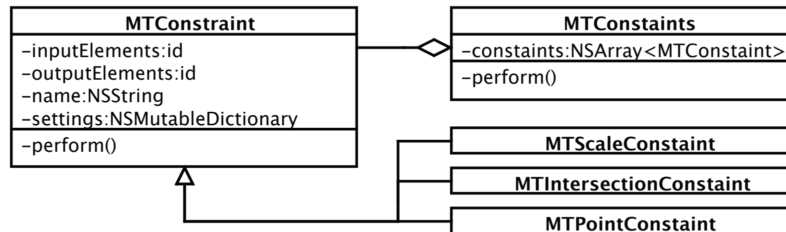


Abbildung 5.12: Klassendiagram oder Modifikatoren.

5.5.1 Aufbau

Modifikatoren erben alle von der selben Basisklasse `MTConstraint`. Diese Klasse verfügt über je ein *Property* für die Ein und Ausgabe-Elemente. Diese sind vom Typ `id` und können somit jeglichen anderen komplexen Datentype halten. Diese Elemente sollten alle das *Interface* `MTMoveable` implementieren. Für Modifikatoren mit mehreren Ein- oder Ausgabe-Elementen ist es jedoch nötig diese in eine Liste zu wie ein `NSArray` zu speichern. Diese Datentypen implementieren dieses *Interface* nicht.

Abbildung 5.12 zeigt das Klassendiagram für die Modifikatoren. Für die Verarbeitung wird ein *Strategy*-Pattern verwendet [5, S. 21]. In der Klasse `MTConstraint` ist die Funktion `perform()` definiert. In den abgeleiteten Klassen wird in dieser Funktion wird das verhalten der einzelnen Modifikatoren implementiert. Jede Glyphe verfügt über einer eigenen Instanz von `MTConstaints`. Dort werden alle verwendeten Modifikatoren hinzugefügt und in einer sortierten Liste gehalten. Bei einer Verschiebung von Elementen im der `MTGylphsView` wird die `perform()`-Methode von `MTConstarints` aufgerufen, die die Liste durchläuft und die `perform()`-Methode jedes Modifikators aufruft.

Die *Properties* `name` und `settings` werden dazu verwendet um Attribute des Modifikators zu verändern. In dem `NSMutableDictionary settings` werden alle Attribute eines Modifikator, samt Wert, abgespeichert. Wird ein Punkt in der `MTGlyphsView` selektiert, wird in der Instanz von `MTConstraints` nach Modifikatoren gesucht die diesen Element als Ein- oder Ausgabe-Element haben. Die gefundenen Modifikatoren werden zurückgegeben. Der Attribut-Editor liest die den Namen und das `NSMutableDictionary` aus und übergibt diese an eine passende `NSCellView`. In dieser werden die Werte in Textfeldern oder anderen Steuerelementen visuell dargestellt und können ausgelesen und verändert werden. Bei einer Änderung werden die Werte in das `NSMutableDictionary` eingetragen und die Modifikatoren erneut mittels `perform()` aufgerufen.

Modifikatoren haben bestehen somit aus einer Implementierung und einer visuellen Komponente die deren Bearbeitung erlaubt. Diese zwei Teile sind

voneinander getrennt. Auf der Darstellungsebene werden Werte nur ausgelesen und bei bei einer Veränderung gespeichert, bzw. das System informiert, dass eine Änderung stattgefunden hat.

Für die Darstellung der Modifikatoren werden alle `MTConstraint` in der Liste von `MTConstraints` durchlaufen und an den Positionen der Ein- und Ausgabe-Elemente eine Markierung gezeichent.

5.5.2 Punkte

Punkt-Modifikatoren nehmen nur ein Element als Eingabe. Aus diesem wird der Positionswert ausgelesen. Die Ausgabe-Elemente werden in einer Schleife durchlaufen und mittels der Funktion `moveToPoint`, die in `MTMoveable` definiert ist, an die Position des Eingabe-Elements verschoben.

Sollte die Kurve der Punkte über einen `BZOffsetPath` verfügen werden zusätzliche Attribute beachtet. In diesem Fall werden werden die Punkte des Striches berücksichtigt. Abschnitt 4.8.2 beschreibt die möglichen Einstellungen. Die Achse des Eingabe-Elements gibt dabei immer die Richtung an in der verschoben wird. Diese Information wird aus dem `BZOffsetPath` ausgelesen.

Das Ausgabe-Element verändert durch den Punkt-Modifikator nie seine Position. Auch bei der Verwendung von Verschiebungen werden immer nur die Ausgabe-Elemente verändert.

5.5.3 Schnittpunkte

Der Modifikator für Schnittpunkte verwendet zwei Elemente als Eingabe. Es muss sich dabei um Linien oder Pfade handeln. Als Ausgabe kann ein Punkt oder eine Linie verwendet werden.

Für die Bestimmung der Schnittpunkte wird die im `BZCurveSegment` vorhanden Funktion verwendet (Abschnitt 5.3.1). Mit hilfe dieser werden die Elemente gegenseitig auf eine Überscheidung gestestet.

Die Berechnung des Abstandes für die Verschiebung des Schnittpunktes erfolgt über die Abtastung der Kurve. Die einzelnen Punkte der Abtastung der Kurve können mit geraden Linien verbunden werden. Der Schnittpunkt wird entlang dieser Linien verschoben bis die gewünschte Länge erreicht wird.

Für eine Berechnung eines fixen Abstands, wird eine Suchalgorithmus verwendet. Dieser tastet die Kurve, vom Schnittpunkt aus, in bestimmten Abständen ab, bis der Abstand überschnitten wurde. Danach kann mit einer binären Suche der genaue Punkt des Abstandes bestimmt. Die Suche vom Schnittpunkt aus soll sicherstellen, dass im Falle von mehreren möglichen Punkten, immer der am nächsten zum Schnittpunkt verwendet wird.

5.5.4 Skalierungen

Der Modifikator für Skalierungen verwendert zwei Elemente als Eingabe. Für die Berechnung der Skalierung wird die Position der Punkte vor der Verschiebung berechnet und an eine entsprechende Position nach der Verschiebung verschoben. Der Verschiebungsfaktor der wird aus dem relativen Abstand zu den zwei Eingabe-Punkten berechnet. Der Faktor eines Punktes p ergibt sich aus der Position des Einagbe-Punktes I_1 an der niedrigeren Position im Raum und dem zweiten Punkt I_2. Der Faktor s ergibt sich somit auf aus der Gleichung

$$s = \frac{p - I_1}{I_2 - I_1}. \tag{5.4}$$

Nach der Verschiebung wird dieser Faktor verwendet um den Punkt mit der Gleichung

$$p' = I'_1 + s \cdot (I'_2 - I'_1) \tag{5.5}$$

zu berechnen.

Die Berechnung der Punkt wird für jede Achse separat durchgeführt. Je nach Einstellung des Modifikators wird die diese nur bei Einer oder bei Beiden angewendet. Bei einer Skalierung deren Seitenverhältnis gleich bleiben soll. Wird der Faktor für die fehlende Breite aus den Ein- und Ausgabe-Elementen berechnet. Der Maximalabstand einzelner Elemente auf der fehlenden Achse bestimmt diesen Faktor.

Kapitel 6

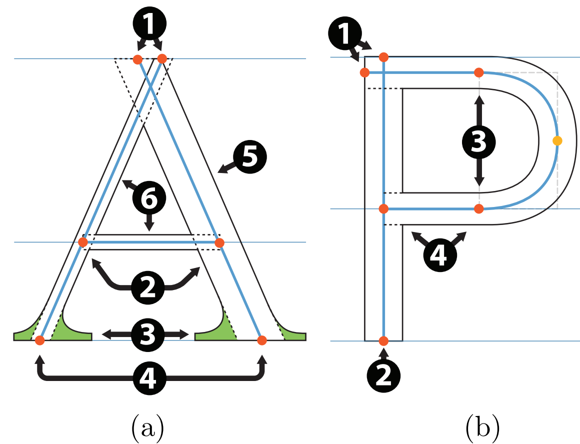
Diskussion

Das im Zuge dieser Arbeit entworfene System soll eine einfache Möglichkeit darstellen, einen Entwurf für eine Schrift zu erstellen. Dieser Entwurf soll so flexibel gestaltet werden können, dass dieser über mehrere Parameter verändert werden kann ohne jeden einzelnen Buchstaben zu bearbeiten. Der Verzicht auf komplexe Beschreibungssprachen und die Kompatibilität zu PostScript-Kurven sollen dessen Integration in einen bestehenden Workflow ermöglichen. Der Anwendungsbereich der Ansätze und deren mögliche weitere Verwendung wird in diesem Kapitel besprochen.

6.1 Aufbau von Schriften im Prototypen

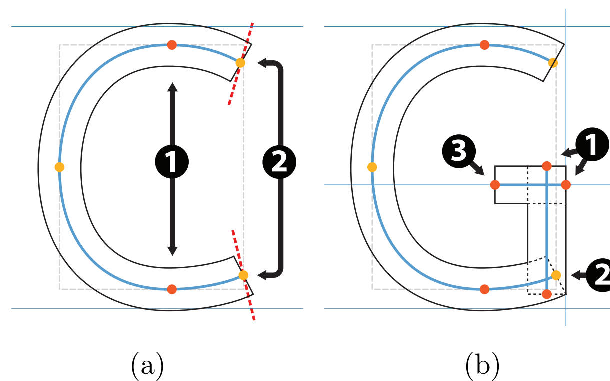
Der Aufbau von Glyphen ist vor allem bei Großbuchstaben sehr einfach und lässt sich gut zeigen. Im Kapitel 4.8 wird an Hand von einigen Beispiel gezeigt, wie die einzelnen Modifikatoren funktionieren. Es werden dabei oft die gleichen Buchstaben verwendet um die Funktionsweise zu demonstrieren. Das Zusammenspiel mehrere Modifikatoren soll einen Überblick über die Anwendung des Systems zeigen. Abbildung 6.1 (a) zeigt die Glyphe eines A. Wie schon im Abschnitt 4.8 erklärt, werden die Punkte Nummer 4 an die Grundlinie gebunden. Die Punkte bei 1 werden erst per Punkt-Modifikator verbunden und danach an die Versalhöhe gebunden. An den Punkten 3 werden Serifen hinzugefügt. Die Striche bei 5 und 6 werden per Parametern auf eine bestimmte Höhe gesetzt, die in einer Abhängigkeit zueinander steht. Bei den Punkten 2 werden Schnittpunkt-Modifikatoren verwendet, um den Querstrich zwischen dem Strichen hinzuzufügen.

Abbildung 6.1 (b) vervollständigt das Beispiel aus dem Abschnitt 4.8. Der vertikale Strich wird an Punkt 1 und 2 an die Linien gebunden. Der Beginn der Rundung wird an den vertikalen Strich gebunden. Die Punkte 3 werden erst an die Linien gebunden. Danach wird ein Skalierungs-Modifikator an die Rundung hinzugefügt. Punkt 4 wird per Schnittpunkt-Modifikator an den vertikalen Strich gebunden.



(a) (b)

Abbildung 6.1: Aufbau von Glyphen mittels des Prototypen.



(a) (b)

Abbildung 6.2: Problem bei der Endungen eines Striches.

6.2 Kritikpunkte

Für einige Fälle kann der Prototyp keine Lösung anbieten. Abbildung 6.2 (a) zeigt dies anhand eines *C*. Das Ende der Striche, bei Punkt 2, sollte wie die roten Linien verlaufen. Durch das Fehlen von beweglichen Winkeln bei dieser Art von Strich ist es jedoch nicht Möglich.

In Vielen anderen Fällen ist die Konstruktion sehr aufwändig. So wie im Falle des *G* aus Abbildung 6.2 (b). Die Ausrichtung des unteren Endes des Bogens muss über eine zusätzliche Linie geschehen. Bei Punkt 2 werden die beiden Striche mittels eines Punkt-Modifikators verbunden. Damit der Strich gerade bleibt, muss Punkt 1 mittels eines weiteren Punkt-Modifikators an eine vertikale Linie gebunden werden. Diese Linie wiederum muss bei Punkt 2 an die Rundung gebunden werden. Wenn sich diese bewegt, verschiebt diese das untere Ende des vertikalen Striches. Gleichzeitig bewegt diese die

vertikale Linie, die wiederum den Punkt bei 1 verschiebt. Dasselbe Konstrukt wird für den Querstrich verwendet. Sowohl horizontale Linie, als auch das Ende des Striches, werden an den vertikalen Strich gebunden. Punkt 3 wird dann an die Linie gebunden. Über diese Konstrukte ist es möglich, verschiede Problemen zu lösen. Jedoch stellen diese eine umständliche Lösung dar, die einfacher sein sollte.

6.3 Einsatzbereich

Der Einsatzbereich ist auf Grund der nur sehr groben Manipulationsmöglichkeiten klar im Bereich des Rapid Prototyping angesiedelt. Es ist schnell möglich eine Schrift zu entwerfen, die eine gewisse optische Eigenschaft hat. Mittels der Modifikatoren und Parameter kann diese in ihrem Aussehen angepasst werden. Die dabei entstehenden Schriften sind jedoch weit davon entfernt ein Endprodukt zu sein. Zu viele kleine Details können nicht beachtet werden und müssten mit einem ausgereiften Schrifteditor ausgebessert werden.

Dieser Prototyp reiht sich mit Programmen ein, die versuchen einen experimentellen Ansatz zur Erstellung von Schriftarten zu bieten. Es könnte am besten als ein Werkzeug verstanden werden um eine Vorlage zu erstellen, die als Ausgangsposition genommen wird um weiter zu arbeiten.

6.4 Vergleich

Das System lässt sich am ehesten mit zwei bestehenden Ansätzen vergleichen. Von METAFONT versucht es die Möglichkeiten, die mittels Parametrisierung zu Verfügung stehen und dessen Art wie Striche gezeichnet werden zu kopieren. Der Grundgedanke wie einzelne Glyphen aufgebaut werden sollten, ähnelt hingegen in vielen Zügen denen von Lettersoup. Dieses Projekt übernimmt hingegen viele Ideen von METAFONT.

Der Unterschied der zu diesen Projekten besteht ist, dass nicht versucht wurde eine eigene Beschreibungssprache zu entwickeln, sondern rein auf visuelle Editoren gesetzt wurde. Ein Vergleich mit den verwandten Technologien soll die Unterschied beziehungsweise die Vor- und Nachteile aufzeigen.

6.4.1 METAFONT

METAFONT ist in seinem Umfang klar überlegen. Es wird von Anfang bis zum Ende eine vollständige Technologie geboten, mit der Schriften erstellt werden können. Mit der Beschreibungssprache ist es möglich, Schriften in hoher Qualität zu erstellen, die für den professionellen Einsatz geeignet wären. Einzig und alleine die fehlende Unterstützung und das nicht vorhanden sein eines visuellen Editors ist ein Nachteil. Die Frage ist auch, ob es überhaupt

einen Editor geben kann mit dem es sinnvoll möglich METAFONTs visuell zu erstellen.

Die Form wie Parameter verwendet werden ist in METAFONT, durch die Tatsache, dass es sich hierbei um eine vollwertige Scriptsprache handelt, von Haus aus gegeben. Modifikatoren sind nicht nötig, da diese nur komplexere Parameter darstellen.

Einen Serifen-Editor gibt es in METAFONT in der Form nicht. Jedoch könnten Serifen über Makros sehr umfangreich programmiert werden, so diese mindestens dieselbe Funktionalität haben. Rein die Möglichkeit visuell zu arbeiten und als Ausgabe Bézierkurven zu verwenden ist ein Vorteil des Prototypen. Hier könnten aber Derivate von METAFONT wie METAFOG helfen dieses Manko zu beseitigen [10].

Der Vergleich des Prototypen mit METAFONT, bezüglich ihrer Verwendung, ist rein wegen ihrer Unterschiedlichen Zielsetzungen nicht sehr aussagekräftig. Es zeigt jedoch, dass diese Ansätze durchaus in der Lage sind zu einem hochwertigen Ergebnis zu führen.

6.4.2 Lettersoup

Lettersoup verwendet sehr viele Ansätze die in dieser Arbeit auch verfolgt werden. Dies ist einerseits die Trennung von der logischen Bedeutung und deren visueller Repräsentation und andererseits die Verwendung von Parametern. Zur Beschreibung der einzelnen Glyphen wird eine Beschreibungssprache für *Parts* verwendet. Diese stellen einzelne Grundbausteine dar. Dies werden in einem *Blueprint* zusammengesetzt. Diese Schritte finden im Prototypen visuell statt. Es wird nur bei den Serifen ein einmal definiertes Muster verwendet, das immer wieder eingesetzt werden kann. Für Elemente wie Stämme und Bögen ist ein erneutes Erstellen oder Duplizieren von bestehenden Objekten nötig. Was diese beiden Projekte jedoch verbindet, ist die Verwendung von Parametern zum Verändern der Schrift.

Lettersoup hat ein schon vordefiniertes Set an Variablen die übergeben werden um den Font zu generieren. Im Prototyp hingegen muss selbst bestimmt werden welche Werte als Parameter verwendet werden sollen. Dies führt zu mehr Aufwand aber lässt mehr Spielraum zu. Der Serifen-Editor unterscheidet sich klar vom Funktionsumfang. Dieser ist in keiner Form in Lettersoup enthalten.

6.5 Fazit

Der Prototyp hat einige Rückschlüsse über die einzelnen umgesetzten Ansätze zugelassen. Manche Teile dieser Arbeit haben einen gewünschten Erfolg gezeigt, bei anderen sind Probleme entstanden deren Lösung nicht trivial genug war um diese zu erarbeiten. Aus diesem Grund wurde der Prototyp in einigen Funktionen beschnitten. Dies vermindert zwar den Umfang mit dem

der Prototyp einzelne Funktionen verwenden kann, aber nicht das Zusammenspiel einzelner Komponenten. Dieses Zusammenspiel zu zeigen war auch Ziel dieser Arbeit.

6.5.1 Prototyp

Der Prototyp setzt Teile der Arbeit technisch um. Dieser ist wie im Kapitel 4 angelegt. Viele der Ansätze sind nur rudimentär implementiert, um eine generelle Idee der Funktionsweise zu geben. Vor allem die Bedienung ist meist über Tastenkürzel und dadurch relativ unintuitiv. Das visuelle Feedback über die Position von Modifikatoren und ähnlichen Elementen ist in manchen Fällen nicht ersichtlich. Jedoch ist es möglich mit den zu Verfügung gestellten Funktionen das System zu testen und Rückschlüsse zu ziehen. Vor allem in Bereich der Bedienung würde eine Verbesserung nötig sein.

6.5.2 Weiterverwendung einzelner Ansätze

Einzelne Teile dieses Projektes könnten für sich alleine in einem anderen Editor durchaus Verwendung finden. Teile könnten herausgenommen und angepasst werden. Der Nutzen der einzelnen Ansätze ist jedoch sehr voneinander abhängig.

Serifen

Der Serifen-Editor würde sich vor allem anbieten weiter verwendet zu werden. Die von ihm erstellen Serifen könnten in der Theorie auch auf normale Outline-Schriften angewendet werden. Umgebungseigenschaften von Punkten an denen die Serifen platziert sind, müssten aus den Kurven errechnet werden. Eigenschaften wie die Neigung oder die der Winkel des Striches würde sich einfach aus dem Verlauf der Kurve ergeben. Eigenschaften wie die Dicke des Strichs müssten über Abstandsberechnungen angenommen werden. Für Eigenschaften die die Breite definieren könnten einfach für jede Serife manuell veränderbare Eigenschaften hinzugefügt werden.

Der Serifen-Editor in seiner jetzigen Form müsste um einiges weiter entwickelt werden. Manche Arten von Serifen lassen sich nicht erstellen oder in der Form anpassen, wie es nötig wäre. Die Grundidee hinter diesem Ansatz scheint jedoch sehr gut zu funktionieren. Serifen stellen eine in sich abgeschlossene Form dar, die einfach an einen Teil der Kurve hinzugefügt wird.

Striche

Die Verwendung von automatisch errechneten Strichen ist der Grundbaustein, der viele andere Ansätze erst sinnvoll macht. Sie ermöglichen die Erstellung eines Striches der nur durch eine Kurve beschrieben wird. Diese

Kurve ist einfacher zu verändern als zwei parallel verlaufende Kurven einer Outline-Schrift.

Das Problem an der automatischen Berechnung von Strichen ist, dass diese im Idealfall sich so verhalten sollten wie eine kalligraphische Feder. Diese Anforderung ist aber kaum zu erfüllen. Für die nötigen Berechnungen der Kurven, wären viele Interpolationen nötig. Dies führt oft zu einer großen Anzahl von Teilkurven, die eine Form beschreiben. Dies ist aber nicht gewünscht, da eine nachträgliche Säuberung der Kurve nötig wäre. Algorithmen zu Vereinfachung könnten dies übernehmen, würden aber auch wiederum einen weiteren Interpolationen hinzufügen. Die Algorithmen müssten sehr stabil sein, damit auch bei kleinen Änderungen im Verlauf der Kurve keine zu großen Rechenfehler entstehen. Diese Fehler würden dazu führen, dass bei kleinsten Verschiebungen der berechnete Strich zu sehr unterschiedlichen Ergebnissen führt.

Der ideale Strich würde eine Breiten- und Winkeländerung in einem beliebigen Punkt zulassen und dabei eine Kontur erzeugen die keine überflüssigen Punkte hat. Dies ist im Prototyp definitiv nicht möglich. Jedoch würde die generelle Idee von dieser Form der Stricherstellung auch in einem Outline-Editor Verwendung finden können. Hier würde jedoch ein ausgereifteres System nötig sein.

Parameter

Die Verwendung von Parametern bringt viele Vorteile mit sich, jedoch ist ihr Nutzen stark davon abgängig wo diese verwendet werden können. Im Prototyp sind Striche, Punkte und Modifikation dafür vorgesehen diese aufzunehmen. In einem normalen Editor jedoch würden viele dieser Komponenten wegfallen. Einzig und alleine Objekte die über eine Position verfügen könnten weiterhin mittels Parametern angesteuert werden. Dies würde nur noch auf Punkte und Hilfslinien zutreffen. Die Sinnhaftigkeit einzelne Punkte einer Outline-Schrift mit Parametern zu versehen ist eher fraglich. In der Theorie könnte zwar ein sehr flexibles System entstehen, wenn eine große Anzahl von Punkten über Parameter gesteuert werden, dieses wäre aber in einem visuellen Editor sehr unübersichtlich und würde kaum Vorteile gegenüber einer reinen script-basierten Lösung darstellen.

Im Zusammenhang mit einem eigenständigen Serifen-Editor und/oder den generierten Strichen, würden die Wiederverwendung von Parametern durchaus Sinn ergeben, jedoch möglicherweise in einer reduzierten Form, die rein als globaler Einstellungsmechanismus von Werten dient.

Modifikatoren

Modifikatoren stellen das Grundgerüst zur Veränderung dar. Das große Problem, das bei diesen Modifikatoren besteht ist, dass diese nur in Kombination

mit den generierten Strichen gut funktionieren. Im Prinzip verändern sie nur einen Pfad und müssen sich keine Gedanken machen über die Dicke der Striche oder wie diese zu Skalieren sind. Modifikatoren die auf eine Outline-Schrift angewendet werden können, müssten um ein vielfaches komplexer sein. Sie müssten in der Lage sein, die Konturen der Striche zu erkennen und diese richtig zu verarbeiten. Systeme wie Multiple master sind zum Teil in der Lage bei Interpolationen solche Dinge zu berücksichtigen, jedoch sind dies sehr aufwendige Verfahren. Eine Implementierung dieser Modifikatoren in ein Outline-System würde bei weitem nicht so einfach funktionieren wie dies im Prototyp möglich ist.

Ein weiteres Problem das Modifikatoren haben, ist dass sie viele Interaktionen erschweren. Sie verhindern, dass Objekte die an sie gebunden sind von Anwender verschoben werden. Um eine Glyphe im Nachhinein optisch zu verändern, ist eine Entfernung der Modifikatoren oft unumgänglich. Diese müssen dann im Anschluss wieder hinzugefügt werden.

6.5.3 Umfang der Arbeit

Die Arbeit befasst sich zum Teil mit sehr umfangreichen Themen, die nur knapp angeschnitten werden. Einzelne Bereiche würden sich durchaus anbieten ausführlicher behandelt zu werden. Der Umfang der Arbeit ist jedoch sehr groß gewählt worden, worunter einzelne Ausführungen zum Teil leiden. Die Grundidee und die gewählten Ansätze sollten jedoch ausreichend vermittelt werden. Eine Beschränkung auf einzelne Ansätze, hätte womöglich eine aufschlussreichere Arbeit ergeben, die nicht alle Aspekte des Prototyp beinhaltet hätte.

6.5.4 Schlussbemerkung

Die Erstellung von Schriften ist ein sehr komplexes Thema und wurde deshalb nur an der Oberfläche behandelt. Der Prototype zeigt jedoch sehr schön, dass es für Großbuchstaben teilweise möglich ist ein System zu schaffen, das diese flexibel beschreiben kann. Der Grad dieser Flexibilität ist dabei nicht all zu hoch aber doch ausreichend. Die einzelnen Teile sind sehr unausgereift und eigene sich kaum für die weiter Verwendung in ihrer jetzigen Form. Das Ziel war jedoch einen Prototypen zu schaffen, der zeigt wie so System funktionieren könnte und welche Ansätze dabei möglich sind. Es wurden zum Teil die technischen Grenzen und Probleme aufgezeigt. Eine Weiterentwicklung von Teilen des Prototypen könnten durchaus zu einem sinnvoll nutzbaren Ergebnis führen, dass abseits der Generierung von Schriftgerüsten liegt.

Quellenverzeichnis

Literatur

[1] Anonymous. „Charles University, Prague, March 1996 Questions and answers with Prof. Donald E. Knuth“. In: *TUGboat* 17.4 (1996), S. 355–367.

[2] Frank E Blokland. *Automating type design processes*. 2008. URL: http://www.dutchtypelibrary.nl/PDF/ATypI08/Automating_Type_Design.pdf.

[3] Frank E Blokland. *Harmonische Systemen binnen het latijnse Schrift*. 2009. URL: http://www.fonttools.org/downloads/LEMO/LeMo_301_MAC.zip.

[4] Karen Cheng. *Designing Type*. Yale University Press, 2006.

[5] Eric Freeman u. a. *Entwurfsmuster von Kopf bis Fuß*. O'Reilly Germany, 2006.

[6] Yannis Haralambous. *Fonts & Encodings*. O'Reilly Media, 2007.

[7] David Harris. *Die Kunst des Schreibens: Eine Anleitung zur Kalligraphie*. Dorling Kindersly Verlag, 2005.

[8] Adobe Systems Incorporated. *Designing Multiple Master Typefaces*. 1997. URL: http://partners.adobe.com/public/developer/en/font/5091.Design_MM_Fonts.pdf.

[9] Adobe Systems Incorporated. *PostScript language reference*. 3. Aufl. 1999. URL: http://www.adobe.com/products/postscript/pdfs/PLRM.pdf.

[10] Richard J. Kinch. „METAFOG: converting METAFONT shapes to contours“. In: *TUGboat* 16.3 (1995), S. 233–243.

[11] Donald Ervin Knuth. *The METAFONT book*. Addison-Wesley, 1986.

[12] Geerrit Noordzij. *Letterletter: An Inconsistent Collection of Tentative Theories That Do Not Claim Any Other Authority Than That of Common Sense*. Hartley und Marks Publishers, 2001.

[13] Walter Tracy. *Letters of Credit: A View of Type Design*. David R. Godine, 2003.

[14] Unknown. „Type basics“. In: *Pts.* 5 (2003), S. 48–86.

[15] Bruce Willen und Nolen Strals. *Lettering & Type: Creating Letters and Designing Typefaces (Design Brief)*. Princeton Architectural Press, 2009.

Online-Quellen

[16] *Anatomie der Buchstaben*. URL: http://www.designtagebuch.de/wiki/anatomie-der-buchstaben/ (besucht am 20.09.2012).

[17] Wolfgang Beinert. *typolexikon.de, Das Lexikon der westeuropäischen Typographie*. URL: typolexikon.de (besucht am 19.06.2011).

[18] *Bezier a primer*. URL: http://processingjs.nihongoresources.com/bezierinfo/ (besucht am 20.09.2012).

[19] *DIN 16518 Norm*. URL: http://de.wikipedia.org/wiki/DIN_16518 (besucht am 20.09.2012).

[20] *DTL LetterModeller*. URL: http://www.lettermodel.org (besucht am 20.09.2012).

[21] *Fontlab Editor*. URL: http://www.fontlab.com (besucht am 20.09.2012).

[22] *Fontstruct*. URL: http://http://fontstruct.com (besucht am 20.09.2012).

[23] *Glyphs Editor*. URL: http://glyphsapp.com (besucht am 20.09.2012).

[24] *Kalliculator*. URL: http://www.kalliculator.com (besucht am 20.09.2012).

[25] *Kalliculator Intro*. URL: http://www.youtube.com/watch?v=UXLWQpjizG0 (besucht am 20.09.2012).

[26] *Lettersoup Concepts*. URL: http://tinkerhouse.net/lettersoup/About/Concepts/ (besucht am 20.09.2012).

[27] *LUA Cocoa Bridge*.

[28] *Mac OS X AppKit Framework Developer Reference*. URL: http://developer.apple.com/library/mac/#documentation/cocoa/reference/ApplicationKit/ObjC_classic/_index.html (besucht am 20.09.2012).

[29] *Mac OS X Cocoa Developer Reference*. URL: https://developer.apple.com/technologies/mac/cocoa.html (besucht am 20.09.2012).

[30] *OpenType specification*. URL: http://www.microsoft.com/typography/otspec/default.htm (besucht am 20.09.2012).

[31] Luca Pacioli. *Luca Pacioli Letter M*. URL: http://en.wikipedia.org/wiki/File:Fra_Luca_Pacioli_Letter_M_1509.png (besucht am 20.09.2012).

[32] Luca Pacioli. *Luca Pacioli Letter T.* URL: http://commons.wikimedia.org/wiki/File:Fra_Luca_Pacioli_Letter_T_1509.png (besucht am 20.09.2012).

[33] *plaque découpée universelle.* URL: http://www.brookechornyak.com/index.php?/projects/open-works---type/ (besucht am 20.09.2012).

[34] *Python Script Language.* URL: http://www.python.org (besucht am 20.09.2012).

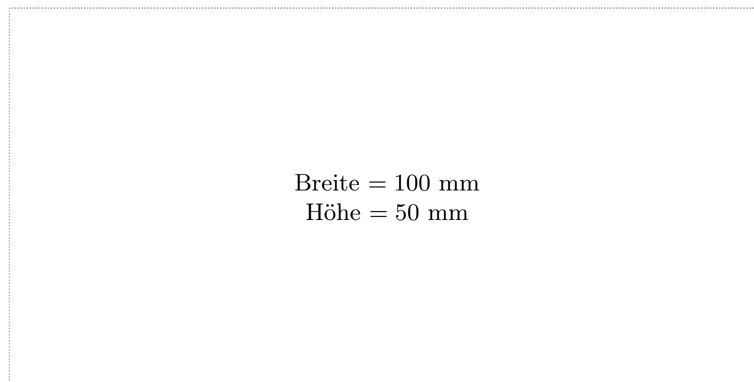
[35] *The AGG Project - Anti-Grain Geometry.* URL: http://www.antigrain.com/research/adaptive_bezier/index.html#PAGE_ADAPTIVE_BEZIER (besucht am 20.09.2012).

[36] *The Unicode Consortium.* URL: http://unicode.org (besucht am 20.09.2012).

[37] *Type design strategies with free software.* URL: http://river-valley.tv/type-design-strategies-with-free-software/ (besucht am 20.09.2012).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —