

## طراحی خودکار قلمهای فارسی

محمد قدسی\* و کیارش بازرگان\*\*

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

(دریافت مقاله: ۷۸/۱۰/۲۲ - دریافت نسخه نهایی ۸۰/۴/۱۶)

چکیده - در این مقاله یک روش سریع و خودکار برای طراحی (( علمی )) حروف مختلف خط فارسی پیشنهاد می شود. منظور از طراحی علمی حروف، استفاده از منحنیهای ریاضی و پارامترهای مختلف در طراحی است به طوری که بتوان اندازه های مختلف یک حرف و سبکهای متفاوت نگارش آن را با کیفیت عالی و تنها با تغییر مقادیر چند پارامتر به سرعت تولید کرد. متافونت، زبانی برای طراحی علمی فونتهاست که در این مقاله از آن استفاده می شود. با استفاده از امکان (( قلم )) در این زبان، می توان حرکت قلم درشت یک استاد خوشنویس را شبیه سازی کرد. استفاده مستقیم از متافونت برای طراحی کیفی فونتها کاری طولانی و همراه با مشکلات متعددی است. در این مقاله الگوریتمهایی پیشنهاد می شود تا بتوان تصویر پویا شده یک حرف را به صورت خودکار به برنامه متافونتی که آن حرف را تولید می کند تبدیل کرد. نرم افزار بر اساس این الگوریتم طراحی شده است که به صورت گرافیکی و با محاوره با کاربر این کار را انجام می دهد.

واژگان کلیدی: طراحی قلم، متافونت، منحنیهای بزیه، طراحی خودکار

## Automatic Design of Persian Typefaces

M. Ghodsi and K. Bazargan

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

**ABSTRACT-** In this paper, a fast method for automatic generation and scientific design of Persian letters is proposed. Scientific typeface design is an approach in which fonts are described by mathematical curves with well-defined parameters, where these parameters can be automatically tuned. METAFONT is a language suitable for the type of design used in this work. This language is particularly useful in designing Persian fonts because it can be used to simulate the pen movements of a calligrapher through automatic conversion of the scanned bitmap image of a font into a METAFONT program, which can in turn, produce the font at a high quality. A complete software has been implemented based on these algorithms that works interactively with the user to facilitate the font design.

**Keywords:** Font design, METAFONT, Bezier curves, automatic design

\*\* - کارشناسی ارشد

\* - دانشیار

## ۱- معرفی

یکی از مراحل مهم تولید هر سیستم حروف چینی فارسی، طراحی مجموعه ای کامل از خانواده های مختلف قلم یا فونت<sup>۱</sup> با کیفیت عالی است. منظور از یک خانواده از فونت، تمام حروف الفباست که با سیاق<sup>۲</sup> یکسان نوشته شده اند. مثالهایی از خانواده های قلم، (( نازک ))، (( سیاه ))، (( ایتالیک )) و (( ایرانیک )) هستند. یکی از مهمترین ویژگیهایی که در طراحی خانواده های 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 می شود، همگن باشد. همچنین تغییراتی که یک حرف از یک خانواده را به همان حرف در خانواده دیگر تبدیل می کنند، باید برای تمام حروف، مشابه باشند.

روشهای مختلفی برای طراحی فونت وجود دارد که از آن میان می توان به روشهای زیر اشاره کرد:

۱- طراحی تک تک حروف توسط خوشنویس و خواندن آنها توسط پویشگر<sup>۳</sup> و ذخیره سازی آنها به صورت بیت نگاشت<sup>۴</sup>. در این روش نمی توان تضمین کرد که حروف به صورت همگن تعریف شوند. همچنین حروف تولید شده توسط این روش را نمی توان به راحتی بزرگ و کوچک کرد.

۲- تعریف هر حرف به طور جداگانه و به صورت مجموعه ای از منحنیهای درجه ۲ یا ۳ و نگهداری آنها به صورت True Type و یا Post Script.. در این روش امکان استفاده از پارامترهایی که ویژگیهای حروف را به صورت 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 دهنده حروف، تغییر ناگهانی نکنند. تعداد این پارامترها و وابستگی آنها به هم نباید خیلی زیاد باشد. با توجه به ویژگیهای روش آخر، در این مقاله از آن استفاده می کنیم. یکی از بهترین ابزار طراحی فونتها با چنین روشی زبان متافونت<sup>۵</sup> است [۱]. متافونت یک زبان برنامه نویسی توصیفی<sup>۶</sup>

و سطح بالاست که اجازه می دهد حروف را با استفاده از پارامترهای مختلف تعریف و با تغییر یک یا چند پارامتر می توان خانواده های مختلف حروف را تولید کرد. در متافونت می توان قطعه های مشترک، و یا خیلی شبیه، حروف متفاوت را به صورت جدا تعریف کرد و سپس با ترکیب این قطعه ها با قطعه های غیر مشترک، حروف همگنی را تولید کرد.

متافونت زبانی توصیفی است که در سال ۱۹۸۲ توسط داندل کنوت<sup>۷</sup> برای طراحی فونتهای سیستم حروف چینی TeX ابداع شد. مهمترین قابلیت متافونت این است که به برنامه نویس اجازه می دهد معادله های توصیف کننده مکان و زاویه ها را به صورت (( ضمنی<sup>۸</sup> )) بنویسد. وقتی تعداد معادله ها به اندازه کافی رسید، متافونت مقدار متغیرهای مجهول را محاسبه می کند. متافونت به جای صدا کردن زیر برنامه، از بسط ماکروها استفاده می کند. این مسئله، خطایابی برنامه را مشکل می سازد.

با وجود این که زبان متافونت امکانات بسیار مفیدی در اختیار قرار می دهد، استفاده از آن بسیار مشکل و نیازمند دقت و صرف وقت فراوان است. در این مقاله مباحث نظری و الگوریتمهایی را پیشنهاد می کنیم تا بتوان تصویر پویش شده یک حرف را به صورت خودکار به برنامه متافونتی که آن حرف را تولید می کند تبدیل کرد. بر این اساس محیط نرم افزاری محاوره ای<sup>۹</sup> را طراحی کرده ایم که در این مقاله گزارش می شود.

## ۲- پایه های نظری و الگوریتمها

نرم افزاری که طراحی شده، مجموعه ای از ماکروها به زبان متافونت و یک محیط گرافیکی تحت (( ویندوز )) است که امکانات مناسب برای تولید نیمه خودکار برنامه های متافونت را در اختیار می گذارد. ابتدا، کاربر تصویر پویش شده حرف مورد نظر خود را تهیه می کند. این تصویر می تواند خاکستری یا سیاه و سفید باشد. سیستم به دقت تصویر خاصی وابسته نیست ولی معمولاً بهتر است ابعاد مستطیل محیطی حرف، حدود ۱۰۰۰ نقطه در ۱۰۰۰ نقطه باشد. این امر باعث

می شود ویژگیهایی از منحنی مرزی مثل زاویه، انحنا و موقعیت، با دقت مناسبی تشخیص داده شود. شکل (۱) تصویرهای پوشش شده حرف (( ب )) و (( ن )) را نشان می دهد.

سپس نویز تصویر برطرف می شود و منحنی مرزی آن به دست می آید. کاربر می تواند منحنی مرزی را اصلاح کند: نقاطی را حذف یا اضافه کند و یا آستانه حذف نویز را تغییر دهد. همچنین، کاربر می تواند مشخصات منحنی، از جمله زاویه مماس بر آن، را در هر نقطه مشاهده کند. تمام اعمال خودکار برنامه، بر روی منحنی مرزی بسته حرف، و یا قطعه منحنیهای وابسته به هم انجام می شود. منحنی مرزی بسته، توسط روشهای خودکار و نیز، در صورت لزوم با دخالت کاربر، به قطعه هایی تقسیم می شود و هر قطعه، با یک منحنی بزیه تقریب زده می شود. سپس مکانهای عبور قلم درشت، تعیین می شوند و با توجه به مشخصات منحنی در آن نقاط، برنامه متافونت، تولید می شود. در انتها، خروجی متافونت پس از اجرای برنامه تولید شده با تصویر اولیه مقایسه می شود تا کاربر اشکالات برنامه متافونت را برطرف کند. جزییات مراحل بالا در ادامه می آید.

## ۲-۱- حذف نویز

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


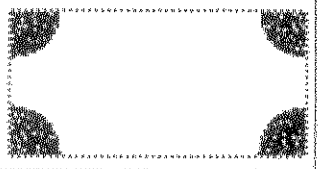
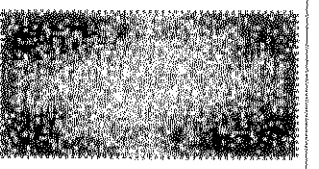
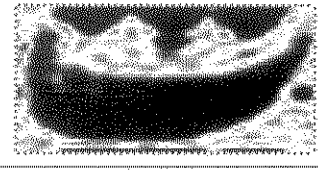
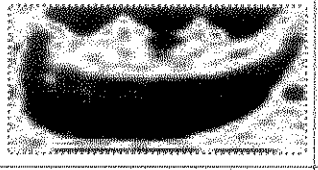
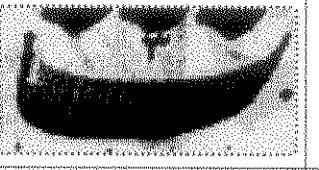
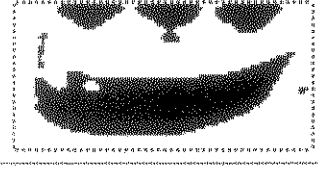
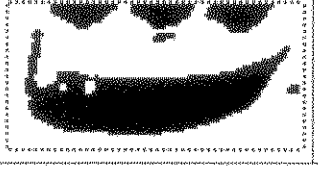
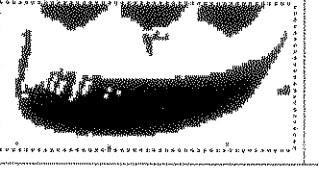
۱- تصویر خاکستری حرف را به صورت آرایه ای دو بعدی در نظر می گیریم که نمایان گر تابعی دوبعدی است. تبدیل فوریه دوبعدی [۵] (2Dim DFT) این تصویر را محاسبه می کنیم و آن را از یک فیلتر پایین گذر ایده آل [۴] می گذرانیم و تبدیل فوریه معکوس نتیجه را محاسبه می کنیم. حاصل، تصویری خاکستری خواهد بود که (( تار )) شده است. تصویر حاصل را با در نظر گرفتن یک آستانه روشنایی، به تصویری سیاه و سفید تبدیل می کنیم. این روش روی تمام تصویر به طور یکسان عمل می کند. برای حذف نویز در قسمتی از تصویر، باید پهنای گذر فیلتر را برای تمام تصویر، کم کنیم. این کار اغلب باعث

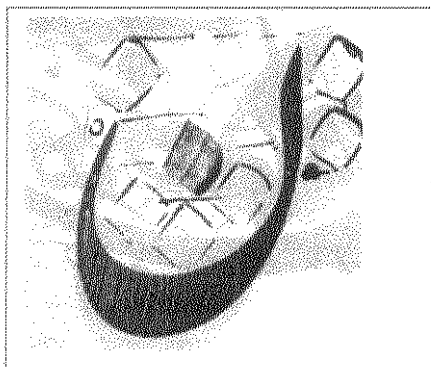
می شود دقت تصویر را در گوشه های حرف یا بقیه قسمت های ظریف آن، از دست بدهیم. از آنجا که این نوع نویز ( یعنی نویز زیاد در یک قسمت از تصویر ) در حروف پوشش شده معمول است، و نیز به علت کندی ذاتی محاسبات تبدیل فوریه، این روش، روش مناسبی برای فیلتر کردن تصاویر ما نیست. مثالهایی از فیلتر کردن تصاویر با این روش را می توانید در جدول (۱) مشاهده کنید. همان طور که می بینید، با کم شدن پهنای فیلتر، نویز بدنه در قسمت چپ، حرف از بین می رود ولی تیزی گوشه بالا - راست نیز تحلیل می شود.

در دو روش بعد، ابتدا توسط الگوریتم کانتور چهارنقطه ای با آستانه تغییر روشنایی بین ۵۰ تا ۸۰ درصد، مرز حروف را به دست می آوریم. نمونه های منحنی مرزی به دست آمده از روش کانتور را در شکل های (۲) ملاحظه می کنید. از آنجا که الگوریتم کانتور تقریباً همیشه، یک مسیر بسته با انشعابات مانند آنچه در شکل (۳) آمده است می سازد و این انشعابات مشکلاتی را ایجاد می کند، خروجی الگوریتم کانتور را توسط یک الگوریتم جدا کننده مسیرهای انشعابی، ساده می کنیم و سپس عملیات فیلتر کردن نویز را روی منحنی مرزی حاصل انجام می دهیم. الگوریتم جدا کننده مسیرهای انشعابی را در انتهای این بخش آورده ایم.

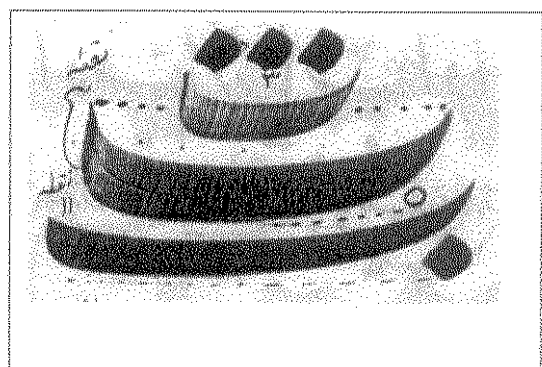
۲- منحنی مرزی را به توابع  $X = g_1(Y)$ ،  $Y = f_1(X)$ ،  $X = g_2(Y)$ ،  $Y = f_2(X)$  تقسیم می کنیم به طوری که این توابع، تمام مرز منحنی را ببوشانند و به صورت یکی در میان توابعی از  $X$  و  $Y$  باشند. سپس تبدیل فوریه هر قسمت را محاسبه می کنیم و نویز هر تابع را به کمک فیلتر ایده آل پایین گذر، برطرف می کنیم. به این ترتیب می توان هر قسمت از منحنی را که تابعی فقط از  $X$  یا فقط از  $Y$  است، با توجه به مقدار نویز، از فیلتری با پهنای باند مناسب عبور داد و نویز آن را حذف کرد. اشکال این روش این است که، به علت (( بسته )) نبودن منحنی  $f$  ها و  $g$  ها و در نتیجه تناوبی نبودن آنها، لبه های انتهای توابع به صورت نوسانی در می آیند و عملاً کارایی این روش از بین می رود و منحنی را در نقاط اتصال این توابع،

جدول ۱- مراحل مختلف و مقایسه نحوه فیلترگیری در روش اول

فیلتر با پهنای گذر ۱۵	فیلتر با پهنای گذر ۱۰	فیلتر با پهنای گذر ۱۵	مراحل روش اول
			تبدیل فوریه فیلتر شده
			تصویر خاکستری حاصل از تبدیل فوریه معکوس
			تصویر سیاه و سفید حاصل از قطع تصویر فیلتر شده

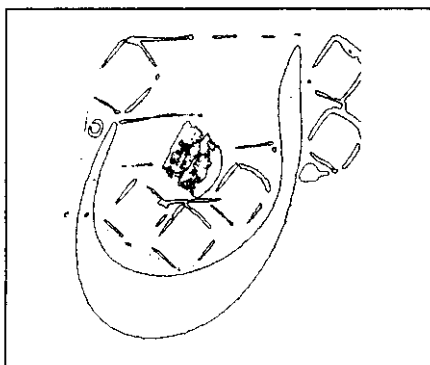


ب - حرف "ن" پوش شده

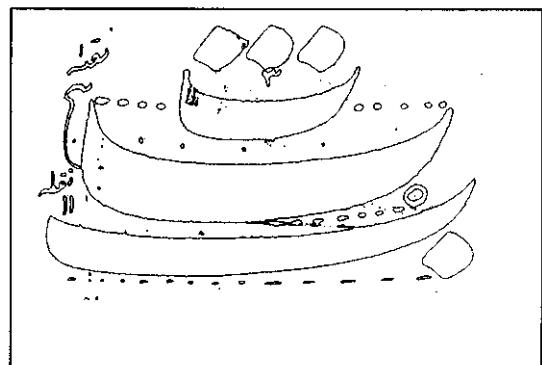


الف - منحنیهای "ب" ۳، ۵ و ۱۱ نقطه

شکل ۱- تصویرهای پوش شده دو حرف (ب) و (ن)



ب - حرف (ن) پس از الگوریتم کانتور با آستانه ۰/۷۰



الف - حرف (ب) پس از الگوریتم کانتور با آستانه ۰/۷۸

شکل ۲- تشخیص مرز با الگوریتم کانتور چهارنقطه‌ای



شکل ۳- انشعاب الگوریتم کانتور

فیلتر گیری گاوسی، به دلایل زیر برای کاربرد مورد نظر بسیار مناسب است :

۱- به صورت حلقه ای عمل می کند. یعنی اگر قسمتی از تصویر دارای نویز زیاد باشد، این نویز روی چگونگی فیلتر شدن منحنی مرزی تأثیری ندارد.

۲- محاسبات آن بسیار سریع است.

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

۴- نقاط به صورت پیوسته تغییر می کنند و خطای کوانتیزه بودن تصویر پوشش شده اولیه را ندارند. این خاصیت باعث می شود محاسبات مراحل بعدی، مثل محاسبه زاویه در نقاط مختلف، بدون نیاز به استفاده از روشهای تقریب زنده و روشهای محاسبات عددی انجام گیرد.

نمونه فیلتر شده شکل (۶- الف) حرف (( ن )) با پهنای  $\sigma = 5$  نشان داده شده است. تابعی که عمل فیلتر گیری روی آن انجام شده است، در شکل (۲-ب) آمده است. به حذف نوسانات زیر روی لبه های منحنی توجه کنید. مقایسه بهتری را شکل (۶- ب) نشان می دهد که منحنی سمت راست، منحنی فیلتر شده است. مقدار اندک تغییر منحنی در گوشه های تیز قابل توجه است.

### ۳- الگوریتم جداکننده مسیر انشعابی

در این بخش، الگوریتم جداکننده مسیره های انشعابی حاصل از الگوریتم کانتور را توضیح می دهیم. چنانچه اشاره شد، تصویر ابتدا توسط الگوریتم کانتور مرزیابی می شود و اغلب شامل مسیره های انشعابی است، مانند آنچه در شکل (۳) آمده است. بنا براین، پس از اجرای الگوریتم کانتور، الگوریتم زیر را اجرا می کنیم:

۱- مستطیل حاوی تصویر را به صورت آرایه ای دو بعدی از نقاط به نام Pic در نظر بگیریم. مجموعه C که در ابتدا تهی است منحنیهای جدا از هم را ذخیره می کند. این مجموعه، در پایان، خروجی الگوریتم خواهد بود.

نایپوسته می کنند. نمونه هایی از منحنی پایین حرف (( ب )) پنج نقطه در شکل (۴) آمده است. در شکل (۵) تبدیل فوریه همین منحنی، قبل از فیلتر شدن آمده است.

۳- منحنی را به صورت مجموعه ای از نقطه ها، با مختصات صحیح، در می آوریم: ( N تعداد نقاط منحنی مرزی است).

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

با توجه به اینکه منحنی مرزی، یک منحنی بسته است، می توان آن را به صورت توابع متناوب x و y تعریف کرد.

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

حال تابع گاوسی g را به صورت زیر تعریف می کنیم [۶]:  
( $\sigma$  پهنای منحنی را تغییر می دهد.)

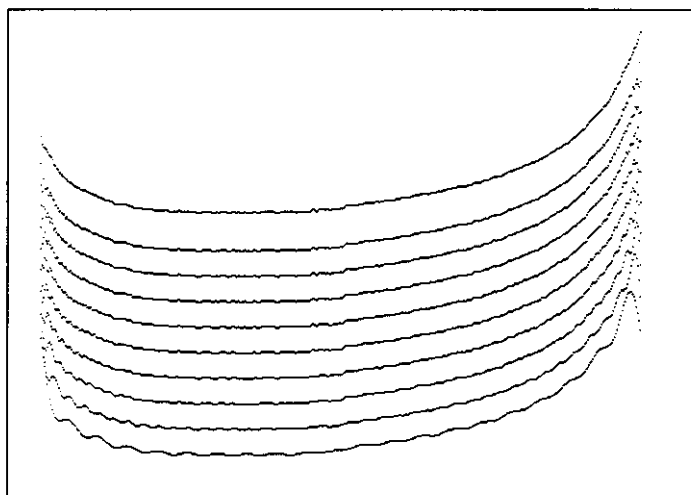
$$g_{\sigma} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{t^2}{2\sigma^2}}$$

تابع گاوسی را با توابع متناوب  $\bar{x}$  و  $\bar{y}$  کانالو می کنیم [۴]. تا توابع متناوب نرم و بدون نویزی به دست آیند. این توابع، دارای مختصات حقیقی هستند و نویز آنها به اندازه مطلوبی حذف شده است.

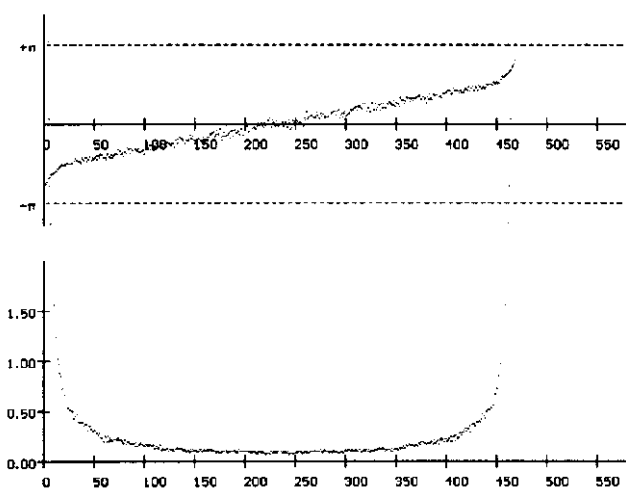
$$X_{\sigma} = \tilde{x}(t)g_{\sigma}(t) = \sum_{u=-\infty}^{\infty} \tilde{x}(t-u)g_{\sigma}(u)$$

$$Y_{\sigma} = \tilde{y}(t)g_{\sigma}(t) = \sum_{u=-\infty}^{\infty} \tilde{y}(t-u)g_{\sigma}(u)$$

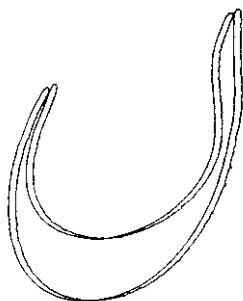
از آنجایی که مقادیر  $g_{\sigma}$  برای t های بزرگتر از ۵ خیلی کم است. می توانیم محدوده محاسبه convolution را از -۵ تا +۵ بگیریم و در نتیجه، سرعت محاسبات را بسیار بالا ببریم. روش



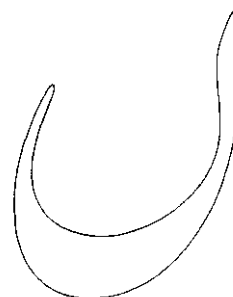
شکل ۴- منحنی پایینی (ب) پنج نقطه‌ای پس از عبور فیلترهای ایده‌آل پایین گذر با پهنای گذر بین ۲۰ تا ۱۸۰



شکل ۵- تبدیل فوریه منحنی پایینی (ب) پنج نقطه‌ای



شکل (۶-ب)- مقایسه منحنی فیلترشده با منحنی اصلی



شکل (۶-الف)- حرف (ن) که از فیلتر گاوسی گذشته است

۲- برای تمام نقاط  $Pic (X_0, Y_0)$  از تصویر، اگر سیاه است، آن را سفید کن و الگوریتم «دنبال کردن مسیر ساده»<sup>۱</sup> (که در زیر می آید) را با نقطه شروع  $(X_0, Y_0)$  اجرا کن. این فراخوانی باعث می شود که یک یا چند مسیر بدون انشعاب، به  $C$  اضافه شود. این مسیرها یک مسیر کانتوری در  $Pic$  را افزای می کنند. فرض کنید تعداد کل مسیرهای ساده که از تمام مسیرهای کانتوری به وجود آمده اند،  $N$  باشد.

۳- برای مسیرهای  $P_i$ ، برای  $i = 1 \dots N-1$ ،

۳-۱- برای مسیرهای  $P_j$ ، برای  $N$  و  $j = i + \dots$

۳-۱-۱- نقاط ابتدا و انتهای  $P_i$  و  $P_j$  را دوبه دو از نظر مجاورت با همسایگی  $\sqrt{2}$  بررسی کن. (اینکه آیا این دو مسیر از یک نقطه مسیر کانتوری منشعب شده اند.)

۳-۱-۲- اگر مجاور باشند، حتماً یک مسیر سومی هم در همین نقطه، با این دو مسیر، مجاور است. حلقه  $j$  را ادامه بده تا هر دوی این مسیرهای منشعب پیدا شوند. این مسیرها را  $first$  و  $second$  می نامیم. 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 نقطه ای که انشعاب از آن به وجود آمده است<sup>۲</sup>).

۳-۲- اگر دو مسیر پیدا شده است، (یعنی  $first$  و  $second$  هر دو مقادیر معنی دار دارند)،  $first$  را برابر مسیر بزرگتر 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 اولیه  $Pic$  تغییر نکند.

این الگوریتم، لزوماً بزرگترین مسیرهای ساده را به دست نمی آورد، چون به صورت کاملاً موضعی سعی می کند مسیر را گسترش دهد. ولی تقریباً همیشه، بزرگترین مسیرها را به دست می آورد. 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، به صورت زیر است:

۱- مسیر جدیدی به  $C$  اضافه کن که فعلاً فقط شامل نقطه  $CurPt$  (پارامتر ورودی) است.  $first-choice$  و  $second-choice$  برابر اولین و دومین نقاطی خواهند بود که در هر لحظه می توان از  $CurPt$  به آنها رفت. فعلاً آنها را برابر مقادیر بی معنی قرار بده.

۲- برای هر هشت نقطه مجاور  $CurPt$ ، اگر نقطه  $Pic(x, y)$  سیاه است، آن را در  $first-choice$  قرار بده و اگر  $first-choice$  قبلاً مقدار گرفته است، آن را در  $second-choice$  قرار بده.

۳-

الف) اگر هیچ کدام از  $first-choice$  و  $second-choice$  مقدار ندارند، به پایان این مسیر رسیده ایم. از الگوریتم خارج شو.

ب) اگر فقط  $first-choice$  مقدار دارد، آن را به مسیر اضافه کن و نقطه را در  $Pic$  سفید کن.  $CurPt$  را برابر  $first-choice$  قرار بده و از مرحله 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 نقطه  $first-Point$  به عنوان نقطه شروع، و بار دیگر با نقطه  $second-Point$  اجرا کن. این عمل باعث می شود که از این نقطه 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 ساده صاف و بدون نویز را توضیح دادیم. این مسیر ساده

۱ مسیر ساده است که هر نقطه آن حداکثر دو همسایه در ۸ نقطه مجاور خود داشته باشد. مسیری که انشعاب ندارد ساده است.

۲ در الگوریتم دنبال کردن مسیر ساده، فقط وقتی از اضافه کردن یک نقطه به مسیر ساده امتناع می کنیم که غیر از نقطه قبلی خود در مسیر، یک همسایه دقیقاً مجاور داشته باشد. هر یک از این دو همسایه دقیقاً مجاور، یک مسیر ساده به  $C$  اضافه کرده است.

باید با قطعه منحنیهای 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. بنابر این اگر سیستم برای تقریب قسمتی از مرز حرف، حرکت قلمی را تعیین کند که متافونت نتواند آن را رسم کند، عملاً روشهای ارائه شده بدون استفاده خواهند بود. پس روشهای تقسیم منحنی و تقریب و نمایش آن به وسیله قلم درشت، باید مبتنی بر ویژگیها و قابلیت‌های متافونت باشد.

تعیین نقاط تقسیم مسیر ساده به زیر منحنیهای بزیه، بسیار مهم است. اگر این نقاط را به طور مناسبی انتخاب کنیم، تقریب زیر منحنیها با سرعت بیشتری انجام می شود و تعداد زیر منحنیها حداقل می شود. درباره چگونگی انتخاب این نقاط، در بخش بعد توضیح خواهیم داد. در این بخش، فرض می کنیم این نقاط را انتخاب کرده ایم و می خواهیم یکی از این قطعه‌های مسیر ساده را تقریب بزنیم. همچنین فرض می کنیم این قطعه مسیر، قابل تقریب زدن با یک منحنی بزیه باشد.

منحنی بزیه، مجموعه نقاط منحنی پارامتریک زیر است:

$$Z(t) = (1-t)^3 Z_{m1} + 3(1-t)^2 t Z_{c1} + 3(1-t)t^2 Z_{c2} + t^3 Z_{m2}$$

برای تقریب قطعه مسیری ساده با منحنی بزیه، و با توجه به داشتن نقاط ابتدا و انتهای منحنی بزیه  $(Z_{m1}, Z_{m2})$ ، کافی است  $Z_{c2}, Z_{c1}$  را پیدا کنیم. اگر مشتق منحنی بزیه را در نقاط ابتدا و انتهای آن محاسبه کنیم، فرمولهای زیر به دست می آید:

$$\left. \frac{dz}{dt} \right|_{t=0} = 3(Z_{c1} - Z_{m2})$$

$$\left. \frac{dz}{dt} \right|_{t=1} = 3(Z_{m2} - Z_{c2})$$

یعنی اگر بتوانیم مشتق  $Z'(t)$  را در نقاط ابتدا و انتهای قطعه منحنی بزیه تعیین کنیم [۲]، می توانیم نقاط کنترلی را به دست آوریم و در نتیجه قطعه مسیر را تقریب بزنیم. اما این کار، ساده نیست. قطعه منحنی بزیه، از مجموعه نقاط  $X$  و  $Y$  تشکیل شده که فاصله هندسی آنها، ارتباطی با فاصله پارامتر  $t$  آنها ندارد. به عبارت دیگر، سرعت پیشروی و حرکت منحنی را

بر حسب  $t$ ، در نقاط مختلف آن، نمی دانیم. ( به عنوان مثالی از ارتباط فاصله فیزیکی با فاصله پارامتریک نقاط، شکل (۷-الف) را ببینید.) بنابر این نمی توانیم با محاسبه  $y'(x)$  (مشتق  $y$  بر حسب  $x$ )،  $y'(t)$  را به دست آوریم<sup>۱</sup>.

درحقیقت، بردارهای  $Z_{c1} - Z_{m1}$  و  $Z_{c2} - Z_{m2}$  سرعت حرکت منحنی نسبت به  $t$  را در نقاط ابتدا و انتها نشان می دهند. پارامتر  $tension$  در متافونت، اندازه این بردارها را کنترل می کند: با کم شدن این پارامتر، فاصله نقاط کنترلی، در همان راستا که بودند، بیشتر می شود. 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;

رسم شده است و منحنی نازکتر عبارت

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

رسم شده است. تأثیر  $tension$  بر روی نقطه کنترلی  $v$ ، نقطه  $w$  را تولید کرده است که به نقطه  $Z_{m2}$  با نسبت ۱ به ۲ نزدیکتر است.

توصیف  $Z_1, Z_0, \dots, W_1, W_0$  and  $\alpha, \beta$  tension  $\dots$  به زبان متافونت را در نظر بگیرید و فرض کنید مقادیر تمام پارامترها معلوم باشد.  $W_1$  و  $W_0$  بردارهای جهت منحنی بزیه در دو سر آن هستند. در ابتدا نشان می دهیم متافونت چگونه این پارامترها را به نقاط کنترلی  $u$  و  $v$  تبدیل می کند به طوری که عبارت بالا با عبارت  $Z_1 \dots$  controls  $u$  and  $v \dots Z_0$  معادل باشد. بعد از آن، الگوریتمی برای تقریب قطعه مسیر ساده با یک منحنی بزیه، ارائه خواهیم داد.

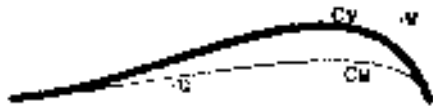
اگر زوایای  $\theta$  و  $\phi$  را به صورت زیر تعریف کنیم [۲].

$$\phi = \arg((Z_1 - Z_0)/\omega_1) \quad (1)$$

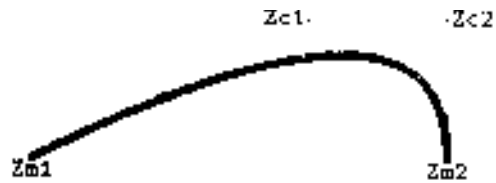
$$\theta = \arg(\omega_0/(Z_1 - Z_0)) \quad (2)$$

۱. 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 به دست 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 کنیم.





ب - اثر پارامتر **tension** بر روی نقاط کنترلی



الف - فواصل هندسی متناظر با فواصل مساوی **t**

شکل ۷- منحنیهای بزیه

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

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

که  $a$  و  $b$  نسبت اندازه بردارهای وصل کننده نقاط کنترلی در دو منحنی هستند. اگر این مقادیر را به دست آوریم، می توانیم از روی نقاط کنترلی  $B$ ، نقاط کنترلی  $Z$  را پیدا کنیم و در نتیجه منحنی  $C$  را تقریب بزیم.

اگر معادله های بالا را از هم کم کنیم (برای  $t$  ثابت و در نتیجه نقاط متناظر از دو منحنی)، معادله برداری زیر حاصل می شود

$$B_t - Z_t = 3(1-t)t^2 (a-1)U + 3(1-t)t^2 (1-b)V$$

که از حل آن، مقادیر  $a$  و  $b$  به صورت زیر نتیجه می شوند

$$a-1 = \frac{-B_{yt}V_x + B_{xt}V_y - V_yZ_{xt} + V_xZ_{yt}}{3(1-t)t^2(U_xV_y - U_yV_x)} \quad (5)$$

$$1-b = \frac{-B_{yt}U_x + B_{xt}U_y - U_yZ_{xt} + U_xZ_{yt}}{3(1-t)t^2(U_yV_x - U_xV_y)} \quad (6)$$

بنابراین، اگر یک تقریب اولیه از منحنی بزیه، مثل  $B$ ، داشته باشیم، می توانیم تقریب ایده آل را به دست آوریم. اما از آنجایی که نمی توان نقطه های کاملا متناظر دو منحنی، که  $t$  برابر دارند را پیدا کرد، باید با تکرار روشهای تقریبی، به جواب نزدیک شد چون ما  $Z$  را نداریم. برای پیدا کردن نقطه  $Z_t$ ، با توجه به فاصله هندسی  $B_t$  (نسبت فاصله هندسی آن از ابتدای منحنی بزیه)، سعی می کنیم  $C_t$  را به دست آوریم (با همان نسبت فاصله هندسی از ابتدا مسیر) و  $Z_t$  را برابر  $C_t$  بگیریم. براساس فرمولهای (۱) تا (۶)، الگوریتمی مکاشفه ای<sup>۱۰</sup>

مختصات نقاط کنترلی  $u$  و  $v$  با فرمولهای زیر محاسبه می شوند

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

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

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

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

برای تکمیل کردن مبانی پایه ای الگوریتم تقریب زدن منحنی، به روشی برای مقایسه دو منحنی بزیه نیاز است. فرض کنید  $Z(t)$  مجموعه نقاط منحنی ایده آلی باشند که (در صورت وجود) قطعه مسیر  $C$  را تقریب می زند، و فرض کنید منحنی بزیه  $B(t)$ ، یک تقریب اولیه از  $C$  است. می خواهیم  $B$  و  $Z$  را با هم مقایسه کنیم. نقاط شروع و پایان  $C$ ،  $Z$  و  $B$ ، و همچنین زاویه مماس بر هر سه در این نقاط، یکسان هستند. این شرط باعث می شود که نقاط کنترلی  $B$  و  $Z$  در یک راستا باشند و تفاوت دو منحنی، فقط در اندازه بردارهایی باشد که نقاط کنترلی را به نقاط اصلی وصل می کنند نه در زاویه آنها، اگر بردارهای  $U$  و  $V$  را به صورت  $U = Z_{c0} - Z_0$  و  $V = Z_1 - Z_{c1}$  تعریف کنیم، منحنیهای  $Z$  و  $B$  به شکل زیر تعریف می شوند

برای پیدا کردن مقادیر  $\alpha$  و  $\beta$  با فرض داشتن زاویه های  $\phi$  و  $\theta$  ارائه می کنیم.

۱- منحنی ورودی را که می خواهیم آن را با منحنی بزیه تقریب بزیم  $C$  می نامیم. نقاط  $Z_0$  و  $Z_1$  را مساوی نقاط ابتدا و انتهای  $C$  قرار بده. زاویه خط مماس بر  $C$  را در نقاط ابتدا و انتها محاسبه کن و با محاسبه اختلاف این زوایا و زاویه خطی که  $Z_0$  و  $Z_1$  را وصل می کند، مقادیر زاویه های  $\phi$  و  $\theta$  را، طبق معادله های (۱) و (۲) به دست آور. از آنجا که منحنی  $C$ ، یک منحنی صاف شده با فیلتر گاوسی است و خطای نویز و کوانتیزه بودن را ندارد، انتظار داریم مقادیر این زاویه ها تا حد مطلوبی دقیق باشند. هرچه این زوایا با دقت بیشتری تعیین شوند، الگوریتم سریعتر به جواب خواهد رسید.

۲- مقدار  $\alpha$  و  $\beta$  را برابر ۱ قرار بده.

۳- طبق معادله های (۳) و (۴)، مقادیر  $u$  و  $v$  را به دست آورد.

۴- با روش مقایسه ای برداری دو منحنی و اختلاف  $\alpha$  و  $\beta$  با مقادیر ایده آل را پیدا کن. این روش مقایسه برداری، به صورت زیر انجام می شود :

۴-۱- منحنی بزیه  $Z_1$  controls  $u$  and  $v$ ..  $Z_0$ ..  $B$  را بنام.

۴-۲-  $L_C$  و  $L_B$  را برابر طول هندسی  $B$  و  $C$  (تعداد نقاط کوانتیزه شده آنها) قرار بده. مقادیر  $\alpha_{ave}$  و  $b_{ave}$  را برابر صفر قرار بده.

۴-۳- مقدار  $t$  را برای  $B$  از صفر تا ۱ با قدمهای  $1/N$  افزایش بده. ( $N$  یک عدد ثابت است که هر چه بزرگتر باشد، در این مرحله از الگوریتم جواب را کندتر ولی دقیقتر به دست می آورد.)

۴-۳-۱- برای هر نقطه  $B_t$ ، مقدار  $g_t$  را برابر فاصله هندسی از نقطه  $B_0$  تا  $B_t$  قرار بده.

۴-۳-۲- نقطه متناظر  $B_t$ ، از منحنی  $C$  را (که فاصله هندسی آن از ابتدای  $C$ ، برابر است)  $C_t$  بنام.

۴-۳-۳-  $a_{ave}$  و  $b_{ave}$  را به اندازه  $(1/N)a$  و  $(1/N)b$  افزایش بده.  $a$ ،  $b$  از معادله های (۵) و (۶) به دست می آیند.

۴-۴- با استفاده از مقادیر  $a_{ave}$  و  $b_{ave}$ ، مقادیر  $\alpha$  و  $\beta$  را به مقدار ایده آل آنها نزدیکتر کن.

۵- مراحل ۳ و ۴ را آن قدر تکرار کن که با به تقریب خوبی از منحنی بررسی و یا تعداد دفعات تکرار مراحل، از حدی بیشتر شود. تقریب خوب منحنی وقتی به دست می آید که مجموع مربع فاصله نقاط متناظر هندسی  $C$  و  $B$ ، از حدی کمتر شود. یعنی

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

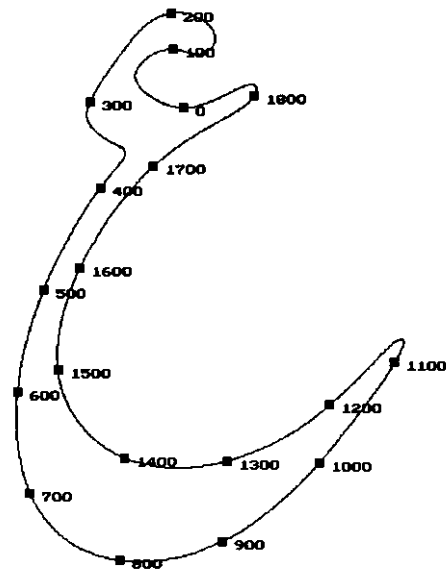
۶- اگر تقریب خوبی به دست آمد، متوقف شو. در غیر این صورت، زاویه های  $\phi$  و  $\theta$  را اندکی تغییر بده (در چهار نوبت، ترکیبات کم و زیاد کردن  $\phi$  و  $\theta$  را امتحان می کنیم) و الگوریتم را دو باره تکرار کن. اگر تقریب خوبی به دست نیامد، شکست الگوریتم را اعلام کن تا در انتخاب نقاط شروع و پایان، تجدید نظر شود.

## ۵- محاسبه ویژگیهای منحنی و تعیین نقاط قلم درشت

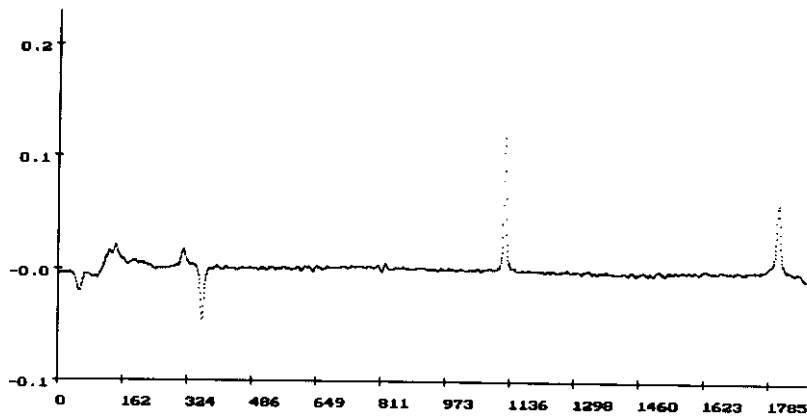
اگر بتوانیم ویژگیهایی از منحنی مرزی حرف، مانند انحنا منحنی، نقاط تیز، قطعه خطهای مستقیم را پیدا کنیم، می توانیم بعضی مکانهای قلم درشت را حدس بزیم. روشی که در [۶] پیشنهاد شده است از این نظر بسیار مفید است. اگر  $x(t)$  و  $Y(t)$  همان توابع فیلتر شده با فیلتر گاوسی باشند، و تابع انحنا  $K$  را به شکل زیر تعریف می کنیم

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

که در آن، مقادیر  $\dot{X}$ ،  $\dot{Y}$ ،  $\ddot{X}$  و  $\ddot{Y}$  به صورت زیر تعریف می شوند:



شکل ۸- نمونه‌ای از منحنی مرزی فیلتر شده



شکل ۹- تابع انحنای مربوط به شکل (۸)

در تابع انحنا، نقاطی که مشتق صفر دارند و نقطهٔ بیشینهٔ موضعی هستند را به عنوان گوشه‌های تیز منحنی در نظر می‌گیریم. نقاط متناظر با دو سر قلم درشت، انحناهایی تقریباً مساوی ولی در جهت عکس دارند.

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

- انحناهای دو سر آن، تقریباً مساوی و در جهت عکس باشند.
- زوایای دو سر آن، تقریباً مساوی باشند.

$$\dot{X} = \frac{dx_{\sigma}(t)}{dt} \quad \ddot{X} = \frac{d^2X_{\sigma}(t)}{dt^2}$$

$$\dot{Y} = \frac{dY_{\sigma}(t)}{dt} \quad \ddot{Y} = \frac{d^2Y_{\sigma}(t)}{dt^2}$$

و مقادیر مشتقها را می‌توان به شکل زیر تقریب زد:

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

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

به این ترتیب، میزان انحنای منحنی در نقاط مختلف آن به دست می‌آید. در شکل (۸) می‌توانید نمونه‌ای از منحنی مرزی و در شکل (۹) تابع انحنای آن را ببینید.

وظیفه چسباندن این قطعات به هم، و اصلاح قلمهای اشتباه، به عهده کاربر است. البته ماکروهایی در متافونت برای کمک به کاربر نوشته شده است.

## ۶- سیستم پیاده سازی شده ( محیط گرافیکی )

سیستم پیاده سازی شده، برنامه‌ای تحت ویندوز است که با مترجم Borland C++ Ver 3.1 ترجمه شده و از کتابخانه Borland OWL استفاده می کند.

پنجره اصلی برنامه، یک MDI Frame است و منوهای Tool، File و Zoom را دارد. در این برنامه می توان به طور همزمان روی یک یا چند تصویر PCX که تصویرهای پوش شده حروف هستند، کار کرد.

هر پنجره تصویر می تواند حداکثر یک پنجره برنامه (حاوی برنامه تولید شده متافونت) و حداکثر یک پنجره ویژگیهای انحنا ( حاوی مشخصات انحنا مسیر پردازش شونده ) داشته باشد. شکل (۱۰) نمونه ای از پنجره برنامه و تصویر را نشان می دهد.

در پنجره تصویر، تصویر PCX را ویرایش و منحنیهای مرزی ناحیه‌ای از آن را پیدا می کنیم. سپس این منحنیهای مرزی را فیلتر می کنیم و منحنی تقریب را می‌زنیم. محل وصل کننده

• مقطع قلم درشت، منحنی مرزی حرف را در نقطه ای غیر از دو سر قلم، قطع نکند.

• فاصله هندسی نقاط دو سر از گوشه های تیز، تقریباً یکسان باشد.

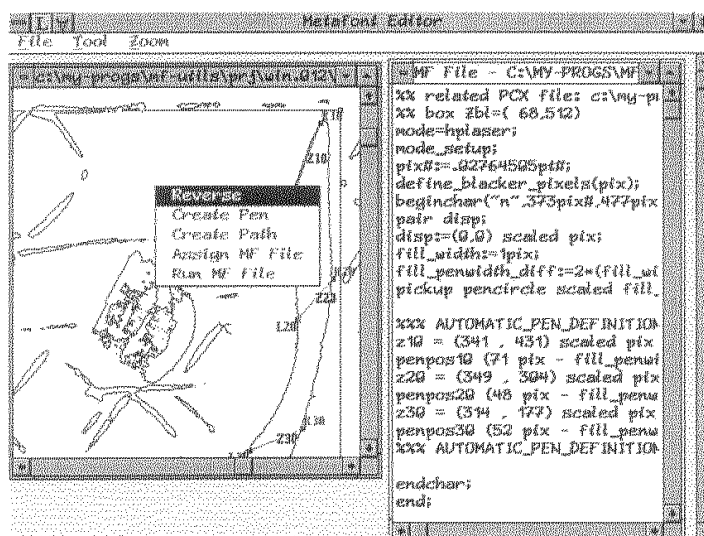
با توجه به این تعریف، الگوریتمی بسیار ساده برای تعیین مکانهای قلم درشت پیشنهاد می کنیم:

۱- اگر گوشه تیزی وجود دارد، از آن شروع کن. در غیر این صورت، از هر نقطه دلخواه دیگری شروع کن.

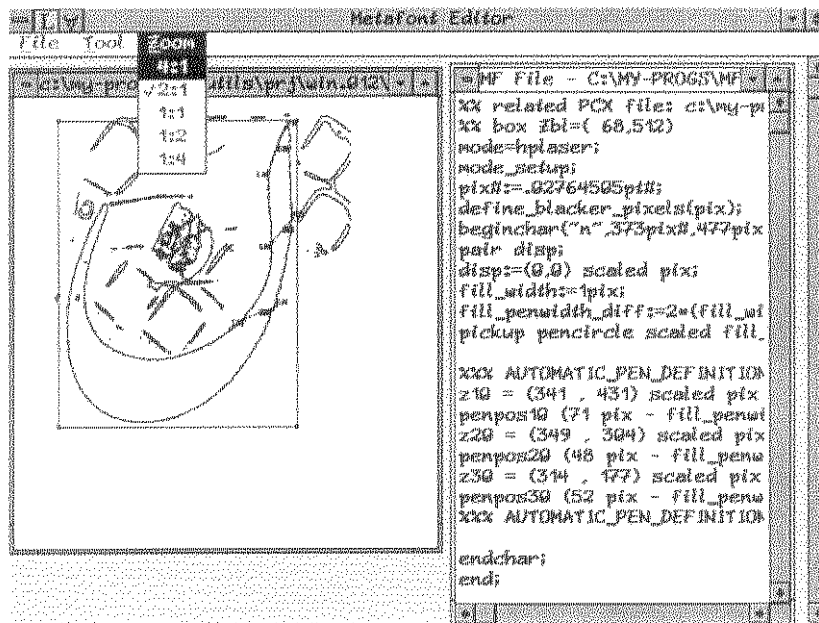
۲- در دو طرف منحنی X حرکت کن و نقاط متناظر را پیدا کن. نقاط متناظر آنهایی هستند که اگر دو سر یک مکان قلم درشت در نظر گرفته شوند، آن قلم سازگار باشد. آن قدر از دو طرف جلو برو که نقاط متناظر پیدا نشوند.

۳- سعی کن دو طرف منحنی بررسی شده را با دو منحنی بزیه تقریب بزنی. اگر الگوریتم تقریب، شکست خورد، منحنیهای دو طرف را نصف کن. ( و در نقطه تقسیم، یک قلم قرار بده) این کار را آن قدر ادامه بده تا یا اندازه منحنی تقسیم شده از حدی کوچکتر شود و یا تقریب منحنی به دست آید.

۴- عملیات بالا را برای تمام گوشه های تیز انجام می دهیم. این عمل باعث می شود قطعه هایی از حرف، تولید شوند.



شکل ۱۰- نمونه‌ای از پنجره تصویر و پنجره برنامه



شکل ۱۱- نسبت کوچک شدن ۱ به ۲

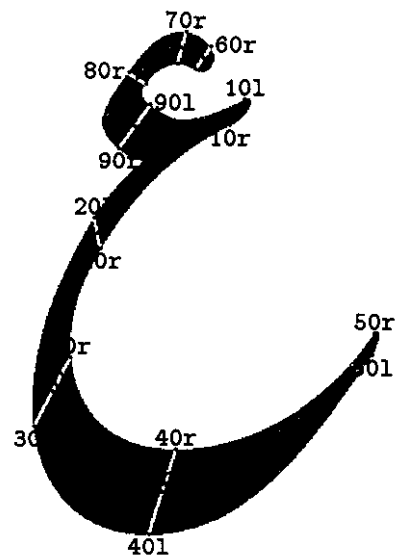
حرف به صورت خودکار تولید می شود و کاربر نیز می تواند آن را ویرایش کند. در پنجره ویژگیهای انحنا، کاربر می تواند ویژگیهایی از منحنی پردازش شونده، مانند میزان انحنا و زاویه، و نقاط متناظر با گوشه های تیز تشخیص داده شده توسط برنامه را ببیند.

در منوی Zoom می توان بزرگنمایی تصویر را تعیین کرد. نمونه هایی از استفاده این منو در شکل (۱۱) نشان داده شده است. آنها را با شکل (۱۰) مقایسه کنید.

در پنجره تصویر، می توان اشیایی، مثل مستطیل، قلم درشت را به وجود آورد. این اشیاء به صورت شفاف روی صفحه رسم می شوند. یعنی وقتی روی هم، یا نقاط زمینه قرار می گیرند، همه اشیاء روی هم قرار گرفته را می توان دید. هر یک از این اشیاء را می توان در صفحه حرکت داد، تغییر شکل و اندازه داد و با 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. نمونه ای از منو در شکل (۱۰) دیده می شود.



شکل ۱۲- حرف (ن) تولید شده



شکل ۱۳- حرف (ع) تولید شده و مکانهای قلم درشت

شکل‌های (۱۲) و (۱۳) نمونه‌های تولید شده این سیستم هستند.

## ۷- نتیجه گیری و گسترش‌های آینده

در این مقاله الگوریتم‌های به کار رفته در یک سیستم نرم افزاری برای تولید خودکار فونتها به زبان متافونت ارائه شد. سیستم پیاده سازی شده می تواند برای طراحی سریع و با کیفیت عالی حروف استفاده شود. در این سیستم تصویر پوشش شده یک یا چند حرف با استفاده از ابزارهای گرافیکی فراهم آمده به صورت نیمه خودکار به برنامه متافونت تبدیل می شود. این کار، با حذف نویز از تصویر، تقریب منحنی مرزی 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 می‌تواند 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. مسئله تشخیص خودکار محل قرار گرفتن قلم درشتها 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 می‌تواند مورد استفاده قرار گیرد. پروژه‌هایی هم در این زمینه در دست اجراست.

## واژه نامه

- |               |                 |                |
|---------------|-----------------|----------------|
| 1. font       | 5. METAFONT     | 9. interactive |
| 2. font style | 6. descriptive  | 10. heuristic  |
| 3. scanner    | 7. Donald Knuth |                |
| 4. bitmap     | 8. implicit     |                |

## مراجع

1. Knuth, D. E., *The METAFONTbook*, Addison-Wesley, 1985.
2. Knuth, D. E., *METAFONT: The program*, Addison-Wesley, 1985.
3. Knuth, D. E., *Computers and Typesetting*, Addison-Wesley, 1986.
4. Oppenheim, A. V., and Willsky, A. S., *Signals and Systems*, Printice-Hall International, 1983.
5. Gonzalez, R. C., and Wintz, P., *Digital Image Processing*, 2nd ed., Addison-Wesley, 1987.
6. Foley, J. D., Van Dam, A., and Feiner, S. K., *Computer Graphics: Principles and Practice*, 2nd ed, Addison-Wesley, pp. 478-491, 1990.
7. Xin, K., Lim, B. and Hong G. S., "A Scale-Space Filtering Approach for Visual Feature Extraction," *Pattern Recognition Journal*, Vol. 28, No. 8, pp. 1145-1158, Aug. 1995.