いま知るべきフリーフォントプロジェクト

東雲&東風フォントの全貌

東雲、東風フォントの制作にあたって

古川泰之
yasu@on.cs.keio.ac.jp

フォントはそれ自体に地味なイメージがあり、PC-UNIXコミュニティーでも、さほど話題にされてこなかった分野です。しかし、コンピュータのユーザーインターフェイスとして欠くことのできない部品の1つです。近年デスクトップ分野での活躍も期待されつつあるLinuxをはじめ、PC-UNIXにとってもそれは同じです。

話を日本語フォントとLinuxという観点に絞ると、現在は、主に商業フォントによって提供されることが多く、同時に、このことは、日本固有のLinuxビジネスモデルの一要素として確立され、重要な位置を占めてきました。

一方で、フリーフォントに目を向けると、膨大な漢字の制作コストの多さが障害になって、必要最低限のレベルからなかなか進展を見せられず、現在に至っています。

それでも、さまざまな才能を持つ幾人かの人たちによって、少しずつ、改善に向かって臨んでいます。ここでは、私が携わっているしがない（自由な再配布、改造が許可された）フリーフォントである、

- 東雲（しののめ）フォント（37ページのResource [1] を参照）。
- 東風（こち）フォント（ [2] ）

を話題の中心として、フリーフォントの経緯、制作過程から得た印象、現状の課題、そして、なぜフリーフォントが必要なのかということなどを順を追って述べていこうと思います。

■ フォントとは

まずはじめに、そもそもフォントとはどんなものなのかを非常に簡単に説明します。コンピュータ上では文字は文字コードという数値によって表されています。これを人間の目に分かるように表示するには、その文字コードに対応した画像、つまり字形データが必要になります。大まかにはこれがフォントです（**図1**）。

コンピュータ上では、明朝やゴシックといった書体と、フォントとを区別して扱うことが多いです。同書体でも、サイズごとにフォントが用意されていたりするためです。しかし、個人的には、コンピュータにおけるフォントと書体の区別は、書体はあくまで絵画的なデザイン（図面）であって、フォントは、それをピクセルなどによって実装したものだという私的な解釈をしています。

コンピュータ上で書体を実装する方法には、大きく分けると

・ビットマップ表現
・アウトライン表現

の2種類があります。前者はビットマップフォントともスクリーンフォントともいわれており、ピクセルパターンとしてデータを保持しています。前述の**図1**もこれにあたります。一般的に、ディスプレイなどの、低解像度の出力装置に向いています。代表的なものとしてはX Window System（以下X）で使われているBDF（Bitmap Distribution Format）フォントがあります。

後者はアウトラインフォントまたはスケーラブルフォントと呼ばれ、ピクセルの代わりに輪郭そのものをデータとして保持しています。実際は、2次Bスプラインや3次ベジェといった曲線式の制御点列です。一般的には、高解像度の出力装置に向いており、特に、印刷に向いています。こちらは、有名なものとしては、PostScriptフォントやTrueTypeフォントがあります。

Xでは、徐々に状況は変化しつつありますが、現在でもビットマップ形式のBDFフォーマット（または、データサイズを小さくしたPCF）がよく使われています。原理が簡単で処理が軽いという長所があります。また技術が枯れているため、インストールノウハウが豊富にあり、さらには安定して日本語が扱える、というのが背景にあるようです。

しかし、この方式は、単純な拡大縮小や斜体といった変形が（不可能ではないものの）不得手であるという欠点を持ちます。例えば、16ドットサイズのフォントを12ドットや20ドットサイズのフォントとしても利用すると、大抵は、ギザギザが目立ち、あまり綺麗ではありません（**図2-a**）。そのため1つの書体のために、複数サイズのフォントをつくらなければなりません。

一方アウトラインフォントは、書体の正確な情報を輪郭として保持しているため、拡大したり、解像度が高くなっても、綺麗に表示させることができます（**図2-b**）。また、アンチエイリアス技術によるグレースケールを使った表現を用いる

【図1】フォントの説明

"HELLO" は、コンピュータ上で、ASCIIコードで（16進数表記）、

48 45 4C 4C 4F

という数値データ列で保存されている。これでは人間の目にはよめない。

48には	45には	4Cには	4Fは
@....@.	@@@@@@.	@.....	.@@@@.
@....@.	@.....	@.....	@....@.
@@@@@@.	@@@@@@.	@.....	@....@.
@....@.	@.....	@.....	@....@.
@....@.	@@@@@@.	@@@@@@.	.@@@@.
.....	.....	.....	.....

という画像データを割り当て、言葉として認識できるように表示している。

【図2】フォント形式による出力の違い

（a）ビットマップ　（b）アウトライン　（c）アウトライン＋アンチエイリアス加工

と、低解像度のディスプレイ上でも、ドットの粗を目立たなく表示することが可能です（**図2-c**）。

ただし、アウトラインフォントが万能かというとそうともいえません。ビットマップフォントでは、複雑な文字が真っ黒になって潰れてしまわないように、デザインを特定サイズに向けてチューニング（間引き、あるいは省略）しています。そのため、オリジナルサイズでの可読性はアウトラインフォントより優れているというのが通例です（**図3**）。

現在主流である米Microsoft社（以下MS）のWindowsでは、日本語フォントに関しては、大きな表示ではアウトラインを用い、小さなサイズではビットマップを使うというハイブリッドな構成を採用しています。これは、ビットマップフォントの組み込みをサポートするTrueTypeフォント形式を用いて実現されています。このようなWindows用フォントファイルをPC-UNIXでも用いることは技術的には可能ですが、ライセンスという法的な問題を抱えており、フリーかつオープンなPC-UNIXに見合った完全な日本語フォントは、なかなか環境が整っていないのが現状です。

■昔のUNIXの日本語フォント

私が大学の研究室に入った頃は（1996年）、Sun OSを含むUNIXマシンが多く、WindowsやPC-UNIXが徐々に研究室に浸透していった時代でした。当時、UNIXデスクトップで当たり前に使われていたフォントとして、k14フォントがありました。これは14ドットサイズのビットマップ明朝フォントです。1987年頃、橘浩志氏が学生時代1人で作成されたというエピソードも有名で、X用に最初にcontributeされた日本語フォントとしても知られています。まさにフリーフォントのパイオニア的存在です。

このフォントは非常に美しく、そして正確さにも定評があり、現在でもデフォルトフォントとして多くのディストリビューションで使われています。フォントという、ソフトウェアとしては毛色の異なるものでありながら、第4回フリーソフトウェア大賞（1996年）に選ばれたのも頷けます（[3]）。

このフォントがなければ、日本のPC-UNIXの普及速度はもっと遅れていたものと容易に想像できましょう。

また、1998年にリリースされた永尾制一氏のNAGA10フォント（[4]）や、ビットマップの高品質な太字フォント作成ツールmkbold（[5]）には感動した人も多いかと思います。私も、フリーフォントでも、ガッツさえあればここまで品質が高いものをきちんと完成させられるものなのだと、感心しました。

このように個々に目を向ければ光るものは存在していましたが、この頃は、全体的に種類が乏しく、フリーフォントをどれだけ寄せ集めても、Netscape用にサイズを揃えるのが精一杯でした。まして、明朝、ゴシックなどの書体を区別して一式揃えるなんてことは夢のまた夢でした。また、やはり別々の人間が別々のルールでデザインしたフォントの寄せ集めですから、HTML文書をパッと見たときに、貧弱な感はぬぐいきれませんでした（中途半端に不満をたれると、それは贅沢だと怒られましたね……）。

特に、WWW（World Wide Web）及びHTML文書が徐々に生活に欠くことのできないものへと変化するにつれて、PC-UNIXにおいてもフォント環境の改善への機運が高まります。そこで、日本の開発コミュニティが誇るX-TT（X-True Type Server）が開発及びリリースされました（[6]）。これを使うことで、初めてPC-UNIX上で、TrueTypeフォントを表示することが可能になりました。さらに、和田研ゴシックや渡邊明朝の再配布可能なフリーのTrueTypeの2書体も同時にリリースされ、PC-UNIXの裾野の拡大に大きく貢献したと、多くの人は認識しているのではないでしょうか。

■東雲（しののめ）フォントの制作

しかし、X-TTで、渡邊明朝及び和田研ゴシックを、初めて実際に表示させてみて正直ちょっとがっかりした人も多かったのではないでしょうか？

フォントの品質ウンヌンより、とにかく普段使うサイズである10ドット～16ドットサイズのレンダリング結果が、実用に耐えられる結果ではないのです。結局、もったいないと感じつつも今まで通りビットマップフォントを使い続けた人も多いかと思います。

学生の頃、個人的に、SolarisのSunフォント（ビットマップ）のゴシックが非常に気に入っていて、そこでNetscapeブラウザにだけはこれをこっそりと使っていました……。が、やはり、いつまでもそのようなことを続ける訳にもいかず、これに似た代替物が欲しくなりました。

そこで、ライセンスに注意を払いつつ、とりあえず遊びで、改変可能であった12ドットの要町フォントや、k14フォントから仮名文字だけを自分の好みに修正するだけ、という非常に簡単な「まねごと」を始めました。このころはkaname-alterやk14gothという名前で呼んでいました。

しかし、やり始めると惰性が働き（?）、少しずつ漢字の方も独自の作成を始めました。このとき、やはりk14フォントが私に及ぼした影響は大きくて、デザインのお手本として多いに参考にさせていただきました。お手本の候補としては、他にMSゴシックやSunフォントなども考慮したのですが、著作権を意識したのと、何より長い間に渡って使用していた（もはや生活の一部と化していた）k14フォントが、PC-UNIXの顔にふさわしいフォントだと考えたため、選択に迷いませんでした。この選択に関しては多くの人に同意していただけると思います（結果として、東雲ゴシックが、他のゴシックフォントより、シャープというか、ほっそりした印象があるのはこのためです。ただ、やっぱり「まねごと」なので、私ごときが何をやってもオリジナルのk14には足元にも及ばないなと今でも思っております……）。

この、勢いだけで続いた地道な積み重ねの結果、さらにjiskan16をテンプレートとして作られた16ドットフォントも加え、最終的に、12／14／16ドットの明朝およびゴシック書体のビットマップフォント「東雲フォント」が完成しました（**表1**）。

途中で、おまけとして上記の太字生成フィルタmkboldに大いに影響を受けたmkitalic（ビットマップフォント用の斜体フォント生成フィルタ）という副生成物もできあがり、私が日常の生活に欲しかったものがだいたい揃いました（ビットマップの斜体フォント生成フィルタは、他に見付からないので、世界唯一!?）。

余談ですが、このmkitalicは、結構頭の体操になって面白かったです。アウトラインフォントならば、斜体生成は1次変換によって簡単にできますが、ビットマップフォントではそうはいきません。もともと誤差を含んだビットマップパターンを無理に斜体にするとさらに誤差が拡大して大抵は汚くなります。

最初、誤差を軽減するため、内挿処理を付けて傾かせ、これを2値化してできないかなどを

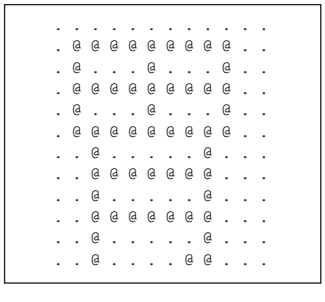
【表1】東雲フォントが補完した領域

	明朝	ゴシック	丸文字	備考
10ドット	○	○	○	NAGA10
12ドット	●	○●	●	要町
14ドット	○●	●	○	k14
16ドット	●	●○	○	jiskan16
20ドット	○			Kappa20
24ドット	○			jiskan24

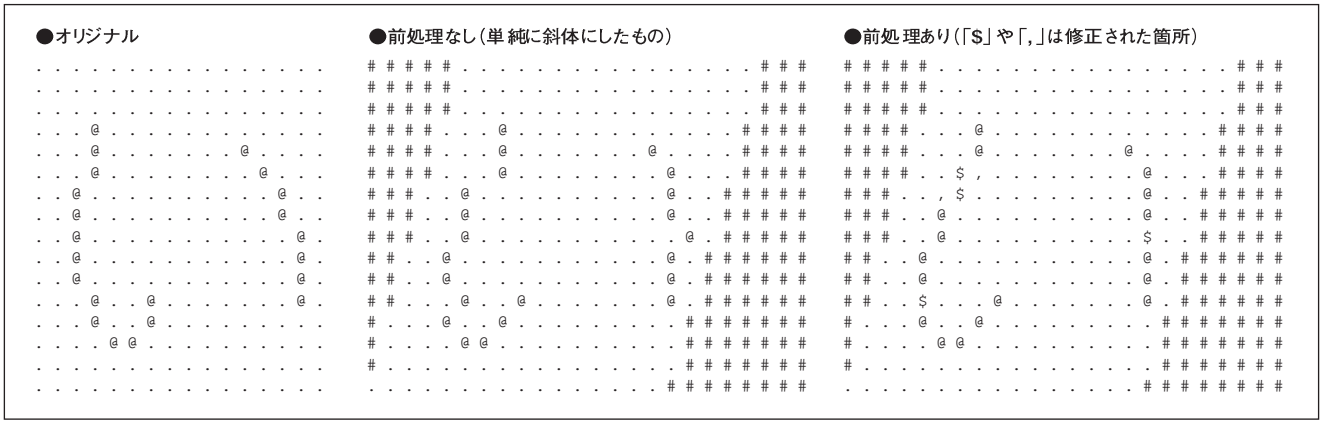
● 東雲フォント
○ 従来からあったフォント（備考欄 参照）

【注意】○●両方ある箇所は、東雲フォントのテンプレートフォントとして備考欄のフォントを使ったフォントと一致しています。

【図3】ビットマップフォントの工夫わざと画数を削り、漢字を可読性をあげている（東雲フォントの「胃」の文字の例）



【図4】mkitalic の出力結果。東雲16ドット「い」のを使って前処理のありなしの違いをみたもの



Gimp上でいろいろシミュレーションしていました。結局自動処理では満足いく結果が得られず、そのため、単純に斜体処理として横にずらしても線が汚くならないような置換パターンを人手でいくつか探りました。これを辞書としてあらかじめ複数用意しておき、それをパターンマッチングさせ、置換していくという前処理を加える手法に落ち着いています（**図4**）。

と、余談はさておき、自分で言うのもなんですが、これだけの量にもかかわらず全部で1年半しかかかっていません。これは、自分で一から作ったのではなく、k14や要町といったベースとなるフォントが存在したからこそです。ベースとなるテンプレートフォントがすでにあることで、試行錯誤する期間をずっと短縮できたり、さらには、JISの規格票と睨めっこすることも全くありませんでした。

さらには、途中成果物を公開することで、仮に規格とずれていても、ご丁寧に狩野宏樹氏（[7]）をはじめ、いろいろな方からご指南をいただき、私は力仕事だけに打ち込めました。こういった先人の偉大な成果物や、周りの人達とのコミュニケーションによった部分は本当に大きいです。このような制作スタイルは、後の東風フォントにもそのまま引き継いでいます。

蛇足ですが、東雲フォントの名前の由来は、学生時代のサークル活動で最後に人形劇／影絵劇を公演した、山梨県のとある小学校の名前を取っています（ああ、やっぱり蛇足ですね）。

また、ちょうどこのころ、フリーのJIS X 0213ビットマップフォントの発表もいくつか行われたり、RINGオープンラボ「/efont/」（[8]）の立ち上がりなど、にわかにフリーフォント周りに少し勢いが出てきました。幸い、/efont/の方が東雲フォントのcontributeを快く引き受けてくださり、私は次節のアウトラインフォント制作のための試行錯誤に集中することができました（感謝!）。

以下補足ですが、k14gothは、後の/efont/へのcontributeとともに、東雲ファミリーという名前になりました。私は偉大なk14フォントの名を上書きすることに抵抗があったのですが、/efont/の方から、管理の分かりやすさと、フォント名がサイズごとにバラバラだと不便という意見があり、こうなっています。他意はありません。また、このとき新たに追加された14ドット明朝フォントは、k14をスクリーニング化（**図3**）した以外は、k14と全く同じものということも断っておきます。

【表2】

Level 1	個人がデスクトップ内の使用に耐えられる（メールやブラウザや印刷プレビューで不満がない）
Level 2	先輩に提出しても失礼がない（グループワークの輪講会やプレゼンに耐えられる）
Level 3	上司に提出しても失礼がない（社内文書に適用できる）
Level 4	お客様に提出しても失礼がない（名刺や、お客向けのちょっとした案内書など）
Level 5	出版にも耐えられる（ライターや小説家が、自分の作品をはべらせるフォントとして耐えうる）

■東風（こち）フォントの制作へ

東風フォントは、ビットマップの次のステップとして行われました。不幸にもJava、GNOME、あるいはKDEといった続々と新しく登場してくるアーキテクチャが、リッチなフォント環境を要求するように作られており、いよいよビットマップフォントのみでは十分な環境を整えるのが難しくなってきたからです。

フリーフォントとてしてすでに定番になっている渡邊フォントや和田研フォントがあるじゃないかという気もします。これらは確かにフリーのレンダリングエンジンの開発に大いに役立ったと敬意を覚えています。が、フリーソフトウェアを作るのに高尚な大義や理由はいらないと思います。ただなんとなく、もう少し品質をbetterにした、せめて生活に耐えうる程度のフリーなフォントが欲しかったというのが希望でした。

なので、特別高いモチベーションがあったわけではなかったのですが、どちらかといえば、ビットマップフォントも一通り終わり、やはり惰性でトリガーを引いてしまったという方が強いです。

●目標レベルをどこに置くのか?

アウトラインフォントは、特に、自由度がビットマップフォントと比べ圧倒的に高い分、目標の設定を、どこら辺に置けばいいのか分からなくなります。この自由度は、凝ろうと思えばどこまでも凝れる可能性を持っていますが、一方で作った人間のセンスの良し悪しまでも、そのまま忠実に表現してしまう怖さも持ち合わせています。

私は、タイポグラフィや文字コードの規格をしっかり勉強して、オーサライズされることを目標にするよりは、普段の実生活で使えるものを作ることを目標にしました。不適切な例えかもしれませんが、「Minixを作りたいのではない、Linuxを作りたいのだ」だと、主張しています……。

この目標をもっと明確にするためにユースケースを具体的に上げ、**表2**のよう大雑把なレベルを設けています。この指標は、他の人とのコミュニケーションの際に、一定のコンセンサスを取るために作ったものです。また、「ここのデザインディテールは省略すべき」とか「いや省略してはならない」などのタイポグラフィの専門的な議論が好きでなかった（というよりついていけない）ので、そういう議論を避けて、私（初心者）でも理解できる物差しとして作りました。

東風フォントの目標は、Level 2をおおよその目標として、あわよくば、Level 3でも使えるというところに設定しています。

目標レベルの設定が低いように見えますが、PC-UNIXを使うことが多い、研究者や開発者のワークスタイルを考えると、Level 3の範囲内で、生活時間の90%以上を過ごしているという個人的な想像が働いています……。何より、Level 4以上はもはや美術工芸のレベルであり、素人が足を踏み入れることは不可能だろうという言い訳も付け足しておきます。

●どう作るのか

目標を定めたところで、これを達成するのに

何が必要かを考える必要がありました。そして、揃えるべき道具として挙げたのは、以下の3つです。

1　日本語TrueTypeフォント編集ツールは安く入手できるか？
2　ビットマップ埋め込みは本当にできるか?
3　元と成る素材はあるか？

フォント作成ツールは特に日本語をサポートするものとなると、一般的に非常に高価です（10万円近くする）。しかし1に関しては、オープンソースなものは見付からなかったものの、幸いTTeditというWindows用シェアウェア（[9]）をすぐに見つけることができました。

また、上記のLevel 1の目標をクリアするために、2のTrueTypeフォントにビットマップフォントを埋め込むことがどうしても必須でした。これについては、偶然に、前述した狩野宏樹氏の膨大なフォントに関するリンク集をたどっていったとき、Microsoft社のタイポグラフィグループ研究所（[10]）が公開している、Sbit32というツールの存在を知ることができました。特筆すべき点として、このツールがXのBDFフォントとの親和性も優れていたことで、簡単に実現の可能性を検証できました。このおかげで、NAGA10、k14、Kappa20といった、従来のX用フリーフォントの成果群をほぼそのまま活用することへつなげられました。

そして、最後の3に関しては、眼力が乏しい私が言ってもあんまり説得力がないのですが、渡邊明朝フォントの骨格が十分に見ごたえがあると思い、これを使うことにしました。渡邊フォントは、アウトラインフォントとして普及していますが、起源がビットマップフォントであるため細部デザインに関しては見劣りする箇所があります。しかし、これをスケルトンとして、TTeditを用いて細かい輪郭を調整すれば、意外と目標とするLevel 2も何とかなるかもしれないという憶測が沸いてきました。

これでほぼ道具は整いました。この次に考えなければならないのは、ビットマップフォントとアウトラインフォントの違いをできるだけなくすことでした。そこで、VFlibとcb2ttを用い、パラメータを調整して、ビットマップフォントに合うように渡邊フォントを拡大したTrueTypeフォントを作り直しています。

最後に、渡邊（明朝）フォントが和田研（ゴシック）フォントより太いのを嫌って、細字フィルタをいったんかけ、とりあえず仮名文字だけを多少調整してできたのが、最初に公開したwatanabe-light（東風明朝の前身）になります。

この3つの道具の中でも特に、改変が可能な渡邊フォントによってデザインの骨格が用意されるということは、非常にありがたいことでした。全体のデザインバランスについて考えなくてもよく、細部だけに注意していればよいのです。最悪、ツールはお金を払えばなんとかなりますが、センスはお金では買えません。これで素人でもできる訳です！

このように、タイポグラフィの知見を全く持たなくても、ここまでのことなら可能だということは、ぜひ、覚えておいてほしいと思います。

さあ、後はもう、ひたすら力技です。詳しい作り方は、後述する「東雲フォントを作ろう！」へまわします。ぜひ、雰囲気だけでも感じ取っていただければと思います。

そして、制作しながら、一番最後に行ったことがあります。それはフォントの名前付けです。これには散々悩まされ、他の方からいくつか案もいただいたのですが、結局、自分で出した東風（こち）という名前を付けました。これは埋め込みビットマップフォントに由来しています。東雲の「東」、Kappaやk14フォントの「K」で始まる名詞、そしてNAGA10の2文字発音からきています。

■フォント制作雑感

まだまだ東風フォントは制作途中ですが、これまでのこのフォントの制作過程を通して、さまざまな出来事や出会い、そしてインタラクションが起こり、これによっていろいろと得たことと苦労したことがありました。

●分かったこと

本当に全く知識がない手探り状態から始めましたが、制作する過程でいろいろタイポグラフィに関して実践的な知識をよそから学んだり、感銘を受けたりしました。

例えば、平仮名を漢字よりも少し小さめにデザインしたほうが、漢字を一瞬で把握しやすく、認知負荷を下げられる（[11]）というアドバイスを聞いたときは、ただボーッと眺めていただけのフォントに対して、今まで認識しなかった奥の深さを知りました。

また、現在の東風明朝書体では横線の左側を微妙に膨らましています（図5）。最初は、「絶対、こんなものは普通の人は見ていない！　作る側の、ただの自己満足だ」と直線的にデザインしていました（実際、MS明朝はまっすぐにデザインされています）。

ところが、/efont/の方から、渡邊フォントは「鱗」が大きい傾向があり、そのため、左端を膨らました方がバランスが取れるという、具体的、論理的なアドバイスを受けまして、感銘を受けたことがあります。まあ、それでも小さいサイズでのレンダリングにはほとんど影響しないのですが、それよりも、この変更を実際に体験したことは重要でした。フォントデザインに必要な、曲線に対する微妙なバランス感覚を学べたおかげで、他の部位のデザインにも後々大きく影響しています。

こういったコミュニケーション、あるいはインタラクションを交わすことで、初期から随分と品質レベルを発展させられました。本当に完全に1人だけでやっていたら、今のレベルに達することは決してなかったと思います。

このようなコミュニケーションは、さらに、自分とは違う「温度」や、思想に触れることへもつながり、「文字」という今まで空気みたいに感じていたものに対するメタな見方をずいぶんと広げてくれたと思います。

例えば、汎用フォントの場合、喜怒哀楽、全ての表現に使われるフォントです。丸文字みたいにはっきりとしたキャラクタが存在しない、こういうデザインは、素人にとって一番悩ましく難しいのです。その上、高度に様式化されている訳ですから、プロの方々は、日々相当努力されているのだと思います。

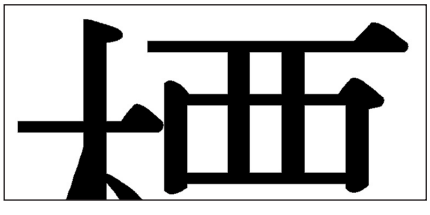
「メールは味気ない」、「文字コードだけでは感情は伝わらない」、あるいはマルチメディアだのと時代の風は流れていますが、そんなことを考えているうちに、単にそれはテキスト、いや文字に対する理解が人間にはまだ足りないのかなとも考えるようになりました（単に年寄りくさくなった、との見方も……。あと「字がうまくなるかなー」とかひそかに期待したのですが、これはならないですねえ……）。

●苦労したこと

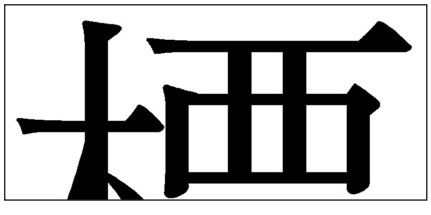
これまで述べてきたことと同時に、苦労した点も結構あります。フリーフォントの開発の何よりの難しさは、フォントはそれ自体、趣味性が非常に強いものだということです。趣味依存性が高い分、ソフトウェア開発に比べれば開発者間

図5】デザインの進展

(i) 初期のデザイン



(ii) 現在のデザイン



でコンセンサスを得るのがはるかに難しいと思います。実際にいくつか協業を打診したこともあります。が、残念ながら、直接的な協業にはなかなか折り合わず、今に至っています……。

もっとも、これは私の「なんとなくトリガーを引いた」というモチベーションやこだわりの低さが原因です。専門的な話についていけなくなり、こわばってばかりいました。それでも私も少しずつ勉強をして、フォント制作とはちょっと距離をおいたところですが、コミュニケーションは維持しています……。

さらに、フリーの汎用フォントを作っている人は国内にはほとんどおらず、そのため私がやっていることに周りから理解を得るのに、苦労しています。大抵は、「趣味でしょ?」と軽く小馬鹿にされて、終わりです。どうもコンピュータマニアよりフォントマニアの方が市民権が低いらしく、そういったレッテルが貼られるだけなのです……。フォントの地味なイメージを考えれば、これは普通の人間の健康な反応だと十分に理解できるのですが、私の実生活に悪い影響をもたらすので、結局、陰でこそこそと制作し続ける羽目に陥ります……(同時に、フォントは細かい作業が多く、意識しないとどうしてもストイックになりがちなので、精神衛生にも気をつけないとならない……)。

その一方で、公開してすぐに分かったこととして、使う方から要求されている品質レベルが予想をはるかに超えて高かったことで、これにも悩まされました(公開した当初はネットワーク上では偉く評判が悪く、実生活でも理解が得られず、「何で作ってるんだろう?」と自問自答する時期もありました……)。なんだかんだで、ベンチマーク対象として商業フォントと比較されることが多く、素人のフォントが受け入れられるには、あまりにも敷居が高いのです。この敷居の高さは、今までなかなかフリーフォントが開発されてこなかった原因の1つなのでは、と感じています。

さらに、作業が進み、徐々にデザインも改善されていくにつれ、商業フォントとの差が微妙になってきます。これは著作権上非常に危険で、どこまで模倣が許されるのかを、多少なりとも把握せざるを得なくなりました。もともと、明朝体やゴシック体という完成された非常に狭い様式の中でのフォントデザインです。

数値化されたフォントデータに関してはプログラムと同等の権利が発生しますが、デザインそのものに対して、どこまで認められているのかが不明でした。いくらオリジナリティを出してみたところでそれは明朝体、ゴシック体という様式の中での話であって、客観的に見た差は微妙なものにならざるをえません。

このときは、フォントの著作権に関して争われた裁判の事例を調べ回ったり、自分のフォントといくつかの商業フォントとを同時に並べて、周りの人に見比べてもらい、実際にフォントの差を識別できるかを聞き回ったりしました。そのため、基本的に、品質向上よりも、オリジナリティを優先させています。

これはぜひ知ってほしいのですが、書体のデザインに関する著作権者の立場は、相当に弱いらしいです。日本タイポグラフィ協会([12])やフォント販売企業、また個人も頭を相当悩ましていると聞きます。例えば、同じデザインなら、実装方法をちょっと変えるだけでも(このとき、数%の差異が生じる)、裁判ではOKになってしまっているケースがあります。それでも、日本タイポグラフィ協会のWebサイトを見るように、少しずつ事態を改善する方向で努力しているようです。

またこの影響で、できるだけお手本を見ないようにして、自分の乏しいイメージやセンスを元にデザインしています。それで随分長い間、デザインルールが安定しなかったです……。

●フリーフォントは必要か?

しばしば聞くメッセージとして、フォントは買えばあるのだから、それでいいという意見があります。確かに、商業ディストリビューションでは、立派な商業フォントがバンドルされ、一定の評価を得ています。PC-UNIXへ求めるものが人それぞれ違いますし、1ユーザーとしてはフォントだけ買って、その時はそれで済むのかもしれません。

しかしコミュニティ全体では、それでは困ることがあります。特に、プラットフォームフリーなフォントが求められているのは確かです。例えば、フォントを指定するワープロ文書も、ディストリビューション間では意図どおりの見た目で受け渡すこともままなりません。自分自身にとっても、ディストリビューションの乗り換えが非常に不便になるという欠点があります。そして、アプリケーション開発側(サードベンダー)は、まばらなフォント環境に対してデフォルトフォントの設定に非常に困っており、その負担はユーザーが負わなければならなくなります。

開発者にとっても、オープンソースで当たり前に行われるグローバル・コラボレーションではアプリケーションの国際化が必須になります。しかしライセンスフリーのフォントがないと、開発者がコンソールで動作を確認できません。試すのが不可能ならば、日本語処理に必要なマルチバイトコードの実装やテストが十分に行われない事態も発生します。

さらに、フリーフォントは、商業活動とまではいかない、初期段階のユーザーコミュニティレベルのディストリビューションや、PC-UNIX以外のOSの開発の活性にもつながります。例えば、

Column ビットマップフォントの制作を終えて

東雲ビットマップフォントの制作途中で、MSフォントなどの複数のフォントと一緒に出力、印刷して、周りにどれがいいのか、聞いて回ったことがあります。残念ながら、このとき、東雲フォントは結果はボロボロでした。

しかし、実に面白い現象が見られました。Macintoshユーザーは、絶対にOsakaフォントが最高だと主張し、Windowsユーザーは MSフォントが無難で実用的であると感じているのです。またSolarisユーザーはSunフォントこそが最も見やすいと言います。つまり、それぞれ自分の使用しているプラットフォームのフォントこそが最高だと擁護する傾向が見られました。

これは、別にフォント趣味のない、普通の人に聞いて回った結果です。言い過ぎかもしれませんが、毎日、意識せずに使っているフォントが、実は自然とそれを使う人のアイデンティティに寄与するという現象は、少なからず興味を覚えた記憶があります。

ただ、この時の唯一の例外が、PC-UNIXユーザーで……。学生のとき使っていたSolarisのフォントか、あるいは会社で使っているMSフォントがいい、または、どれも同じじゃんという人が多いように思いました。

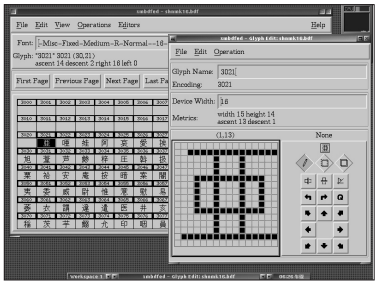
まあ、結果を見るまでもなく、東雲フォントがそういった顔を担えるだけのパワーを持っているといえる自信はありませんし、そもそもそういった野望があって作られたものでもありません。

しかし、k14や東雲のようなフリーフォントの面白いところは、ユーザーが改造する権利を持ち合わせている点にあります。育てる楽しみがありますし、ビットマップフォントのデザインは、学習コストも低く、結果も目で確認できるため、ちょっといじるには楽しいものです。

加えて汎用フォント作成の別の面白さとして挙げられるのは、どのアプリケーションからもほとんど必ずといっていいほど利用されるので、スクリーンショットとして、自分の成果が多くの雑誌に載るということです。これはフォントでしか味わえない面白さだと思います。

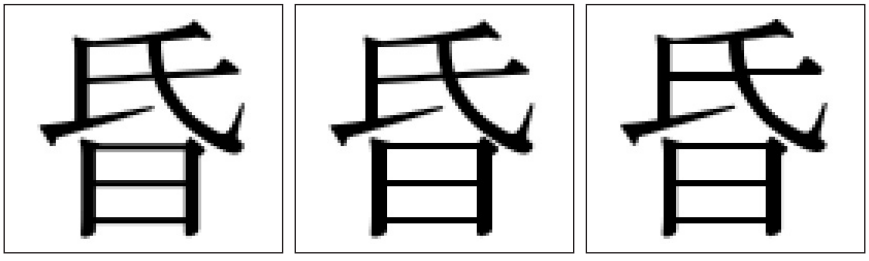
XmBDFEd([13]、画面A)などのGUIツールも簡単に入手できるので、まずは、仮名文字だけ替えるところから始めて、ぜひ遊んでみてほしいと思います。 (古川泰之)

【画面A】



【図7】ヒンティングがレンダリングに及ぼす影響

(a) ヒンティングなし　(b) TrueType内のhinting　(c) freetypeによるauto-hinting



PalmのJ-OSでは東雲フォントが変換され使われていると聞いています。こういったことは改変可能なフリーフォントでなければ、不可能です。

個人的に、フォントというのはOS以上に公共インフラに近い存在だと感じています。まあ、私がそれを実装するにふさわしいかどうか、という疑問は残りますが……。

●レンダリングエンジンの重要性

フォントに対する評価は、必ずしもデザインだけでは決まりません。第3、第4水準漢字サポートや、エンコードの種類(Unicodeなど)も重要な要素です。後述するhinting情報や、埋め込みビットマップといったファシリティも、フォントの重要な側面になってきています。このように、フォントがどんどん多機能化するにつれ、フォントレンダリングエンジンの持つ性能や機能も、ますます重要になってきています。

TrueTypeのレンダリングエンジンも、すべてが一様な実装がされているわけではありません。それぞれに微妙に差があります。一番分かりやすい例だと、輪郭を構成する部品同士がオーバーラップしたときのレンダリング結果がエンジンによって異なります。VFlib+ghostscriptや、Windows上の一部のPostscriptドライバでは、オーバーラップ箇所が白く抜けてしまいます(**図6**)。

さらには、hinting処理に関しては1つのレンダリングエンジンを使っても微妙に異なってきます。hintingは、線幅が一定であることや、直線であることなどの情報を含み、レンダリングエンジンが出力の品質を向上させるのに使う補助的な情報です。PC-UNIXでお馴染みのFreeTypeライブラリ([**14**])には、hintingに関して、3つの手段が選べます。

1. **hintingなし**
2. **TrueTypeフォント内にあるhinting**
3. **FreeTypeライブラリ内蔵のauto-hinting**

hitingは複雑なパラメータ群であり、フォントごとに独自のチューニングが施されているので、必ずしもここで提示する例が全てではないことを、最初に断っておきます。(正確には知らないのですが、hitingパラメータの複雑性はもはやプログラムと呼べるものらしいです)。その上で、東風明朝を一例にその差を見てみます。**図7**の**a～c**は東風明朝の「昏」の字を使って比較したものです。誌面でどれだけ再現されるか少し不安ですが、**図7-a**のhintingなしの例では、水平線、垂直線の輪郭が、他の2つに比べて、ぼやけていることが分かります。

東風フォントは、フォントの中に独自のhintingを持っています。これは、作成ツールであるTTeditが自動的に生成したものです。2と3の比較では、そういう意味では両方ともauto-hinting処理には違わないのですが、FreeTypeは、リアルタイムにhinting情報を出力するため処理速度を優先しているのでしょう、結果だけみると、上部の「氏」の斜め線が、水平線へ補正されてしまったり、どうしても分が悪くなってしまいます。

さらに、hinting処理実装に、特許に抵触する箇所があるため([**15**])、FreeTypeには性能に制約があります。FreeTypeの提供するauto-hinting機能はまさに、この特許を避けるための代替策です。この制約は技術的なものというより、法律的なものです。そのため、短期的な解決はなかなか難しいかもしれません。これは、非常に大きな問題となっています。

この制約は、FreeType 2.0から自主的に行っているようです。実装はなされていて、コンパイル時のオプション設定で、修正することができます。しかし、背後に法的な問題が絡んでいることは知っておいても損はないでしょう。

また、太線や細線の処理もレンダリングエンジンによって微妙に印象が変わってきます。こういった差は、普通の人はほとんど分からないと考えていたのですが、東風明朝がMS明朝に比べ、細いと言う人がいることを知ったときは、多少なりともショックを受けたことがあります。実際は、ほんの少し東風明朝の方が太いはずなのですが、小さいサイズでレンダリングすると、さまざまな条件や要因が重なって細く見える印象を受けるのでしょう。

このように、意図したデザインと実際のレンダリング出力から受ける印象が一致しないことは、しばしばあります。プロの方の中には、レンダリングエンジンとフォントデータをセットで1つの製品にするべきだとまで主張する人がいるくらい、レンダリングエンジンはフォントの品質に影響を与えます。

■課題

このフォントは、まだまだたくさん課題を抱えています。特に、大きなものとして下の2つが挙げられます。1つはフォント自体に対するもの、もう1つはフォントの作り方に関するものです。

1. JISX0213(第3、第4水準漢字)への適合
2. グループワーク用の開発フレームワークの確立

1に関しては、嬉しいことに、内田明氏が拡張Watanabeフォントシリーズ([**16**]、旧Kandataフォント)というプロジェクトを行っており、感心するほどしっかりしたモチベーションで確実に成果を上げつつあるところです。詳しくは**コラム「拡張Watanabeフォントについて」**を参照してください。また内田氏は第3、第4水準を中心にやっておられるので、将来コーディネイトできればなと考えています。

しかし、2に関しては、まだまだ課題が多いです。このフォントはただ日常の生活で「使う」ことだけを念頭において作成されているフォントであるため、「作る」ことに関して完全に無頓着になっている点が災いしています。

現状の最大の課題は、後でもっと詳しく書きますが、しっかりした開発フレームワークが確立していないことです。フォントそのものへの制作に手伝ってくれる人がいないということも、これが災いしているのだと考えています。

例えば、CVS(Concurrent Versions System)のように、バザール的に気軽にみんながデザインやその実装をコミットできる環境がありません。加えて、情報共有のためのデザインルールマニュアルも整っていません。

この他にも、根本的に私個人のデザインに関する能力が不足しているといった問題がありま

【図6】Ghostscript+VFlibによる、部品同士が重なった時のレンダリング結果

す。まだまだ、東風フォントのデザイン品質に関して、芳しくない評判も耳にするのも確かです。しかしそれは逆に言えば、私よりもずっと才能やセンスを持ち合わせた人間が世の中に溢れていることの裏返しだと心得ています。そういった人たちが、気軽にデザインに参加、コミットできる開発フレームワーク及び環境があれば、この問題は簡単に解決できるでしょう。何より、フォントは直接目に見える部分だけに、ずっと意見を集めやすく、同時に面白いものなのです。

拡張Watanabeフォントや、東風フォントに関しては、ユーザーは自由に改変する権利を持ち合わせています。読者のみなさんも、どうでしょうか?

■最後に

最後になりますが、東風フォントは今のところ実験プロジェクトです。完全な素人(あるいはコミュニティ)による制作でどこまで食い込めるのか、はたして完成できるのか。

私は、タイポグラフィの教養を持たないままフォント制作を開始しました。未知の分野への挑戦だったにもかかわらず、実践及び公開することで、外部とのインタラクションが発生して、最初のころより視野も広がり、確実により良い方向に進んでいると実感しています。

特に、拡張Watanabeフォントの内田氏や、/efont/の方々には、私の乏しい知識や眼力を補ってくださいました。改めて、この場を借りて感謝したいと思います。

東風明朝に関しては、やっと第1次水準漢字部分を終え、1つの段落を終えました。ビットマップフォントの制作から数えると、2年半。フォントに打ち込んで、やっと、ここまで来たという感があります。一方で、拡張Watanabeフォントも第3水準漢字の作業を完了し、さらに嬉しいことにフリーのビットマップフォントのバリエーションも確実に増えつつあるようです([17])。長い間、問題視されつつも、なかなか大きな前進を見込めなかったフリー日本語フォントの環境も、少しずつですが前進していると捉えて問題ないでしょう。

何ごとも、「できるかできないか」が問題なのではなく、「やるかやらないか」が重要なのだと思います。千里の道も一歩から。3歩進んでは2歩下がるというオマケ付きな上、まだまだ解決するべき課題も多く、先も長いですが、余分な気分の高揚は避けつつ、のんびりとやっていこうと思っています。

Column 拡張Watanabeフォントについて

JIS X 0213:2000「7ビット及び8ビットの2バイト情報交換用符号化拡張漢字集合」は、X0213と呼ばれたり、2000JISと呼ばれたり、新JISと呼ばれたりしています。この規格は、「普通の日本語の読み書きに必要であるのにJISX0208:1997では交換できないような文字」について、第3水準漢字や第4水準漢字として符号を与えています。

拡張Watanabeフォントプロジェクトは、このX0213フォントを作るため、渡邊フォントベースに第3、第4水準漢字の拡張実装を目指して制作を行っている最中です。

東風フォントが常用部分(JIS X208)を目標に、渡辺フォントのいわばリプレースのみを行っているのに対して、このフォントの特徴は、最新のX0213の規格にのっとった公有フォントのパイオニアとして存在している点にあります。

拡張Watanabeフォントは、東風フォントと同様、渡邊フォントをベースにしており、作業ツールも同じTTeditを用いて開発されており、両プロジェクトは非常に似ています。そのため、いろいろな情報をシェアしたり、刺激しあいながら活動しています。

■拡張Watanabeフォントの歴史

作成者の内田氏がそもそもフォント制作に携わった直接的なきっかけは、X0213を実装する明朝体アウトラインフォントのKandataフォントが原作者のwakaba氏の手を離れたことにあります。

Kandataフォントの原作者であるwakaba氏は、現在、jiskan16-2000の改刻・字種拡張に基づく「Habianフォント」の制作([18])に注力なさってます。

Kandataフォント最終版が、「改変/再配布等を自由に行ってよいフォント」として発表され、それならせめて当時判明していた規格不適合字形2文字を修正する作業くらい手伝おうと思い、実際に交換用字形データを作ったのが始まりです。

その後、他に「育ての親」のなり手が現れなかったこともあり、そのまま内田氏が交換用字形データを組み込んだ、新しいKandataフォントを2000年12月にリリースしました。

2001年10月、いよいよ内田氏完全オリジナルによる第3水準漢字の実装が完了し、これを機会に拡張Watanabeフォントに名前を改変して、現在に至っております。

このフォントは、当然LinuxなどPC-UNIXでも利用可能です。貴重かつフリーなX0213アウトラインフォントとして、ぜひ利用してみてください。

■フォントを作るということ

内田氏は非常にしっかりしたコンセプトを築いており、デザインなど私とは違ったところでいろいろ苦労されています。

例えば「涙の許容字体」([19])のように、規格票の不具合というよりは「日本国における漢字運用基準の不具合」などもいろいろ調査しながら、取り組んでいます(画面C)。

デザインについては数ある明朝書体(平成明朝、リョービナウ、創英プリリアントなど)を事前に調査した上で、初心者に配慮して、簡略化できるところは可能な限り簡略化されています。「直線で済ませてしまえるところは馬鹿馬鹿しいほど直線で済ませる。曲げるところも、「字母の再現」である必要がないのだから、2次スプライン曲線の計算処理に任せて簡単に曲げる。点は丸くする」いうおおまかな方針を打ちたて、さらに誰が作っても、よりバラツキが少なくなるように、デザインルールノートの作成に関しても思案に含まれています。

こういった思想、哲学は、「共通する部品の組み合わせ」を最大限に行かすような設計、開発にも生かされており、実に1年を経ずして第水準3漢字の完成を達成しました。

このようなさまざまな苦労は、拡張Watanabeフォントの Webページで、随時記録として残すように綴られており、私も内田氏からは非常に多くの情報を提供して役に立っております。ぜひご一読ください。

■面白いプロジェクト

内田氏は、この他にも、このプロジェクトの本筋あるいは横筋で、いくつか興味深いことも思索しています。せっかくの公有(誰のものでもなく、みんなのもの)フォントです。ぜひ拡張Watanabeフォント制作そのものや、こういったプロジェクトへ参加してみましょう。

●「歴史に名を刻め」プロジェクト

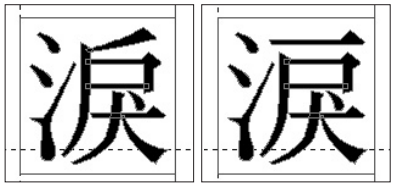
これは「拡張工作員の手引き」が出現しないことには始まらない幻のプロジェクト案です。第2水準漢字や第4水準漢字において、氏名や地名などで縁のある文字を改刻しようという方々を広く募ってみたいと空想しています。

●クワクチヤウWatanabe明朝

先に実装水準4に適合することよりも、もしかすると「こっちを先にやれ」という声が大きいのではないかとも思われるくらい、面白いプロジェクト案です。「クワクチヤウ」は、拡張の旧仮名ふりです。この名がイメージするように、常用漢字といわゆる旧字体とが区別されない面句点位置を、全て旧字体で実装しようというものです。

内田氏曰く「江戸川乱歩のテキストが公有化する頃に『新青年』に連載されていたオトナ向け短編を旧字体で眺められたら楽しいかなぁ」だそうで、こういう人は結構いるのでは……? (古川泰之)

【画面C】



東雲フォントを作ろう!

古川泰之
yasu@on.cs.keio.ac.jp

ここからは、ビットマップフォント内蔵のTrueTypeフォントである「東風フォント」を例にとり、実際にフォントをどのように制作しているのかを大まかに説明します。

この記事を読んでいる方のほとんどは、フォントを作ったこともないし、作り方も知らない人だと思います。ここで説明する内容は東風明朝フォントに特化した内容になっており、一例にすぎませんが、少しでも雰囲気を知っていただければ、幸いです。

/efont/([8])のリンク集は、作成ツールを探すのに適切なスタートポイントになると思います。少しでも自分で探ってみてると分かりますが、「日本語」のフォントを作成するには、まだオープンソースによるフォント制作環境が整っているとは言い難い状況です。これは、フリーフォントの製作促進の大きな障害の1つになっています。

■用意するもの

前述したように、私が現在使っているフォント作成ツール類には、残念ながらオープンソースなものは一つもありません。しかし、どれもリーズナブルなコストで入手することが可能です(**表3**)。

TTeditは、TrueTypeのアウトラインを編集するためのツールです。日本人によって作られているため、縦書きフォントなど、日本語フォントに特化した編集が可能です。大変丁寧によく作られており、セミプロの方が利用するのにも十分耐えられる品質です。Sbit32は、TrueTypeフォントへビットマップを埋め込むためのものです。コマンドラインのツールですがフリーで入手可能です。

TTeditをVectorのWebページ([20]などから、Sbit32は、Microsoft社のタイポグラフィグループ研究所のWebページ([10])から、それぞれダウンロードしてください。TTeditは、1カ月間なら無料で試用できます。

さらに用意するものは、時間的余裕と精神的余裕です。日本語フォントの制作は長期戦になります。寝る前、1時間を割り当てる感じで無理をしないのが長続きさせるコツです。フォントを作るのに特に資格は必要ありませんし、タイポグラフィに関する知識も必ずしも必要ではありません。

最後に必要なのは、モチベーションです。高さよりも持続性のほうが大切です。ただし、平成不況の長さに比べれば短くていいです……。

【表3】用意するもの

名称	解説
TTedit	安沢勝昭氏によるシェアウェア。3000円
Sbit32	Microsoft社タイポグラフィグループ研究所。フリー
時間的、精神的余裕	日本語フォント制作は長期戦になります
モチベーション	高さより、持続性の方が大切です

*1　TTedit附属オンラインマニュアル(安沢勝昭氏)参照

■作業全体のフロー

作業はすべてWindows上で行います。一部、WINE(WINdows Emulator)で代替することも可能です。おおまかな全体のタスクの流れは**図8**のようになります。このうち、最新のTTeditを用いることで、Sbit32は1度だけ実行すればよくなります(最新のTTeditは2001年11月19日現在、Ver.3.20です)。以前は、リリースの度に実行する必要がありました。

しかし、それでも埋め込みビットマップ情報を壊してしまうTrueTypeフォントツール(フリーソフトウェアでは特に)もいくつかあり、修復が難しいバイナリのフォントファイルの安全確保のため、普段の編集用のフォントファイルにビットマップフォントを埋め込むことは、避けた方が無難です。実際に泣きを見たことが2回ほどありますので……。

以下、TTeditによるアウトラインフォントの編集と、その次にSbit32によるビットマップフォントの埋め込み方法についての具体的な説明を続けます。

■アウトラインフォントの編集

まず、最初にTTeditと一緒に、東風明朝も事前にインストールしておきます。TTeditでできることは沢山ありますが、ここでは自分で一から作ろうとせずに、

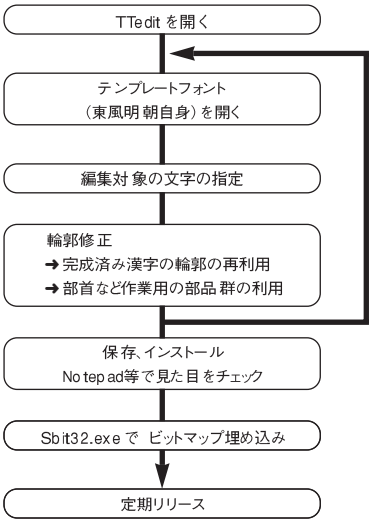
・テンプレートフォントに基づく編集

を中心に話を進めます。テンプレートフォントを用いることで、タイポグラフィの知識不足や、デザインセンスを補うことができ、ずっと作業を容易にすることが可能になります。東風フォントも、渡邊フォントをテンプレートフォントとして制作されています。

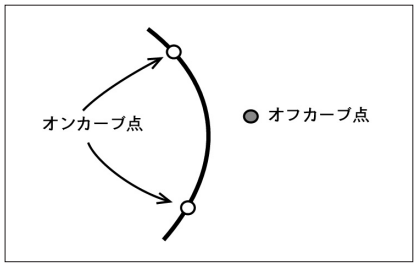
作業をするのに事前に知らなければならない、お約束が3つ程あります。

1つ目は、TrueTypeは曲線表現に2次Bスプラインを用います。制御点には**図9**のように、オンカーブ点とオフカーブ点の2種類あり*1、オフカーブは輪郭から外れた制御点で、曲線を表現するのに主要な役割を果たします。また、多くのドローツールが用いている3次ベジェとは異なるため、最初は曲線の制御が若干扱いにくいと思うかもしれません(ちょっとピーキーな印象を受けるはずです)。これは多少の慣れが必要です。

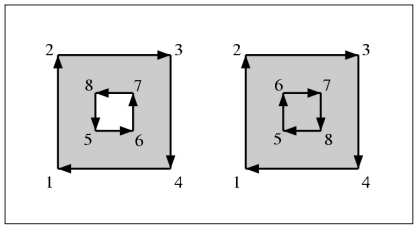
【図8】タスクフロー



【図9】曲線制御点の種類



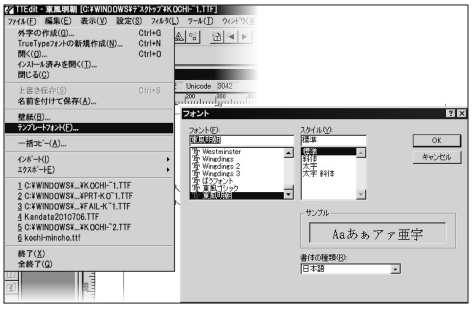
【図10】輪郭を描く向きによるレンダリングの変化



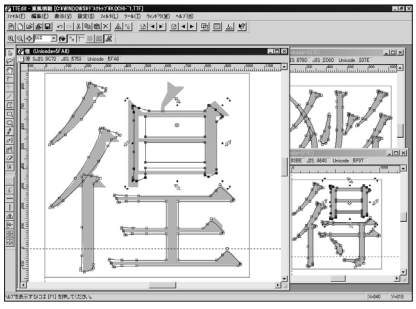
3次ベジェが4つの制御点から1つの曲線を作るのに対して、2次Bスプラインが1つ少ない3つの制御点でこれを行います。このため1制御点あたりの影響度が相対的に強く、これがピーキーな印象を与えます。

2つ目は、輪郭を描写するとき、右回り、左回りを意識しなければならないということです。原則ルールとして、制御点を描いた順番が大きくなる方向に進んで右側が黒く塗られます*1。このルールに従うと、外側の輪郭は時計回りに、内側の輪郭は反時計回りにする必要があります(**図10**)。これを違えると、意図したとおりに

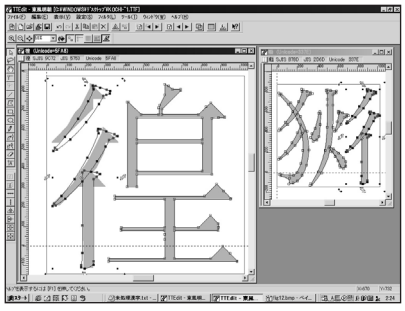
【画面1】テンプレートフォントの選択



【画面3】制作作業の画面

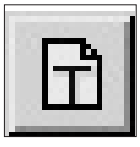


【画面4】作業用の部品を利用する

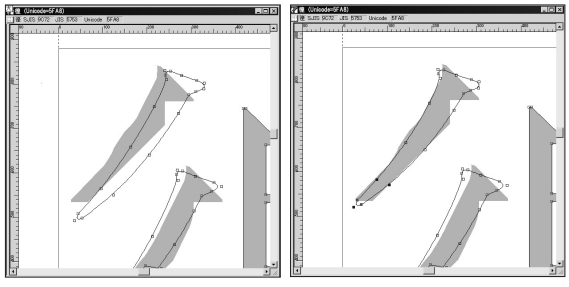


【画面2a】

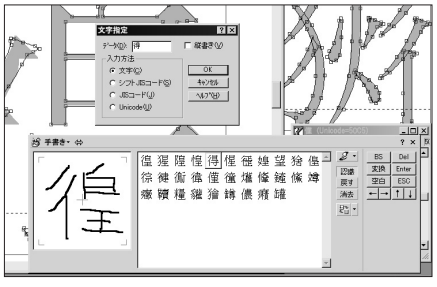
【画面2b】



【画面5】ペーストした部品を調整する様子



【画面6】似た輪郭を探すのに手書き認識を有効に使う



輪郭が塗りつぶされなかったり、ワープロなどのアプリケーションで指定した太字の指定がうまく再現されなくなります。

最後のものは非常に小さいことです。東風フォントはUnicodeベースのフォントですが、TTeditの機能により、JIS順列で編集するようになっています。

■作業の実際

では、TTeditを起動して、kochi-mincho.ttfファイルを読み込みます。その後、ファイルメニューにあるテンプレートフォントを選び、事前にインストールした東風明朝自身を選びます(**画面1**)。これは、東風明朝がすでに渡邊フォントを含んでいるのが大きな理由ですが、さらには、自分自身のフォントをテンプレートとすることで、変更分を認知しやすくなるという利点があり、お勧めです。

また、このテンプレートフォントの表示については、ツールバーにある水色で表示された「A」アイコンによって、ONとOFFを切り替えられます(**画面2a**)。

では、これから文字の編集作業に移ります。ここでは、すでに完成している第1水準漢字の部品をベースにして、第2水準の漢字を仕上げていきます。実例として、「彷徨(さまよ)える」の「徨」(JISコード5fa8)を取り上げます。

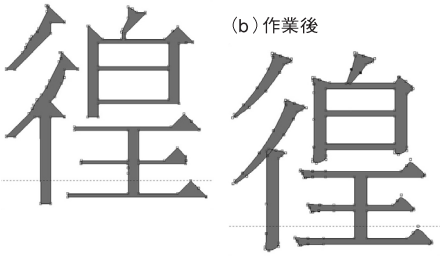
ツールバーに「T」(**画面2b**)で示されたアイコンを押し、「徨」を入力し、その輪郭データを開きます。

この後、輪郭を1つ1つ人手によって調整していきます。**画面3**は、実際の制作途中の画面です。水色(誌面では薄いグレーでしょうか)で表示される渡邊フォントの字形をなぞるように古い輪郭を捨てて、ディテールを調整した新しい輪郭に置き換えていきます。

【図11】部品置き換え作業前後

(a)作業前

(b)作業後

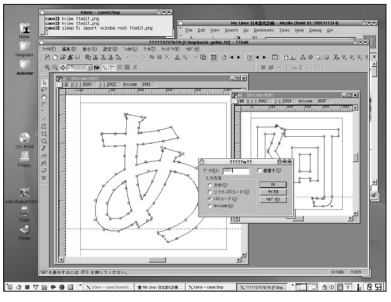


ただ、すべてを手作業で行ったのでは大変です。このため、東風明朝の場合、少しでも作業の効率化を測るために、特に部首などの頻繁に使う部品を一時的に作業用に残しています(**画面4右側**)。第3水準の記号のあたり(JISコード2d79辺り)にいろいろ詰まっていますが、将来、完成したら削除する予定です(これは、TTeditで「亜」を表示して、ひたすらコードをdecrementしていったものです。気まぐれです。深い意図はありません)。

例えば、「徨」の字の場合、「彳」をこの部品群から使うことができます。これをカット&ペーストし、拡大しながら多少の微調整を施せば(**画面5**)、偏に関して簡単に作業が完了します。

それ以外の輪郭に関しては、すでに完成している第一水準の漢字から部分的に流用するのが効率的です。たいていここで、似た部品を含む漢字を探すのに時間を取られるのですが、Windowsの標準日本語入力ツールMS-IMEに含まれる、「手書き」認識機能を使うと効率的に探せます(**画面6**)。

【画面7】WINEを使ったTTeditの画面



また内田明氏の拡張Watanabeフォントの場合、文字を構成する各部品を結合せずにそのまま残していますので、これと同じ要領で似た字を探せば、東風明朝よりも、もっと効率よく作業ができます。

この置き換え作業を続けてゆくと、**図11**のように仕上ります。トメ、ハネといった細部だけでなく、曲線の輪郭も滑らかになっているのがすぐに分かるでしょう。

最後のステップとして、オーバーラップした輪郭を結合すれば一文字の作業が完了します。ここまでで大体、7分〜15分くらいかかります。基本的に、この繰り返しになります(内田明氏の拡張Watanabeフォントの場合は、結合しないでそのまま残すという開発方針になっています)。

おまけとして、蛇足ながら、なんとかLinux上で作業できないかと、WINE上でTTeditを起動してみたのが**画面7**です。VinePlus 2.1のWineを使うと、そこそこに動きました。素晴らし

い！　一部日本語メッセージが出なかったり、細かい不具合がありますが、だましだまし使えば、結構作業できそうな気がします……。あとは実績を作るだけでしょうか。

さらに詳しい情報を、私のWebページ「東風明朝の制作課程」（[21]）に（未完成ながら）まとめています。よろしければ参考にしてみてください。

■ビットマップフォントの埋め込み作業

次に、TrueTypeフォントへのビットマップフォントの埋め込み作業を説明します。TrueTypeフォントの制作に絶対必要な作業ではありません。加えて、ここでの説明は相当量のエディタ編集や、コマンドライン操作が入るため、申し訳ありませんが、スクリプトプログラム経験がある人向けの内容になります。

最初に紹介したSbit32では、以下の処理が可能です。

- フォントサイズごとのBDFフォントをTrueTypeフォントへ埋め込む
- 埋め込みフォントからBDFフォントへのエキスポート
- 埋め込みフォントの削除
- 埋め込みフォント状況の確認

ご覧のように、Xでなじみの深い、BDFフォントを入出力ファイルとします。さらにこのツールは、WINEでも動かすことが可能[*2]ですので、PC-UNIXユーザーでも使いこなすことができます。

しかし、XのBDFファイルはそのまま使えず、以下のような変更が必要になります。

1　改行コードを「\n」から「\r\n」へ変更
2　SIZE行にあるレゾリューション値を「72」へ変更
3　すべてのSTARTCHAR行をSTARTCHAR　.XXXXに変更。.XXXXはUnicode（UTF16）
4　スペース文字のデータを削除（した方がよいらしい）

変更項目はこれだけですが、日本語フォントの場合、量が多いためフィルタプログラムを書かなければ、不可能でしょう。私の場合は、恥ずかしくてここでは公開できないような非汎用的かつ不完全なスクリプトプログラムを書いてしまって対処しています……。

ここまでの変更を行うと、BDFファイルは、**リスト1**のようになります。このうち、「PROPERTIES（STARTPROPERTIES～ENDPROPERTIES），SWIDTH，ENCODING」行はSbit32では基本的に無視されます。

TrueTypeフォントへビットマップフォントを埋め込むには、入力用に、TTFファイルと、BDFファイルの他に、さらにMET（Metrics）ファイルを用意する必要があります（[21]）。METファイルは、非常に沢山のオプションがあり、うまく使えば1つのBDFフォントを複数のサイズ用埋め込みフォントとして使うことも可能になります。ここでは、最低必要なビットマップフォントのサイズを示す、PPEM値及び、オリジナルのBDFファイルから省略されたASCENDER値や、DESCENDER値を記します（**リスト2**）。

用意できたら、Sbit32を「-a」オプションを付けて、**実行例1**のように起動します。

これで、12ドットビットマップフォントが埋め込まれたout.ttfが生成されます。Linux上のWINEから起動することもでき、その場合、**実行例2**のようにタイプしてください。きちんとビットマップフォントが埋め込まれたかを確認するには、**実行例3**のように「-i」オプションを使って出力される内容から確認できます（**実行例4**）。

反対に削除するには、「-d」オプションを使い、**実行例5**のようなコマンドを実行します。

最後に、いちいち書かなくても理解されていると思いますが、まあしきたりとして……。商業フォントは一般的に、ここで紹介したようなフォントファイルの改造や分解は、ライセンスによって制限されています。それを遵守した上で、臆せず、いろいろ遊んでみて下さい。またここまでの内容は、Red Hat Linux 7.2で確認しています。

【リスト1】Sbit32用BDFフォントの例（12ドット東雲フォント）

```
STARTFONT 2.1
FONT Shinonome12
SIZE 12 72 72
FONTBOUNDINGBOX 12 12 0 -2
STARTPROPERTIES 1
COPYRIGHT "Public Domain"
ENDPROPERTIES
CHARS 6944
ENDCHAR
STARTCHAR .0021
ENCODING 33
SWIDTH 960 0
DWIDTH 6 0
BBX 6 12 0 -2
BITMAP
00
20
20
20
20
20
20
20
00
20
00
00
ENDCHAR
 ⋮
```

【リスト2】METファイルの例（input.met、12ドット東雲フォント用）

```
PPEM 12
ASCENDER 10
DESCENDER 2
```

【実行例1】

```
C:\TEMP\> Sbit32.exe -a input.ttf input.bdf
input.met out.ttf
```

【実行例2】

```
$ wine -- Sbit32.exe -a input.ttf input.bdf
input.met out.ttf
```

【実行例3】

```
C:\TEMP\> Sbit32.exe -i out.ttf
```

【実行例4】

```
EBLC information:
 version=2.000000
 glyphs: 0xXXXX - 0xYYYY  <--ここと
   ppem: 12 x 12          <--ここ
```

【実行例5】

```
C:\TEMP\> Sbit32.exe -d input.ttf input.met out.ttf
```

■まとめと課題

以上、簡単な説明でしたが、これらの操作で、TrueTypeフォントの輪郭を編集したり、それにビットマップフォントを埋め込んだり、外したりできます。ただ、これは現状でできる手段としては妥当な方法であると考えていますが、必ずしもベストな解ではないということを覚えておいてください。

特に、まだまだ大きな課題が残されていることを挙げておきます。用意するツールの話で説明したような、オープンソースによるツールを使っていない点もその1つですが、ここで紹介した方法は、ネットワーク上のコミュニティが共に開発するという手段としては、決して最適とはいえません。

例えば、ソースコードのグループ開発ではよく使われるCVS（Concurrent Versions System）を活用したいところですが、TrueTypeフォーマットという数メガバイトサイズのバイナリファイルを中心に扱っているため無理があります。このため、気軽にネットワークコミュニティが、わいわいとデザインやその実装をcommitできる環境を提供できないでいます。

とりあえず、1文字、1ファイルのテキストに落せれば、安心してCVSリポジトリを活用できるので、現在はこの方向で模索している最中です。1文字1ファイルなら、efont-serif（[22]）を持ってきて、英字部分をプロポーショナル化したりできます。また拡張Watanabe明朝シリーズ（[16]）とのコーディネイトなども、簡単なファイル操作だけで実現できるという利点もあります。

*2　Red Hat Linux 7.2で確認

そのためのオプションとして、今のところ、

- TTeditの出力するCSV（テキスト）形式を用いて、VBScriptで処理（簡単そうですが、COMベースでGUIをバッチ的に処理するので遅そう……）。
- Microsoft社のTTFdump、TTF Open Assembler（[10]）を補助的に使う。これだと、うまくいけばビットマップフォント部分も包括して管理できる？（WINEでも動くので、Linuxからも管理できる）。

の2つを考えています。このユーザーインターフェイスとしては TTeditのマクロとしてできれば理想ですが、当面はコマンドラインから行う「make commit」や、「make update」などのCUIで十分だと考えています。しかし、まだまだブレインストーミングの最中であり、実際に手を動かして実装したり、検証してる訳ではありません。このあたりで手を貸して（というより委譲かも……）いただける方がおられましたら、ぜひ協力をお願いします。

Column 東風フォントの見分けかた

ここでは私が使っているフォントの見分けかたを紹介しましょう。ほとんどの場合Turbolinuxのリョービ製フォントと、東風フォントを見分けることが多いと思いますので、その2つをどこで見分けるか、ちょっとしたコツを書いておきます。

東風フォントは、私の趣味によって「つ」、あるいは、「る」などの丸い部分の最大高さが個人的な趣味により全体的に小さめになっています。

最近KDEのUIとして用いられるゴシック系の見分け方に関しては、カタカナの「ク」で見分ける方が早いでしょう。東風ゴシックでは明朝系のデザインをそのまま真似て、最後の画を左へ大きく伸ばしています。

ただし、東風フォントは制作途中であり、特に仮名文字は今でも調整し続けているので、今後はどうなるかは定かではありません……。

（古川泰之）

Resource

[1] 電子書体オープンラボ「/efont/」の「東雲ビットマップフォントファミリー」

http://openlab.ring.gr.jp/efont/shinonome/

[2] My Linux日本語化計画の「東風フォント」（古川泰之）

http://www.on.cs.keio.ac.jp/~yasu/jp_fonts.html

[3] フリーウェア大賞での橘浩志氏のコメント

http://www.nmda.or.jp/enc/fsp/jis/commt95.html

[4] SNAGAO.WEBの「NAGA10フォント」（永尾制一氏）

http://hp.vector.co.jp/authors/VA013391/

[5] mkbold（永尾制一氏）

http://hp.vector.co.jp/authors/VA013391/tools/

[6] The X-TrueType Server Project

http://x-tt.dsl.gr.jp/index-ja.html
残念ながら、現在はメンテナンスが停止しています。

[7] 狩野宏樹氏のフォント関係リンク集

http://kappa.allnet.ne.jp/kanou/bookmarks.html
非常に充実しているリンクです。必見。

[8] 電子書体オープンラボ「/efont/」

http://openlab.ring.gr.jp/efont/

[9] 武蔵システム（安沢勝昭氏）

http://www.interq.or.jp/www1/anzawa/
外字及びTrueTypeフォントエディタ「TT Edit」（シェアウェア）と、OpenTypeフォントエディタ「OT Edit」（シェアウェア）の配布を行っています

[10] Microsoft Typography（Microsoft社タイポグラフィグループ研究所）

http://www.microsoft.com/typography/

[11] かなの大きさ（内山孝憲氏）

http://macptex.appi.keio.ac.jp/~uchiyama/kana.html

[12] 特殊非営利活動法人（NPO）日本タイポグラフィ協会の「タイポグラフィの知的所有権について」

http://www.typo.or.jp/info/morals/moral1.html

[13] XmBDFEd

http://crl.nmsu.edu/~mleisher/xmbdfed.html

[14] The FreeType Project

http://freetype.sourceforge.net/

[15] FreeType and Patents

http://freetype.sourceforge.net/patents.html

[16] 拡張Watanabeフォントシリーズについて（内田明氏）

http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/extended-watanabe-mincho/

[17] X11で使できるビットマップフォント一覧（狩野宏樹氏）

http://kappa.allnet.ne.jp/kanou/fonts/x11bdfs.html

[18] Habianフォント（wakaba氏）

http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/AlternativeViews/tears/

[19]「涙の許容字体」（内田明氏）

http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/AlternativeViews/tears/

[20] VectorのTTeditダウンロードサイト

http://www.vector.co.jp/soft/win95/writing/se033302.html

[21] 東風明朝の制作課程（古川泰之）

http://www.on.cs.keio.ac.jp/~yasu/jp_fonts_ttedit

[22] /efont/のefont-serif英字フォント

http://openlab.ring.gr.jp/efont/serif/