FONTFORGE 与字体设计

介绍如何使用 FONTFORGE创造新字体的书

Published with GitBook

目錄

引言

本书编写的目的是帮助人们参与字体设计流程。字体设计看起来很复杂，就像高科技一样。但是现在字体制作比以前更为容易，一方面是由于自由免费工具的出现，比如FontForge。FontForge是一个适合起步的便利工具，但并不仅仅适用于初学者。它拥有一系列高级工具，并且在本书编写的同时正在快速改进中。

本书的目标，是为打算进行字型设计的项目提供技术帮助和基础知识，同时提供如何让你的工作流程更加高效的建议。

如果你希望帮助我们，你可以通过在GitHub上给予反馈，甚至贡献内容与修订错误。 如果你在FontForge中遇到任何bug，请搜索Github上的issue tracker，确定你遇到的是否是已知bug，查看目前解决状态。如果不是，在这里查看报告bug的指导。

我们希望你能够享受阅读这本指南的过程，就像我们写的这本书一样。

— FontForge项目贡献者

什么是字体

什么使得手写、书法、印刷和Logo的字体样式不同?

使字体设计不同的一个最大的问题是需要字体样式中的每一个字形与其他字形的一起工作。这通常意味着字体样式中的每一部分的设计和间距最终要成为仔细折中的一个系列。这些折中意味着我们最好能够将字体样式设计看作是创造一个字母的美妙集合而不是一个美妙字母的集合。换句话说我们必须思考整体以及他们如何一起运转，并且其优先级要高于一个字母的精妙之处的问题。

这需要我们要使系统优先于每一部分，也导致了我们在系统的层面来分析我们的设计进程。特征超越字母成为我们想要关注的东西，尤其是在设计进程初期。

在字体设计中其他怪异的地方在于我们设计的形式在很大程度上已经显著建立。我们设计字体的任务并没有建立全新形式那么重，而是建立一个已有形式的新版本。这可能使新的字体设计者感到困惑。找到合适数量的改变来激励他们同时也不使读者感到疏远使一件棘手的事情。设计者通常会陷入具体字母的思考。如果你一开始意识到字体样式中最有意义的是其中重复最多的那些组成部分，那么能够很容易地避免这个错误。字体样式设计不是仅是关于设计特征应用在我们所认识的通用形式上，也是应用在最常出现的形式上。

特征不仅有助于创造字体的声音或者氛围，也能确定字体对什么有用或者没用，并且又是也有助于确定一个字体所适合的技术背景。认同这些是有帮助的。

按照这种方式思考字体设计可能是吓人的和过度抽象的。但是习惯这些想法是使得字体设计进程更加快速、高效和令人满意的关键。

让我们将识别字体设计的主要的系统特征作为开始。

结构

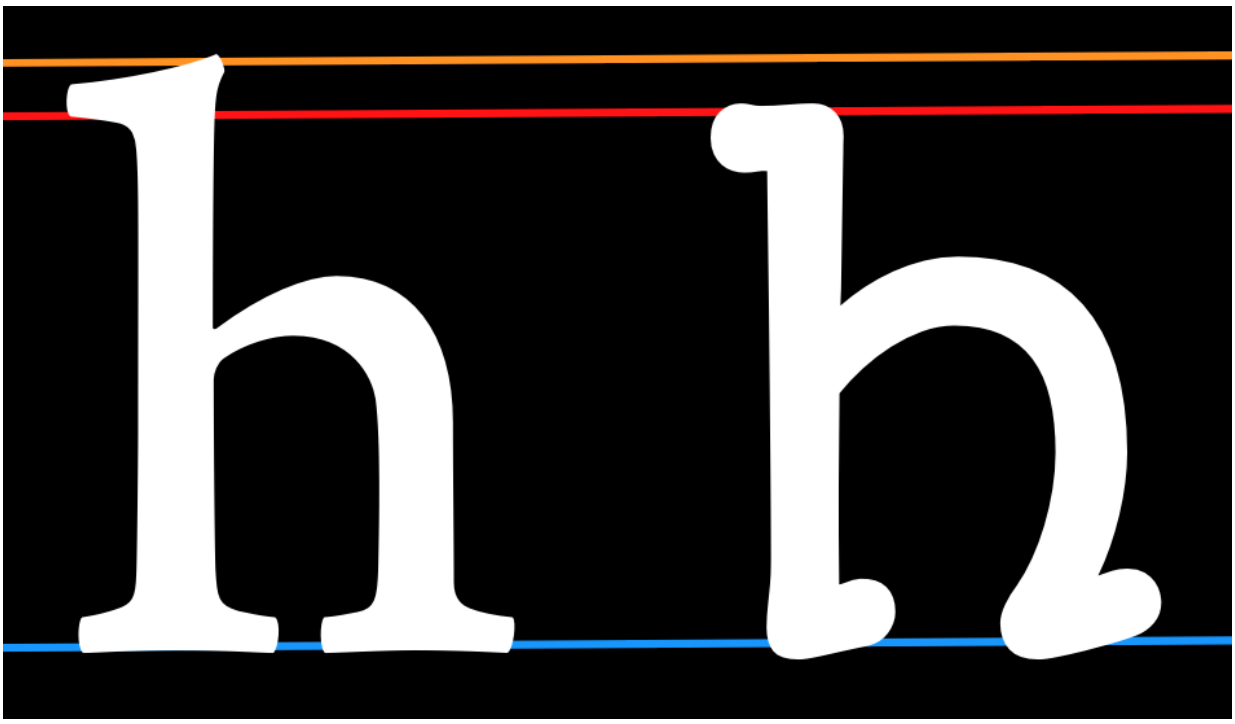
结构指的是形成特定字形的基础笔画的构造。你可以想象是字形的骨架。可以说采用什么构造是需要思考的最重要的问题之一，因为结构影响到如此多余下的选择，尤其是当你的设计需要让读者感到几分熟悉的时候。在上述例子中，字母中白色的线条指示出字母本身所暗含的粗略估计的结构。

但是末端的方式（终止节点）和衬线（如下）通常不是“结构”的含义的一部分。结构是一个字形的骨架，而剩下的 — 宽度、粗细、末端 — 都是血肉的一部分。

X高度到大写高度的比例

左边的字母来自Playfair Display，相比于大写高度，x高度较大。右边的字母来自EB Garamond，X高度较小。上面的例子中，字母H的高度被调整过，这样他们可以对齐。

顶高



上面的例子中，为了演示在顶高上的相对差别，我们对齐了X高度。

顶高经常至少超过大写高度一点，尤其是在文字设计中。但是在某些情况下他们可能相等或者顶高比大写高度更低。更高的顶高能够让字体样式看起来更加优雅。他们通常X高度更小。

底深

类似于顶部，更长的底部也能让人觉得更加优雅。

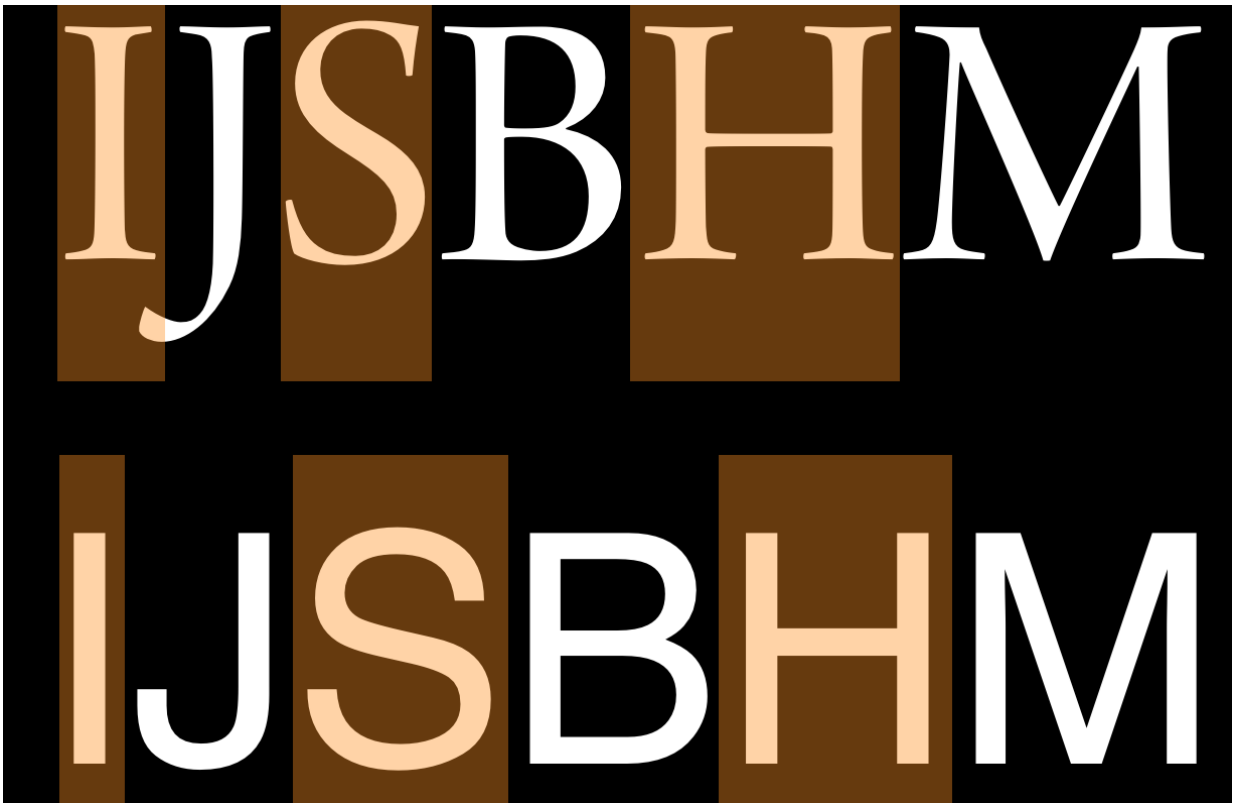
总之，更长的顶部和底部能够变得难以管理。如果字体样式使用了小的行高，典雅意味着字体可能跨越字体的行导致碰撞。

宽度

字体设计的宽度将不仅改变其感觉，也会改变其用途。右边的例子来自一个文本样式。左边的例子来自一个显示设计，目的是要醒目。字母比文本样式例子更加细，能够节约空间或者用于在小的空间内放下更多文本。

宽度不变与可变

上面的一行字母的宽度变化比下面一样要更大。

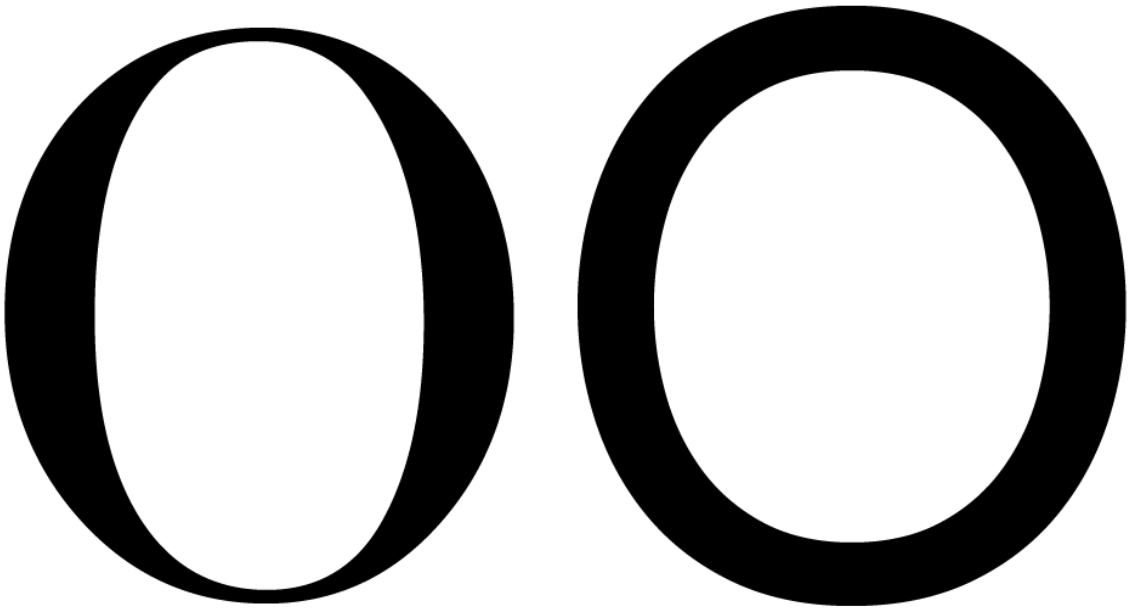


粗细

倾斜

对比

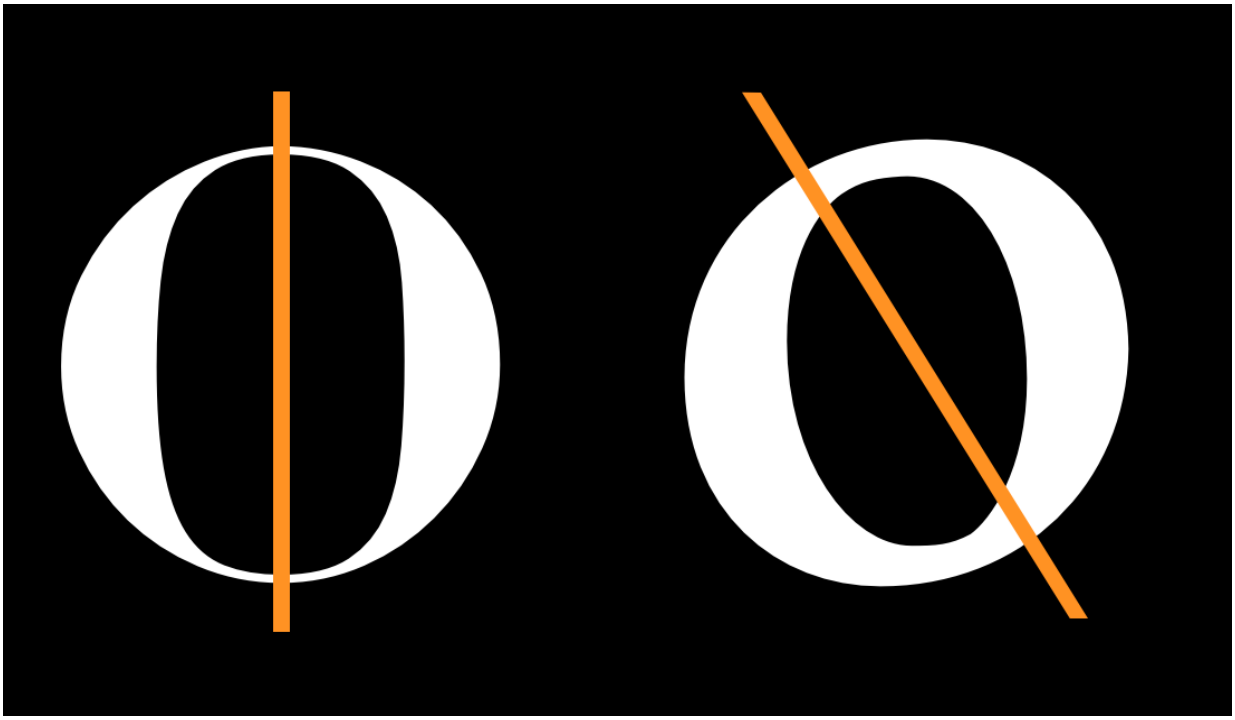
对比指的是一个字形内能够找到多少笔画宽度的变化。注意在下面的两个“O”字形中，左边的从顶部到侧边的线条粗细比右边的变化更大。两个字形都有一些对比，但是左边那个字母的对比比右边的要大得多。



在其字母形式中有一致的粗细（笔画宽度）或者没有可见的对比的字体产生了与对比字体的完全区别。就像衬线与与无衬线之间的选择，对比时字体设计中的一个早期选择。有趣的是，厚衬线设计通常在字母中使用一致的笔画宽度。厚衬线设计不像听起来那样仅仅是关于衬线！需要记住的是感觉应用的规则（参见“相信你的眼睛”） — 对比是关于粗细看起来怎样，而不是测量后应该是多少。

对比的角度

在下面的图中，我们看到小写字母“o”较细部分的形状是不同的。在左边的字形里，细的点完美地落在竖直轴上。在右边的字形中坐标轴是对角线。



粗细分布

如果你很少使用对比，那么你并不需要考虑它。但是大多数字体都包含一定程度的对比。在这些情况下，在你的字体中如何分布粗细将会有更加广泛的选择空间。

竖直

粗细的竖直分布很常见。上面的9和8是尤其强烈的例子。

水平

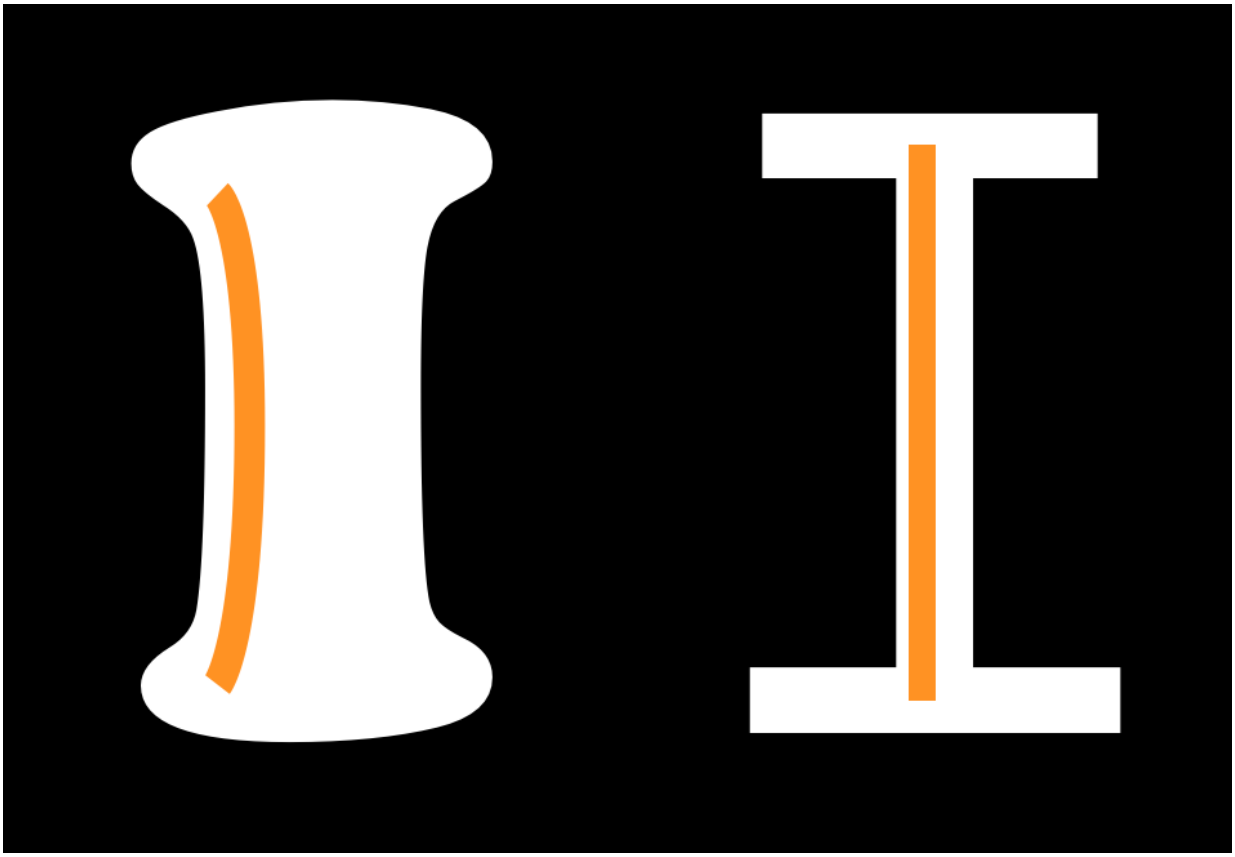
粗细的水平分布很少见，但是仍然能看到很多字体。

底重

顶重

不规则

茎

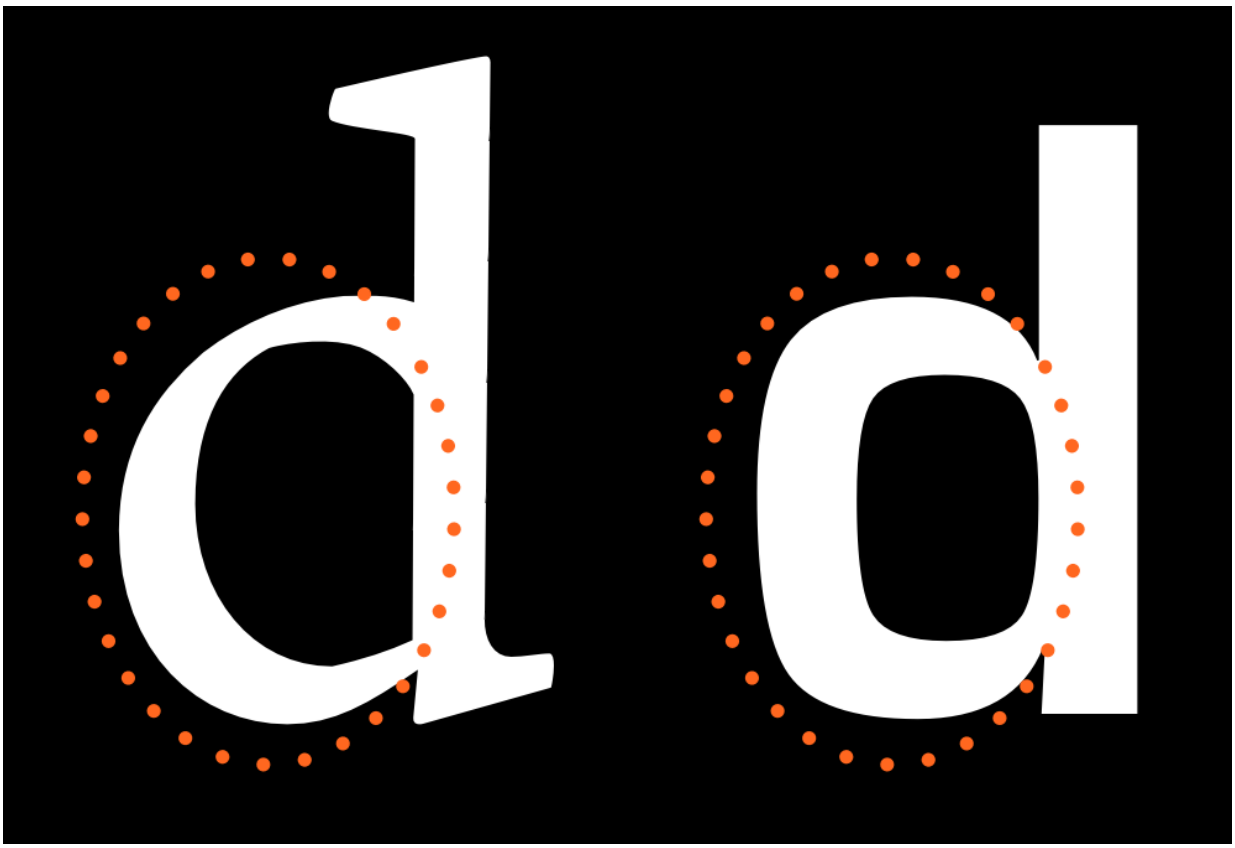


可能你容易假设你的茎会简单地呈笔直，这一点不是真正应该关心的。但是茎的粗细和形状是你应该做出深思熟虑的选择的。

连接

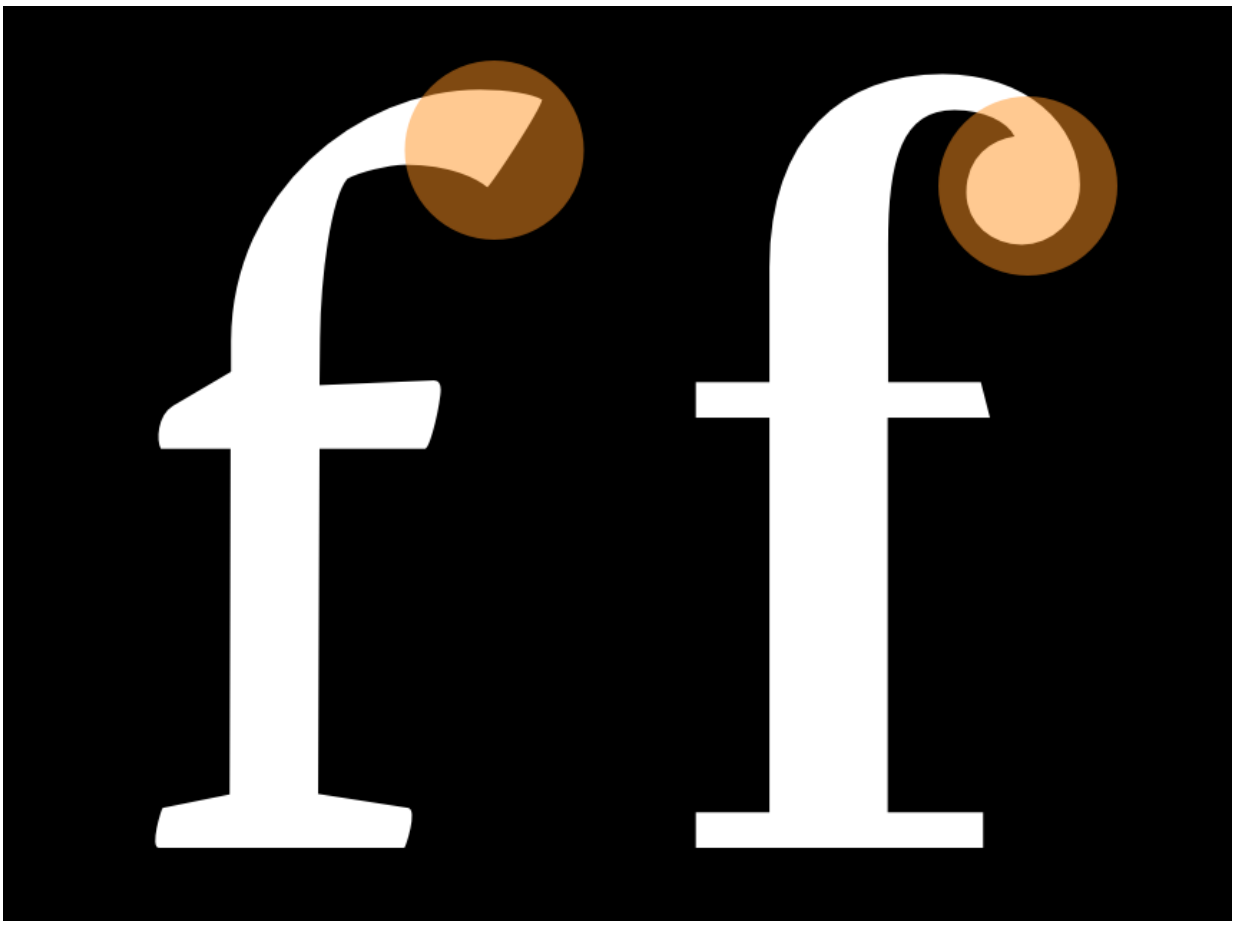
弧

需要注意的是弧（bowl，译者注：指的是在字符中创造出闭合空间的弧线笔画）指的是下面标示出来的笔画的部分，而不是内部的黑色部分。内部绘制的被称为“对立面”。在设计字体时，你可能经常发现需要修改你的作品，不是因为笔画的形状和宽度，而是由于对立面的形状和颜色。



末端

末端指的是末端的形式。它们与衬线是不同的。它们通常末端的角度的垂线，或者水平或竖直地修剪。它们也反映出笔尖的形状或者字母形式的其他创作工具。



速度

左边的“n”看起来比右边的字母写得要快很多。对速度更详细的讨论参见章节斜体。

规则性

下列特征并不会出现在所有的字体设计中，但是它们是可能成为你设计的一部分的可变因素。如果是这样的话，值得考虑它们作为可变因素的程度。

茂盛

需要注意的是上方字体的茂盛更多地体现在大写字母中，而第二个的茂盛更多地体现在小写字母中。

衬线

衬线是一个字体中最显著区别的方面之一。通常字体的一级分类就是衬线和非衬线字体。

这个选择影响了末端看起来怎么样。衬线可以是两端或者一端的。它们可以是垂直绘制的，或者有自己的方向（好像总是水平或者竖直的）。衬线可能有或者没有括号。任何衬线设计时上述的所有特征与特别字母的差异，尤其是“S”，“C”和“Z”,一致地应用在字体设计中（例如一个所有字母都包含水平衬线的字体的s，c和z字母经常有竖直的衬线。）。

一个都市传奇主张认为衬线字体笔比非衬线字体更易读 — 一个独特的迷思，直到另行通知。

衬线的形式与末端的形式相关。

支架

连接着主要笔画的衬线的拐角部分被称作“支架”。一个特别的设计可能采用它们来给衬线加入柔和感（Times New Roman就是一个例子）或者可能选择不加支架。一些设计可能只在一端使用支架或者两端比例不同。

这是一个给字体渲染感觉的比较强烈的参数 — 优雅（Times New Roman的平滑或大的支架）或者粗短干脆（Arvo没有支架）。

厚板衬线

也成为机械或者埃及字体，厚板是厚的块状衬线。厚块衬线不使用支架。通常说来，又这样的衬线的字体设计在字形上会更少使用对比 — Rockwell、Courier或American typewriter体现了这一点。

我们可以可靠地假设厚板衬线使用来在一个其他方面没有对比的字体设计中加入装饰或者节奏。但这也不是绝对的规则。

衬线结束

就像字母结束那样，衬线本身的结束形状对字体的感觉有帮助 — 使其柔软或短粗。衬线结束可以是柔软圆角的（Courier）或者有棱角的（Rockwell）。

装饰

维度

待办事项：正式的元素如何适应文化和文化如何形状和字母间的间距表现出来是在设计学校中没有人会教你的"秘密"，应该在本页应解释 :)

相信你的眼睛

字体设计是迭代地测试每个选择，共同合成一个完整的设计的进程。你将会一直测试字体，看你做出选择的合并结果是否能够：

- 对你来说可读
- 字体让你觉得好
- 字体对于你希望它能够完成的工作有帮助

在你测试设计的时候，你要相信自己的认知，设计实际的东西。许多字体设计都需要你设计的字体类似并且形式上重复。

人们很容易认为如果你测量字形的组成部分和间距，那么你将得到可靠的结果。虽然很有用，但是这个方法有实际的限制。如果有什么看起来错了，那么你应该期望做出调整。更进一步来说，你应该确信正确的做法是作出修改直到它“看起来正确”。

这正确的原因是由一些所有读者都会有的自然的视觉错觉。这些错觉必须在你修改字母形状的时候计算在内，直到他们看起来正确。

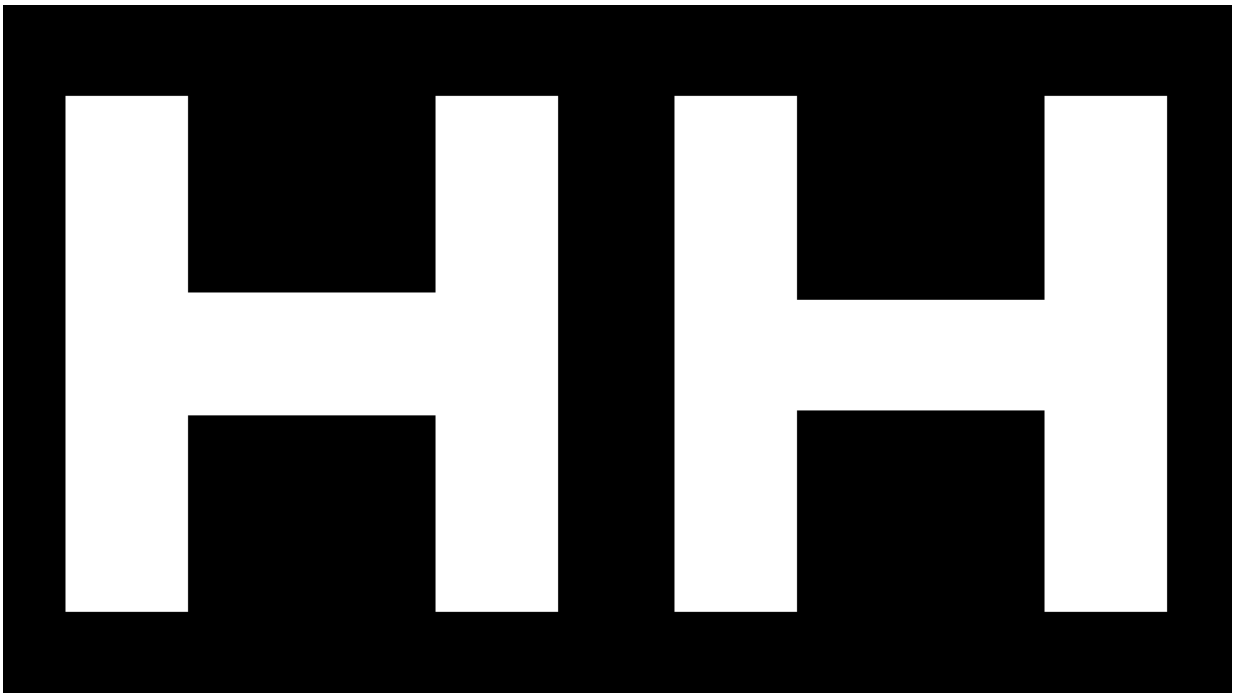
通过这些视频 （外部链接）你可以大致的了解一般都是通过修改或者注意哪里来修正这些视觉错觉。

错觉的例子

一些错觉涉及对线条的粗细的感知，一些涉及线条的长度，另一些设计形状的感知。

水平与竖直的粗细程度

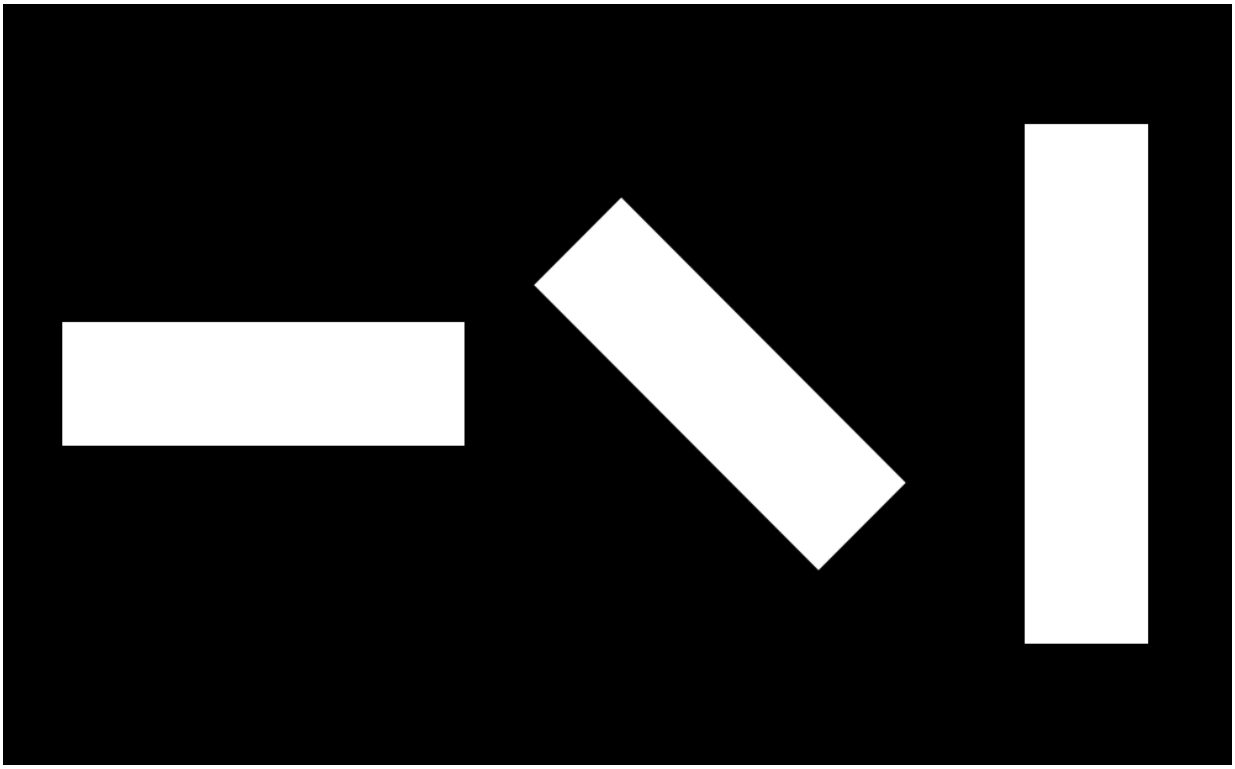
左边的例子展示了一个条形粗细相同的“H”。这看起来是不对的。你能感受到吗？右边的另一个水平的条形做的更细，这样看起来粗细相等。



需要按照视觉做出调整的字形有很多，包括A、E、F、L、H、f、t和z。

斜向粗细

类似地，如果一些条形宽度相等，其中一个放在斜线上，那么斜向的条形看起来比竖直的条形粗并且比水平的条形细。如果你想让它看起来正确，那么你不得不像水平的例子一样将其调细一点，但是只是细一点点。

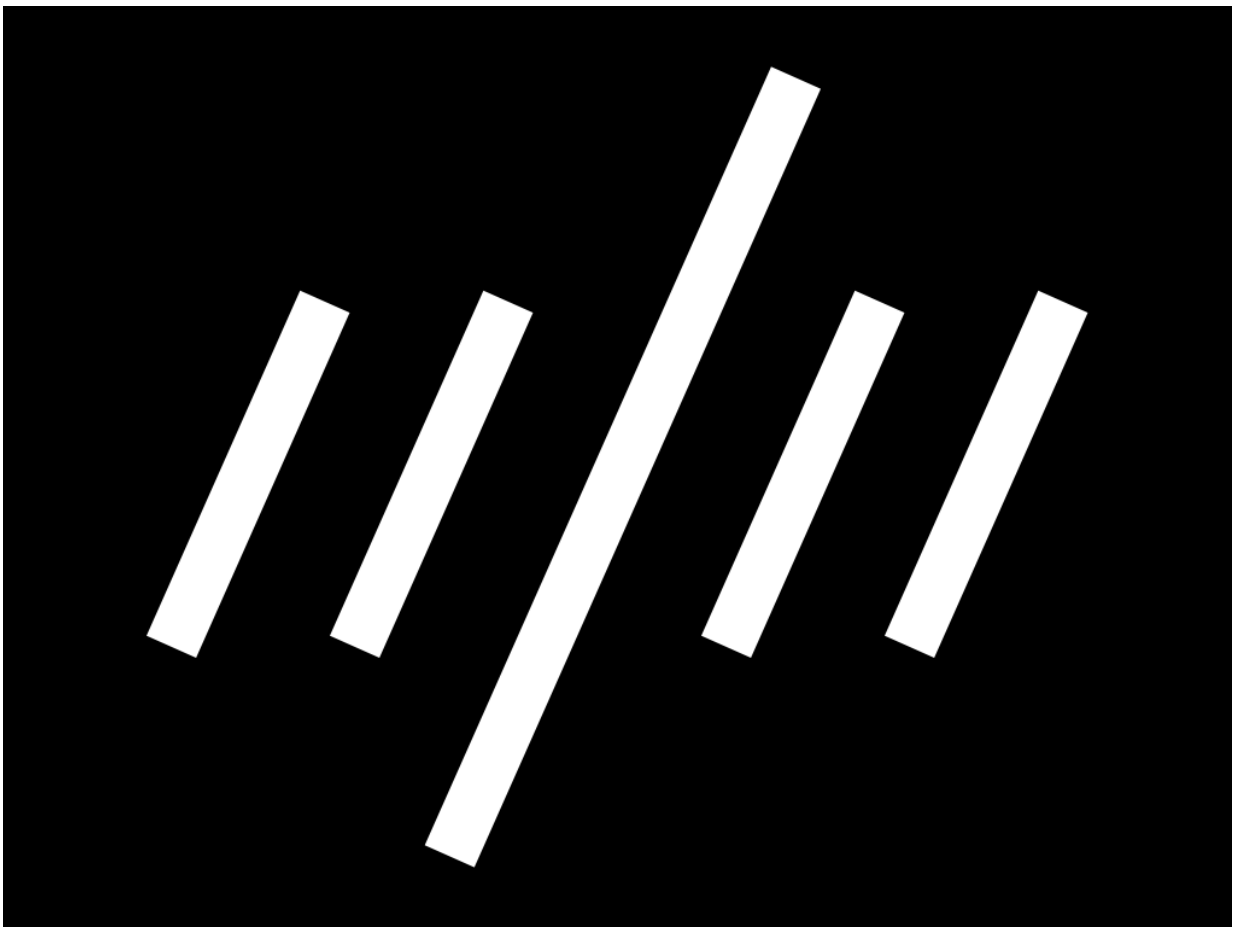


令人产生错觉的相关字形十分多，包括k、K、N、Q、R, v、V、w、W、x、X、y、Y、7、2、&、ł、Ł、ø、Ø、√、〈、‹、›、 «、»、½、⅓、¼、≤、≥和×。

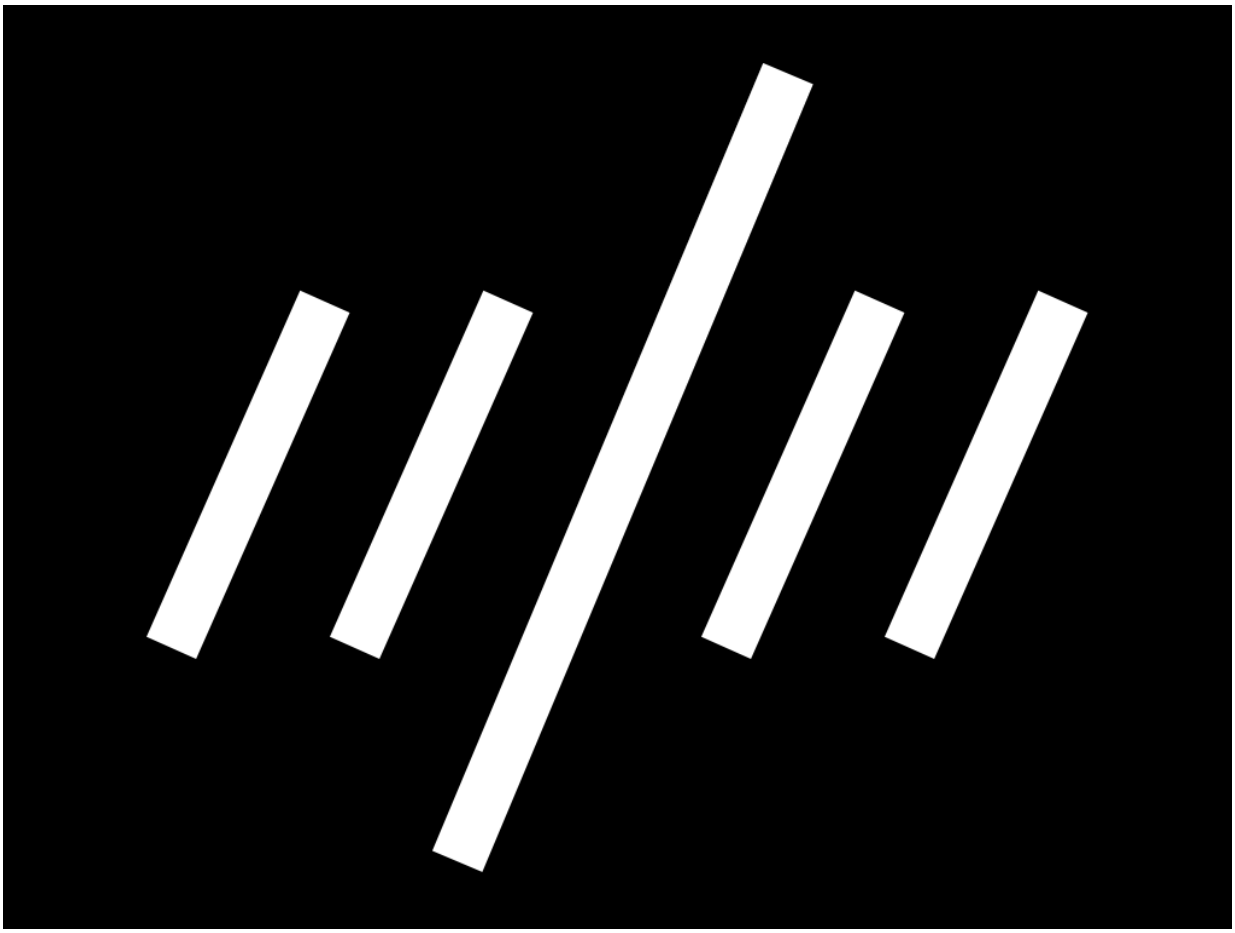
宽度和对斜向角度的感知

更长的图形需要比短的图形倾斜更少，这样表面上看起来他们倾斜度相同。

下面的图中的斜线倾斜角度相同。长的那个看起来角度不同。



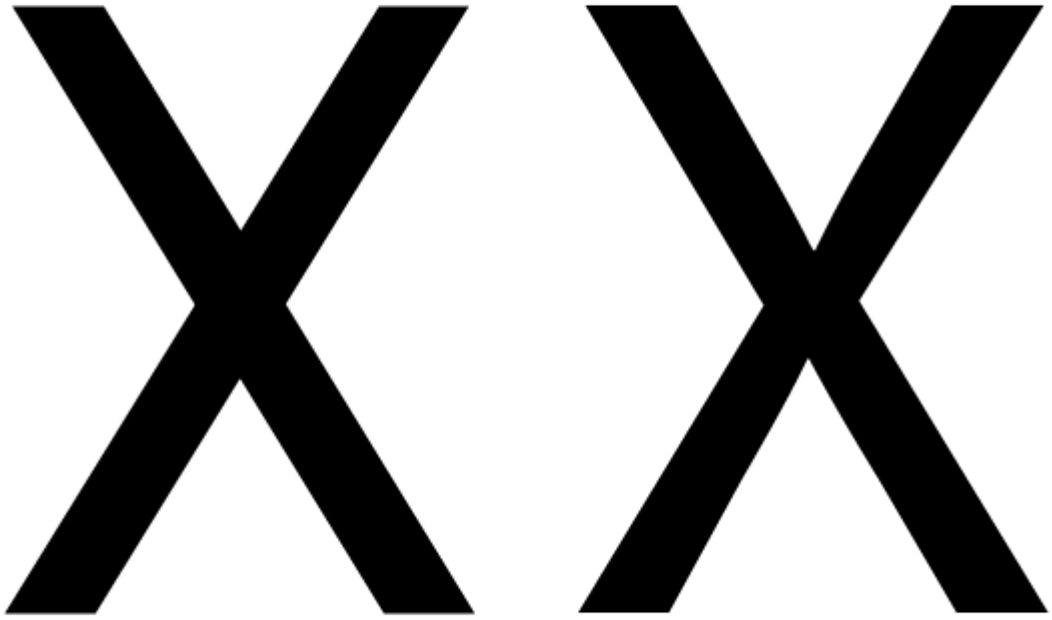
在下图中，长的线条的倾斜程度被调整过：



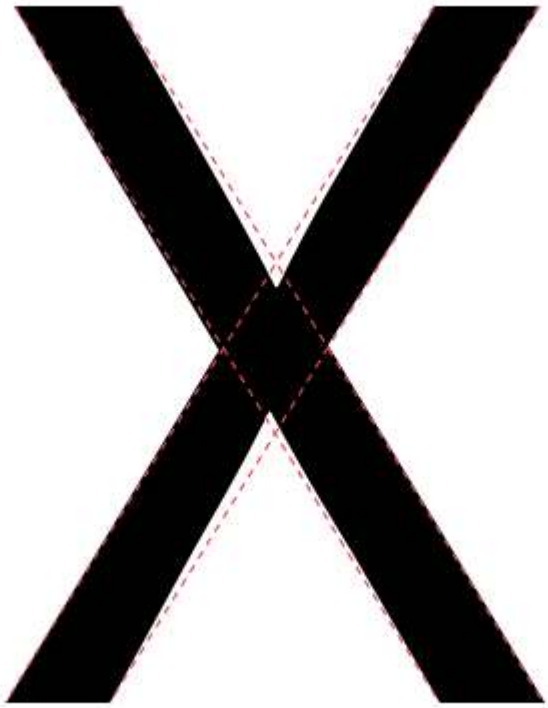
现在我们来看真实的斜体，在字形上应用了这些修正：

交叉斜线

当一个条形穿过另一个斜线或者直线时，需要调整它以便看起来不像是没有对齐。



上面的例子中，左边的X由两个未调整的条形互相交叉。右边的例子被调整过，这样看起来是对齐的。

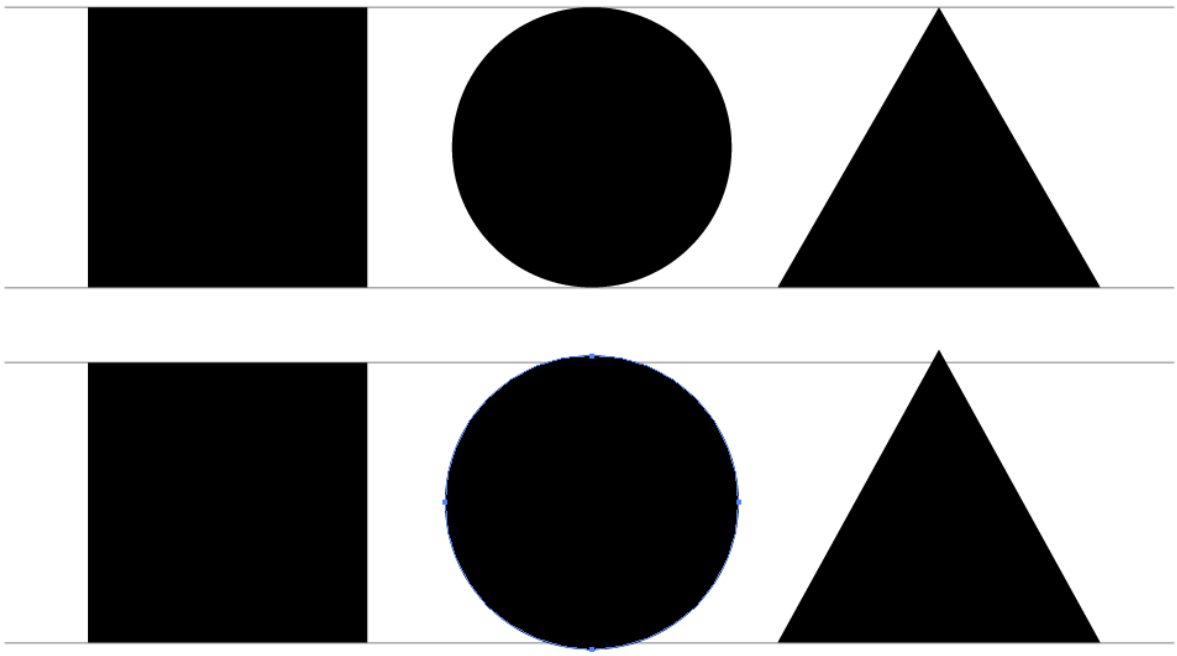


在这个X中通过虚线你可以看到看起来显得对齐的X实际上有偏移。

涉及到这个错觉的字形包括x、X、k、K、×、#和冰岛语字母“eth”（ð）。

高度感知

一个字形的形状将会对它看起来和其他字形高度一致时的高度产生作用。圆的字形需要超过平的字形的高度一点点。包含尖的形状的字形需要超过更多。形状越尖锐，它为了看起来正确需要超过的就更多。



在上图中，上面的三个图形没有经过调整，也就是他们有完全相同的高度。下面的三个图形经过调整，看起来有更加接近的高度。

这一错觉涉及到包含圆的或尖的部分的人和字形，包括O、Q、C、S、A、V、W等。

你完全有资格来纠正这些错觉

由于你既能看到错觉，也能看到修正错觉的效果，所以你能够自行作出这些修正。你仅仅需要相信你的印象。

测试目的的适当性

就像你能够看到光学错觉并纠正他们一样，你也有能力去分辨一个字体是否胜任你心目中特定的用途。在这一点上你也同样需要相信你的判断。

值得一提的事没有字体能够在离开其使用方式和目的的情况下被评估。这也是测试需要从设计进程最开始一直到你觉得项目完成的原因。

这些测试会是什么样子呢？开始的测试比较简单，你只需测试第一个设计的选择。随着你的设计变得更加完整，你的测试需要跟上进度并且能够评估你所做的最新的决定相对成功还是失败 — 或者更好的情况是能够比较你所考虑的2个（或者3个或者更多…）选项。

有时你将发现你不得不折返修改你之前认为已经足够好的设计选择。这是常态。制作一个字体需要平衡很多因素，惊奇经常会发生。设计的字体越多，你做任意选择的经验也就越多。

当接近进程的尾声的时候，如果字体采用简单的方式来使用，那么测试也应该保持简单。但是如果一个字体将会有多种使用方式，或者在大范围的打印或屏幕环境下使用，那么测试应该覆盖这些情况的所有范围，包括打印字体的各种样例。

如果你对你打算的最终用途有明确定义的想法，那么设计时间将得到节省。但是这并不总是可行的，你的想法可能发展。关键的东西是考虑并定义尽你所能完整的用例，然后确保在设计字体时你的测试跟上你问自己的问题。

计划好你的项目

现在你能感觉到一个字体如何设计是多样的。如果你的项目会是几个内部关联的字体的集合，或者是一个三四种样式（现在已经是约定）的字体家族，或者是更庞大的东西，那么你可能想选择你的项目是否只有一个字体。

字体家族的通常样式包括：

- 一个常规体和一个粗体
- 常规体，粗体，斜体乃至粗斜体
- 细体，瘦体，书体，常规体，半粗体，粗体，超粗体，重体和黑体
- 常规体，紧缩体，粗体和紧缩粗体
- 窄体，紧缩体，宽体和超宽体
- 常规体，半茂盛体，茂盛体，很茂盛体，极茂盛体

典型的模式在字体家族中存在是有原因的，你也可能发现一个非常不同的分组。

项目的范围能够通过你的雄心和自由时间总数来唯一地确定。但是项目范围经常通过字体的集合或家族的用途来确定，或者更进一步说，通过你的客户的需要来确定。当然对于专业字体设计者来说，下面的两个问题通常是确定的因素。

字形覆盖率

即使一个字体内只有一个字形，它仍然是一个字体。但是一个字体也可以有几百甚至几千个字形。如果你的项目是自己发起的，那么这个选择最终时随意的。你可能选择你只需要大写字母，或者你希望包含你使用的其他字体。如果你为客户工作，那么你可能希望阐明字体需要支持的语言。你的目标也可能是扩展现有字体，加入一些字形使得它增加一个或多个语言下发挥作用。

慎重地作出选择，包含更少而不是更多来减少错误当然是好主意。通常一个字体制作后，可能尝试去包含越来越多的字形，但是更多时候相比于添加一些新的，继续提高字形的核心集合更有价值。

多样式字体家族工作流程

如果你从一开始就知道将会制作不止一个字体，系统地计划和构建字体家族，某种程度上并行地开发样式而不是一次完成一个样式，那么你将节省自己的时间。

当然不可能采用完全并行地方式创造*每个*样式，但是有可能完成每个样式的一个特定的设计步骤，达到在进程早期检查确保样式间关系的目的。你可能发现完成一个测试字母的完整集合（例如“adhesion”）的常规体，接下来制作这些字母的其他样式是有帮助的。但是你也可以让进程粒度更小，为所有样式的基本字母的特定部分（例如“n”he“o”）共同作出选择。

取决于你计划的字体家族的尺寸和组成，你可能会发现制作字形的可插入实例会节约时间，这样做不仅可以插入中间样式，也可以对在字体家族成员间存在的排版变化所作出的设计选择有所补救。你应该考虑的变量在“什么是字体”一章。

技术: 版本控制

你应该学习使用Git和Github来保存你的文件，使用“SFDir”格式来存放你的源。

- https://help.github.com/articles/what-are-other-good-resources-for-learning-git-and-github

- http://justinhileman.info/article/git-pretty/

EM Square

— 也被称作“EM size”或者“UPM”。在一个字体中，每个字符都放置在其空间容器内。在传统的金属字模中，这个容器就是每个字符的实际金属块。每个字符的高度是统一的，这样每个字模可以整齐地放进行和块中（如下）。

字模的高度被称为“em”，起源于大写的字符“M”的宽度；这个字母的比例被做成了方形（因此有了“EM Square”的称呼）。em size是根据字模计算出的点值。因此一个10磅的字体em也是10磅（如下）。

在数字化字体中，em是空间的数字化定义总量。在OpenType字体中，UPM或em大小通常是1000单位。在TrueType字体中，UPM约定是2的幂，通常是1024或2048。

当时用字体来设置样式时，em将会缩放到需要的点值。这意味着对于10磅的字体样式，1000单位在这个实例中将会缩小到10磅。

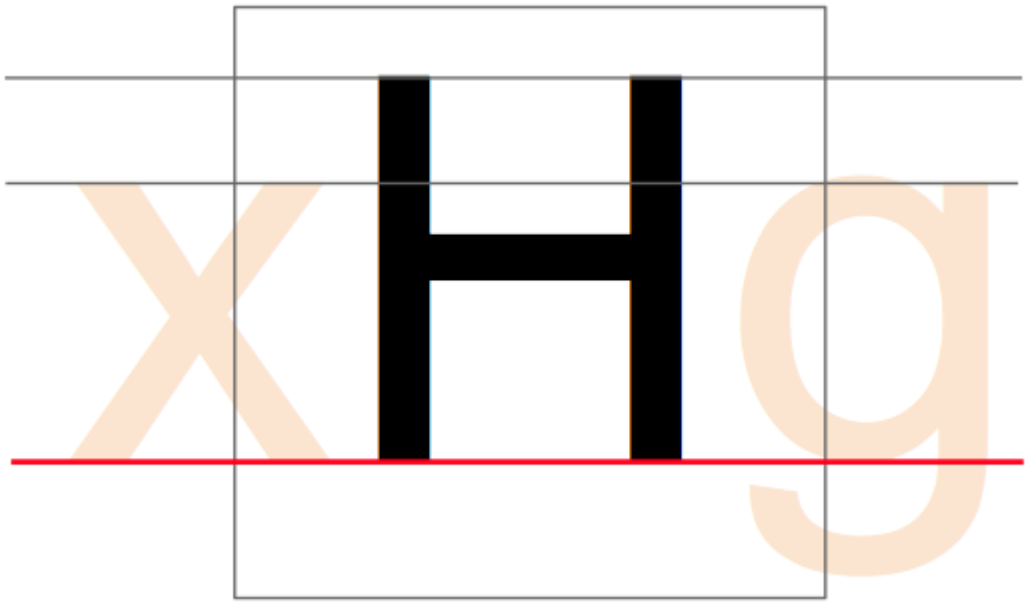
因此如果你的大写的“H”时700单位高，那么它在一个10磅的字体中将会被缩放到7磅高。

在字形窗口中设定

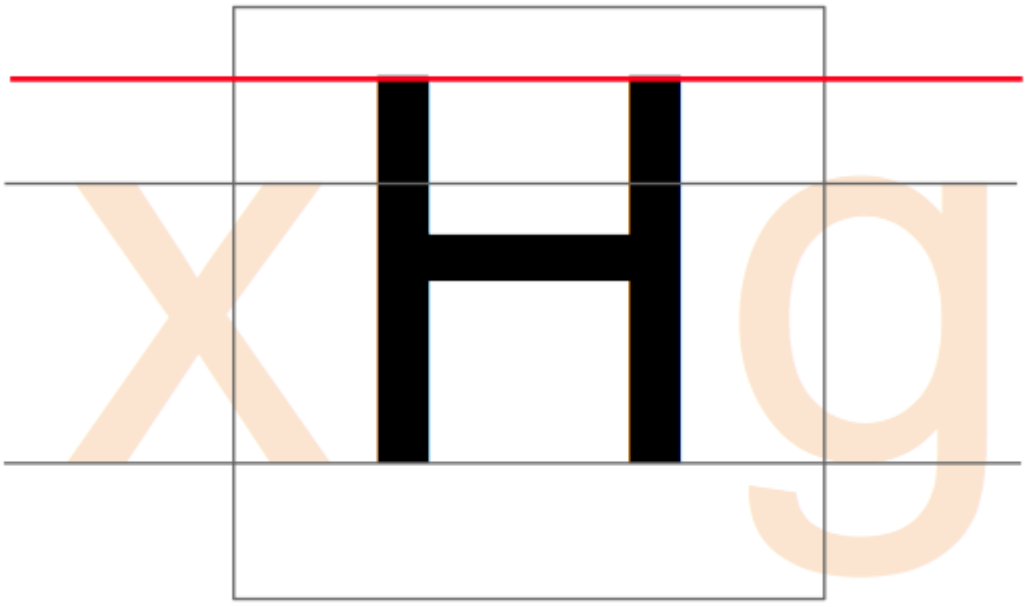
知道了你的字体将会使用1000，1024或2048UPM后，你需要设定你的字形的绘制，以保证你字体样式的所有面都充分地填入UPM空间内。

Em square的大小设定方法是*Element* > *Font Info…*然后点击General选项卡，你将会看到*EM*设置。这个值将会分配到*顶高*和*底深*上，分别在基线的上下。

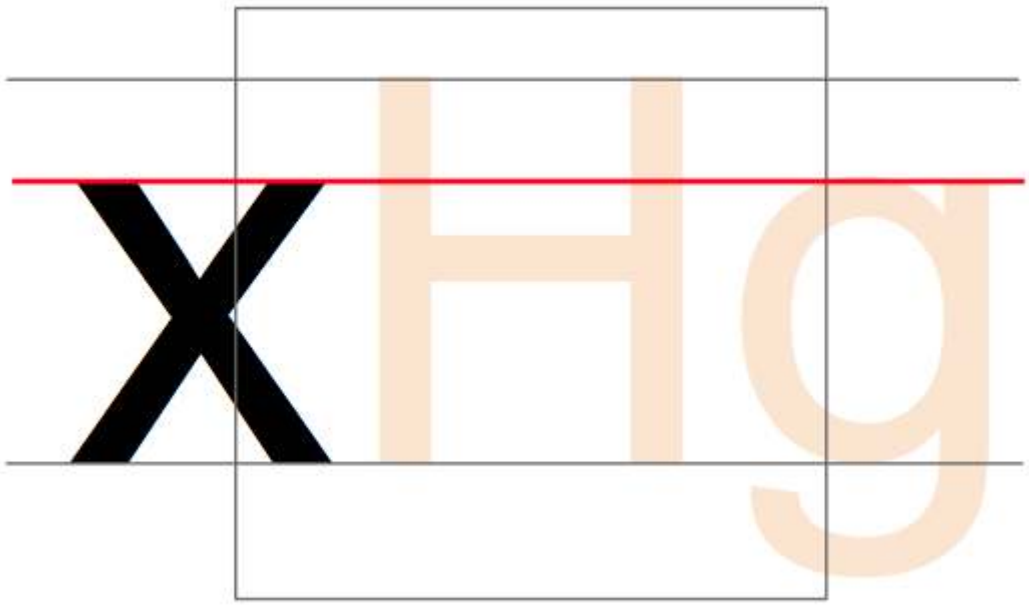
基线：



大写高度:



x高度:



在你设计字体的后期，你需要设置Blue值，这个值为PostScript画轮廓而保留，也保留给FontForge自动微调 — 无论你在做哪个轮廓。

你可以在*Element > Font Info…*，*PS Private*中找到这个设定。FontForge可以首先根据你的轮廓猜测初始值，但是为了上突/下突，你必须自行修改 — 这个概念在几章后的创造“o”和“n”中；让我们先上手FontForge及其绘制功能。

安装FontForge

FontForge是一个自由开源软件，这显然意味着你可以不受限地下载和安装。这也意味着这是一个由社区维护的应用（任何人可以贡献源代码）。

FontForge支持Windows，Mac OS和GNU/Linux (“Linux”)操作系统。本节的关注在Linux机器上安装FontForge。由于许多FontForge的开发者使用Linux作为日常开发环境，所以在这个平台上用源代码构建时最简单的方式。

注意：如果你使用FontForge时遇到了问题，或者FontForge缺少一个功能，你可以在软件的库打开一个问题。即使你只是刚刚入门，开发者也会去看这个问题。

安装预编译包

在FontForge网站的主菜单上点击下载按钮将会带你到FontForge下载页。页面内列出了三个操作系统下的安装链接。链接页面内都提供了二进制包下载。（译者注：安装预编译包一节按照FontForge网站最新结构编写，原文与网站已经脱节）

在Windows下安装

FontForge的Windows版本页面提供了二进制安装包下载，以管理员身份安装，并以管理员身份运行软件即可。

另外，Jeremy Tan提供了Windows下FontForge的最近构建版本。从2012年之前的稳定版安装包可以在旧的SourceForge库中找到。

在Mac OS X下安装

正在建设中的新网站上提供了安装指南。

在GNU/Linux下安装

在你的Linux机器上安装FontForge最简单的的方式是使用你的分发版的包库。

Debian或Ubuntu

FontForge包从2012年开始就默认包含在Ubuntu 14.04中，因此通过FontForge [Personal Package Archive (PPA)](https://launchpad.net/~fontforge/+archive/ubuntu/fontforge)可以安装最新的包。

检查辅助脚本 `add-apt-repository` 已经安装：

```
sudo apt-get install software-properties-common
```

添加FontForge PPA（同时添加认证密钥）：

```
sudo add-apt-repository ppa:fontforge/fontforge
```

升级软件列表，使得PPA包含包：

```
sudo apt-get update
```

安装FontForge：

```
sudo apt-get install fontforge
```

Fedora

以root用户身份运行下面的yum命令可以在你的Fedora Linux桌面机上安装FontForge。完成安装的下载量大概是10MiB。

```
yum install fontforge
```

如果在你的Fedora机器上没有编译软件，那么安装gcc，automake，autoconf和其他软件后你可能在执行libtoolize的autogen.sh的时候遇到错误。如果遇到这种情况你需要安装Fedora的libtool-ltdl-devel包，或者其他Linux分发版的类似开发包。

上述yum install完成后你可以在你的菜单运行FontForge，或者从konsole或gnome-terminal直接使用*fontforge*命令运行。

在Github上编译你自己的版本

某些情况下，可能你需要使用一个预编译版本中尚不存在的功能，你可能希望从Github拿到代码编译自己的版本。Github是一个源代码托管服务，每个人都可以为软件一部分的开发做出贡献。本节的说明只针对Ubuntu 14.04。

安装准备软件

安装一些包以准备软件的编译

```
sudo apt-get install build-essential automake flex bison
```

安装*unifont*包使引用字形完整显示。[Unifont] (http://savannah.gnu.org/projects/unifont)包含所有Unicode编码的字形，如果安装了FontForge将使用它。

```
sudo apt-get install unifont
```

安装其他的必需包：

```
sudo apt-get install packaging-dev pkg-config python-dev libpango1.0-dev
libglib2.0-dev libxml2-dev giflib-dbg libjpeg-dev libtiff-dev uthash-dev
```

构建*libspiro*

FontForge使用[libspiro] (http://github.com/fontforge/libspiro)来简化曲线绘制。

下载代码：

```
git clone https://github.com/fontforge/libspiro.git
```

按顺序执行下面的命令（也就是说等一个执行完再执行下一个）：

```
cd libspiro
autoreconf -i
automake --foreign -Wall
./configure
make
sudo make install
cd ..
```

构建*libuninameslist*

FontForge使用[libuninameslist] (http://github.com/fontforge/libuninameslist)来访问每个Unicode编码点的属性数据。

下载代码：

```
git clone https://github.com/fontforge/libuninameslist.git
```

按顺序执行下面的命令（也就是说等一个执行完再执行下一个）：

```
cd libuninameslist
autoreconf -i
automake --foreign
./configure
make
sudo make install
cd ..
```

构建FontForge

下载代码：

```
git clone https://github.com/fontforge/fontforge.git
```

按顺序执行下面的命令：

```
cd fontforge
./bootstrap
./configure
make
sudo make install
cd ..
```

让系统知道新的库：

```
sudo ldconfig
```

如果你需要单步调试TrueType字体提示或者其他高级功能，还需要使用 `--with-freetype-source` 配置选项。

调试FontForge软件

如果在某个阶段你发现FontForge的可复现的稳定性问题，你可能需要安装调试信息，才能给FontForge团队提供回溯信息以纠正问题。

如果你从Linux分发版的包库安装了FontForge，安装调试信息的方法与从源代码构建时安装调试信息不同。在两种情况下，你都可以使用*nm*命令来检查你安装的FontForge中调试信息是否已经可以使用。使用“type”命令来找到你的 `Fontforge二进制文件` 。如果你看到下面显示的“no symbols”信息，你需要升级你的安装包来包含调试信息，能够给FontForge开发者提供良好的反馈。

```
$ type -all fontforge
fontforge is /usr/bin/fontforge
$ nm /usr/bin/fontforge
nm: /usr/bin/fontforge: no symbols
```

当你想要为来自Fedora库的FontForge添加调试信息的时候，使用下面的命令。需要注意的是如果你还没装好许多依赖的调试信息包，那么可能需要下载几百兆字节的数据。

```
debuginfo-install fontforge
```

更多信息参见调试一节。

使用FontForge绘制工具

在FontForge中设计字体的时候将会用到一些工具和实用工具，我们开始先使用一些能让用户对矢量图形感到熟悉的绘制工具 — 这方面的经验有显著不同。

我们在看FontForge的绘制工具前首先来理解Bézier曲线如何工作。

理解Bézier曲线

Bézier曲线的概念指的是一个特别的数学上的表示，用来数字化地产生平滑的曲线。通常使用*二次方*和*三次方*Bézier曲线 — FontForge也支持*Spiro*曲线，设计者的另一种可以替换的表现。

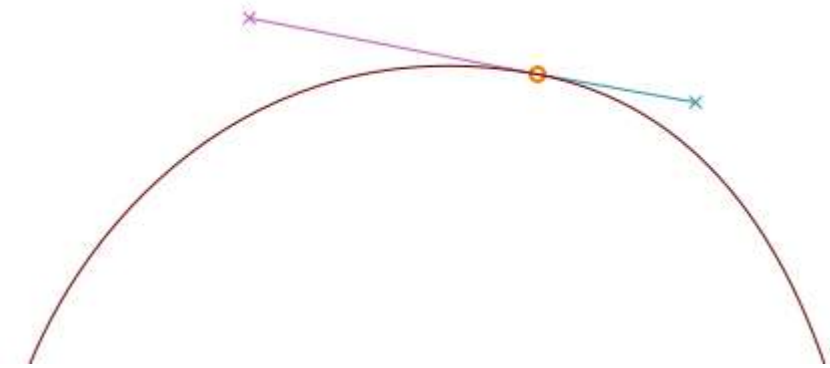
在本章中，我们只讨论*三阶*路径，它在绘制字形时普遍使用。*Spiro*路径将在下一章讨论，*二阶*曲线在绘制时很少用到，只会在TrueType字体中找到 — 他们更常出现在构建时.

一个典型的Bézier曲线由一个锚点和标示全部方向的两个手柄组成 — 每个手柄的长度决定了一端的曲线的长度 — 如下。

不同类型的点

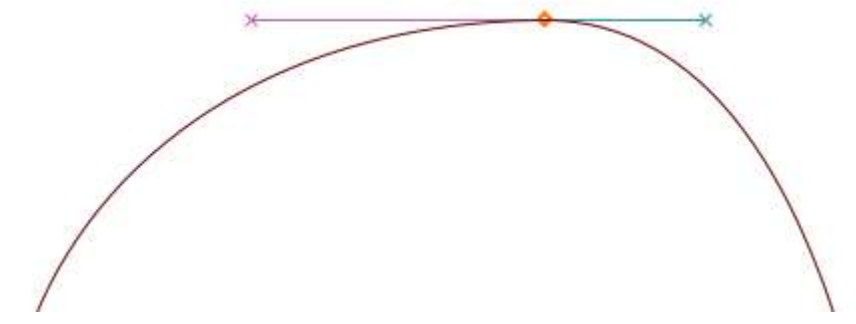
曲线点（显示为圆形的点）

*曲线点*有两个手柄，每个都连接到另一个，因此他们之间的线是直的，目的是在每一端都产生平滑的曲线。



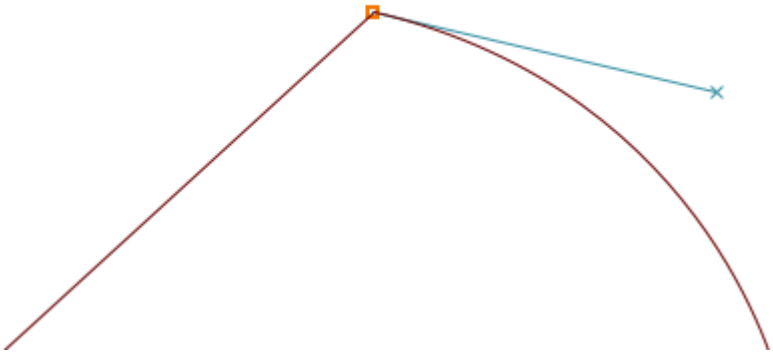
H/V曲线点（显示为菱形的点）

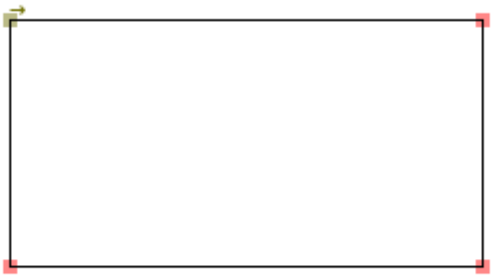
H/V曲线点（horizontal/vertical）是对齐到水平或竖直轴曲线点的变体 — 一个使Bézier曲线形态正确的必要工具（其他同样作用的工具见下节）。



拐角点（显示为正方形）

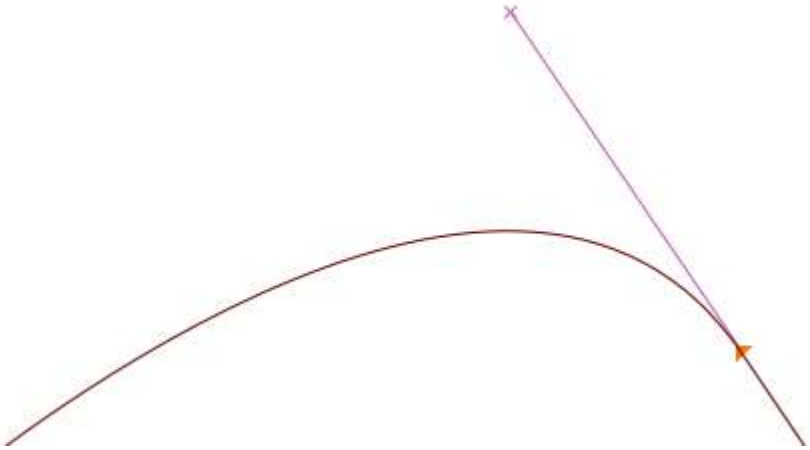
*拐角点*可以有0、1或2个Bézier手柄。每个手柄的位置相对其他独立，适合用来构造不连续的轮廓。没有手柄时，拐角点产生直线。





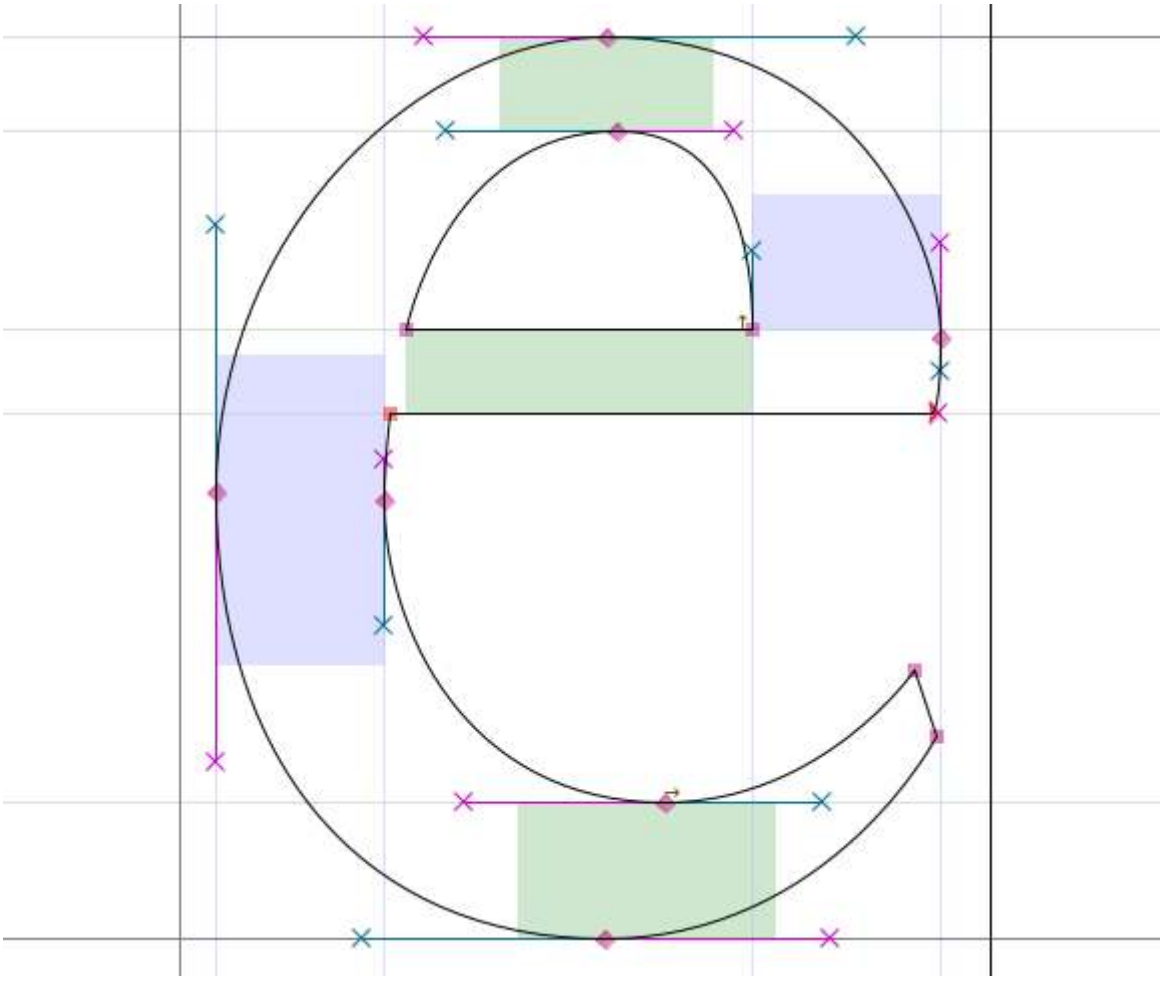
切点（显示为三角形或箭头）

如果你希望从一条直线开始然后是平滑的曲线，那么你需要使用切点。 一个*切点*在一端留下直线，另一端的Bézier手柄是其方向 — 这保证了线条和曲线间连续过渡。

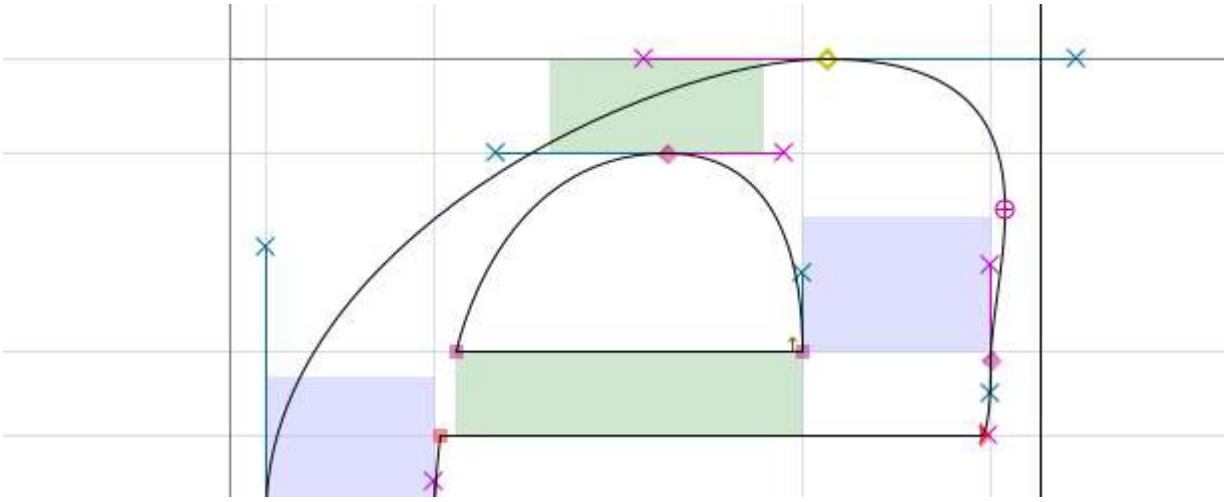


使其正确

为了产生合适的曲线 — 使用最少控制点并减轻栅格化，锚点应该始终放置在**曲线极值**处，并且除非你的字母中有中断，否则确定路径的线应该是**水平或者竖直**的。



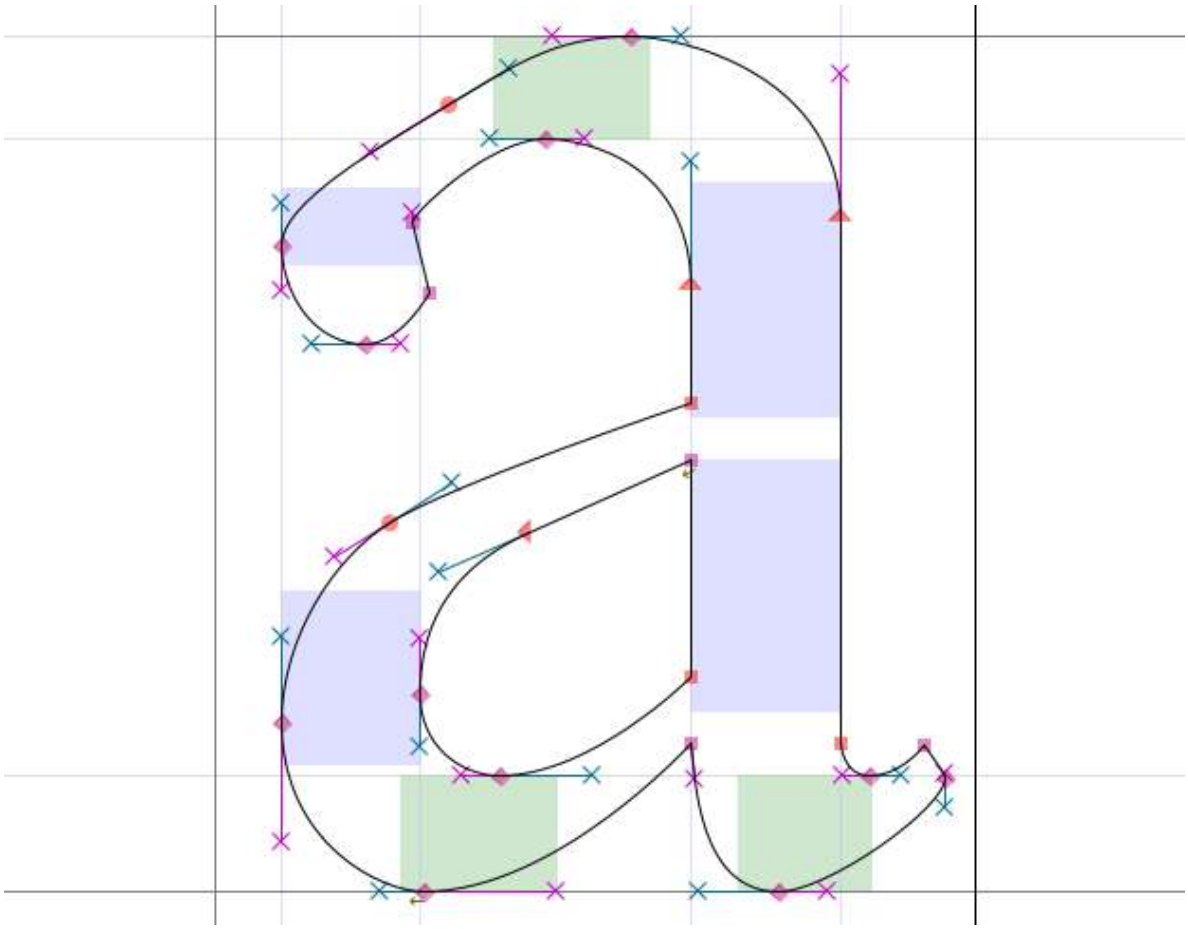
注意：如果你的控制点没有放置在极值出，FontForge将会用一个瞄准图标指出实际的极值处：



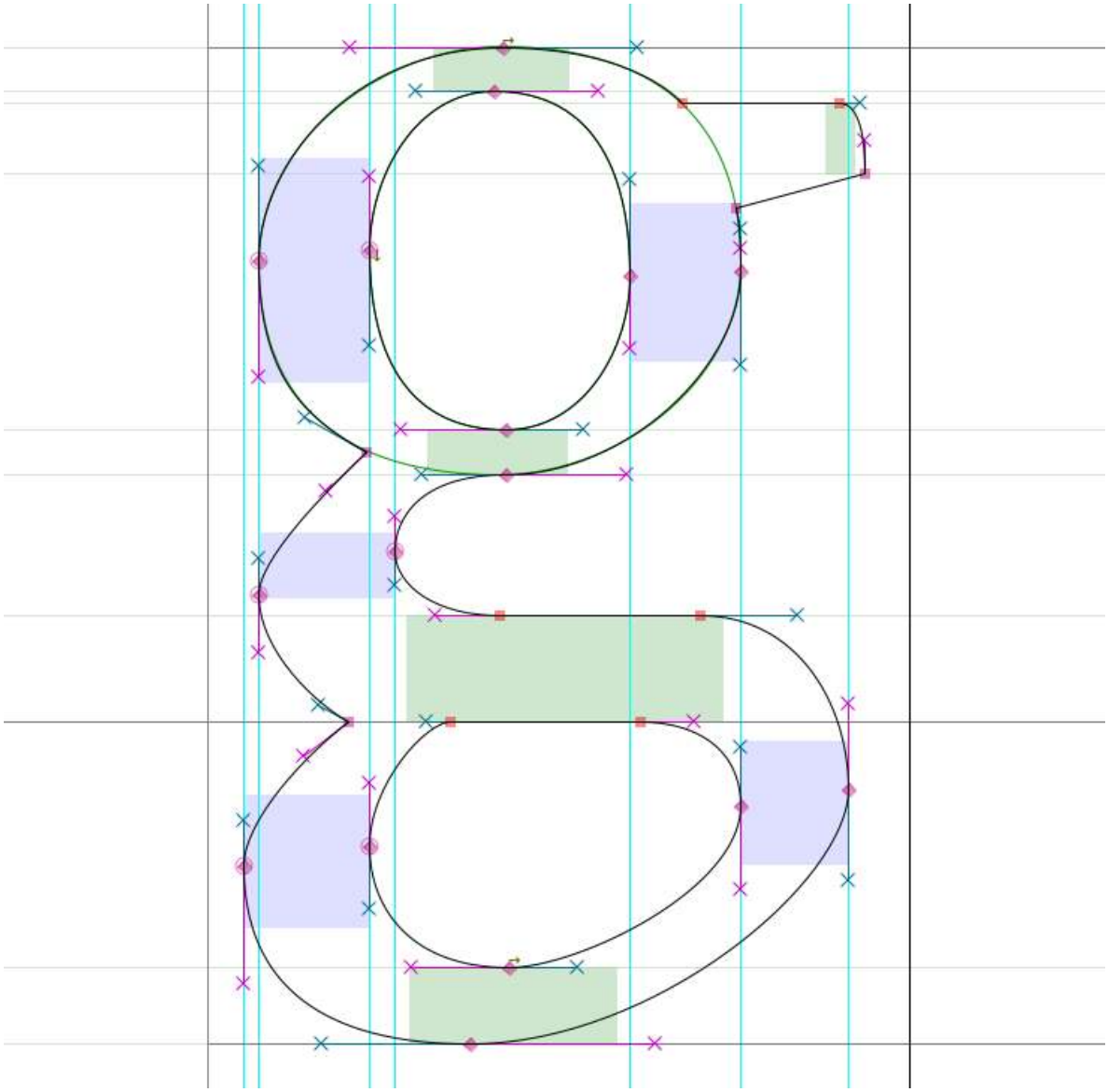
那么你这样修正：复制你当前的轮廓到另一个图层，移动周围的控制点使其排列正确 — 否则FontForge验证工具将会自动在极值处添加点，在该点处你可以通过*右击 > Merge*来合并
你放错的锚点。 更多相关信息在稍后的 确保你的字体有效，验证一章。

详细说来，有两种情况你需要放弃水平/竖直的Bézier路径：

- 如果你希望改变你的曲线使其整体倾斜，正如下面的“a”的左上部保持者平坦：



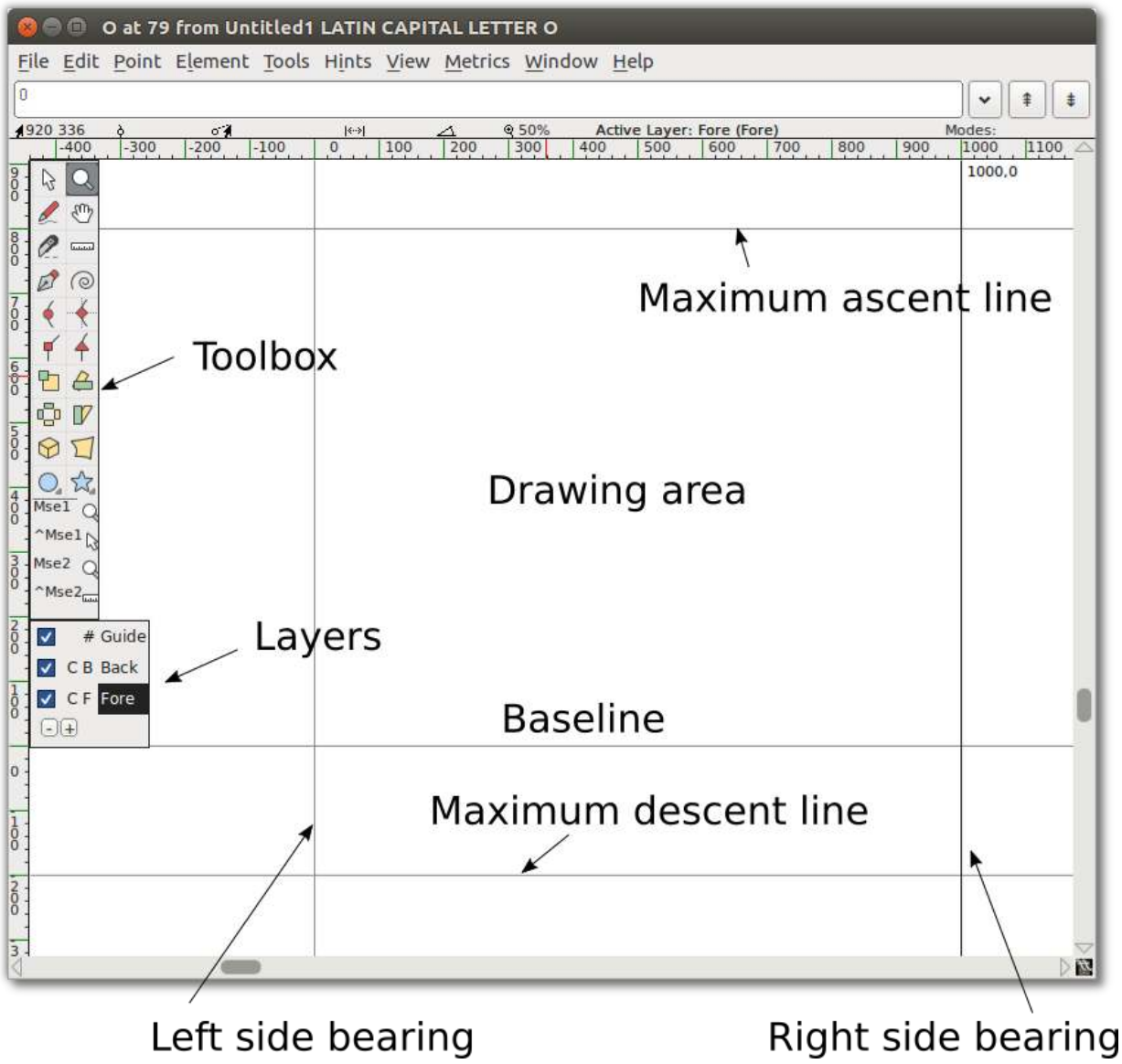
- 如果你希望在字母形式中放置中断，正如下面的“g”的左下部 — 这是你希望使用拐角的典型（除了画线以外）：



注意：正如你所看到的那样，当你用一个*拐角*设置中断的时候，每个手柄的方向应该是曲线延伸方向的切向。

掌握FontForge的绘制工具

在主窗口中，双击其中一个字形的格子来打开字形窗口。

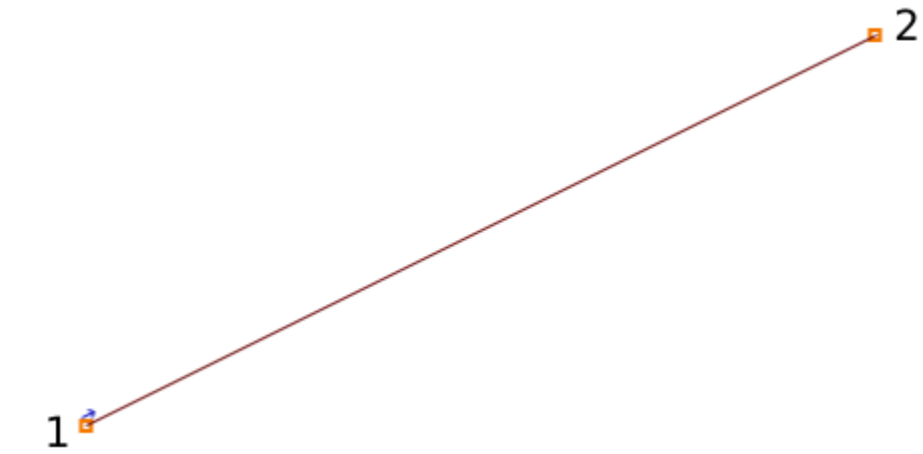


注意：x轴和y轴交叉的地方上面的数字从左到右分别指示：

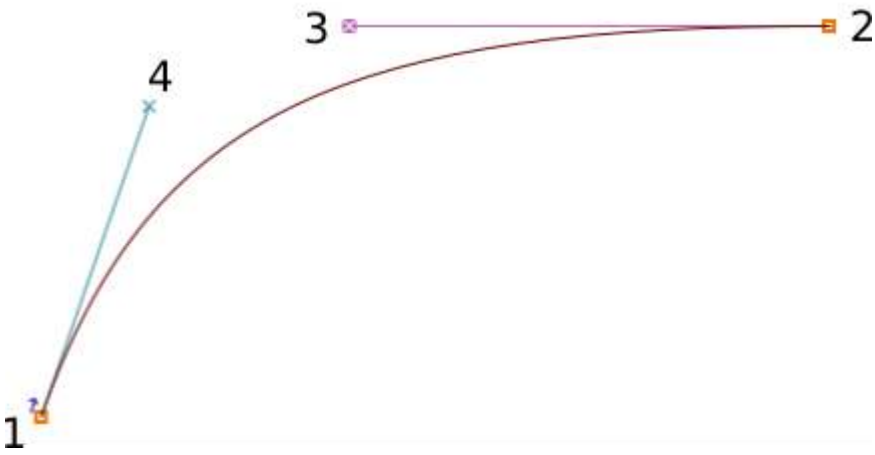
- 你的鼠标指针在当前画布的(x,y)位置
- 最近选择的点的位置
- 你的鼠标指针与选择的点的相对位置
- 你的鼠标指针和选择的点之间的距离
- 选择的点和鼠标指针之间的角度（相对于基线）
- 当前放大级别，活动图层的名称紧随其后

当心：有时当你在字形窗口内时，FontForge看起来无响应。这可能是由于一个打开的对话框隐藏在其后 — 所以只需要移动它并处理对话框。

2个点组成的直线。



一个*样条曲线*包含了4个点：2个结束点和2个描述样条曲线在结束点的斜率的手柄。



样条曲线和直线的复制、粘贴、剪切和删除

就像大多数绘制软件一样，FontForge允许你复制、剪切、粘贴和删除任何点、直线和样条曲线。这些命令可以在Edit菜单找到，或者使用你的操作系统的快捷键（也展示在了菜单中每个命令的旁边）。

熟悉绘制工具

现在你已经知道了画布周边功能用法，是时候熟悉画布工具了。

指针和缩放

指针和缩放工具的用法和其他应用的的等价工具类似。 指针是一个选择工具，用来选择画布上的点、路径和 其他对象。 缩放工具让你（在Z轴上）更方便地缩放；想要缩小：到View菜单下选择*Zoom out*（X）或者*Fit*。

需要注意的是，在你使用其他工具的时候，可以通过按住`Ctrl`键来暂时切换到指针工具。

自由绘制

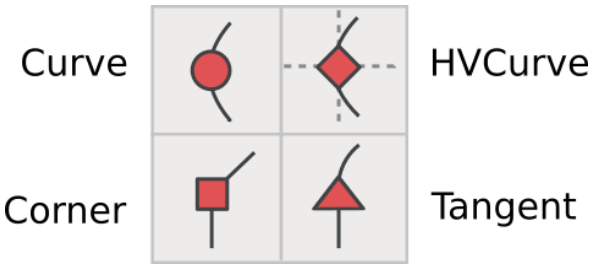
自由绘制让你可以画出不规则的路径。

在绘制区域，鼠标按住拖动来绘制。切换回指针工具，你可以选择你刚才绘制的路径上的点。

当你选择路径上的一个点的时候，它将会变成一个黄色的圈。如果选择的点在曲线上，那么他将会显示出带有一个洋红色手柄和一个青色手柄的控制点。你可以拖动他们来改变曲线的形状。

指针工具

那么我们开始学习指针工具。



为了在路径上添加一个点，我们首先选择这些工具中的一个，然后在路径上点击并轻推一下。你就在线上添加了一个新的点。

曲线点工具用来在曲线段上添加一个点。 HVCurve点工具约束新点为水平或者竖直的控制点 — 这对设置极值点来说是重要的。 拐角点工具用来在路径上制作尖锐的转弯。 切点工具用来实现直线段到曲线段的沿着路径的过渡。

钢笔工具

钢笔工具用来在曲线上添加点并拖出其控制点。

Spiro

选择Spiro工具会进入Spiro绘制模式。Spiro绘制可以在你调整节点位置的时候重排你的曲线。有的人相对于标准方法（称为Bézier编辑），更偏好这个方法，但是如果你习惯了Bézier编辑，你可能发现这种方法会做没有料到的事情。

小刀

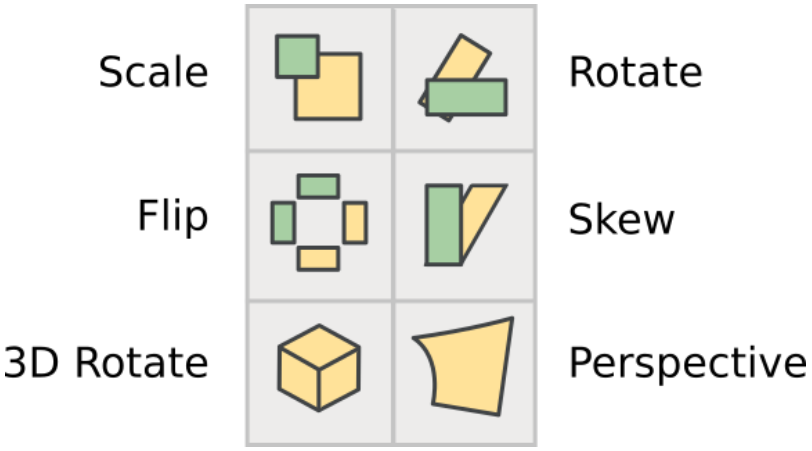
小刀工具让你可以将样条曲线且为两段。如果你希望绘制一个图形但是只需要一部分，那么这个工具是合用的。

尺子

尺子工具提供测量和坐标信息给你。当你使用时，在鼠标指针旁边显示一个浮动的提示框。如果你的鼠标指针悬停在一个点上，提示框提示框会展示更详细的测量和坐标信息。如果你在样条曲线上使用，它将告诉你曲率和半径。最有用的是，如果你点击拖动尺子工具，你将看到你拖动鼠标指针的距离，以及穿过图形的每个交点的信息。

变形工具

变形工具有6个：



注意：对于每个变形工具，如果你双击工具，可以输入数值。

缩放工具可以让你自由手动缩放一个对象。按住Shift键可以在缩放的同时保持比例。

旋转工具可以自由地旋转一个物体。始终围绕你最初点击的位置来旋转对象。

3D旋转工具可以在第三维度旋转对象，并将结果投影在x-y平面上

翻转工具可以水平或者竖直地翻转选中的对象。旋转的原点是最初点击鼠标的点。

注意：翻转一个点后你很可能需要应用Element > *Correct Direction*。

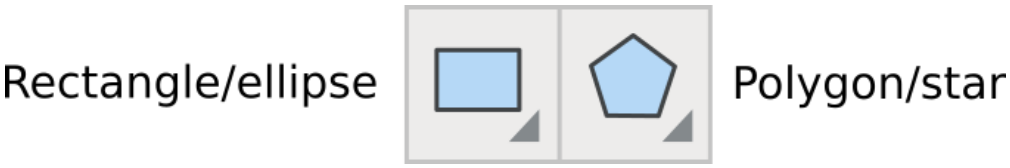
扭曲工具可以将选中的对象顺时针或者逆时针做水平扭曲。

透视工具让你能够以非线性的方式扭曲图形。

注意： 透视转换并没有数值选项。

矩形/椭圆和多边形/星形工具

这些工具让你可以绘制简单的几何形状，这样比使用分开的线段构建形状更快。



点击工具区域可以切换到可选的另一个工具。如果你双击这个工具，会打开形状样式的选项。

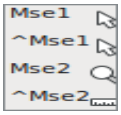
矩形选项：拐角样式和扩展方式（拐角还是中心扩展）。

椭圆选项：边框扩展还是中心扩展。

多边形选项：顶点数量。

星形选项：星形的顶点数量和点的深度百分比。百分比设置越高，星形的角越长。

Mse1和Mse2



在工具栏中你可以看到当前的工具和鼠标按键可以使用的操作：

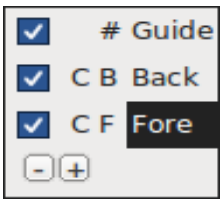
- 左键（Mse1）
- 左键 + `Ctrl`（^Mse1）
- 滚轮（Mse2）
- 滚轮 + `Ctrl`（^Mse2）

这样你不必反复点击工具栏就可以使用一些不同的工具。

当心：Mse功能当前看上去不能正确工作。

图层

FontForge的画布默认有3个图层：引导图层，背景图层和前景图层。引导图层用来插入引导（比如x高度或者大写高度引导）。前景图层和背景图层都是用来绘制的，但是只有最顶层的引导图层将会呈现在最终的字体上。



眼睛图标指示每个图标是否可见，并且你可以点击眼睛图标来使图层不可见。C（或者Q）你是否在使用三阶或二阶曲线。

、B或者F代表每个图层是否是引导图层、背景图层或者前景图层，在你添加了多个图层后更有意义。你可以使用工具栏的加（+）或减（–）来创建和删除额外的图层。图层类型和曲线类型也可以通过右击来控制（如果你有额外的图层时）。

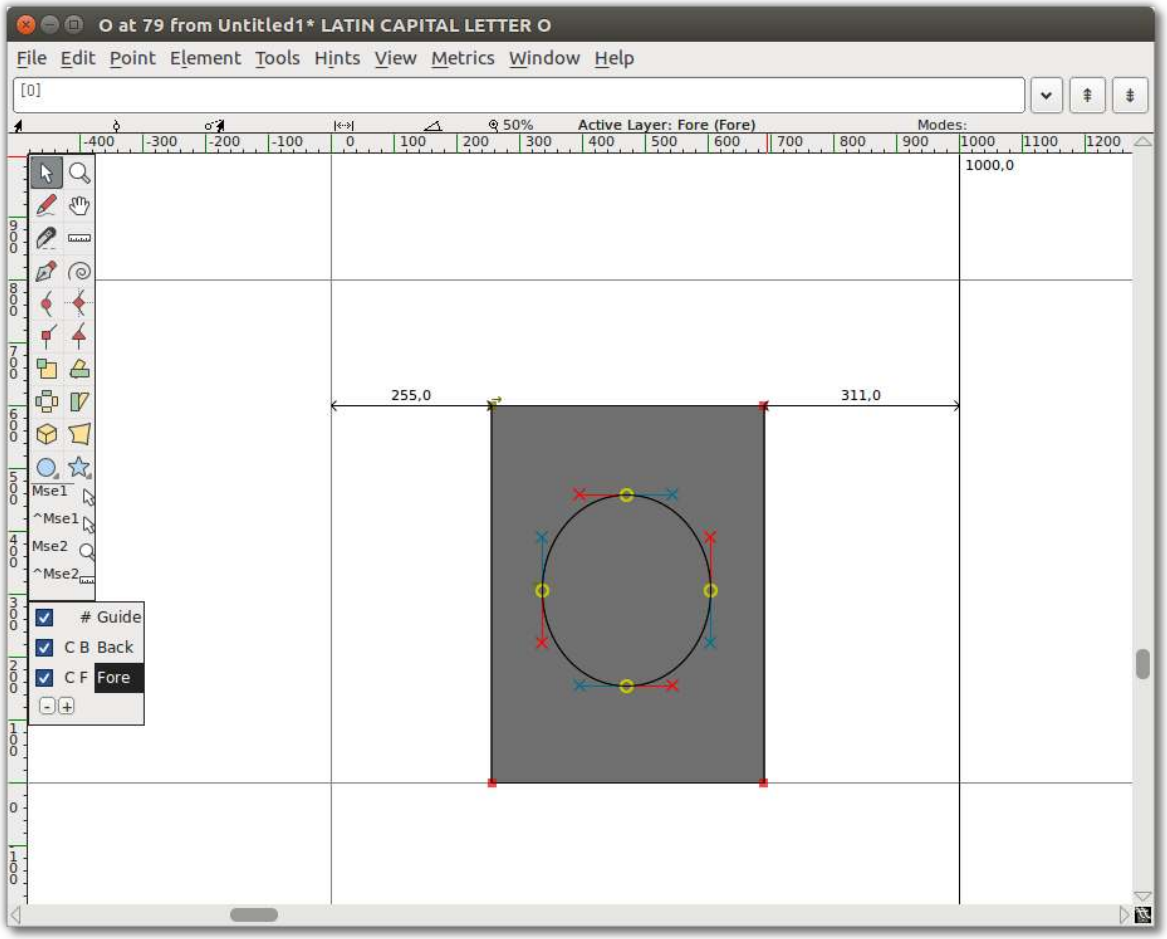
绘图基础

接下来我们重温基本绘制的工作流程。你经常会发现你需要它。

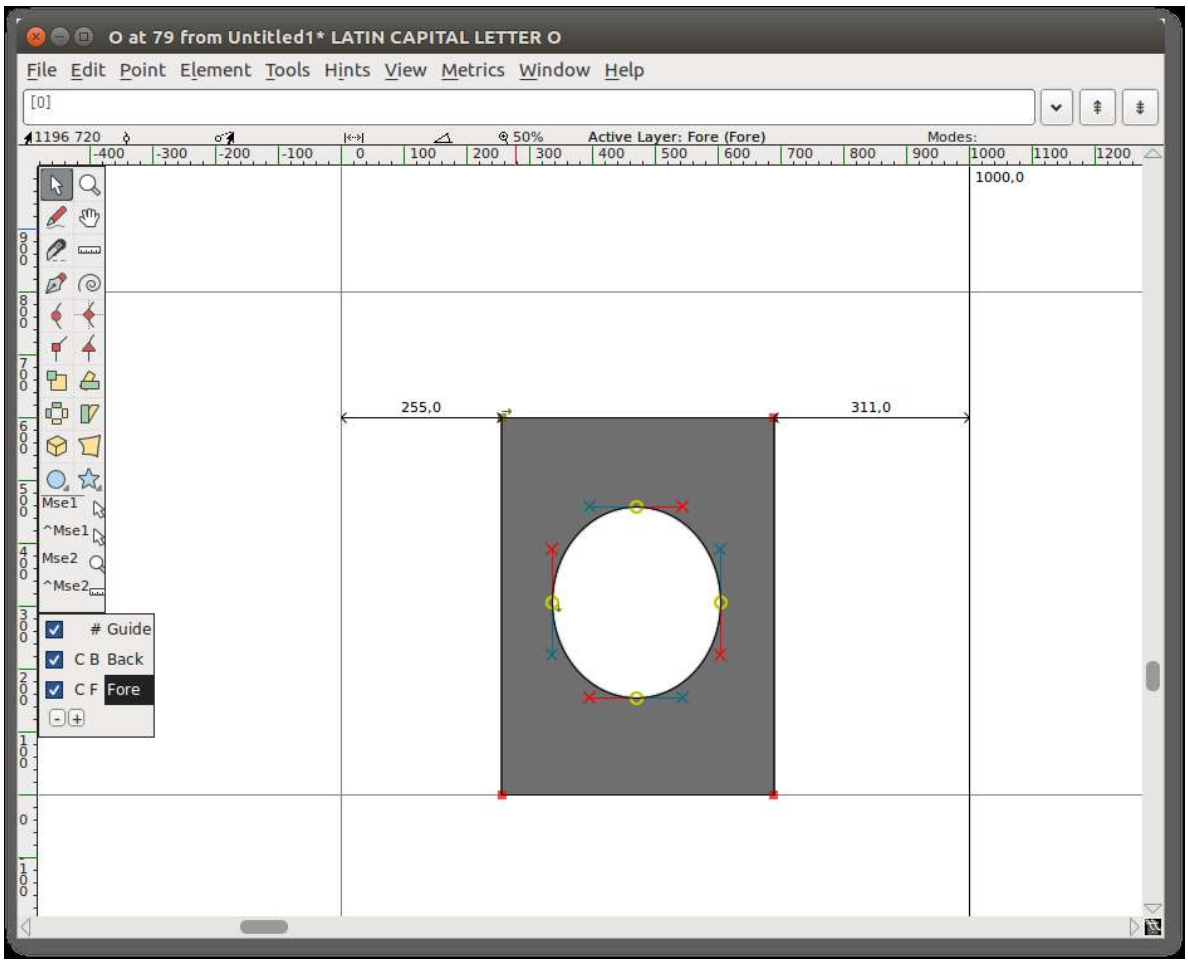
在一个图形内切掉另一个图形

1. 首先在字形窗口的绘制区域用矩形工具绘制一个矩形。

2. 接下来使用椭圆工具在你刚刚绘制的矩形内绘制一个椭圆。

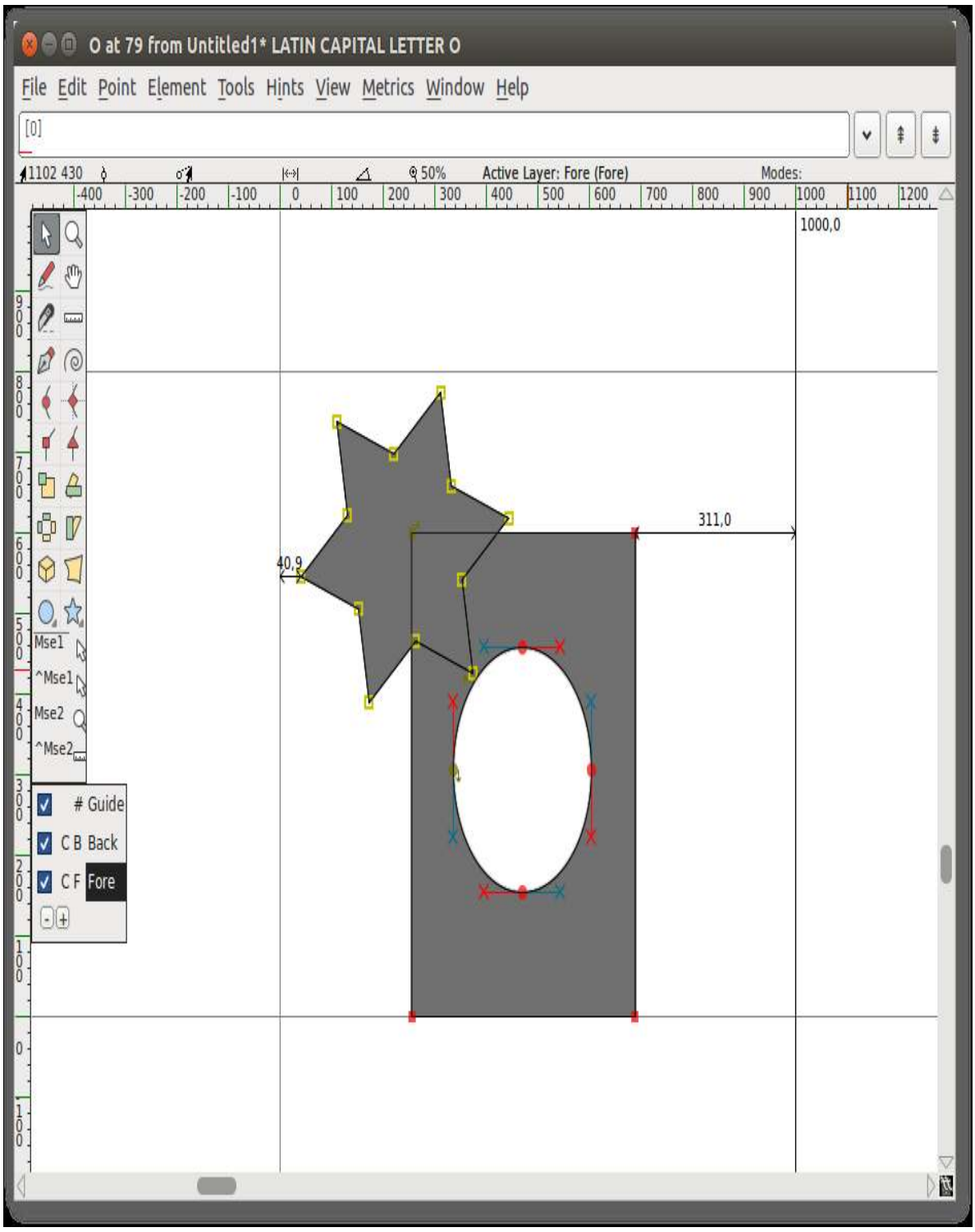


3. 在Element菜单选择*Correct Direction*。你将会看到两个图形合并了，在矩形的中央穿出一个洞来。



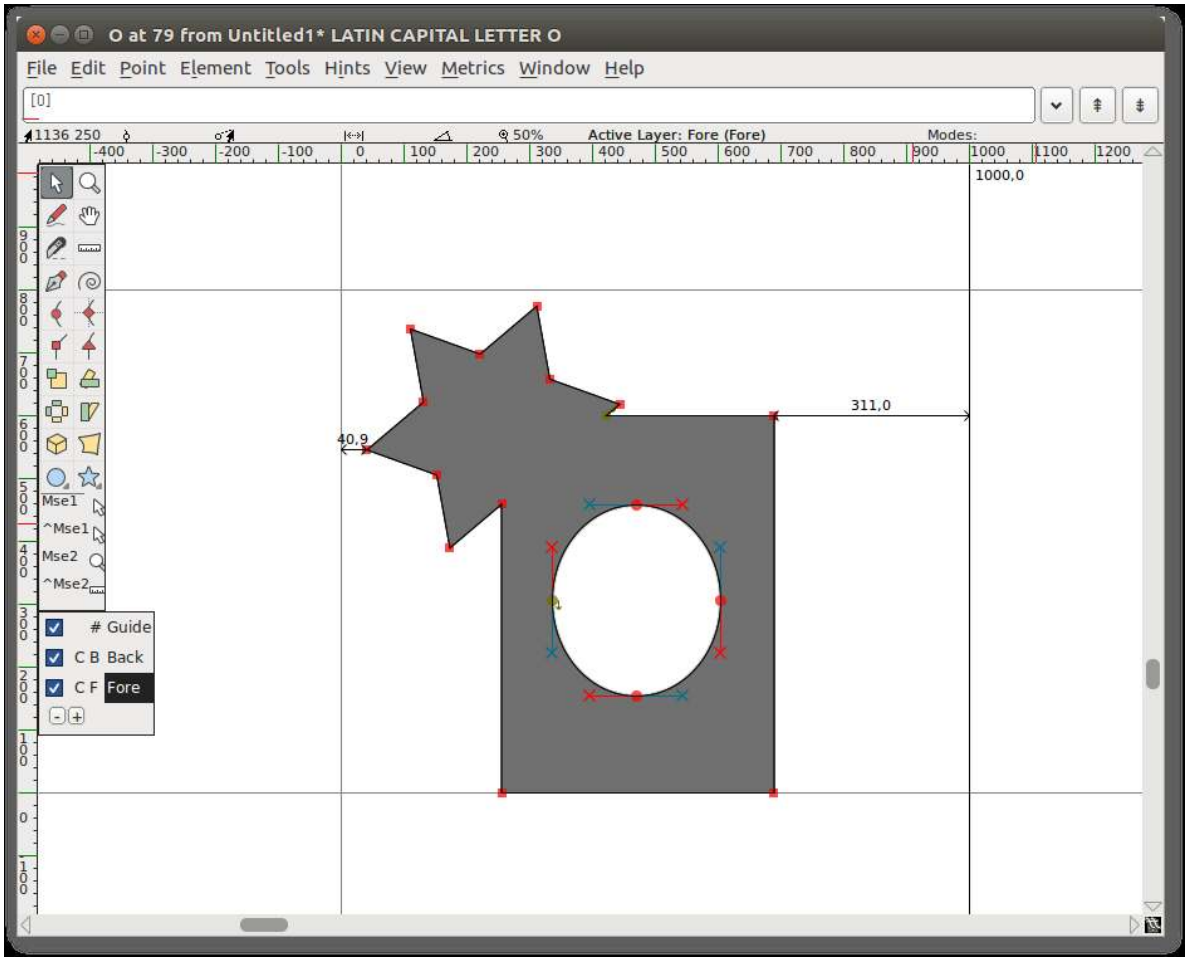
移除重叠区域

1. 添加一个星形，与矩形的角重叠。



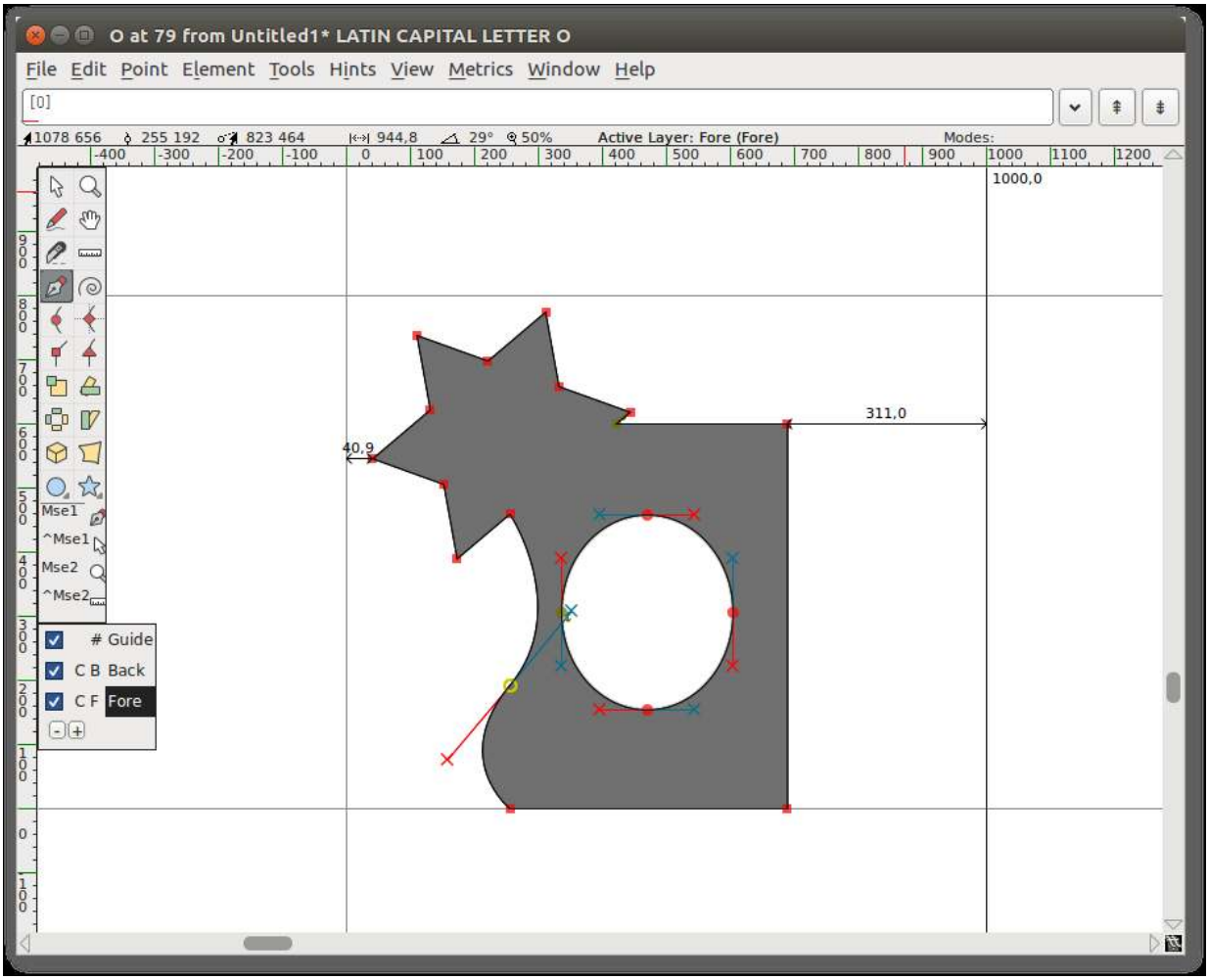
2. 选中星形和之前的图形。你只需要选择每个重叠图形的一个点，但是选择额外的点也可以。

3. 点击Element > Overlap > *Remove overlap*。你将会看到两个图形合成一个。



添加一个点

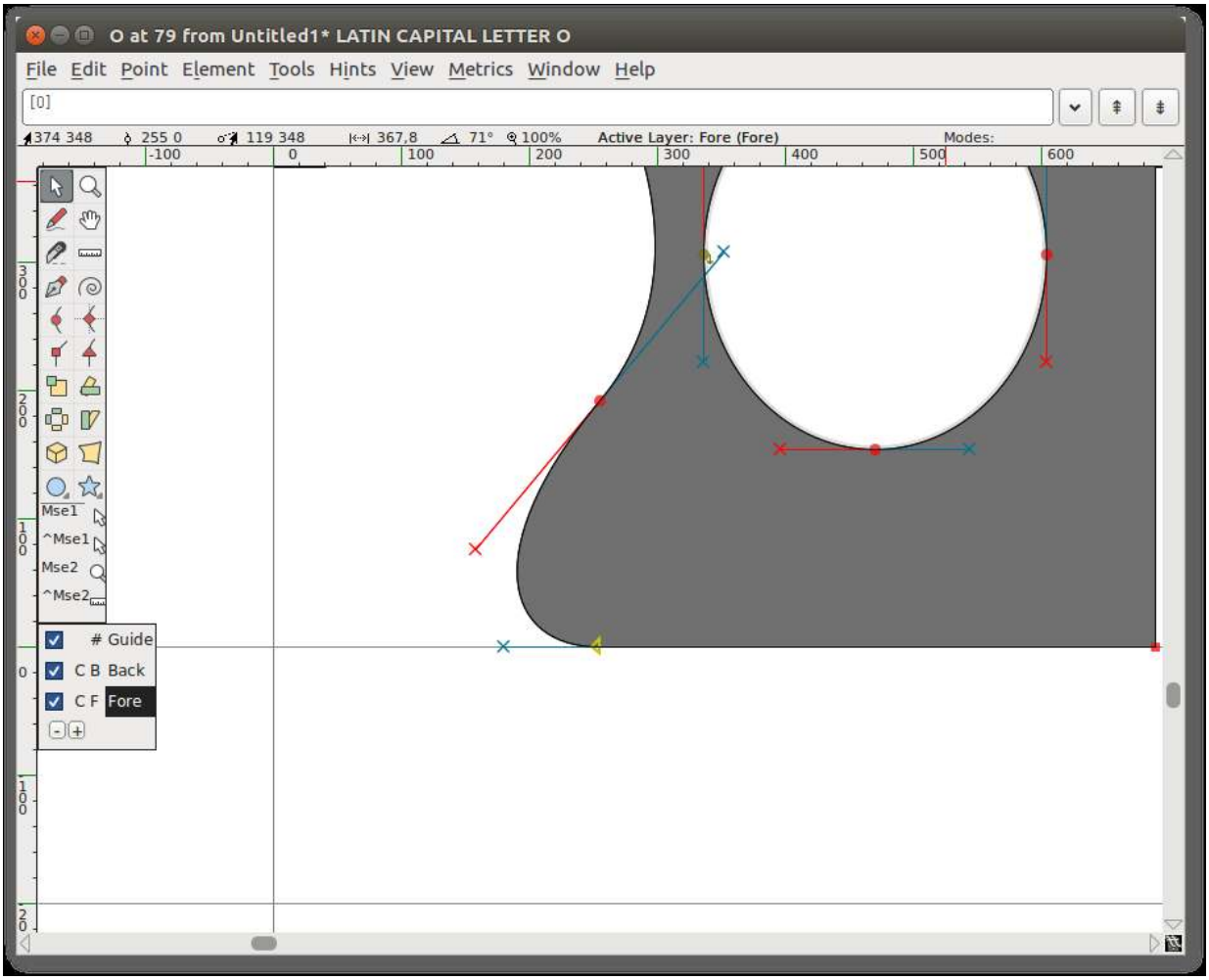
使用钢笔工具，在一条线段的中间点击不放，并拖动鼠标改变图形。



切点

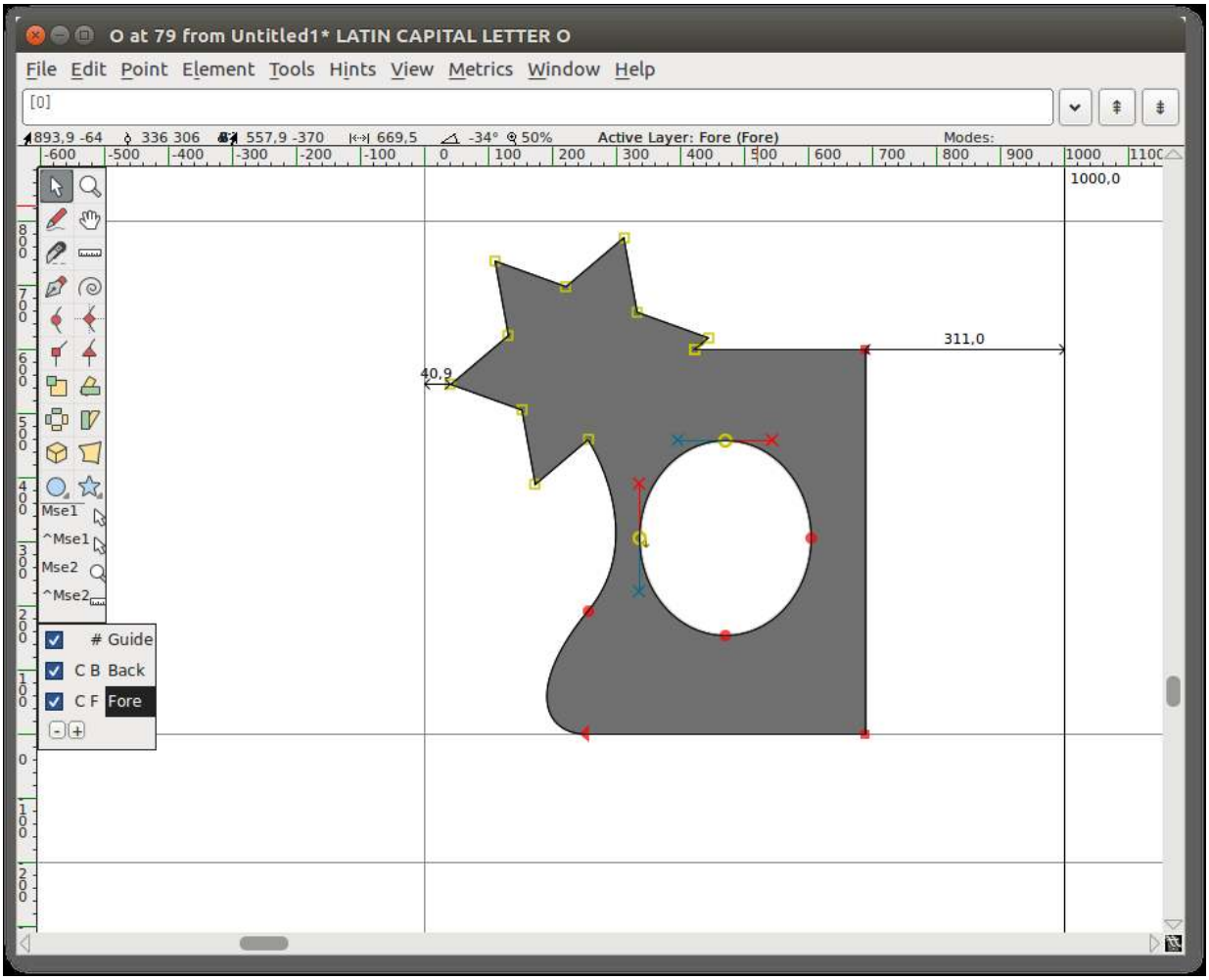
选中你的新图形左下角的拐角点（曲线和直线的交点）。在Point菜单可以看到*Corner Point*是勾选的。选择*Tangent*。这将方形节点改为了三角形，但是做完下一步才算全部做完：扩展控制点。

为了这么做，选择Element > *Get Info*，打开了点信息窗口。在窗口中的Location选项卡的Next CP设置，设定一个大数，比如75。点击OK。你将会看到曲线平滑地进入直线。

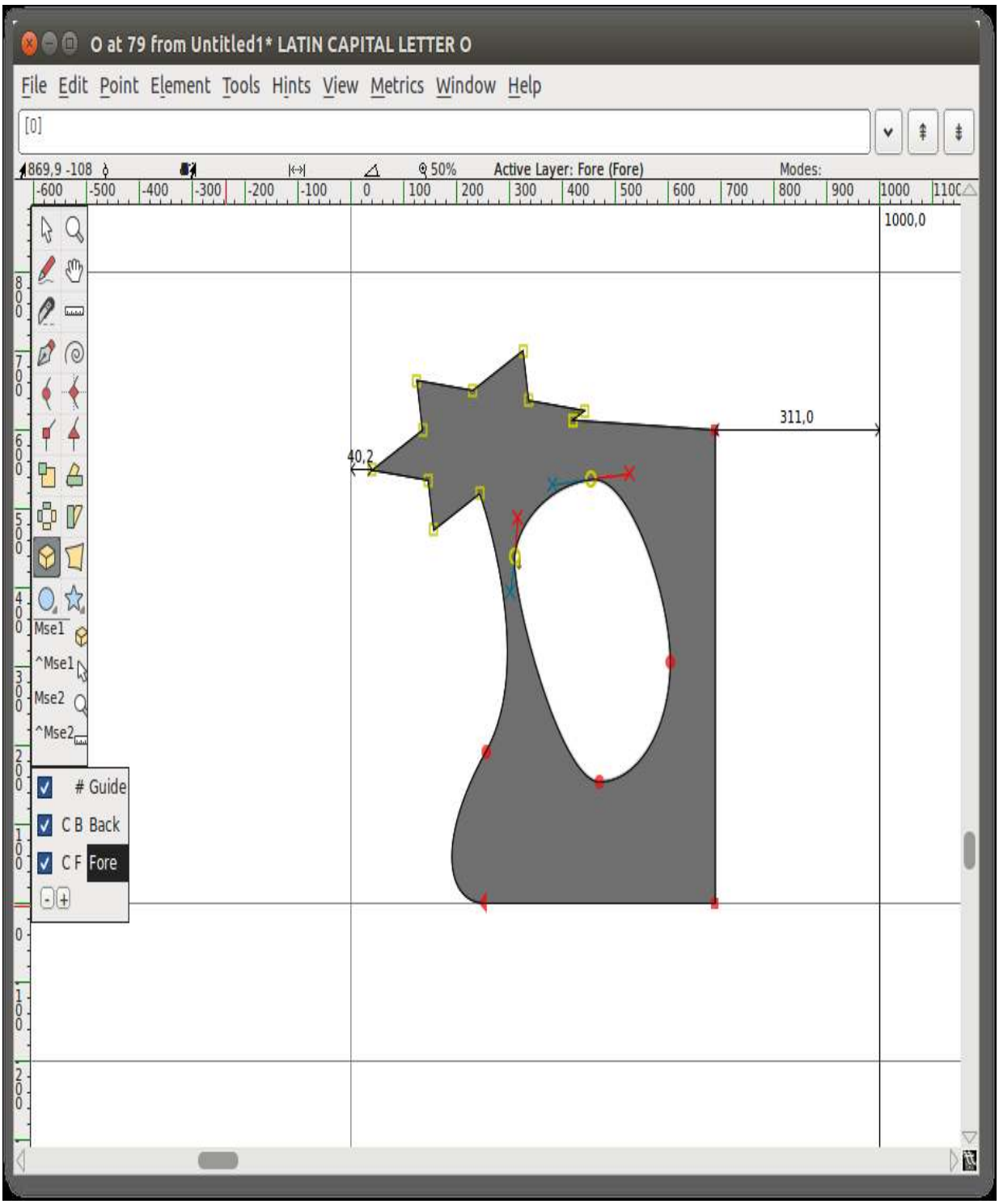


变形

现在选择大概图形的四分之一 — 星形和矩形的一部分在中间。

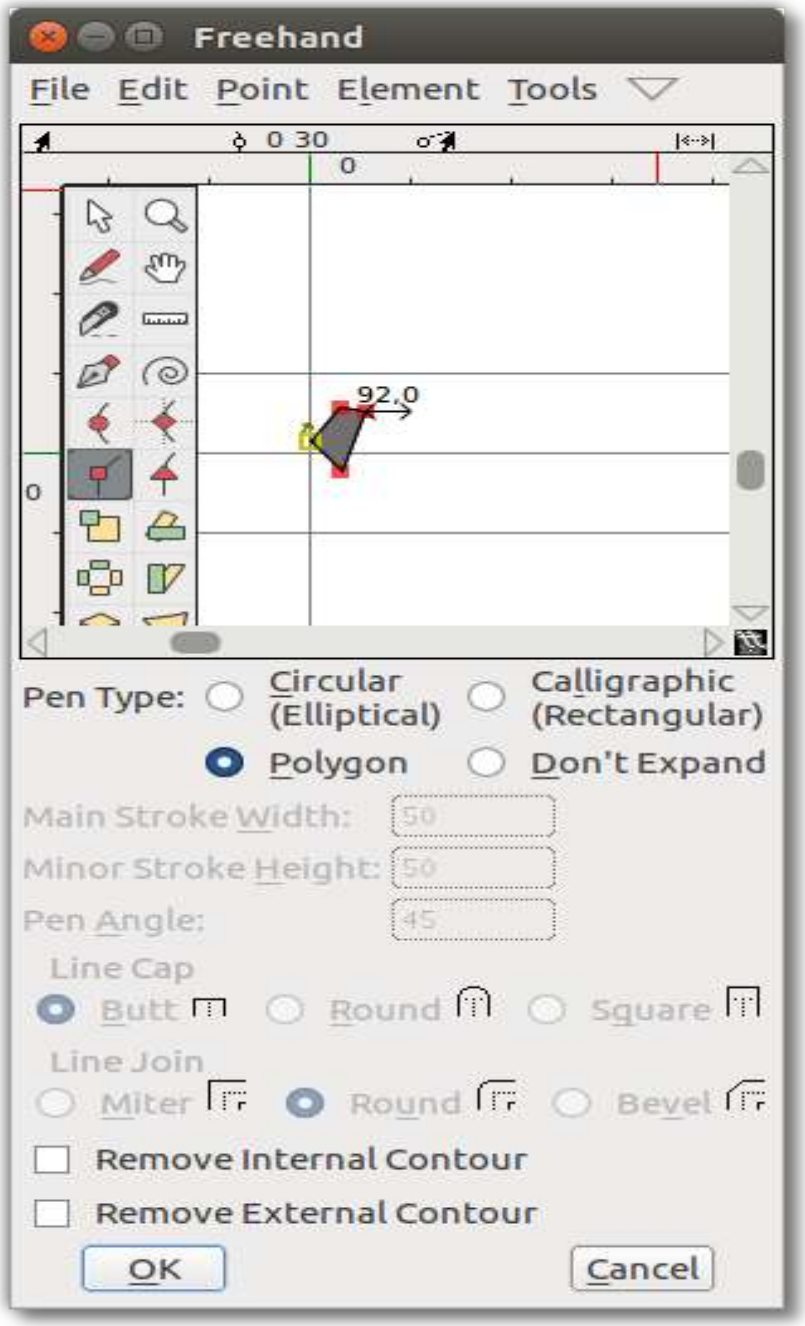


选择3D旋转工具，移动到选择区域的中央，慢慢地点击拖动知道看到你喜欢的形状，然后松开。这是在实际图像中应用3D旋转的例子：



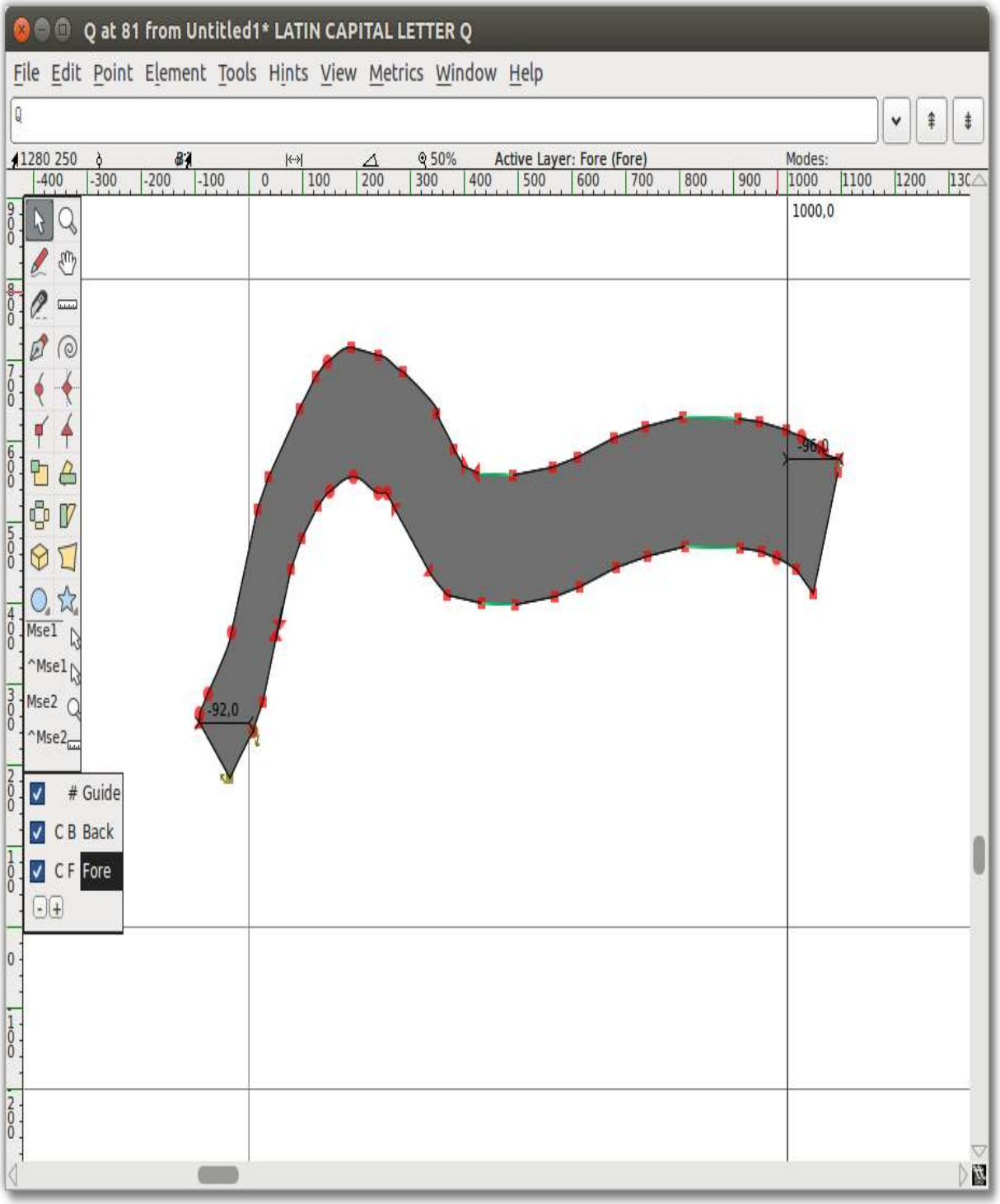
设置绘制形状和宽度

现在为止你已经使用自由绘制工具来绘制一条线。如果你双击自由绘制工具，将会展示包含一个绘制窗口的自由绘制对话框。这里是你选择笔触形状和尺寸。当你选择*Expand Stroke*选项的时候这个对话框也出现在Element菜单中。



使用拐角工具，绘制多边形，点击OK。

现在使用自由绘制工具绘制一条线。当你松开鼠标按钮的时候，像下面一样，新的路径自动使用你在自由绘制对话框里选择的宽度和形状和绘制。



继续绘制！

你应该继续试验绘制工具直到你觉得可以使用它们绘制和变形出你需要的任何形状。到这个时候，你准备好开始构造字形的组成部分，但是你也应该花些时间看看FontForge的其他工具集。下一章“使用Spiro绘制”描述了Spiro绘制模式。Spiro绘制与Bézier曲线编辑太不同了，以至于需要解释其本事。

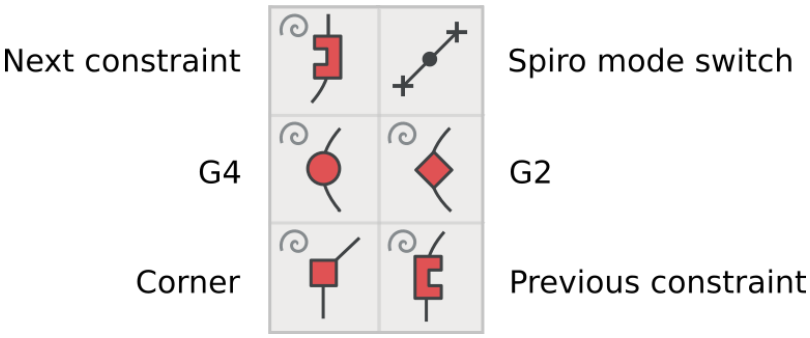
使用Spiro绘制

Spiro是一个使用更传统的Bézier曲线的替换方法来设计曲线的工具包。 尽管这是可选择的，FontForge可以包含Spiro模式安装，提供给你创造特定类型曲线的工具。如何在你的程序中包含Spiro库的详细信息参见“安装FontForge”。

Spiro绘制是一个不同的方法，可以用不同的方式完成你的曲线，解决你概念上的问题。请试验一下！

Spiro工具集

“使用FontForge绘制工具”一章描述的许多工具在Spiro模式下有相同的工具，但是其中一些在Spiro模式下工作地很不同。



Spiro点共有五种类型：

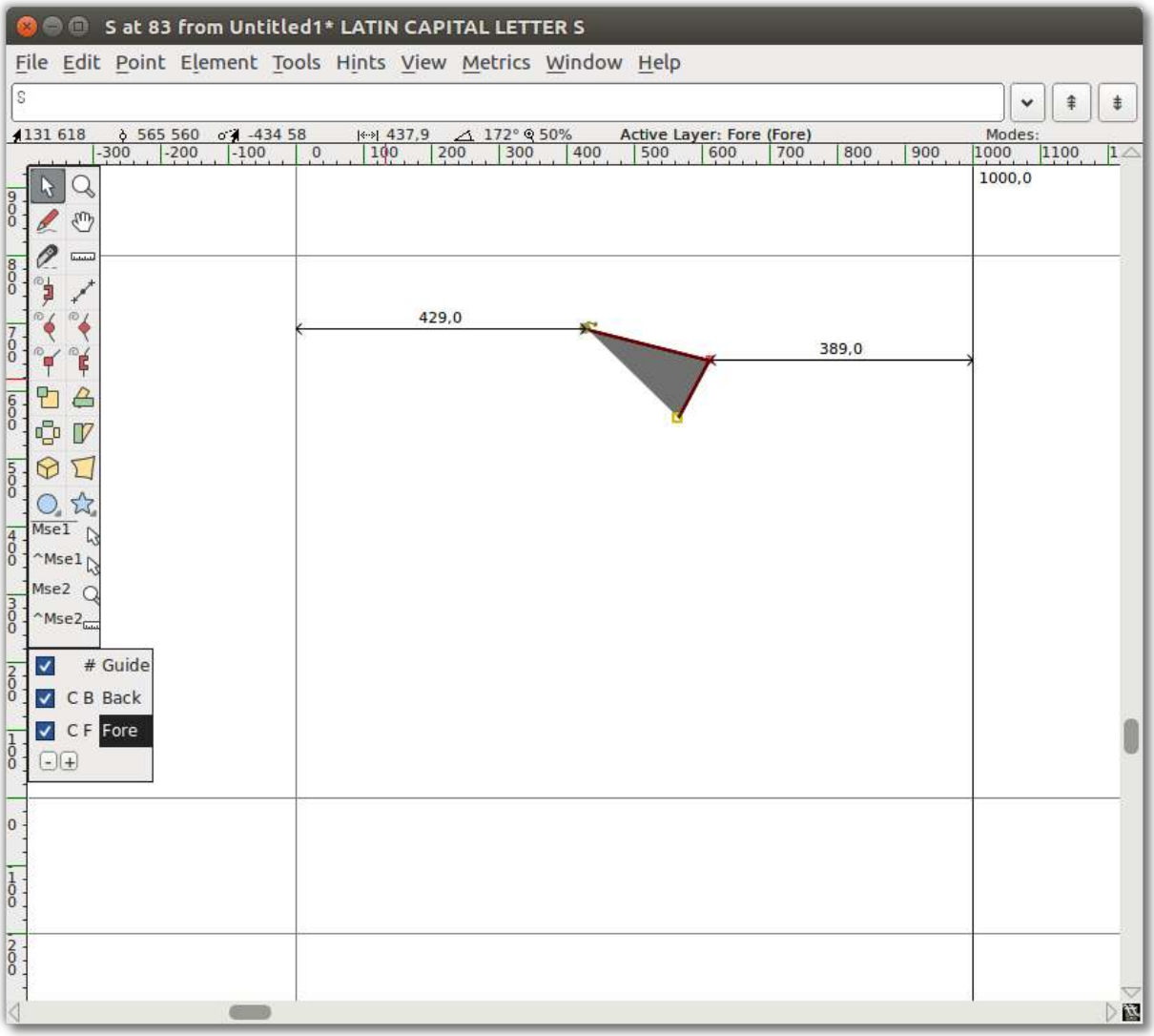
1. G4点，用于更温和的曲线
2. G2点，用于更尖锐的曲线
3. 拐角点，用于尖角的连接
4. 前约束点，在路径从曲线到直线转变的时候使用
5. 后约束点，在路径从直线到曲线转变的时候使用

使用Spiro绘制一个“S”

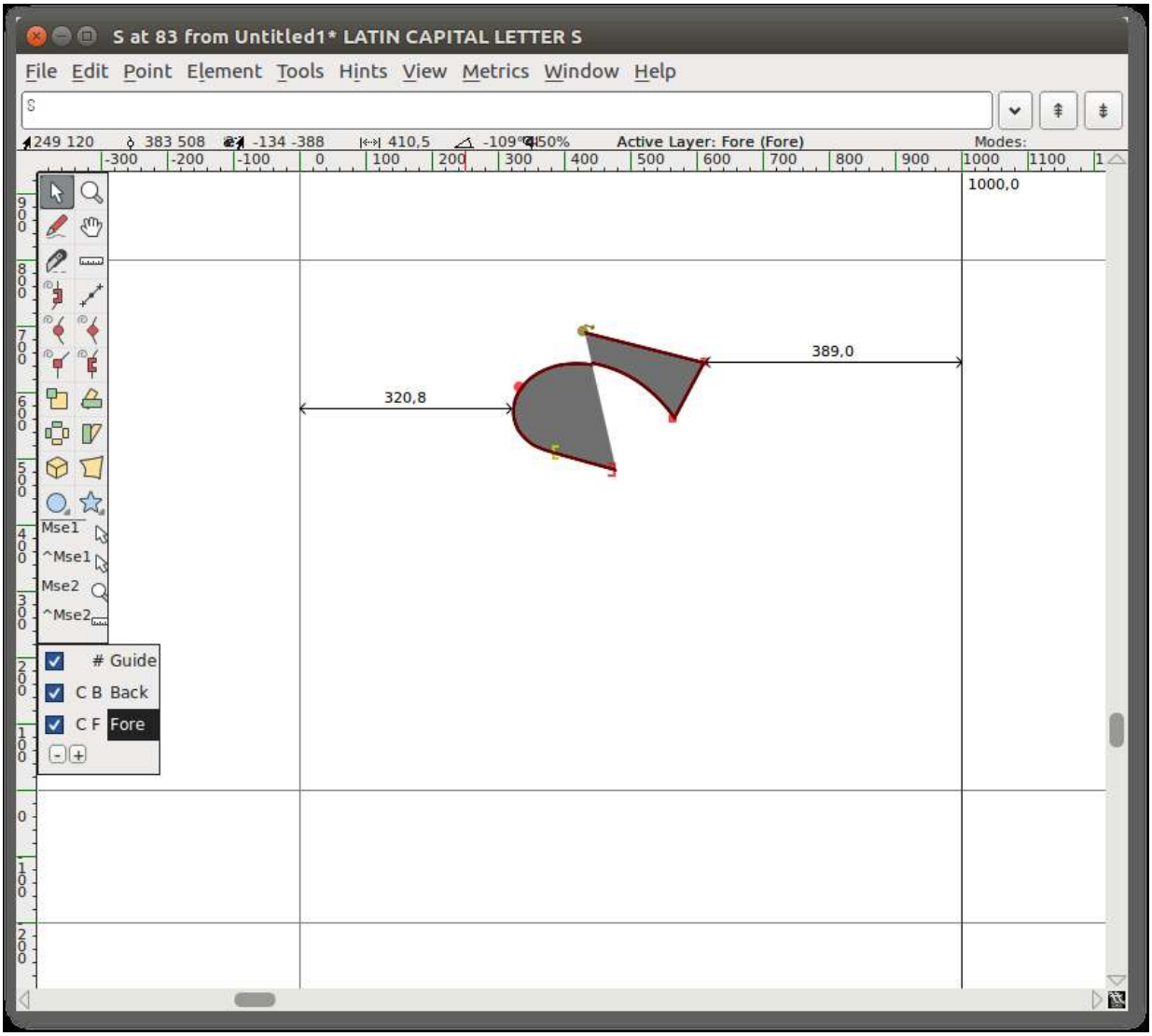
做完用Spiro绘制一个“S”的练习后，你将会熟悉Spiro。

提示：在Spiro模式绘制的时候，经常从一个G4或者 G2点开始。在FontForge中从其他类型的点开始并不真的生效。

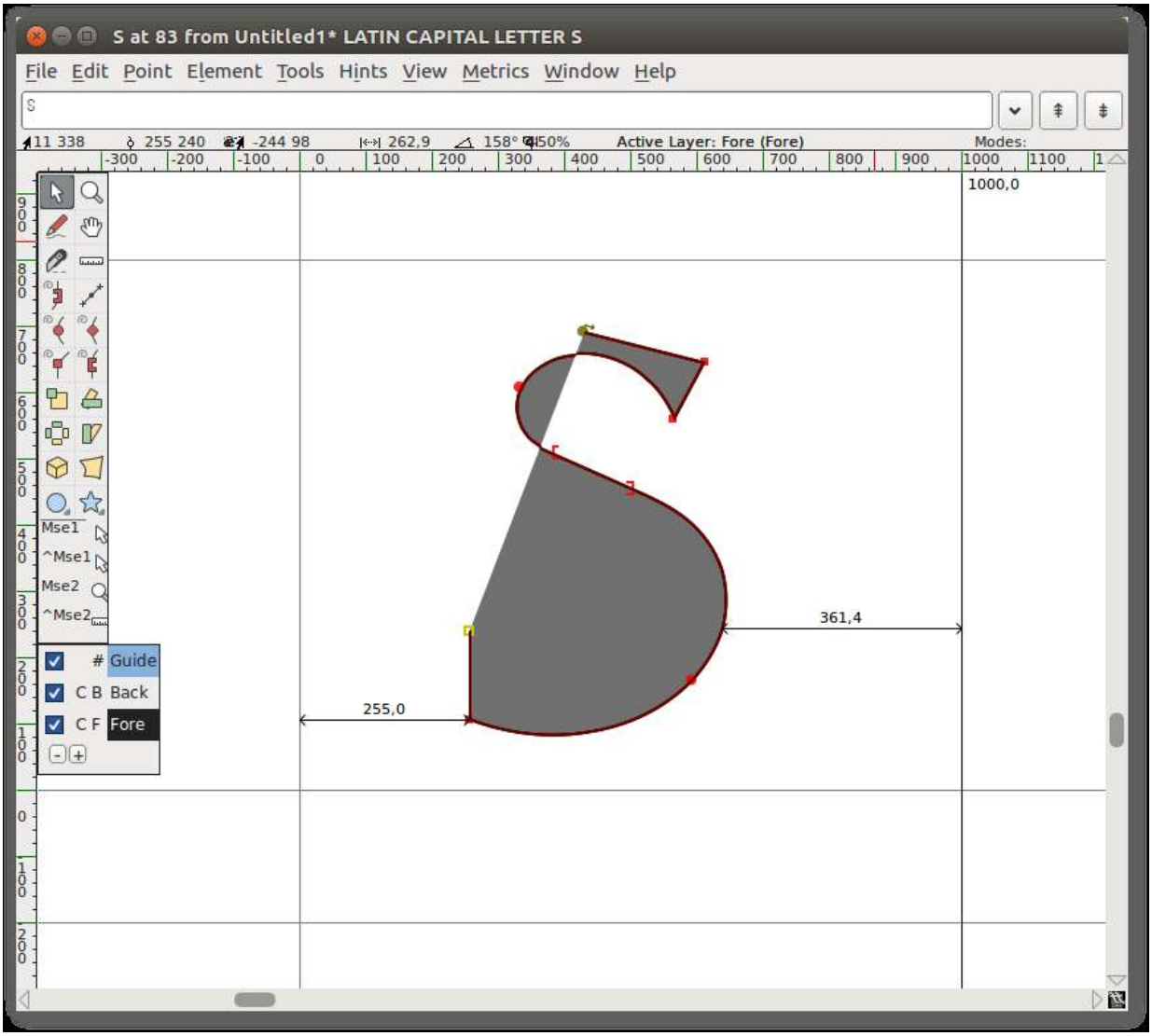
在你的“S”的最上点出放置一个G4点，接下来是拐角点，然后是另一个拐角点。顺时针地完整字母的形状。



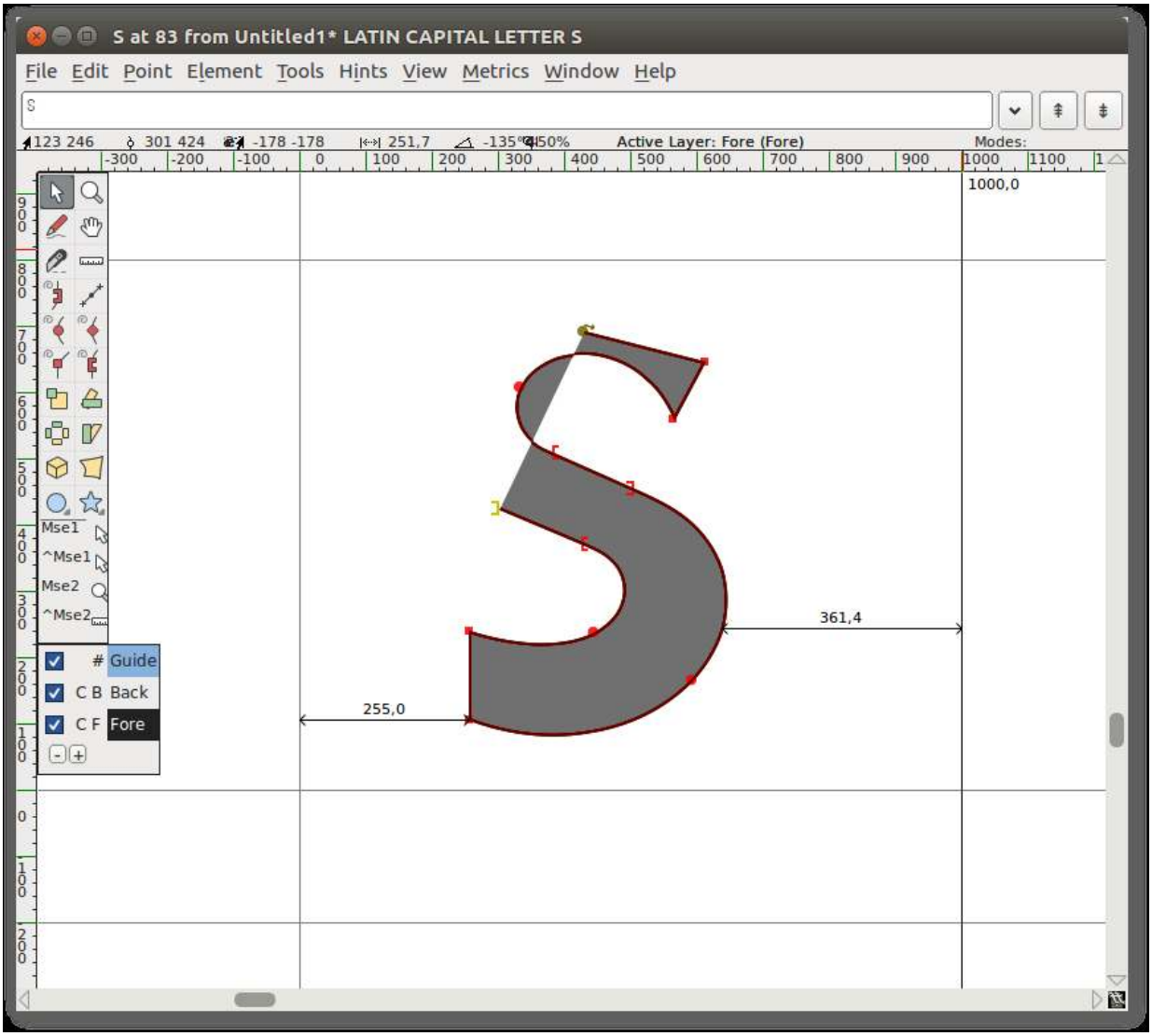
接下来是一个G4，一个前约束点和一个后约束点。



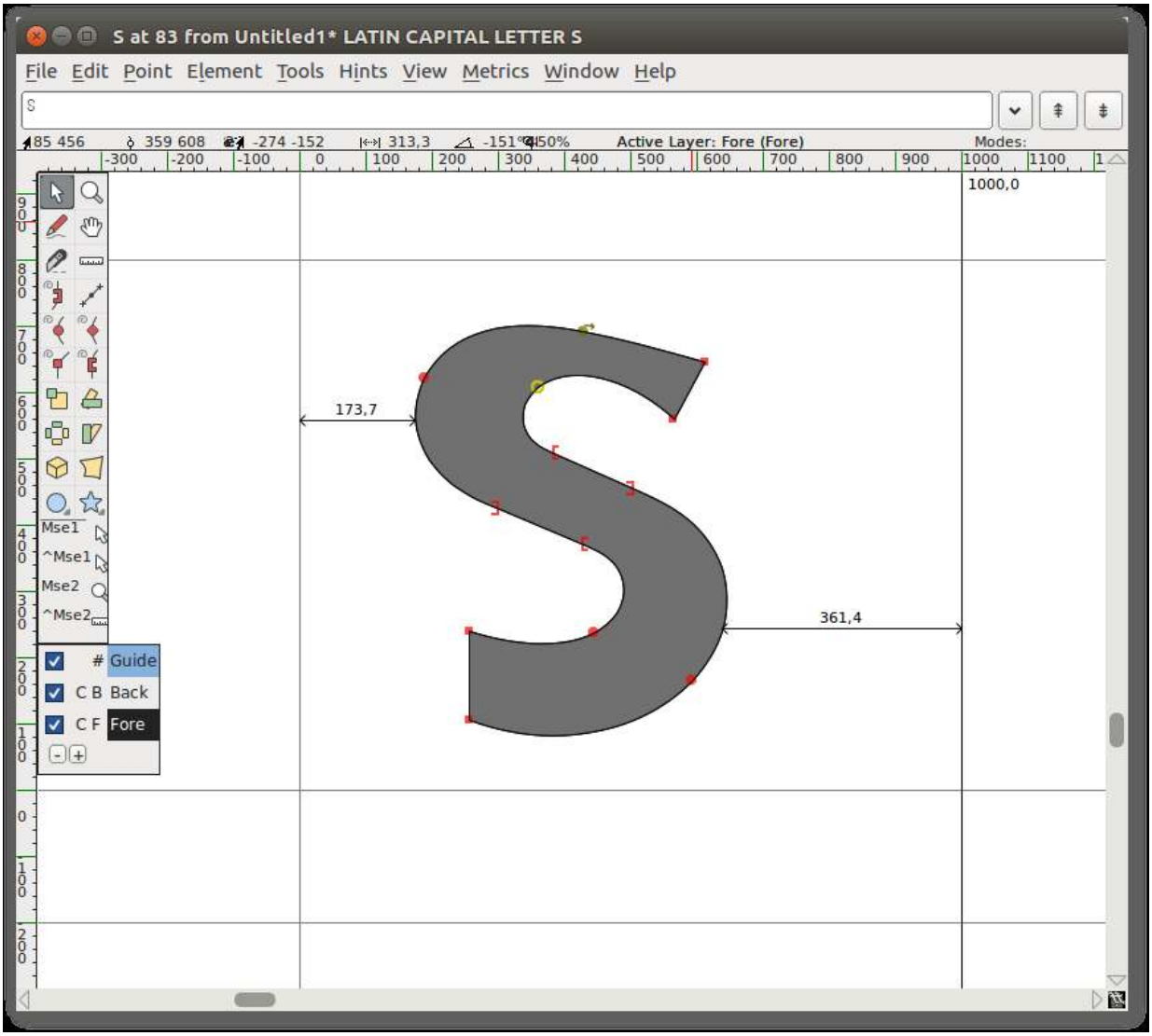
然后在两个拐角点后添加另一个G4点。



然后一个G4，紧跟着是一个前约束点，一个后约束点。

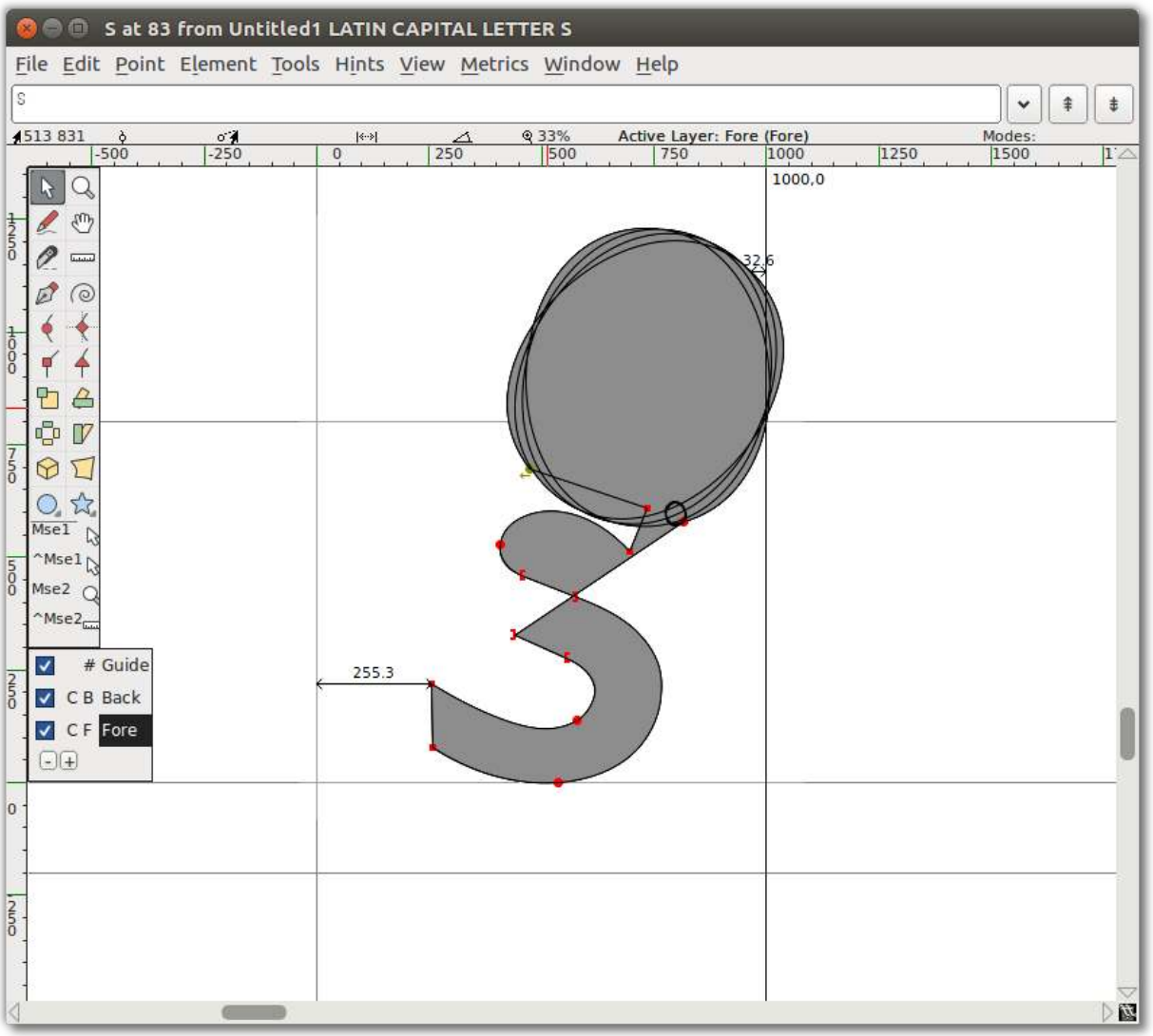


然后添加一个或多个G4点，最终通过在G4点工具中点击开始点，从开始点关闭图形。



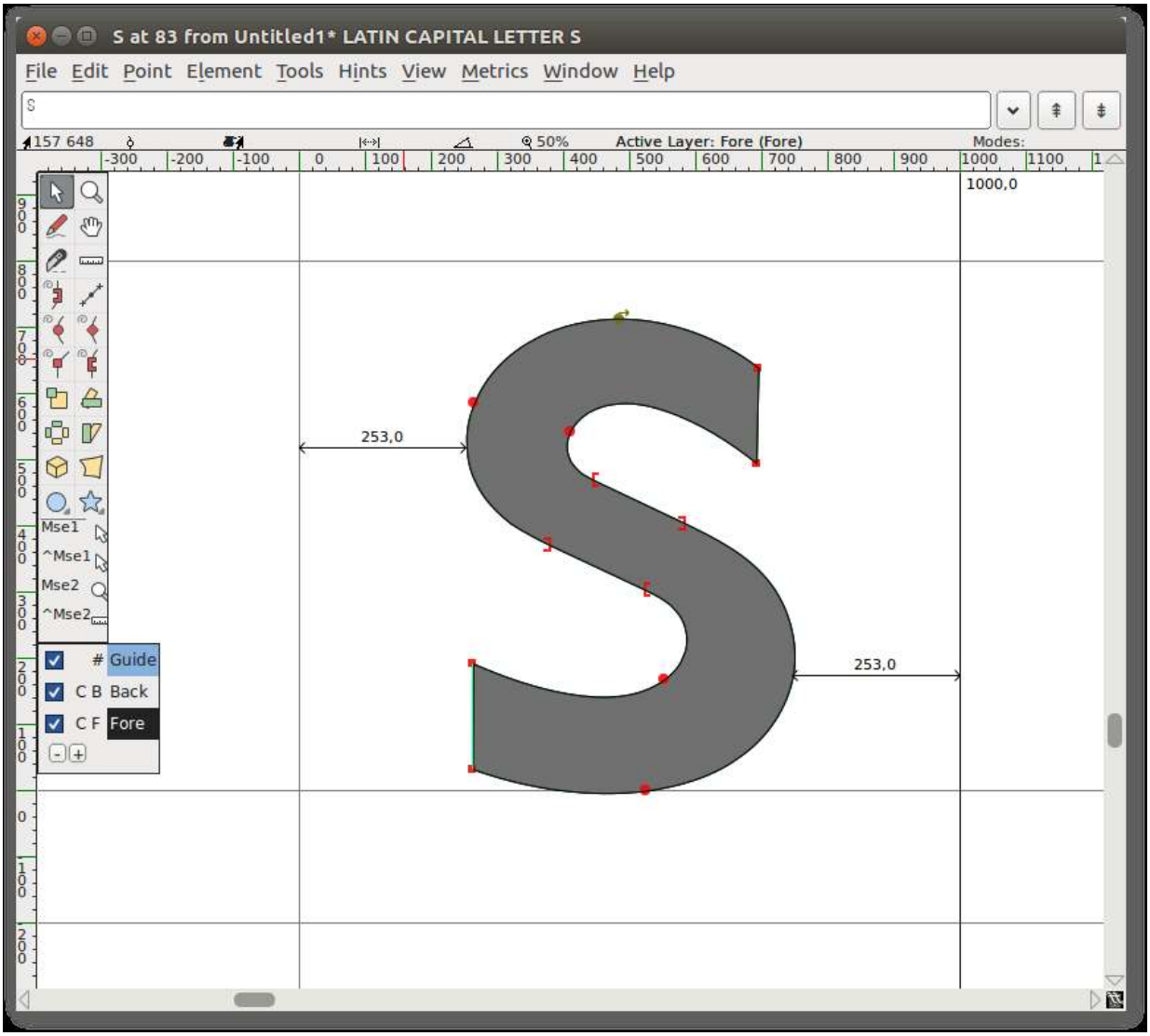
现在你几乎有了一个“S”！开始微调周围的点，得到一个你想要的S。

哎呀，怎么了？

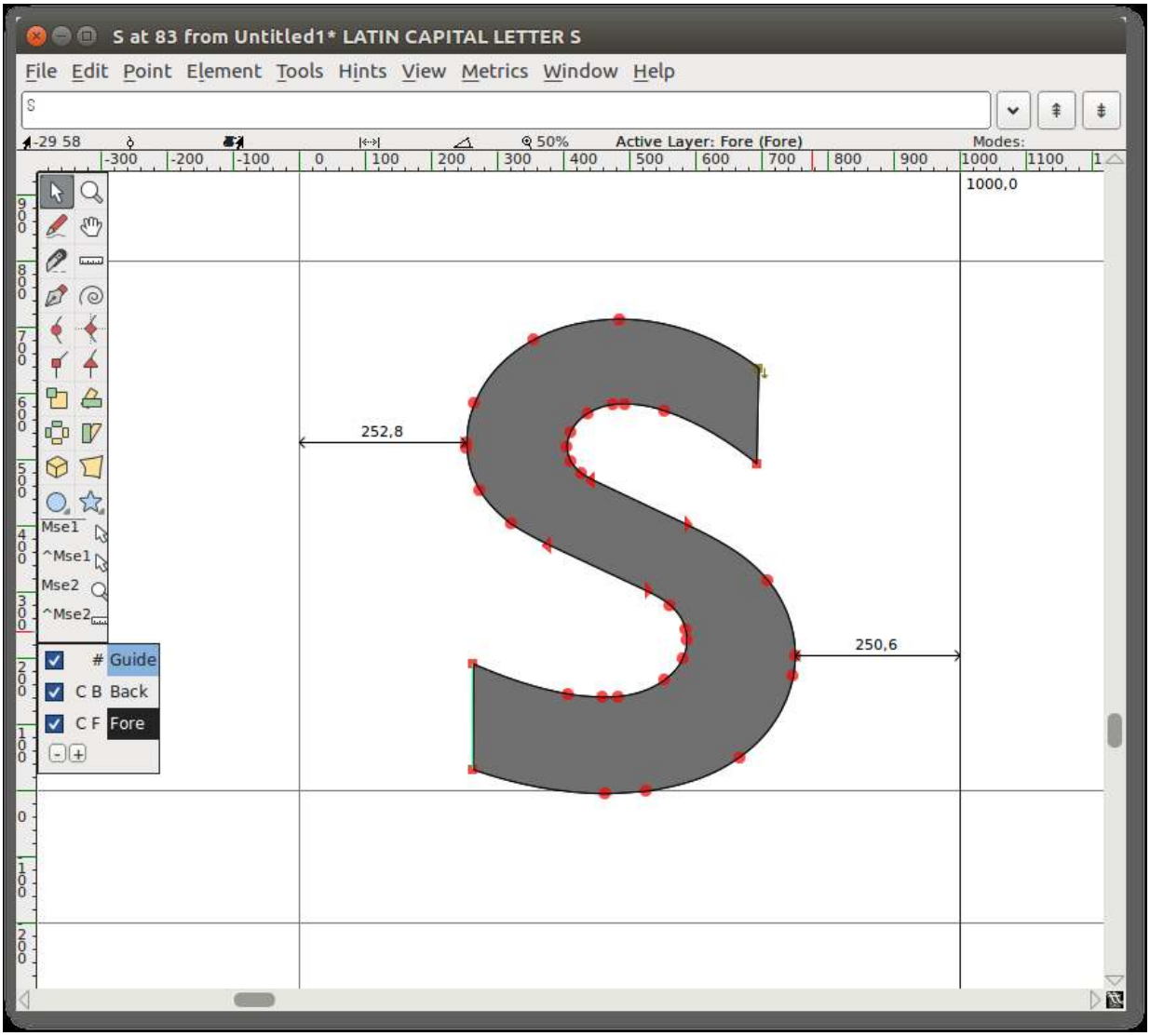


不要担心 — Spiro有时会做出一些滑稽的事情。只需要点击*Undo*,或者一直微调使事情回到正轨上来。

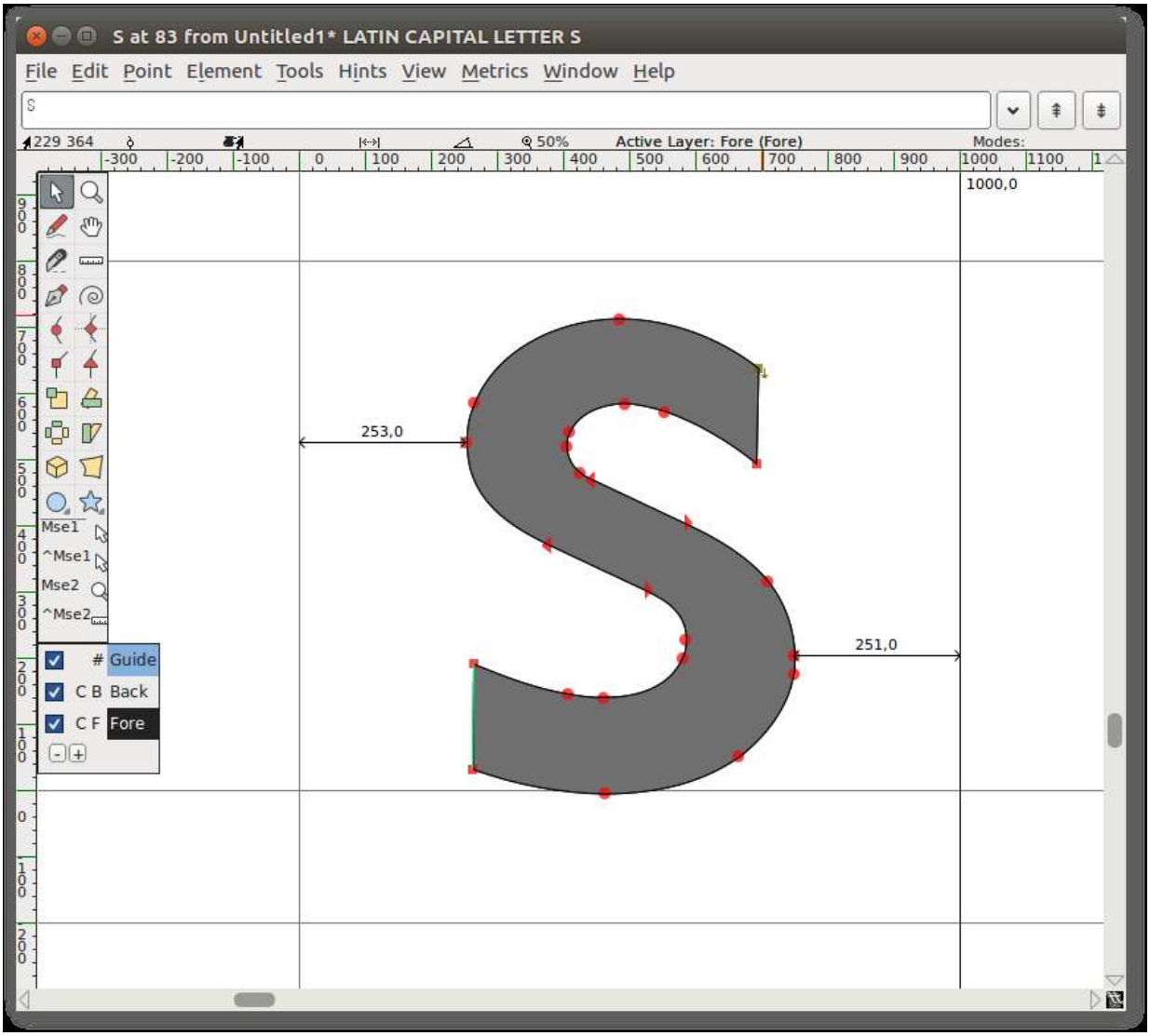
现在你应该有了类似这样的东西：



从Spiro模式切换回Bézier模式。你会看到得到的曲线上有许多点。你可能希望清理其中一些。



为了清理额外的点，到Element菜单选择*Simplify* > *Simplify*。然后点击*Element* > *Add Extrema*。最终点击*Element* > *Round* > *To Int*。做完这些操作后你将会看到类似这样的东西：



你可以继续试验Spiro模式，感受到与Bézier绘制的不同。 术语是不同的，但是有了FontForge的其他绘制和调整工具，练习将会使你作出你希望的东西来。

创造“o”和“n”

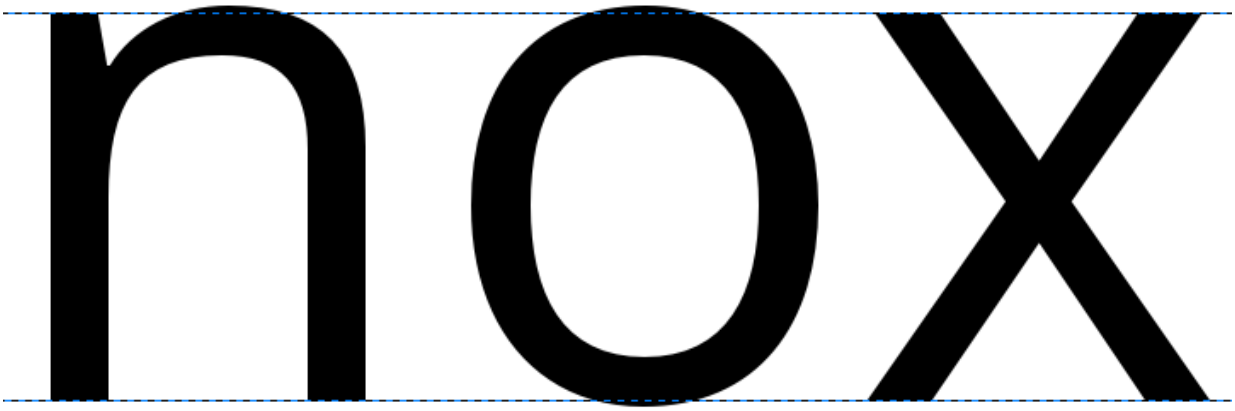
设计字体有很多种方法。解构涉及的大规模的进程，以快速开始并为整个字体的字符的价值提供坚实的基础。这样做是有益的。 达到这个目的的一个受欢迎的有价值的方法是首先设计“o”和“n”，在定下形式、间距和平衡的必要元素，然后再将它们合在一起形成其他字符。创建小写的“o”和“n”字符可以提供给我们一些基本的形式和结构，可以支撑所有其他的需要的字符。

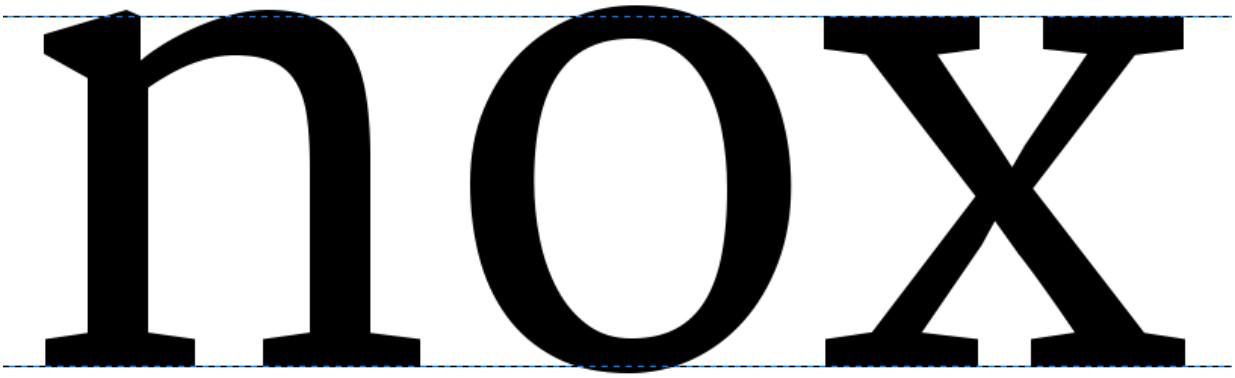
尽管设计“o”可能看起来是十分容易的事情，但是所有在“什么是字体？”一章中提到的字符都开始起作用。你对每个字符所做出的选择应该是深思熟虑的。

下垂和上突

光学效果影响字体设计的一种方式是曲线和直的边在眼睛里是如何出现。 比如说，如果要曲线和直边看起来正确地在基线上对齐，那么实际上曲线必须放在基线以下一点点，这就产生了下突。为了看起来放置在基线上而压低到基线以下的字符的那部分称作 *下垂* —如下演示。没有下垂，基线附近有曲线的字符将会看起来没有在文本的线上对齐。

类似于下突，提供x高度和大写高度的对齐错觉需要一部分区域*上突*（如下）。





设计小写“o”

设计“o”不仅仅是字母中黑色部分的问题。“o”提供了非常基本的弧（bowl）的粗细和形状，白色部分 — 或对立面 — 提供了字体其他部分使用的尺寸和形状。 一般说来，我们也可以在其他字符中看到“o”中的圆形。这些字符包括b，c，d，e，p和q，这些形式也会影响字体中其他字符的曲线的形状和形式，比如O，C，D和Q。

此外，“o”中的白色部分应该在设计字体间距的时候利用起来；“o”也会设定字体中其他字形间使用的间距。这两个值之间关联很大，因此基本上你也需要设计你的“o”两旁的空白数量。作为一般原则，除了斜体或者意大利斜体，其他字体的“o”左右两边的间距是相同，“o”字符的字符串的空白间距应该与“o”内的空白间距相平衡。

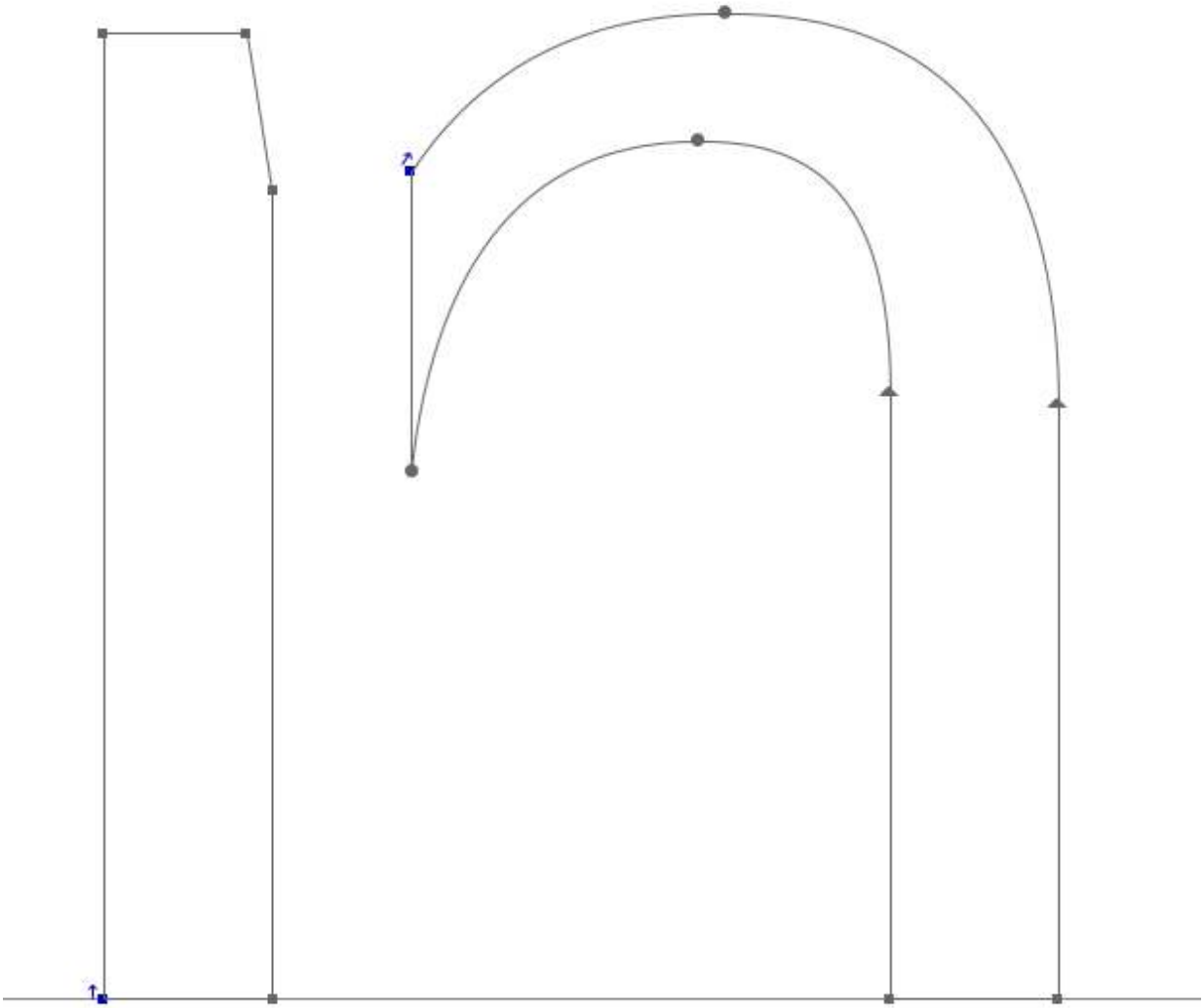
我们已经接触了很多间距和度量值，所以即使在这一较早阶段，你也可能希望看一下“间距，度量值和紧缩”一章，涵盖了字体间距的基本含意。 这样将会使你做出间距很好的字符“o”，这样对你设计“n”也有帮助。

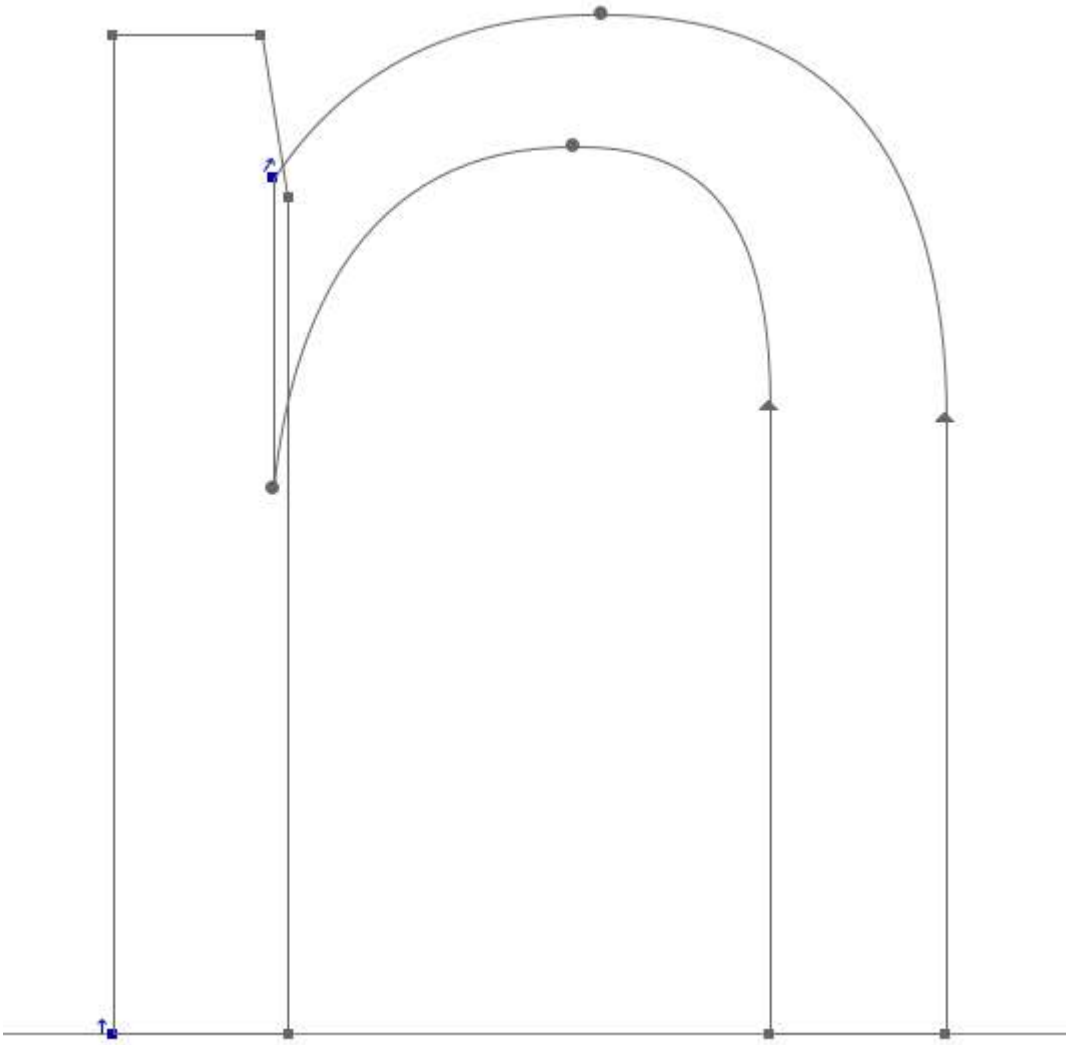
设计小写的“n”

一旦你可以对你的小写字符“o”的形式和间距感到高兴，就像展示的示例字符串那样，那么下一步就是创造形状合适、平衡、间距良好的小写字母“n”，这样你就可以将其插入“o”的字符串。

如果我们观察“n”的解刨，我们可以将其分割成2或3个组成部分，包括一个*茎*和一条*曲线*。

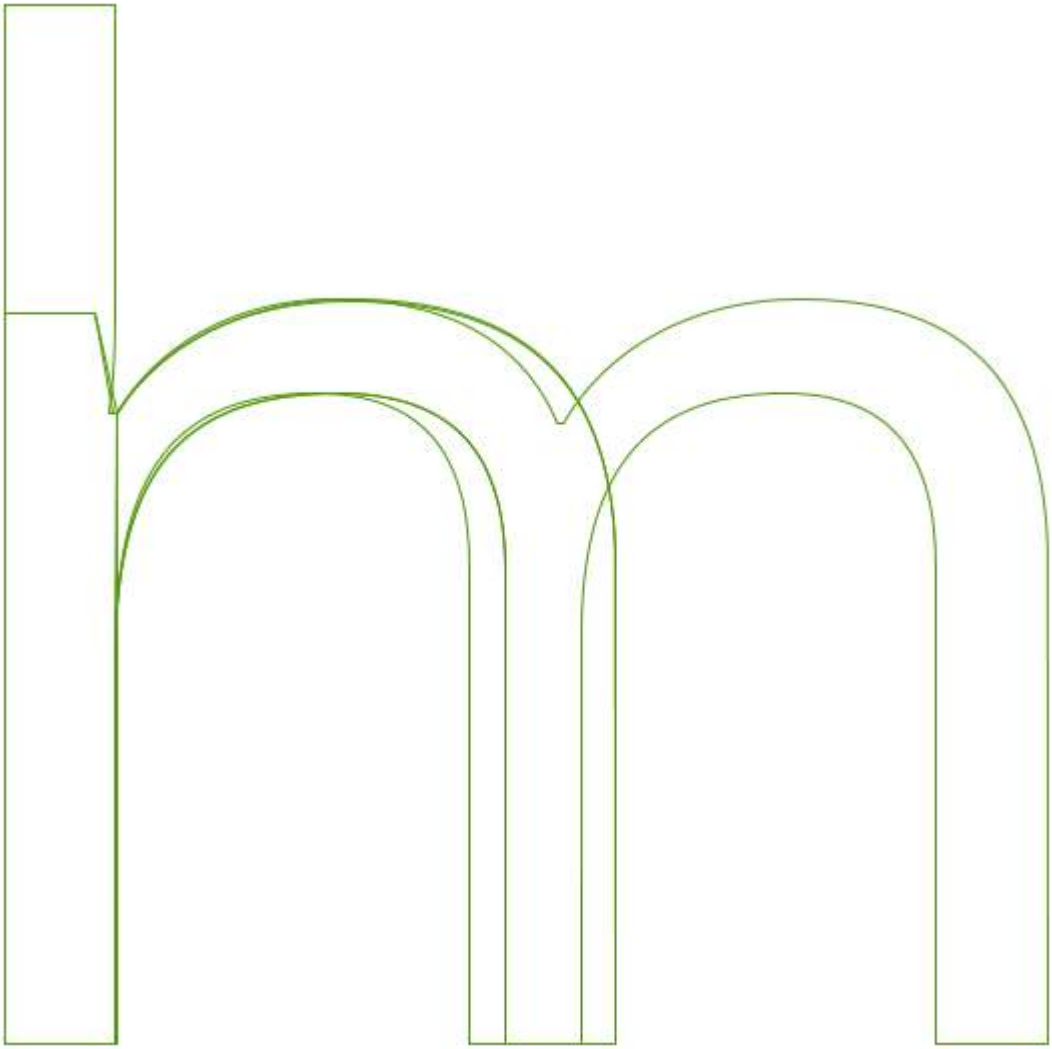
这种方式可以在我们的字符集增长的时候，给你形成字符时与其他字符保持平衡和谐的捷径。 看下面的例子“n”；他被分为两部分。分开的组件合在一起就形成了一个“n”，但是同样的组件在稍后形成其他字符的时候可以再次使用；比如左边的“n”的茎可以在形成其他所有小写字母的茎的时候使用。

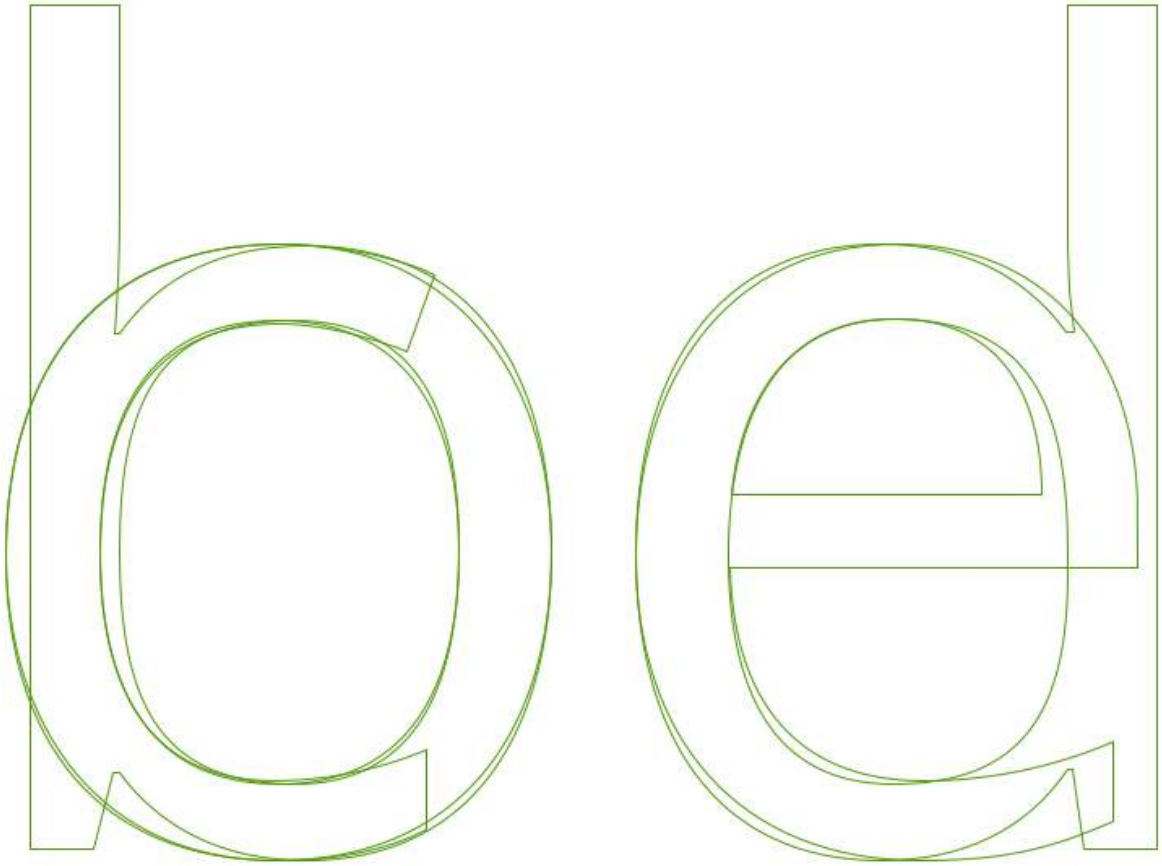


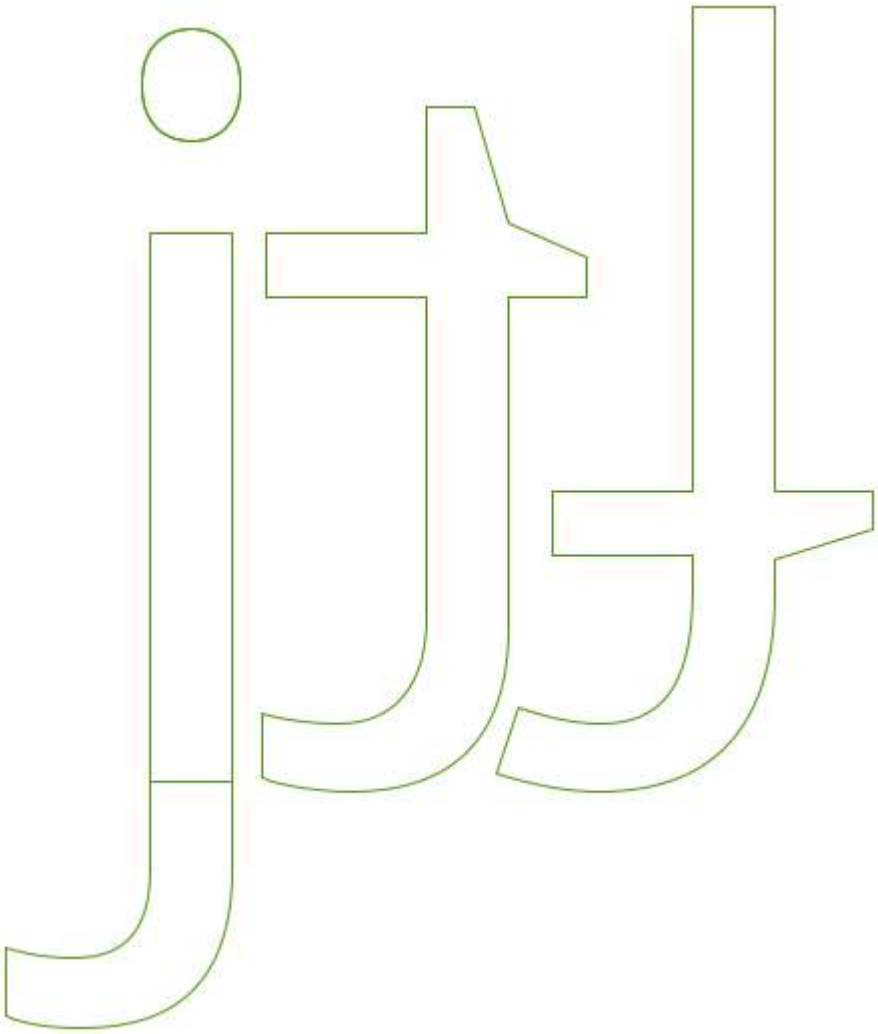


你可以再次快进到间距和度量值的章节，字符“n”的设计应该跟上调整字符“n”和“o”间距的进程。

现在利用你所使用过的方法来创造一个“n”和一个“o”字符，你已经准备好扩大小写字符集。“o”和“n”的茎和曲线组件为你形成其他字符指明道路。 如果我们学习下面的来自Open Sans的字符，我们可以看到各个字符的正式面貌以及他们如何重复出现，通过一些调整形成新的字体的组件。





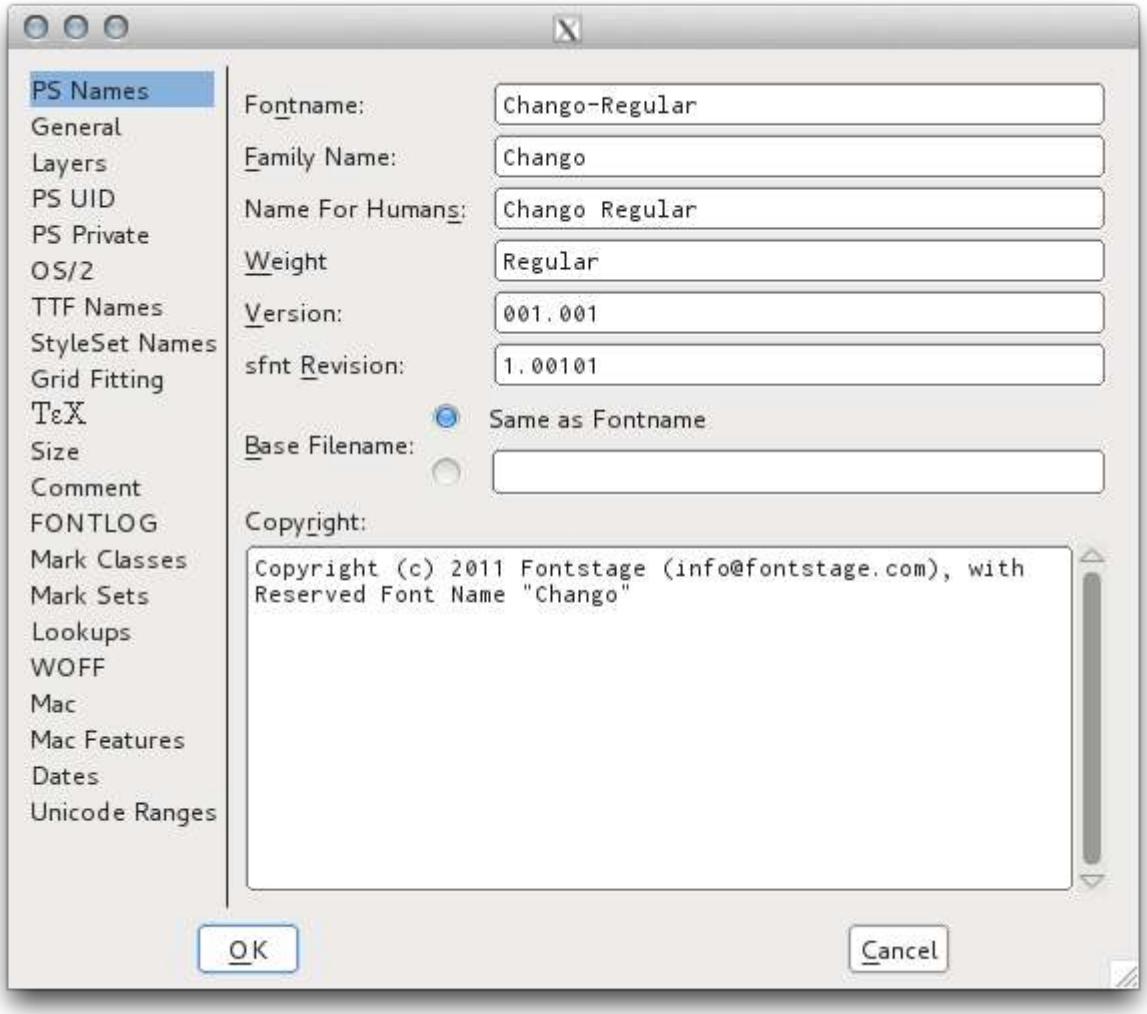


字体信息和元数据

字体信息与元数据

元素，字体信息

任何一款字体编辑器中都有一个编辑字体信息的窗口，FontForge 也不例外。第一次打开这个对话框，你可能有点不知所措。FontForge 严格遵循 OpenType 标准，所以你可以通过它熟悉 OpenType 标准；同样，去了解 OpenType 标准也可以让你熟悉这个对话框中的内容。



版本编号

软件开发者喜欢给他们的每一版程序都起一个代号，字体也可以这样。

从某种程度上来讲，字体是文字操纵用户情感的 API。

主版本号的更新对应较大的重构和调整。

拿 Exo （此 Exo 非彼 EXO）和 Exo 2 举个例子，如果你写东西的时候用到了 Exo，你就不会换用 Exo 2 了，因为人的感官会很敏锐地察觉到它们之间微小的差别。

此外，添加了高度相似的多种文字，添加了新语种的支持，或垂直／水平规格（vertical or horizontal metrics）的改变，也可能导致主版本号更新。

然而，如果设计了两个互补的文字，最好的做法可能是制作两到三个字体族，每种文字放到一个字体族中，然后把剩下的文字单独作为一个字体族，以在多语言文字排版中作为回退（fallback）。

副版本号的更新则对应小的改变，比如改变纵向规格（vertical metrics）和水平跨距（horizontal side bearings）、调整字间距（kerning）或对某些字形（glyphs）做微小的修改。这些改变会使使用这个字体排版的文档结构有细微的改变（尽管多数情况下很小）。这里有一个例子：

为了完成对一种文字的支持而加入十几个字形（glyphs）很有可能只更新副版本号，特别是在垂直规格（vertical metrics）没有改变的情况下。

补丁级别的更新一般指的是在不明显改变规格（metrics）和字形（glyphs）设计的调整，例如改善 hinting 或修改元数据，这样的更新一般都不会影响最终文字排版的效果。但可惜的是，补丁版本 OpenType 并不支持在元数据中写入补丁版本号（就是在第三个位置的版本号），所以常见的替代做法是更改副版本号。

版本号最多只能有五位数，比如 `2.00099`。

如果你想发布了一款自由字体，Github Releases 的信息会对你有所帮助。

字体族命名

微软费了好大的劲让为旧版 Windows 设计的软件在新版本 Windows 上也能运行，以此鼓励人们更新换代。同样，在 Windows 3 中引入的基础 TrueType 字体模型现在仍然可用，这也就是为什么 Windows 不支持超过 4 种基本风格（正体、粗体、斜体和粗斜体）的字体族。

这就是说字体设计者们要设计一个在所有操作系统中都能用的字体族名称。使用 OpenType 可以轻松做到这一点。OpenType 在分别在字体族名和风格名中补充了「Preferred Family Name」和「Preferred Style Name」，使得不支持 OpenType 的软件也能兼容。

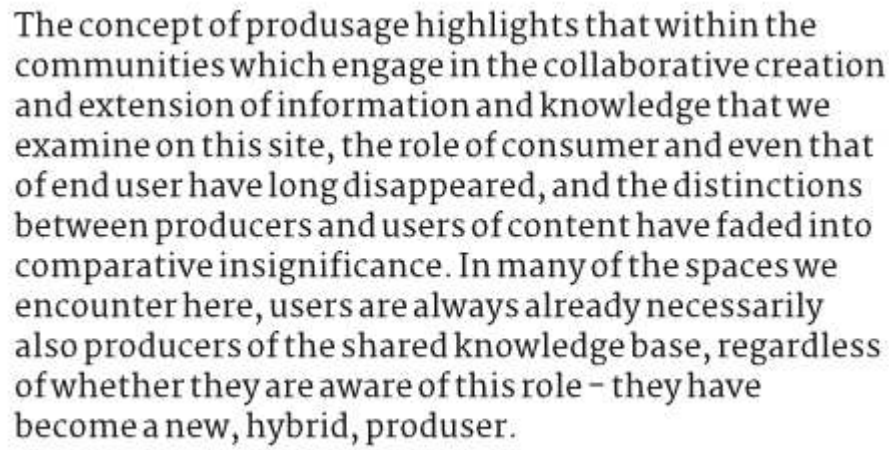
《字体族命名》这份文档会给你更多帮助，它是由来自波兰的字体专家 Adam Twardoch 在 Fontlab 论坛上的帖子中提供的信息整理而来的。

又及：

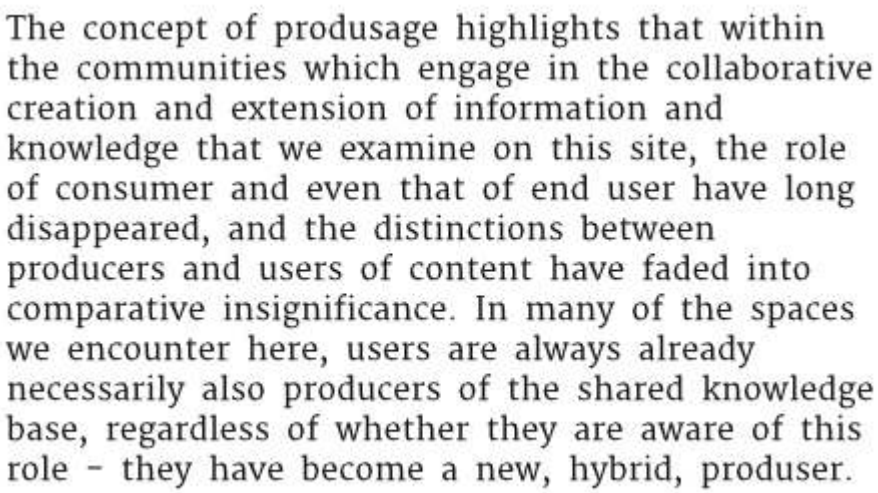
- 删除 NAME 表中所有 PID=1 的键 — 这是为 Mac OS 9 或者更早版本准备的，已经是十多年前的了
- 删除 id 为 18 的 NAME 表 — 这个表也是只为 Mac OS 9 准备的

字间距

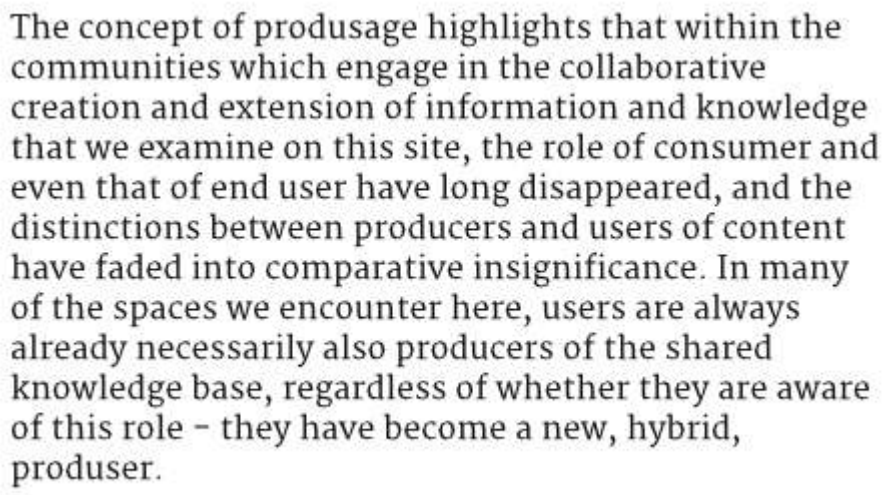
特别关注字间距可能听起来是可笑的，但是它是字体设计中最通用的部分之一。 太宽或者太窄的字间距将会毁掉字体的设计。在你刚刚建立了第一个字符的时候就开始考虑字间距并不会太早。在这一点上你做的决定会在字体设计的过程中逐渐调整。



字间距太紧了…



…这里又太宽了。



现在这个平衡地很好.

如果你的字体将会以大尺寸来使用，那么字间距可以减少。反之亦然。

研究表明过大的字间距比过小的更可以容忍；所以如果你不确定的话可能希望错误往这个方向发展。

注意：类似的研究发现相比于成人读者的标准尺寸，更大的字间距将会对幼儿更有益处。

Linda Reynolds and Sue Walker (2004) — 'You can't see what the words say: word spacing and letter spacing in children's reading books', *Journal of Research in Reading*, vol 27, no.1, pp. 87-98.

创造你字体的基因

在你为“o”和“n”完成了良好的设计和间距，下一个要做的事情就是使用一些字母来繁殖字体，这些字母的结构特征为制作字体中许多其他的字母提供了有用的基础。

可能急切地尽快繁殖所有的字母是很诱人的 — 忍住这个强烈的欲望！ 尽管“o”和“n”提供了设计基础的极好的开始，但是我们需要建立剩余的部分。在这之前快速扩张意味着整个项目会变的难以管理 — 并且花费比必须时间更多的时间。

我们还需要什么来建立我们设计的基础？ — 首先让我们看看通过“o”和“n”我们得到了什么。

尽管“o”对于计算基本间距格外有用，但是它并不能帮助我们设计其他字符 — “b”或者“d”也不一定。 另一方面，字母“n”很有用，因为它对于制作“m”、“h”和“u”有帮助。在我们为我们的基础选择字母的时候，其他我们需要权衡的因素是字母使用的频率。一个使用很多的字母将会帮助我们制作测试字。由于这个特别的原因，一些字母可能被唯一地选中。

你选择的字母不必是这里推荐的。他们仅仅拥有讨论过的特征。因此实际过程中你可能希望以“a d h e s i o n”开始。这些字母集是英国雷丁大学艺术硕士的字体设计课程中所使用的。 另一个可选的是“v i d e o s p a n”。这是*Type Together*工作室选中用来开始他们自己的字体设计项目。其中任何一个都有足够的DNA来从而很有意义，两个都很小，这样可以方便地作出全局改变。

最简单的方式是简单地使用上面集合中的字母，同时你也可以构建你自己的：问问应该挑选什么字母添加到“n”和“o”中？考虑下面的选项：

- “a” — 字母“a”也是一个非常通用的开始选择。“a”对于“s”的结束期望的样子很有帮助。
- “d” — “d”的形状可以使你对“b”、“p”和“q”的设计认识很多。
- “e” — 在英语和许多其他语言中，字母“e”尤其多变 — 这使得它尤其有价值。“e”的形状也可以用来设计“c”。
- “h” — 虽然“h”可以相当快地从“n”构造出来，但是它也提供了测试包含顶部的结构的多样性。
- “i” — 就像“e”，字母“i”相当通用，并且可以让你了解一点“j”的样子。“i”的形状也可以通过“n”的形状推断出一部分来。
- “s” — 早绘制字母“s”是好的，因为它为你将测试的字母的结构添加了可见的多样性。字母“s”也非常难以做正确，因此早点开始它很可能让你在项目结束之前可以花费更多的时间让它正确。 “s”的结束有时对预测“a”、“c”、“f”、“j”和“y”的结束有帮助。
- “v” — 字母“v”可能对预测“y”和“w”有帮助。

一旦你有了这些字母，通过测试来自它们的字母来花费时间改善他们是很好的。在开始“n”和“o”之前，应该花费大量时间在字母的间距和对立面与这些间距的关系上。

构建一个测试文本

有许多在线资源可以快速构建你的测试文本：

- LibreText是一个自由软件解决方案。
- Adhesion Text，由Miguel Sousa制作的第一个这类资源。
- JAF Generator，由Just Another Type Foundry提供。

大写字母

制作大写字母应该按照与制作小写字母非常类似的模式。你以一些关键字符为开始，这些关键字符的形状和特性影响到那些使用公用形状的字符。就像设计小写字母时那样，字母使用的频率也是字母的选择的一个重要因素。

首选要设计的两个字母是“H”和“O”。这些字母设计时不应该只注意他们之间的互相关系，也要同时注意与已有的小写字母的关系。

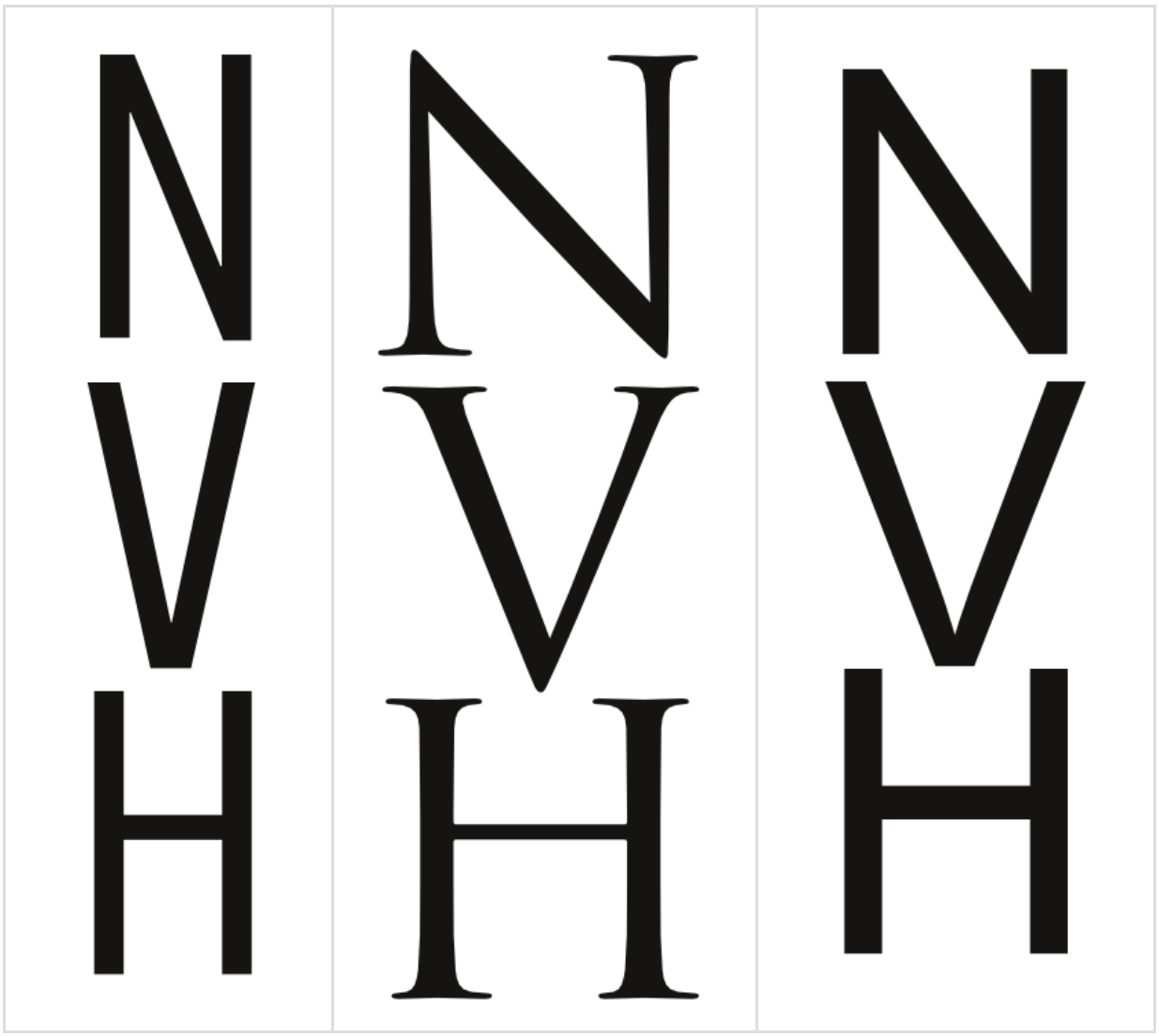
在这个阶段你应该确定小写字母和大写字母的比例。你可能希望调整你的小写字母的顶部或者底部，或者调整大写字母使得与小写字母的比例适合你设计的目的。

大写字母的描绘宽度经常需要比小写字母重一点。你可能希望建立插入试验来快速找到应该重多少。

下一个考虑添加的字符集时A E S I N和P与D中的一个，可能还有V。

根据你已经制作的字体的样式，你可能发现大写字母需要比小写字母更多的宽度上的变化。E、S和P的宽度大体上可能比H更窄或者近似。

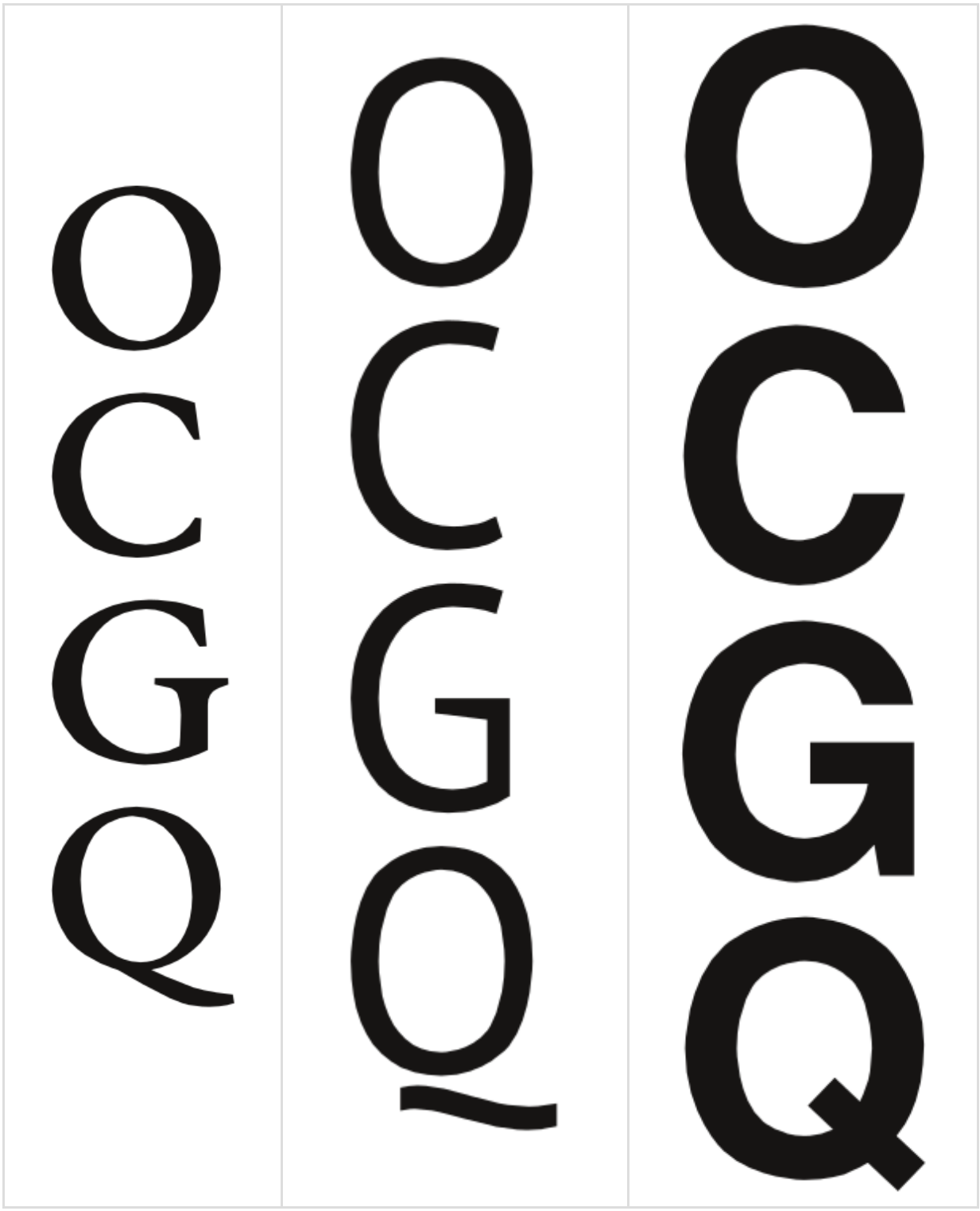
通常说来，N和V经常类似于H但是更宽一点。



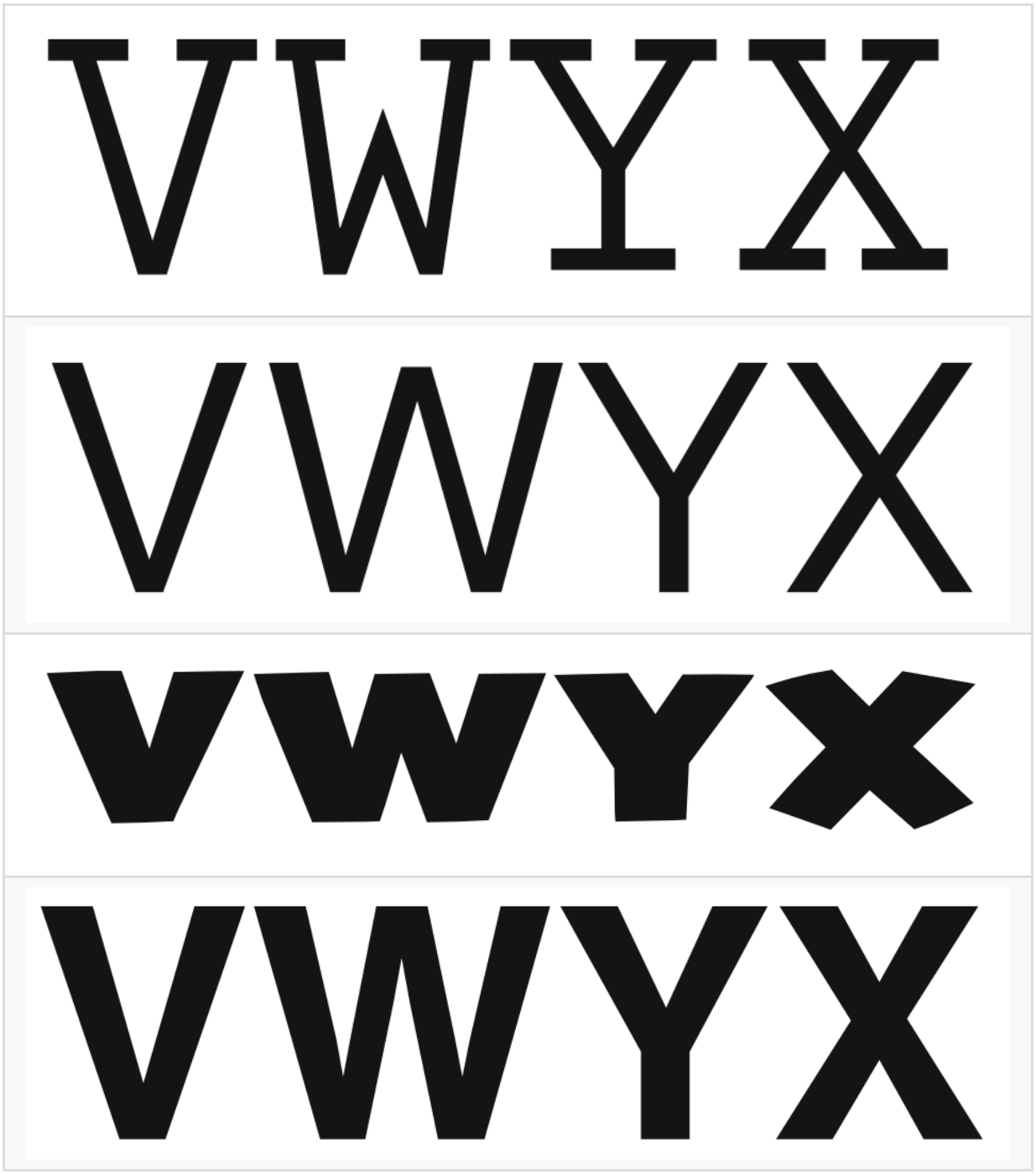
D可能类似于H或者更宽一点点。

O的形状可以在设计C、G和Q时参考很多。H的形状会让你知道I和J和B D E F K L P R的左边的一些信息。

它也告诉你T和U的一点信息。A的形状可以让你知道非常多V的信息。


OCGQ
OCGQ
OCGQ

V的形状和比例告诉你一点如何设计Y W X的信息。Z的形状是与众不同的。


VWYX
VWYX
VWYX
VWYX

行间距

当你有了字间距和字母n与o的集合，你可以开始看行间距。但是行间距的最终选择直到你了解了大写字母和一些标点符号后才可能做出。

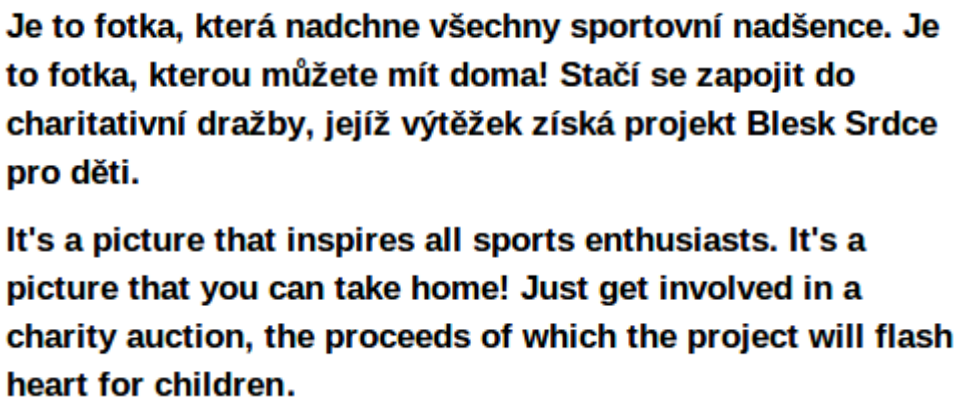
有意地思考行间距

就像字母和字间距的情况，行间距太多或者太少会使得你的字体在真实世界的使用中显得尴尬。找到合适的行间距平衡就是有意地考虑这个问题并考虑测试做出最终决定的过程中的一些选项范围，这笔其他更重要。

一般规则是，大多数新的字体设计者更倾向于犯字体行间距太小的错误，所以你不确定的情况下加入额外的间距经常是个好主意。

在考虑行间距的时候，你也应该考虑你的项目的语言覆盖范围。如果你只使用无重音的字符来测试你字体的行间距，那么你设定的行间距很可能没有空间来放重音。如果你确定你的字体永远不会使用带重音的字符，那么这是可以接受的 — 但是你的字体被用来设置重音文本的几率还存在。在那种情况下，太小的行间距将会导致一行的重音进入上面字形的底部。使得读者难以阅读文本。

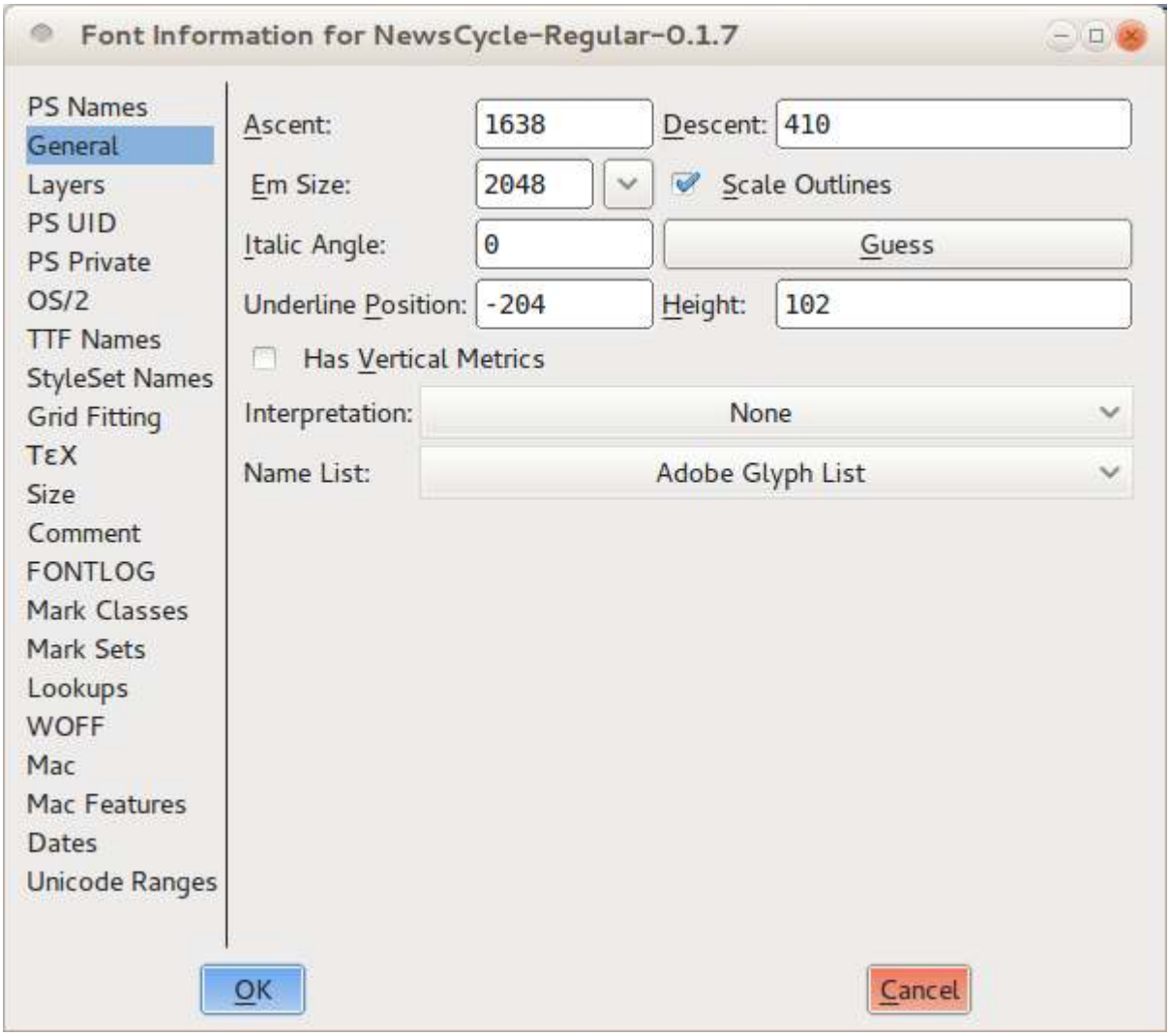
一个测试你的字体的行间距是否适合重音字符的方法是从多个语言中挑选示例文本。



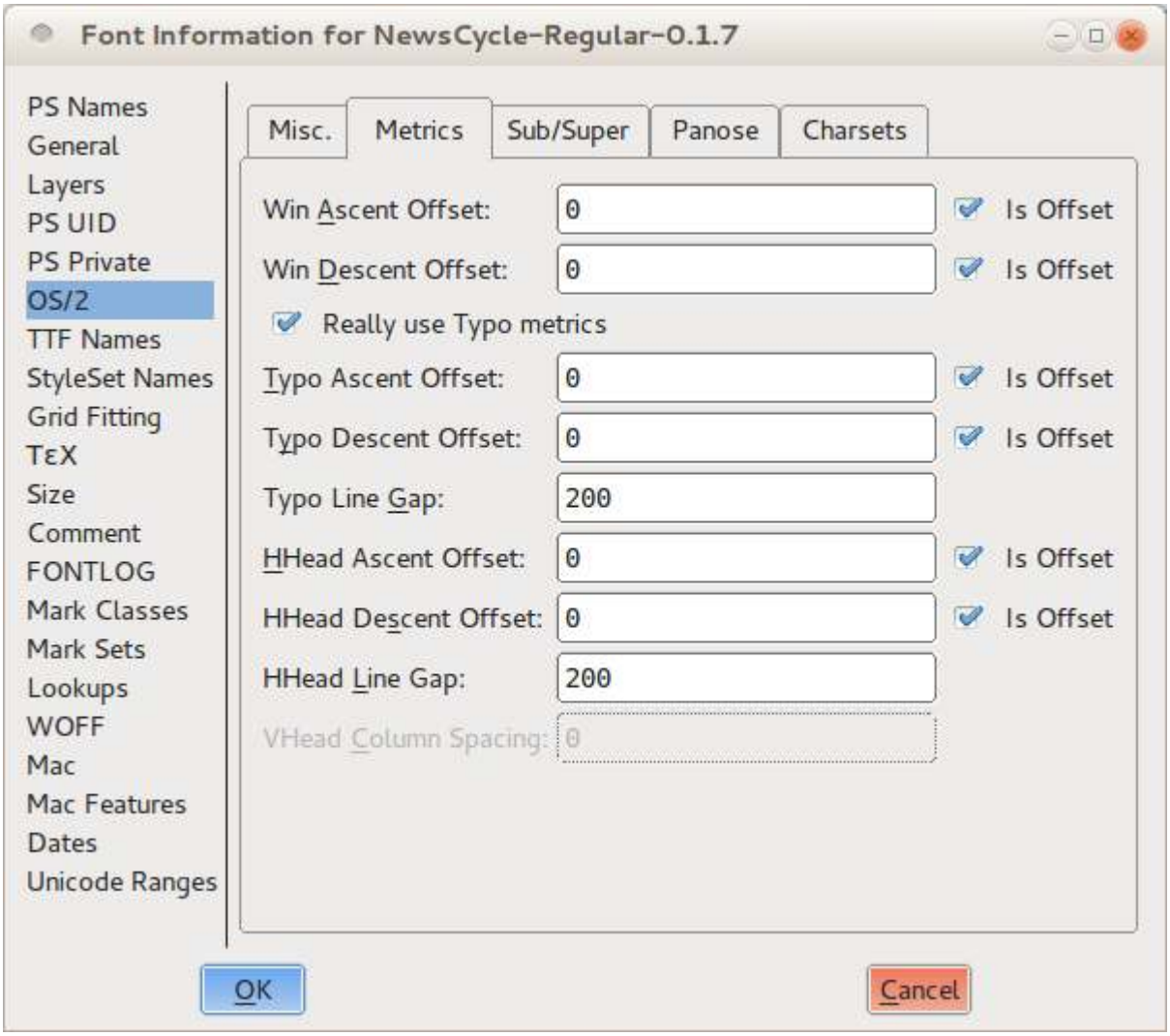
对于变音标记重的语言（比如捷克语），行间距应该比没使用变异符号的大。例子展示了相同的行间距的捷克语（上面的）和英语。

在FontForge中试验你字体的行间距

在FontForge中，你可以在字体信息窗口中设置调整你的字体项目的行间距。在“Element”菜单中选择*Font Info*，然后点击General选项卡可以打开这个窗口。需要注意FontForge列在顶部（Ascent）和底部（Descent）的数值。除非你已经手动改变了数值，否则这两个数字加起来等于Em size的值。



现在切换到“OS/2”选项卡。你的字体在几乎所有的电脑上的行间距将由你在Metrics选项卡输入的顶部和底部值决定。下一步你应该设置所有的



这里有三组值：Win Ascent与Descent，Typo Ascent与Descent和HHead Ascent与Descent。你应该将所有的Ascent设置与你在General选项卡下的值相同。接下来你应该将所有的Descent设置为与你在General选项卡下的值相同，除非你要设置Typo Descent数值为*负数*。 在这种情况下把数值设置为相同，但是在前面输入负号。最后取消对“is offset”选项的勾选。

这些设置将给你一个可感觉到的起点。现在你可以开始通过这一行间距来测试你的字体，并且增量调整直到你得到让眼睛舒服的结构。

如果你发现你的行间距太紧并且你不希望或不能让其字体的竖直距离更大，那么你可以将字形缩小来留给行间距更多空间。

标点和符号

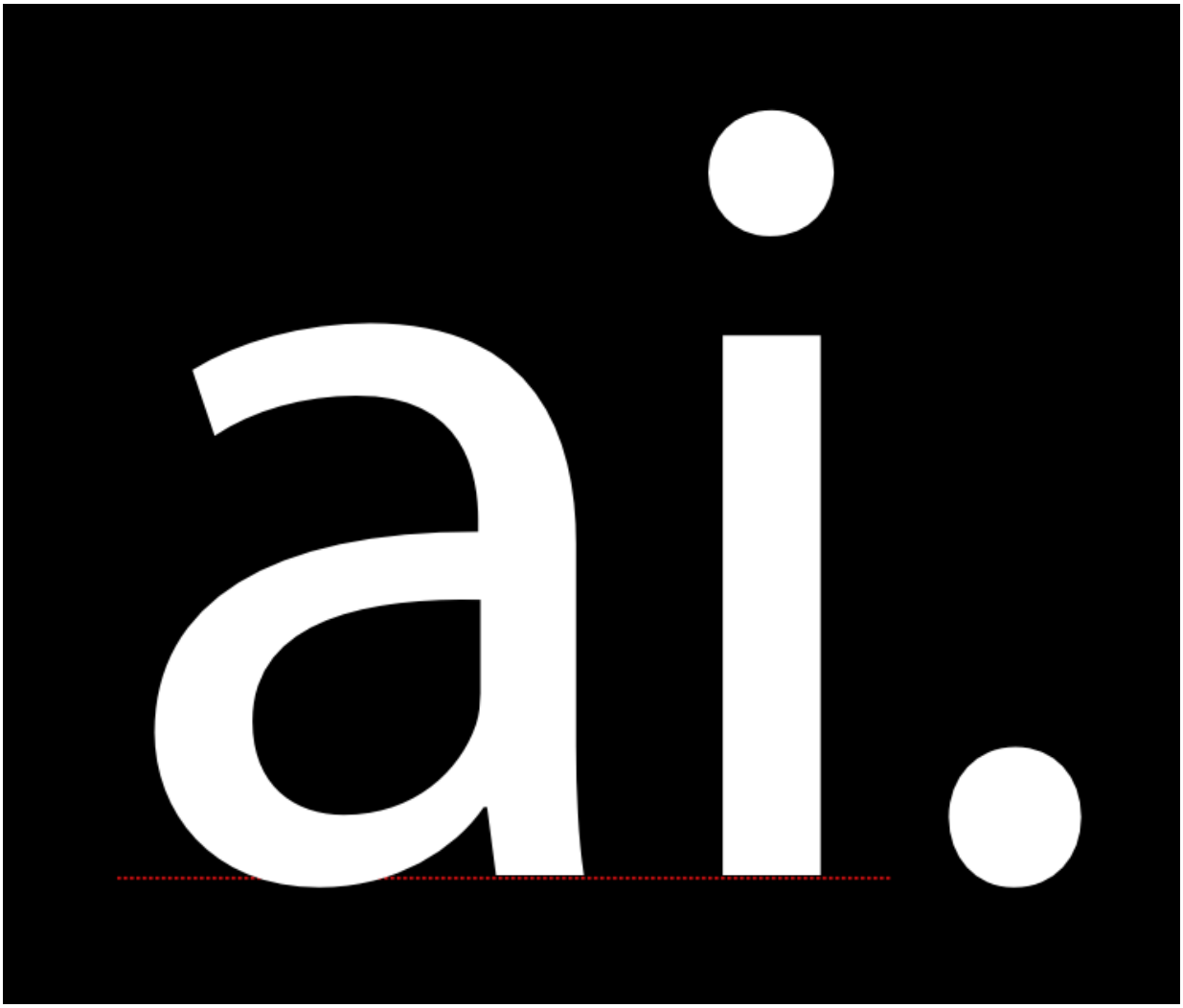
标点和其他印刷使用的符号都有他们自己的与字母的发展独立分开的历史。但是你会发现相同的设计流程仍然可以应用，包括重用和适配组件元素和迭代地测试你的设计选择。

简单标点字形

在设计标点时首先要做的事创造“.”符号，也就是句号或句点。

这个字形的形状经常来自于“i”上面的点，有时也被称作头部。在你复制这个点之后，你可能希望让它变得大一点，在打印文本或者屏幕上测试几种不同的尺寸是合适的。

一旦你建立了让你觉得快乐的尺寸，这个点会被作为多种其他标点的基础，包括这些字形：; : ? ! ¡ ¿ · …



下一个要制作的字形是逗号。逗号的形状可以变化到让你觉得吃惊的程度。在你设计自己的逗号前看一下各种各样的逗号设计是有价值的。

下面的图片展示了逗号可能使用的两种最常用的形式。

逗号的上部通常比句号小一点点，因为如果一样大小的话看起来太重。在同一张图片上的右边的逗号是应用这种补偿的例子。设计这个字形需要注意的另一种错误是做的太短。

当你有了自己的逗号时，制作分号（;）是非常容易的。

叹号和问号

叹号可能让你产生*看起来*容易制作的错觉。如果你看了一些字体样式，你会距的优势设计真的是相当简单。

但是这是一个有令人吃惊的机会总数可以表达设计的字形。即使在对比非常小的字体中，点上面的条形经常上大下小。叹号的形式经常与逗号的设计有着某种程度的联系。

问号也非常难制作，因为它需要你平衡点上面的开放曲线。

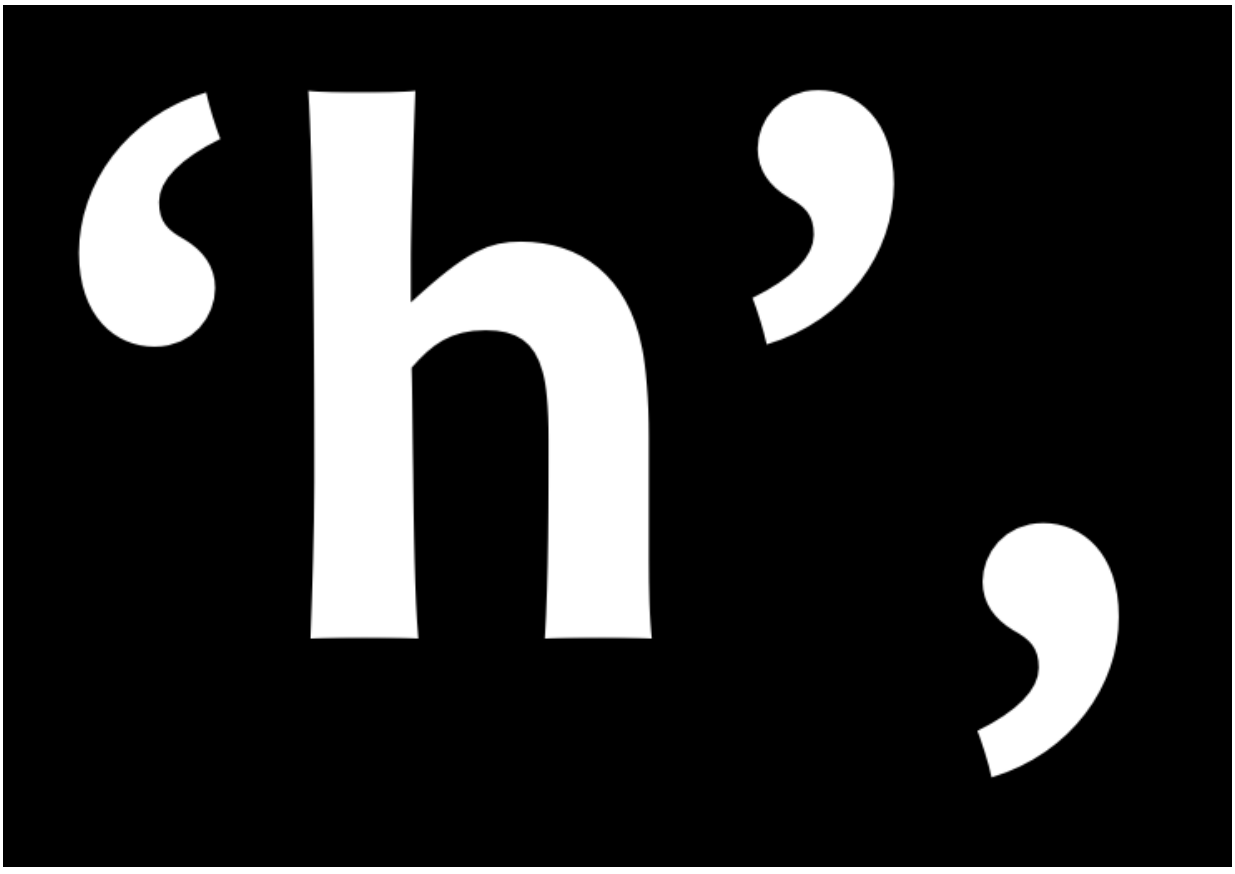
设计叹号时可取的方法是在为你的设计选择一个方案之前看甚至测试一些不同的方案。

字形c、C、G、s和s可能提供了设计这个字形的基础，但是你也可以选择一个不同的形式。

额外的符号

简单的或者竖直的引号 — '和" — 与印刷上的引号‘ ’和“ ” , „不同。

通常印刷上的引号与逗号非常接近，但是他们应该比逗号更长，并且经常更弯。



方括号[ ]制作起来相对简单，因为他们在形状上四四方方。不过他们的设计应该反映出你在其余字体样式上所作出的选择。

选择的主要问题是他们应该多高多深。约定是他们的高度应该轻微地超过大写字母的，并且底部比基线低约小写字母底深的3/4。

这些选择也会反映到圆括号()和花括号{}中。这三个符号的茎应该比大写和小写字母的茎的细。

圆括号应该在有关形状设计的基础上绘制，比如D、C和G。

花括号最明显的是他们的设计可以在非常大的程度上变化。花括号在这一点上与叹号相同。花括号的粗细分布可能类似数字的分布，因为它有时可能违反你在其余的设计中要遵循的规则。

完成小写字母

你可能在你之前见过的字体上注意到，虽然每个字母有其自己的形状，但是他们都互相有联系。通过解构一些字形，你能够得到几乎所有其他字母用来构建的积木。

注意看c和f的上部结束的相似性：

尽管他们巧妙地不同，但是他们的形状表明他们在相同的分组中。结束是字体的区别特征之一，通常在许多字母的形式中反复出现。

但是在模块上过多地依赖模块化来展示设计的标记应该避免 — 除非这是你想要的样子。

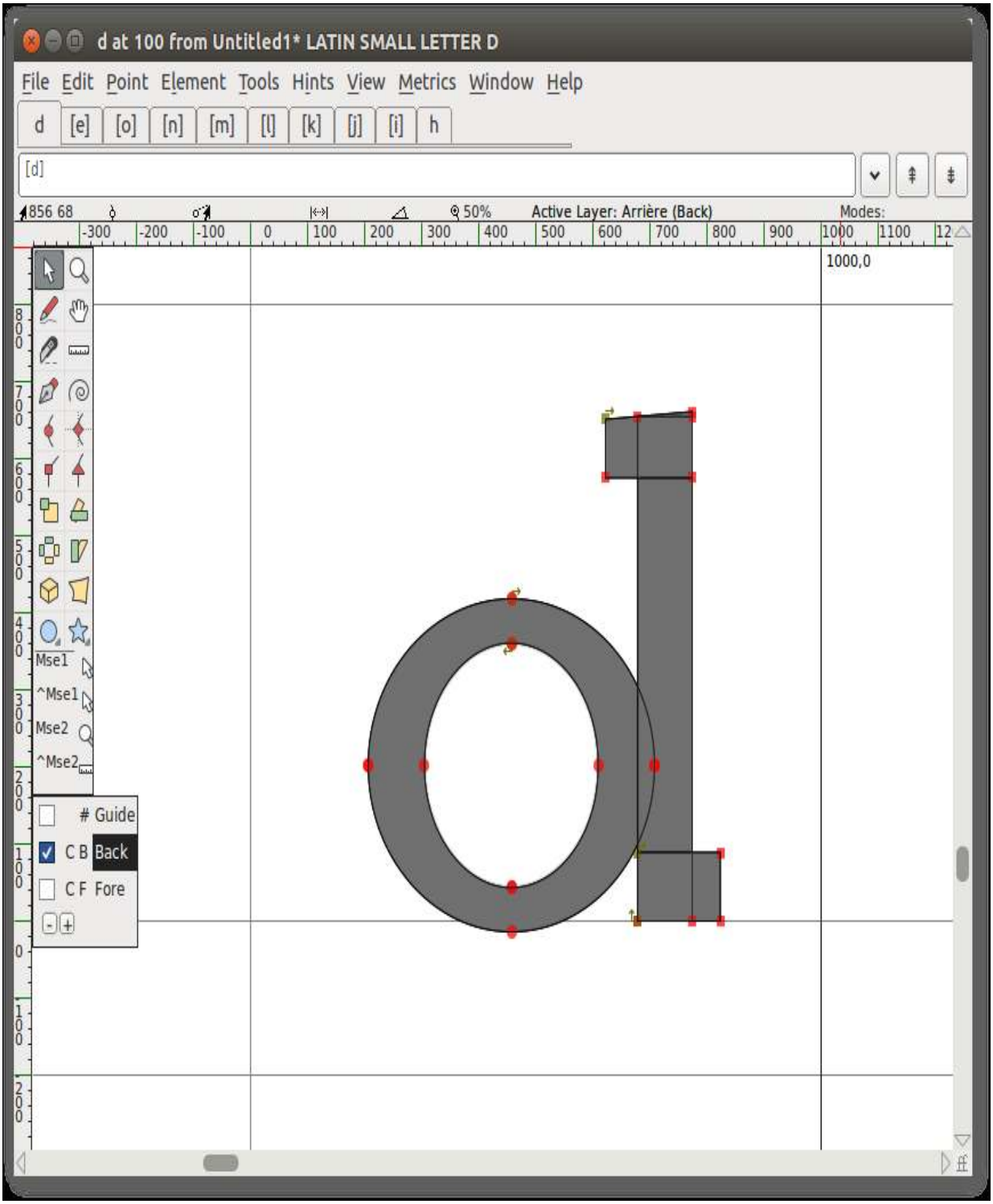
处理其他小写字母

你已经制作了字母“n”。从这个字母我们可以容易地分别通过克隆、拉伸和旋转导出m、h和u。m和u的茎的间距有微妙的改变。 u不仅改变了间距，也改变了衬线。这并不会自动发生；由你来到那推动这几点。

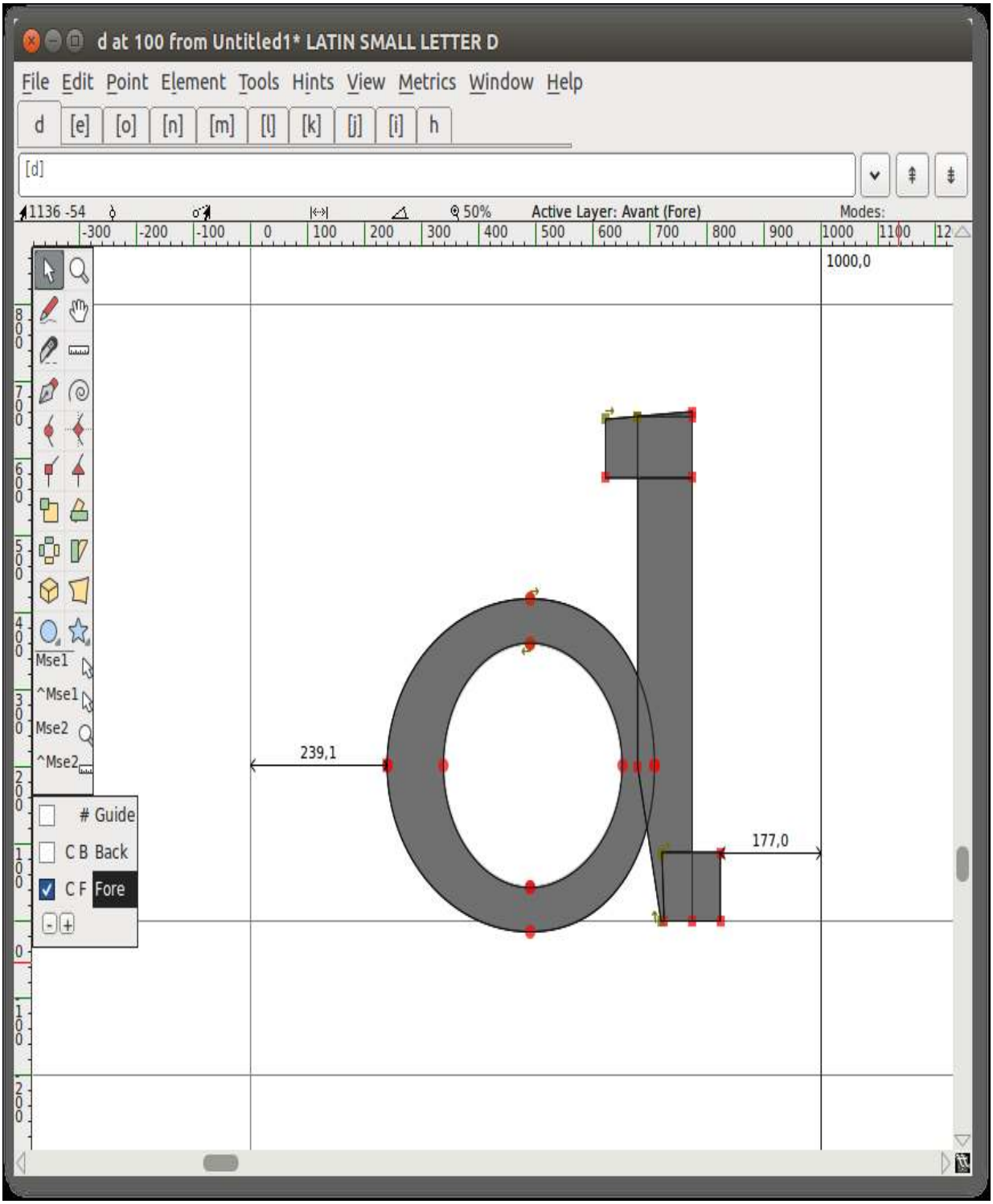
“i”可以由“n”的茎导出。“l”可以由“n”的茎通过一些调整作出。

由h和o的茎制作d

在字形窗口的字体视图中通过双击“d”打开字母“d”的字形窗口。在字体窗口中复制“o”并粘贴进字母“d”的字形窗口。然后为“h”做同样的事情。在这时你可以删除h中不准备使用的一部分。把剩下的部分摆放在一起，这样他们像一个d一样。

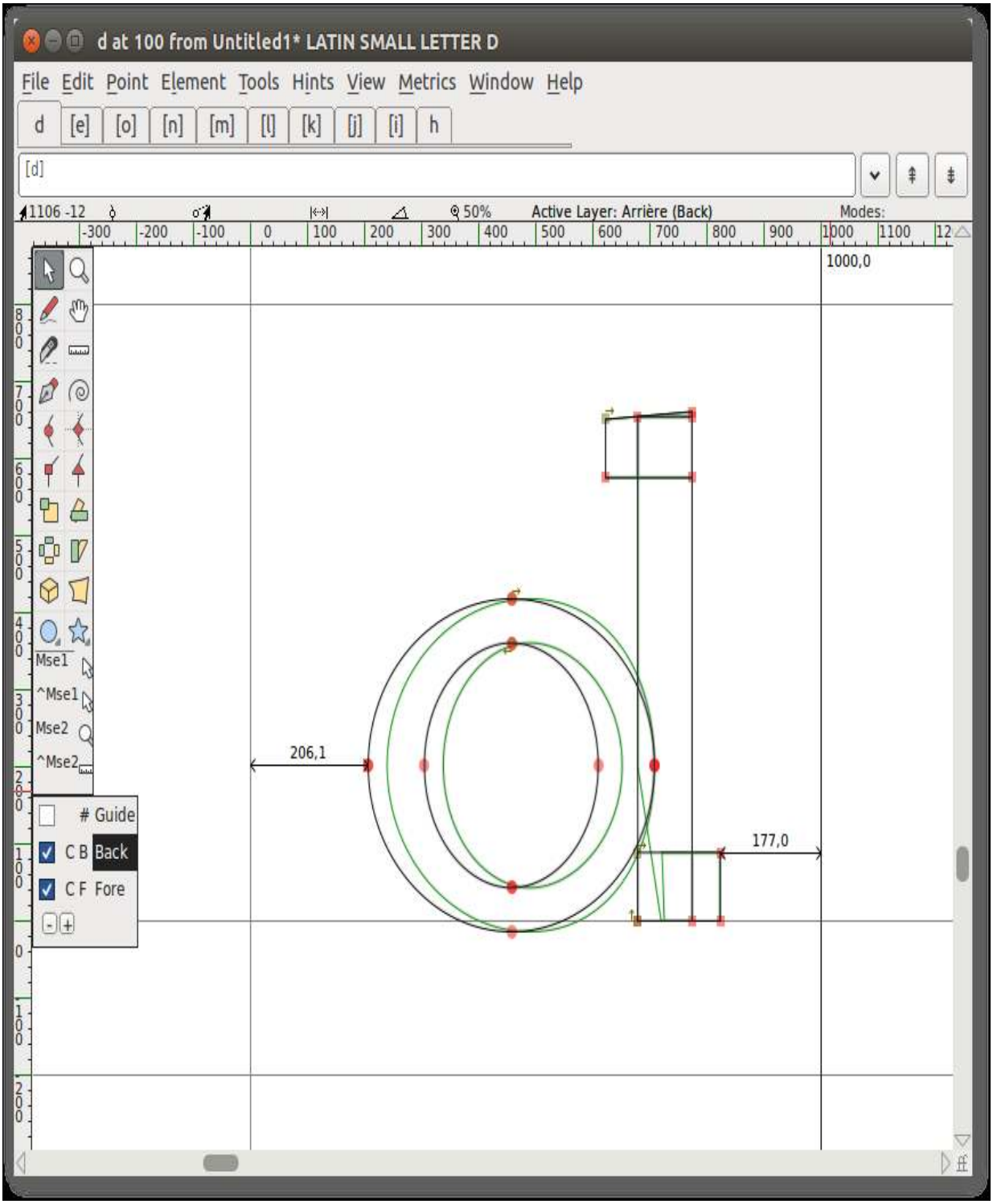


很明显这里还有一些工作需要做。我们将做出一些调整。让o的右边变窄，这样可以与茎匹配。



为了改善视觉间距并让图形看起来更平衡，通过在茎上添加一个点，让底部指向右边，在衬线上腾出地方来。

下面是最初的图形和新图形的覆盖图。



现在你知道该如何组装已有的部分，你可以制作其他类似的字母。要时时注意使得每个字母独特但仍处在一个字体家族中的微妙之处。

导出b、p和q

现在你可以通过翻转和旋转制作出合理的b、p和q。再次提醒注意每个字符的衬线和对比的如何不同。你的字体不必完全相同地做这件事，但是这是你应该思考的事情之一。

制作g

你可以从q开始，拉伸和修改尾部来制作单个弧线（bowl）的g。没有形状看起来接近双眼的g。双眼的g通常需要细一点，这样与其他字母放在一起的时候看起来没错。

朝f和t迈进

t有一个头部，但是通常比其他小写字母的头小。通过比较发现f比其旁边的字母高很多并且经常侵占其空间。他们都有横着交叉的条形，这些条形有相同的高度、宽度和厚度。你经常可以从一个复制到另一个。

现在制作e

e将会宽松地基于o。e的交叉条形比t的更低，但是有相同的厚度。e下部的钩子可以由t的底部提供。

从e到c

从e创造c可能需要删除交叉条形并在顶部添加结束。c的上部的结束可以与其他字母的上部结束类似，比如a、f和r。c的结束也可以形成s的基础。e也可以影响a的比例。

v，w，x，y和z

这些字母有一点难，因为他们没有与其他字母相关的形式。这意味着你不得不直截了当地绘制v。绘制下面的比划与你的其他字母的粗的茎一样粗，让你的上面的比划像你的其他字母的细的茎一样细。一旦你有了v，你就有了w和y的基本计划。对于x和y，在补偿斜线和交叉斜线的错觉时，聚焦在匹配设计的其他部分的对比。

变音和重音

变音是添加或结合到一个字母上的标记，经常用来字母的声调改变为添加的标记。一些变音标记（比如“acute”和“grave”）经常被称为重音。变音标记可能出现在一个字母的上面或者下面，一个字母内或两个字母之间。

à â ã ä ą ç ċ ǵ ő

一些变音的例子

小写的“a with grave”（unicode u+00e0）。在一个字体中通过将小写的“a”字形（unicode u+0061）和“combining grave accent”字形（unicode u+0300）联合起来创造。

小写的“a with circumflex”（unicode u+00e2）。在一个字体中通过将小写的“a”字形（unicode u+0061）和“combining circumflex accent”字形（unicode u+0302）联合起来创造。

小写的“a with ogonek”（unicode u+0105）。在一个字体中通过将小写的“a”字形（unicode u+0061）和“combining ogonek”字形（unicode u+0328）联合起来创造。

小写的“c with cedilla”（unicode u+00e7）。在一个字体中通过将小写的“c”字形（unicode u+0063）和“combining cedilla”字形（unicode u+0327）联合起来创造。

小写的“o with double acute”（unicode u+0151）。在一个字体中通过将小写的“o”字形（unicode u+006f）和“combining double acute accent”字形（unicode u+030b）联合起来创造。

FontForge可以用2种主要的方式自动地创造带重音的字符：

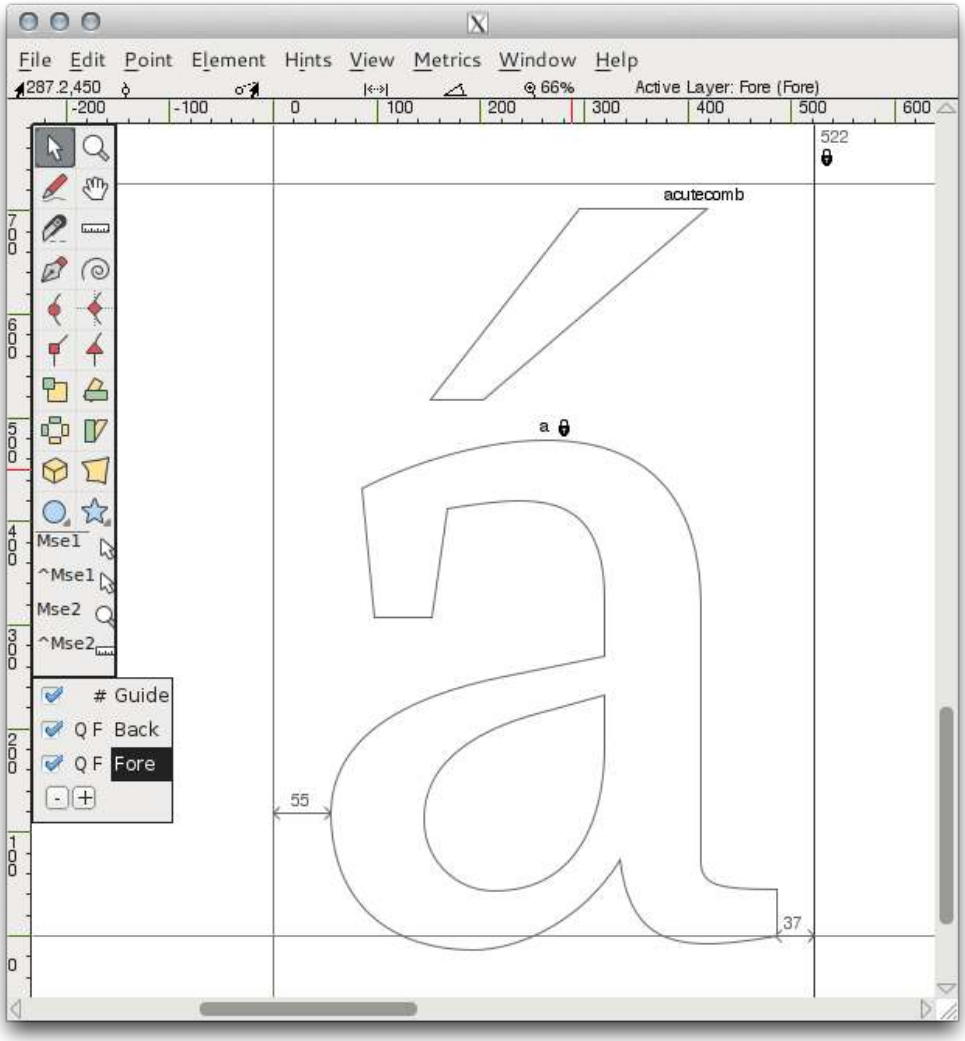
1. FontForge包含了放置变音标记位置的基本信息，所以可以自动地构建几乎所有的重音符号。
2. 为了对变音符号放置有更好的控制，FontForge可以给予用户创建的锚点的位置来放置变音标记。

注意：如果你没有使用锚点和查找表来控制变音标记的位置，那么如果一个特定的变音标记没有出现在你的字体中，FontForge会使用一个类似的间隔字符来替代。例如如果要联合的“acutecomb”（u+0301）没有出现在你的字体中，那么FontForge将在自动构建任何添加acute accent的字形时会使用标准的“acute”（u+00b4）字符。如果“acutecomb”出现了，那么FontForge将会总是使用它，除非你特别强制FontForge使用间隔字符来构建有重音的字形。

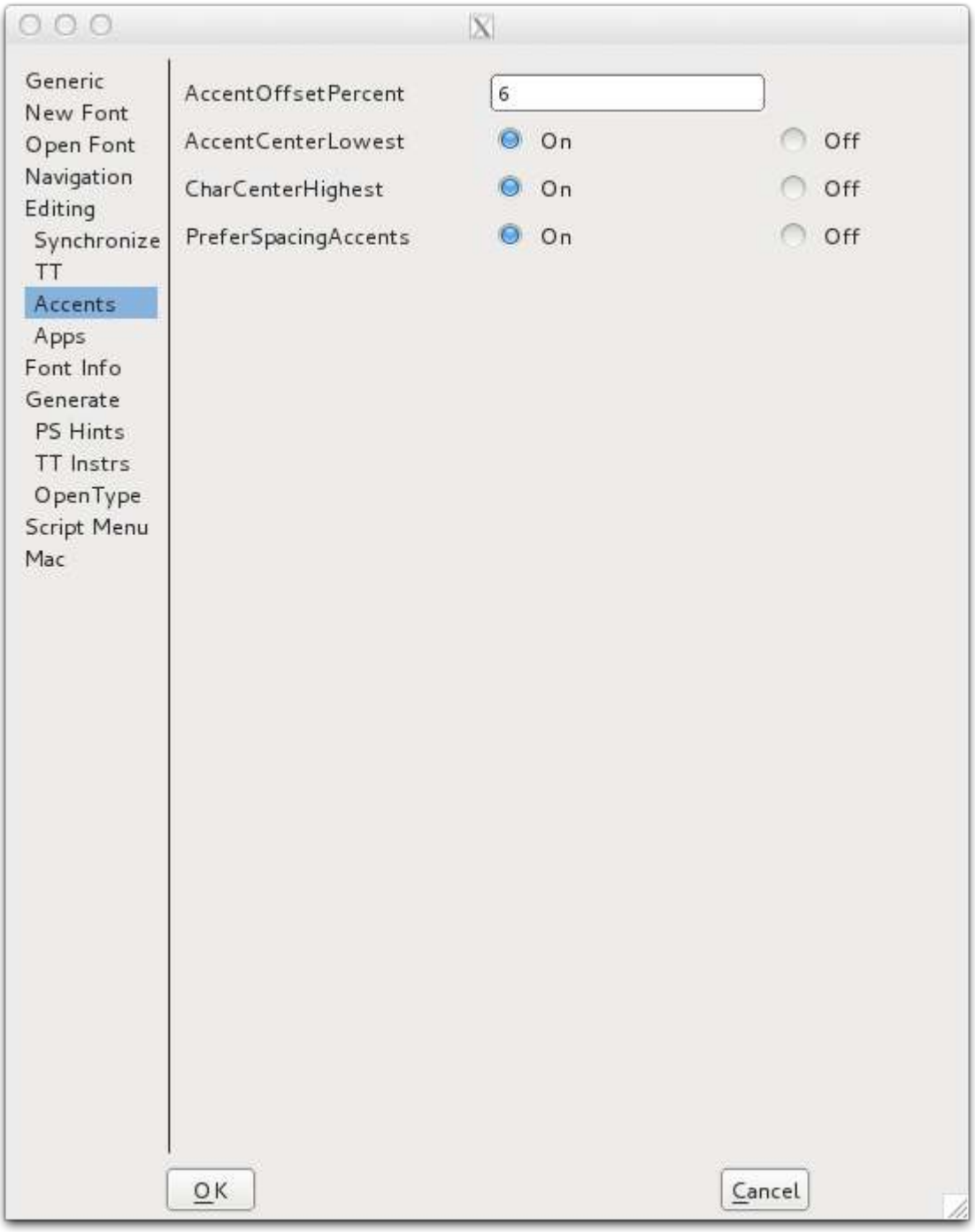
FontForge对变音标记的基本自动替换

在FontForge的“Element”菜单，有一个“Build”功能可以用来创建变音字符、某些组合字符和一些重复的字符。使用“Element > Build > Build Accented Glyph”功能可以在FontForge中自动构建带重音的字符。这个功能也可以通过快捷键`Ctrl + Shift + A`来使用。因此如果要使用“a acute“字符（u+00e1），我们需要已经创造出小写的“a”（u+0061）和“acutecomb”字形（u+0301）。然后选择“a acute”字符位置并使用“Element > Build > Build Accented Glyph”功能，FontForge将会放置一个到小写的“a”字形引用和一个到“acutecomb”字形的引用到“a acute”字符位置（如下）。

注意：如果一个变音标记字形没有出现在你的字体中，那么FontForge将会使用一个类似的间隔字符来替代。例如如果要联合的“acutecomb”（u+0301）没有出现在你的字体中，那么FontForge将在自动构建任何添加acute accent的字形时会使用标准的“acute”（u+00b4）字符。如果“acutecomb”出现了，那么FontForge将会总是使用它，除非你特别强制FontForge使用间隔字符来构建有重音的字形。



变音标记的自动放置可以通过偏好设置来调整，可以在FontForge的偏好设置菜单“File > Preferences > Accents”下的“accents”区域找到（如下）。



“PreferSpacingCharacters” - 选择这个选项为“On”将会强制FontForge使用间隔字符来构建重音字形，即使适当的联合字符出现。这个选项在使用锚点来放置变音标记时失效。

“AccentOffsetPercent”控制基本字形和标记字形的竖直间距的总量。这里输入的数值时字体的em square的百分比。所以如果数值是“6”将会使标记字形到基本字形的距离将是字体的em square的百分之六。

标记字形水平放置的偏好设置页可以设置。在“AccentCenterLowest”的偏好设置选择“On”将会使重音字形处于基本字形的最低点的中央。

在“AccentCenterHighest”的偏好设置选择“On”将会使重音字形处于基本字形的最高点的中央。

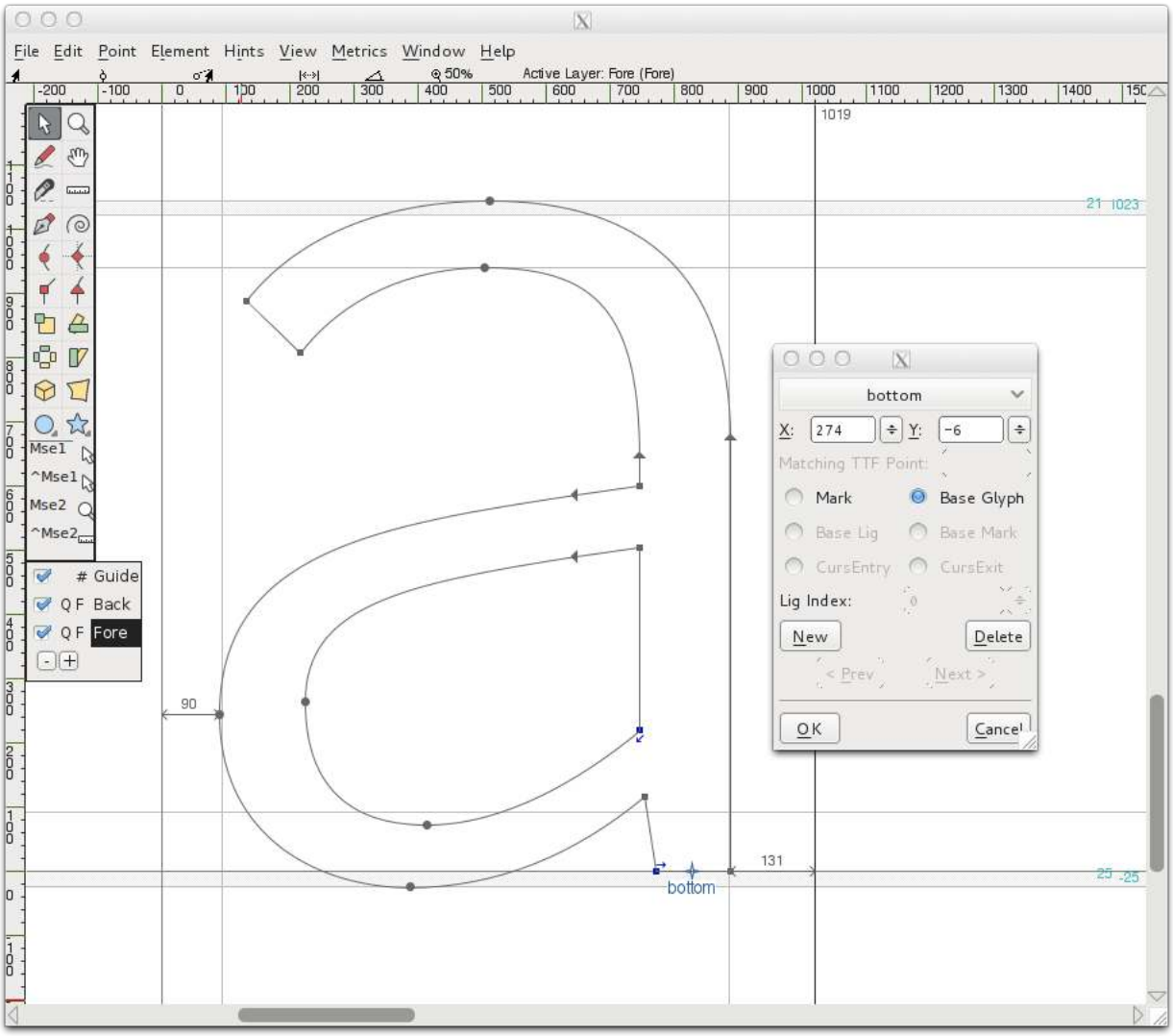
上面两个偏好设置都选择“Off”将会使重音放置在基本字形的宽度中央。上面两个偏好设置都选择“On”将会使重音放置到字符空间的中央。

使用锚点来放置变音

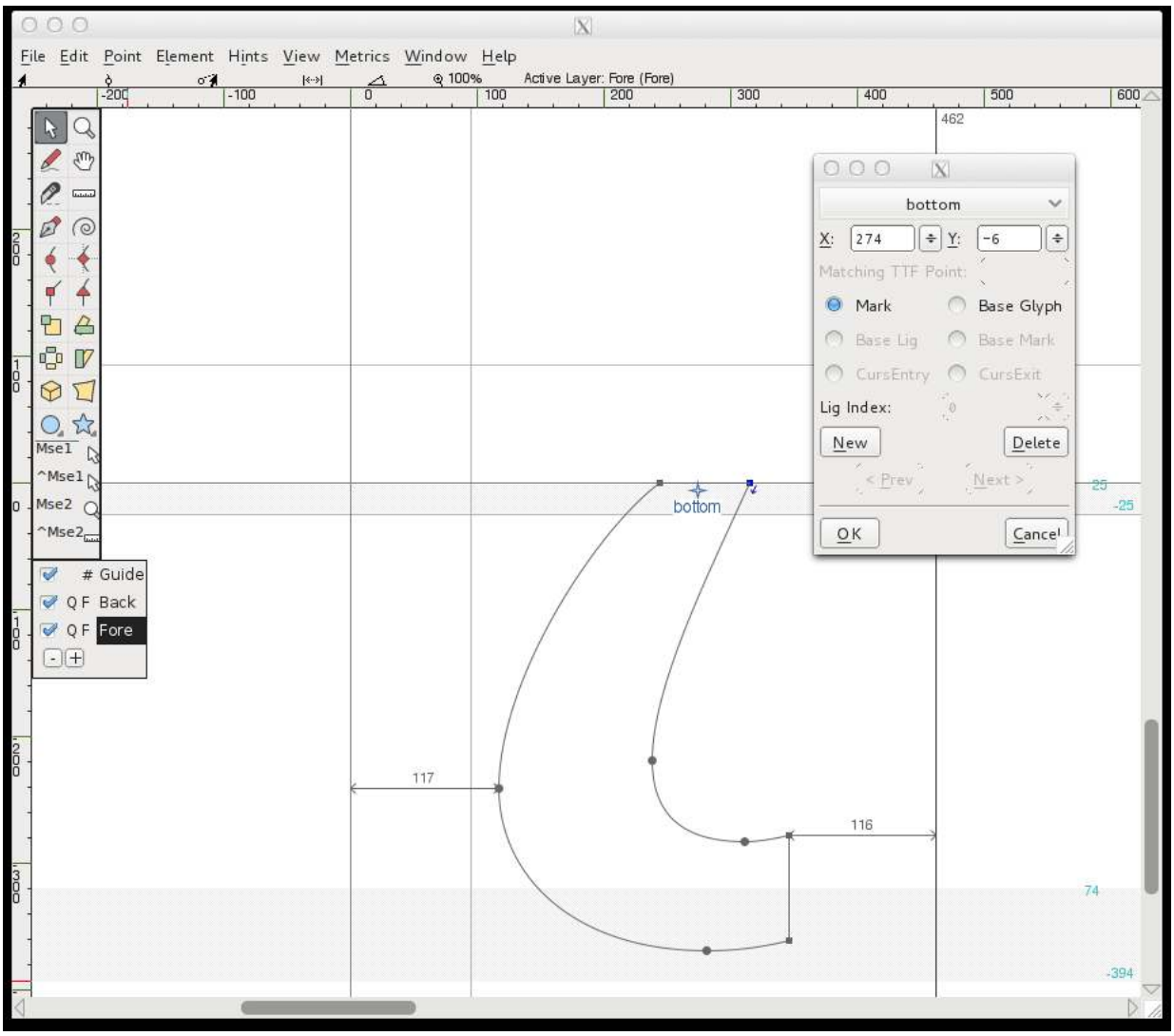
在FontForge中构建重音字符的最精确有效的方式是使用“锚点”。

锚点允许你精确地控制变音标记在有重音的字符上相对于每个基本字形的位置。因此在字符“a ogonek”中，“a”字形时基本字形并会放置在正常位置，“ogonek”字形是“标记字形”并会放置在某个位置使得“标记字形”的锚点与基本字形的锚点一致。

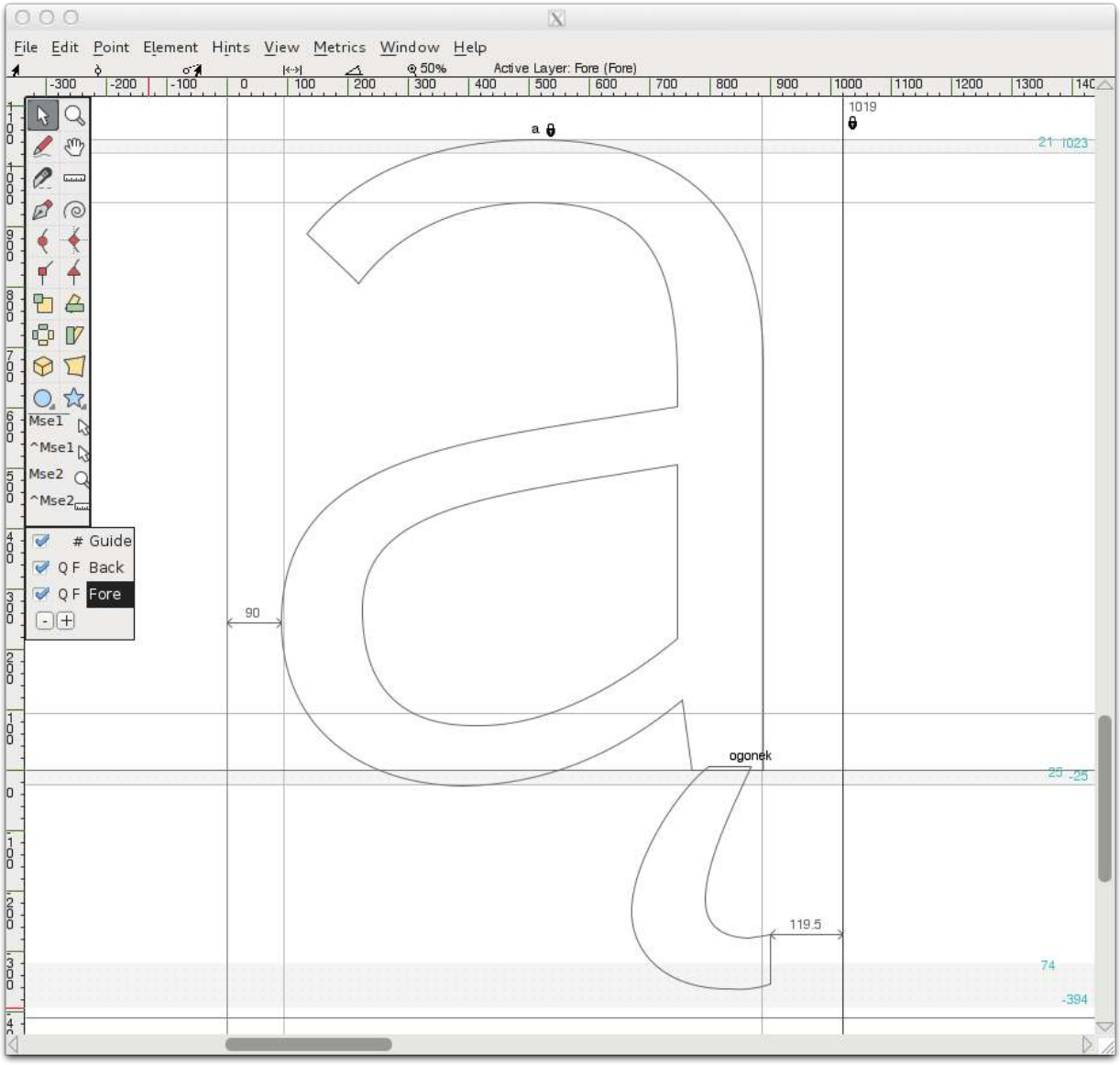
在下面的例子中，创造一个“a ogonek”字符，创建了一个称作“底部”的锚类。在小写的“a”字形中，“底部”锚点放置在“a”的茎的底部。这是“基本字形”的锚点的形式。



在“ogonek”字形中，标记锚点的形式中的底部锚点放置在字形的顶部（如下）。



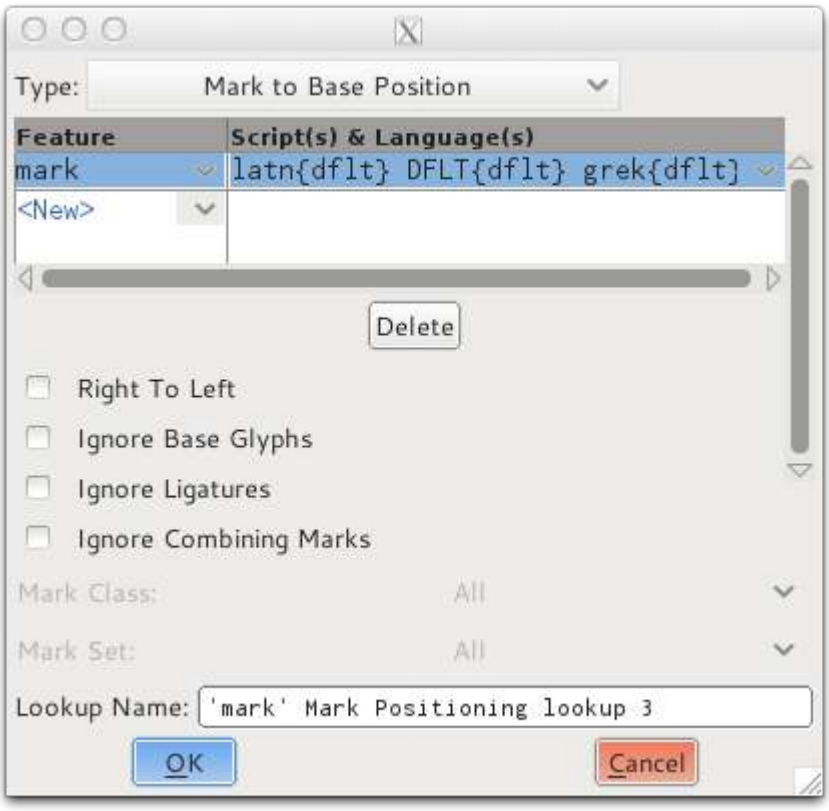
那么在“a ogonek”字符构建时（使用“Build Accented Charcter”功能），“底部”标记锚点将会放置在与“底部”基本锚点相同的位置，保证ogonek字形的引用正确地放置在“a”的引用的茎的尾部（如下）。这一精确自动的位置只有在使用锚点来放置基本和标记字形的时候才可以使用。



为放置变音标记创建锚点（到基本位置的标记）

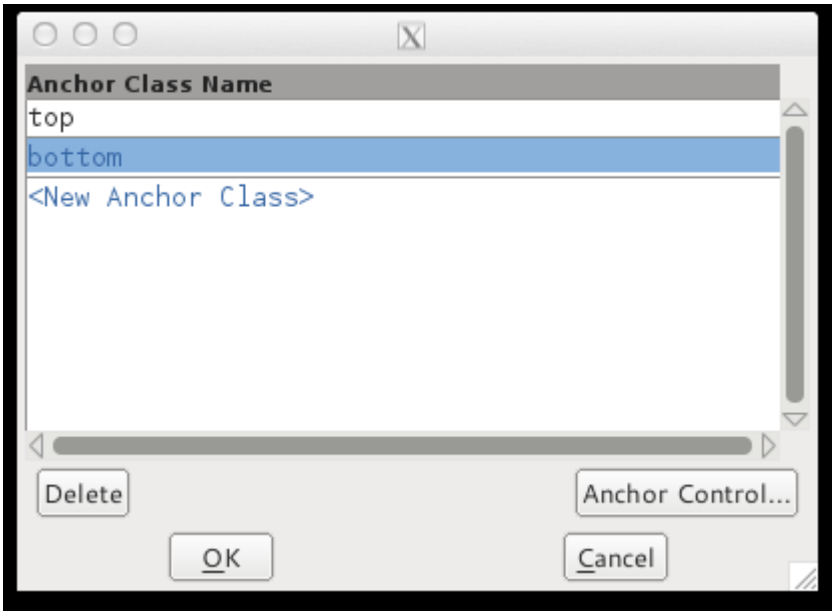
FontForge使用被称为“mark-to-base”的查找功能来创建和放置锚点。这些mark-to-base查找可以通过你的字体的Font Info的GPOS Lookups区域来创建和编辑（“Element>Font Info>Lookups>GPOS”）。

在GPOS Lookups窗口中，点击“Add Lookup”并选择类型“Mark to Base Position”，然后在Feature窗格的“New”一列选择“Mark Positioning”（如下）。点击“OK”关闭窗口。



选中新的查找，点击“Add Subtable”。在结果窗口（如下）你可以创建你的锚类。

在这个例子中（如下），创建了两个锚类“top”和“bottom”。“top”锚类将会用来放置字形上面的变音标记，“bottom”类将会用来放置字形下面的标记。

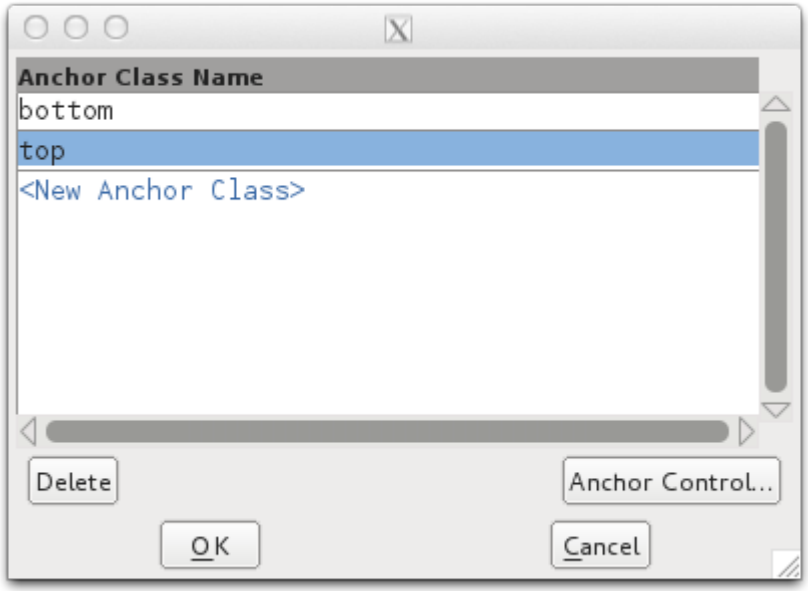


要在一个字形上放置锚点，只需要在字形编辑窗口鼠标右击，在右键菜单中选择“Add Anchor”功能。出现的对话框让你可以设置锚点是基本锚点还是标记锚点。锚点的位置也可以通过这个对话框来调整。锚点也可以通过鼠标拖拽或方向键来移动到你想要的位置。右击锚点选择右键菜单中的“get info”也可以修改锚点。

锚类的控制

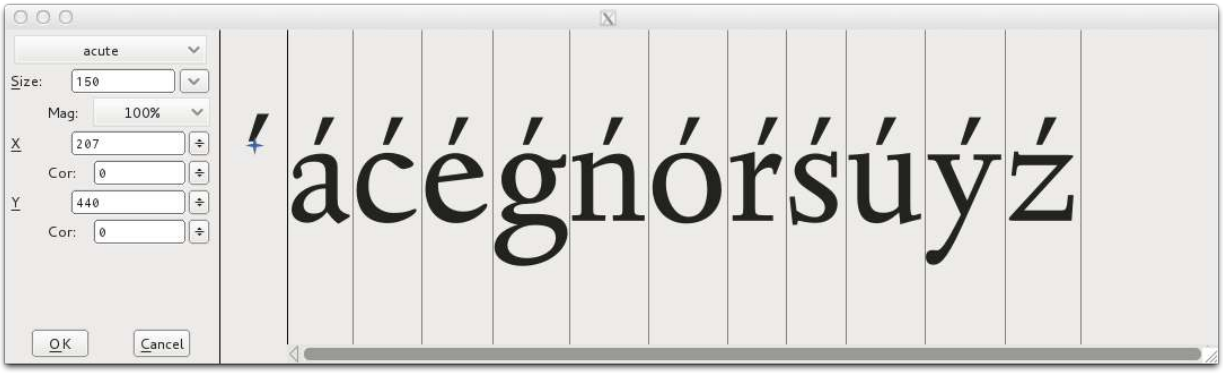
FontForge也包含了一个有用的图形界面来控制锚点的所有类的位置，使用户可以很好地调整位置，例如一次性调整位置字体中所有的尖重音（acute accent），或者调整一个类别中包含的所有锚点，例如引用小写字母“e”的字符。下面的例子中我们可以看到如何使用这个图形界面来良好地调整一个字体中所有尖重音的位置，调整所有引用小写字母“e”字形的字符的锚类别。

一旦你在你的mark-to-base位置查找中创建了锚类别，并为一些字形添加了锚点，你可以通过“Element>Font Info>Lookups>GPOS”来控制这些类别，然后编辑包含锚类别的子表。你将会看到下面的窗口：



在这里我们可以选择你希望编辑的类别病点击“Anchor Control”按钮。这个类别的一个图形界面将会展现在你面前。在下面的例子中我们在编辑“top”类别的控制。在第一个例子中（如下），在“Bases”区域的下拉菜单中选中选中小写字母“e”。当一个基本字形被选中，所有引用这个字形并包含“top”基本锚点的字符将会显示在预览窗格。然后我们可以调整“top”基本锚点的位置来看它如何影响所有包含“top”标记锚点的字形的位置。

在第二个例子中，如下，在“Makrs”区域的下拉菜单中选中“acute”字形。当一个标记字形选中时，所有引用选中字形并包含“top”标记锚点的字形将会显示预览效果。



其他资源

- http://urtd.net/projects/cod/about
- http://ilovetypography.com/2009/01/24/on-diacritics/
- http://diacritics.typo.cz/
- http://scripts.sil.org/ProbsOfDiacDesign
- http://www.microsoft.com/typography/developers/fdsspec/diacritics.htm

数字

对于字体设计者来说数字经常是困难的 — 原因有几点。一个原因是数字有大量的曲线。另一个原因是数字经常在他们的形状中使用的约定与在字体设计的其他部分中可见的约定不同（甚至是违背）。此外，数字可能有非常大量的笔画（像8和5那样），或者他们有大片的空白空间（像1，7，有时包括2和4）。两种情况都可能难以处理。最后还有一个问题，如何让零看起来与大写字母O不同。

看各种各样字体中的数字从而对设计者们应对这些问题的的方式更加熟悉，这样做是有帮助的。

在笔画密集的数字（比如8）中，你可能发现设计者们允许笔画宽度变得比字体中的典型字母更细。类似的方法也可以在双眼的g种看到。

相反，为了补偿空白空间比例大的数字，一些笔画很可能比典型的更重。

区分零和大写字母O有广泛的解决方案 — 比如将零做得比O窄，或者零做成完美的圆，或者可能有斜线穿过零（尤其是monospace字体）。

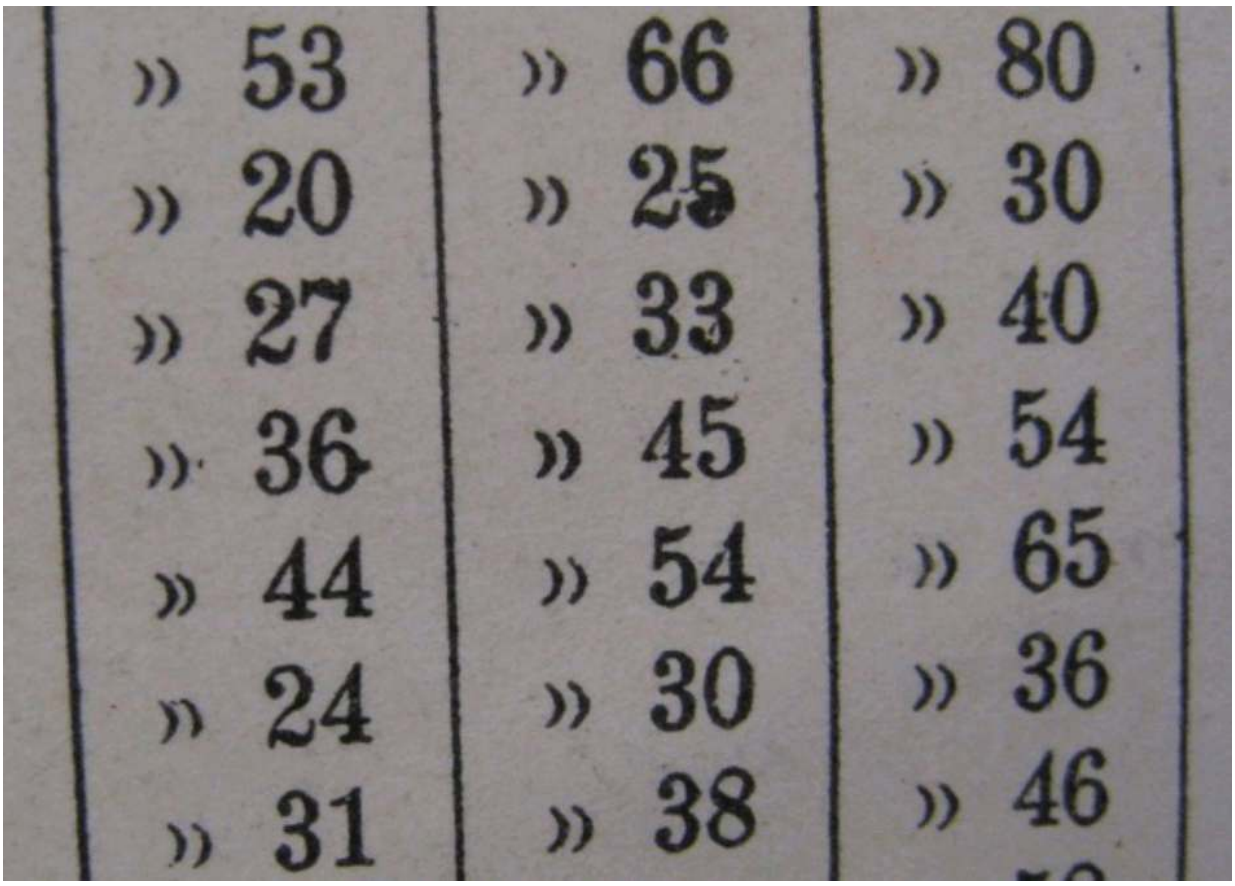
零比大写的O更窄但是高度相同是通用的方法。这是衬里数字的典型方法。衬里数字（Lining Numberal）是数字最通用的样式。使用这种方法的字体包括：许多Garamonds、Futura和Google的网页字体Open Sans。下面是Open Sans字体展示的零、大写字母O，然后是其他数字。

一个完美的圆或者接近完美的圆圈不常用，但是也存在。使用这种方法的字体的例子包括Google网页字体Vollkorn和其他商业字体比如Mrs Eaves，Vendeta和Fleischman BT Pro。使用老式的按比例的数字的字体更多地使用了这种方法。有时也会看到一个达到x高度但是更窄的零。

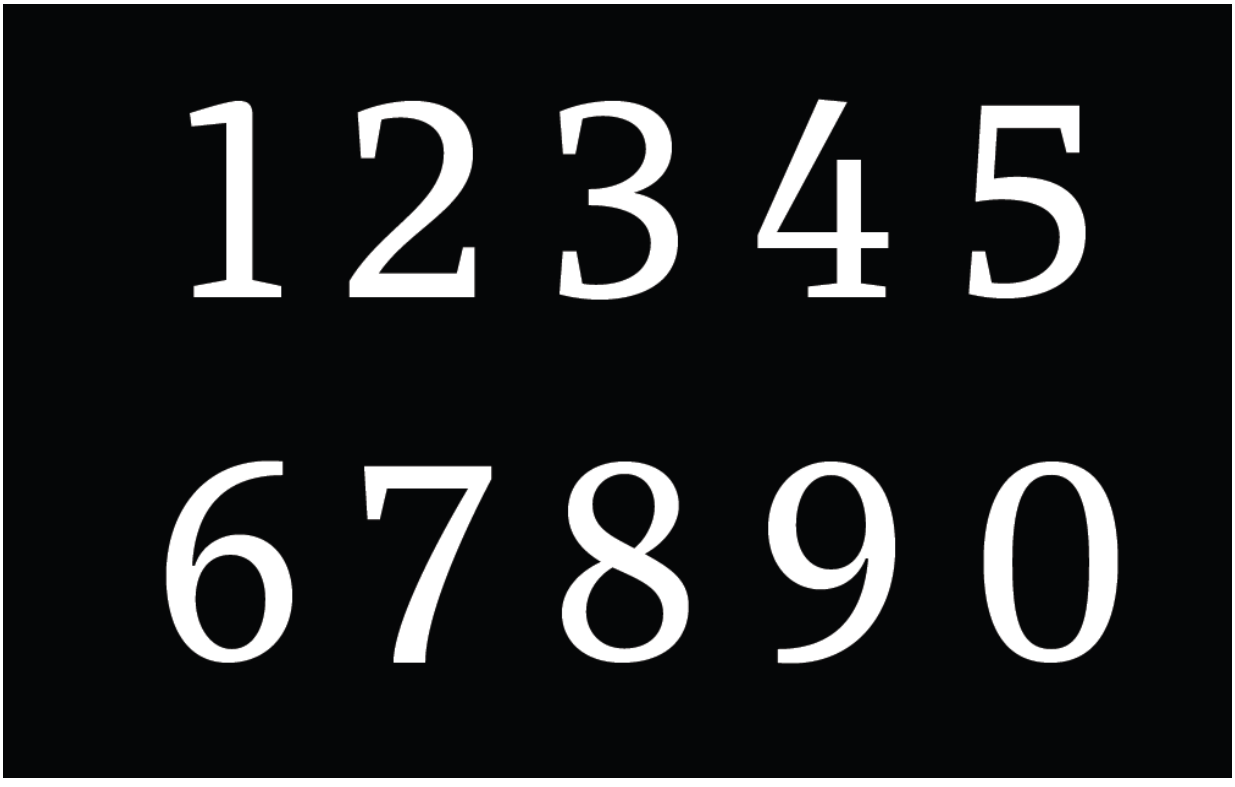
当你包含了分数、上标和下标的时候，数字可能有多达11种可以分辨的样式。我们来看其中5种最常见的。

衬里样式数字

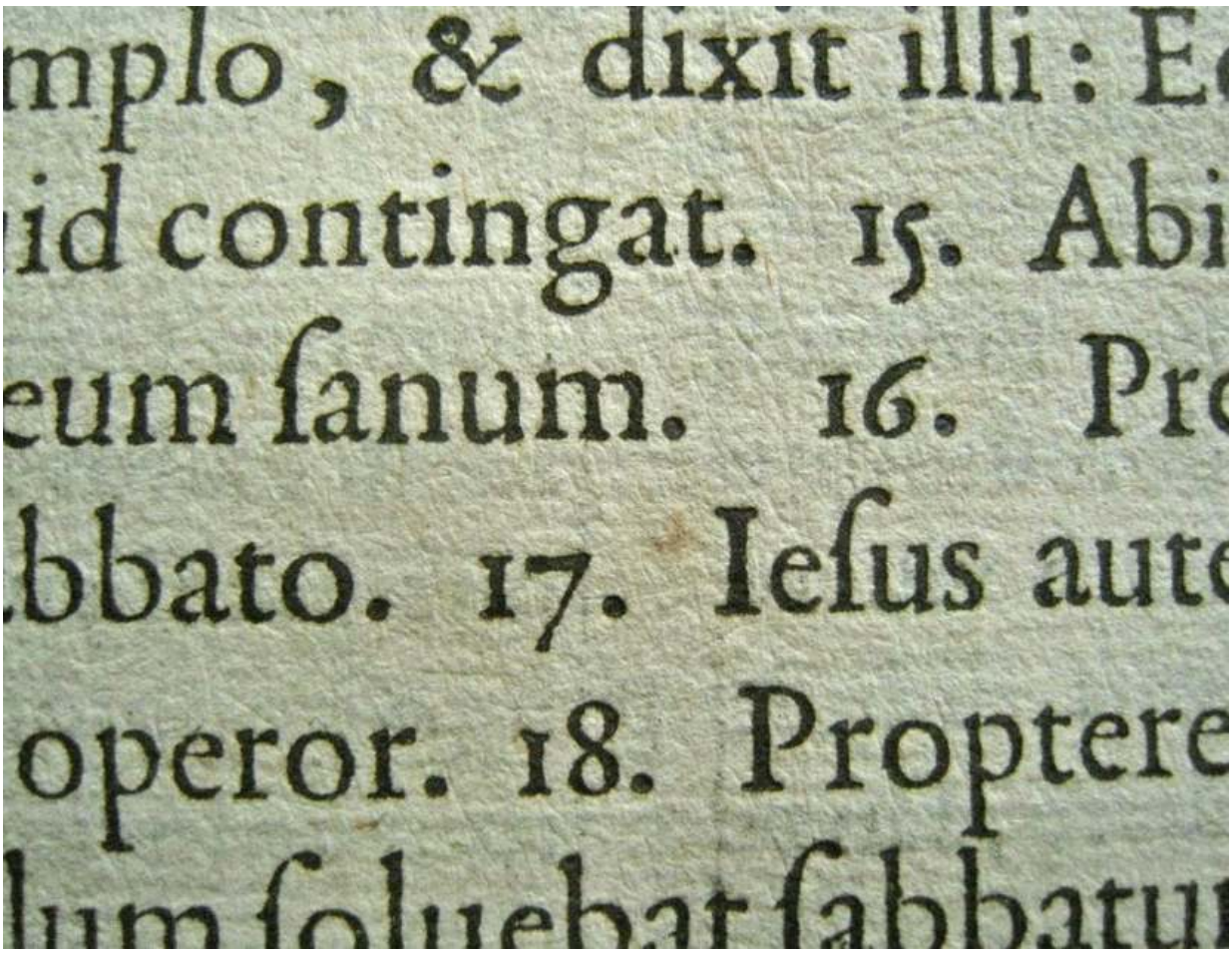
字体中见到的最常用的样式是扁平衬里（Tabular Lining）和比例衬里（Proportional Lining）。衬里指的是数字使用的高度。衬里样式指的是所有数字的高度视觉上一致。扁平衬里和比例衬里数字的区别是扁平衬里中所有的数字共享宽度。这个样式的用处在于电子扁平和其他数字堆叠在整齐的水平和竖直线上有好处的地方。



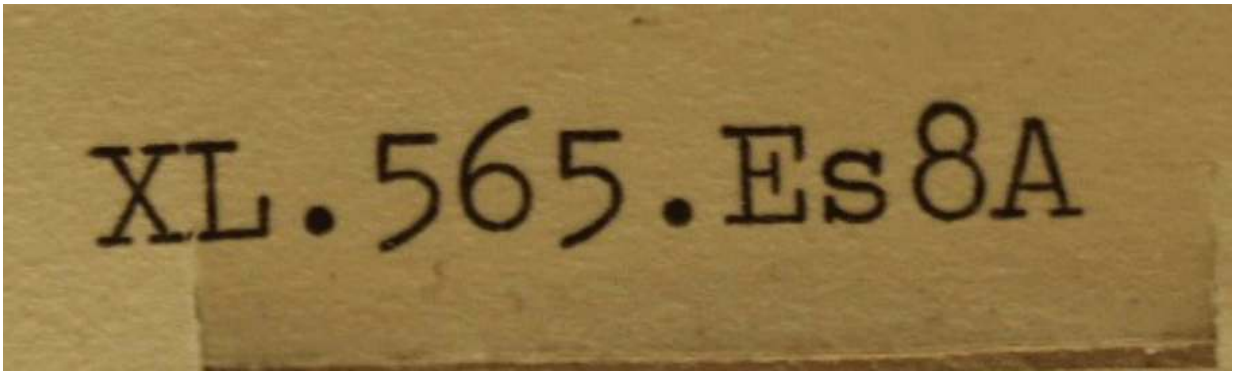
比例衬里数字的好处在于其看起来更相等，因为数字的形式和间距可以补偿不同的笔画密度。



排列或者老式的数字



扁平数字是历史上的术语中相对较新的发明。在其存在前有许多老式的比例数字。如果你希望数字和文字混合并使用文字的样式，那么老式的数字是有帮助的。



扁平老式数字是非常少见的。它在年报中有用，在年报中你想要老式数字的感觉，扁平间距是这种类型文件的典型。上面的图片来自打印机制作的图书馆分类卡。

混合类型数字

混合（hybrid）数字并不使用字体的大写高度和x高度，而是使用它们自己的高度。术语“混合”指的是老式与衬里样式的数字。使用混合类型数字的字体包括Georgia和Google网页字体Merriweather和Donegal。Merriweather字体的零、大写字母O、0、1、2、3等字形展示在下面。

0O0123456789

粗体

当我们谈论“粗体”时，我们实际谈论的是更加广泛的变量，也就是粗细。粗细可以包含从非常非常细的细线字母到非常粗的字母。这个变量用于在文字印刷上为正文创建强烈的间隔，在图形设计中引起对词或短文本的注意，或者为文本加上特殊的感觉（而不是为了与其他文本的对比）。

你可能希望做广泛的关于粗细的事情，很可能你调整粗细的初体验是尝试伴随你的常规字体的粗体。

由于你使用FontForge，你有一个明显的优势。不像许多字体编辑程序，你从FontForge样式筛选得到的结果实际上可能很适合使用 — 你在商业字体设计软件得到的更是如此。这是由于其使用的算法异常复杂。

创造一个字体的粗体版本可以通过运行一个称为*Change weight*的筛选（你可以在Element > Styles菜单找到）快速地近似地加重你的字形。

这个自动性质和高速的进程使得它非常适合测试你的粗体想要多粗的。你可能想要尝试运行几次这个筛选并保存几个版本在文本上与你的常规字体做比较。这意味着你可能仍然需要在运行筛选后进一步修改结果，或者为了得到令人满意的结果而手动地调整独立的字形。

还需要记住的是有笔画密度的字形（比如1，i，l，I，L，j和J）可能需要更粗，而没有笔画密度的字形（比如a，e，g，x，B，R，8和&）将需要比其他字形更细。

字体插值

FontForge有一个功能可以在分开的字体中做插值（在Element菜单中可以看到*Interpolate Fonts*功能）。字体插值技术可以用来创建两种字体粗细的中间粗细程度。因此选择你的粗体的粗细的一种方法是创建一个比你肯定比你所需更粗的粗体，然后在这个过粗的设计和你的常规字体之间插入几个不同的粗细程度。

使用这一技术，你可以更加快速地找到你觉得你的项目更加适合的粗细程度。同样的技术甚至也可以用于帮助选择更粗的字体，比如“重体”和“黑体”，更细的如“书体”和“细体”也一样。

依照这种逻辑，可能看起来最好最有效的制作常规和其他所需粗细字体的方式是制作一个非常细和一个超粗的字体，然后从这些字体生成你需要的所有字体。但是这个方法的结果过分乏味。取而代之的是，每个粗细上的重大改变将会需要其总体设计，其他中等粗细程度可以由总体设计来制作。

斜体

斜体很可能是字体设计中最容易误解的样式，但是它也有最大的潜在兴奋和乐趣，这是由于作为设计者的你有大量的变量可以玩。

斜体与粗体不同，这是由于斜体并不意味着与常规体有不同的粗细。相反他们意味着提供与常规体不同的*纹理*。在这点不同上更大的强度意味着斜体尤其对创造与常规体的对比上有用。这一强烈影响对突出文本中单个词或短的段落有用。相反不同更小的纹理经常在你设置多条线、整个段落或整页为斜体的情况下有用。

倾斜

与斜体相关的最常见的变量是倾斜。事实上当网页浏览器在一个CSS规则中请求一个斜体但是不存在时，它将会简单地倾斜常规体来合成或仿造斜体。或许当人们第一次设计字体的时候会考虑这种方法，这或许并不令人惊奇。这一方法的来源要追溯到20世纪中叶现代主义应用到设计的时候。这是字体样式中最早的斜体比如Helvetica也是常规体的倾斜版本的原因。

倾斜多少?

一些斜体并不倾斜。是这样的，真的！这些斜体被称为垂直斜体。但是很可能如果你的设计中只有一个斜体，你将选择那个斜体来获得一定程度的倾斜。通常斜体倾向于倾斜4-14度。大多数当代字体倾斜6-9度。

倾斜对一个斜体的设计可以是重要的，因此这很容易注意到甚至通过使用自动筛选的有限成功来实现。这不是你可以用来帮助使斜体与常规体分开的唯一变量。除了倾斜之外你可能想要考虑使用一个或多个下面的变量。

斜体构造

术语“斜体”（Italic也称意大利斜体）实际上指的并不是在许多斜体设计中看到的倾斜，而是在14世纪的意大利开始流行的手写体。这种样式是快速连接的手写形式，其字母使用了与常规体字母不同的构造。字体设计者们说他们设计了一个“真正”的斜体时指的就是这种不同的笔画构造或模式。这个构造有许多你可能选择在斜体设计中包含的子特征。

三角形对立面

这些特征中最显而易见的就是通过连接的字母创造出的三角形对立面。这些字母包括a，b，d，g，h，m，n，p，q和u。这一变量是强有力的，部分原因是对立面是一个强有力的变量，但是也由于大量有这一特征的数字。那个事实，因其在大部分语言中高频出现，也是一个非常大的因素。

当你设计斜体时，你可以通过相对小地调整连接的高度来非常高效地调整你的斜体的效果。微妙的改变可以带来令人惊奇的大的结果。尽管如此，并不是所有的斜体字体利用了这一变量。

向外和向内的笔画

许多斜体在向外和向内的笔画上使用了不对称的衬线。当使用了其中之一时，会更普遍地在向外笔画上使用，并在向内笔画存在的地方应用垂直样式。向内笔画和向外笔画的效果的强度可以通过调整笔画粗细和长度来控制。像三角形对立面，他们的效用和力量很大部分来自于许多字母使用它们这一事实。

缩合

通常斜体比常规体更窄或者紧缩几分。因为缩合是出现在斜体的所有字母上的特征，是一个非常强有力的变量。这个变量可以通过粗略或者精细的方式使用。如果你选择使用这个变量，那么有必要调整笔画的粗细来使得斜体看起来相同，或者与常规体设计粗细接近。你的斜体越缩合，你越需要做更多这样的调整。

混合变量

大多数斜体以多样的比例使用上面列出的全部变量。你可能会发现看一看广泛的斜体设计并分析他们的变量及多少是很有用的。当你这么做时，你将会注意到变量中的任何一个都不会用到最强。相反其中一个趋向于领先，其他有限地使用。变量使用地越强，你的斜体相对于常规体对比越强。

在过去的十年中，我们看到了选择在其字体家族中提供两种或者更多斜体的设计者的数量在增长，这也是值得注意的。同时需要注意到字典有时使用不止一种斜体样式。

当斜体最初制作出来用于打印时，斜体并不被认为是相同字体设计或字体家族中的一部分。这个主意在19世纪和20世纪变为标准。甚至将斜体与常规体混合的想法也并不是这个书写字母背后的原本想法。最早的斜体字是用于整本书的，代替处置罗马样式。认为斜体的角色将会继续发展，这很可能是可靠的假设。

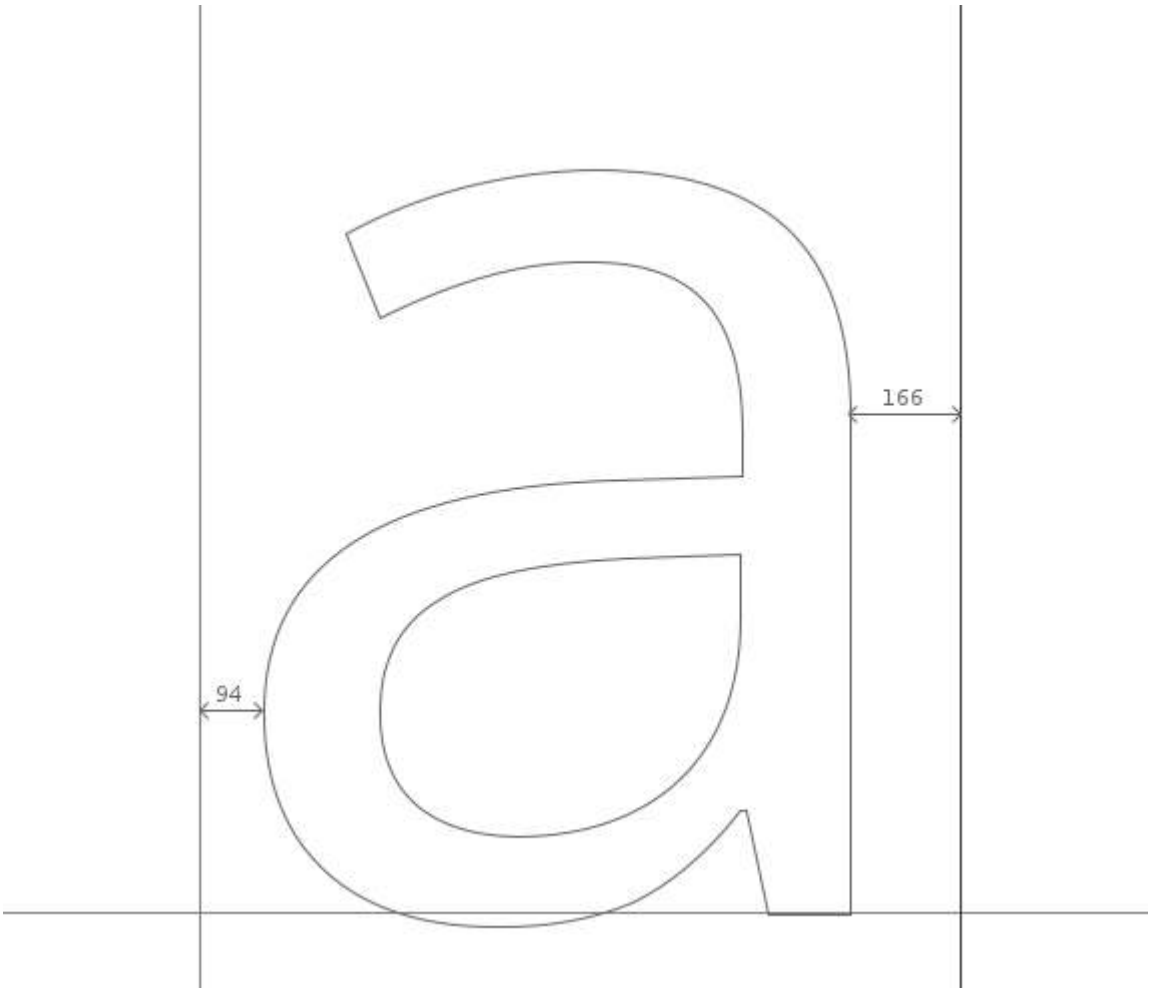
间距，度量值和紧缩

字符之间的距离是字体设计中重要的必须的一部分。

字体的字母间距的设计应该作为字体设计整个进程的必须部分来展开。好的间距对一个字体正常运行时必须的。

在FontForge中，度量值（Metrics）窗口允许你设计字体的度量值，修改他们之间的间距，并测试字形放在一起看起来怎么样。度量值窗口可以从“Window”菜单或者`Ctrl + K`命令打开。

任意两个字形间的间距分为两部分；第一个字形后的间距和第二个字形前的间距。字形间的这些间距是由两个字形间的“旁边空白”组成。每个字形有一个左跨距和一个右跨距，在下面的Open Sans字体的小写字母“a”的例子中，右跨距的值是166单位，左跨距的值时94单位。



度量值窗口的基本功能

字符的旁边空白在FontForge的度量值窗口中可以用5种方式编辑：

- 手动拖动每边空白的边界。
- 手动拖动一个字符。需要注意的是拖动一个字符将只影响左跨距的值。
- 旁边空白的值可以通过在度量值窗口的Metrics表格中直接编辑。
- 旁边空白的值可以通过快捷键增减/减少。

- 使用度量值窗口的Metrics菜单中的命令。

使用键盘调整旁边空白的值。

在FontForge中快速精确地调整度量值的一个方法是使用键盘的上下左右键。上下键用来增加和减少值，`Alt` + 上、`Alt` + 下、`Alt` + 左、`Alt` + 右用来到在度量值窗口中导航到周围不同的字段。

一般原则

作为一般原则，对称字符比如“A”、“H”、“I”、“M”、“N”、“O”、“T”、“U”、“V”、“W”、“X”、“Y”、“o”、“v”、“w”、“x”将会有对称的旁边空白，比如“H”的左跨距和右跨距值将会相同。需要注意的是尽管如此，这并不是一个硬性规则，而是一般规则。

在你调整你设计的字符的空间时，你应该相信自己的眼睛。大概方法是“设计-观察-调整-再观察”。

对于不折不扣的新手，不要假设可靠的结果以来测量间距来取得。例如虽然两个字符间的测量值不相等，但是眼睛看起来是相等的。一个明显的这样的例子在尝试调整字符“H”和“O”的间距的时候可以看到。因此对于下面的例子，“H”和“O”的旁边空白相等，但是看起来不相等。在下面的一条线上，旁边空白不相等但是间距看起来是平衡的。

HOOOHHHOHHOOO

HOOOHHHOHHOOO

产生这样文本的一个工具在http://tools.ninastoessinger.com/

用来编辑度量值的Metrics菜单命令

“Center in Width” - 这个命令将当前字形放置在其宽度的中央。

“Window Type” - FontForge的度量值窗口内可以通过两种方式调整度量值：

- “Advance Width Only” - 在这种模式下，度量值视图只能用来调整字形的步进宽度。
- “Both” - 在这种模式下度量值视图将会调整步进宽度和字距值。

“Set Width” - 这个命令允许你改变当前字形的宽度。

“Set LBearing” - 允许你改变左跨距的值。

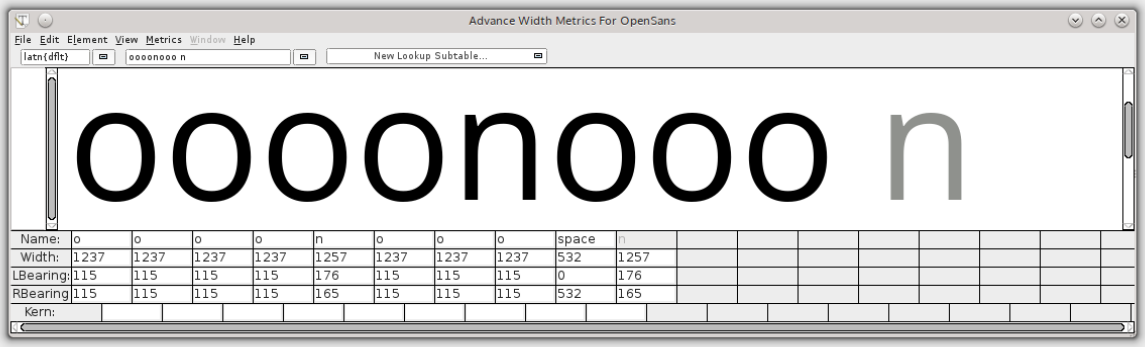
“Set RBearing” - 允许你改变右跨距的值。

一个调整间距的基本方法

下面的方法是用来让你开始高效地设计你的字体的度量值。

从度量值窗口的一个小写字母“o”的字符串开始，左边和右边的空白可以调整直到字符间距看起来感觉对。寻找这个正确的位置的一个方法是寻找“o”字符之间的空白来平衡字符“o”内的空白。通常出了斜体字体外，小写字母“o”的左边和右跨距应该值相等。一旦你满意“o”字符串间距，从你的字体中引进“n”（如下）并通过观察调整“n”的旁白空白着这

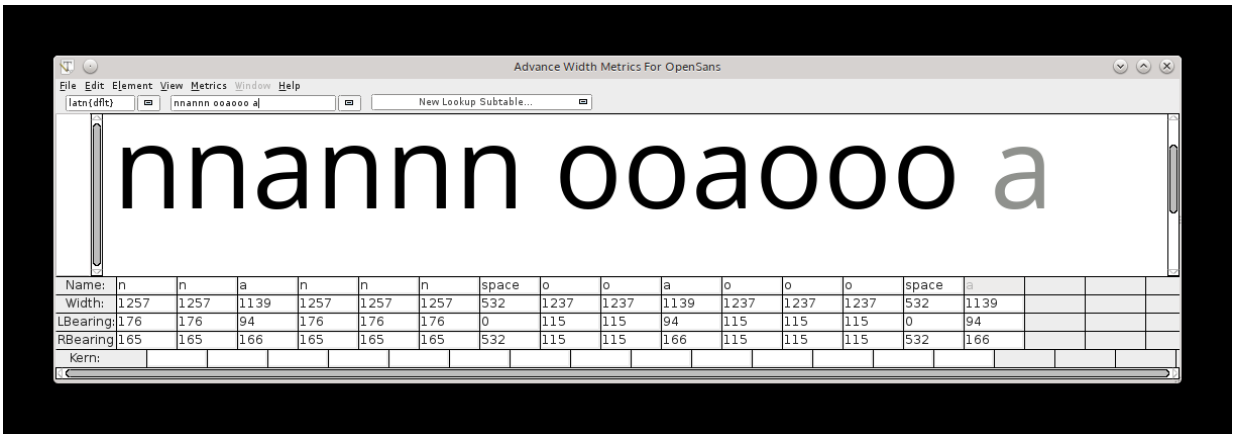
样它的间距适合“o”字符串的平衡（如下）。需要注意的是由于我们的眼睛看东西的本性，“n”的右跨距值总是比左跨距的值小，“o”的旁白空白比“n”的旁边空白小。



一旦“n”和“o”都有了充分间距，那么他们的旁边空白可以用来创建一组其他字母的旁边空白，例如：

- “o”的右跨距可以用于“c”，“d”，“e”和“q”的旁边空白。
- “o”的左跨距可以用于“b”和“p”的左跨距。
- “n”的右跨距可以用于“h”和“m”的左跨距。
- “n”的左跨距可以用于“b”，“h”，“k”，“m”，“p”和“r”的左跨距。

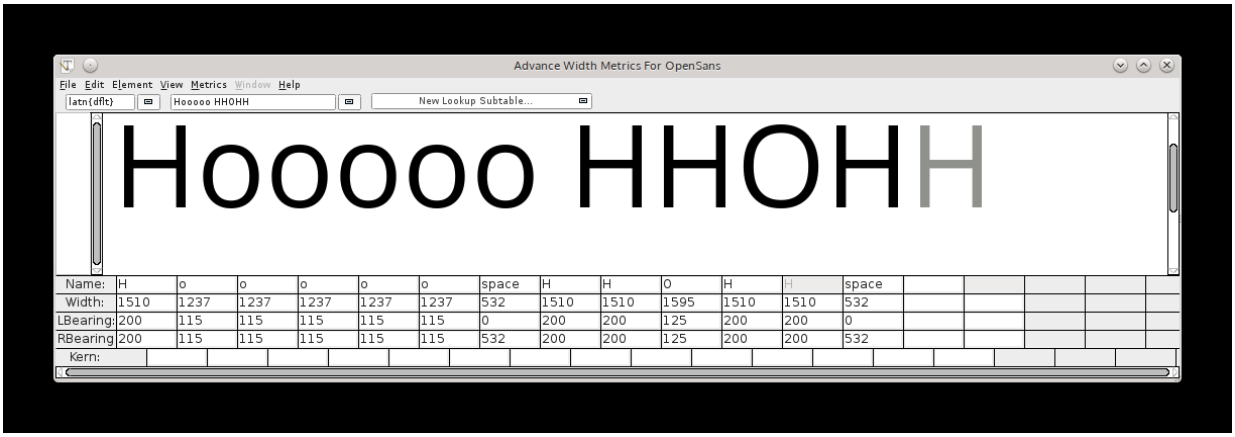
注意：上述应该用作指南，可以用作找到这些旁边空白值的一个超有效的出发点。



如上图所示，这对于使用“n”和“o”的字符串来调整剩余小写字母的旁边空白间距是有意义的。再次强调，相信你的眼睛来达到字符的正确平衡。

大写字母

大写字母可以用如上相同的原则来调整间距。例如从字符串“Hooooo”开始，调整“H”的右跨距直到感觉与“o”字符串平衡。由于“H”的左跨距等于右跨距，那么大写字母“O”可以通过“H”来调整间距（如下）。



通过已经调整好间距的字符来从这里调整所有其他字符间距。需要注意这个方法可以用作调整字体间距的好的起点，但是很可能需要花费更多时间很好地微调间距来实现较高层次的好的字母间距。其他在这里有用的字符串包括“naxna”，“auxua”，“noxno”，“Hxndo”。

紧缩

紧缩是特定字母对之间间距的调整。紧缩允许你在两个字符之间应用除了字符提供的旁边空白外的独立间距。需要紧缩来改善间距的常见一对字符例子有“WA”，“Wa”，“To”，“Av”。在下面的例子中，我们可以看到没有紧缩间距的一对字母“T-o”和“V-a”太宽了，这些字符对在紧缩间距后字体的其他间距的感觉更加平衡。

Toned Avenue
Toned Avenue

FontForge的度量值窗口可以用来设计旁边空白和紧缩值。在FontForge中紧缩值可以通过一些方法来应用，其中两种方法在下面展示，类别紧缩和独立对紧缩。

FontForge的Metrics菜单

“Window Type” - FontForge的Metrics窗口可以通过两种方式实现紧缩调整：

- “Kerning Only” - 在这种模式下度量值视图只能用来调整紧缩。
- “Both” - 在这种模式下度量值视图将会调整步进宽度或紧缩值。

“Kern By Classes” - 这个命令提供一个操作紧缩类别的对话框给用户。

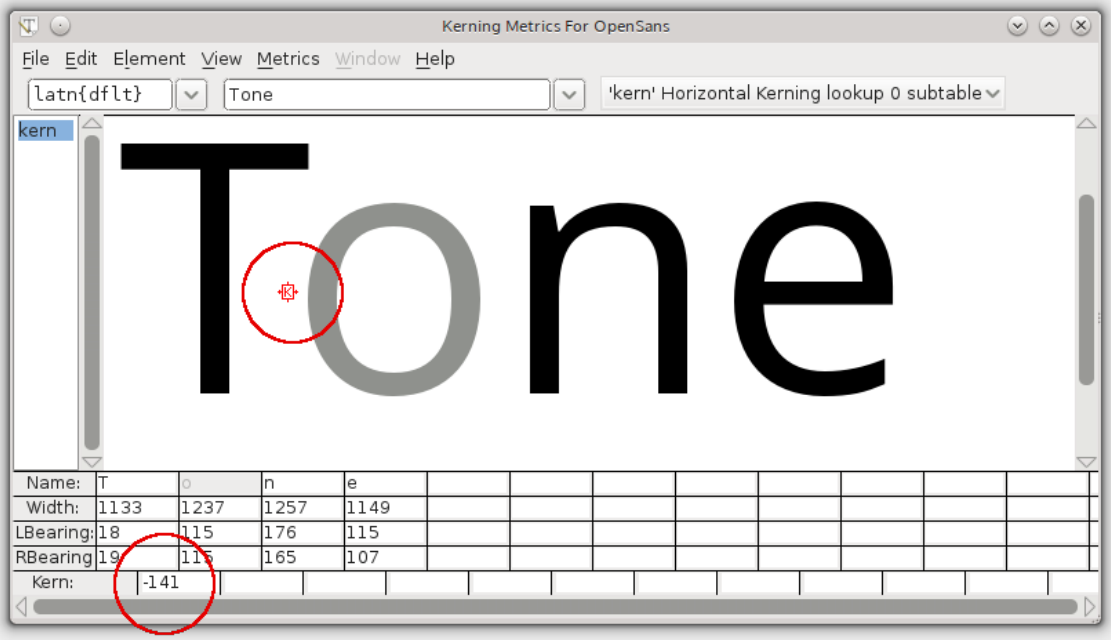
“Kern Pair Closeup” - 这个命令提供一个可以调整已有的紧缩对或者创建新的紧缩对的对话框给用户（如下）。

使用键盘调整紧缩值

像调整旁边空白值一样，在FontForge中紧缩值也可以通过键盘的上下左右键快速准确修改。上键和下键用来增加/减少值，alt+上、alt+下、alt+左、alt+右用来在度量值窗口中导航到周围的不同字段。

紧缩独立的对

在FontForge中这是创造紧缩的对的最基本的方法。在度量值窗口中，两个字符的紧缩值可以通过拖动右边字符接近或远离左边字符或者在窗口的度量值表格中直接编辑紧缩值来手动调整。通过拖动字符来修改紧缩值，则使用在鼠标指针悬停在两个字符之间时出现的紧缩工具手柄（屏幕截图如下）。度量值表格中的紧缩值可以通过手动输入值编辑或者使键盘上下键来增加/减少值。

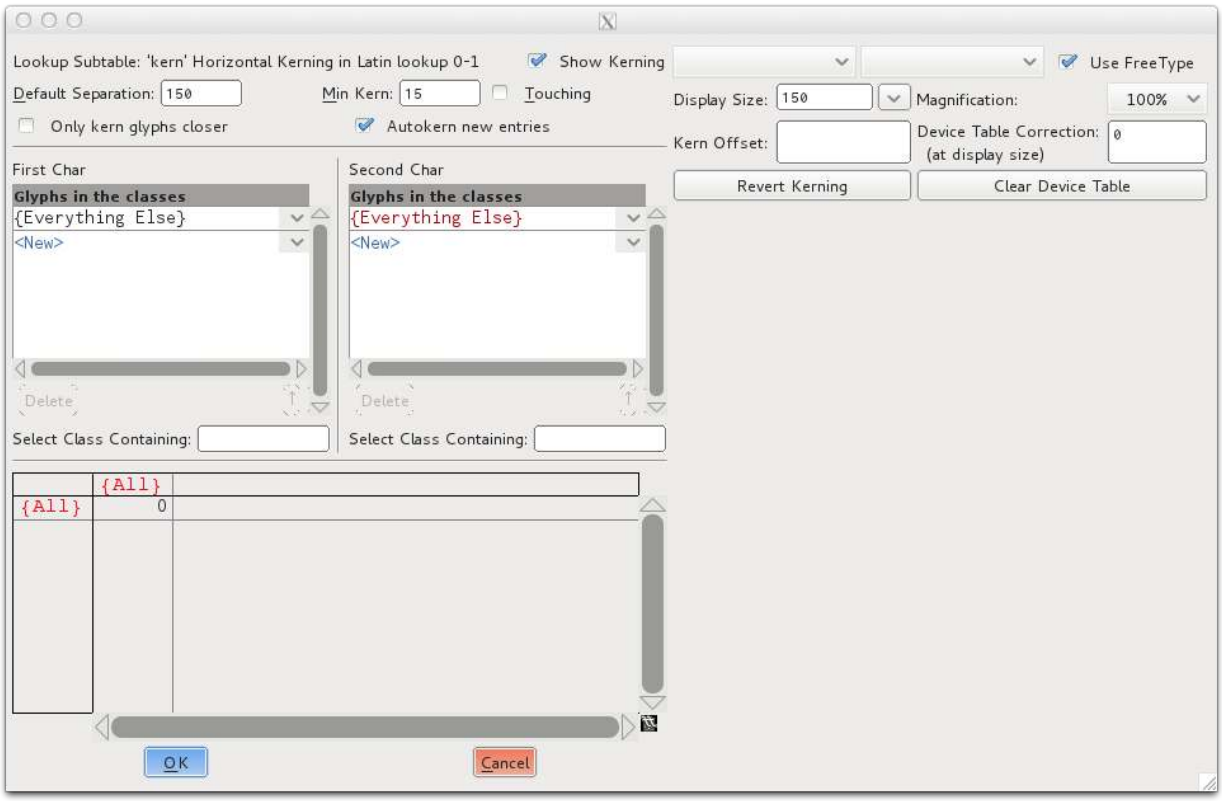


使用类别紧缩

“紧缩类别”可以在FontForge中创造来构建拥有相同紧缩值的字符组。例如我们创造了一个称为“o_left_bowl”的类别，其中包括的字符“o”、“c”、“d”、“e”、“q”在前面有字符例如“T”的时候将总是有相同的紧缩值。“T”本身也可以是包含其他字符的另一个类别，比如Tcaron和Tbar的成员。类别紧缩可以有效地节省你的许多时间。

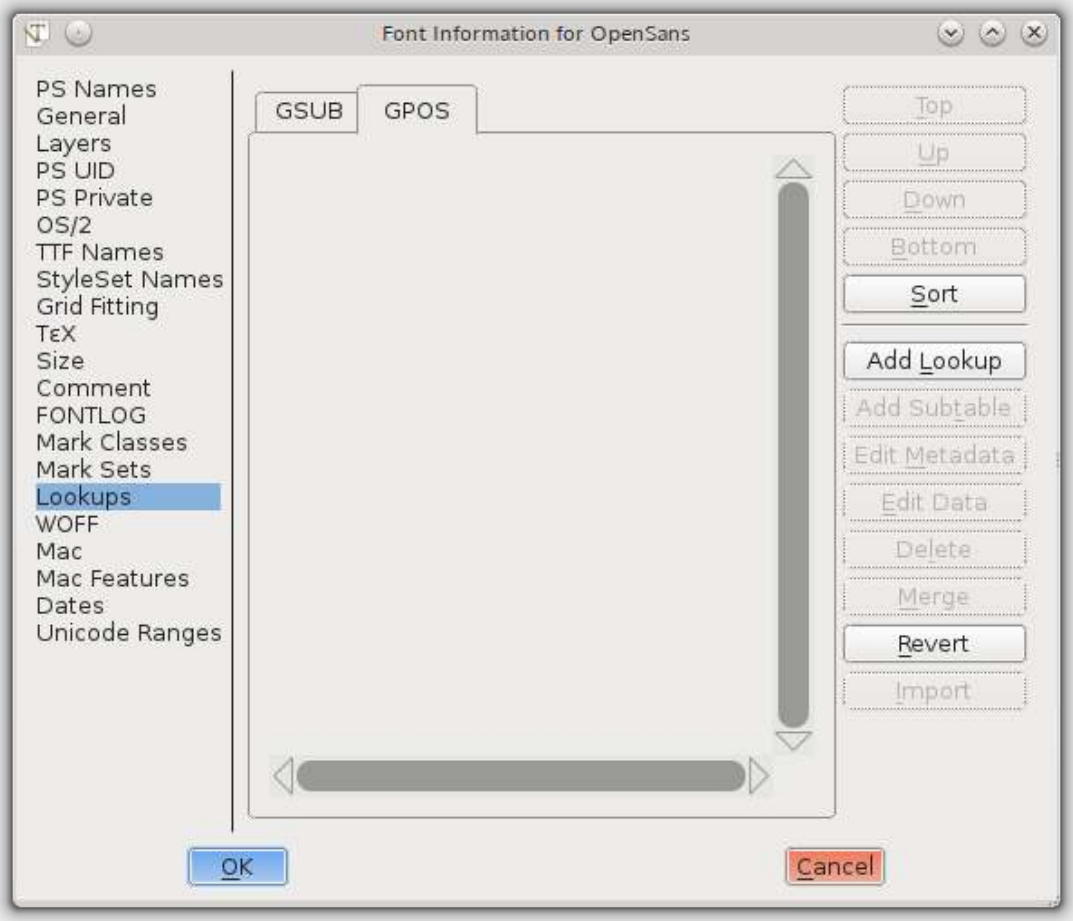
创造紧缩类别的最直接的方式是使用FontForge的度量值菜单的“Kern by classes”。

- 选择“Kern by classes”，可以看到“new lookup”按钮。
- 点击“New Lookup”按钮，另一个窗口将会弹出，你可以在这里创建一个紧缩特征查找。
- 在“Type”下拉列表中选者“pair position kerning”条目。
- 现在点击“Feature”列“NEW”旁边的向下剪头，在下拉列表中选择“Horizontal Kerning”。
- 点击“OK”。你可以保留FontForge为你创造的默认名称。

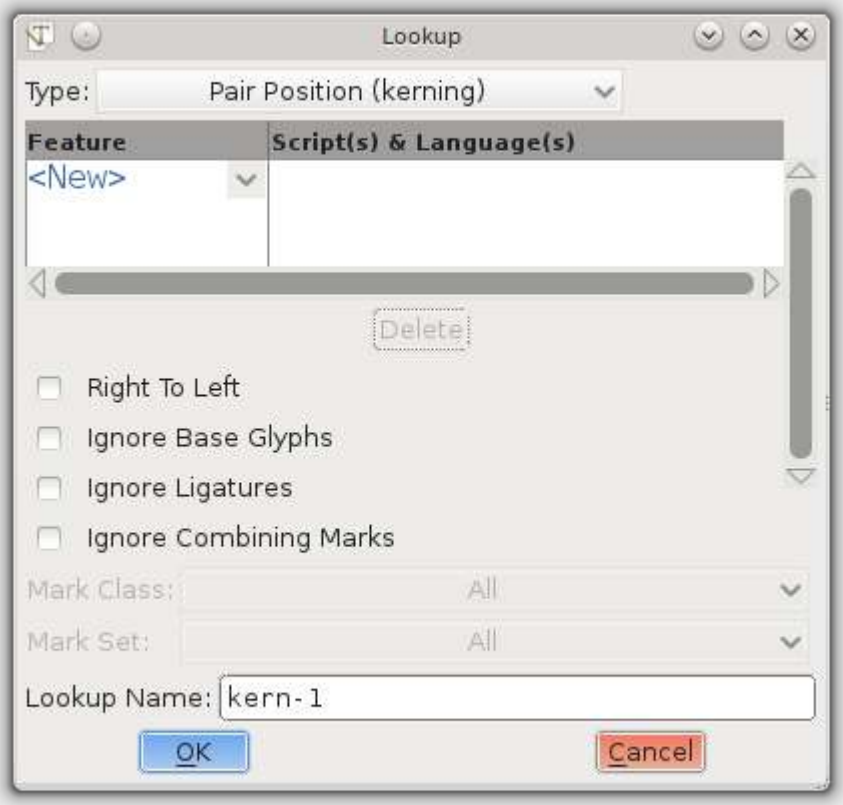


现在你可以看到一个窗口，可以在其中构建你的实际紧缩类别（如上）。紧缩对的第一个字符可以从左边的列中选择，第二个字符可以从右边的列中选择。

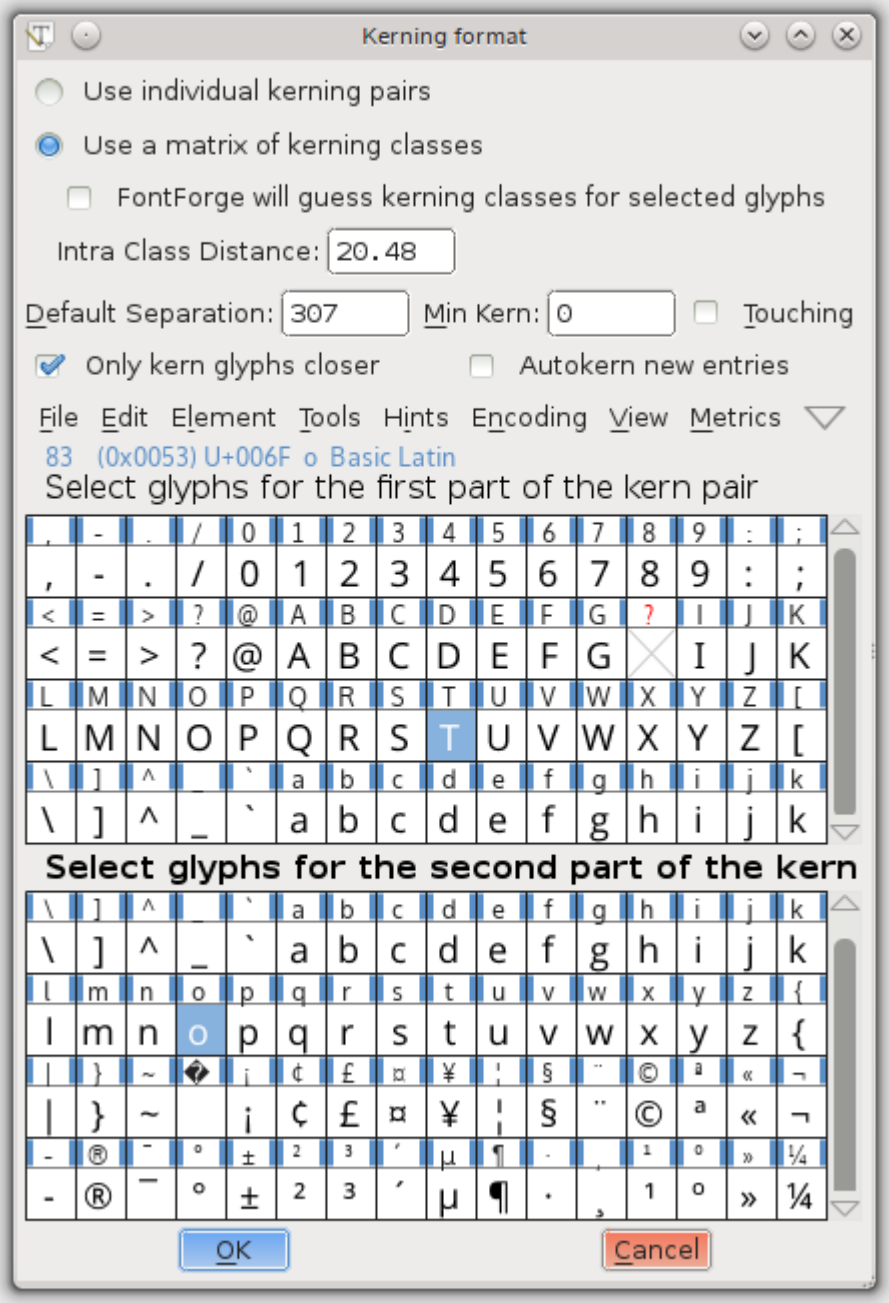
在FontForge中Element > Font Info > Lookups选项卡提供了一个类别紧缩的界面。它提供了一个对话框展示所有的GPOS查找（紧缩是其中之一）和其子表。屏幕截图如下：



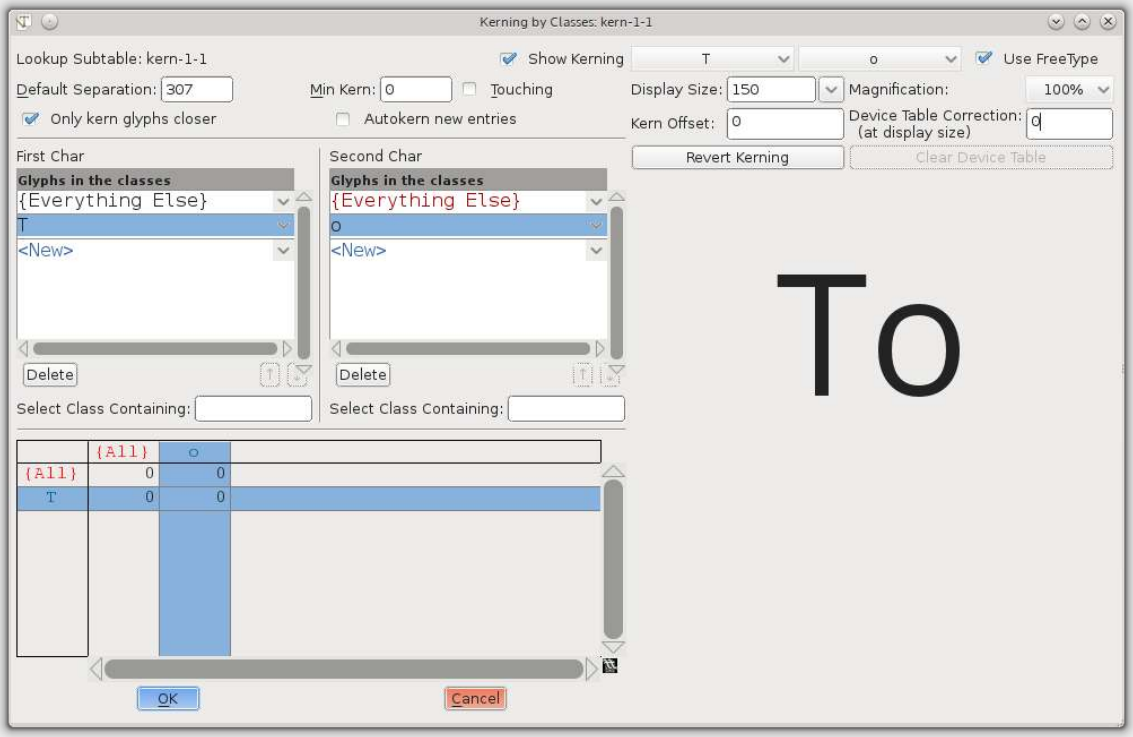
为了创建一个新的紧缩查找，点击“Add Lookup”并选择“Pair Position（kerning）”作为查找类型，并赋予这个查找自己的唯一的名字（如下）。



紧缩类别的每个集合都存在他自己的子表中。为了创建一个子表，点击“Add Subtable”。当你创建一个紧缩子表的时候，你会被询问想要一个独立紧缩对的集合还是基于类别的矩阵。如果你选择类别，你将会看到如下对话框，你可以在其中创建类别。需要注意的是你可以选择启用FontForge来“推测”或“自动紧缩”你在对话框中创建的类别之间的紧缩值。如果使用FontForge来推测紧缩值，你将确实需要许多尝试和错误和试验，但是使用自动紧缩功能作为紧缩你的字体的起点是有意义的。



例如在上面的屏幕截图中，创建了2个类别；一个类别包含字符“T”，另一个包含字符“o”。点击上面对话框中的“OK”，你将会看到下面的对话框，在里面你可以很好地调整这两个“T”和“o”类别之间的紧缩总量。



手动紧缩

如果自动紧缩的值需要调整（他们将会需要！），那么你可以用几种方法来完成。

- 通过“kerning by classes”对话框窗口。
- 使用度量值窗口。
- 使用度量值窗口中的“Kern Pair Closeup”命令。

另请参阅

决定字符间距的策略

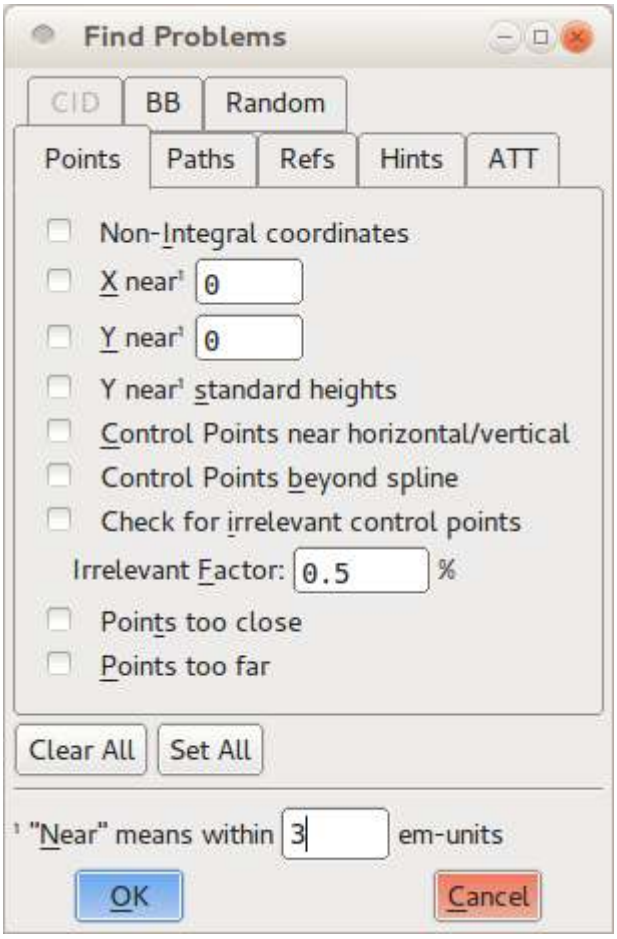
最终输出，生成字体文件

在一个完美的世界中，你的字体随时可以构建并安装在任何现代计算机上而不需要做任何额外的努力，但是现实是混乱的 — 尤其是在设计进程中。字体可能有阻止他们正常工作和显示的技术性错误。例如和自己相交的曲线不会正确渲染，因为他们没有“里面”和“外面”。各种各样的字体文件类型也期望字形附加在某些简化文本放在屏幕上的某些规则，破坏这些规则的字体可能引起意外问题。这种类型问题的一个例子是一条曲线上的所有点的坐标应该是整数。最后，有一些非技术性错误的样式错误，但是你也要修复它们 — 比如想要完全水平或竖直的线，但是偶尔轻微离开。

FontForge提供了工具让你可以定位（并且在许多情况下修复）所有的三类问题。验证你的字体来消除这些错误不仅可以确保用户可以安装并享受字体，而且可以确保完成的项目表现得耀眼。

发现问题

第一个工具叫做em>Find Problems，可以在Element菜单下找到。你必须首先选择一个或多个字形 — 从字体视图或者轮廓视图或者度量值视图 — 然后打开Find Problems工具。这个工具在八个分开的选项卡下展示潜在问题的混合。



你选择你感兴趣寻找的问题，方法是勾选它们旁边的复选框，有些还提供了用于检查字体的数字值。当你点击OK按钮时，这个工具将会检查所有选中的字形并在一个对话框内报告它找到的任何问题的报告。

Find Problems工具找到的问题被分成八类：

- 点相关的问题
- 路径和曲线的问题

- 引用的问题
- 提示（Hinting）的问题
- ATT的问题
- CID字体特有的问题
- 边界框问题
- 各种其他问题

并不是每个检查都是必要的；一些只应用在特定的书写字母或者语言（比如“CID”选项卡中的那些），另一些只应用在特定可选择的字体特性（比如在引用选项卡中的检查）。但是你应该检查你的字体是否通过字形的必要特性的测试和一些可选但是通常的行为的测试。几个其他测试在设计进程中为你提供给了反馈和指导，因此值得探索。

先说重要的：测试必要特性

在“Points”选项卡，选择*Non-Integral Coordinates*测试。这个测试确保每个字形中的所有点（包括弧线上的点和控制点）都有整数坐标。并不是每个字体输出格式需要这个行为，但是一些需要。

在“Paths”选项卡，选择选项*Open paths*和*Check outermost paths clockwise*。他们都是所有字体的强制特性；第一个寻找任何不是闭合形状的曲线，第二个确保每个字形的外部曲线是沿着顺时针顺序。这也是一个检查*交叉路径*的非常好的主意；尽管现代字体格式可以支持两个交叉路经，但是并不允许与自己交叉的曲线。另外如果一个字形有与自己相交的路径那么FontForge不能执行*Check outermost paths clockwise*测试。

在“Refs”选项卡，选择所有的6个测试。这些将检查所有相关的引用，引用中一个字形包含另一个字形的路径。例如一个带重音的字母包含了一个到原始（无重音的）字母的引用，加上一个到重音自负的引用。在“Refs”选项卡下的所有测试至少对于一个通常输出格式是强制的，对所有的来说是好主意。

类似地，选择“ATT”选项卡下的所有测试。这些测试寻找缺失的字形名称，引用不存在字形的替代规则，和其他字形名称活OpenType特性相关的问题。它们防止的问题并不常用，但是所有的都将导致字体被一个或多个电脑系统认为是无效的，因此它们值得被包含进来。

让你的用户生活更容易：测试好的行为

上面列出的测试将会保证你的字体按照多种字体格式设置的规则集合正确地安装和渲染，但是你应该在考虑添加一些其他测试 — 尤其是在设计进程的结束 — 仅仅因为它们检查大部分现代印刷格式遵循的公约。

在“Points”选项卡，选择*Control points beyond spline*。这个测试将寻找处于其所属的曲线段的端点外的控制点。很少有一个控制点应该处于曲线以外的原因，因此这样的情况通常意味着意外。选择*Points too far apart*也是一个好主意，它将会查找距离最近的点超过32767单位的点。这个距离比大多数计算机能够内部处理的更大，并且一个那么远的点几乎必然不是故意的（可以对比的是，一个字形倾向于会只在一个大约1000单位的网格中），因此删除这样的点是重要的。

在“Paths”选项卡，*Check Missing Extrema*和 *More Points Than [val]*测试可以是有价值的。第一个寻找处于极值的点 — 也就是字形的最高点、最低点和最左点和最右点。现代字体格式强烈建议每个路径都有一个处于其水平和垂直上极值的点；在字体渲染在屏幕或页面上时，这将让生活更美好。检查将会查找缺失的极值点。第二个测试是一个对字形中点的数量的明智的检查。FontForge中这个检查的默认值是1500个点，这是PostScript文档中建议的值，对几乎所有字体都足够好。

正如其名字一样，“Random”选项卡列出不属于其他类别的杂项测试。当然，最后三个是有价值的：*Check Multiple Unicode*，*Check Multiple Names*和*Check Unicode/Name mismatch*。它们寻找字形名称和Unicode码之间映射的元数据错误。

帮助你自己: 运行测试可以援助设计

Find Problems工具中的许多其他测试有助于找到并定位你的字形集中的矛盾之处；不是错的或者无效的但是作为设计者的你希望打磨的东西。例如“Points”选项卡中的*Y near standard heights*测试将字形与一个有用的竖直测量值集合来比较，包括基线、字形“x”的高度、字母“p”的最低点和最高点等。在一个一致的字体样式中，大多数字母将依附

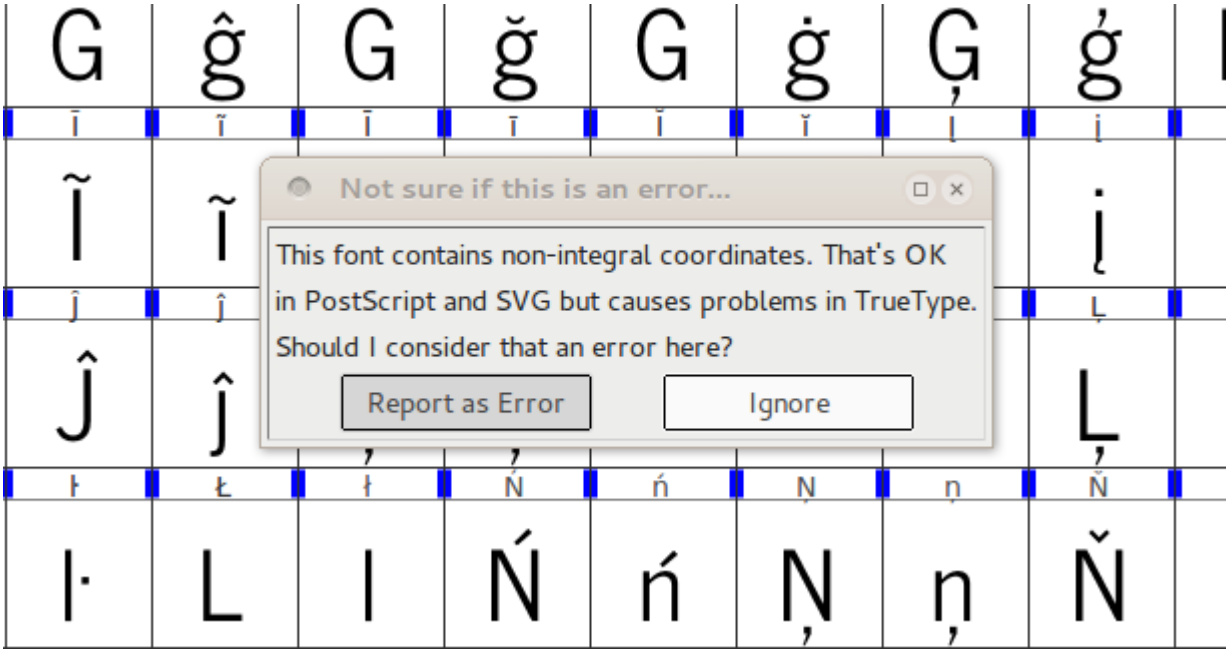
到这些标准测量值中的几个，因此一个并不靠近其中任何一个的字形需要大量的工作。

在“Paths”选项卡中的*Edges near horizontal/vertical/italic*功能寻找几乎是精确地处于水平、竖直或者处于字体倾斜角度的线段。将你几乎竖直的线条做成完美竖直意味着在你的字体使用时图形将会锐利地渲染，这个测试是找到在眼睛没有帮桌的情况下难以察觉到的不十分正确的线段的可靠的手段。

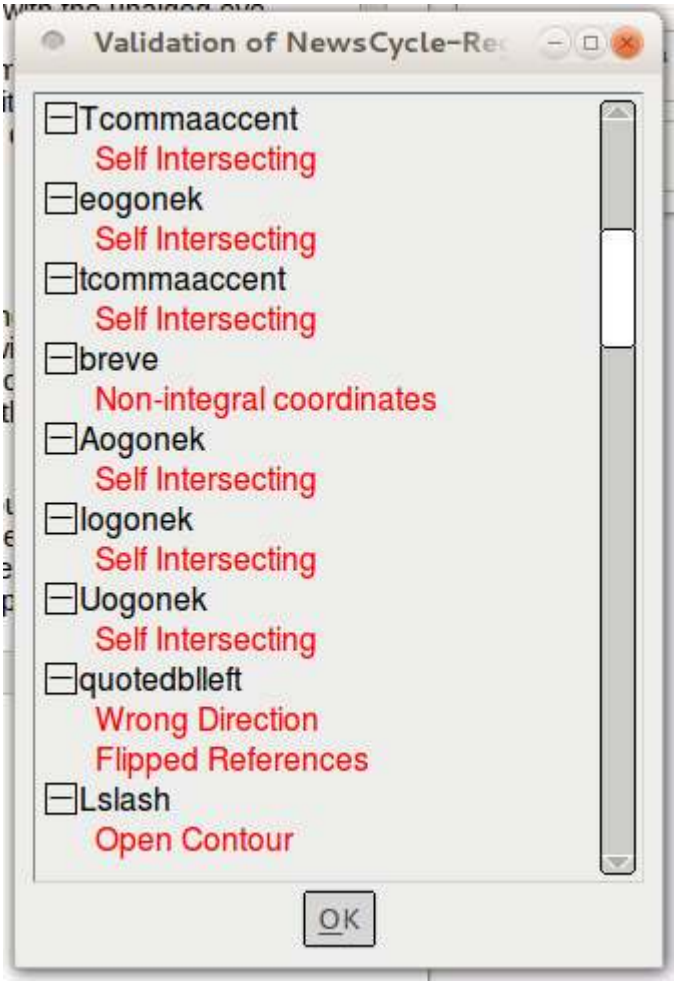
你可以使用其他测试来定位曲线上互相太靠近的点而没有意义，来比较形状类似字形的旁边空白，来执行一些当你有古怪的字符时揭露问题。改善进程的一部分是采用你最初的设计并使他们更加精确；像字体设计的其他方面一样，这是一个迭代的任务，因此使用使用内建的工具以减少一些重复工作。

验证字体

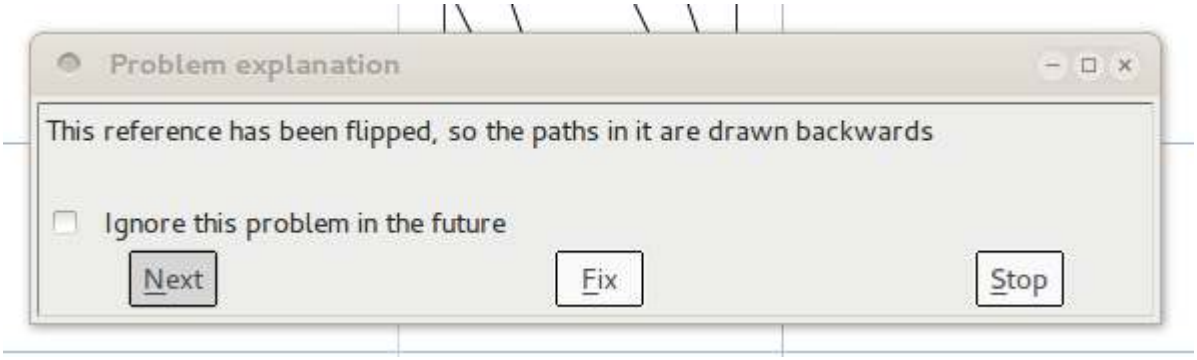
FontForge的其他验证工具时全字体验证器，在整个字体上做一系列的测试和检查。由于验证器是用来检查整个字体的，因此你只能从字体视图窗口运行他；你可以再Element菜单的Validation子菜单中找到它。验证器设计用于运行一些检查字体技术上正确性的测试 — 本质上是上面“测试必要特性”一节描述的测试。但是它确实在整个字体上执行测试，并且比你自己用Find Problems工具一步步完成的进程要迅速很多。



在一个特别的编辑期间你第一次运行验证器的时候，他将会弹出一个对话框询问你它是否应该将非整型的点坐标当作是错误。安全的答案是选择“Report as an error”，因为附着在整型坐标上是一个好的设计实践。当验证器完成它对字体的扫描时（将仅仅是几秒钟之后），它将会弹出一个新的名为*Whatever Your Font Name Is*的验证对话框这个窗口将会列出验证器找到的所有问题，展现在按字形排序的列表中。



但是这个窗口不仅仅是一个错误的列表；你可以双击列表中的每个条目，FontForge将会跳转到相关字形并高亮显示确切的问题，最后在它自己的窗口中展示文本说明。然后你可以再字形编辑器中修复问题，关联的错误条目将会立即从验证器的错误列表中消失。在许多情况下，错误可以被FontForge自动修复；在这些情况下说明窗口将会在底部包含一个“Fix”按钮。你可以点击它执行修复而不需要额外努力。



对于一些问题，并不能自动修复，但是在屏幕上看到问题将有助于你立即修复。例如一个自交曲线在一个特定的地方上路径穿过它自己 — 这对于你来说可能很难一眼瞥到，按时放大后你就可以改造形状消除问题。

对于另一些问题来说，并不能将错误定位到某一个特定的点。例如如果一条曲线沿着错误的方向（也就是应该是顺时针的时候反而是逆时针），那么整个曲线都会受到影响。在这些情况下FontForge不能自动修复问题，验证器也无法高亮没有字形中特定的点，你可能四处寻找以期手动改正问题。

最后，有一些验证器执行的测试可能并不是来自你心里想要的最终输出格式的问题 — 例如之前提到的非整型坐标。在这些情况下你可以点击错误说明窗口“ignore this problem in the future”复选框，在未来运行验证的时候抑制特定错误信息。

在你编辑时修复问题

大多数Find Problem工具和整个字体验证器找到的错误可以在编辑进程中改正，因此在你工作时不要觉得有任何推迟故障排除的需要。例如View > Show 子菜单有选项可以在编辑时高亮问题区域；Element菜单下的命令比如*Add Extrema*将会为大多数输出文件格式添加期望的极值点，复选框指示指示选择的路径是否朝向顺时针还是逆时针方向。如果你在字形编辑器中翻转一个图形（水平或者竖直翻转），那么你讲注意到它的方向也被自动颠倒。如果你点击Element菜单下的*Correct Direction*命令，FontForge将会立即修复顺时针/逆时针方向。养成在工作时像这样做小的修复的习惯，将会在随后的验证阶段为你节约一点时间。

设计是否起作用?

字体样式可以通过两种方式工作地更好或更差；可读性可易读性。

易读性意味着字形的设计足够明显可以立即正确地认出。有一些经常太过类似的字符对：

- 字母“L”和数字“1”
- 字母“O”和数字“0”
- 字母“Z”和数字“2”
- 数字“1”和“7”

可读性意味着所有字形为了更熟悉、舒适的阅读体验而共同工作。创建测试文档是最好的确保这一点的方式。如果你有完整的字母表，那么你可以排版真实的文本 — 例如使用FontFriend将你的字体拖动你希望读的一个长新闻中，然后打印出它来。

但是如果你的字体只包含字母部分，你可以使用一个测试文本生成器，比如LibreText.org和任何字处理器，桌面出版应用程序或者一般演示程序（比如Inkscape）来创建测试文档。

最终输出，生成字体文件

尽管你可以在FontForge中做大范围的测试，但是你将会需要生成可安装的字体文件，从而在开发进程中执行真实世界的测试。此外，你的终极目标当然是创造一个其他人可以安装和使用的输出格式的字体。不管构建字体是为了测试目的还是为其他人使用而发布，你将使用*Generate Fonts* 工具（在File菜单可以找到）来构建一个有用的输出字体。但是在构建最终产品的时候，你会希望使用一些额外的步骤。

FontForge可以将你的字体导出为各种不同的格式，但是在实践中只有两种是重要的：TrueType（使用*.ttf*文件名扩展）和OpenType CFF（使用*.otf*文件扩展名）。从技术上来说OpenType格式可以包含一些其他选项，但是CFF类型是广泛使用的一个。

用于测试的快速的脏的一代

为了测试目的的构建字体文件 — 比如在一个网页浏览器中检查艰巨 — 你需要只确保你的字体通过了必要的验证测试。

你可以使用Element菜单下的*Validate Font*工具（更详细的解释参见验证字体一章），或者你可以选择所有的字形（按`Ctrl` + A或者从“Edit”菜单选择Select -> Select All），然后运行一些命令来应用一些零散的基本改变。确保在你的进程更进一步之前保存你的工作，一些为了导出的验证会以微妙的方式修改你的字形的形状。

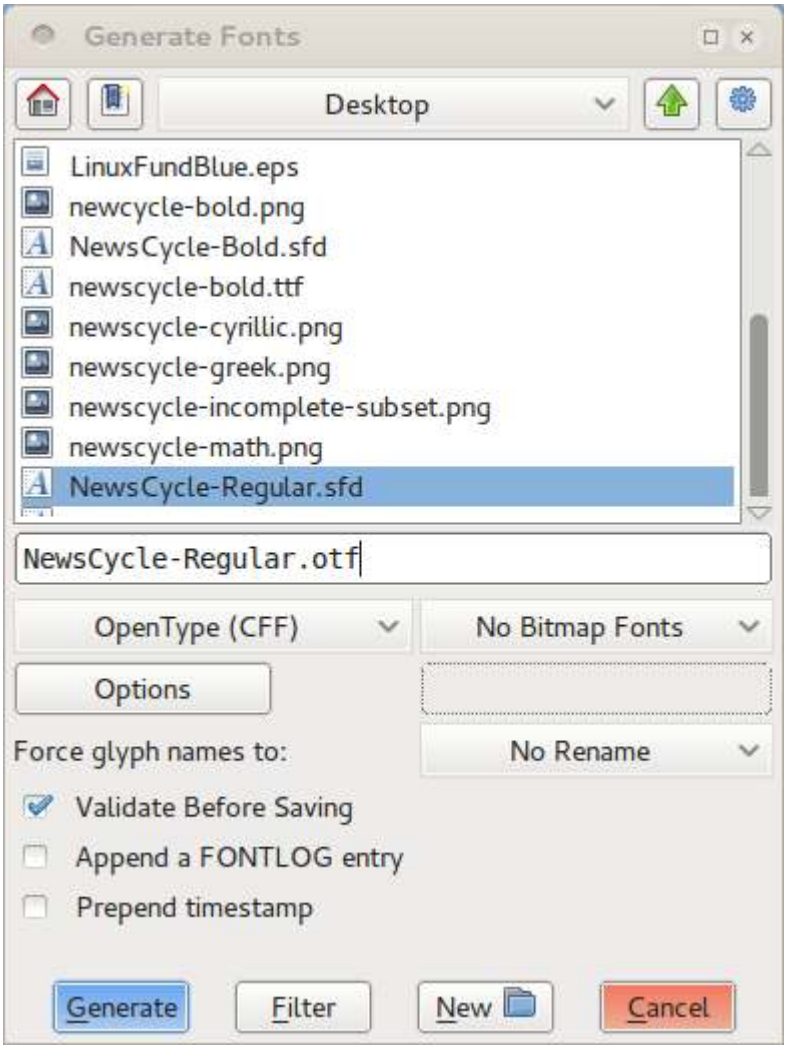
对于OpenType字体来说，首先要修正所有路径的方向。按`Ctrl` + `Shift` + `D`或者在“Element”菜单下选择Correct Direction”。接下来检查以确保没有未闭合的路径。勾选“Paths”选项卡下的*Open paths*选项，点击OK来运行测试。一旦你的字体通过测试没有错误，你就准备好生成OpenType输出。

对于TrueType字体来说，需要一些额外的步骤。你应该首先想上面描述的一样修正所有路径的方向。接下来调整所有的点使其拥有整型坐标：按`Ctrl` + `Shift` + `_`（下划线）或者从Element -> Round菜单中选择*To Int*。最终，像上面描述的那样打开“Find Problems”工具，选择*Open paths*，同时在“Refs”选项卡下选中所有。

在你运行这些测试没有错误时，你需要将你的路径转换为二次曲线。从“Element”菜单打开“Font Info”窗口。点击“Layers”选项卡，勾选*All layers quadratic*选项。点击窗口底部的OK，你就准备好生成TrueType输出。

构建字体文件

打开“File”菜单下的*Generate Fonts*窗口。窗口的上半部分展示了熟悉的文件选择视图 — 当前目录的文件列表，一个文本框可以让你输入文件名，和导航到其他文件夹或者目录的按钮。这确实是帮你快速找到保存你的输出文件位置或者在你希望覆盖之前保存的字体文件时选择已有字体的方法。所有你需要看的选项在窗口下半部分。



在左边是一个下拉菜单，你可以选择你希望输出的字体格式。如上所述你应该选择*TrueType*或*OpenType (CFF)*。右边确保选中*No Bitmap Fonts*。在下面一行，确保“Force glyph names to:”的选项*No Rename*选中。你如果你希望，你可以检查“Validate Before Saving”选项，但是这是可选的。保留“Append a FONTLOG entry”、“Prepend timestamp”和“Upload to the Open Font Library”不勾选。

点击“Generate”按钮，FontForge将会构建你的字体文件。你可以用其他应用程序加载字体，并运行任何测试，但是当你返回编辑时，记得重新打开在生成你的*.ttf*或*.otf*输出之前保存的字体的版本。

生成最终版本

设计你的字体是一个迭代的进程，但是最终你必须宣布你的字体完成或者至少准备好公众消费的那天到来了。在那时，你将会再次生成一个.ttf或者.otf输出文件（或者甚至两者都有），但是在做这之前，你将需要进行一些额外步骤来创建最符合标准和用户友好的字体文件版本。

首先，遵循用于测试的快速的脏的一代一节概述的相同的准备步骤。尤其是如果你准备创建一个TrueType文件时，记得修改你的字体为*All layers quadratic*。

删除重叠部分

如你所知，保持你的字母形式为独立组件的联合是好主意：茎、弧、衬线和每个字形的其他部分。但是尽管这一技术对于设计和改善形式来说非常好，但是你希望你最终发布的字体每个字形有简单的轮廓。这会减少一点文件大小，但是更重要的是它减少渲染错误。

FontForge有*Remove Overlap*命令可以自动联合一个字形分开的组件组成一个轮廓。选择一个字形（或者甚至使用`Ctrl`+A来选择所有字形），然后按`Ctrl`+`Shift`+`O`或者从“Element” —> “Overlap”菜单选择Remove Overlap。但是需要特别注意的一点是FontForge无法合并含有错误方向的形状（也就是形状外侧路径是逆时针方向）。虽说方向搞反了是路径本身的错误，不过也得需要你来修复它。

简化轮廓和添加极值点

你也应该尽可能简化你的字形 — 不是消除细节而是消除无用的点。这会轻微减小每个字形的文件大小，字体中全部字符集合加起来非常可观。

从“Element”菜单选择“Simplify” -> *Simplify*（或者按`Ctrl`+`Shift`+`M`）。这个命令将会合并去掉所有选择的字形上的多余的曲线上的点。在某些情况下，仅会有一些点被删除，其他情况下可能有很多。但是应该执行简化后并不显著地改变任何字形的形状。如果你注意到一个特定的字形被*Simplify*修改了太多，随意撤销这个操作。你也可以尝试使用相同菜单下的*Simplify More*命令；它提供了可调分组参数可能会很有帮助。

无论如何，在你完成了简化步骤后，你将需要添加丢失的极值点。从“Element”菜单选择*Add Extrema*（或者按`Ctrl`+`Shift`+`X`）。如前所述，在你编辑时在每个字形的极值处放置曲线上的点是好主意。然而你仍然必须在准备最终输出版本时执行这一步，因为*Simplify*步骤偶尔会删除极值点。

一切取整为整型坐标

最后执行的准备步骤是将所有点（包括曲线上的点和控制点）取整为整型坐标。这对于生成TrueType输出来说是强制的，但是对OpenType来说也是非常推荐的。它会导致在字体显示时更锐利的渲染和更适合网格，而不需要任何额外的设计工作。

为了使所有点取整为整型坐标，选择“Element” -> “Round” -> *To Int*。

一旦这个操作完成，你可能注意到有东西令人迷惑。有时候仅仅由于曲线独特性引起，取整到整型坐标的进程，简化字形，增加缺失的极值点可能互相冲突。这种情况出现的一个例子是一个曲线外边有一个控制点错过了水平或者竖直方向；在这种情况下对它取整到整型坐标可能轻微地移动曲线并改变其极值点位置。

这并不是这个难题的一个一蹴而就的解决方案；仅有的保证的修复是对受影响的字形循环重复步骤直到它们在一个点稳定下来，三个操作不再互相干扰。这可能需要耗费多次循环，但是它发生是罕见的。

验证

你的字体在生成最终输出之前应该通过必要的验证测试。像将点取整到整型坐标步骤一样，尽管有时其他准备操作能提出错误，因此在构建最终输出之前运行全字体验证是好主意。FontForge的验证工具一章将会给你检查什么的细节。

关于提示（Hinting）

提示指的是使用数学上的指引在字体中渲染矢量曲线，渲染的方式是用栅格化输出设备的像素网格（不论是墨水点或墨粉在纸上构成的网格还是计算机显示器的发光点构成的网格）精细地排列起来。

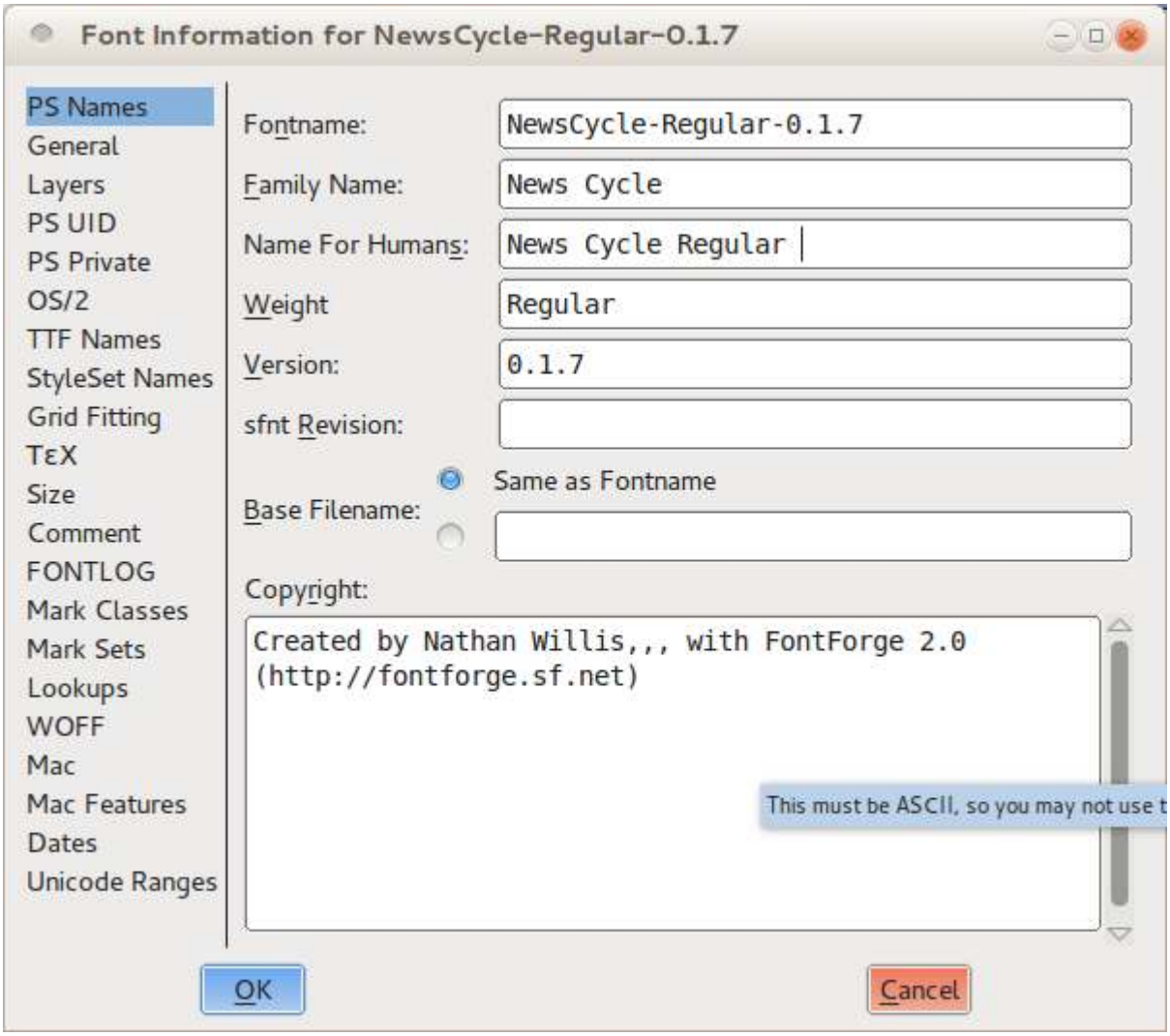
FontForge允许你提示你的字体（甚至提供了一个*Autohint*功能），但是实际上这个步骤并不是严格必须的。现代操作系统经常有构建在其文本渲染引擎中的更好的网格适合功能，而不需要话费可观的时间和努力。实际上Mac OS X和Linux都会*忽略*任何嵌入字体文件中的提示。如果为了Windows用户的利益，你确实选择你的字体确实需要提示，那么你最好的办法是不使用嵌入的提示来构建字体，然后使用专门的程序比如**ttfautohint**来事后增加提示。

CFF提示参见RoboThon上来自Adobe的这段视频。

检查你的元数据

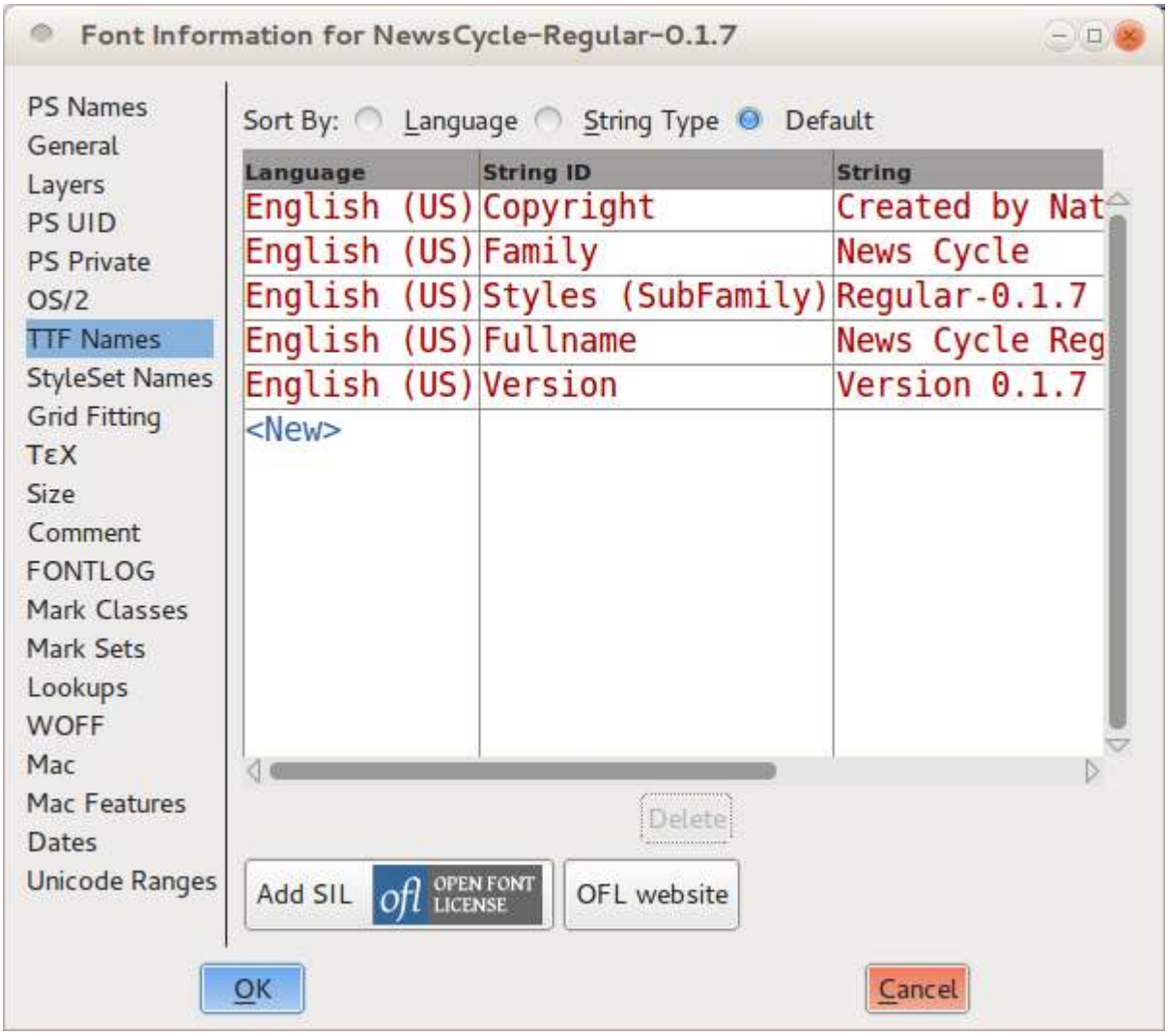
最后但不是最小的事情是，一但你的字体在技术上已经完全准备好导出，你应该暂停并升级字体的元数据，确保重要的元数据信息包含在内并且是最新的。

首先，如果这是你字体的初始发布版本，从“Element”窗口打开*Font Info*对话框，选择“PS Names”选项卡。首先填写字体的Family Name和Weight，然后复制这个信息到“Name for Humans”框。尽管使用版本号不是必要的，但是它对于你这样的设计者区分你的工作的版本是非常有帮助的。如果你不确定的话输入“1.0”作为版本号。接下来打开“TTF Names”选项卡并输入同样的信息。



在使用版本号的情况下，长期为每个版本制作日志项是有帮助的。切换到“FONTLOG”选项卡并写下简明的一两句话解释你为了发布正在构建的这个版本加入的改变。如果这是你的初始日志项，那么你也应该用一两句话描述你的字体和目的。

字体像所有的创意作品一样需要有一个证书，这样用户将会知道它们允许做什么不允许做什么。FontForge的“TTF Names”选项卡有一个“Add SIL Open Font License”标签的按钮。Open Font License（OFL）是一个字体证书，设计用于允许你分享你的字体给公众，而使用的时间和用在哪里的限制很少，同事仍然保护你这样的设计者不会被其他人将你的字体归功于他自己或者创造容易被误读为原创的你的字体的衍生物。惦记按钮将会为TTF Names元数据添加“License”和“License URL”字符串。如果你更愿意使用另一个许可证而不是OFL，那么在“License”域输入它。



如果你对你的字体的其他特点做了显著的改变，那么在Font Info窗口加倍检查字宽设置，确保一切仍然是最新是好主意。例如行间距信息可以再“Mertrics”下的“OS/2”选项卡下找到。

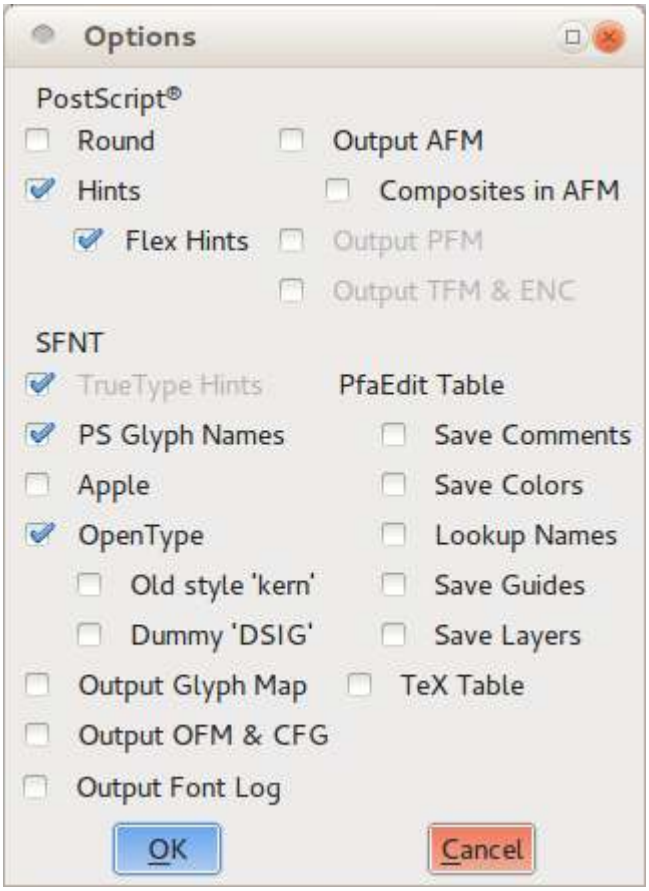
构建字体文件

生成字体输出文件的进程与你构建最终发布版，测试为构建快的脏的复制版本一样，但是你将希望对选项中的一些花费更大精力。

通过选择“File”下的*Generate Fonts*来打开这个窗口。窗口的上半部分再次允许你选择输出文件的目录和文件名 — 仅需要主意你并没有覆盖上次保存的。

像前面讲的一样，在左边的下拉菜单中选择你生成的字体的格式，*TrueType*或者*OpenType (CFF)*。在右边确保*No Bitmap Fonts*被选中。在下面的一行确保“Force glyph names to:”一项选中*No Rename*。如果你希望（可能捕获额外的错误）就可以勾选“Validate Before Saving”，但这是可选的。“Append a FONTLOG entry”、“Prepend timestamp”和“Upload to the Open Font Library”选项保持不勾选。

接下来点击“Options”按钮，在弹出的窗口中选中*PS Glyph Names*、*OpenType*和*Dummy DSIG*选项，并且不选中其他所有项。



点击“Generate”按钮，FontForge将会构建你的字体文件。最后一句话：重要的是不要用你在这一节单独为了生成你的*.ttf*或*.otf*输出而做的修改覆盖你的FontForge工作保存的版本。例如你当你执行*Remove overlaps*操作的时候丢失了许多独立字形组件。但是下次你继续你的字体工作时，你将肯定希望找回你在原作中丢下的独立字形组件填充的版本。

因此，如果你决定保存你的FontForge文件修改过的版本，确保将其重命名为显著的名字，比如*MyFont-TTF.sfd*或*MyFont-OTF.sfd*。但是你完全不必保存你的文件的这些面向输出的多种版本 — 实际上下次你在FontForge中修改你的原作时，无论如何你将再次做完输出准备步骤。

接下来要恭喜了！你创建了你的第一个字体。对你来说剩下的只是分享你的作品；上传到网上，贴到博客中，告诉你的朋友们。

毫无疑问你将会继续校正和改善你的字体样式 — 如你所见，字体设计终究是一个高度迭代的进程。但是确保你暂停一下，先把握这个时刻享受你所完成的。

当FontForge自己出错时

当你使用FontForge的时候发现它崩溃了，那么你可能想要发送信息到FontForge developer mailing list。使用这种方式，一个软件开发者可能能够找到代码哪里出错并修复。然后你可以给你本地的FontForge打补丁或者只是像安装一章描述的那样从Github获取源来的到一个不包含你报告的缺陷的升级后的FontForge。

为了帮助开发者找到什么出错了，他们可能想要你的会话的回溯。回溯包括一个程序函数调用其他函数到达程序停止工作位置的函数调用的列表。如果回溯包含了函数的行号，那么它将是最有用的。因为回溯将会制作到源文件和行号的引用，因此不要忘记也告诉开发者你在使用的FontForge版本。可选择地，你可能也希望提到你在做什么的时候导致了崩溃。

使用GNU Debugger来报告崩溃

一个回溯是使用GNU Project Debugger (gdb)来生成的。你可以附加gdb到一个已经运行的FontForge上或者在gdb会话中启动FontForge，如下：

```
$ gdb fontforge
GNU gdb (GDB) Fedora (7.3.50.20110722-16.fc16)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /usr/local/bin/fontforge...done.
```

然后一旦你给调试器发出了运行命令，FontForge将会在屏幕上打开。

```
(gdb) run
Starting program: /usr/local/bin/fontforge
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Copyright (c) 2000-2012 by George Williams.
 Executable based on sources from 14:57 GMT 31-Jul-2012-ML-TtfDb-D.
 Library based on sources from 14:57 GMT 31-Jul-2012.
```

到这里你可以用通常的方法使用FontForge，但是FontForge拥有了有效捕获并报告问题的好处。

从gdb内运行FontForge造成的一个主要不同之处在于如何让你一个崩溃变得明显。没有gdb时，当FontForge崩溃时它将从你的屏幕消失。但是当你在gdb中运行FontForge时，崩溃的FontForge及其窗口和用户界面将保持打开。

如果你发现你的界面不响应，切换回你启动gdb的终端，你可能在（gdb）提示后面的文本中看到一些东西比如“SIGSEGV”。如果你看到（gdb）提示那么FontForge不再运行。在此时使用“bt”命令来的到回溯，如下面的例子所示。

如你在示例回溯中看到的，FontForge在 `copy()` 函数内崩溃。 `copy()` 函数被 `KCD_AutoKernAClass` 函数调用。回溯将会告诉一个软件开发者这些调用的确切行，同时也提示传给 `copy()` 的参数无效（超出边界），从而去解决程序出错的地方。

```
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff74a7c01 in ?? () from /lib/x86_64-linux-gnu/libc.so.

 (gdb) bt
#0  0x00007ffff74a7c01 in ?? () from /lib/x86_64-linux-gnu/libc.so.6
#1  0x00007ffff6389a80 in copy (str=0x900000008) at memory.c:82
#2  0x00007ffff7a4aeb5 in KCD_AutoKernAClass (kcd=kcd@entry=0xe80c40, index=2, is_first=is_first@entry=1)
    at kernclass.c:236
#3  0x00007ffff7a51405 in KCD_FinishEdit (g=0xeb0fe0, r=1, c=, wasnew=1) at kernclass.c:2020
#4  0x00007ffff5effe2d in GME_SetValue (gme=gme@entry=0xeb0fe0, g=0xe94760) at gmatrixedit.c:988
#5  0x00007ffff5f00554 in GME_FinishEdit (gme=0xeb0fe0) at gmatrixedit.c:997
#6  0x00007ffff5f01c1a in GMatrixEditGet (g=g@entry=0xeb0fe0, rows=rows@entry=0x7fffffffcf78)
    at gmatrixedit.c:2214
#7  0x00007ffff7a4ea3c in KCD_Expose (event=0x7fffffffd1e0, pixmap=0x83ae00, kcd=0xe80c40)
    at kernclass.c:1446
#8  kcd_e_h (gw=0x83ae00, event=0x7fffffffd1e0) at kernclass.c:1762
#9  0x00007ffff5eabe8f in _GWidget_Container_eh (gw=gw@entry=0xe7f040, event=event@entry=0x7fffffffd1e0)
    at gcontainer.c:269
#10 0x00007ffff5eac385 in _GWidget_TopLevel_eh (event=0x7fffffffd1e0, gw=0xe7f040) at gcontainer.c:734
#11 _GWidget_TopLevel_eh (gw=0xe7f040, event=0x7fffffffd1e0) at gcontainer.c:606
#12 0x00007ffff5ef86ce in GXDrawRequestExpose (gw=0xe7f040, rect=0xef72b0, doclear=)
    at gxdraw.c:2687
#13 0x00007ffff5eea075 in gtextfield_focus (g=0xef72a0, event=0x7fffffffd2e0) at gtextfield.c:1888
#14 0x00007ffff5eaa857 in _GWidget_IndicateFocusGadget (g=0xe94760, mf=mf@entry=mf_normal)
    at gcontainer.c:143
#15 0x00007ffff5eaac97 in GWidgetIndicateFocusGadget (g=) at gcontainer.c:155
#16 0x00007ffff5f02b1e in GME_StrSmallEdit (event=0x7fffffffd670, str=0xe10e60 "A", gme=0xeb0fe0)
    at gmatrixedit.c:890
#17 GMatrixEdit_StartSubGadgets (gme=gme@entry=0xeb0fe0, r=1, c=c@entry=0, event=event@entry=0x7fffffffd670)
    at gmatrixedit.c:1472
#18 0x00007ffff5f03d69 in GMatrixEdit_MouseEvent (event=0x7fffffffd670, gme=0xeb0fe0) at gmatrixedit.c:1499
#19 matrixeditsub_e_h (gw=, event=0x7fffffffd670) at gmatrixedit.c:1735
#20 0x00007ffff5eabd98 in _GWidget_Container_eh (gw=0xeeb2e0, event=0x7fffffffd670) at gcontainer.c:393
#21 0x00007ffff5ef6555 in dispatchEvent (gdisp=gdisp@entry=0x769a50, event=event@entry=0x7fffffffd9b0)
    at gxdraw.c:3475
#22 0x00007ffff5ef7d1e in GXDrawEventLoop (gd=0x769a50) at gxdraw.c:3574
#23 0x00007ffff7ad353a in fontforge_main (argc=, argv=) at startui.c:1196
#24 0x00007ffff736676d in __libc_start_main () from /lib/x86_64-linux-gnu/libc.so.6
#25 0x00000000004006e1 in _start ()
(gdb) quit
A debugging session is active.

       Inferior 1 [process 19196] will be killed.

Quit anyway? (y or n) y
```

在gdb中使用gdb的quit命令来退出gdb并关闭崩溃的FontForge。如果你可以发送好的回溯给FontForge开发者，那么你可以为每个人提升程序的稳定性！不要对报告这些问题感到害羞，崩溃如果没有报告，那它被修复的可能性也很小。

设计天城体样式

感谢Adam Twardoch、Erin McLaughlin、Neelakash Kshetrimayum、Dan Reynolds、Pooja Saxena、Dr Girish Dalvi为本页贡献了如此多的想法

设计一个新的原创的天城体样式遵循的流程非常类似设计新的原创的拉丁文的流程。从自由字体的*自由*中能够获得的独特的好处是你可以为字体的初创者从没相处的新目的修改并重用它们 — 例如设计一个天城体并改造一个已有的拉丁字体来解决它。

天城体字形

天城体包含这些不同类型的字形：

- 辅音（36）
- 独立的元音（28）
- 元音maatras
- 字距
- 天城体数字（10）
- 拉丁数字（新的或者或者将已有的调整到与天城文字相适应）
- nukta组合
- 半形式
- 连接（独特的连字字形）
- 不同长度的“I”元音maatras
- 天城体标点符号
- 拉丁标点符号（新的或者调整已有的）
- 拉丁字母

咨询（天城体Unicode页面）和（微软天城体OpenType字体开发页面）来学习更多关于这些字形和印度语形状引擎如何工作的内容。

写书法或者与此接近学习字帖来学习书写字母如何工作是有帮助的，这样你会明白什么字母在结构上应该与什么其他字母相似。这些Aksharaya的天城体书法字帖中的2页可以用作笔的角度和字母比例的参考。.

首先做什么

在设计一个天城体和拉丁字体样式时，以在天城体旁边绘制拉丁文字开始是重要的。最早一步是设计“关键”字形，以此通过基础形状和间距（这在拉丁文中可能是‘adhesion’或者‘videospan’）来建立字体样式的个性。进程早期设计最低和最高的“高度极值”。

你将会需要大量的元音标记来开始质地和缩放的测试。

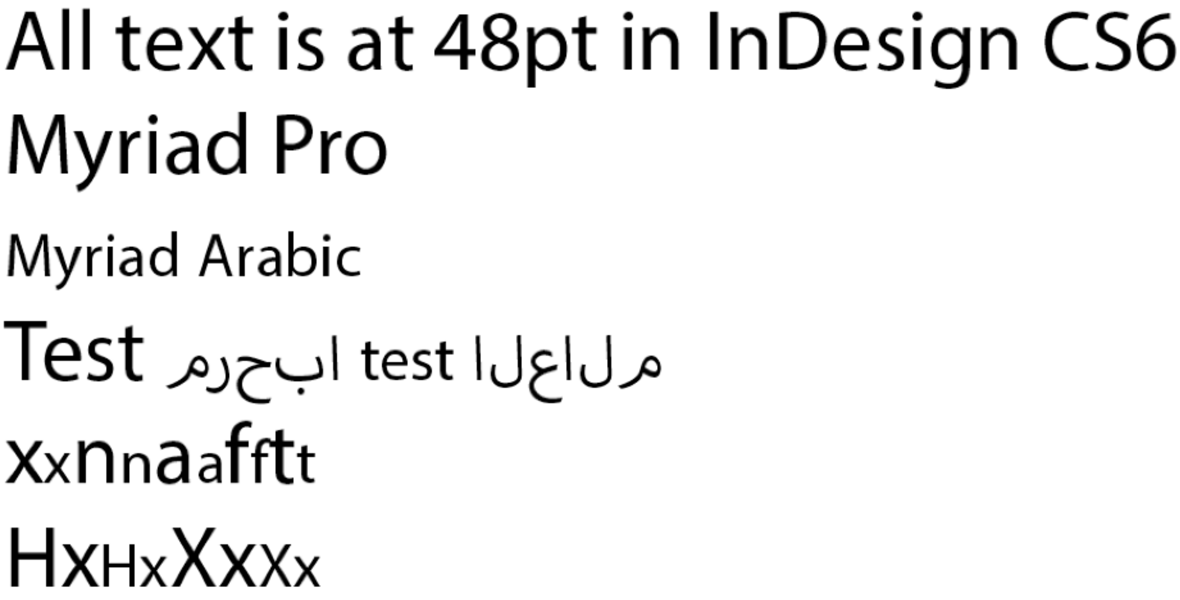
印度理工大学孟买分校的字体排印教授Girish Dalvi博士在他的博士论文中写到，

> 通过这一研究的结果我们可以推断出10个字母अ इ ए ख त भ द ध थ ष可以捕获几乎所有的剩余天城体字母的正式属性。在这些字母中，字母अ इ ख भ द ध ष定义了大多数字母的特性，是最具决定性的。因此我们认为通过首先设计这些字母，天城体设计的流程可以为学生和字体设计师简化，剩下的字母可以通过这些衍生出来。

Erin McLaughlin建议将这些字形作为一个初始连字：**पाव + किमीनुफू + भरसगदह + मौं ह्रु (height extremes) + यथधआछड … 连续字符集合**并建议关注“Au”元音标记 + reph + anusvara连字！这里的Ma是为了后继。

字形的高度极值允许你确定竖直度量值和如何缩放两个书写系统来共同工作。Adobe发布了非常大的字体家族覆盖各种不同的正字法。它们根据共享的一般比例分成不同的家族；Myriad Pro包含Latin,、Greek和Cyrillic，而Hebrew与Arabic设计打包成分来的家族，包含在**改进的**Latin中.

下面是Myriad Pro Latin和Myriad Arabic并排：



（认出Adobe的整洁的选择：Myriad Arabic的大写高度是Myriad Pro Latin的x高度）

需要注意的是在Lohit字符集中，最低的字形是形式，意味着靠下的字符将会下沉到比基线低很多的地方：

待办事项：添加vattu+U，vattu+Uu，U，Uu和subscript V（用于合词）的图片

（Vattu是底基线样式的reph。详情参见微软术语页面）

理想地，这些应该堆叠在你最低的竖直堆叠结合以下，就像左边的例子（Lohit并不完全竖直适应，在右边）

间距方法

设计拉丁字体有代表性地涉及了一系列艰巨字符串像这样：

> HHxHOHOxOO
> nnXnonoXoo

其中X代表了你关注间距的字母，概念是看这个字母挨着有点平的字符和圆的字符。

Pa与Va或Da是天城体相等的：

> पपXपवपवXवव
> पपXपदपदXदद

当刚刚开始一个项目的时候，从使用Pa完全填充一个页面开始，以此得到笔画粗细、对立面大小和间距的正确平衡。

> पपपपपपपपपपपपपपपपपपपपप

一旦Pa有了正确的“color”，你可以开始添加其他基本常见的字符：

> पपपवपपपपपवपववपपव （）va，随机化的）
> पपपापपपपापपापप （Aa maatra，岁计划的）
> पपपदपपपपपदपददपपद （da，随机化的）

然后你可以开始使用上面的间距字符串来添加新的字形：

> पपरपदपदरदद
> पपकपदपदकदद
> पपलपदपदलदद
> पपपीपदपदपीदद

诸如此类！

你将希望在一个像那样长的列表中看看这些，这样你可以在屏幕或打印介质上向下滚动时一个一个地比较字形。做竖直检查比只是长的一行连续文本更有效。原因是：

当你在竖直列中看间距字符串时，你可以很容易地比较当前字符到上下行之间的间距。按同样的方式，你可以容易地从坏地设置完全调整的文本中识别出“rivers”，如果你比较剩下的间距字符串，那么可以容易地在间距上看到白色间隙或者暗点。

上面的间距字符串允许你比较非常不同的形状，这样间距甚至更加贯穿（而不是所有太松或太紧的圆的字符）。

中间的四个字形Pa/Da/Pa/Da允许你将比较字符与两个三字符集合相比较，也就是Pa/Da/Pa或者Da/Pa/Da。

在对一些元音和辅音绘制和调整间距后，你将能够只使用这些字母做出数量有限的词并开始用实际文本测试你的设计。

作品分解结构

在任何一个字体样式设计项目中，描绘出一个作品分解结构是好主意。

对于一些非常熟悉的人来说，有可能在大约4-6个月内设计出天城体样式的初始的细体和粗体。

这里有一个经验丰富的设计者在设计有些简单的“sans”时，制作9种不同粗细、直立和倾斜的插值替换的字体家族的时间表：

Week	Goal	Glyphs
1	Establish design in 7-10 key glyphs	10
2	Refine, design tallest glyphs, match heights and weights to Latin in Regular & Bold, test screen rendering with ttfautohint	20
3	Refine proportions with native reader feedback	40
4	Get native reader feedback, refine and add more conjuncts	100
5	Get native reader feedback, refine and add more conjuncts	200
6	Get native reader feedback, refine and add more conjuncts	300
7	Get native reader feedback, refine and add more conjuncts	400
8	Get native reader feedback, refine and add more conjuncts	500
9	Get native reader feedback, refine and add more conjuncts	600
10	Get native reader feedback, refine and add more conjuncts	700
11	Get native reader feedback, refine and add more conjuncts	800
12	Get native reader feedback, refine and add more conjuncts	900
13	Derive Bold	1,800
14	Refinements, Kerning, testing with native reader feedback	1,800
15	Extrapolation and clean-up of Thin and Black weights, generation and clean-up of slanted styles	3,600
16	Interpolated styles refinement	3,600
17	General refinement of spacing, kerning & testing in all styles	3,600
18	Finalisation	3,600

有用的资源

简介

- http://www.linotype.com/6896/devanagari.html

从哪里寻找灵感和创意

在http://indiantypefoundry.comGoogle Fonts发布的网站查看天城体，来找到字体形状变化的灵感。

另一个好的地方是搜索印地语“e-paper”电子报网站来看实际使用字体的广告通常有更多的字体多样性。
http://epaper.jagran.com是一个流通非常广泛的印度电子报纸。

Flickr也是一个形象化创意的好的来源：

- https://www.flickr.com/groups/devanagari-script/
- https://www.flickr.com/groups/37703106@N00/
- https://www.flickr.com/groups/indicscripts/
- https://www.flickr.com/photos/pauldhunt/sets/72157603715699186

历史资源

得到H. M. Lambert编写的由牛津大学出版社于1953年出版的Introduction to the Devanagari Script和B. S. Naik编写的孟买语言理事会1971年出版的Typography of Devanagari（3卷）的副本。

除了那些，还有至少两种欧洲19世纪字体的一般来源值得一看：英国和德国（主要是莱比锡）的字体样式。这些字体更多地用来设置梵文字文本而非印度文本。

同时也尝试从印度字体铸造中找到19世纪和20世纪的文本字体样式的例子。像你期望的那样，它们明显很少欧化。从19世纪起在欧洲学院 梵文字体中有一些靠不住的东西，它们看起来根本不存在于20世纪的印度的排印中。这些印度来源可能更难从西方图书馆中找到，但是可能Erin McLaughlin更加领先。例如Matthew Carter的19世纪70年代的天城体Linotype排字机是基于Nirnaya Sagar公司的字体样式。它们的字体和孟买字体公司的字体应该可以在一些西方大学或者国家图书馆中找到。我也推荐查找Monotype的天城体和LinoType天城体（1970年代版和1980/90年代版，而非指示同名的原始的1935版）。

在荷兰的Typefounders并没有天城体（Charles Enschede, Harry Carter 1978）。无论你做什么，不要看Bodoni在1818年手工制作的字体。

一些来自H. Berthold AG的德国制造的天城体可能在Reichsdruckerei于1924年在柏林出版的Alphabete und Schriftzeichen des Morgen- und des Abendlandes的45-47页看到。

文章

Sarang Kulkarni写了"Issues with Devanagari Display Type (PDF)"

Yashodeep Gholap写了Designing a Devanāgarī text font for newspaper use (PDF)

Vaibhav Singh的MATD论文Devanagari in multi-script typography

Lohit2天城体

Lohit2天城体可以通过使用其字形列表和OpenType布局码来作为新OFL字体的基础。想要使用可以通过原始FontForge来源 or as a UFO zip下载

OpenType布局

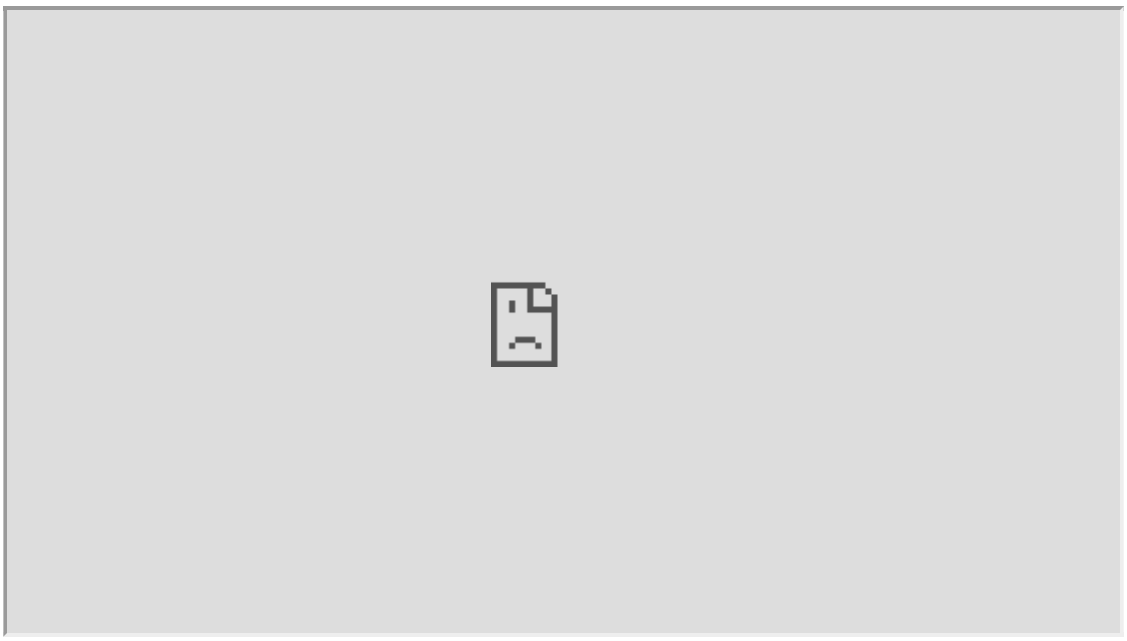
微软天城体OpenType字体开发页面

天城体刨析

Aksharaya的天城体书法指南的2页，可以用来作为笔锋角度和字母比例的参考。

印度理工学院孟买分校（比得上美国的麻省理工学院媒体实验室）的Girish Davli教授发表了这个天城体刨析文章（PDF）

如果你对天城体书写字母的不熟悉，那么知道传统书法笔锋力道是重要的，这与拉丁不同。这里有一个笔画角度和曲线如何传统地赋予粗细的快速文档。如果你遵循这些粗细原则来绘制你的曲线，而不是剪切粘贴拉丁字母的一部分，那么你的设计将会更加成功，并且看起来更少“拉丁化”。



天城体Unicode页面 展示了基本字母，但没有连字。

测试工具

Adobe InDesign对非欧洲书写字母的OpenType字体的支持只有在Creative Cloud中才可靠，甚至最新版本也没有改善。自由Harfbuzz与微软OpenType的实现是完成度最高的，因此你应该在Windows和Mac OS X下使用最新版本的Chrome、Firefox和微软Word来测试你的字体，以保证形状上的错误来自于字体而非底层引擎。

Pablo Impallari的天城体测试页面 （以及Github上的源码）提供了一些已经制作好的测试布局，你可以拖动你的OTF或者TTF字体到页面上来加载它。

Pooja Saxena的字体工具（以及Github上的源码）来生成字母连接的测试文本。

Adhesion天城体文本是Miguel Sousa构建的用来制造只包含可能使用你已经绘制的字形来书写的单词的假文本 的工具的一个特殊版本。插入你已经绘制的字形（अआईईउऊ等），将会诞生一些你用于证明的示例单词。

Huerta Tipografica的Devanaguide是一个开源工具，用来看和比较不同的天城体。它也允许你打字并同时在所有字体上预览。Devanaguide也包含了一个天城体词表，对测试文本设计有帮助。

论坛讨论

Typophile

- Adobe天城体

Google字体目录讨论

- 关于相比于拉丁，放置肩线、顶部、底部等等的约定?
- 从Lohit2开始与pdf
- Adobe字形列表
- rVocalic与rrVocalic
- 天城体字符优先顺序

从其他程序导入字形

在通用插图应用程序（Inkscape，Adobe Illustrator等）中绘制字形并将其作为EPS或SVG导入是有可能的。

手编SVG

如何准备

- SVG文件确实需要 `viewBox="0 0 1000 1000"`
- 宽度事实上并不重要，只要它比你的字形更宽。但是高度是1000对最容易的导入来说是重要的。
- `y=0` 将会是顶部线并且 `y=1000` 将会是底部线。
- (可能有一些字形超出了这些线，FontForge可能会正确处理，但是我们并未测试。)
- FontForge默认会设置你的基线为 `y=800` 。在FontForge的坐标系统中，基线处在竖直范围的 `0` 点处。
- 为了在FontForge中设置基线到你希望的地方，在SVG中将y坐标作为你的基线。在FontForge的坐标系统中，这将是其顶部线的竖直点。(1000 - y)作为底部。打开 `Element -> Font Info` ，在General选项卡下的“Ascent”和“Descent”输入框输入顶部值和底部值。两者都是正数。字模高度（Em Size）应该保留1000（因为这是SVG单位的高度）。
- 当绘制字形时，我喜欢使用相对坐标。因此我以 `<path d="M Xvalue,Yvalue` 作为字形的开始。如果我可以自始至终从左边的点开始绘制字形，那么Xvalue将会是FontForge使用的默认的左跨距。你可以在字形导入后容易地调整它，并可能在测试字体后无论如何都需要调整。当我可以从基线开始绘制时，将基线值作为Yvalue很好。
- 总是使用a z来结束路径的d属性。导入的时候不会导入它，但是如果你忘记在路径最后的点后放置a z，那么在主窗口中字形不会显示正确除非你重启FontForge。
- 当绘制洞（像字母P）的时候，不要开始一个新的路径节点，只需要在第一个路径的结尾使用a z并使用mNewX,NewY开始新的路径，然后开始绘制洞。为路径使用属性fill-rule="evenodd"，它将会工作正常。

工作流程

使用一个网页浏览器来渲染你正在制作的SVG。你可以使用一个被称作“template.svg”的1200乘1200的文件却渲染为800乘800，这样它在浏览器窗口中不会滚动。

在模板中，在 `y=100, y=1100, y=(100 + {baseline, capheight, etc.}, x=100, x=1100` 绘制引导线。

然后使用文档 `<image xlink:href="LC_p.svg" x="100" y="100" width="1000" height="1000" />` 将你正在制作的SVG字形导入。

现在你可以在一个窗口中手工编码你的字母，刷新浏览器来看它是否绘制在引导线的顶部。

自定义字形列表

1. 创建一个 `namelist.txt` 文件，可能使用一个电子表格来列出Unicode码点和字形名。例如：

```
0xEC00 octDotDhe
0xEC01 octDotDheDbl
0xEC02 octDotDheTrpl
0xEC03 octDotDheQdrpl
0xEC04 octDotLik
0xEC05 octDotLikDbl
0xEC06 octDotLikTrpl
0xEC07 minirLik
0xEC08 minirDhe
0xEC09 minirBawah
0xEC0A soroganDhe
0x-001 soroganLik
```

对于没有Unicode点的字形来说，使用-1的码点，正如上面例子的最后一行。

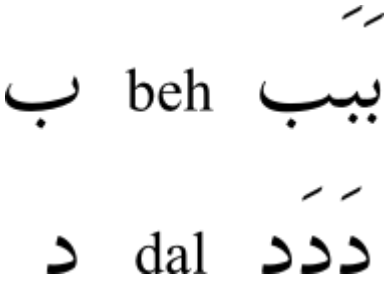
然后夹在FontForge并打开 `Encoding -> Load NameList` 然后使用 `Rename glyphs` （由于 `Load NameList` 只添加自定义名称列表到重命名之后的命令 的可用选项集合）。

添加字形到阿拉伯字体

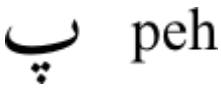
简介

在某些情况下一个字体可能缺少一个在你的应用程序中使用时必要的字形。阿拉伯字体在这里展示了特殊的问题，由于字形的形状不仅依赖于它在词中的位置，也依赖于字母本身的属性。因此（使用无意义的序列*babab*），字母*beh*有三种不同的形状，依赖于是否处于开头、中间还是结尾。然而（使用无意义的序列*dadad*），字母*dal*只有一个形状，而无论其处于词中的什么位置。

在开源协议（例如GPL或OFL下的字体允许用户做出修改。如果你修改了一个基于一个开源协议的字体然后分发它，那么你必须保持原作者的版权条款和许可信息，尽管你可以在你的分发版本的版权条款后添加你的条款。



本章介绍为一个阿拉伯字体添加一个字形。我们将使用的字体是Graph，我们将添加的字形是*peh*（U+067E），它在阿拉伯字体中并不出现，但是在使用阿拉伯书写字母的一些语言中指定*p*（阿拉伯书写字母的字形全列表参见Unicode图表）。



制作字体的工作副本

从网页下载并字体并解压。运行FontForge并加载字体。将其保存为*sfd*，在保存前编辑建议的名字来读取**GraphNew.sfd**。

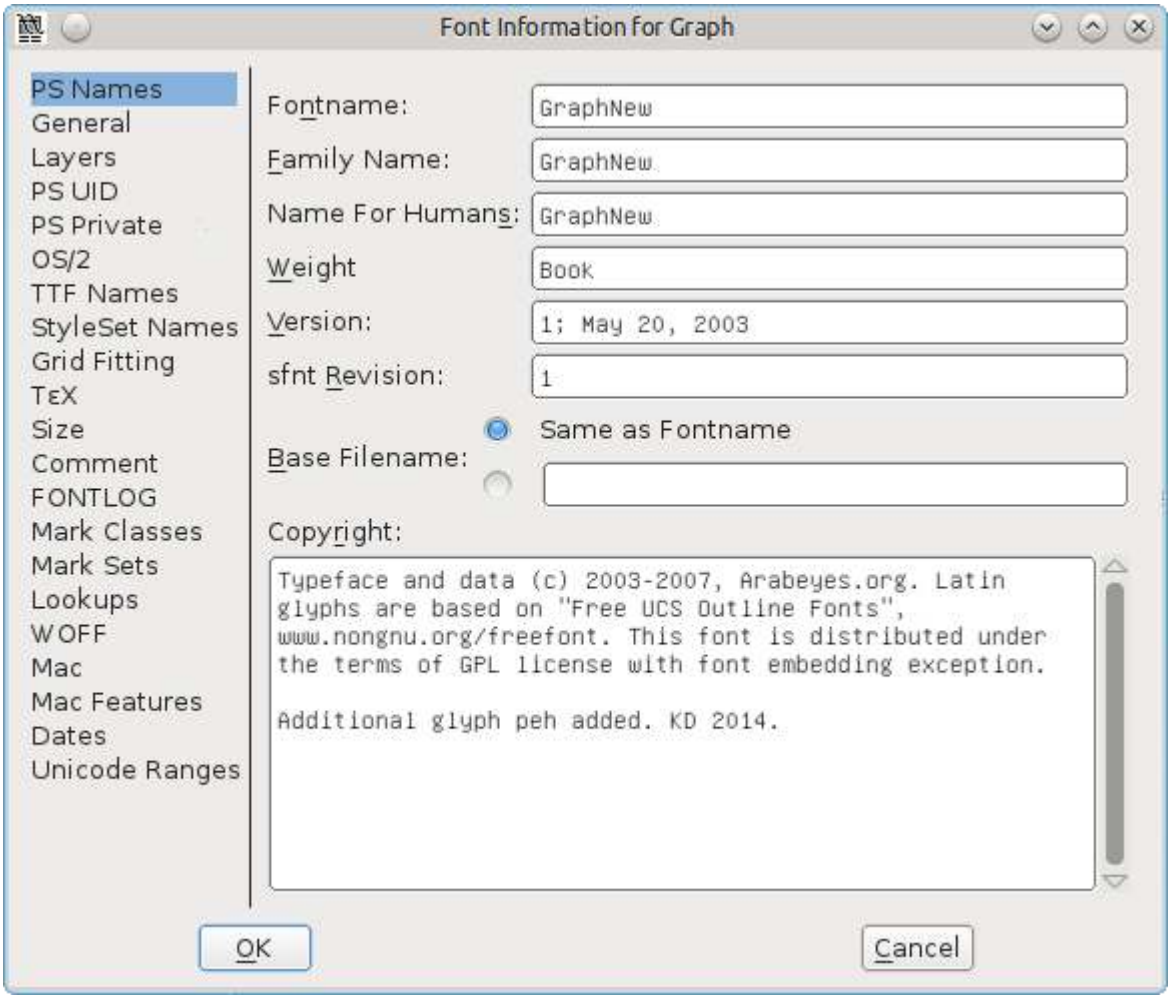
重命名字体

为什么要重命名字体?

如果你不重命名字体，你修改的字体将不会与原始的区分开安装 — 你将不得不首先卸载原始字体。如果你准备分发你的修改版的时候重命名字体也是明智的 — 如果字体原作者使用Reserved Font Name（RFN）机制保留了字体名，那么原始名字只能用于原作者的字体版本。

修改名称数据

选择**Element -> Font Info**，在*PS Names*面板下将*Fontname*，*Family Name*和*Name For Humans*修改为**GraphNew**。



如果有必要，你可以在*Copyright*一项的文本后添加一条信息“Additional glyphs added by”。

在*TTF Names*面板下的，t*Family*和*Fullname*的名称取自*PS Names*一项，应该显示的是*GraphNew*（你不能直接编辑他们）。将*Preferred Family*和*Compatible Full*两项修改为**GraphNew**。现在如你所希望的那样，这些名称的修改将会允许你将字体与原始字体安装在一起。

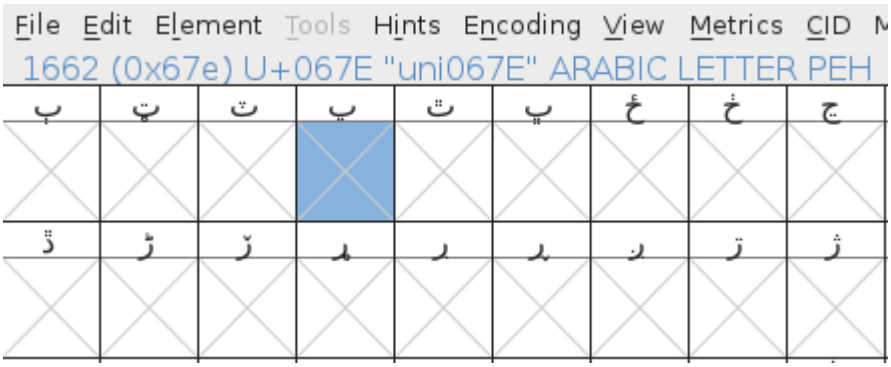
如果有必要，你可以在*Designer*一项的文本后添加一条信息“Additional glyphs added by”。

点击**OK**来保存修改。你将会看到一条关于为字体生成一个新的UniqueID（XUID）的消息 — 点击**Change**。

为*peh*的孤立形式添加字形

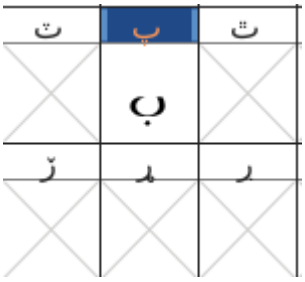
打开字体图表的阿拉伯区域：选择**View -> Go to**，点击下拉框并选择**Arabic**，然后点击**OK**。

点击字体图表中的一个单元格将会在面板顶部用蓝色显示其Unicode数字和名称。转到位置1662，将会显示蓝色的*1662 (0x67e) U+067E "uni067E" ARABIC LETTER PEH*。引用的字形下面的单元格包含一个灰色的X，意味着字体并不包含这个字形。



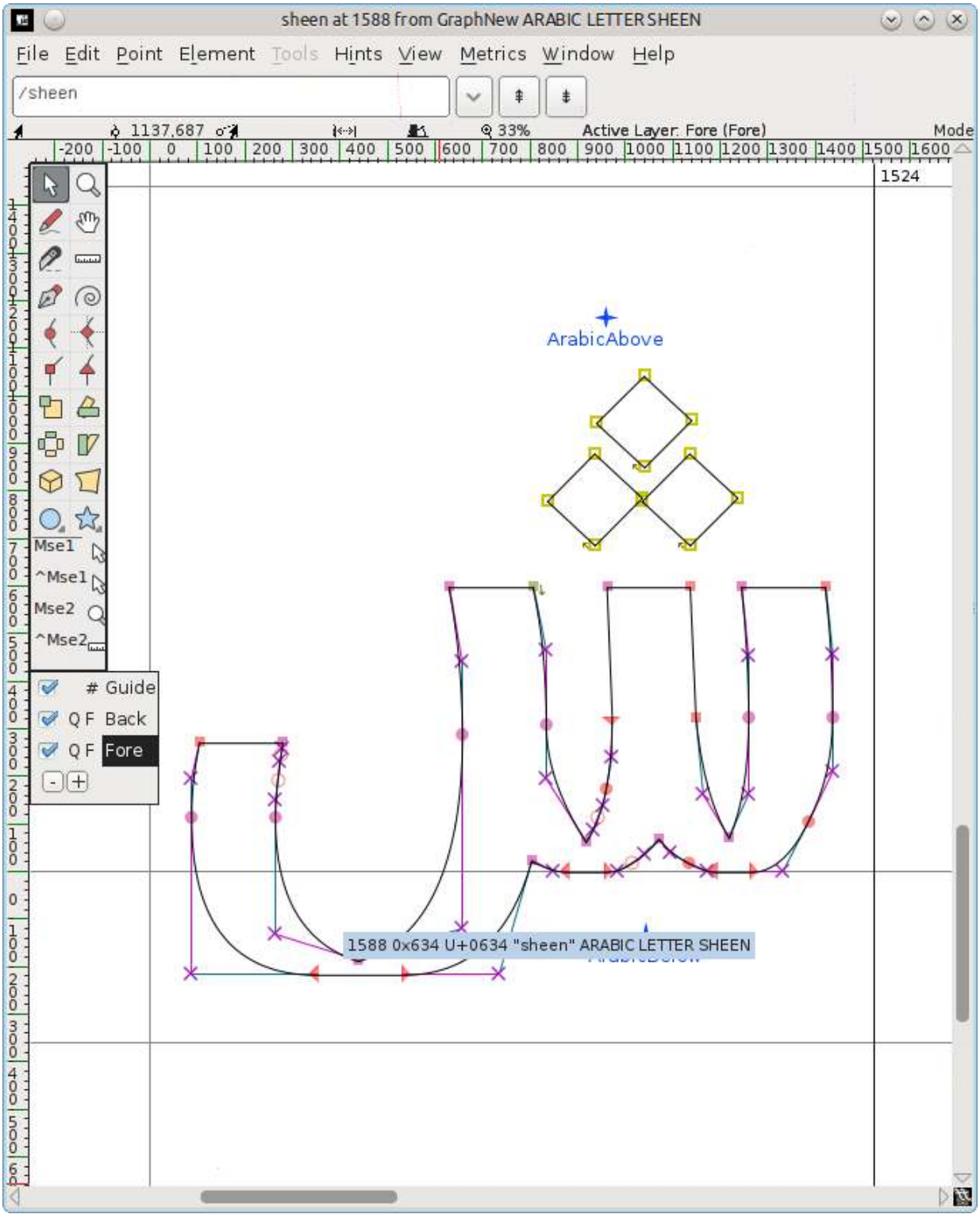
我们将通过复制*beh*（U+0628）并将其一个点替换为三个点来制作*peh*。

点击*beh*单元格（位置1576），然后右击并选择**Copy**。然后右击*peh*单元格并选择**Paste**。现在*beh*被复制进*peh*单元格，接下来要修改的是点。



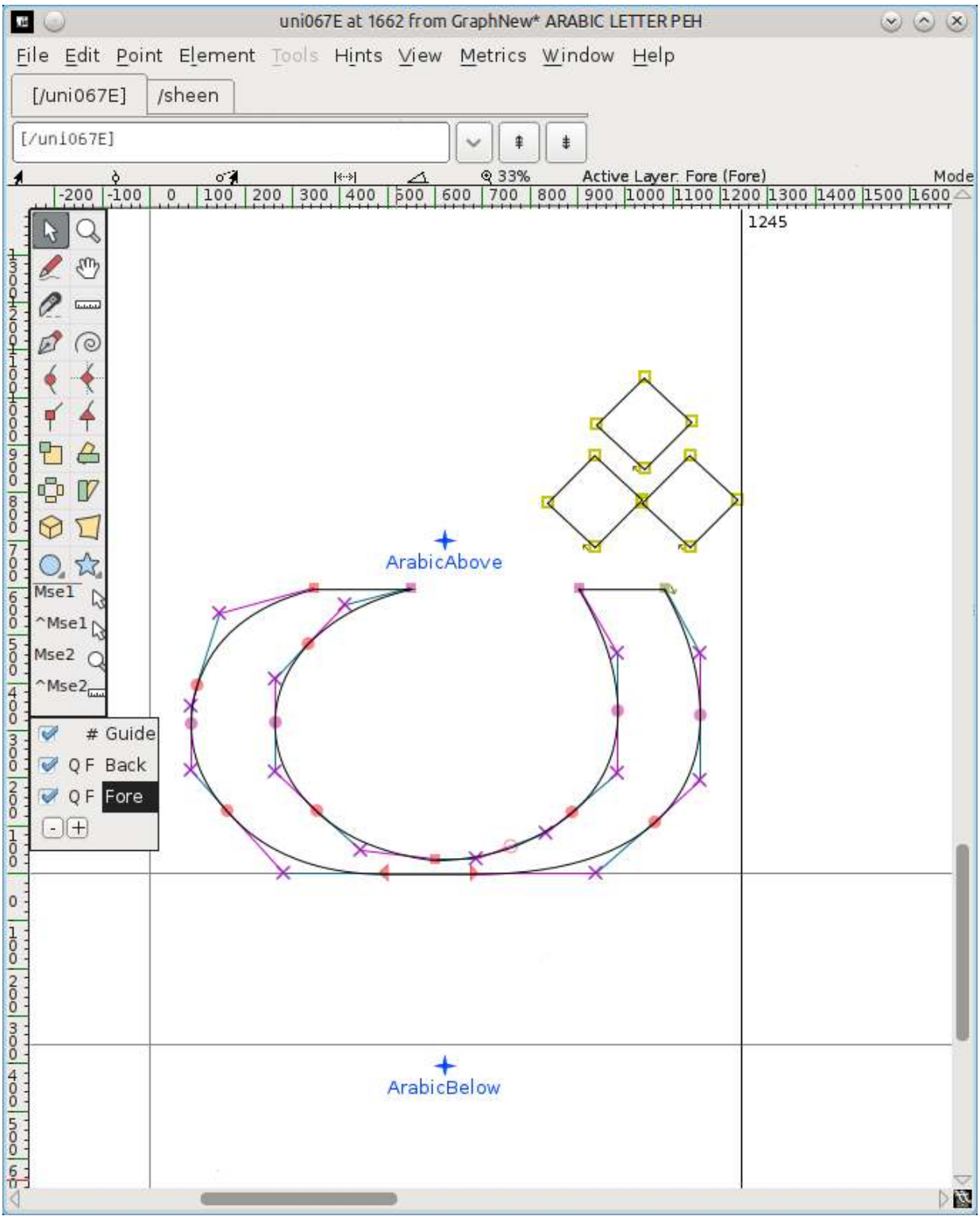
找到一个包含三个点的字形 — *sheen*（位置1588，U+0634）符合。双击这个单元格 — 将会打开一个字形设计面板。按V来确保工具箱的指针工具（剪头）选中，按Z扩大面板来给你一个字形的好的视图。

点击拖动sheen上的三个点的节点，颜色从粉色变为米色。如果你意外地包含或者忽略了一个节点，那么取消选择或者通过按`Shift`并点击来选择。按`Alt` + `C`来复制。

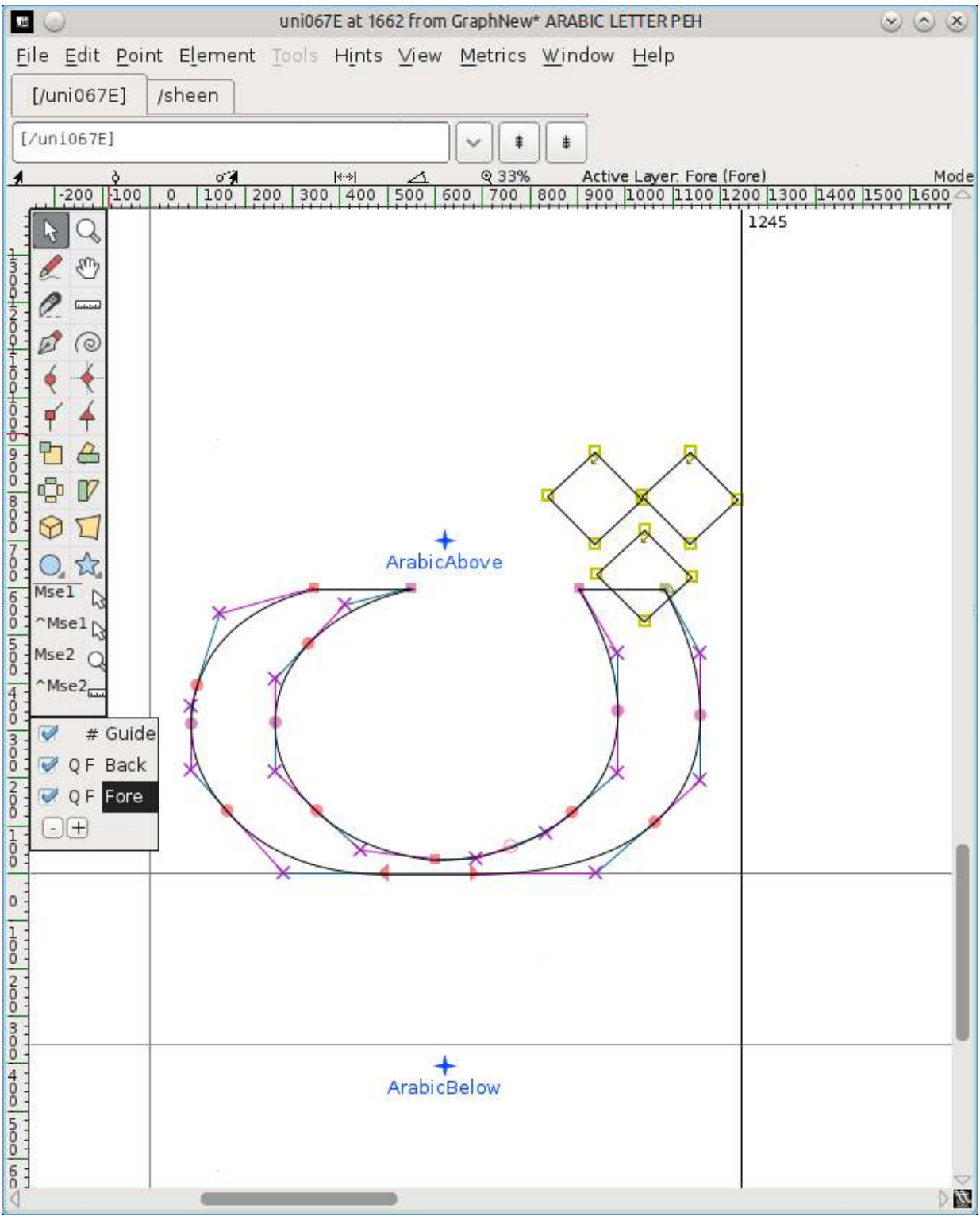


返回字体表格并双击*peh*单元格 — 这会将*peh*加载到字形设计面板*sheen*选项卡旁边的另一个选项卡。

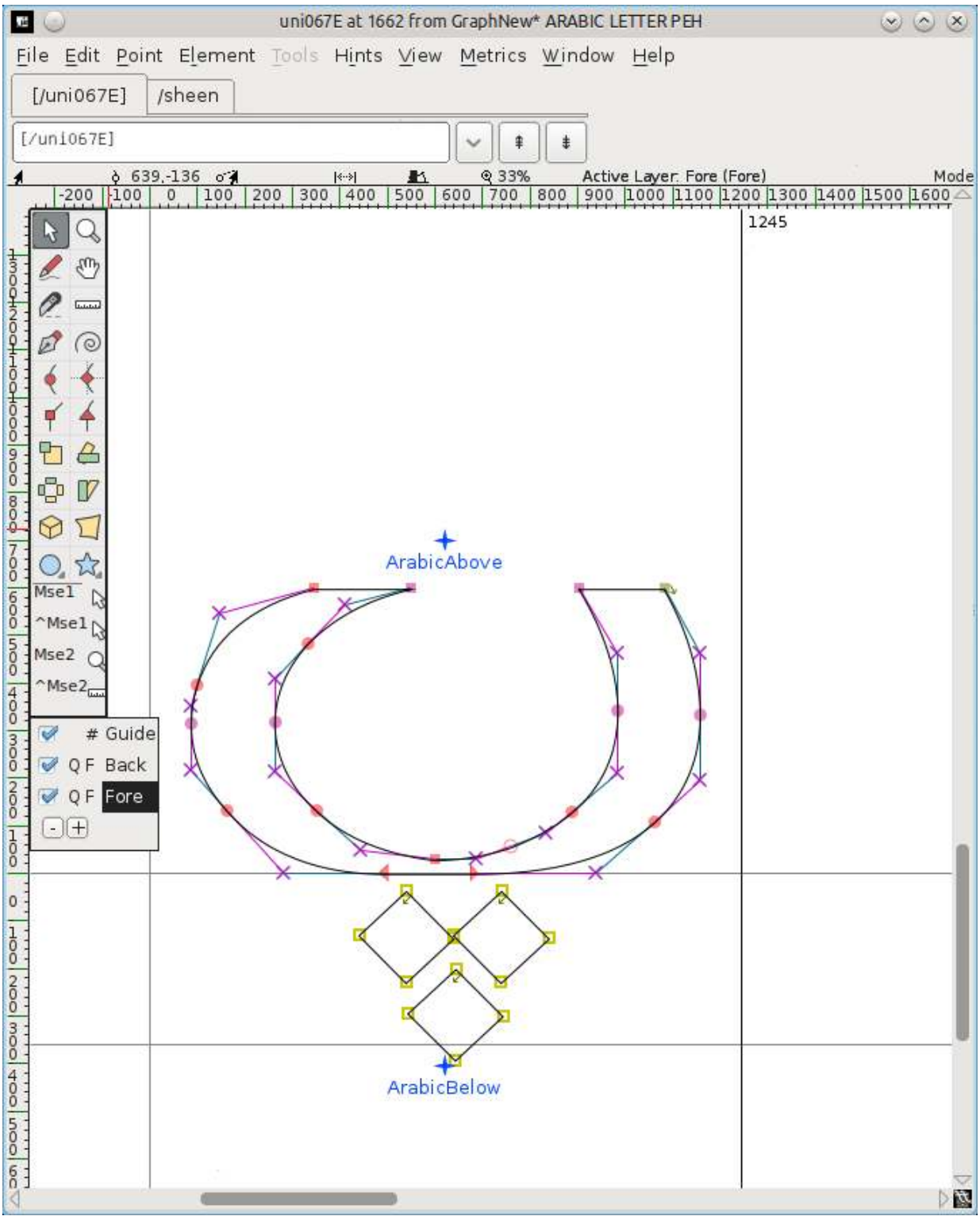
点击拖动来高亮*peh*下面的点，然后按`Delete`。按`Alt + V`来粘贴三个点，很可能出现在*peh*主体的上面。留下高亮的点的节点，这样你可以很容易地翻转或者移动它们。



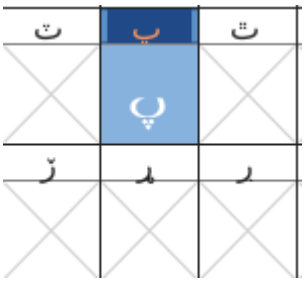
翻转点：从工具箱选择翻转工具（两个三角形中间有一条红色虚线）。（也可以右击点的中间，从弹出菜单中选择 **Flip the selection**。）点击点的节点中的一个，轻微向左或右拖动鼠标。



移动翻转的点：按V来再次选择指针工具，点击点的节点中的一个，然后将其拖动到字形主体的下面。将他们放在*ArabicBelow*标记上的中间位置。



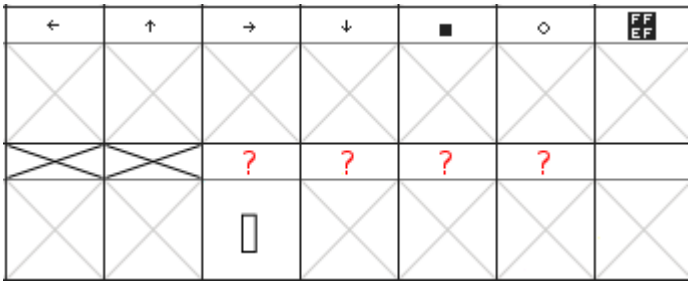
关闭字形设计面板。现在字体表格中的*peh*应该有一个新的字形。保存修改的字体（**File -> Save**）。



为*peh*的连接形式添加字形

但是这只是字形的孤立（独立）形式。如果你尝试使用你修改过的字体，你将会发现开头、中间和结尾形式并不可用。它们必须被分开制造。“这些形式作为未编码字符（FontForge约定编码是-1的字形）来构建。它们没有预定义的位置。”（Khaled Hosny）

选择**Encoding -> Add Encoding Slots**并输入你想要的字形的数量 — 在这种情况下是**3**。FontForge将会在字体的后部添加同样数量的位置，你将会被移动到字体表格中那个位置。最后三个单元格（位置65537，65538，65539）的引用字形处有一个问号，在这些单元格中你将会通过重复上面的流程添加未编码字形。

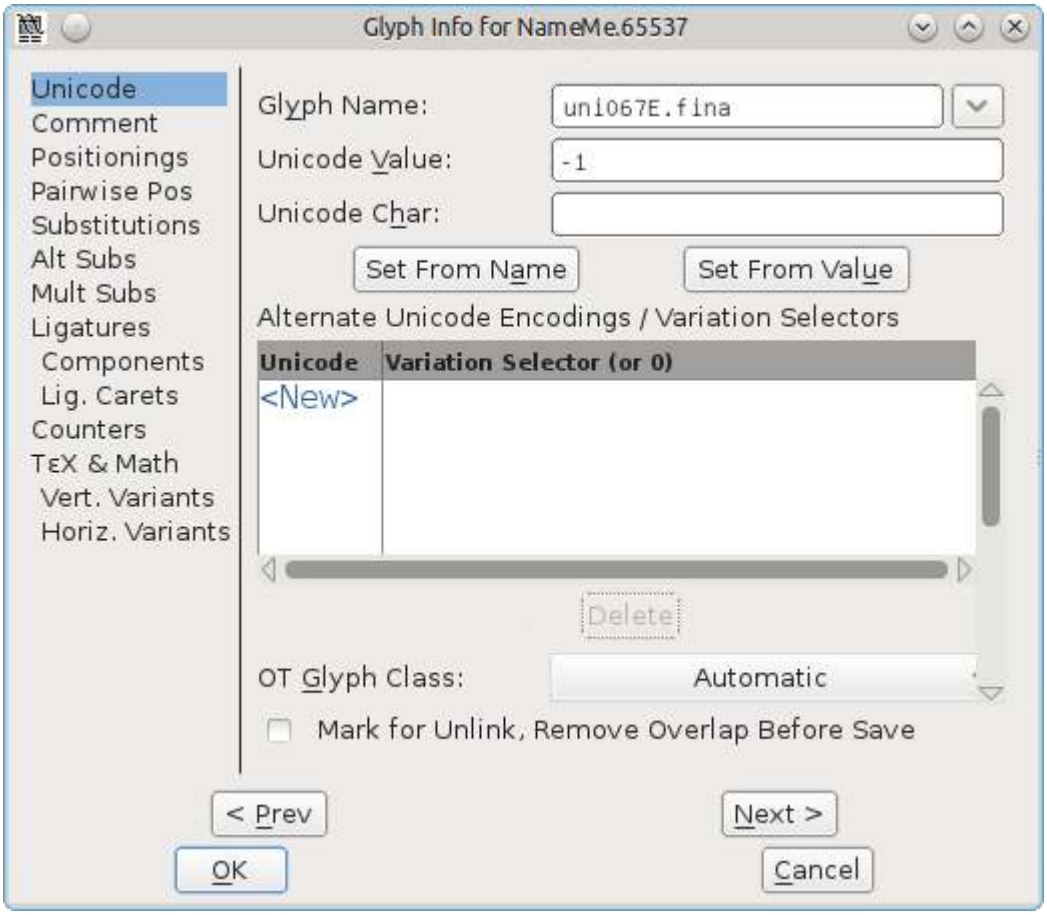


需要注意的是如果你在字体表格仍然拥有叫点的时候错误地开始输入，你会移动到顶部的European区域。要回到底部，选择**View -> Go to**，点击下拉框并选择**Not a Unicode Character**，然后点击**OK**。

创建最终形式

向上滚动一点字体表格直到你向前到达位置65152（U+FE80）的一系列阿拉伯字形。在U+FE90（位置65168）你讲看到一个*behfinal*字形 — 点击它并按`Ctrl`+`C`来复制它。向下滚动到表格的倒数第三个单元格（位置65537），点击它，并按`Ctrl`+`V`来粘贴*behfinal*字形。

右击单元格并选择**Glyph Info**。命名规范是使用孤立字形的数字 + 一个形式的后缀，因此将*Glyph Name*修改为**uni067E.fina**，然后点击**OK**。引用单元格的问号将会修改为*peh*。



得到三个点：双击*sheen*（U+FEB5）来将它加载到字形设计面板，选择三个点并按`Ctrl + C`。

双击新的*pehfinal*来将它加载到字形设计面板，点击拖动来高亮点的节点并按`Delete`。

`Ctrl + V`来插入来自*sheen*的三个点，翻转它们，将它们移动到字形主体以下的位置。按来保存修改过的字体表格。

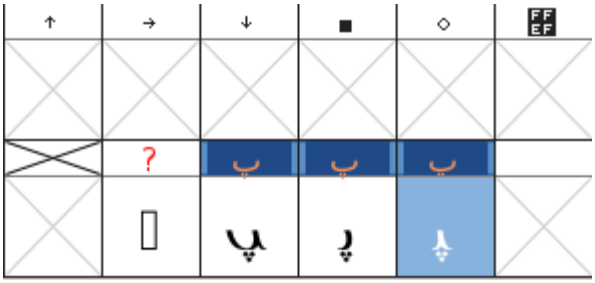
创建开头和中间的形式

复制初始形式U+FE91（位置65169）到倒数第二个单元格（位置65538），删除单个点并粘贴三个点。

右击单元格，选择**Glyph Info**，将*Glyph Name*修改为**uni067E.init**，并点击**OK**。

复制中间形式U+FE92（位置65170）到最后一个单元格（位置65539），删除单个点并粘贴三个点。

右击单元格选择**Glyph Info**，将*Glyph Name*修改为**uni067E.medi**，并点击**OK**。



选择**File -> Save**来保存修改过的字体表格。

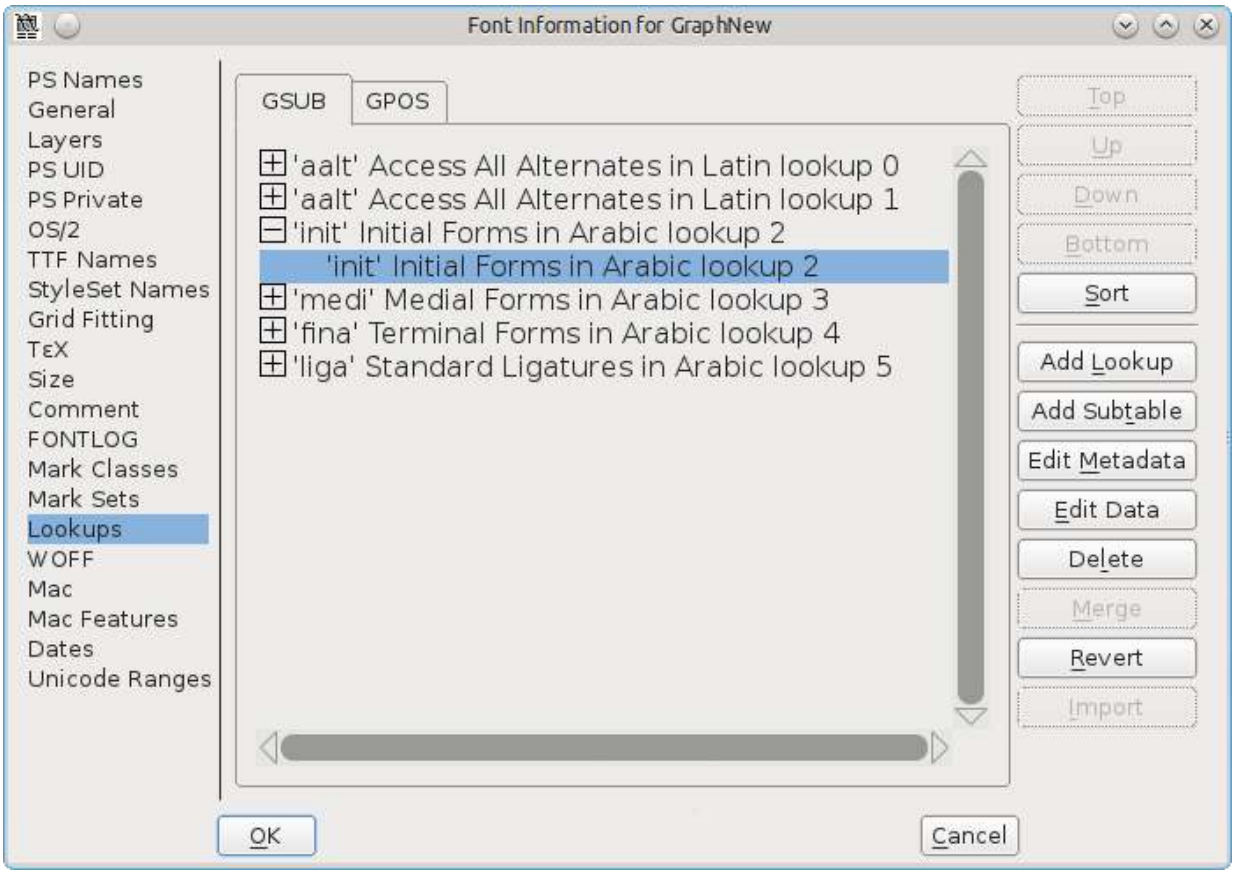
添加查找

孤立的形式必须被映射（连接）到它的开头、中间和结尾形式。

选择**Element -> Font Info -> Lookups**。

点击*'init' Initial Forms in Arabic lookup 2*旁边的**+**。这将会打开同名的子菜单。点击这个子菜单。

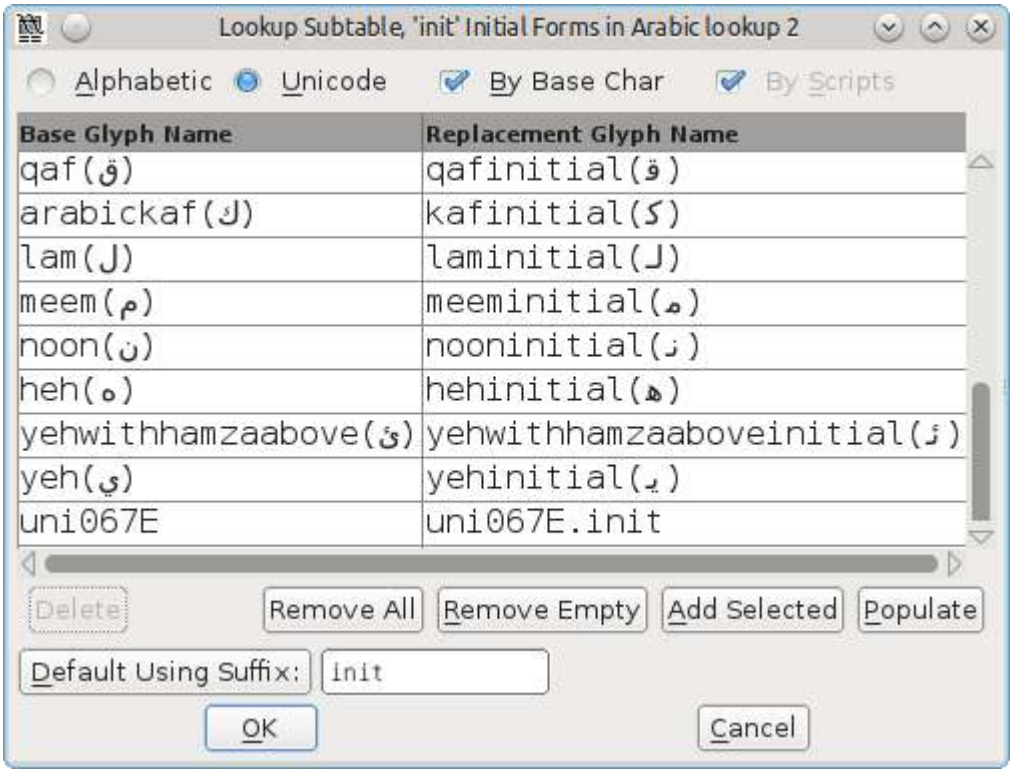
右边的*Edit Data*按钮现在将会变成可用 — 点击它。



在弹出的*Lookup Subtable*面板中，确保*Unicode*按钮勾选。将字符列表向下滚动知道最后。

在*Default Using Suffix*旁边的输入框里，输入相关的后缀（在这种情况下是**init**），然后点击**Default Using Suffix**。

一个新的映射将会被添加到字符列表，从uni067E（*peh*的孤立形式）到uni067E.init（初始形式）。点击**OK**。



对*'medi' Medial Forms in Arabic lookup 2*和*'fina' Terminal Forms in Arabic lookup 2*下的子菜单做同样的操作，选择*medi*和*fina*作为相关的后缀。

再次点击**OK**来关闭面板，并保存字体表格（`Ctrl + S`）。

需要注意的是*Default Using Suffix*看起来只能工作在Unicode 06（*Arabic*）块 — 在Unicode 07（*Arabic Supplement*），比如带两个点的*ain*，可能必须通过点击带行标记的*New*并输入名称来手动添加。

生成修改的字体

选择**File -> Generate Fonts**。

在*PS Type 1 (Binary)*的下拉框中选择**TrueType**，并检查文件名是*GraphNew.ttf*。

跳转到你想要保存字体的地方，然后点击**Generate**。在弹出的两个信息消息中点击**Yes**和**Generate**。

然后你可以使用一般字体安装步骤来安装修改的字体。然后新的字形*peh*可以与已有的字形在相同的本章开始提到的无意义的例子中一起使用：

注意：如果你在LibreOffice中使用一个字体并修改了这个字体，那么你需要重启LibreOffice来看到任何改变 — 否则它将使用之前的字体而不是改变了的。

感谢Khaled Hosny对使用FontForge编辑阿拉伯字形的建议。

延伸阅读

视频

- Optical Compensation by Thomas Phinney and FontLab

网站

曲线数学

- http://pomax.github.io/bezierinfo/ Excellent guide to the math of Bezier curves
- http://raph.levien.com/phd Raph Levien's PhD on Spiro curves

拉丁

- Letter-by-letter tutorials

西里尔（Cyrillic）

- Learn Cyrillic
- Serbian Cyrillic
- Bulgarian Cyrillic
- Paratype Language Help - Cyrillic encodings
- Cyrillic Typography on Facebook
- http://luc.devroye.org/cyrillic.html
- The ATypI Cyrillic SIG mailing list

阿拉伯

- Khaled Hosney's FontForge Guide

天城体

- Microsoft Specifications for Devanagari Fonts
- South Asia Language Resource Center The University of Chicago
- Professor Mahendra Patel
- [Dhruvi Tolia Graduation Report] (http://issuu.com/dhruvi/docs/graduation_report)
- Sanchit Sawaria Project 3 Document
- Yashodeep Gholap - Designing a Devanagari text font for newspaper use
- Mustafa Saifee - Devanagari Font for Optical Character Recognition
- Vaibhav Singh - Devanagari in multi-script typography
- Virtual Keyboards: lexilogos and branah

书籍

http://www.type-library.com有广泛的图书列表

标题: Detail in Typography (Paperback)

作者: Jost Hochuli (Author)

ISBN: 9780907259343

标题: Adrian Frutiger - Typefaces: The Complete Works

作者: Heidrun Osterer, Philipp Stamm, Swiss Foundation Type and Typography

ISBN: 9783764385811

标题: Printing Types: Their History, Forms, and Use: A Study in Survivals (with ILLUSTRATIONS)

作者: D. B Updike, Daniel Berkeley (Authors)

标题: Creative Characters Format: Flexibound

作者: Jan Middendorp (Editor)

ISBN: 9789063692247

标题: Now Read This: The Microsoft Cleartype collection

作者: John D. Berry (Author)

标题: The Stroke: Theory of Writing (Paperback)

作者: Gerrit Noordzij (Author)

ISBN: 9780907259305

标题: Shaping Text: Type, Typography and the Reader

作者: Jan Middendorp (Author)

ISBN: 9063692234

标题: Thinking with Type: A Critical Guide for Designers, Writers, Editors, & Students (Design Briefs)

作者: Ellen Lupton (Author)

ISBN: 9781568984483

标题: LETTERS OF CREDIT: A view of type design

作者: Tracy, Walter (Author)

ISBN: 0879236361 / 0-87923-636-1

标题: The Elements Of Typographic Style: Version 3.1

作者: Bringhurst, Robert (Author)

ISBN: 0881792063 / 0-88179-206-3

标题: Type: The Secret History of Letters

作者: Simon Loxley (Author)

ISBN: 1845110285

标题: Type Designs

作者: AF Johnson (Author)

标题: Typography: Macro & Microaesthetics

作者: Willy Kunz (Author)

标题: ISBN: 3721203488

标题: Fonts and encodings

作者: Yannis Haralambous (Author), P Scott Horne (Translator)</tt>

ISBN-10: 0596102429 | ISBN-13: 978-0596102425

标题: The Unicode Standard, Version 7.0.0

作者: The Unicode Consortium (Author)

ISBN: 978-1-936213-09-2

术语表

A

Abjad

辅音音素文字（Abjad）是一个闪语族（Semitic）语言使用的书写系统类型（希伯来文、阿拉伯文等）的技术术语，有表示所有辅音的字形但是读者必须准备好猜测两个辅音之间添加什么元音。

希伯来文与阿拉伯文有可选的元音标记，被称作“不纯的”辅音因素文字。

参见：字母表，元音附标文字，音节字母和相关的维基百科条目（http://en.wikipedia.org/wiki/Abjad）。

Abugida

元音附标文字（Abugida）是字母表和音节字母之间的地方。印度语言书写系统可能是最著名的元音附标文字。

在大多数元音附标文字中有独立的辅音字形，每个辅音都隐含地后面跟着一个默认的元音发音。所有的元音除了默认的都将被标记变音符号或一些其他基本辅音的修改。

一个元音附标文字与音节字母区别在于代表以给定的辅音为开始的音节的图片（也就是辅音的字形）有一个共同主题，而音节字母中即使两个以共同辅音为开始的音节字母其音节也不同。

一个元音附标文字与辅音音素文字不同之处在于元音（除了默认的）必须标记在在元音附标文字上。

参见：字母表，辅音音素文字，音节文字和相关的维基百科条目（http://en.wikipedia.org/wiki/Abugida）。

Accent（重音）

参见变音符号

Advance Width（步进宽度）

当前字形的开始与下个字形的开始之间的距离。有时候被称作是字形的宽度。参见竖直步进宽度。

Alphabet

全音素文字（Alphabet）是一种拥有所有音素的字形书写系统——类似于辅音和元音——并且（无论如何在理论上）一个词中所有的音素会以一个适当的字形标记。

参见：辅音音素文字，元音附标文字，音节文字和相关的维基百科条目（http://en.wikipedia.org/wiki/Alphabet）

Apple Advanced Typography

Apple对基本TrueType字体的扩展。包括上下文的替换，连字，字距紧缩等。也包括变形的字体。

Arm

字母悬挂离开右边的部分。

Ascender

升部（Ascender）是小写字母超出x高度的字干。“l”有一个升部。

参见x高度，大写高度，降部，上突，基线。

Anchor Class（锚类）

用来指定标识符到基本字形和手写体GPOS子表。

Ascent

在传统的排印中字体的顶部是从区块的顶到基线的距离。

在现代排印中得确切意义似乎在不同的定义者中会变化。

ATSUI

苹果的先进的排印系统。也称为Apple Advanced Typography。

B

Baseline

基线（Baseline）是字母（拉丁字母、希腊字母、西里尔字母）放置的水平线。基线在不同的书写字母中可能有不同的位置。在印度语书写字母中多数字母下降超过基线。在CJK书写字母中竖直基线通常在字形的中间。BASE与bsln表允许你指定不同书写字母的基线相对于其他应该如何对齐。

参见x高度，大写高度，顶部，底部，上突。

贝塞尔曲线或者贝塞尔样条曲线

贝塞尔（Bézier）曲线在手册主要部分的贝塞尔一章。

Bidi

双向字符集文本。这是包含从左到右和从右到左书写字母的文本区域。例如英文文本引用阿拉伯语。嵌套引用可能让事情更加复杂。Unicode标准包含了一个放置Bidi文本的算法。参见：Boustrophedon。

Black letter

黑体（Black letter）是基于中世纪手写的任何类型字体家族。

参见哥特体（gothic）。

BMP（Basic Multilingual Plane，基本多语种平面）

Unicode的前65536码点。它们包含了现代世界中大多数的普通字符。参见

- SMP: Supplementary Multilingual Plane (0x10000-0x1FFFF)
- SIP: Supplementary Ideographic Plane (0x20000-0x2FFFF)
- SSP: Supplementary Special-purpose Plane (0xE0000-0xEFFFF)

Bold

粗体（Bold）是一个常见的字体样式。字形的字干比普通字体更宽，给字体以较暗的印象。粗体是几个容易地转换成其他书写字母的LGC样式中的一种。

Bowl

弧（Bowl）是字母的圆的部分。

Bopomofo

注音符号（Bopomofo）是一个（1911~现代）汉语（普通话）字母表，用来在字典中表示汉字的注音。

Boustrophedon

牛耕式转行书写法（Boustrophedon）指的是“像牛耕一样”书写，书写方向从左到右和从右到左交替。早期字母表（老迦南语和非常早的希腊语（和出人意料的fuþark））使用它。从右到左的字形常常是从左到右的字形的镜像。据我所知，没有现代书写系统使用这种方法（OpenType也对它没有任何支持）。参见Bidi。

C

Cap-height

大写高度（Cap-height）是一个大写字母在基线上的高度（一个顶部平的字母比如“I”与弯曲的字母比如“O”截然相反）。

参见X高度，顶部，底部，上突，基线。

CFF

紧凑字体格式（Compact Font Format）在OpenType postscript字体内使用最普遍，即使没有SFNT包装也是一个有效的字体格式。这是带有PostScript Type2 CharStrings的字体的原生字体格式。

Character

字符（Character）是柏拉图式的理想具体化到至少一个字形中。例如字母“s”是一个具体化到几种不同字形的字符，包括“S”，“s”，“*s*”，“ß”，长s等。需要注意的是这些字形可能看起来互相非常不同，但是尽管积分符号的字形可能与长s字形相同，但它们是不同的字符。

Character set

字符集（character set is）是字符的一个无序集合。

CID

字符识别符（Character Identifier），一个数字。在一些CJK PostScript字体中，字形并没有命名但是通过字符识别符数字来引用。

以CID为键的字体（CID-keyed font）中的字形通过CID而不是名称来索引。

CJK

中文，日文，韩文（Chinese，Japanese，Korean）。这三种语言的需要字体带有数量庞大字形。这三者使用了相同的基于汉字的书写系统（尽管它们在各自国家经历了分割演化，中国大陆的字体也确实与台湾和香港所使用的不同）。

日文和韩文也有语音音节表。日文有两个音节表，平假名和有约60个音节的片假名。韩文有一个音节表，有数万个音节的韩文（Hangul）。

CJKV

中文，日文，韩文，越南文（Chinese，Japanese，Korean，Vietnamese）。这四种语言需要字体有数量庞大的字形。

Condensed

紧缩（Condensed）字体的字形的字干之间的间距和字形之间的间距被缩减。

Conflicting hints

如果一个字形包含两个提示，其中一个的开始或者结束点在另一个的范围内，那么这两个提示冲突。它们可能不会同时有效。

Counter

一个字形的对立面（Counter）是字形中完全或部分闭合的白色部分。o和n都有对立面。i和l没有。e有对立面。B有两个对立面。

D

Descender

降部（Descender）是扩展到基线以下的小写字母的字干。“p”有一个降部。

参见x高度，大写高度，顶部，上突，基线。

Descent

在传统排印中字体的下降是从字体块的底部到基线的距离。

这意味着在现代排印中变得精确度更低。

Device Table

设备表（Device Table）是OpenType中的一个概念，允许您输入间距的调整以准备好栅格化到特定的像素尺寸。如果一个在大多数时间都有效的字距值在一个12像素字体的字形中排列起来很丑，那么你可以添加一个特别的 到只应用在12像素的间距上（另一个是14，18，或者你需要的任何大小）。类似的功能锚标记也需要。

Diacritics

许多语言采用了顶部或底部有标记字母，有时标记甚至穿过字母。这些标记被称作变音符号（Diacritics）。有时它们也被称作“重音（accents）”，尽管这是个不太准确的术语。这些字母的例子包括À à å Ü ü Ø ø Ç ç。

Didot point

迪罗磅（Didot point）是欧洲的一种磅。每23.566毫米是62 2/3磅（2.66磅/毫米或者67.55磅每英寸）。也有“公制”的迪罗磅：.4毫米。

Distortable font（变形字体）

参见多主字体。

E

em

一个线性单位，等于字体的磅值。在一个10磅的字体中，em将会是10磅。em-space是与磅值一样宽的空白空间。em-dash是一个与磅值一样宽的水平的条形。

em-square是一个每个边都是1em的正方形。在传统排印中（此时每个字母被铸造在金属上）字形必须绘制在em-square中。

em unit

在一个可伸缩的字体中，“em”细分到单位。在一个Postscript字体中通常em有1000个单位。在一个TrueType字体中em可能有512，1024或2048个单位。在一个Ikarus字体中有15000个单位。FontForge使用这些单位作为其坐标系统的基础。

en

“em”的一半

Encoding

编码（Encoding）是一个字节码集合到一个字符集之间的映射。它确定了什么字节串代表什么字符。名词“编码”和“字符集”经常作为同义词使用。ASCII规范同时指定了一个字符集和一个编码。但是CJK字符集经常会一个字符集有多个编码（并且一些编码有多个字符集）。

在更复杂的情况下，有可能有多个字形关联到每个字符（在阿拉伯文中大多数字符有至少4个不同的字形）并且客户端程序必须选取适合当前上下文字符的字形。

Eth -- Edh

旧日尔曼语字母“ð”对应浊音的（英文的）“th”发音（“this”中的发音 — 大多数说英语的人甚至不知道英语中的“th”有两个发音关联到它，但它确实如此，参见Thorn）。

Even-Odd Fill rule

为了使用奇偶填充规则（Even-Odd Fill rule）确定一个像素是否应该填充，从一个点到无穷远（在任何方向上）绘制一条线，然后数轮廓穿过这条线的次数。如果数字是奇数，那么填充这个点，如果是偶数，那么不填充这个点。这个方法被2.0级别以后的Postscript栅格化用在字体上。参见Non-Zero Winding Number Fill。

Extended

一个扩展（Extended）字体是一种字形的字干之间的间距和字形之间的间距增加的字体。

Extremum *(plural:* Extrema *)*

极值（Extremum）是数学上的曲线上的一点，曲线在这点达到其最大值或最小值。在一个连续曲线上，这个点会出现在端点（是钝的）或者dx/dt=0或dy/dt=0处。

在字体设计中，字形的极值是轮廓的最高或最低点，也包括其最左点和最右点。确保一个字形所有极值处都有曲线上的点是重要的，因为在字体使用时它简化了文本渲染。

F

Features (OpenType)

创建用于复杂文字（或者比较不复杂的文字）的字体时各种各样的变换（如连字）必须在准备用于显示之前被应用到输入的字形。这些变换被定义为字体功能并以4个字母（OpenType当中）或2个字母（苹果当中）为标识打上标签。这意味着这些功能是由微软或者苹果预先定义的。FontForge允许你在创建时（或稍后）为每一个查询信息标以一个或几个功能。

Feature File

这是Adobe设计的用来描述OpenType功能的文本语法。他可以用来将功能和查询信息从一个字体移动到另一个。

Feature/Settings (Apple)

它们粗略地等于上面的OpenType的功能，它们是由苹果定义的。

Font

字形的一个集合，通常有至少一个字形关联到字体的字符集中的每个字符，通常还有带有编码。

一个字体包含将字节串转换成代表这些字节指定的字符的图片集所需的信息。

在传统的排字中，一个字体是小金属块的集合，每个上面都有雕刻字母的图像。传统上每个磅尺寸都有一个不同的字体。

Font Family, or Family

字体家族（Font Family，或者Family）是相关字体的集合。经常包含普通体，斜体和粗体样式。

FreeType

一个栅格化字体的库。在FontForge中广泛地用来理解TrueType字体的行为并能够比FontForge在不受帮助下完成的栅格化更好。

Fuþark (Futhark)

老日尔曼语的古代北欧书写字母。

G

Ghost Hint

有时指出水平边缘确实是水平的是重要的。但是边缘并没有制作普通的字干的相应的边缘。在这种情况下一个特殊的提示在-20（或-21）宽度下使用。一个幽灵提示（Ghost Hint）必须全部放置在一个字形总。如果它处在一个轮廓的顶部使用-20宽度，如果在底部则使用-21。幽灵提示也应该处在BlueZones中。

（这一规格也提到竖直的幽灵提示，但是由于没有竖直BlueZones，所以并不清楚它们是不是应该被使用）。

Glyph

一个字形是一张图片，通常关联到一个或多个字符。所以用来绘制“f”的字形关联到字符f，而连字“fi”的字形同时关联到f和i。在简单的拉丁字体中，关联关系通常是一对一（也就是一个字形恰好地关联到一个字符），而在更复杂的字体或手写中可能两个字形关联到每个字符（在文艺复兴时期的印刷中字母“s”与两个字形关联，一个是长s，用在开头和中间，另一个是短s，用在词的结尾）。在手写中一个字形关联到两个或更多字符。

字体是字形和一些从字符到字形的映射形式的集合。

Grid Fitting（网格适应）

在TrueType字形栅格化之前，它们要经过一个被称为网格适应的过程，会运行一个 很小的程序（关联到每个字形），这个程序将会移动字形轮廓周围的点直到它们更好地适应像素网格。

Gothic

德国僧侣在古腾堡生活的时代使用的一个黑体字书写样式，他在他的印刷字体样式中复制了它们的书写。意大利的字体设计者（在印刷术传播到南方以后）嘲笑这个样式，它们更喜欢罗马人留下的字体设计。它们使用表示轻蔑的哥特（Gothic）一词，帮助破坏罗马王朝的哥特人的样式。

Graphite tables

Graphite是TrueType的一个扩展，将几个表嵌入到一个字体中，包括上下文形状、连字、重排序、分割字形、双向、变音符号堆叠、复杂位置等的规则。

这听起来更像OpenType — 除了OpenType依赖于文本布局惯例来了解关于涉及字形的许多东西。这意味着OpenType字体在没有新版本操作系统推出的情况下不能设计用于一种新的语言或者手写。而Graphite表包含了所有这些隐藏的信息。

Apple Advanced Typography提供了更好的对比，但是Graphite表被认为更容易构建。

SIL International提供了一个自由的Graphite编译器。

Grotesque

参见sans-serif。

H

Han characters（汉字）

汉字，在中国、日本、韩国使用的表意文字（并且我认为在许多其他亚洲国家也是这样（越南？）），都基于在中国形成的书写系统。

Hangul

韩文音节表。基于一个字母表的唯一的音节表（无论如何我这么认为） — 字母表中的字母从不单独出现，只有两个或三个作为一组才能组成一个音节。

Hanja

汉字的韩文名称。

Hints

它们在手册主要部分详细描述了。它们帮助栅格化程序在小的磅值下绘制好字形。

Hint Masks

在轮廓上的任何给定点上提示（Hint）可能不会冲突。但是一个字形中不同点上可能有冲突的提示。因此偶尔一个轮廓可能改变那个提示是激活的。激活的提示的每个列表被称作提示遮罩（Hint Mask）。

Hiragana

两个日语音节表中的一个。平假名（Hiragana）和片假名（Katakana）有相同的发音。

I

Ideographic character

不必拼写出来就可以代表一个概念的单个字符。通常用来表示汉字。

Italic

斜体（Italic，也称意大利体）字体的一个倾斜的样式，用来表示强调。

斜体与倾斜（Oblique）不同之处在于从水平到倾斜形式所涉及的不仅仅是偏斜字体。通常小写的a修改为*a*，小写字母的衬线比如i（*i*）改变，并且字体通常获得更加流动的感觉。

J

Jamo

韩文字母表中的字母（Jamo）。它们几乎从不单独出现，通常作为Hangul音节的一部分以三个为一组出现。字母被分为三类（第一类和第三类有相当大的重叠），choseong — 初声，词首辅音，jungseong — 中声，中元音，jongseong — 终声，词尾辅音。一个音节通过将初声放在em-square的左上，中声放在右上，终声可选地放在下部来组成。

K

Kanji

汉字的日文名称。

Katakana

两个（现代）日文音节表之一。平假名（Hiragana）和片假名（Katakana）有相同的发音。

Kerning

当两个字形之间的默认间距不适合，字体可能包含额外信息来指出当一个给定的字形（比如“T”）后面跟着另一个字形（比如“o”）那么“T”的步进宽度应该调整一定数量来获得更加讨人喜欢的显示。这个过程就是字距紧缩（Kerning）。

在使用金属字体的时候，实际上不得不切削字体金属块上的金属来提供更整洁的适应。

Kern pair

字距紧缩信息所指定的一个字距紧缩对。

Kerning by classes

字体中的字形被分开到一些字形类别。有一张很大的表来指定列别之间所有可能的联合的字距紧缩。通常由于每个类别经常包含多个字形，所以这将会比设置字距紧缩对要小。

Knuth, Donald（高德纳）

他是一个数学家，对糟糕的排版感到如此厌倦以至于在二十世纪七八十年代创造了自己的字体设计系统排字布局程序，分别被称作MetaFont和TeX。

L

Left side bearing

左跨距（Left side bearing）从一个字形的原点到其最左扩展的水平距离。这可能是正数或负数。

Lemur

狐猴（Lemur）是原猴亚目的猴灵长类动物的一个单型属，目前只在马达加斯加发现，但是该科的正式成员（大约五千万年前）曾经更加广泛地传播。

Ligature

连字（Ligature）是由两个毗连的字形组成的单个字形。拉丁文手写中的一个常见的例子是“fi”连字比顺序排列感觉更好。

Linespace

行距是字体的连续的行之间的距离。

LGC

拉丁文，希腊文和西里尔文。这三个字母表在过去的几千年里一起进化。字母形式非常类似（一些字母是共享的）。许多概念比如“小写”，“斜体”可以应用在这三个字母表上而不能应用在任何其他上面（好吧，亚美尼亚语也有小写字母）。

M

Manyogana

一个早期的日语手写体，是平假名和片假名的祖先。万叶假名（Manyogana）使用汉字表示它们的发音，许多年后这些汉字简化为平假名和片假名。

Monospace

等宽字体（Monospace）是一种所有字形都有相同宽度的字体。它们有时也被称作是打字机字体。

Multi-layered fonts

多层字体是FontForge的自有术语。PostScript type3字体和SVG字体比普通字体拥有更多的绘制可能性。普通字体可能只能被来自图形环境的一种颜色填充。这两种字体可以被几种不同的颜色填充或描边，包括图片，渐变条虫等。FontForge可以配置为支持这些字体（由于需要占用更多的内存，它默认并不这么做）。

```
$ configure --with-type3
$ make
$ make install
```

Multiple Master Font

一个多主字体（Multiple Master Font）是一种定义了无穷的相关字体的一个PostScript字体模式。多主字体可以在沿着几个轴变化，例如你可能有一个定义了一个字体家族的不同粗细和不同宽度多主，它可以用来生成：细体，普通体，半粗体，粗体，紧缩字体，扩展字体，粗紧缩字体等。

Adobe不再开发这种格式。苹果有一种格式可以实现同样的效果但是不会产生许多例子。FontForge对两者都支持。

N

Namelist

名称表（Namelist）是Unicode码点到字形名称的一个映射。

Non-Zero Winding Number Fill rule

为了使用非零环绕数填充规则（Non-Zero Winding Number Fill rule）确定是否应该填充一个像素，从这点到无穷远绘制一条直线（沿任意方向）并计算轮廓穿过这条线的次数。如果轮廓沿着顺时针方向穿过这条线则加1，如果轮廓按逆时针方向穿过则减一。如果结果非零则填充像素。如果是零则留空。这个方法被TrueType和更老（版本2之前）的PostScript字体用来栅格化字体。

参见奇偶填充规则

O

Ogham

欧甘文（Ogham）是旧的凯尔特铭文手写体。

OpenType

字体的一个类型。是合并PostScript和TrueType字体到一个规范的一次尝试。

一个OpenType字体可能包含一个TrueType或者一个PostScript。

它包含了许多与TrueType相同的信息数据表，像编码。

令人困惑的是它也用来表示Adobe和微软（没有苹果）添加到TrueType中的先进的印刷表。这些包括上下文连字，上下文字距紧缩，字形替换等。

并且微软Windows使用它表示包含“DSIG”（Digital Signature）表的字体。

OpenType Tables

每个OpenType字体中都包含一系列的表(OpenType Table)，每一个表分别包含某一特定类型的信息。

Oblique

字体的一个倾斜样式，通常用来表示强调。

伪斜体（Oblique）与斜体（意大利斜体）不同之处在于从水平到倾斜形式涉及到一个数学上或机械上的偏斜字母形式。

Overshoot

为了“O”的弧形与“I”的平坦顶部看起来高度一致，它倾向于“上凸（Overshoot）”大写高度（或者x高度），或者将基线下凸（undershoot）大约大写高度（或x高度）的3%。对于一个三角形（比如“A”）来说上突甚至更大，或许是5%。

这些指引基于我们眼睛的工作方式和其产生的视觉错觉，来自Peter Karow的Digital Formats for Typefaces，第26页。

上凸也依赖于一个字体的磅值，磅值越大上凸应该越小。通常现代字体将会用在多个磅值上，但是在一些字体家族中对不同的磅值有不同的字体样式，在这一情况下上凸将会依样式不同而变化。

参见X高度，大写高度，顶部，底部，基线

P

Panose

描述字体的一个系统。参见HP's PANOSE classification metrics guide。也有一个扩展叫做Panose 2。

FontForge只知道拉丁字体的分类方案。其他方案为其他手写体而存在。

PfaEdit

这是FontForge的早期名称。最初的设想是它应该只编辑type1 ASCII字体（因此得名），它迅速演化超过了这一点，但是重命名花掉了我三年时间。

Phantom points

在TrueType字体中有一些添加到每个字形上的点，它们并不被组成字形的轮廓所指定。它们被称作幽灵点（Phantom points）。这些点中的一个代表左跨距，另一个是字形的步进宽度。TrueType说明（提示，hints）允许移动这些点，就像其他任何可能被移动的点 — 从而改变左跨距或者步进宽度。TrueType的早期版本只提供这两种幽灵点，较新的版本也支持上跨距幽灵点和竖直步进宽度的幽灵点。

Pica

宽度的一个单位（至少在美国）定义为35/83cm（或者大约1/6英寸）。它用来测量文本行高（比如“30picas和4磅长”），但是不是用来测量字体高度。

在文艺复兴排印中，在有磅之前字体的尺寸有有名称，“pica”用在这种上下文环境中。比如：“Great Canon”，"Double Pica”，"Great Primer”，“English”，“Pica”，“Primer”，“Small Pica”，“Brevier”，“Nonpareil”和“Pearl”（每个名称代表一个渐进更小的字体尺寸）和参见Wikipedia上Caslon的字体范例表。

Pica point

英美磅值。72.27磅美英寸（2.85磅/毫米）。

Point

磅是一个度量单位。在计算机诞生之前（至少）有三种不同的对“磅”的常用定义。一个用在Anglo-Saxon印刷世界的是72.27磅每英寸（2.85磅/毫米）的“pica point”，用在欧洲大陆的是62 2/3磅每23.566毫米的迪罗磅（2.66磅/毫米或者67.54磅/英寸），法国有时使用中值磅（Median point，72.78磅每英寸，2.86磅/毫米）。

迪罗磅和pica磅被安排成在两种系统下使用给定的磅值文本将会有近似相同的大写高度，迪罗磅将会在大写上有额外的空白来包含在大多数非英语拉丁文手写体中经常出现的重音。

这有有趣的副作用，为欧洲人使用而设计的字体的文本主体给出的竖直em应该有更小的比例。我认为计算机字体倾向于忽略这点，因此大概欧洲的打印机现在设置了更多行距。

如我可以告诉你的，计算机倾向于以pica磅的近似值工作（但是这可能是因为我在美国），PostScript使用1/72英寸的单位。

最初字体并不通过磅值来描述，而是通过名称。直到17世纪30年代Pierre Fournier创造了用于指定字体高度的磅系统。接下来这被François-Ambroise Didot改进（因此有了磅的名称）。在1878年芝加哥字体公司首先在美国使用磅系统。在1886年美国磅被标准化 — pica被定义为35/83厘米，pica磅被定义为它的1/12。

Point Size

在传统的排印中一个10磅的字体是每个字形的金属块都10磅高的字体。一个字体的磅值不插铅条的基线到基线的距离。

Point of inflection

曲线上的一个点，在此处它从下凹变为上凹（反之亦然）。或者在数学表示上（连续曲线上）$d^2y/dx^2=0$或无穷大处。

三次样条曲线可能包含拐点，二次样条曲线可能不包含。

PostScript

PostScript是许多打印机使用的页布局语言。语言包含集中不同字体格式的规范。手册主体（FontForge）有一节描述PostScript与TrueType如何不同。

- Type 1：这是PostScript字体的一个旧的标准。这样的字体通常有.pfb（或.pfa）的扩展名。一个Type 1字体被限制为单字节编码（比如只有256个字形可能被编码）。
- Type 2/CFF：这是一个OpenType字体内使用的格式。它几乎与Type 1相同，但是有一些扩展和一个更紧凑的格式。它通常处于OpenType字体经常包含的CFF包装内。同样CFF字体格式只允许单字节编码，但是OpenType包装扩展它来提供更复杂的编码字体。
- Type 3：这个格式在字体内完整支持PostScript，但是这意味着不允许有任何提示，因此这些字体在小的磅值下看起来不好。同时大多数（屏幕）栅格花程序并不能处理他们。一个Type 3字体被限制为单字节拜纳姆（比如只有256个字形可能被编码）。
- Type 0：这个格式用来将许多（Type 1，2或3的）子字体用多字节编码收集到一个大的字体中，用于CJK或Unicode字体。

- Type 42：一个包装在PostScript中的TrueType字体。某种程度上是OpenType的对手。
- CID：这个格式用于CJK字体，包含了大量的字形。字形本身被指定为Type 1或Type 2字形格式。CID字体本身没有编码，只是一个CID（一个数字）到字形的映射。一系列外部的CMAP文件用来提供所需的合适的编码。

Python

一种强调代码可读性的计算机编程语言。

R

Reference

一个引用（Reference）是在一个字形中保存另一个字形的一种方式（比如重音字形中的例子）。有时也被成为一个组件。

Right side bearing

右跨距（Right side bearing）是一个字形最右扩展到字形步进宽度的水平距离。它可能是正数或负数。

S

Sans Serif

参见Serif。

Script

文字（Script）是一个字符集和将字符放在一起的相关规则。拉丁文，阿拉伯文，片假名和朝鲜汉字都是文字。

Serif

回溯到两千年前罗马人在石碑上雕刻它们的字母的时候，它们发现可以通过在字形主干的结尾添加精细的线条来减少石头破裂的机会。

这些精细的线条被称作衬线（Serif），能够增加美的作用。早期的字体设计者将它添加到它们自己的字体是为了美观而不是功能性的原因。

在十九世纪末二十世纪初，字体设计者开始设计无衬线的字体。它们最初因为形式看起来如此奇怪而被称为畸形字体（grotesques），它们现在通常被成为无衬线字体。

其他书写系统（希伯来文算一个）有它们自己的衬线。希伯来文衬线与拉丁文（西里尔文或希腊文）衬线非常不同，我也不知道它们的历史。希伯来文衬线只出现在一个字形的顶部。

SFD

SplineFont Database。这些是FontForge自己的个人字体标示。文件是ASCII的并且含糊难读，这里描述格式。2008年5月14日格式在IANA注册为一个多用途互联网邮件扩展类型（MIME）类型： application/vnd.font-fontforge-sfd。

（不幸的是）其他人也使用首字母缩写“sfd”。

- Digital PDP-10迷你电脑的Tops-10操作系统使用sfd来表示“子文件目录”。Tops-10在被称作“用户文件目录”的顶层（Home）目录和子目录之间加以区别。

- TeX使用它来表示“子字体定义”，一个Tex的sfd文件包含如何为CJK或Unicode字体拆分成小的子字体的信息，每个都有Tex（或者旧版的Tex）需要的单字节编码。

SFNT

名称用来表示一般字体格式，包含TrueType、OpenType、苹果的位图、X11的位图、过时的“typ1”字体和Adobe的SING字体（毫无疑问还有其他）。SFNT格式描述字体表如何在一个文件中放置。每个上面格式都遵循这一一般想法但是包含了更多的具体要求（比如需要什么表和每个表的格式）。

SIP

Unicode的表意文字补充平面（Supplementary Ideographic Plane，0x20000-0x2FFFF）。用在罕见的汉字上（大多数平时不再使用）。参见

- BMP: Basic Multilingual Plane (0x00000-0x0FFFF)
- SMP: Supplementary Multilingual Plane (0x10000-0x1FFFF)
- SSP: Supplementary Special-purpose Plane (0xE0000-0xEFFFF)

SMP

Unicode的多文种补充平面（Supplementary Multilingual Plane，0x10000-0x1FFFF）。用在古代人造的字母表和音节表 — 像Linear B，Gothic和Shavian。参见

- BMP: Basic Multilingual Plane (0x00000-0x0FFFF)
- SIP: Supplementary Ideographic Plane (0x20000-0x2FFFF)
- SSP: Supplementary Special-purpose Plane (0xE0000-0xEFFFF)

Spline

样条曲线（Spline）是一条弯曲的线段。FontForge中使用的样条曲线都是第二类或第三类样条曲线（平方或立方）和Raph Levien的回旋样条曲线。

SSP

Unicode的特别用途补充平面（Supplementary Special-purpose Plane，0xE0000-0xEFFFF）。用得并不多。参见

- BMP: Basic Multilingual Plane (0x00000-0x0FFFF)
- SMP: Supplementary Multilingual Plane (0x10000-0x1FFFF)
- SIP: Supplementary Ideographic Plane (0x20000-0x2FFFF)

State machine

状态机（State machine）就像是一个非常简单的小程序，它们用在mac上来执行上下文替换和字距紧缩。状态机对话框可以从Element->Font Info->Lookups打开。

“状态机”包含了一个状态表，每个状态依次包含了一系列依赖于输入的潜在转换（到相同或不同的状态）。在字体内的状态机中，机器从一个被称作起始状态的特殊状态开始，读取文本中的字形流。每个独立的字形将会导致一个状态转换的发生。机器中发生的转换也可能指定字形流的改变（条件替换或紧缩）。

Stem

字干（Stem）是字母中竖直的部分。I和l除了衬线外都是字干。H由两个字干和一个横梁组成。其他包含字干的字形有B b F f K k P p R r 1 4。

Strike

删除线（Strike）是一个字体的特殊实例。最常见的位图删除线是字体的一个特殊的像素尺寸。

Style

一个字体有多种常规的变形。大概在任何书写系统中字形的字干的厚度可能是变化的，被称作是字体的粗细（weight）。常见的粗细是普通体和粗体。

在LGC字母表中一个斜体出现并用来表示强调。

字体经常被压缩为一个紧缩的样式，或者扩展成为一个扩展样式。

各种其他系统偶尔用到：underline，overstrike，outline，shadow。

SVG

可缩放矢量图形（Scalable Vector Graphics）。一个用于绘制矢量图的XML格式。它包含了一个字体格式。

Syllabary

音节表（Syllabary）像字母表一样，是一个语音上的书写系统。与字母表不同，书写的发音单元是音节而不是音位。在日文片假名中“ka”音由一个字形来代表。音节表倾向于比字母表大（日文片假名需要大约60个不同的字符，而韩文Hangul需要几万个）。

参见：abjad，abugida，alphabet和相关维基百科条目。

T

Terminal

一个字形的末端（Terminal）是末端的那部分。f的顶部有一个末端。s有两个末端。当一个字形有多个衬线时，衬线可能与衬线不同。因为如果f在一个衬线样式中那么底部将有一个衬线，那么不认为底部是一个末端。但是j和y的底部被认为是末端。同样3有两个末端，一个在顶部一个在底部。中部被认为是连接而不是末端。这些部分的分类可能更多地通过惯例来确定而非严格的逻辑。

TeX

一个文字排版程序包。

Thorn

用于不发音（英文的）“th”发音（正如单词“thorn”中）的德文字母“þ”，我认为这与希腊语Theta有相同的发音值。目前这个字形的损坏的版本存在于“the”中的“ye”。参见Eth。

True Type

苹果发明并分享给微软的一个字体。它用来指明轮廓的是二次（二次方）贝兹曲线，包含创新的提示控制和包含任何认为对字体重要的附加信息表的一个可扩展的系列。

苹果和Adobe/微软用不同的方式扩展这些表来包含非拉丁手写体（或复杂的拉丁手写体）所需要的先进的排字特性。参见Apple Advanced Typography和OpenType。

TrueType Tables

每个TrueType字体包含表的一个集合，每个表包含一些特定的信息。

Type 1

PostScript字体的一个类型。

Type 2

PostScript字体的一个类型，在OpenType字体包装中使用。

Type 3

PostScript字体的一个非常普通的类型。

Type 0

PostScript字体的一个类型。

Type High

在金属字体的年代，这是金属块的高度 — 印刷面到它所放置的平台的距离。

Typewriter

参见Monospace。

U

Unicode

一个字符集/编码，尝试包含当前世界上使用的所有字符，也包含许多历史上使用的。更多信息参见Unicode consortium。

- BMP: Basic Multilingual Plane (0x00000-0x0FFFF)
- SMP: Supplementary Multilingual Plane (0x10000-0x1FFFF)
- SIP: Supplementary Ideographic Plane (0x20000-0x2FFFF)
- SSP: Supplementary Special-purpose Plane (0xE0000-0xEFFFF)

Undershoot

参见Overshoot。

UniqueID

这是PostScript字体使用的一个字段，以前作为唯一确认字体的机制，随后Adobe做出决定认为它不够好并创建了XUID（扩展的Unique ID）字段。Adobe现在决定不需要两者。

TrueType的“name”表中有一个非常类似的字段。

UseMyMetrics

这是一个TrueType的概念，强制一个复合字形（例如一个带重音的字母）的宽度与其组件相同（例如被添加重音的基本字母）。

V

Vertical Advance Width

CJK文本经常会按竖直方向书写（有时按水平方向），因此和水平步进一样，每个CJK字形有一个竖直步进。

W

Weight

一个字体的粗细是字形中字干的粗细程度。传统上粗细是被命名的，但是目前数字被应用在粗细上。

细体（Thin）
100

超轻体（Extra-Light）
200

轻体（Light）
300

常规体（Normal）
400

中等体（Medium）
500

半粗体（Demi-Bold）
600

粗体（Bold）
700

重体（Heavy）
800

黑体（Black）
900

Nord

Ultra

White space

字体设计的空白间距包括文本行的距离，字母间的距离，字距和字母内的距离。这是一个广泛的包罗万象的术语。

Width

这是一个轻微模糊的术语，又是用来表示步进宽度（从字形开始到下一个字形开始的距离），又是用来表示从左跨距到右跨距的距离。

X

X-height

基线上小写字母（顶部平坦的像“x”或“z”或“v”，相反的顶部弯曲的像“o”或者有顶部的像“l”）的高度。

参见大写高度，顶部，底部，上凸，基线。

XUID

PostScript字体的Extended Unique ID。现在有些过时。参见Unique ID。