طراحی خودکار قلم‌های فارسی

محمّد قدسی* کیارش بازرگان**

دانشکده‌ی مهندسی کامپیوتر
دانشگاه صنعتی شریف

چکیده

در این مقاله یک روش سریع و خودکار برای طراحیِ «علمیِ» حروف مختلف خط فارسی پیشهاد می‌شود. منظور از طراحیِ علمیِ حروف، استفاده از منحنی‌های ریاضی و پارامترهای مختلف در طراحی است به‌طوری که بتوان اندازه‌های مختلف یک حرف و سبُک‌های متفاوت نگارش آن را با کیفیت عالی و تنها با تغییر مقادیر چند پارامتر به سرعت تولید کرد. مِتافونت، زبانی برای طراحی علمی فونت‌هاست که در این مقاله از آن استفاده می‌شود. بااستفاده از امکانِ «قلم» در این زبان، می‌توان حرکت قلمْ‌درشتِ یک استاد خوش‌نویس را شبیه‌سازی کرد. استفاده مستقیم از متافونت برای طراحی کیفی فونت‌ها کاری طولانی و همراه با مشکلات متعددی است. در این مقاله الگوریتم‌هایی پیشنهاد می‌شود تا بتوان تصویر پویش‌شده‌ی یک حرف را به‌صورت خودکار به برنامه‌ی متافونتی که آن حرف را تولید می‌کند تبدیل کرد. نرم‌افزاری بر اساس این الگوریتم طراحی شده است که به‌صورت گرافیکی و با محاوره با کاربر این کار را انجام می‌دهد.

*دانشیار
**در حال حاضر عضو هیأت علمی دانشگاه مینسوتا، آمریکا

Automatic Design of Persian Typefaces

Mohammad Ghodsi Kiarash Bazargan

Computer Engineering Department,
Sharif University of Technology

Abstract

In this paper, a fast method for automatic generation and scientific design of Persian fonts is proposed. Scientific typeface design is an approach in which fonts are described by mathematical curves with well-defined parameters. By tuning these parameters different fonts (or shapes) of a family of typefaces is generated automatically. METAFONT is a language suitable for such design and we use in this article. This language is particularly useful in designing Persian fonts, because it can be used to simulate the pen movements of a calligrapher who designs the font. In this paper, several algorithms are proposed for semi-automatic conversion of the scanned bitmap image of a font into a METAFONT program, that in turn can produce the font at high quality. A complete software has been implemented based on these algorithms that works interactively with the user and facilitates the font design.

۱ معرفی

یکی از مراحل مهم تولید هر سیستم حروف‌چینی فارسی، طراحی مجموعه‌ای کامل از خانواده‌های مختلف قلم یا فونت[۱] با کیفیت عالی است. منظور از یک خانواده از فونت، تمام حروف الفبا می‌باشد که با سیاق[۲] یکسان نوشته شده‌اند. مثال‌هایی از خانواده‌های قلم، «نازک»، «سیاه»، «ایتالیک» و «ایرانیک» می‌باشند. یکی از مهم‌ترین ویژگی‌هایی که در طراحی خانواده‌های قلم باید در نظر گرفت، همگن بودن حروف است. قسمت‌های مشترک حروف شبیه به هم در یک خانواده، باید دقیقاً مثل هم تعریف شده باشند تا متنی که با آن قلم نوشته می‌شود، همگن باشد. همچنین تغییراتی که یک حرف از یک خانواده را به همان حرف در خانواده‌ی دیگر تبدیل می‌کنند، باید برای تمام حروف، مشابه باشند.

روش‌های مختلفی برای طراحی فونت وجود دارد که از آن میان می‌توان به روش‌های زیر اشاره نمود:

۱) طراحی تک تک حروف توسط خوشنویس و خواندن آن‌ها توسط پویشگر[۳] و ذخیره‌سازی آن‌ها به‌صورت بیت‌نگاشت[۴]. در این روش نمی‌توان تضمین کرد که حروف به‌صورت همگن تعریف شوند. هم‌چنین حروف تولید شده توسط این روش را نمی‌توان به‌راحتی بزرگ و کوچک نمود.

۲) تعریف هر حرف به‌طور جداگانه و به‌صورت مجموعه‌ای از منحنی‌های درجه ۲ یا ۳ و نگه‌داری آن‌ها به‌صورت TrueType و یا PostScript. در این روش امکان استفاده از پارامترهایی که ویژگی‌های حروف را به‌صورت کلی توصیف کنند، وجود ندارد.

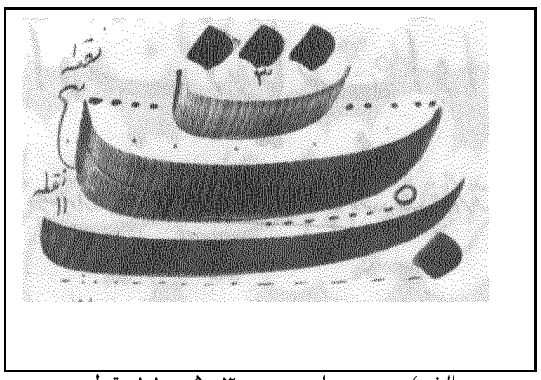
۳) استفاده از روش‌های طراحی هندسی و تعریف سطح بالای حروف با پارامترهای پایدار. پارامتری پایدار است که با تغییر اندک آن، منحنی‌های تشکیل دهنده‌ی حروف، تغییر ناگهانی نکنند. تعداد این پارامترها و وابستگی آن‌ها به‌هم نباید خیلی زیاد باشد.

با توجه به ویژگی‌های روش آخر، در این مقاله از آن استفاده می‌کنیم. یکی از بهترین ابزار طراحی فونت‌ها با چنین روشی زبان متافونت[۵] است [۱]. متافونت یک زبان برنامه‌نویسی توصیفی[۶] و سطح بالاست که اجازه می‌دهد حروف را با استفاده از پارامترهای مختلف تعریف و با تغییر یک یا چند پارامتر می‌توان خانواده‌های مختلف حروف را تولید کرد. در متافونت می‌توان قطعه‌های مشترک، و یا خیلی شبیه، حروف متفاوت را به‌صورت جدا تعریف نمود و سپس با ترکیب این قطعه‌ها با قطعه‌های غیر مشترک، حروف همگنی را تولید نمود.

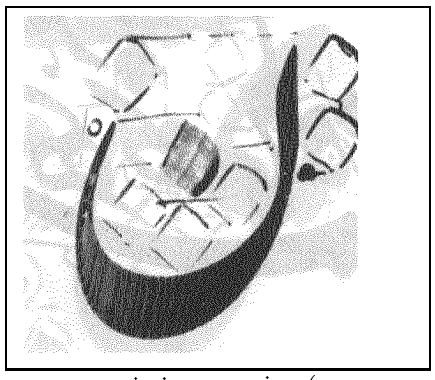
متافونت زبانی توصیفی است که در سال ۱۹۸۲ توسط دانلد کنوث[۷] برای طراحی فونت‌های سیستم حروف‌چینی TEX ابداع شد. مهم‌ترین قابلیت متافونت این است که به برنامه‌نویس اجازه می‌دهد معادلات توصیف کننده‌ی مکان و زاویه‌ها را به‌صورت «ضمنی[۸]» بنویسد. وقتی تعداد معادلات به‌اندازه‌ی کافی رسید، متافونت مقدار متغیرهای مجهول را محاسبه می‌کند. متافونت به‌جای صدا کردن زیربرنامه، از بسط ماکروها استفاده می‌کند. این مسئله، خطایابی برنامه را مشکل می‌سازد.

---

[۱] font
[۲] font style
[۳] scanner
[۴] bitmap
[۵] METAFONT
[۶] Descriptive
[۷] Donald Knuth
[۸] implicit



الف) منحنی‌های «ب» ۳، ۵ و ۱۱ نقطه



ب) حرف «ن» پویش‌شده

شکل ۱: تصویرهای پویش‌شده‌ی دو حرف «ن» و «ب».

با وجود این که زبان متافونت امکانات بسیار مفیدی در اختیار قرار می‌دهد، استفاده از آن بسیار مشکل و نیازمند دقت و صرف وقت فراوان است. در این مقاله مباحث نظری و الگوریتم‌هایی را پیشنهاد می‌کنیم تا بتوان تصویر پویش‌شده‌ی یک حرف را به‌صورت خودکار به برنامه‌ی متافونتی که آن حرف را تولید می‌کند تبدیل کرد. بر این اساس محیط نرم‌افزاریِ محاوره‌ای[۹] را طراحی کرده‌ایم که در این مقاله گزارش می‌شود.

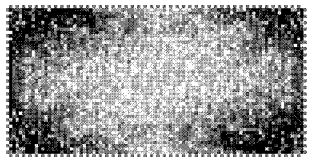
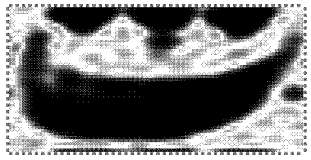
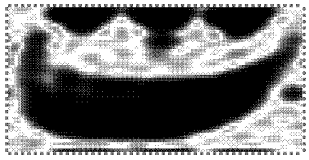
۲ پایه‌های نظری و الگوریتم‌ها

نرم‌افزاری که طراحی‌شده، مجموعه‌ای از ماکروها به زبان متافونت و یک محیط گرافیکی تحت «ویندوز» است که امکانات مناسب برای تولید نیمه‌خودکار برنامه‌های متافونت را در اختیار می‌گذارد. ابتدا، کاربر تصویر پویش‌شده‌ی حرف مورد نظر خود را تهیه می‌کند. این تصویر می‌تواند خاکستری یا سیاه و سفید باشد. سیستم به دقت تصویر خاصی وابسته نیست ولی معمولا بهتر است ابعاد مستطیل محیطی حرف، حدود ۱۰۰۰ نقطه در ۱۰۰۰ نقطه باشد. این امر باعث می‌شود ویژگی‌هایی از منحنی مرزی مثل زاویه، انحنا و موقعیت، با دقت مناسبی تشخیص داده‌شود. شکل ۱ تصویرهای پویش شده‌ی حرف «ب» و «ن» را نشان می‌دهد.

سپس نویز تصویر برطرف می‌شود و منحنی مرزی آن به‌دست می‌آید. کاربر می‌تواند منحنی مرزی را اصلاح کند: نقاطی را حذف یا اضافه کند و یا آستانه‌ی حذف نویز راتغییر دهد. همچنین، کاربر می‌تواند مشخصات منحنی، از جمله زاویه‌ی مماس برآن، را در هر نقطه مشاهده کند. تمام اعمال خودکار برنامه، بر روی منحنی مرزی بسته‌ی حرف، و یا قطعه منحنی‌های وابسته به‌هم انجام می‌شود. منحنی مرزی بسته، توسط روش‌های خودکار و نیز، درصورت لزوم با دخالت کاربر، به قطعه‌هایی تقسیم می‌شود و هر قطعه، با یک منحنی بزیه تقریب زده می‌شود. سپس مکان‌های عبور قلم‌درشت، تعیین می‌گردند و باتوجه به مشخصات منحنی در آن نقاط، برنامه‌ی متافونت تولید می‌شود. در انتها، خروجی متافونت پس از اجرای برنامه‌ی تولید شد، با تصویر اولیه مقایسه می‌شود تا کاربر، اشکالات برنامه‌ی متافونت را برطرف کند.

جزییات مراحل بالا در ادامه می‌آید.

[۹] Interactive

مراحل روش اول	فیلتر با پهنای گذر ۱۰	فیلتر با پهنای گذر ۱۰	فیلتر با پهنای گذر ۱۰
تبدیل فوریه‌ی فیلترشده			
تصویر خاکستری حاصل از تبدیل فوریه‌ی معکوس			
تصویر سیاه‌وسفید حاصل از قطع تصویر فیلترشده			

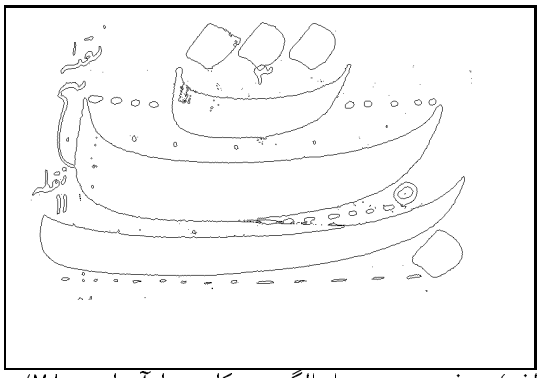
جدول ۱: مراحل مختلف و مقایسه‌ی نحوه‌ی فیلترگیری در روش اول.

۱.۲ حذف نویز

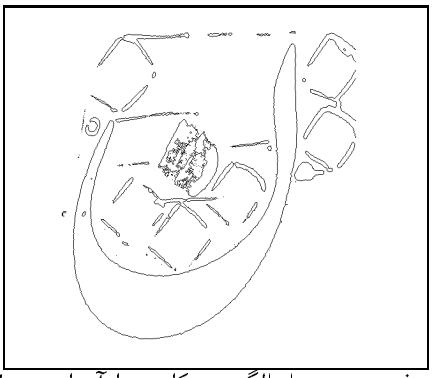
برای حذف نویز تصویر پویش‌شده، روش‌های زیر مورد بررسی قرار گرفتند.

۱) تصویر خاکستری حرف را به‌صورت آرایه‌ای دوبعدی درنظر می‌گیریم که نمایان‌گر تابعی دوبعدی است. تبدیل فوریه‌ی دوبعدی [۵] (2Dim Dft) این تصویر را محاسبه می‌کنیم و آن را از یک فیلتر پایین گذر ایده‌آل [۴] می‌گذرانیم و تبدیل فوریه‌ی معکوس نتیجه را محاسبه می‌کنیم. حاصل، تصویری خاکستری خواهد بود که «تار» شده است. تصویر حاصل را با درنظر گرفتن یک آستانه‌ی روشنایی، به تصویری سیاه و سفید تبدیل می‌کنیم. این روش روی تمام تصویر به طور یکسان عمل می‌کند. برای حذف نویز در قسمتی از تصویر، باید پهنای گذر فیلتر را برای تمام تصویر، کم کنیم. این کار اغلب باعث می‌شود دقت تصویر را در گوشه‌های حرف یا بقیه‌ی قسمت‌های ظریف آن، از دست بدهیم. از آن‌جا که این نوع نویز (یعنی نویز زیاد در یک قسمت از تصویر) در حروف پویش شده معمول است، و نیز به علت کندی ذاتی محاسبات تبدیل فوریه، این روش، روش مناسبی برای فیلتر کردن تصاویر ما نیست. مثال‌هایی از فیلتر کردن تصاویر با این روش را می‌توانید در جدول ۱ مشاهده کنید. همان‌طور که می‌بینید، با کم شدن پهنای فیلتر، نویز بدنه در قسمت چپ حرف از بین می‌رود ولی تیزی گوشه‌ی بالا—راست نیز تحلیل می‌رود.

در دو روش بعد، ابتدا توسط الگوریتم کانتور چهارنقطه‌ای با آستانه‌ی تغییر روشنایی بین ۵۰ تا ۸۰ درصد، مرز حروف را به‌دست می‌آوریم. نمونه‌های منحنی مرزی به‌دست آمده از روش کانتور را در شکل‌های ۲ ملاحظه می‌کنید. از آن‌جا که الگوریتم کانتور تقریباً همیشه، یک مسیر بسته با انشعاباتی مانند آن‌چه در شکل ۳ آمده است، می‌سازد و این انشعابات مشکلاتی را ایجاد می‌کند، خروجی الگوریتم کانتور را توسط یک الگوریتم جداکننده‌ی مسیرهای انشعابی، ساده می‌کنیم و سپس عملیات فیلترکردن نویز را روی منحنی مرزی حاصل انجام



الف) حرف «ب» پس از الگوریتم کانتور با آستانه‌ی ۰/۷۸



ب) حرف «ن» پس از الگوریتم کانتور با آستانه‌ی ۰/۷۰

شکل ۲: تشخیص مرز با الگوریتم کانتور چهار نقطه‌ای.

شکل ۳: انشعاب الگوریتم کانتور

می‌دهیم. الگوریتم جداکننده‌ی مسیرهای انشعابی را در انتهای این بخش آورده‌ایم.

۲) منحنی مرزی را به توابع $Y = f_1(X)$، $X = g_1(Y)$، $Y = f_2(X)$، $X = g_2(Y)$ تقسیم می‌کنیم به‌طوری‌که این توابع، تمام مرز منحنی را بپوشانند و به‌صورت یکی در میان توابعی از X و Y باشند. سپس تبدیل فوریه‌ی هر قسمت را محاسبه می‌کنیم و نویز هر تابع را به‌کمک فیلتر ایده‌آل پایین‌گذر، برطرف می‌کنیم. به این ترتیب می‌توان هر قسمت از منحنی را که تابعی فقط از X یا فقط از Y است، باتوجه به مقدار نویز، از فیلتری با پهنای باند مناسب عبور داد و نویز آن را حذف نمود. اِشکال این روش این است که، به علت «بسته» نبودن منحنی f ها و g ها و در نتیجه تناوبی نبودن آن‌ها، لبه‌های انتهایی توابع به‌صورت نوسانی در می‌آیند و عملاً کارآیی این روش از بین می‌رود و منحنی را در نقاط اتصال این توابع، ناپیوسته می‌کند. نمونه‌هایی از منحنی پایین حرف «ب» پنج نقطه در شکل ۴ آمده است. در شکل ۵ تبدیل فوریه‌ی همین منحنی، قبل از فیلترشدن آمده است.

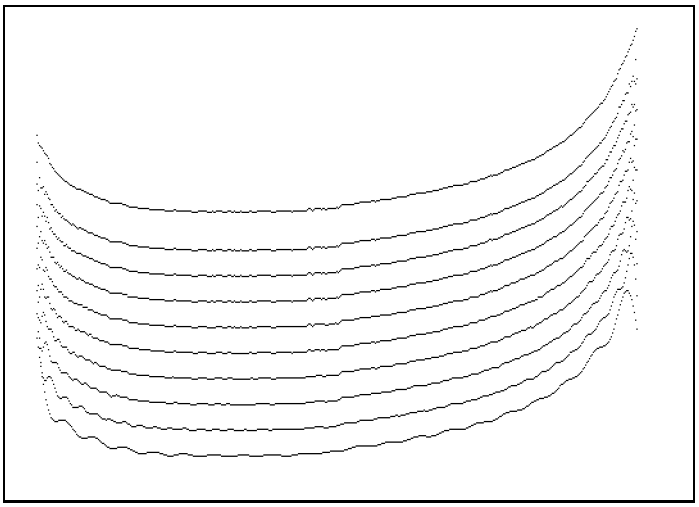
۳) منحنی را به‌صورت مجموعه‌ای از نقطه‌ها، با مختصات صحیح، در می‌آوریم: (N تعداد نقاط منحنی مرزی است)

$$C = \{x(t), y(t)\}, \qquad t = 0, 1, \ldots, N$$

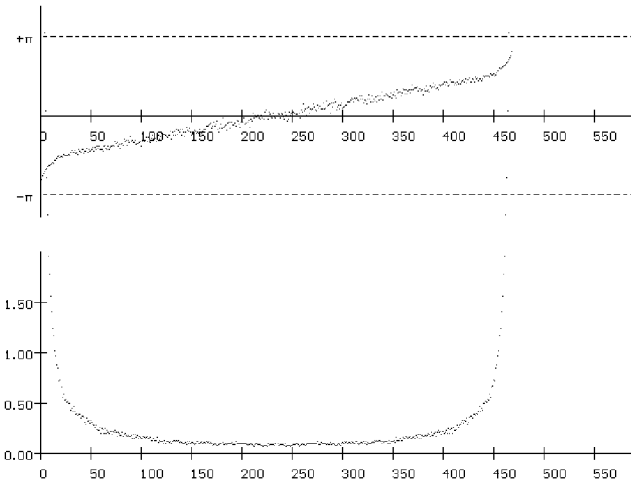
باتوجه به این‌که منحنی مرزی، یک منحنی بسته است، می‌توان آن را به‌صورت توابع متناوب x و y تعریف نمود.

$$\tilde{x} = \begin{cases} x(t) & 0 \le t \le N \\ x(t \bmod N) & \text{otherwise} \end{cases}$$

حال تابع گاوسی g را به‌صورت زیر تعریف می‌کنیم [۶]: (σ پهنای منحنی را تغییر می‌دهد.)



شکل ۴: منحنی پایینی «ب» پنج نقطه‌ای پس از عبور از فیلترهای ایده‌آل پایین‌گذر با پهنای گذر بین ۲۰ تا ۱۸۰



شکل ۵: تبدیل فوریه‌ی منحنی پایینی «ب» پنج نقطه‌ای

$$g_\sigma = \frac{۱}{\sigma\sqrt{۲\pi}} e^{\frac{t^۲}{۲\sigma^۲}}$$

تابع گاوسی را با توابع متناوب $\tilde{x}$ و $\tilde{y}$ Convolve می‌کنیم [۴]، تا توابع متناوب نرم و بدون نویزی به‌دست آیند. این توابع، دارای مختصات حقیقی هستند و نویز آن‌ها به‌اندازه‌ی مطلوبی حذف شده است:

$$\begin{aligned} X_\sigma &= \tilde{x}(t)g_\sigma(t) = \sum_{u=\infty}^{\infty} \tilde{x}(t-u)g_\sigma(u) \\ Y_\sigma &= \tilde{y}(t)g_\sigma(t) = \sum_{u=\infty}^{\infty} \tilde{y}(t-u)g_\sigma(u) \end{aligned}$$

از آن‌جایی که مقادیر g_σ برای t های بزرگ‌تر از ۵ خیلی کم است، می‌توانیم محدوده‌ی محاسبه‌ی convolution را از ۵− تا ۵+ بگیریم و در نتیجه، سرعت محاسبات را بسیاربالا ببریم.

روش فیلترگیری گاوسی، به‌دلایل زیر برای کاربرد مورد نظر بسیار مناسب است:

۱) به‌صورت حلقه‌ای عمل می‌کند. یعنی اگر قسمتی از تصویر دارای نویز زیاد باشد، این نویز روی چگونگی فیلتر شدن منحنی مرزی تاثیری ندارد.

۲) محاسبات آن بسیار سریع است.

۳) حاصل این روش، یک منحنی با مختصات حقیقی است.

۴) نقاط به‌صورت پیوسته تغییر می‌کنند و خطای کوانتیزه بودن تصویر پویش‌شده‌ی اولیه را ندارند. این خاصیت باعث می‌شود محاسبات مراحل بعدی، مثل محاسبه‌ی زاویه در نقاط مختلف، بدون نیاز به استفاده از روش‌های تقریب زننده و روش‌های محاسبات عددی انجام گیرد.

نمونه‌ی فیلتر شده‌ی شکل ٦—الف حرف «ن» با پهنای $\sigma = ۵$ نشان داده شده است. تابعی که عمل فیلترگیری روی آن انجام شده‌است، در شکل ؟؟ آمده است. به حذف نوسانات ریز روی لبه‌های منحنی توجه نمایید. مقایسه‌ی بهتری را شکل ٦—ب نشان می‌دهد که منحنی سمت راست، منحنی فیلتر شده است. مقدار اندک تغییر منحنی در گوشه‌های تیز قابل توجه است.

۳ الگوریتم جداکننده‌ی مسیر انشعابی

در این بخش، الگوریتم جداکننده‌ی مسیرهای انشعابی حاصل از الگوریتم کانتور را توضیح می‌دهیم. چنان‌چه اشاره شد، تصویر ابتدا توسط الگوریتم کانتور مرزیابی می‌شود و اغلب شامل مسیرهای انشعابی می‌باشد، مانند آنچه در شکل ۳ آمده است. بنابراین، پس از اجرای الگوریتم کانتور، الگوریتم زیر را اجرا می‌کنیم:

الف) حرف «ن» که از فیلتر گاوسی گذشته است. ب) مقایسه‌ی منحنیِ فیلترشده با منحنی اصلی

شکل ٦:

۱– مستطیل حاوی تصویر را به‌صورت آرایه‌ای دوبعدی از نقاط به‌نام Pic در نظر بگیر. مجموعه‌ی C که در ابتدا تهی است منحنی‌های جدا از هم را ذخیره می‌کند. این مجموعه، در پایان، خروجی الگوریتم خواهد بود.

۲– برای تمام نقاط $\mathrm{Pic}(X_\circ, Y_\circ)$ از تصویر، اگر سیاه است، آن‌را سفید کن و الگوریتم «دنبال کردن مسیر ساده[10]» (که در زیر می‌آید) را با نقطه‌ی شروع $(X_\circ, Y_\circ)$ اجرا کن. این فراخوانی باعث می‌شود که یک یا چند مسیر بدون انشعاب، به C اضافه شود. این مسیرها یک مسیر کانتوری در Pic را افراز می‌کنند. فرض کنید تعداد کل مسیرهای ساده که از تمام مسیرهای کانتوری به وجود آمده‌اند، N باشد.

۳– برای مسیرهای P_i، برای $i = ۱ \ldots, N - ۱$،

۳–۱ برای مسیرهای P_j، برای $j = i + \ldots, N$،

۳–۱–۱ نقاط ابتدا و انتهای P_i و P_j را دوبه‌دو از نظر مجاورت با همسایگی $\sqrt{۲}$ بررسی کن. (این‌که آیا این دو مسیر از یک نقطه‌ی مسیر کانتوری منشعب شده‌اند.)

۳–۱–۲ اگر مجاور باشند، حتماً یک مسیر سومی هم در همین نقطه، با این دو مسیر، مجاور است. حلقه‌ی j را ادامه بده تا هر دوی این مسیرهای منشعب پیدا شوند. این مسیرها را First و Second می‌نامیم. البته مقدار Second را فقط در صورتی قرار بده که First و Second در همسایگی باشند. یعنی حداکثر یک نقطه بین آن‌ها فاصله باشد. (همان نقطه‌ای که انشعاب از آن به وجود آمده[11])

۳–۲ اگر دو مسیر پیدا شده است، (یعنی First و Second هر دو مقادیر معنی‌دار دارند،) First را برابر مسیربزرگ‌تر قرار بده.

۳–۳ نقاط First را به مسیر P_i اضافه کن و First را از C حذف کن. در نتیجه P_i مسیر بزرگ‌تری شده است در حالی که هنوز یک مسیر ساده و بدون انشعاب است.

۴– نقاط تمام مسیرهای C را سیاه کن که تصویر اولیه‌ی Pic تغییر نکند.

[10] مسیری ساده است که هر نقطه‌ی آن حداکثر دو همسایه در ۸ نقطه‌ی مجاور خود داشته باشد. مسیری که انشعاب ندارد ساده است.

[11] در الگوریتم دنبال کردن مسیر ساده، فقط وقتی از اضافه کردن یک نقطه به مسیر ساده امتناع می‌کنیم که غیر از نقطه‌ی قبلی خود در مسیر، از یک همسایه‌ی دقیقاً مجاور داشته باشد. هریک از این دو همسایه‌ی دقیقاً مجاور، یک مسیر ساده به C اضافه کرده است.

الف) فواصل هندسی متناظر با فواصل مساوی t ب) اثر پارامتر tension بر روی نقاط کنترلی

شکل ۷: منحنی‌های بزیه

این الگوریتم، لزوماً بزرگترین مسیرهای ساده را به‌دست نمی‌آورد، چون به‌صورت کاملاً موضعی سعی می‌کند مسیر را گسترش دهد. ولی تقریباً همیشه، بزرگترین مسیرها را به‌دست می‌آورد. علت این است که در حروف پویش شده انشعابات ناخواسته، معمولاً کوتاه هستند. همچنین تعداد انشعابات مسیر کانتوری در تصویرهای حروف، اغلب کمتر از ۱۰ است.

الگوریتم دنبال کردن مسیر ساده، به‌صورت زیر است:

۱– مسیر جدیدی به C اضافه کن که فعلاً فقط شامل نقطه‌ی CurPt (پارامتر ورودی) است. first-choice و second-choice برابر اولین و دومین نقاطی خواهند بود که در هر لحظه می‌توان از CurPt به آن‌ها رفت. فعلاً آن‌ها را برابر مقادیر بی‌معنی قرار بده.

۲– برای هر هشت نقطه‌ی مجاور CurPt، اگر نقطه‌ی $\mathrm{Pic}(x, y)$ سیاه است، آن‌را در first-choice قرار بده و اگر first-choice قبلا مقدار گرفته است، آن‌را در Second-choice قرار بده.

۳–

الف) اگر هیچ کدام از first-choice و Second-choice مقدار ندارند، به پایان این مسیر رسیده‌ایم. از الگوریتم خارج شو.

ب) اگر فقط first-choice مقدار دارد، آن‌را به مسیر اضافه کن و نقطه را در Pic سفید کن. CurPt را برابر first-choice قرار بده و از مرحله‌ی ۲ کار را ادامه بده.

ج) چون هر دوی first-choice و second-choice مقدار دارند، مسیر دارای انشعاب شده است. هر دو نقطه را در Pic سفید کن و الگوریتم دنبال کردن مسیر ساده را یک بار با نقطه‌ی first-point به‌عنوان نقطه‌ی شروع، و بار دیگر با نقطه‌ی Second-point اجراکن. این عمل باعث می‌شود که از این نقطه‌ی انشعاب، سه مسیر ساده تولید شود. از الگوریتم خارج شو.

۴ تقریب قطعه مسیر ساده با منحنی بزیه

در بخش قبل چگونگی تبدیل منحنی مرزی حرف به یک مسیر ساده‌ی صاف و بدون نویز را توضیح دادیم. این مسیر ساده باید با قطعه‌منحنی‌های بزیه‌ای تقریب زده شود که هر زوج، نشان‌گر مسیرهای چپ و راست حرکت قلم درشت

باشد. علت این امر این است که در نهایت، حروف قرار است با متافونت تولید شوند. بنابراین اگر سیستم برای تقریب قسمتی از مرز حرف، حرکت قلمی را تعیین کند که متافونت نتواند آن‌را رسم کند، عملاً روش‌های ارائه شده بدون استفاده خواهند بود. پس روش‌های تقسیم منحنی و تقریب و نمایش آن به‌وسیله‌ی قلم‌درشت، باید مبتنی بر ویژگی‌ها و قابلیت‌های متافونت باشد.

تعیین نقاط تقسیم مسیر ساده به زیر منحنی‌های بزیه، بسیار مهم است. اگر این نقاط را به‌طور مناسبی انتخاب کنیم، تقریب زیرمنحنی‌ها با سرعت بیشتری انجام می‌شود و تعداد زیرمنحنی‌ها حداقل می‌شود. درباره‌ی چگونگی انتخاب این نقاط، در بخش بعد توضیح خواهیم داد. در این بخش، فرض می‌کنیم این نقاط را انتخاب کرده‌ایم و می‌خواهیم یکی از این قطعه‌های مسیر ساده را تقریب بزنیم. همچنین فرض می‌کنیم این قطعه‌مسیر، قابل تقریب زدن با یک منحنی بزیه باشد.

منحنی بزیه، مجموعه نقاط منحنی پارامتریک زیر است:

$$Z(t) = (۱-t)^۳ Z_{m۱} + ۳(۱-t)^۲ t Z_{c۱} + ۳(۱-t)t^۲ Z_{c۲} + t^۳ Z_{m۲}$$

برای تقریب قطعه‌مسیری ساده با منحنی بزیه، و با توجه به داشتن نقاط ابتدا و انتهای منحنی بزیه ($Z_{m۱}$ و $Z_{m۲}$)، کافی است $Z_{c۱}$ و $Z_{c۲}$ را پیدا کنیم. اگر مشتق منحنی بزیه را در نقاط ابتدا و انتهای آن محاسبه کنیم، فرمول‌های زیر بدست می‌آید:

$$\left.\frac{dZ}{dt}\right|_{t=۰} = ۳(Z_{c۱} - Z_{m۲})$$
$$\left.\frac{dZ}{dt}\right|_{t=۱} = ۳(Z_{m۲} - Z_{c۲})$$

یعنی اگر بتوانیم مشتق $Z'(t)$ را در نقاط ابتدا و انتهای قطعه منحنی بزیه تعیین کنیم [۲]، می‌توانیم نقاط کنترلی را به‌دست آوریم و در نتیجه قطعه‌مسیر را تقریب بزنیم. اما این کار، ساده نیست. قطعه‌منحنی بزیه، از مجموعه نقاط x و y تشکیل شده که فاصله‌ی هندسی آن‌ها، ارتباطی با فاصله‌ی پارامتر t آن‌ها ندارد. به‌عبارت دیگر، سرعت پیش روی و حرکت منحنی را بر حسب t، در نقاط مختلف آن، نمی‌دانیم. (به عنوان مثالی از ارتباط فاصله‌ی فیزیکی با فاصله‌ی پارامتریک نقاط، شکل ۷—الف را ببینید). بنابر این نمی‌توانیم با محاسبه‌ی $y'(x)$ (مشتق y بر حسب x)، $y'(t)$ را به‌دست آوریم[۱۲].

در حقیقت، بردارهای $Z_{c۱} - Z_{m۱}$ و $Z_{m۲} - Z_{c۲}$ سرعت حرکت منحنی نسبت به t را در نقاط ابتدا و انتها نشان می‌دهند. پارامتر `tension` در متافونت، اندازه‌ی این بردارها را کنترل می‌کند: با کم‌شدن این پارامتر، فاصله‌ی نقاط کنترلی، در همان راستا که بودند، بیشتر می‌شود. در شکل ۷—ب مثالی از اثر این پارامتر بر روی نقاط کنترل آمده است. در این شکل، منحنی ضخیم‌تر، با عبارت

```
draw Zm1 dir 5 ..  dir-70 Zm2;
```

[۱۲] البته یک روش تقریب این است که در ابتدا، تغییرات x و t را یکسان بگیریم و در نتیجه $x'(t)$ را برابر ۱ فرض کنیم. سپس تقریبی از منحنی به‌دست آوریم و سعی کنیم $x'(t)$ را از میزان اختلاف منحنی تقریب زده شده و منحنی اصلی، در نقاط مختلف، پیدا کنیم و تقریب خود را در هر مرحله بهتر نماییم.

رسم شده است و منحنی نازک‌تر با عبارت

```
draw Zm1 dir 5 ..  tension 1 and 2 ..  dir-70 Zm2;
```

رسم شده است. تاثیر tension بر روی نقطه‌ی کنترلی v، نقطه‌ی w را تولید کرده است که به نقطه‌ی Zm2 با نسبت ۱ به ۲، نزدیکتر است.

توصیف Z0 W0 .. tension α and β .. W1 Z1 به زبان متافونت را در نظر بگیرید و فرض کنید مقادیر تمام پارامترها معلوم باشد. (W0 و W1 بردارهای جهت منحنی بزیه در دو سر آن هستند.) در ابتدا نشان می‌دهیم متافونت چگونه این پارامترها را به نقاط کنترلی u و v تبدیل می‌کند به‌طوری که عبارت فوق با عبارت Z0 .. controls u and v .. Z1 معادل باشد. بعداز آن، الگوریتمی برای تقریب قطعه‌مسیر ساده با یک منحنی بزیه، ارائه خواهیم داد.

اگر زوایای ϕ و θ را به‌صورت زیر تعریف کنیم (صص ۱۲۸–۱۳۲ از [۲]):

$$\phi = \arg((Z_1 - Z_0)/w_1) \quad (1)$$

$$\theta = \arg(w_0/(Z_1 - Z_0)) \quad (2)$$

مختصات نقاط کنترلی u و v با فرمول‌های زیر محاسبه می‌شوند:

$$v = Z_1 - \frac{e^{-i\phi}(Z_1 - Z_0)f(\phi,\theta}{\beta} \quad (3)$$

$$u = Z_0 - \frac{e^{-i\theta}(Z_1 - Z_0)f(\phi,\theta}{\alpha} \quad (4)$$

که در آن تابع $f(\phi,\theta)$ به صورت زیر تعریف می‌شود:

$$f(\phi,theta) = \frac{2 + \sqrt{2}(\sin\theta - \frac{1}{16}\sin\phi)(\sin\phi - \frac{1}{16}\sin\theta)(\cos\theta - \cos\phi)}{3(1 + \frac{1}{2}(\sqrt{5} - 1)\cos\theta + \frac{1}{2}(3 - \sqrt{5})\cos\phi}$$

برای تکمیل کردن مبانی پایه‌ای الگوریتم تقریب زدن منحنی، به روشی برای مقایسه‌ی دو منحنی بزیه نیاز است. فرض کنید $Z(t)$ مجموعه‌ی نقاط منحنی ایده‌آلی باشند که (درصورت وجود) قطعه مسیر C را تقریب می‌زند، و فرض کنید منحنی بزیه‌ی $B(t)$، یک تقریب اولیه از C است. می‌خواهیم B و Z را باهم مقایسه کنیم. نقاط شروع و پایان C، Z و B، و همچنین زاویه‌ی مماس بر هر سه در این نقاط، یکسان هستند. این شرط باعث می‌شود که نقاط کنترلی B و Z در یک راستا باشند و تفاوت دو منحنی، فقط در اندازه‌ی بردارهایی باشد که نقاط کنترلی را به نقاط اصلی وصل می‌کنند نه در زاویه‌ی آن‌ها. اگر بردارهای U و V را به‌صورت $U = Z_{co} - Z_0$ و $V = Z_1 - Z_{c1}$ تعریف کنیم، منحنی‌های Z و B به شکل زیر تعریف می‌شوند:

$$Z_t = (1-t)^3 Z_0 + 3(1-t)^2(Z_0 + U) + 3(1-t)t^2(Z_1 - V) + t^3 Z_1$$

$$B_t = (1-t)^3 Z_0 + 3(1-t)^2(Z_0 + aU) + 3(1-t)t^2(Z_1 - bV) + t^3 Z_1$$

که a و b نسبت اندازه‌ی بردارهای وصل‌کننده‌ی نقاط کنترلی در دو منحنی هستند. اگر این مقادیر را به‌دست آوریم، می‌توانیم از روی نقاط کنترلی B، نقاط کنترلی Z را پیدا کنیم و در نتیجه منحنی C را تقریب بزنیم.

اگر معادلات بالا را از هم کم کنیم (برای t ثابت و در نتیجه نقاط متناظر از دو منحنی)، معادله‌ی برداری زیر حاصل می‌شود:

$$B_t - Z_t = ۳(۱-t)t^۲(a-۱)U + ۳(۱-t)t^۲(۱-b)V$$

که از حل آن، مقادیر a و b به‌صورت زیر نتیجه می‌شوند:

$$a - ۱ = \frac{-B_{yt}V_x + B_{xt}V_y - V_yZ_{xt} + V_xZ_{yt}}{۳(۱-t)t^۲(U_xV_y - U_yV_x)} \quad (۵)$$

$$۱ - b = \frac{-B_{yt}U_x + B_{xt}U_y - U_yZ_{xt} + U_xZ_{yt}}{۳(۱-t)t^۲(U_yV_x - U_xV_y)} \quad (۶)$$

بنابراین، اگر یک تقریب اولیه از منحنی بزیه، مثل B، داشته باشیم، می‌توانیم تقریب ایده‌آل را به‌دست آوریم. اما از آن‌جایی‌که نمی‌توان نقطه‌های کاملاً متناظر دو منحنی، که t برابر دارند، را پیدا کرد[۱۳]، باید با تکرار روش‌های تقریبی، به جواب نزدیک شد. براساس فرمول‌های (۱) تا (۶)، الگوریتمی مکاشفه‌ای[۱۴] برای پیدا کردن مقادیر α و β با فرض داشتن زاویه‌های ϕ و θ ارائه می‌کنیم:

۱– منحنی ورودی را که می‌خواهیم آنرا با منحنی بزیه تقریب بزنیم C می‌نامیم. نقاط Z0 و Z1 را مساوی نقاط ابتدا و انتهای C قرار بده. زاویه‌ی خط مماس بر C را در نقاط ابتدا و انتها محاسبه کن و با محاسبه‌ی اختلاف این زوایا و زاویه‌ی خطی که Z0 و Z1 را وصل می‌کند، مقادیر زاویه‌های ϕ و θ را، طبق رابطه‌های (۱) و (۲) به‌دست آور. از آن‌جا که منحنی C، یک منحنی صاف‌شده با فیلتر گاوسی است و خطای نویز و کوانتیزه بودن را ندارد، انتظار داریم مقادیر این زاویه‌ها، تا حد مطلوبی دقیق باشند. هر چه این زوایا با دقت بیشتری تعیین شوند، الگوریتم سریع‌تر به جواب خواهد رسید.

۲– مقدار α و β را برابر ۱ قرار بده.

۳– طبق روابط (۳) و (۴)، مقادیر u و v را به‌دست آور.

۴– با روش مقایسه‌ای برداری دو منحنی، اختلاف α و β با مقادیر ایده‌آل را پیدا کن. این روش مقایسه‌ی برداری، به صورت زیر انجام می‌شود:

۴–۱ منحنی بزیه‌ی Z0..controls u and v..Z1 را B بنام.

۴–۲ L_C و L_B را برابر با طول هندسی B و C (تعداد نقاط کوانتیزه شده‌ی آن‌ها) قرار بده. مقادیر $a_a ve$ و $b_a ve$ را برابر صفر قرار بده.

۴–۳ مقدار t را برای B از صفر تا ۱ با قدم‌های $۱/N$ افزایش بده. (N یک عدد ثابت است که هر چه بزرگتر باشد، در این مرحله از الگوریتم جواب را کندتر ولی دقیق‌تر به‌دست می‌آورد.)

[۱۳] چون ما Z را نداریم. برای پیدا کردن نقطه‌ی Z_t، با توجه به فاصله‌ی هندسی B_t (نسبت فاصله‌ی هندسی آن از ابتدای منحنی بزیه)، سعی می‌کنیم C_t را به‌دست آوریم (با همان نسبت فاصله‌ی هندسی از ابتدا مسیر) و Z_t را برابر C_t بگیریم.

[۱۴] Heuristic

۴–۳–۱ برای هر نقطه‌ی B_t، مقدار g_t را برابر فاصله‌ی هندسی از نقطه‌ی B_0 تا B_t قرار بده.

۴–۳–۲ نقطه‌ی متناظر B_t از منحنی C را (که فاصله‌ی هندسی آن از ابتدای C، برابر است C_t) بنام.

۴–۳–۳ a_ave و b_ave را به‌اندازه‌ی $a(1/N)$ و $b(1/N)$ افزایش بده. a و b از فرمول‌های (۵) و (٦) به‌دست می‌آیند.

۴–۴ با استفاده از مقادیر a_ave و b_ave، مقادیر α و β را به مقدار ایده‌آل آن‌ها نزدیکتر کن.

۵– مراحل ۳ و ۴ را آن‌قدر تکرار کن که یا به تقریب خوبی از منحنی بررسی و یا تعداد دفعات تکرار مراحل، از حدی بیشتر شود. تقریب خوب منحنی وقتی به‌دست می‌آید که مجموع مربع فاصله‌ی نقاط متناظر هندسی C و B، از حدی کمتر شود. یعنی:

$$\frac{1}{\max\{L_B, L_C\}} \sum_{j=1}^{\max\{L_B, L_C\}} (C_t - B_t)^2 \leq \epsilon$$

٦ – اگر تقریب خوبی به‌دست آمد، متوقف شو. در غیر این صورت، زاویه‌های ϕ و θ را اندکی تغییر بده (در چهار نوبت، ترکیبات کم و زیاد کردن ϕ و θ را امتحان می‌کنیم) و الگوریتم را دوباره تکرار کن. اگر تقریب خوبی بدست نیامد، شکست الگوریتم را اعلام کن تا در انتخاب نقاط شروع و پایان، تجدید نظر شود.

۵ محاسبه ویژگی‌های منحنی و تعیین نقاط قلم‌درشت

اگر بتوانیم ویژگی‌هایی از منحنی مرزی حرف، مانند انحنای منحنی، نقاط تیز، قطعه خط‌های مستقیم را پیدا کنیم، می‌توانیم بعضی مکان‌های قلم درشت را حدس بزنیم. روشی که در [٦] پیشنهاد شده است از این نظر بسیار مفید است. اگر $X(t)$ و $Y(t)$ همان توابع فیلترشده با فیلتر گاوسی باشند، تابع انحنای K را به شکل زیر تعریف می‌کنیم:

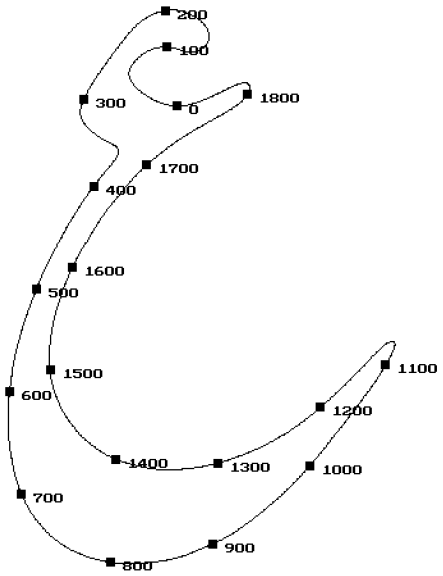
$$K_\sigma(t) = \frac{\dot{X}\ddot{Y} - \dot{Y}\ddot{X}}{(\dot{X}^2 + \dot{Y}^2)^{3/2}}$$

که در آن، مقادیر $\dot{X}$، $\dot{Y}$، $\ddot{X}$ و $\ddot{Y}$ به‌صورت زیر تعریف می‌شوند:

$$\dot{X} = \frac{dX_\sigma(t)}{dt} \qquad \ddot{X} = \frac{d^2X_\sigma(t)}{dt^2}$$
$$\dot{Y} = \frac{dY_\sigma(t)}{dt} \qquad \ddot{Y} = \frac{d^2Y_\sigma(t)}{dt^2}$$

و مقادیر مشتق‌ها را می‌توان به شکل زیر تقریب زد:

$$\dot{X} = X_\sigma(t+1) - X_\sigma(t-1)$$
$$\ddot{X} = X_\sigma(t+1) - X_\sigma(t-1) - 2X_\sigma(t)$$



شکل ۸: نمونه‌ای از منحنی مرزی فیلتر شده

به این ترتیب، میزان انحنای منحنی در نقاط مختلف آن به‌دست می‌آید. در شکل ۸ می‌توانید نمونه‌ای از منحنی مرزی و در شکل ۹ تابع انحنای آنرا ببینید.

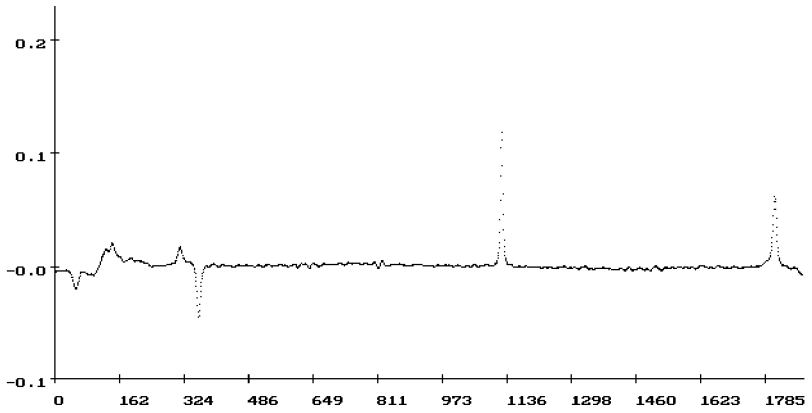
در تابع انحنا، نقاطی که مشتق صفر دارند و نقطه‌ی بیشینه‌ی موضعی هستند، به‌عنوان گوشه‌های تیز منحنی در نظر می‌گیریم. نقاط متناظر با دو سرِ قلم‌درشت، انحناهایی تقریباً مساوی ولی در جهت عکس دارند.

سازگاری یک مکان قلم‌درشت را تعریف می‌کنیم. قلمی سازگار است که :

- انحنای دو سر آن، تقریباً مساوی و در جهت عکس باشند.
- زوایای دو سر آن، تقریباً مساوی باشند.
- مقطع قلم‌درشت، منحنی مرزی حرف را در نقطه‌ای غیر از دو سر قلم، قطع نکند.
- فاصله‌ی هندسی نقاط دو سر، از گوشه‌های تیز، تقریباً یکسان باشد.

باتوجه به این تعریف، الگوریتمی بسیار ساده برای تعیین مکان‌های قلم‌درشت پیشنهاد می‌کنیم:

- اگر گوشه‌ی تیزی وجود دارد، از آن شروع کن. در غیر این صورت، از هر نقطه‌ی دلخواه دیگری شروع کن.
- در دو طرف منحنی، حرکت کن و نقاط متناظر را پیدا کن. نقاط متناظر آن‌هایی هستند که اگر دو سر یک مکان قلم‌درشت در نظر گرفته شوند، آن قلم سازگار باشد. آن‌قدر از دو طرف جلو برو که نقاط متناظر پیدا نشوند.



شکل ۹: تابع انحنای مزبوط به شکل ۸

- سعی کن دو طرف منحنی بررسی‌شده را با دو منحنی بزیه تقریب بزن. اگر الگوریتم تقریب، شکست خورد، منحنی‌های دو طرف را نصف کن (و در نقطه‌ی تقسیم، یک قلم قرار بده) این کار را آن‌قدر ادامه بده تا یا اندازه‌ی منحنی تقسیم‌شده از حدی کوچک‌تر شود و یا تقریب منحنی به‌دست آید.
- عملیات بالا را برای تمام گوشه‌های تیز انجام می‌دهیم. این عمل باعث می‌شود قطعه‌هایی از حرف، تولید شوند. وظیفه‌ی چسباندن این قطعات به‌هم، و اصلاح قلم‌های اشتباه، به عهده‌ی کاربر است. البته ماکروهایی در متافونت برای کمک به کاربر نوشته شده است.

٦ سیستم پیاده سازی شده (محیط گرافیکی)

سیستم پیاده‌سازی شده، برنامه‌ای تحت Windows است که با مترجم Borland C++ Ver 3.1 ترجمه‌شده و از کتابخانه‌ی Borland OWL استفاده می‌کند.

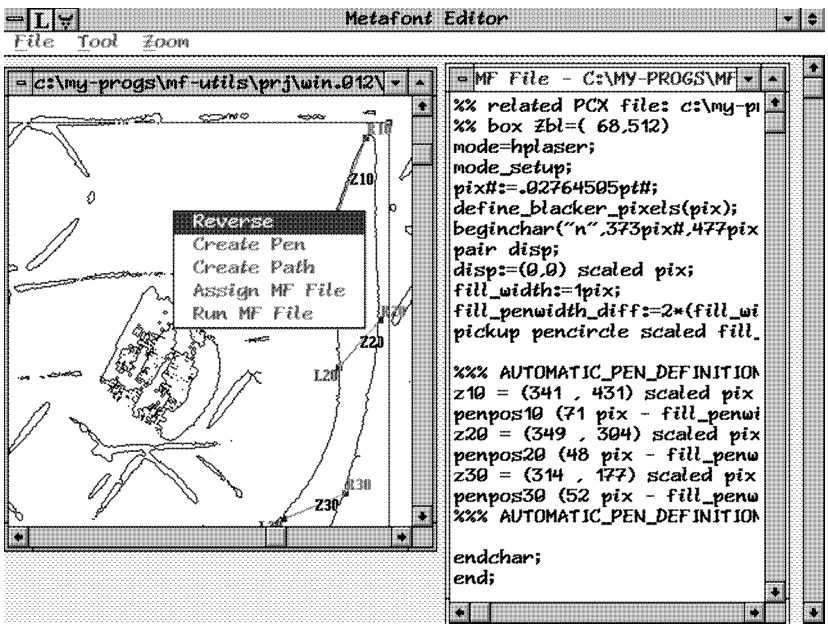
پنجره‌ی اصلی برنامه، یک MDI Frame است و منوهای Tool, File و Zoom را دارد. در این برنامه، می‌توان به‌طور هم‌زمان روی یک یا چند تصویر PCX که تصویرهای پویش‌شده‌ی حروف هستند، کار کرد.

هر پنجره‌ی تصویر می‌تواند حداکثر یک پنجره‌ی برنامه (حاوی برنامه‌ی تولید شده‌ی متافونت) و حداکثر یک پنجره‌ی ویژگی‌های انحنا (حاوی مشخصات انحنای مسیر پردازش شونده) داشته باشد. شکل ۱۰ نمونه‌ای از پنجره‌ی برنامه و تصویر را نشان می‌دهد.

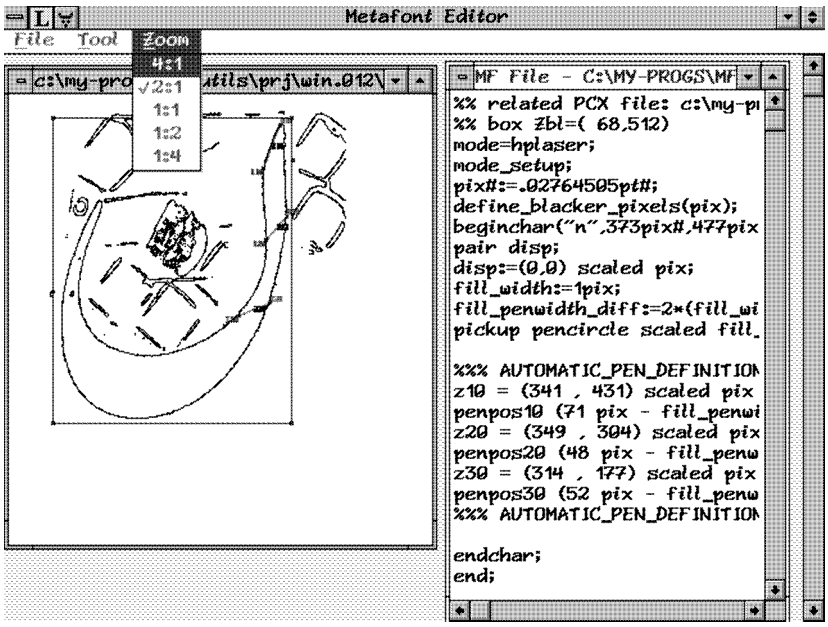
در پنجره‌ی تصویر، تصویر PCX را ویرایش و منحنی‌های مرزی ناحیه‌ای از آن‌را پیدا می‌کنیم. سپس این منحنی‌های مرزی را فیلتر می‌کنیم و منحنی را تقریب می‌زنیم. محل وصل‌کننده‌ی حرف به‌صورت خودکار تولید می‌شود و کاربر نیز می‌تواند آن‌را ویرایش کند. در پنجره‌ی ویژگی‌های انحنا، کاربر می‌تواند ویژگی‌هایی از منحنی پردازش شونده، مانند میزان انحنا و زاویه، و نقاط متناظر با گوشه‌های تیز تشخیص داده شده توسط برنامه را ببیند.

در منوی zoom می‌توان بزرگ‌نمایی تصویر را تعیین نمود. نمونه‌هایی از استفاده‌ی این منو در شکل ۱۱ نشان داده شده است. آن‌ها را با شکل ۱۰ مقایسه کنید.

در پنجره‌ی تصویر، می‌توان اشیائی، مثل مستطیل، قلم‌درشت را به‌وجود آورد. این اشیاء بصورت شفاف روی صفحه

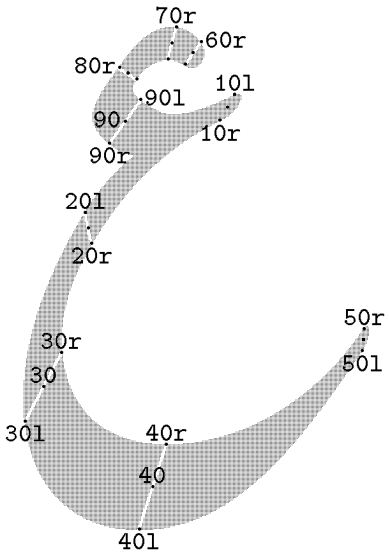


شکل ۱۰: نمونه‌ای از پنجره‌ی تصویر و پنجره‌ی برنامه



شکل ۱۱: نسبت کوچک شدن ۱ به ۲

شکل ۲۱: حرف «ن» تولید شده



شکل ۳۱: حرف «ع» تولیدشده و مکان‌های قلم‌درشت

رسم می‌شوند: یعنی وقتی روی هم، یا نقاط زمینه قرار می‌گیرند، همه‌ی اشیاء روی هم قرارگرفته را می‌توان دید. هر یک از این اشیاء را می‌توان در صفحه حرکت داد، تغییر شکل و اندازه داد و یا اعمال مخصوص به آن اشیاء را روی آن‌ها انجام داد. مثالی از عمل خاص یک شیء، فیلتر کردن مسیر بسته است که فقط برای این نوع شیء قابل انجام است. نمونه‌ای از منو در شکل ۱۰ دیده می‌شود.

شکل‌های ۲۱ و ۳۱ نمونه‌های تولیدشده‌ی این سیستم هستند.

۷ نتیجه‌گیری و گسترش‌های آینده

در این مقاله الگوریتم‌های به‌کار رفته در یک سیستم نرم‌افزاری برای تولید خودکار فونت‌ها به زبان متافونت ارائه گردید. سیستم پیاده‌سازی شده می تواند برای طراحی سریع و با کیفیت عالی حروف استفاده شود. در این سیستم تصویر پویش شده یک یا چند حرف با استفاده از ابزارهای گرافیکی فراهم‌آمده به‌صورت نیمه‌خودکار به برنامه‌ی متافونت تبدیل می شود. این کار، با حذف نویز از تصویر، تقریب منحنی مرزی با قطعات منحنی بزیه و سپس تعیین مشخصات قلم متافونت انجام می‌شود. هر چند که این کار در حال حاضر با کیفیت بسیار خوبی انجام می‌شود و سیستم موجود

عملاً می تواند زمان طراحی یک مجموعه فونت را از هفته‌ها به چند ساعت کاهش دهد، ولی کار برای تکمیل مراحل خودکارسازی ادامه دارد. در این راستا، ایده‌های مطرح شده در [۷] می‌تواند مؤثر باشد. مسئله‌ی تشخیص خودکار محل قرار گرفتن قلم‌درشت‌ها می‌تواند بهبود یابد. برای این منظور، عملیات نازک سازی بسیار مهم است.

ایده‌های مطرح شده در این مقاله در تشخیص نوری کلمات و حروف (OCR) نیز می تواند مورد استفاده قرار گیرد. پروژه‌هائی هم در این زمینه در دست اجرا است.

مراجع

[1] D. E. Knuth, *The* METAFONT*book*, Addison-Wesley, 1985.

[2] D. E. Knuth, METAFONT*: The program*, Addison-Wesley, 1985.

[3] D. E. Knuth, *Computers and Typesetting*, Addison-Wesley, 1986.

[4] A.V. Oppenheim, A.S. Willsky, *Signals and Systems*, Prentice-Hall Innternational, 1983.

[5] R.C. Gonzalez, P. Wintz, *Digital Image Processing*, Addison-Wesley, 1987 2nd ed.

[6] J.D. Foley, A. van Dam, S.K. Feiner, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990, 2nd ed, 478-491.

[7] K.Xin, B.Lim, G.S. Hong "A Scale-Space Filtering Approach For Visual Feature Extraction", Pattern Recognition Journal, Vol 28 #8, Aug 1995, 1145-1158.