

# 字体渲染编年史



# 为什么要研究字体渲染

- 页面上最多的元素是文字。
- 当你要自己实现渲染引擎的时候，字体渲染可能是引擎最复杂的部分。
- 细节非常多，世界上没有完美的字体渲染和排版系统。

# 字体渲染流程



字体

Typeface

# 字体

## 类型

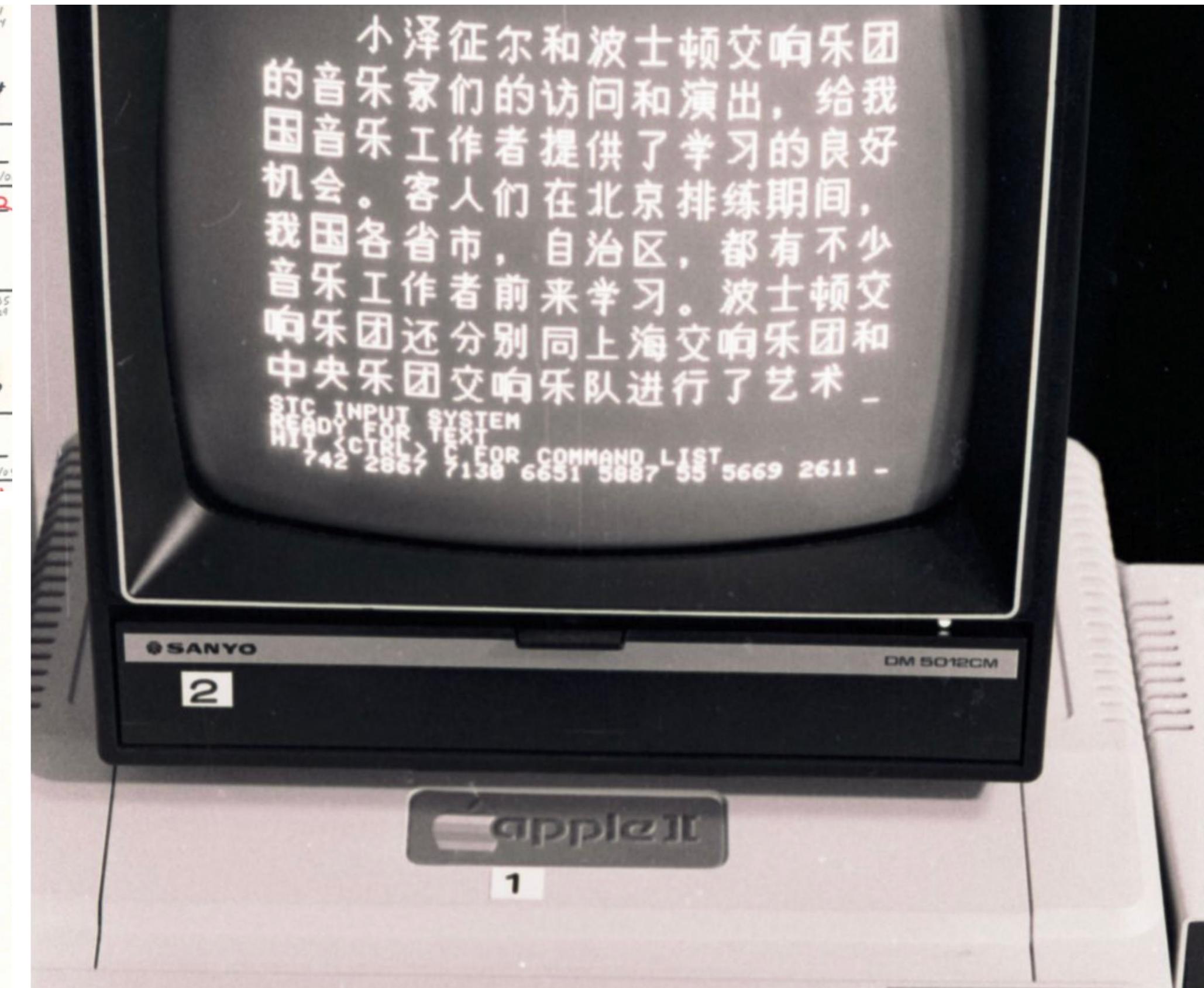
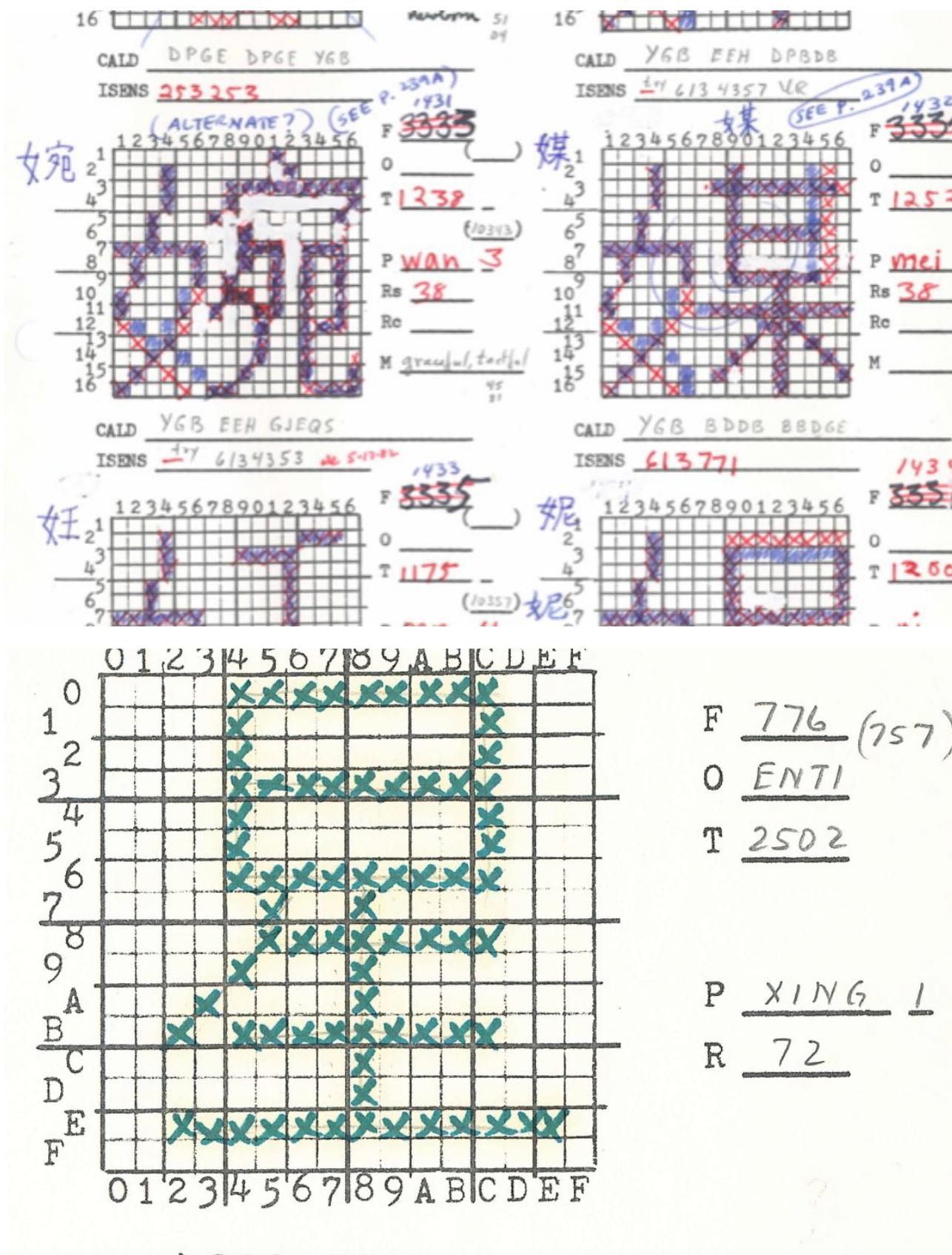
- 点阵字体
- 矢量字体

## 格式

- ttf
- otf
- woff

点阵字体

# 世界上第一款中文字体

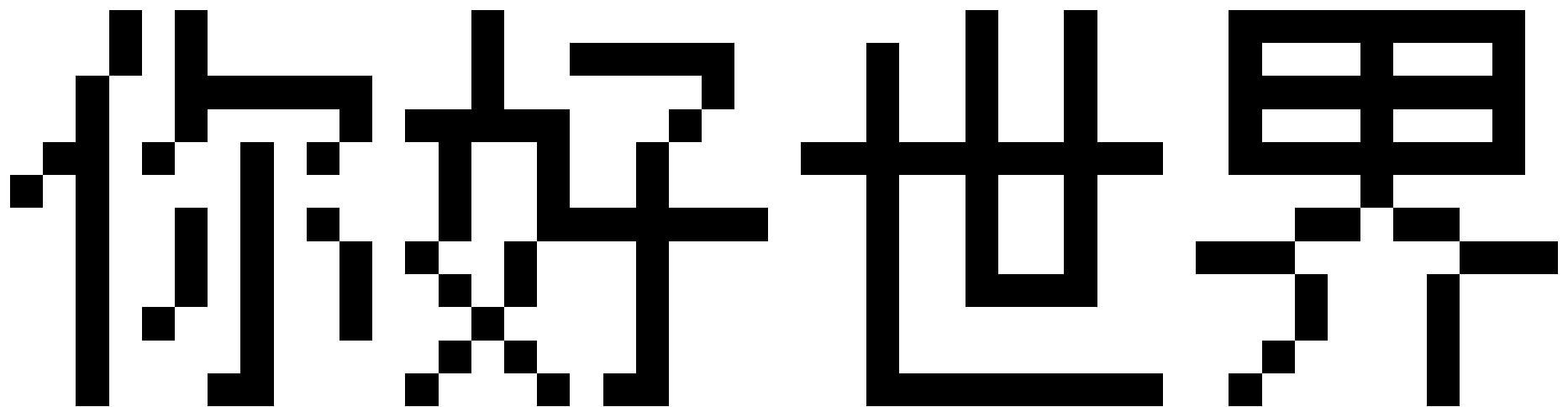


# 点阵字体的优势

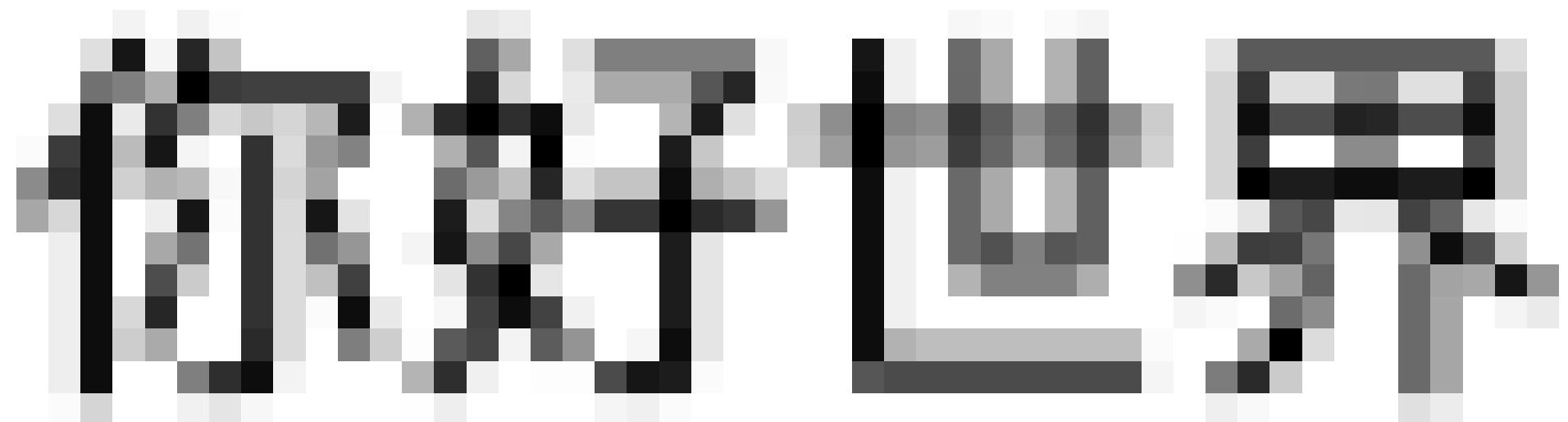
- 渲染速度极快且简单
- 更容易制作。
- 始终提供完全相同的显示效果。
- 经过优化可以极小尺寸的显示器上清晰的显示字形

分别在 `windows` 和 `macos` 下使用 `simsun` 字体绘制 `12px` 的文字

点阵字体



矢量字体



# 点阵字体至今依然在 windows 上使用

```
let ctx = document.getElementById('canvas').getContext('2d');
let y = 10;
for(let i = 11; i < 20; i++) {
  ctx.font = `${i}px simsun`;
  ctx.fillText(`羸膏蠹 ${i}px`, 0, y += i);
}
```

羸膏蠹	11px
羸膏蠹	12px
羸膏蠹	13px
羸膏蠹	14px
羸膏蠹	15px
羸膏蠹	16px
羸膏蠹	17px
羸膏蠹	18px
羸膏蠹	19px

# 极限小的点阵字体

丁卯点阵体 7X7

TPX  
我说道：「爸爸，你走吧。」他望车外看了看，说：「我买几个橘子去。你就在此地，不要走动。」我看那边月台的栅栏外有几个卖东西的等着顾客。走到那边月台，须穿过铁道，须跳下去又爬上去。父亲是一个胖子，走过去自然要费事些。我本来要去的，他不肯，只好让他去。  
9PX  
我说道：「爸爸，你走吧。」他望车外看了看，说：「我买几个橘子去。你就在此地，不要走动。」我看那边月台的栅栏外有几个卖东西的等着顾客。走到那边月台，须穿过铁道，须跳下去又爬上去。父亲是一个胖子，走过去自然要费事些。我本来要去的，他不肯，只好让他去。

CG PIXEL 3X5

CG PIXEL 3X5: A 100% HANDMADE TINY PIXEL FONT.  
A QUICK BROWN FOX JUMPS OVER THE LAZY DOG.  
(OH, REALLY?) 1+1=2. CALL 867-5309/JERRY!  
0123456789 !"#\$%&/()=?+^{}[],.:;-~`^~`>`<`

美咲フォント 8X8

間わずがたりの洋酒外史

佐藤 幸

洋酒といえば、誰でも最初に思い浮かべるのがウイスキー。いわば洋酒のシンボル的な存在なのだが、英語表記が「一般に（米）では Whiskey, （英）では Whisky.」であることはあまり知られていない。米英両国では、このスペルの差で自国产と輸入品を区別しているという。わが和製ウイスキーの“Whisky”という英國式表示は、手本にしたスコッチのフォルムに倣ったものであり、それ以上の意味はないようだ。カナ表記にしても、ごくまれにくくウヰスキー」という書き方を見かけるが、これとて差別化を意図したものではなく単にカナづかいの時代性にすぎない。

現在、カナ表記はくウヰスキーに一本化しており、そこに国产・舶来の区別はない。最近は価格面での差もちぢまり、20年前に ¥10,000 だった本場のスコッチが半値近くに

# 今天的点阵字体

点阵屏



像素风游戏



矢量字体

# 矢量字体

矢量字体是与点阵字体相对应的一种字体。矢量字体的每个字形都是通过数学方程来描述的，一个字形上分割出若干个关键点，相邻关键点之间由一条光滑曲线连接。

矢量字的好处是字体可以无级缩放而不会产生变形。

在 FIGMA 中可以看到矢量字体的关键点



今天常见的矢量字体格式有 `OpenType` 和 `TrueType`。

# PostScript

PostScript 是一种页面描述语言，能够描述页面上的文本和图形。由 Adobe 于 1984 年开发，最初是为了打印机而开发的。

PostScript 最初由 Apple 打印机支持，一路到 1990 年代几乎成为了打印机的标配，后面被 Adobe 自己的 PDF 所取代。（但 PDF 其实只是 PostScript 的阉割版）

PostScript 的概念与 SVG 很相似：

## POSTSCRIPT

```
newpath  
15 25 moveto  
70 90 lineto  
stroke
```

## SVG

```
<path d="M 40 20 L 80 20"  
      stroke="black" fill="none"/>
```

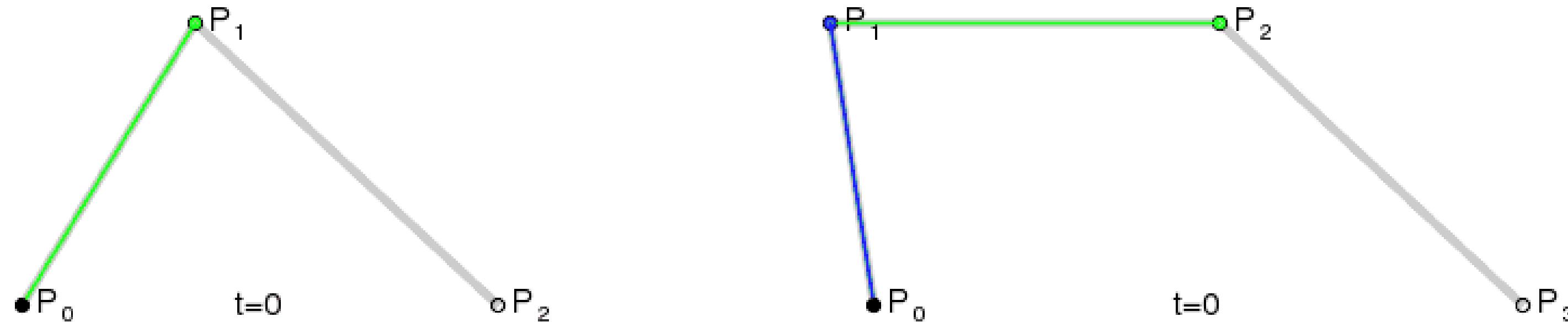
# TrueType 字体

为了让 MacOS 用上矢量字体，Apple 在 1991 年开发的字体格式 `TrueType`。

`TrueType` 的文件后缀是 ` `.ttf`，Windows 上的字体基本上都是 `TrueType` 格式。

一个 `TrueType` 文件可以同时包含点阵字体和矢量字体，并且允许开发者根据不同的字体大小调用不同的字形。

`TrueType` 中的矢量字体由一系列直线或二次贝塞尔曲线构成，而 `PostScript` 使用的是三次贝塞尔曲线，因此理论上 `TrueType` 不如 `PostScript` 精确。



# OpenType 字体

在 1996 年，Adobe 发现 Apple 的 `TrueType` 很成功，于是与微软联合开发了一种新的字体格式 `OpenType`。

`OpenType` 也可以叫做 `TrueType2.0`，和 `TrueType` 大部分定义都是一样的，`OpenType` 能够使用 `PostScript` 字体格式，功能比 `TrueType` 更强一些。

`OpenType` 的文件后缀是 `otf`，目前的 `OpenType` 已经成为 `ISO` 的公开标准。是目前最受欢迎的字体格式。

# WOFF 字体

WOFF 是一种专为网页设计的字体格式标准，由 W3C 标准化，现在已经是推荐标准。

WOFF 与其说是一种字体格式，实际上只是一种容器格式，WOFF 只是将 `TrueType` 或 `OpenType` 字体（这 2 种字体数据格式几乎一样）重新打包并压缩而已。

WOFF 体积比不压缩的字体格式要小得多，便于在网页中使用。WOFF 1.0 采用 `gzip` 压缩一般可以压缩 40%，WOFF 2.0 使用 `Brotli` 压缩体积还要小 30%。

# 字体功能 · 小型大写字母

升级到新款 iPhone, 轻轻松松。

加入 iPhone 年年焕新计划, 每年获取新 iPhone, 享受 AppleCare+ 服务计划, 还可选择分期付款†。

[进一步了解 >](#)



**iPhone XR**

[比较各款 iPhone 机型 >](#)

用智能手机来换购,  
享受折抵优惠。

通过 Apple Trade In 换购计划, 你可以用符合条件的智能手机来换购新 iPhone, 享受折抵优惠\*\*。这样一来, 你受益, 地球也受益。

[购买 iPhone XR >](#)

iPhone XR 的 R 明显比较小, 但他依然是大写的。这不是简单的缩放, Apple 使用了 css 设置了 `font-variant-caps: all-small-caps;` , 调用了字体的小型大写字母功能, 在支持的字体中会有单独设计的字形。

# 字体功能 · 繁体

台

臺

```
font-feature-settings: 'trad';
```

# 字体功能 · 半角标点

你好、『世界』。

你好、『世界』。

```
font-feature-settings: 'halt';
```

# 字体功能 · 连字

ffi

ffl

```
font-feature-settings: 'liga' 0; /* 关闭连字 */
```

# 字体功能 · 本地化

俄语

гпклдвзитжц

保加利亚语

гпклдвзитжц

```
<p lang="ru">гпклдвзитжц</p>
<p lang="bg">гпклдвзитжц</p>
```

# 字体管理器

字体管理器通常由操作系统提供，用于管理系统字体。

- Windows: DirectWrite
- MacOS / iOS: CoreText
- Android: minikin
- linux: fontconfig

字体管理器会给应用程序提供查找字体的接口，接口通常如下

```
Typeface MatchCharacter(string familyName, FontStyle style, string[] locale, char character)
```

现实世界的文本通常混合了不同字体，也可以在一段文字中包含不同的样式。应用程序通常使用一个叫`TextRuns`的迭代器来处理文本流。

## TEXTRUNS

```
while (ch = nextChar()) {
    if (typeface?.has(ch)) {
        yield typeface;
    } else {
        yield typeface = fontMgr.MatchCharacter(familyName, style, locale, ch);
    }
}
```

# 文字塑性 (text shaping)

从字体文件中找到文本中每个字所对应的字形，这个过程就叫文字塑性。

OpenType 有一份文字塑性规范文档，非常复杂，里面针对一些小语种有额外的处理规则。

# HarfBuzz

---

HarfBuzz 是一个文字塑性库，只需要输入一个字体和一串 Unicode 字符串就可以得到对应的正确排列的字形序列和对应字形应该放置的位置。期间会根据规范处理复杂的布局规则，重新排序和定位等操作。

HarfBuzz 被用于 Firefox, GNOME, ChromeOS, Chrome, LibreOffice, XeTeX, Android 和 KDE 等。



# 布局

在实际使用中的文本并不是一行从左向右排到底的，有一系列布局规则。

通常需要根据容器的大小控制文字断行。不同的语言不同的国家还有不同的排版规范。

# 自动断行算法

Unicode 定义了一堆换行规则 ([UAX #14](#))，定义了换行机会和禁止换行规则，排版引擎只能在换行机会的时候执行换行。以下是一些例子

## 换行机会

- 空格、` 或连字符`-`之后能够换行
- 数字加减乘除后面能够换行
- 泰语等不使用空格的语言，需要分析音节边界进行换行

## 禁止换行

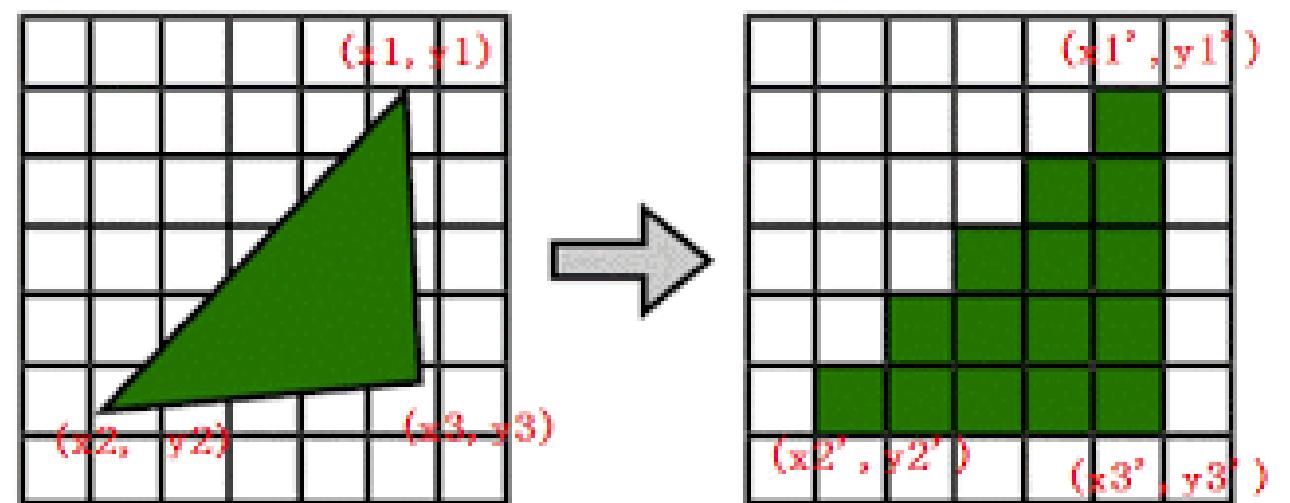
- 不换行特殊字符的前后 `&NoBreak;`、`&nbsp;`、` 等等
- 数字的小数点`.`、和负号`-`，百分号`%`，或表示单位的`\$`、`¥`等。
- ...

这里推荐使用开源库 [ICU](#) 处理。

渲染

# 栅格化

矢量字体每个字形都是由若干条线条组成，但我们的显示屏是由像素点组成的。将矢量字形转换为屏幕上显示图像的过程叫做栅格化。



# FreeType

---

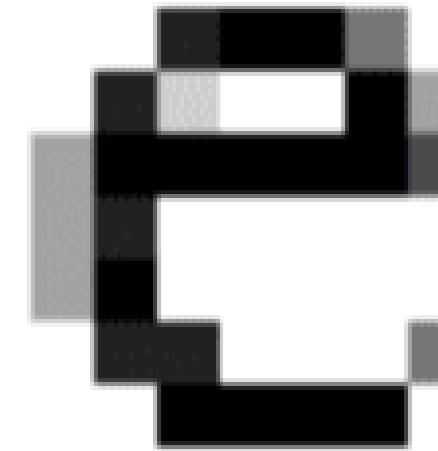
FreeType 是一个字体栅格化库，能够处理 TrueType、OpenType 的矢量字体。FreeType 被用于 Android, ChromeOS 和各类 Linux 发行版中。

# 抗锯齿

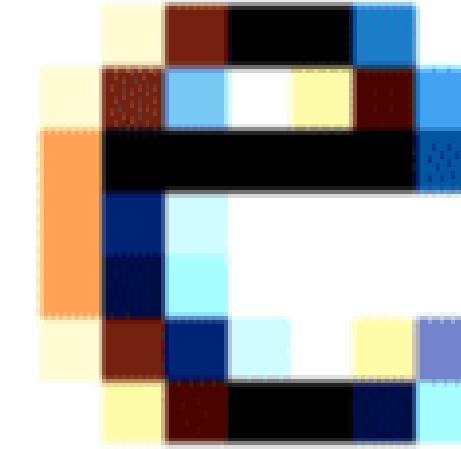
在低分辨率下，文字很小，栅格化通过抗锯齿显示更多细节。

灰度抗锯齿在字体边缘让像素透明，看起来就像覆盖了一半的感觉。

次像素抗锯齿更加极端，利用显示器每个像素有红绿蓝三个子像素的原理，单独控制子像素，获得三倍的分辨率提升。代价是字体边缘看起来会有彩边。



灰度抗锯齿

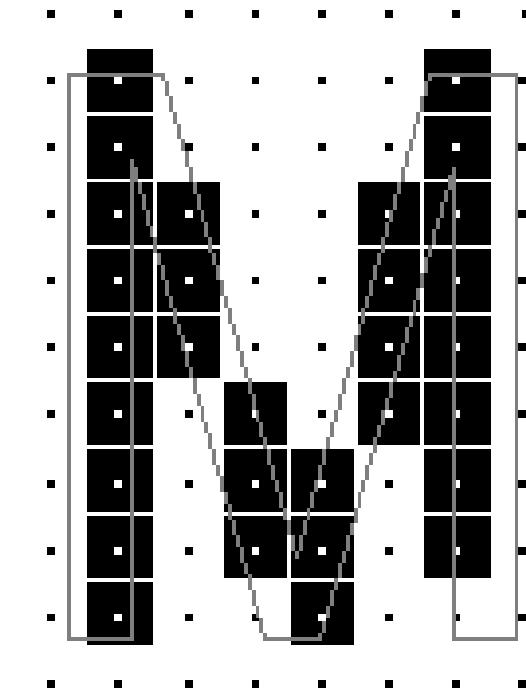


次像素抗锯齿

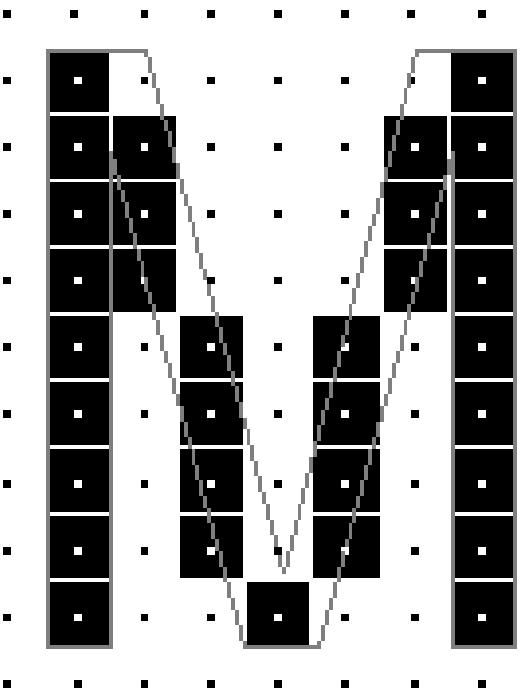
# hinting 渲染提示

在低分辨率下，hinting 功能可以使用一些指令对字体轮廓进行一些调整，使其与网格对其。能大大提高文字在低分辨率下的可读性。

hinting 实际上是内置于字体中的一段程序，栅格化的时候会被执行。优秀的字体会精心设计 hinting 程序。



没有 hinting 的字体



有 hinting 的字体

# 字体缓存

栅格化是相对比较慢的过程，渲染引擎应该尽可能缓存栅格化的结果。

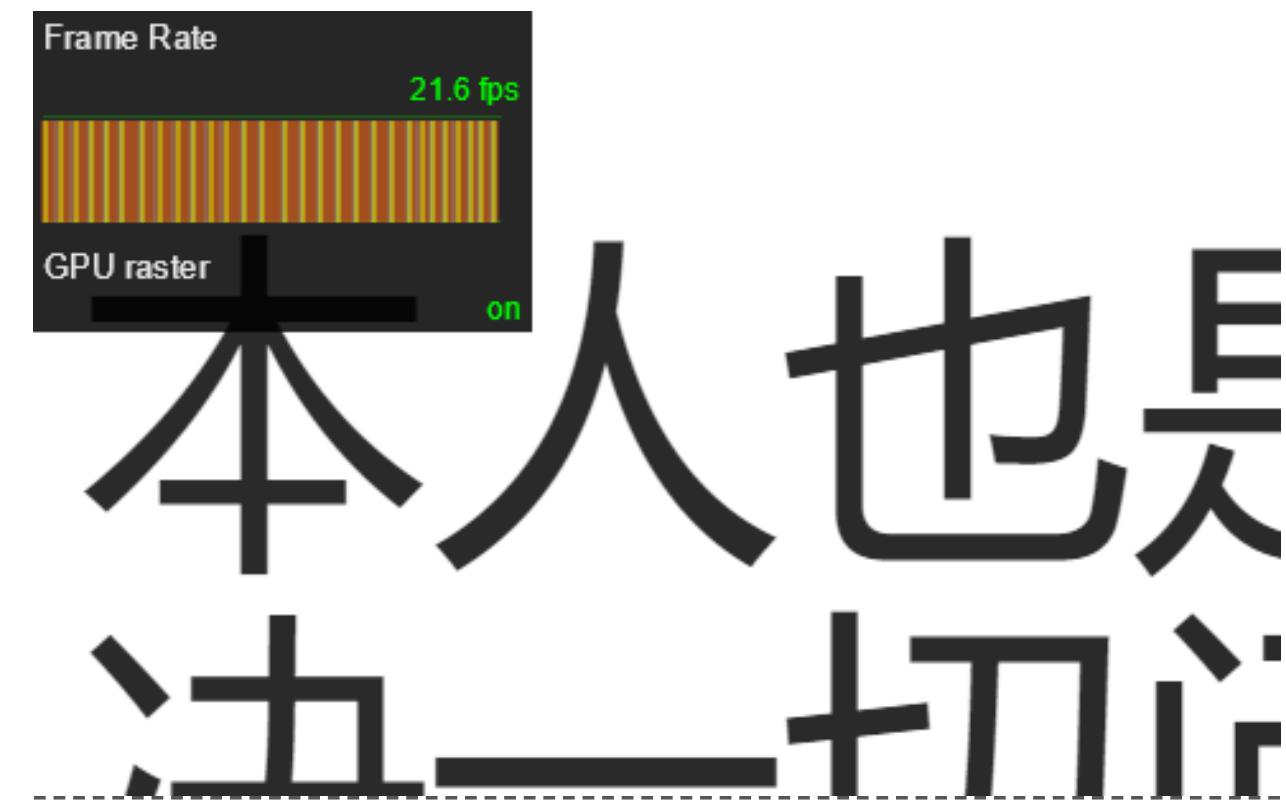
这里选取比较典型的 `skia` (`Chrome` 和 `Android` 所用的底层绘图库) 来做讲解。

在 `skia` 中字体缓存被实现为一个 `LRU` 缓存，`skia` 会给每个字形计算一个唯一的 `ID`，然后使用 (`ID` + 字体大小 + 字体样式) 做 `key` 将图像储存在缓存中。下次渲染到同样的字就可以直接从缓存中提取，不需要重新栅格化。

`skia` 默认的缓存大小是 `2MB`，考虑到文字是单色的，一个像素只需要一个字节，`2MB` 就足够缓存 `1920x1080` 的字形图像。

# 字形缓存缺陷

`skia` 的字形缓存非常有效，但是仔细思考会发现一个小问题，对于同一个字形，每个字体大小都得重新渲染缓存。假如我们做一个文字放大的动画，字体大小每一帧都在变化，字体缓存就会失效。



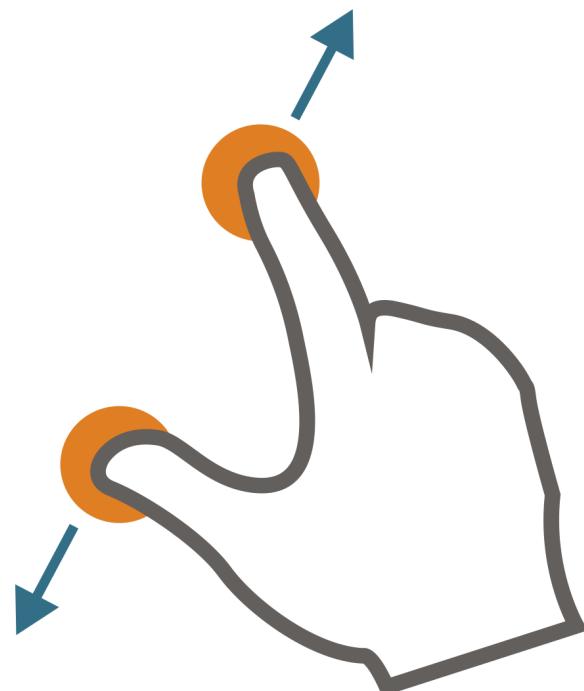
在 Chrome(canvas) 中的字体放大动画只有 20 fps

想象字体放大动画的应用场景：

移动端浏览器上两指缩放，可以放大网页。所有文字也随之放大。

经过观察所有 Android 浏览器在快速放大时，文字会出现模糊现象。怀疑是在放大过程，只是通过插值将图像放大，并没有重新渲染。

但 iOS 没有观察到模糊现象。



# 论外：Apple CoreText

`MacOS` 和 `iOS` 应用程序使用 `Apple CoreText API` 来绘制字形，底层是一个叫作 `Quartz 2D` 的图像引擎，`Quartz 2D` 将整个屏幕视为一个 `PostScript` 画布，所有的文字和线条都会转化为 `PostScript`。

应用程序不需要自己做栅格化，`Quartz 2D` 会统一栅格化，`Quartz 2D` 栅格化速度非常快，动画也能非常流畅。

在 `Mac` 和 `iPhone` 上 `Safari` 浏览器中运行上面的文字放大测试，均能流畅 60 帧。

在 `MacOS` 上的 `Chrome` 和 `Firefox` 中运行还是很卡，是因为他们的绘图还是为传统渲染引擎设计的，没有利用好 `Quartz 2D` 的能力。

# 合成

经过一顿操作，我们终于得到了文字渲染的结果，其他元素合成到一起输出最终画面。

# 总结：如何实现一个自渲染文本库

1. 解码要渲染的文本
2. 遍历文本，调用操作系统的字体管理器，获得每段对应的字体。
3. 调用 HarfBuzz 获得每个字对应的字形和每个字形的正确位置。
4. 调用 ICU 搭配自己写的算法，对文字进行排版。
5. 调用 FreeType 把字形渲染成图像，放到排版对应位置。
6. 合成最终画面