UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Schriftmaschine - Erzeugung von Schriftarten durch Parameter

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von
Dénes Weiß

Betreuer: Dipl.-Inform. Dominik Grüntjens
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Mai 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Schrift begegnet uns in unserem Alltag ständig in vielen verschiedenen Varianten und Ausprägungen, und dennoch ist sie nicht beliebig. Die Schrift auf einem Produkt, auf Werbung oder in einem Magazin ist von einem Grafiker, Layouter oder Typographen gezielt ausgewählt, um eine ganz bestimmte Aussage nicht nur über den Inhalt des Textes, sondern ebenso über die Gestalt des Textes zu übertragen. Nicht nur die Form, sondern zuweilen auch die Geschichte einer Schriftart kann in bestimmten Kreisen zu Assoziationen führen. [1, S.11]

Diese nicht triviale Aufgabe, eine passende Schriftart für einen ganz bestimmten Zweck auszuwählen, liegt in der Verantwortung des Typographen. Der Hohe Aufwand, der mit einem guten Schriftsatz verbunden ist, schlägt sich aber auch im Preis nieder. Der Preis für eine hochwertige Schriftart mit vielen Schriftschnitten kann bis in den vierstelligen Bereich ragen. Die Auswahl einer Schriftart bedeutet demnach oft, dass entweder sehr viel Geld für die passende Schrift ausgegeben wird, oder man sich mit einer nicht ganz zufrieden stellenden Lösung abfinden muss. Die Menge an guten kostenlosen oder gar Open Source Schriften wächst stetig, dennoch wird man auch hier nicht immer fündig.

Ein Werkzeug zur schnellen Erstellung individueller Schriftarten für die jeweiligen akuten Bedürfnisse wäre ein hilfreiches Instrument für Grafiker und Typographen. Die Anforderung für ein solches Instrument kann kaum sein, gute Schriftsätze zu erzeugen, dies liegt in den Händen des Gestalters, jedoch sollte sie jedem, der sich mit dem Thema befassen möchte, einen leichten Einstieg in die Gestaltung geben. Diese Arbeit versucht somit eine möglichst simple Lösung für das komplexe Thema der Schriftgestaltung zu liefern.

1.2 Einführung

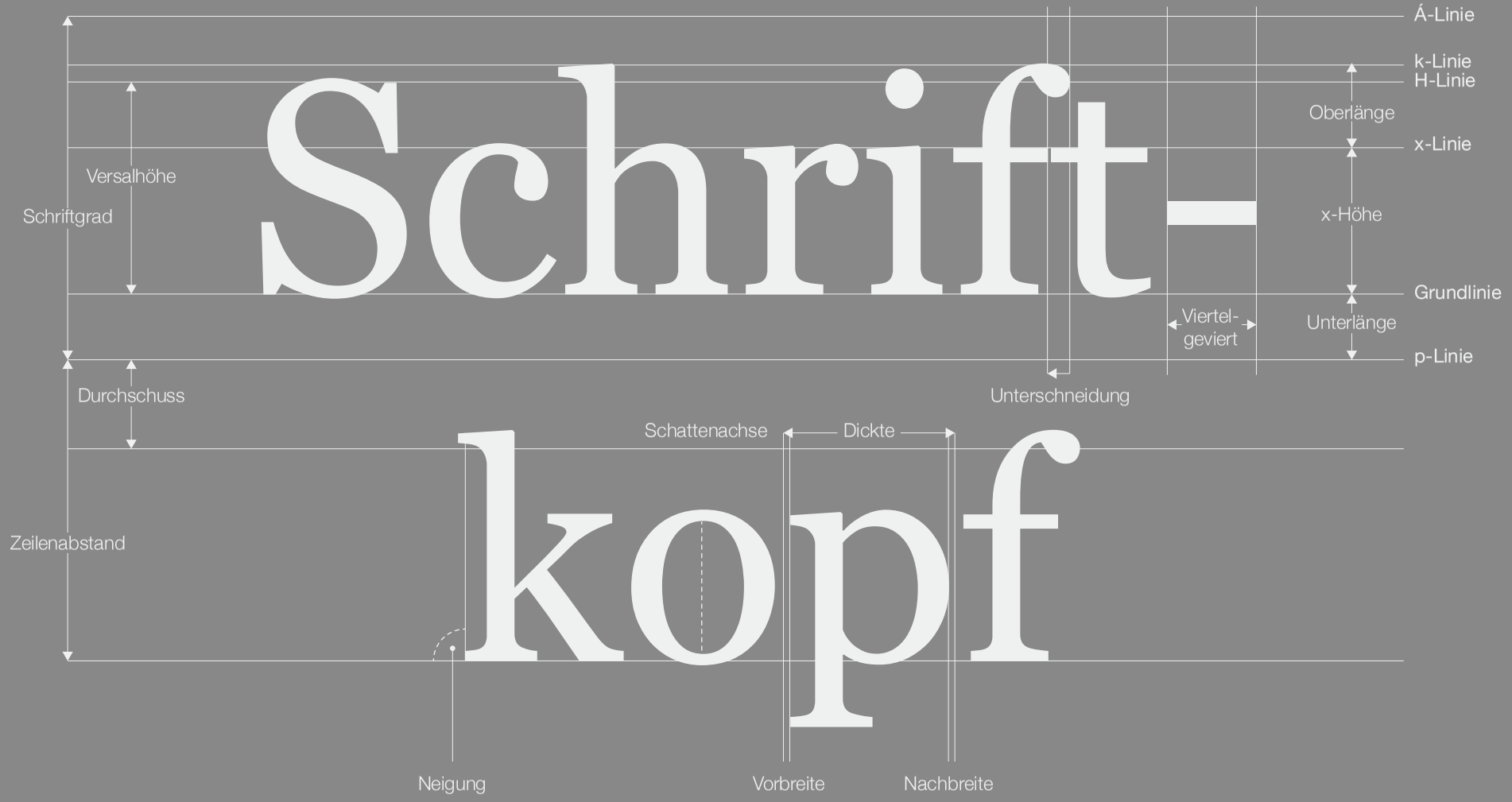
Das Erstellen einer Schriftfamilie ist mit heutigen Hilfsmitteln ein sehr aufwendiger Prozess. Abgesehen von dem Design einer guten Schrift, muss jeder Buchstabe einer Schriftart einzeln vektorisiert werden. Beim Vorgang des Vektorisierens werden die Konturen des Buchstabens mit Bézierkurven nachgefahren. Gute Software bietet hierfür ein automatieriertes Umranden an.

Eine Schriftfamilie benötigt für jeden Schriftschnitt eine andere Ausführungen jedes Buchstabens. Bei Schriftgewichten von Ultraleicht bis Ultrafett und Dickten von Ultraschmal bis Ultrabreit mit jeweils bis zu neun Unterteilungen plus Kursive, kommt man auf 162 Schnitte einer Schriftart.

Das neuste Update auf die Version 5.2 (Stand 18.05.2013) des Standardpro-

gramms zur Fontgestalung, FontLab Sudio, wirbt mit einem neuen Feature zur automatischen Berechnung der Buchstaben in verschiedenen Schriftgewichten.

Die Schriftmaschine soll einen anderen Ansatz verfolgen. Anstatt die Outlines der Buchstaben zu definieren, wird die Form jedes Buchstabens definiert und mit Hilfe eines veränderbaren Pinsels nachgefahren. Diese Methode mag für viele Schriftgestalter, die analog arbeiten, intuitiver sein, da sie so funktioniert, wie man es aus der realen Welt gewohnt ist.


Schrift-
kopf
Á-Linie
k-Linie
H-Linie
Oberlänge
x-Linie
x-Höhe
Grundlinie
Unterlänge
p-Linie
Versalhöhe
Schriftgrad
Durchschuss
Zeilenabstand
Viertel-
geviert
Unterschneidung
Schattenachse
Dickte
Neigung
Vorbreite
Nachbreite

2 Grundlagen

Für ein Programm, das Schriftarten erzeugen soll, müssen die Grundbegriffe der Typografie bekannt sein. Dieses Kapitel wird auf die Grundlagen der Typografie eingehen, ohne zu tief in die Hintergründe, Herkünfte oder Geschichte einzutauchen. Nach dieser Einführung in die Grundlagen der Typografie folgt eine kurze Wiederholung zu Bézierkurven und es werden die Möglichkeiten aufgezeigt, welche Fontformate zur Darstellung und Speicherung dititaler Fonts zur Verfügung stehen.

2.1 Typografie

Um Überlegungen darüber anstellen zu können, wie man eine Schriftart generieren könnte, muss man über die Zusammensetzung eines *Glyphen*, also der Repräsentation eines Buchstabens, und die Terminologie im Bilde sein. Typografie im allgemeinen beschäftigt sich mit der Gestaltung von Texten und deren Wirkung. Der Teilbereich der Typografie, der sich mit dem einzelnen Buchstaben befasst, nennt sich *Mikrotypografie*.

2.1.1 Anatomie der Buchstaben

Jeder Bestandteil eines Buchstaben findet in der Typografie eine Bezeichnung. Nicht nur die sichtbaren, sondern auch die ausgesparten Teile eines Glyphen haben eine Bezeichnung. Angefangen bei der noch bekannten *Serife* über die eher weniger bekannten *Tropen*, die rundlichen Endungen am kleinen r oder f, oder die *Cauda*, der Schwanz des großen Qs. Zu den nicht gedruckten Teilen der Schrift gehören zum Beispiel die vollständig oder teilweise geschlossene *Punze*, die quasi das Loch im p oder n bezeichnet, und das *Fleisch*, welches den Raum außerhalb des Buchstaben selbst benennt.

Da die Glieder der Buchstaben wenig Relevanz für diese Arbeit haben, wird hier nicht weiter auf sie eingegangen. Wer sich weiter in die Fachterminologie dieses Teils der Mikrotypographie einarbeiten möchte, kann dies in Wolfgang Beinerts Typolexikon [1] oder Achim Schaffrinnas Designtagebuch [2] tun.

2.1.2 Maße

In Abbildung 1 (links) sind die für die Schriftmaschine relevantesten Zeichenmaße zu sehen, die im Folgenden kurz erklärt werden sollen.

◀ **Abbildung 1 (links):** Abhängigkeiten in Frage kommender Parameter

[1] http://www.typolexikon.de

[2] http://www.designtagebuch.de/wiki/anatomie-der-buchstaben/

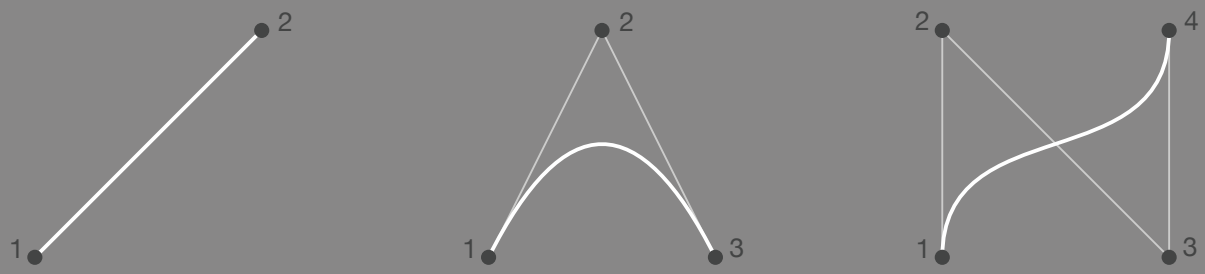


Abbildung 2: Eine Bézierkurve ersten Grades (links), 2. Grades (mitte) und 3. Grades (rechts)

Die vertikal eingezeichneten Linien zeigen das *Liniensytem*. Dazu zählt die *Á-Linie*, die die Obergrenze für Großbuchstaben mit Akzent bemisst. Direkt darunter die *k-Linie*, die die Minuskeloberlänge für Renaissance-Antiquas festlegt. Darunter steht die *H-Linie*, welche die Höhe der Majuskel aufzeigt. Die Größe der Großbuchstaben hat sich im Laufe der Zeit immer wieder nach oben und unten gewandelt,ïmmer mit der Begründung, das nütze der Ruhe der Zeilen und der Lesbarkeit."[1, S.24] Minuskel ohne Oberlänge werden durch die *x-Höhe* abgeschlossen. Auch hier macht sich mit der Zeit ein Trend fest, der die Minuskel tendenziell immer größer werden lässt. Die Linie auf der die Buchstaben stehen nennt sich *Grundlinie*. Zuunterst befindet sich die *p-Linie*, die die untere Grenze des Schriftsatzes festlegt.

Parallel zu den Höhenmaßen der Schrift gibt es die Angabe der Längen zwischen zwei der im Liniensystem beschriebenen Höhen. Der Raum zwischen Grundlinie und x-Höhe ist die *Mittellänge*. Die Länge von Grund- bis H-Linie ist die *Versalhöhe*. Der Bereich oberhalb der x-Höhe bis zur k-Linie nennt sich *Oberlänge*, unterhalb der x-Höhe bis zur p-Linie *Unterlänge*. Die gesamte Höhe der Schrift heißt *Schriftgrad*.

Maße, die Breiten in der Horizontalen bemessen, werden in Abbildung 1 mit senkrechten Linien umrandet. Die Breite eines Buchstabens heißt *Dickte*, wobei in ihr der Leerraum vor (*Vorbreite*) und hinter (*Nachbreite*) dem Glyphen enthalten ist. Bei Monospace-Schriften hat jede Glyphe die selbe Dickte. Die *Unterschneidung* beschreibt das Einrücken des Buchstaben, um die optische Distanz zweier aufeinander folgender Zeichen zu minimieren. Bei dem f und t in Abbildung 1 sieht man deutlich, wie sich der linke Arm des t unter den Tropen des f schiebt. Das Gegenstück dazu ist die *Spationierung*, dabei werden die Buchstabenabstände verlängert. Unter den Überbegriff *Kerning* fällt sowohl das Unterschneiden als auch das Spationieren. Das Kerning wird in Bruchteilen des *Gevierts* angegeben, welches das Quatrat mit dem Schriftgrad als Seitenlänge ist. [1, 2, 5]

2.2 Bézierkurven

Eine *Bézierkurve* ist eine durch den *de Casteljeau-Algotithmus* beschriebene Kurve. Sie verläuft immer durch den ersten und letzten Kontrollpunkt. Eine Bézierkurve ersten Grades hat zwei Kontrollpunkte und bildet die Gerade zwischen diesen.

Abbildung 3: Der OTF-Font Zapfino mit Ligaturen (oben) und ohne (unten)

Jede Bézierkurve höheren Grades erzeugt eine Kurve, die durch die mittleren Kontrollpunkte in ihrer Bahn gelenkt wird, wobei die lenkenden Punkte nicht auf der Kurve liegen, außer in dem Fall, dass alle Punkte auf einer Geraden liegen. Eine Bézierkurve vom Grad x hat genau $x + 1$ Kontrollpunkte. In Abbildung 2 sind Beispiele für Bezierkurven ersten bis dritten Grades zu sehen.

2.3 Digitale Schriftsätze

Die Wahl von Bézierkurven zur Darstellung von Schriftzeichen ist eine optimale Wahl in Hinblick auf die Möglichkeiten der Verwendung eines für Betriebssysteme verständliche Fonts. True Type Font (TTF), CCF-Opentype und Postscript Type1 verwenden alle Bézierkurven. [3] Wichtig für die Schriftmaschine soll Postscript Type1 werden, da Open Type Font (OTF) als Ausgabe für den erstellten Font angepeilt ist und OpenType Postscript Type1 verwenden kann.

Die Wahl zu OTF ist auf Grund verschiedener Vorteile gegenüber anderen Fontformaten gefallen. OTF verbraucht durch Optimierung zusammenfassbarer Eigenschaften wie die Unterschneidungen verschiedener Zeichen weniger Speicherplatz. Zudem ist eine gute Plattformübergreifende Verwendbarkeit gegeben. Der wichtigste Grund jedoch sind die vielfältigeren typografischen Fähigkeiten von OTF wie sprachspezifische Ligaturen und dynamische Zeichenkombinationen. Letzteres ermöglicht einer Schriftart für einen Glyphen mehrere Zeichen festzulegen, die je nach Buchstabenkombination anders aussehen. [4]

Abbildung 3 zeigt den Unterschied zwischen dem selben Font Zapfino mit und ohne Ligaturen und dynamischen Zeichenkombinationen. Im Gegensatz zu der unteren Zeile, in der vier Mal das selbe kleine f zu sehen ist, verwendet der Font mit Ligaturen vier verschiedene f. Bei dem Wort SZapfino"könnte man fast meinen es handele sich um zwei verschiedene Schriftarten. Die einzigen beiden Buchstaben, die identisch bleiben sind das kleine i und n. Auch bei SSchifffahrt"verändert sich neben den drei f auch noch die Verbindung zwischen dem r und t. Eigentümlich an diesem Schriftsatz ist, dass er sich kaum an seine typografischen Grenzen hält. Das langgezogene f in Zapfino ragt weit über die Maße des Schriftgrades hinaus. Der untere Schweif reicht bis unter die Versalhöhe der nächsten Zeile, wobei es oben in die Unterlänge der Zeile darüber ragt.

Abhängigkeiten in der Typografie

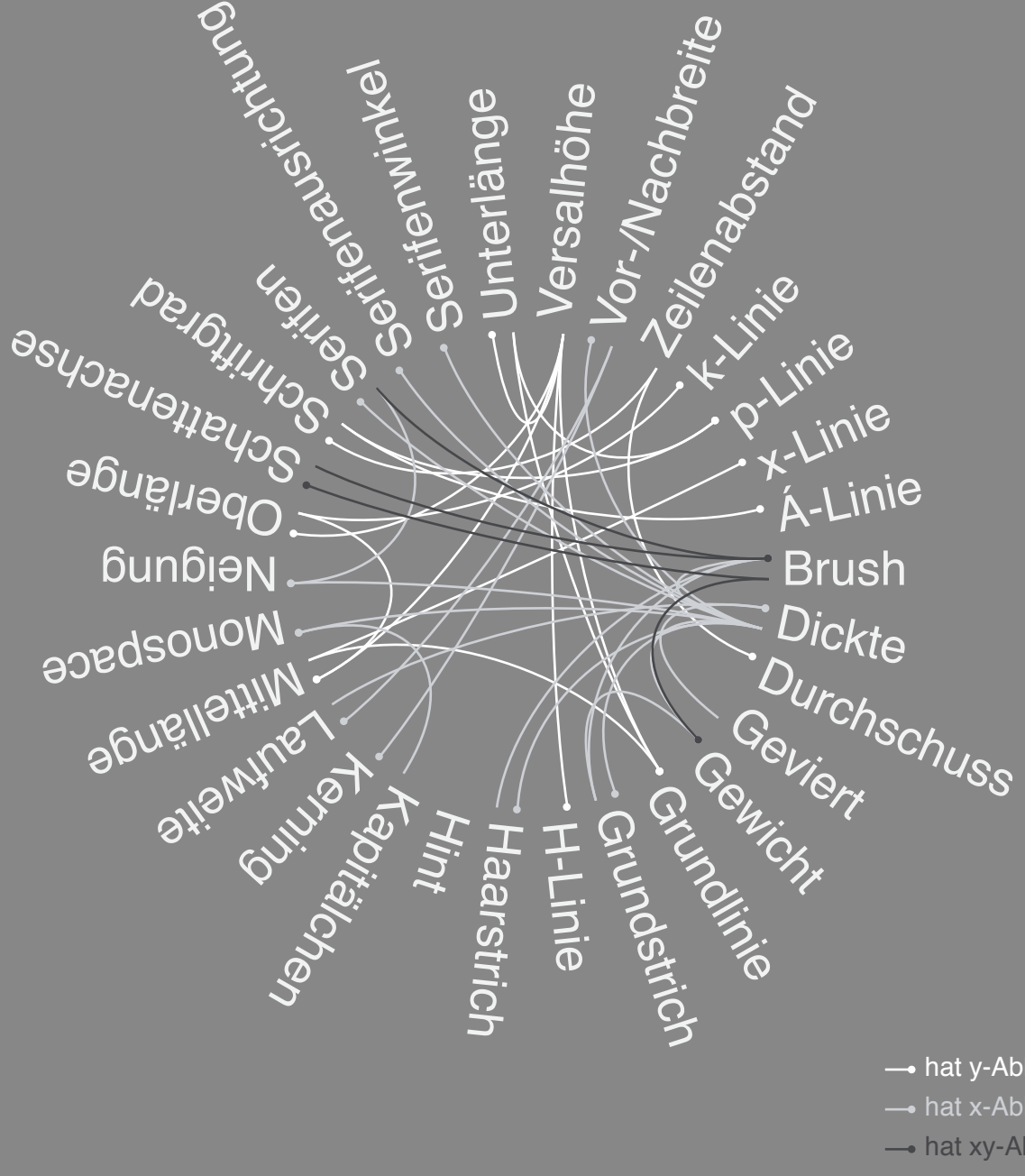


Abbildung 5: Das Schreibwerkzeug prägt deutlich die Form des Glyphen. Abbildung aus [1, S.14]

3 Entwicklung der Schriftmaschine

3.1 Die Idee

Die Grundform einer Schrift entsteht durch das verwendete Schreibwerkzeug. Es gibt viele verschiedene Werkzeuge, die alle eine andere Auswirkung auf die Schriftart haben, dadurch kam die Idee eine Art P̈inselz̈u benutzen, der verschiedene Formen annehmen könnte, um somit runde oder flache und breite Schreibutensilien zu imitieren. Die Auswirkungen der Wahl des Schreibwerkzeugs sind in Abbildung 5 deutlich zu sehen. Der Pinselwinkel würde dabei, bei einem nichtquadratischen Pinselpfad, die Schattenachse angeben. Feinheiten wie Serifen können durch das einfache Auswählen eines Pinsels leider nicht beachtet werden. Das zeigt die Grenzen, die bei dieser simplen Art der Schrifterzeugung bestehen. Um diese Grenzen zu überwinden, bedarf es Erweiterungen, die am Ende der Arbeit in Ausblick gestellt werden.

Um eine Schrift schnell den eigenen Vorstellungen anpassen zu können, sind global veränderbare Einstellungen unumgänglich. Anstatt jedes im Schriftsatz vorkommende Zeichen einzeln zu bearbeiten, sollte es die Möglichkeit geben, Einstellungen für alle zugleich zu setzen. Der Pinsel ist hierbei einer von vielen möglichen globalen Parametern, der einfach umzustellen ist und anschließend auf alle Buchstaben angewendet wird. Nur in seltenen Fällen ist es gewünscht, dass zum Beispiel das A sehr dünn mit abgerundeten Enden, das B hingegen mit einem breiten kantigen Pinsel geschrieben wird. Der Pinsel ist von allen noch der trivialste Parameter, denn er beeinflusst das Aussehen des Glyphen, ohne den Buchstaben selbst zu verändern. Eine globale Veränderung der Dickte oder des Liniensystems können zudem Einfluss auf die Kontrollpunkte der Bézierkurven nehmen.

Das Ziel ist es, ein einfach zu bedienendes Tool zur Erstellung einer Schriftart zu erstellen, mit dem man schnell erste Ergebnisse sieht. Einfach zu bedienen heißt unter anderem, dass der Nutzer dieses Programms nicht von einer Vielzahl an Parametern, die er setzen muss, erschlagen wird. In den folgenden Abschnitten wird erläutert, warum welche Parameter sich für das Erzeugen der Schriftart geeignet haben und welche ausgelassen wurden.

◀ **Abbildung 4 (links):** Abhängigkeiten in Frage kommender Parameter zur Erzeugung von Schriftarten.

3.2 Abhängigkeiten in der Typografie

Die in Abbildung 4 ablesbaren Parameter sind zu Beginn der Implementierung von einer Unmenge an möglichen Parametern übrig geblieben. Nicht alle sind direkt in der Schriftmaschine einstellbar. Zum Teil weil sie noch nicht implementiert wurden, zum anderen Teil, weil sie indirekt durch andere Einstellungen bedingt sind. So zum Beispiel der *Grund-* und *Haarstrich*. Sie beschreiben den etwas kräftigeren Abwärtsstrich und den meist sehr dünnen Aufstrich, den man beim handschriftlichen Schreiben mit einer breiten Feder erzeugen kann. Diese Eigenschaften werden indirekt durch die Form und den Winkel des Pinsels, mit dem die Bézierkurven nachgefahren werden, nachgeahmt. Ein hauchdünner jedoch breiter Pinsel um 45° geneigt, bewirkt genau einen maximal breiten Grundstrich und einen möglichst feinen Haarstrich.

Einige Abhängigkeiten sind kaum nachzuvollziehen und müssen eher festgelegt werden. Was passiert, wenn sich die Grundlinie verschiebt? Verschiebt sich die H-Linie um den selben Wert, weil die Versalhöhe des Schriftsatzes festgelegt wurde, oder verändert sich die Höhe, weil die H-Linie ihren bestimmten Platz hat? Eine klare Antwort darauf gibt es nicht. Für die Schriftmaschine wurde sich für letzte Variante entschieden. Die Verschiebung einer Linie im Liniensystem soll keinen Einfluss auf den Rest der Einstellungen haben. Dies würde schnell zu einer für den Nutzer unüberschaubaren Menge an Einflüssen führen.

3.3 Auswahl geeigneter Parameter

Die in Kapitel 2.1 eingeführten Begriffe der Typografie lassen sich grob in zwei Gebiete unterteilen. Zum einen in Buchstabenglieder, die sowohl sichtbare als auch ausgesparte Elemente von Glyphen bezeichnen. Die Eigenschaften dieser Elemente sind maßgeblich für die Form eines Buchstaben verantwortlich. Die Form bestimmt zum Beispiel wie rund der Bauch und wie schwunghaft die Cauda ist. Zum anderen gibt es verschiedene Längen-, Höhen- und andere Maßangaben, die das Schriftbild eines Fonts deutlich prägen.

Eine Form anhand von Parametern zu erzeugen ist überaus umständlich und komplex. Um eine kubische Bezierkurve zu definieren, benötigt es bereits acht numerische Parameter. Einen ganzen Buchstaben mit Koordinaten für Kurven und Geraden zu beschreiben, kann dem Nutzer nicht zugemutet werden und widerspricht auch der Idee eines möglichst einfach zu bedienenden Programms. Stattdessen verwendet die Schriftmaschine Bézierkurven, deren Kontrollpunkte graphisch bewegt werden können, ähnlich wie in gängigen vektorbasierten Zeichenprogrammen wie Adobe Illustrator und Inkskape.

Globale Parameter, die den ganzen Schritsatz beeinflussen, sollen in den Einstellungen zum Font verändert werden können. Dahingegen gibt es Parameter, die

pro Glyph oder gar auf einzelne Kontrollpunkte eines Glyphen angewendet werden müssen.

3.4 Benutzungsoberfläche und Bedienung

Es gibt zwei Ansichten, eine die einen Überblick über die Glyphen des Schriftsatzes gibt und eine Großansicht eines einzelnen Glyphen zum Bearbeiten der Form mit Hilfe von Bezier-Splines.

Zusätzlich werden auf der rechten Seite die aktuellen Einstellungen angezeigt. Die Einstellungen umfassen die Pinselbreite, -höhe und seinen Winkel. Eine Vorschau des Pinsels wird unterhalb der Schieberegler angezeigt. Unter den Pinseleinstellungen befinden sich Schieberegler für die Höhenangaben des Liniensystems, angegeben in Prozent des kompletten Schriftgrades.

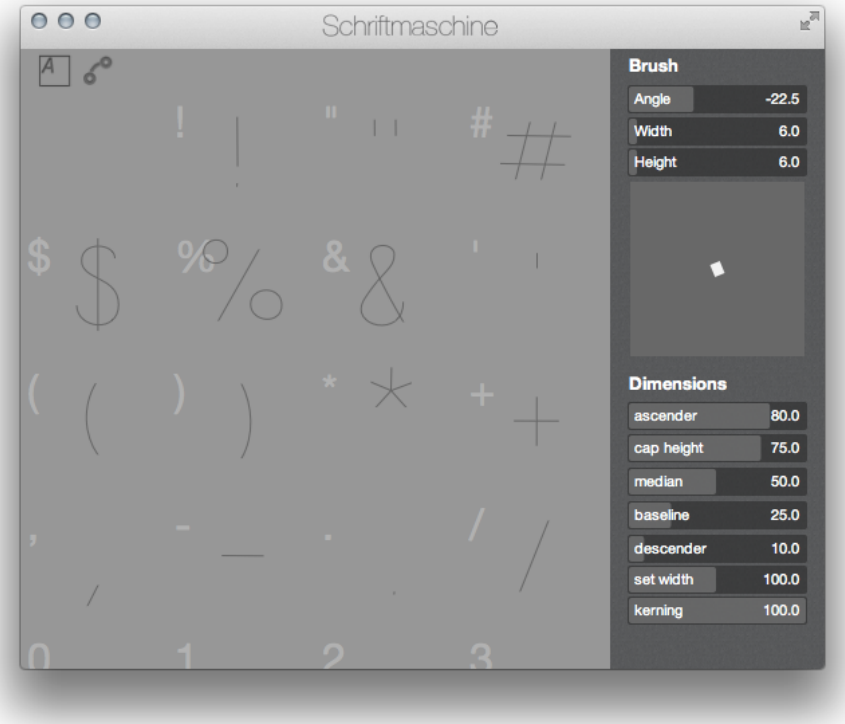


Abbildung 6: Schriftmaschine in der Gesamtübersicht.

Diese Einstellungen beziehen sich global auf den ganzen Schriftsatz. Wird die x-Höhe oder der Winkel des Pinsels verändert, so werden die neuen Einstellungen auf alle Glyphen im Fontsatz angewendet.

Wird in der Glyphenansicht ein Kontrollpunkt des Bézier-Splines ausgewählt, ändert sich die Einstellungsübersicht, sodass nur Parameter, die für den einzelnen Punkt relevant sind, angezeigt werden. Zuerst werden die Koordinaten des Punktes angezeigt, darunter eine Ankreuzliste der Höhen, mit denen der einzelne Punkt verknüpft werden kann. Diese Verknüpfung ist monodirektional. Wird ein Punkt mit der x-Höhe verknüpft, so wird der Punkt automatisch verschoben, wenn sich der Wert der x-Höhe verändert. Andersherum hat die Höhe des Punkts keinen Einfluss auf die globale Variable der x-Höhe.

Die Toolbar am oberen Rand des Programmfensters passt sich der aktuellen Ansicht und den damit möglichen Einstellungsmöglichkeiten an. In der Fontübersicht hat man die Möglichkeit, die zu repräsentierenden Buchstaben eines Glyphen und die Bézier Splines, die die Form der Glyphen beschreiben, zu verstecken.

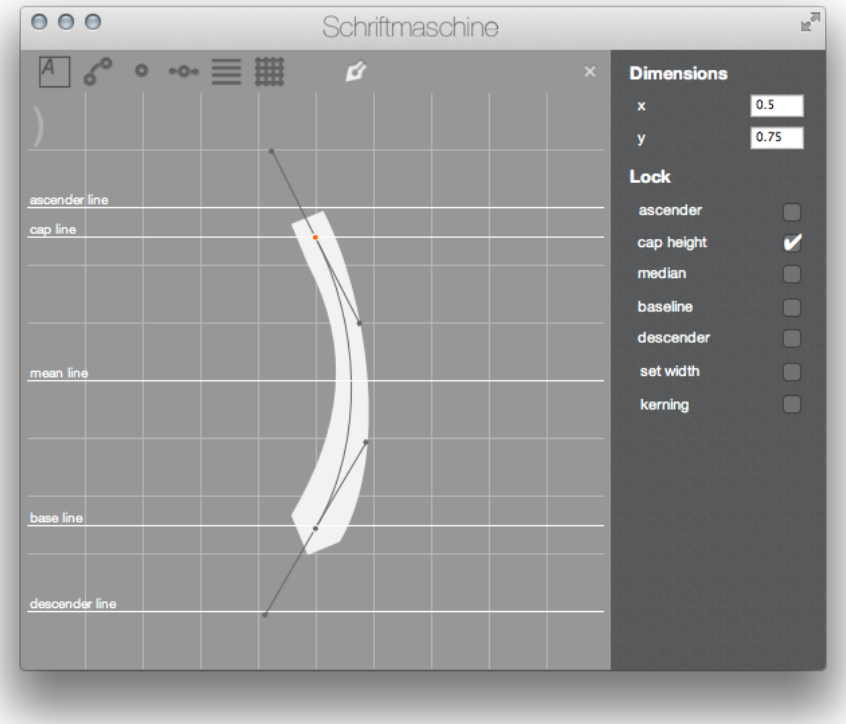


Abbildung 7: Schriftmaschine in Glyphansicht mit ausgewähltem Kontrollpunkt.

In der Glyphansicht kommen weitere Toolbarsymbole hinzu. Die ersten Symbole schalten weitere Ansichtseigenschaften ein und aus. So hat man zwei Einstellungsmöglichkeiten, Kontrollpunkte der Bézier-Kurven anzuzeigen oder aus-

zublenden. Zudem kann ein Gitter aus Hilfslinien, das in Zehntel der gesamten Ansicht unterteilt ist, eingeblendet werden. Das Einblenden des Liniensystems erleichtert die Platzierung der Kontrollpunkte auf die gewünschte Höhe. Das Liniensystem repräsentiert immer die aktuellen Einstellungen der Höhenangaben für x-Höhe, Versalhöhe usw.

Zuletzt zeigt die Toolbar das Pfadtool, mit dem Bézierpfade gezeichnet werden. Wie in gängigen vektorbasierten Grafikanwendungen, kann man Punkte durch Klick setzen. Durch klicken mehrerer Punkte wird ein Linienzug erzeugt. Bei gedrückt halten und bewegen der Maustaste werden Bézierpunkte mit zusätzlichen Kontrollpunkten erstellt. Die Kontrollpunkte beschreiben die Biegung der entstandenen Bézier-Kurve. Die Schriftmaschine verwendet ausschließlich kubische Bézierkurven. Um einen neuen Linienzug, der abgetrennt von der aktuellen Kurve dargestellt wird, zu erzeugen, muss das Pfadtool deselektiert und nochmals ausgewählt werden. Andernfalls würde eine durchgehende Kurve entstehen. Dies ist leider wenig intuitiv, lässt sich aber leider nicht eindeutig besser lösen. Andere Lösungsansätze, wie Shortcuts zum unterbrechen der Linie oder ein extra dafür ausgelegter Button sind nicht weniger unintuitiv.

3.5 Frameworks

3.5.1 Darstellung der Glyphen

Für die Darstellung der Bezierkurven für die Glyphen kamen zwei Alternativen in Frage. Erstens die Darstellung mittels OpenGL. Zweitens die im Cocoa-Framework enthaltene Klasse der `NSBezierPath`. Angesichts der Tatsache, dass das Cocoa-Framework ohnehin für die GUI benutzt wird und im Gegensatz zu OpenGL Bézierkurven ohne Mehraufwand zur Verfügung stellt, lag es nahe, den Weg mit dem wenigsten Aufwand zu nehmen.

Die Performance der dargestellten Glyphen steht nicht im Vordergrund. Die Darstellung der Glyphen hat bislang keine bemerkbaren Rechenkapzitäten verbraucht. Der Fall, dass der komplette Unicode Zeichensatz ausgenutzt würde, konnte im Zuge dieser Arbeit nicht getestet werden. Sollten sich unbenutzbare Zustände einstellen, sollte man die Verwendung von OpenGL in Betracht ziehen.

Die Klasse `NSBezierPath` beinhält keine überschreibbare Klasse für Kontrollpunkte, noch für die Pfadsegmente. Stattdessen arbeitet `NSBezierPath` mit einem C `struct NSPoint`, das nur 2D Koordinaten speichert. Zudem weiß `NSBezierPath` an einem gegebenem Index entlang der Kontrollpunkte um welche Art von `enum NSBezierPathElement` es sich handelt. Das Pfadsegment könnte sich zu den aktuellen Koordinaten bewegen, eine Linie dorthin ziehen oder eine Kurve ziehen.

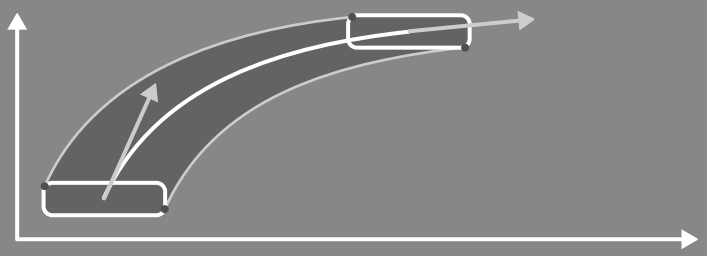


Abbildung 8: Der Pinselstrich setzt sich aus drei Pfaden zusammen

Die Kontrollpunkte, die den Bézierpfad des Glyphen beschreiben, beinhalten jedoch mehr Informationen als die reinen x- und y-Koordinaten. Jede Klasse `SMGlyph` besitzt ein Array von `SMPoint`s, die während des Programmablaufs verändert werden. Beim Rendern des Glyphen wird ein NSBézierPath anhand der Punkte im Array der `SMGlyph`s erstellt und gezeichnet.

3.5.2 Boolesche Operationen auf Bezierkurven

Der Vorgang des Erzeugens der Kontur für jeden Glyphen läuft wie folgt ab:

- `NSBezierPath` für die gezeichnete Form des Glyphen aus den vorliegenden `SMPoint`s erzeugen.
- Anhand der Tangente des soeben berechneten Bézierpfads und des Pinselpfads wird ein Pfadsegment nach dem anderen mit einer Form versehen. (Auf diesen Punkt wird in 3.6.1 näher eingegangen)
- Die einzelnen Formen, die die mit dem Pinsel entlang gefahrenen Pinselstriche darstellen, müssen zu einem Pfad vereint werden, da bei der Darstellung sich überschneidender Pfade sonst Löcher entstehen.

Für die letzte, nichttriviale Aufgabe muss eine Library herhalten. Bei der Suche nach in Frage kommenden Librarys standen nur zwei zur Auswahl. Erstens GPC - General Polygon Clipper Library [1] von Adam Murta, zweitens Andy Finnells VectorBoolean [2] Open Source Code.

Auch hier fiel die Entscheidung auf die am schnellsten verwendbare Lösung. GPC ist für viele Sprachen verfügbar, leider nicht für Obj-C. Die Verwendung dieser Library hieße, dass die bereits berechneten `NSBezierPath`s wiederum umgerechnet werden müssten, damit sie als Eingabe der Library verwendet werden können. VectorBoolean hingegen ist in Obj-C geschrieben und verlangt zum vereinen zwei `NSBezierPath`s. Das heißt kein Umkopieren der Pfade.

Der Beispielcode von VectorBoolean schien auch vielversprechend. Formen mit Rundungen und Kanten wurden problemlos miteinander verundet und verodert. Die Verwendung des Codes ist auch höchst simpel. VectorBoolean enthält

[1] http://www.cs.man.ac.uk/ toby/gpc/
[2] https://bitbucket.org/andyfinnell/vectorboolean

eine `NSBezierPath` Category. Categorys ermöglichen es einer Klasse, weitere Methoden zu injizieren. Nach dem Import der Category Header-Datei NSBezier-Path+Boolean.h verfügt die Klasse `NSBezierPath` nun also über vier weitere Instanz-Methoden. Einer Union-, Intersect-, Difference-, und XOR-Methode. Die Schriftmaschine verwendet von diesen Methoden nur die Vereinigung.

Der simulierte Pinselstrich besteht aus drei einzelnen Pfaden. Jeweils ein Mal die Pinselform, transformiert an die Koordination des ersten und letzten Kontrollpunktes einer Bézierkurve, und die Form zwischen den beiden Kontrollpunkten, wie in 8 zu sehen, die sich aus der Pinselform und der Bézierkurve ergibt, die die Form des Glyphen angibt.

Nach dem Einbinden des Codes fielen sofort die ersten Fehler auf. Verschiedene Teile der vereinigten Pfade verschwanden scheinbar wahllos. Das Vereinigen von mehr als drei Kurven brachte den Vereinigungsalgorithmus in eine Endlosschleife. Geraden waren, wenn auch falsch, mit beliebig vielen Punkten darstellbar.

Glücklicherweise kam zwei Wochen vor Abgabe der Arbeit nach einem Jahr Commit-Pause ein Update heraus. Dieses Update behob die meisten Probleme. Auch Kurven sind nun vereinbar. Es entstehen aber noch immer Löcher in den Pfaden, wo keine sein sollten. Umgekehrt werden auch Löcher gefüllt, die leer bleiben sollten. Auch wenn der Code nun besser läuft, waren gut zwei Wochen Debug-Versuche umsonst.

3.6 Implementierung

Auf Grund des Cocoa Frameworks ist die Arbeit in Objective-C geschrieben. Cocoa bietet unter Mac OS X eine Fülle an leicht zu integrierenden GUI-Elementen, die auf die eigenen Bedürfnisse anpassbar sind. Die Verwendung des Frameworks hilft auch dabei, eine übersichtliche Struktur im Code beizubehalten, da das Model-View-Controller-Konzept stark durchgesetzt wird. Die Dokumentation ist in den meisten Bereichen hervorragend. Die Verwendung mancher View-Klassen jedoch wird nur halbherzig erläutert, da das Benutzen des Interface-Builders, mit dem die GUI in einem WYSIWYG Editor im Baukastenprinzip zusammensetzbar ist, stärker gefördert wird, als die programmatische Umsetzung der Benutzungsoberfläche. Das hat zwar den Vorteil, dass ein Programm in manchen Fällen mit nur wenigen Zeilen Code entstehen kann. Der Nachteil liegt aber auf der Hand; Wer sein Programm lieber als Code liest, anstatt bei eventuell auftretenden Fehlern in GUI Einstellungen herumzusuchen, wird von der Dokumentation links liegen gelassen.

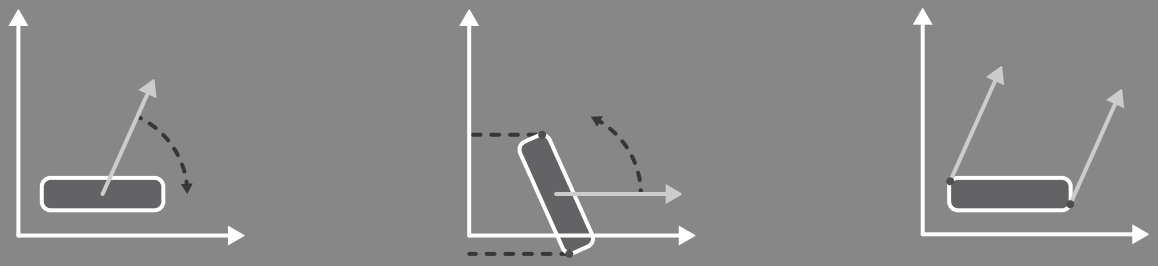


Abbildung 9: Graphische Darstellung zur Berechnung der Schnittpunkte für Pinselstriche
Schirtt 1: Pinselpfad um den Winkel der Tangente rotieren. (links)
Schritt 2: Höchsten und niedrigsten Punkt auf der Y-Achse finden. (mitte)
Schritt 3: Die zwei gefundenen Punkte zurück rotieren. (mitte)

3.6.1 Berechnung der Glyphenpfade

Die Suche nach einer wissenschaftlichen Arbeit über das Zeichnen mit Vektorpinseln an Bézierkurven entlang, war leider erfolglos. Also musste ein eigener Ansatz her.

Der erste Ansatz versuchte die Kontur des Glyphen im Ganzen zu errechnen. Schnittpunkte von sich kreuzenden Pfaden müssten abhängig von der Strichdicke berechnet werden. Auch Fälle, in denen Pfade sich nicht kreuzen, der Pinselstrich aber dick genug wäre, um dass die Konturen sich berühren, müssten berücksichtigt werden.

Der Aufwand für das Vorhersehen möglicher Kollisionen schien zu hoch, also musste ein neuer Ansatz her. Anstatt den kompletten Glyph in einem Durchgang zu berechnen, werden Liniensegmente nach und nach erzeugt. Dabei muss nicht auf eventuelle Überschneidungen geachtet werden. Sind alle Segmente berechnet, müssen diese am Ende vereinigt werden, um die Kontur des Glyphen zu ergeben.

Um den Pinselstrich anhand eines Bézierkurvensegments und der Form des Pinselpfads zu erzeugen, werden drei einzelne Pfade benötigt. Wie in 3.5.2 bereits erwähnt, wird der Pinselpfad einmal um die Koordinaten des ersten und letzten Kontrollpunktes einer Bézierkurve verschoben. Dazwischen soll eine Form entstehen, die dem gezogenen Pinselstrich entspricht. Dafür müssen die Eckpunkte dieser Form gefunden werden. Sie befinden sich auf den Pinselpfaden. Die Eckpunkte sind die von der Tangente eines jeden Kontrollpunkts senkrecht am weitesten entfernten Punkte, die noch auf dem Pinselpfad liegen. Diese Punkte gilt es zu finden.

Die folgende Lösung, die grafisch in Abbildung 9 nachzuvollziehen ist, findet genau die gesuchten Punkte.

1. Drehen des Pinselpfads um den Winkel der Tangente auf die x-Achse.

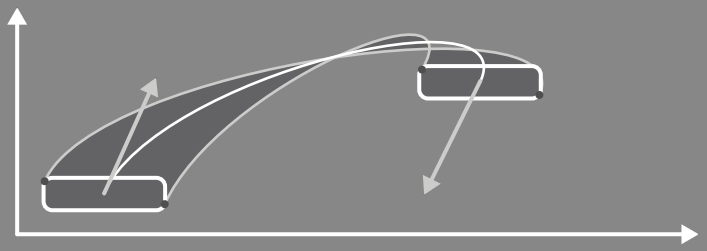


Abbildung 10: Die Konturen des Pinselstrichs überschneiden sich

2. Den ganzen Pfad ablaufen und den höchsten und tiefsten Punkt entlang der Y-Achse merken.
3. Die gefundenen Punkte um den selben Winkel zurück rotieren.
4. An die gefundenen Punkte werden Bézierkurven mit der selben Ausrichtung wie die der Ausgangskurve gesetzt.

Dieses Verfahren erzielt in bestimmten Fällen unerwünschte Ergebnisse, die in Abbildung 10 zu sehen sind und in 4.2 näher erläutert werden.

3.6.2 Verknüpfung der Kurven mit dem Liniensystem

Um Veränderungen an den Einstellungen des Liniensystems an die verknüpften Kontrollpunkte weiterzugeben, wurde das `NSNotificationCenter` zu Hilfe genommen. Wenn ein Punkt in den entsprechenden Einstellungen das Häkchen gesetzt kriegt und von nun an mit einem der Höhenangaben im Liniensystem verbunden ist, erhält die Klasse die Aufgabe auf Benachrichtigungen aus dem Notification Center zu hören, die dem verändern der jeweiligen Linie entsprechen. Ein großer Vorteil dieses Systems ist, dass nicht über alle im Schriftsatz enthaltenen Punkte iteriert werden muss, um die Punkte zu finden, die eine Verknüpfung aufweisen. Auch wird keine weitere Container-Klasse benötigt, die alle Punkte beinhaltet. Stattdessen werden nur genau die Kontrollpunkte aktiv, die man dazu aufgefordert hat.

4 Fazit

4.1 Ergebnisse

Es ist gelungen, Formen für Glyphen mittels Bézierkurven zu zeichnen. Aus einer Übersicht über alle Zeichen des ASCII-Zeichensatzes kann man sich den zu bearbeitenden Buchstaben oder eine Zahl auswählen und seine eigene Repräsentation dieses Glyphen erzeugen. Verschiedene ein- und ausblendbare Hilfslinien helfen bei der Gestaltung der Zeichen. Mit Hilfe eines Pinsels, der im Moment immer rechteckig, aber in Breite, Höhe und Winkel einstellbar ist, verleiht man den Kurven eine Form.

Sind die Kurven für Zeichen festgelegt und wie gewünscht eingestellt, lässt sich die Form auch im Nachhinein einfach durch Anpassen des Liniensystems verändern.

Die erzeugte Schriftart lässt sich in eine für die Schriftmaschine lesbare Datei schreiben und wieder laden. Alle Einstellungen sind so, wie beim Speichern des Projekts, doch der Versuch eine Einstellung zu ändern führt noch zum Absturz. Vom Betriebssystem lesbare Fontformate können noch nicht exportiert werden.

4.2 Darstellungsfehler

Darstellungsfehler treten auf verschiedenen Ebenen auf. Einige Fehler sind bekannt, der Zeitmangel hat verhindert, dass diese behoben werden. Andere Fehler sind zwar bekannt, eine sinnvolle Lösung existiert aber noch nicht. Und Fehler aus dem verwendeten Framework sind zwar sichtbar, jedoch käme das Debuggen mehr einem Ratespiel gleich.

Beim Erzeugen des Pinselstrichs muss zwischen den Übergängen von einem zum nächsten Segment unterschieden werden. Der Pinselstrich von einem Liniensegment zu einem Liniensegment wird anders behandelt als ein Liniensegment zu einem Kurvensegment, und wiederum anders ist der Fall Kurvensegment zu Liniensegment. An dieser Stelle tritt ein Fehler bei der Reihenfolge der Punkte auf.

Ein Problem zu dem noch keine Lösung gefunden wurde ist, dass die Kurven der Kontur eines Liniensegments sich in bestimmten Fällen überschneiden. In Abbildung 10 ist zu sehen, wie sich die Bézierkurven überkreuzen. Dies sollte nicht passieren. Das Problem ist, dass die Auslenkung der inneren Kurve zu stark, die der äußeren Kurve hingegen zu schwach ist. Jeder Punkt auf der Bézierkurve beeinflusst den Verlauf der Kurve, was dazu führt, dass Kurven mit selber Ablenkung, deren Kontrollpunkte aber weiter voneinander entfernt liegen, flacher sind, als Kurven der mit selber Ablenkung aber engerem Abstand. Ein Lösungsansatz könnte sein, dass die Tangenten an den Kontrollpunkten der Kontur anders gewählt werden müssen, als die der formgebenden Bézierkurve. Wie genau diese verändert werden müssten, um ein gutes Ergebnis zu bekommen, steht noch offen. Im allgemeinen ist die Berechnung des Pinselstriches nicht ausreichend ausgereift.

Der Code von Andy Finnell zur Vereinigung von Bézierkurven setzt noch einen oben drauf. Es schließt Pfade und schneidet Teile des Pfades ab. Da der Code auch in dem kommerziellen Programm Artboard [4] verwendet wird, kann man davon ausgehen, dass die Methoden durchaus verwendbar sind, aber die übergebenen Bézierpfade bestimmte Eigenschaften erfüllen müssen.

[4] http://www.mapdiva.com/artboard/artboard-gallery/

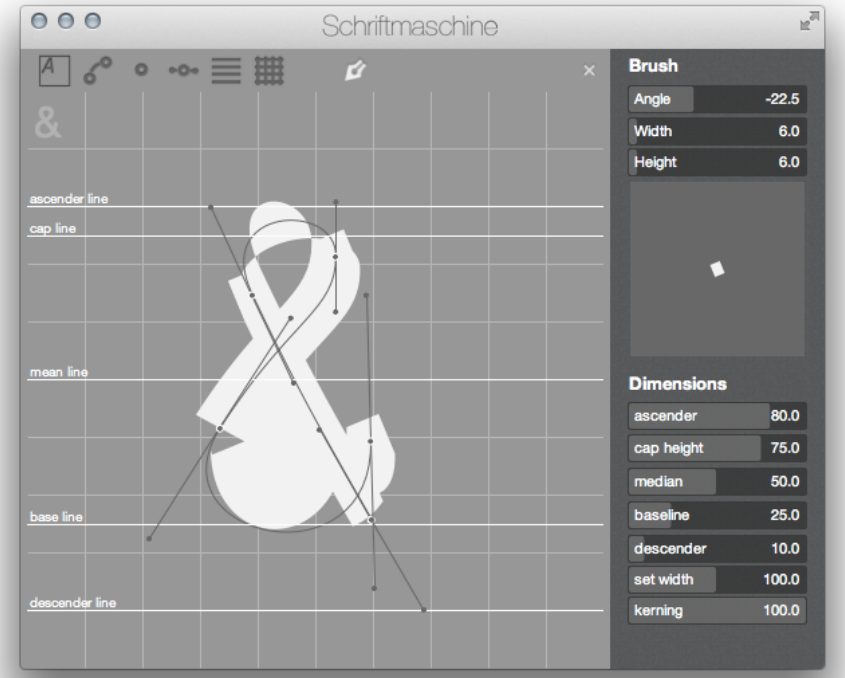


Abbildung 11: Zu sehen sind die meisten Fehler, die beim rendern des Glyphen entstehen.

In Abbildung 11 sind die sich überschneidenden Konturen und die ungewollt gefüllten Bereiche gut zu sehen. Die untere Kurve des & fehlt komplett. Es ist kaum nachvollziehbar, welche Teile abgeschnitten und ungewolltermaßen hinzugefügt wurden.

4.3 Ausblick

Der aktuelle Stand der Software ist noch nicht zum Erstellen von Glyphen, die man im System weiterverwenden könnte, geeignet. Die folgenden Erweiterungen wären wünschenswert, um ein brauchbares Programm präsentieren zu können:

- Die Zeichenwerkzeuge sollten erweitert werden, um zum Beispiel geometrische Formen zeichnen zu können
- Pfade sollten geschlossen werden können. Aktuell gibt es visuell geschlossene Pfade durch Doppelklick auf den ersten Kontrollpunkt eines Pfades. Dies legt aber lediglich den letzten Kontrollpunkt auf den ersten, ohne sie zu verbinden.

- Die Schriftart muss in OTF abgespeichert werden können, damit sie nützlich sein kann.
- Das Zurichten einer Schrift, also das gezielte Unterschneiden bei ausgewählten Zeichenpaaren ist für einen angenehm lesbaren Font unumgänglich. Die Schriftmaschine sollte diesen Parameter einstellen können.
- Punkte können mit dem Liniensystem verankert werden, sodass der Punkt sich mit der Verschiebung z.B der H-Linie mitbewegt. Eine sinnvolle Erweiterung wäre das Verknüpfen von Punkten mit Pfaden und anderen Punkten.

Es sind durchaus noch viele weitere Erweiterungen Denkbar, die man diesem Programm beifügen könnte, jedoch ginge dies weit über das parametrisierte Erzeugen von Schrift hinaus.

Literatur

[1] Hans Peter Willberg, *Wegweiser Schrift: Erste Hilfe mit dem Ungang mit Schriften. was passt - was wirkt - was stört*, Verlag Herrmann Schmidt, Mainz (2001)

[2] Internetquelle: Ehringer, *Mikrotypografie*, typo.ronn23.de/?pid=16 (abgerufen am 08.05.2013)

[3] Internetquelle: Wikipedia, *Bézierkurve*, de.wikipedia.org/wiki/Bézierkurve#Anwendung (abgerufen am 18.05.2013)

[4] Internetquelle: Wikipedia, *OpenType*, de.wikipedia.org/wiki/OpenType#Vergleich_mit_TrueType_und_PostScript (abgerufen am 18.05.2013)

[5] Internetquelle: Wolfgang Beinert, *Typolexikon*, www.typolexikon.de (abgerufen am 18.05.2013)