プログラム肉付けによる複数漢字書体間のスケルトンデータの共有

田中哲朗[†] 石井裕一郎[†]
竹内幹雄[†,*] 和田英一[†,**]

計算機による文書作成の一般化に伴い種々のサイズ，書体のフォントが必要とされるようになってきている．JIS X 0208, X 0212 に含まれる文字だけでも1万文字以上になる漢字の多様なフォントを安価に提供するためには，高品質のフォントを人間の手をなるべくかけずに製作することが望まれる．フォントを文字サイズの変更が可能なアウトライン形式で表現すると，サイズごとにデザインする必要はないが，太ゴシックと細ゴシックのように同一のフォントファミリーに属する書体も別々にデザインする必要がある．最近，この点を改良し文字サイズ以外のパラメータの変化にも対応するスケルトンフォントの研究が行われているが，書体の変更へ対応するのが難しかった．本論文ではスケルトンフォントの肉付けのうち書体に依存しない部分を切り分け，書体を記述する少量のプログラムを変更するだけで書体を変更できるシステムを提案し，実際にシステムを作成した．作成したシステムによって，明朝体とゴシック体のように見かけの異なる書体間でも大部分のデータを共有でき，デザインの手間を大幅に減らすことができることを確認した．

Sharing Skeleton Data by Multiple Kanji Fonts through Programmable Rendering

Tetsurou Tanaka,[†] Yuichiro Ishii,[†]
Mikio Takeuchi[†,*] and Eiiti Wada[†,**]

Various typeface and size of fonts have become necessary with the wider use of computers in document creation. A set of Japanese fonts includes thousands of kanji characters, whose creation and maintenance cost is very high, and needs enormous resources. Automated creation of fonts by means of outline format has been proposed to solve this problem, but every typeface should be designed independently even if the faces are similar, say "Fats" Gothic and "Thin" Gothic. In this paper, we propose a skeleton font system for kanji which can generate various kinds of typefaces automatically only with a slight modification to the programs describing the features of the typeface. Our main idea is the generalization of the algorithms to flesh out the skeletons as the style-independent parts of the system. By the proposed system, most of the skeleton data is shared between different faces such as "Mincho" and "Gothic." The system has only one set of skeleton data which is maintained with minimal resources, and the design cost can be much reduced.

1. はじめに

ワードプロセッサの普及によって，計算機を文書作成の道具として使用することが一般的になってきた．それに加え高品質の出力装置が安価に手に入るようになり，多様なフォントに対する要求が広範囲に生じてきている．商業出版のような品質でなくてもよいから，内部的な印刷物やディスプレイ画面への出力用に中程度の品質で多様な書体を用いることが求められている．

計算機で扱うフォントとして一般に用いられるドットイメージフォントは，品質を落とさずに文字サイズを変更するのが困難である．多様なサイズで高品質に出力するためには，文字サイズと解像度に応じたフォントを用意する必要があるが，これは製作コストと記憶容量が問題になる．

近年，文字サイズの変更が可能なアウトラインフォントが普及してきた．アウトラインフォントは，出力の際に塗りつぶし計算を必要とするが，文字サイズに

† 東京大学工学部
Faculty of Engineering, University of Tokyo
* 現在，日本 IBM 基礎研究所
Presently with IBM Research, Tokyo Research Laboratory
** 現在，富士通研究所
Presently with Fujitsu Laboratories

応じてデザインする必要がなく，高解像度用のドットイメージフォントほどは記憶容量を必要としない．

しかし，アウトラインフォントでは，太明朝と細明朝のように同一のフォントファミリーに属していても輪郭の異なる書体間でデータを共用できず，別々にデザインする必要がある．太さが違う二種類のフォントに対して内外挿を適用して，任意の太さのフォントを得る方法も提案されているが[1]，適用できるのはゴシック体のような太さが均一な書体に限られている．

一方，フォントをストロークの集合からなる骨格（スケルトン）で定義する手法もある．このようなフォントはスケルトンフォントと呼ばれる．ストロークは，字を線で表現した時の一画か，それを更に細分したものにあたる．出力の際には，太さや書体の種類に応じて肉付けをアウトライン形式やドットイメージに変換する．

1つのフォントセットに数千文字が含まれる漢字フォントではデザインコストの軽減が重要である．太さ等の形状をパラメータの調整によって変更できるスケルトンフォントは，同一のフォントファミリーに属するフォントのデザインを共有できるので，漢字フォントの表現に有効であると考えられ，各所で研究が進められている．

よく用いられる明朝体とゴシック体の2書体のうみ，漢字スケルトンフォントの研究はゴシック体に関する研究が先行している[2,3]．ゴシック体はストロークの太さ変化が小さく，ストロークの接合部分の形も単純で，簡単な肉付けアルゴリズムで実現できるためである．一方，明朝体に関する研究も文献4）および文献5）で報告されている．

これらの研究では，ストロークを肉付けの種類に応じて分類する方式（エレメント方式）を用いているが，エレメントの肉付けは書体の特徴を表現できる範囲でなるべく少数のパラメータで表現する方法をとっている．この方法を複数書体を扱うシステムに拡張するのは困難である．明朝体とゴシック体の肉付けをパラメータの変更で実現するためには多数のパラメータを持つ必要があるが，パラメータ数の増加はデザインコストの増加につながるし，扱う書体を増やす際に既存のパラメータ調節で実現できない場合はシステムを書き直す必要が生ずる．

本研究では肉付けの自由度の小さいパラメータ方式は用いず，肉付けの枠組だけをシステムで用意しておいて，各エレメントの具体的な肉付けをデザイナが書体ごとにプログラムで指定する方法を提案する．既存の書体を基にデザインする時は，差分だけをプログラムすれば良いので，それほど手間がかからない．画数が異なる，エレメントの種類が異なるなどの理由で異なる書体間ではスケルトンデータを共用できないこともあるが，大部分の漢字のスケルトンデータが複数書体で共有できるので，デザインコストを軽減することができる．

この方針にもとづいて，スケルトンデータを基に各書体ごとの肉付けをおこないアウトラインフォントを作成するシステムを UtiLisp/C[6),7)]を用いて製作し，JIS X 0208, X 0212 に含まれる漢字 12156 文字について，明朝体の試作をおこなった．また，同じスケルトンデータについて，角ゴシック体，丸ゴシック体の肉付けをおこない，大部分の文字がスケルトンデータを共有できることを確かめた．

以下，2章では，本システムにおける肉付けの基本方針を示す．3章では肉付けの手順を詳しく説明する．4章では明朝体フォントを実現することによって，本研究における肉付けの自由度の大きさをどう生かすことができるかを示す．5章では記憶容量，書体の変更に伴う作業量等の評価をおこなう．6章でまとめと今後の課題について述べる．

2. 基本方針

英字フォントの Courier のような単純な書体は，骨格を曲線で定義して均一な幅に肉付けする単純な方法で実現できる．しかし，漢字書体の多くは複雑な肉付けを必要とするので，ストロークを何種類かの“エレメント”に分類しストロークの種類に応じて肉付けをする方法（エレメント方式）がよく試みられている．

エレメントは**図1**の太い実線のように，少数の制御点によって定義される．エレメント間の接続によって

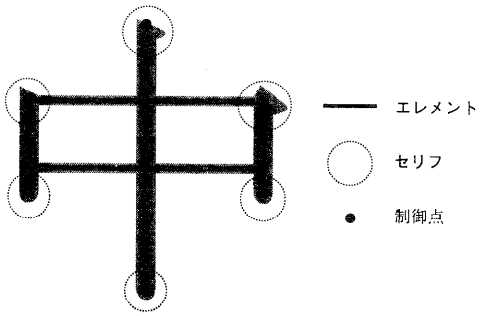


図1 エレメントとセリフ
Fig. 1 Elements and serifs.

図1中の円内のようにヒゲ状の飾りが生ずることがある．このようなヒゲ状の飾りを一般にセリフというが，本論文ではエレメントの接続や始点終点付近の飾りすべてをヒゲ状であるかどうかにかかわらずセリフと呼ぶことにする．

エレメント方式では，エレメントの種類を増やすほど肉付けの多様性を実現できるが，デザインコストと折り合いをつけて，どの程度に種類を抑えるかが重要である．

よく用いられる明朝体とゴシック体の2つの漢字書体のそれぞれについてスケルトンフォントを試作した研究はいくつかあり，一応の水準に達する肉付けアルゴリズムが考案されている．これらを大まかに分類すると以下の2種類になる．

1. 見本アウトラインの変形
   あらかじめデザインした見本アウトラインに座標変換をほどこす方法．文献5)では明朝体でテストしている．エレメントの長さや，制御点間の角度，太さなどのパラメータがある程度以上変化すると対応できないので，エレメントの種類を増やさなくてはいけない．
2. 制御点付近の相対位置指定
   アウトラインの特徴的な点をストロークの制御点に対する相対位置で定義する方法，エレメントの種類によって相対位置を決定するためにいくつかのパラメータを用いる．太さ変化の少ない書体に向いており，文献2)ではゴシック体に適用している．

いずれの方法も一種類の書体に関しては有効だが，デザインの自由度が小さく複数書体への対応が難しい．本研究ではエレメント，セリフの種類に応じた肉付けの変化をパラメータで指定する代わりにプログラムで指定することによって，デザインの自由度を増すことを試みた．指定するプログラムはプログラミングとデバッグの手間を軽減するため，実験システムと同様に対話型言語である Lisp で記述する．

肉付けをプログラムで指定する方法は文献8)でも用いられているが，本研究では書体に依存しない肉付け部分を切り離し，書体の変更に伴うプログラムの変更を最小限にとどめている．

3. スケルトンデータの表現と肉付けの手順

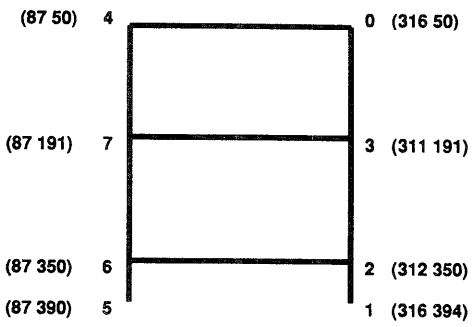
3.1 スケルトンデータの表現

スケルトンで表現されたフォントから直接ドットイメージを生成するか，それとも一旦アウトラインに変換するか2種類の方法が考えられる．前者は文献9)や文献10)などで用いられた方法で，肉付けアルゴリズムの自由度が大きく，解像度に応じた肉付けが可能になる．しかし，本研究では生成したフォントを少ないファイル容量で表現できることを重視して，後者の方法をとった．

漢字の肉付けは，構成するエレメントの種類と座標情報だけでなく，エレメント間の接続の影響も受ける．本研究のシステムは直接テキストエディタで編集することも考慮して**図2**のような形式の Lisp のS式で定義する．S式の各部分はそれぞれ以下のような意味を持つ．

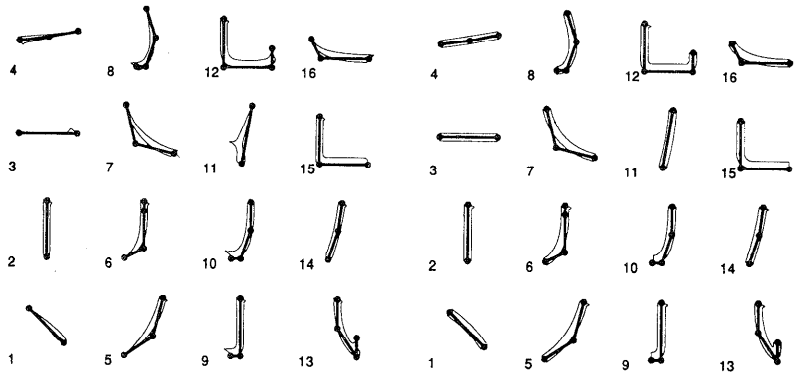
（1） 制御点の座標
スケルトンデータ中のすべての制御点のX，Y座標の対をリストとしたもの．デザインする際に X-window 上のエディタを使う関係で，X，Y座標の範囲は0から399まで，座標原点は画面の左上とする．

（2） 各エレメントの種類と制御点との対応
エレメントの種類は，tate, yoko などの Lisp




```
(defprimitive nil 日
  '(((316 50)(316 394)(312 350)(311 191) ; (1)
     (87 50)(87 390)(87 350)(87 191))
    ((tate (0 1) ; (2)
           (link 2 3)) ; (3)
     (tate (4 5) (link 6 7))
     (yoko (6 2))
     (yoko (7 3))
     (yoko (4 0)))
))
```

図 2 スケルトンデータの表現
Fig. 2 Representation of skeleton data.



1. ten, 2. tate, 3. yoko, 4. migiue, 5. hidari, 6. tatehidari, 7. migi, 8. kozato, 9. tatehane, 10. tsukurihane, 11. sanzui, 12. kokoro, 13. tasuki, 14. magaritate, 15. kagi, 16. shin-nyuu

図 3 明朝体，ゴシック体のエレメント
Fig. 3 Elements of "Mincho" and "Gothic" type faces.

シンボルで表す．明朝体とゴシック体に用いたエレメントは**図3**の 16 種類である．エレメントは種類に応じていくつかの制御点を持つが，制御点の指定は(1)の座標対リストの要素番号でおこなう．

(3) 各エレメントの属性リスト
エレメントの肉付けの際にプログラムに渡される属性リスト．ただし，エレメントの途中に接続する点を記述する link という属性は，エレメントの肉付けの際には無視され，セリフの決定のみに用いられる．

文献2)では接続エレメントの始点あるいは終点を他のエレメントの始点，終点と共有する JOINT 接続と，始点あるいは終点が他のエレメントの中途に接続する LINK 接続に分け，始点か終点か，左右どちらかなどによってさらに細かく指定している．ここで採用した表現方式も文献 2) とほぼ同等の接続情報を表現することを以下に示す．

- 他のエレメントと関連を持たない場合
  図2の点1，5がこれにあたる．
- JOINT 接続
  点0のように，複数のエレメントの制御点となっている場合．
- LINK 接続
  点2，3，6，7のように他のエレメントの属性リストの link 情報に含まれている場合．

接続が左右どちらかという情報は接続エレメントの種類によって特定できる．

3.2 肉付けの手順

接続の種類に応じて肉付けを変えるために，これまで以下のような方法が提案されてきた．

1. エレメント変形によるセリフ定義
   文献2)と文献4)では接続によって生ずるセリフを，接続がない場合の始点や終点のセリフの変形として表現している．
2. セリフを含んだエレメント定義
   文献5)ではセリフを含んだエレメントを定義している．始点と終点がそれぞれ別のエレメントに接続するエレメントの始点，終点付近のセリフをこの方法で表現すると多種類のエレメントを用意する必要があるので，1の方法も併用している．

1も2もセリフの生成をエレメントの属性として表現しているが，異なるエレメントの組合せでも似たセリフを生ずる場合が多く，これを2のように別のエレメントで表すのはエレメントの数を増やすことになるし，1のようにエレメントを変形させるのは，接続するエレメントの種類によってより滑らかに接続するように修正する必要があり，デザインコストの増加につながる．

本研究ではエレメントの肉付けの定義とは独立にセ

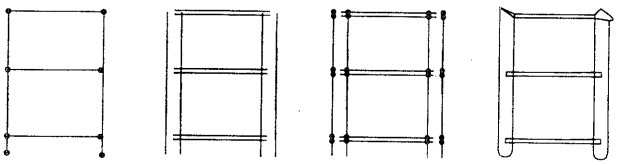
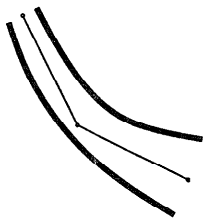


図 4　肉付けの手順
Fig. 4　Process of rendering skeleton data.



```
(defelement mincho migi
  (lets ((w0 (times mw 0.5))
         (w1 (times mw 0.7))
         (w2 (times mw 1.0))
         (w3 (times mw 1.2))
         (p0 (first points))
         (p1 (second points))
         (p2 (third points)))
    (curve3 p0 p1 p2 0.3 0.3
            w0 w1 w2 w3)))
```

図 5　エレメント肉付け
Fig. 5　Rendering element.

リフの肉付けをプログラムで指定し，エレメントの種類，始点終点などの情報によって肉付けプログラムを呼び出す方法を採用した．エレメントの肉付けを変更した際に両方の肉付け結果が不連続にならないように，以下の手順で肉付けをおこなう（**図4**）．

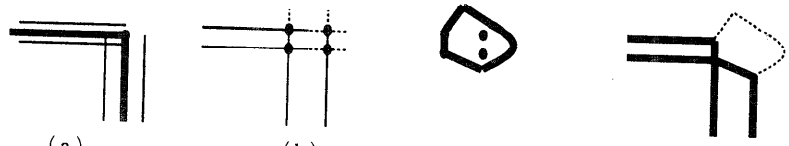
1. エレメント肉付け
   接続関係を無視して各エレメントに肉付けをする．肉付けの結果としては，各エレメントに対して，二本ずつのアウトラインが得られる．**図5**は明朝体の右払いの肉付けプログラムと肉付けの結果得られるアウトラインを示す．プログラム中の curve 3 は制御点といくつかの部分での肉付けの太さを引数として2本のアウトラインを返すシステムで用意した関数である．
2. JOINT 接続のセリフ肉付け
   エレメントの始点，終点同士が接続する時は，接続するエレメントの種類，始点，終点の区別に応じて，対応するセリフの肉付けプログラムを呼びだす．セリフ肉付けプログラムはエレメントのアウトライン同士の4つの交点を引数として呼び出される．
   セリフの肉付け後，エレメントのアウトラインの始点，終点はセリフ部分のアウトラインデータの始点，終点と一致するように変更される．**図6**が横棒の終点と縦棒および左払いの始点の接続のためのセリフプログラム（実際のものよりも簡略化している）とセリフ肉付けの様子を示している．
   セリフが定義されていない時は，アウトラインの交点にアウトラインの始点，終画点を変更する．注意が必要なのは，「しんにゅう」のように3つのエレメントが一点で接続する場合である．この場合はエレメントの始点，終点を変更するだけでは中央に空白が生じてしまうので特別に扱う（**図7**）．4つ以上のエレメントの接続は JIS X 0208, X 0212 中の漢字の定義では現れなかったので扱っていない．
3. 中途接続
   始点，終点が他エレメントの中途から接続しているときは，アウトラインの始点終点を，他エレメントのアウトラインの内部に入るように修正する（**図8**）．これは，アウトライン同上の4つの交点を求めて，その中点をとることによっておこなう．
4. 孤立した始点，終点付近のセリフ
   接続のない始点，終点付近のセリフも JOINT 接続によるセリフと同じ指定法を用いる．座標データも同じく4点を与える必要があるが，これは左右アウトラインの端点と，端点からアウトラインに沿っておよそ端点間の距離だけ離れた点とする（**図9**）．

4. 書体の定義

本研究のシステムでは書体が**図10**のような階層構造を持つようにした．書体を定義するには，スケルトンデータとエレメントの肉付けプログラムとセリフの肉付けプログラムが必要だが，たとえば，

(subfont maru-gothic gothic)

のように宣言して，角ゴシック体を基に丸ゴシック体定を義すると，スケルトンデータ・エレメントの肉付けプログラムとセリフの肉付けプログラムの大部分が



(a) 各エレメントへの肉付け　(b) 4個のアウトライン接続　(c) 各アウトラインの接続に対する肉付け　(d) エレメントのアウトラインの始・終点の変更

```
(defkazari mincho (tate hidari) 1 yoko 3)
 (lets ((p1 (vref cross 1))
        (p2 (vref cross 2))
        (p3 (vref cross 3))
        (d1 (diff2 p1 p3))
        (d0 (rot90 d1))
        (w0 (times mw 1.333 tatekazari))
        (w1 (times mw kazariheight))
        (p6 (plus2 p1 (normlen2 (times w1 -0.7) d1)
                   (normlen2 (times -1.0 w1) d0)))
        (p8 (cross2 p6 p2
                    (plus2 (normlen2 1.0 d0)(normlen2 -1.3 d1))
                    (diff2 p3 p2)))
       (p4 (plus2 p8 (normlen2 (times 1.0 w0) (diff2 p2 p3))))
       (p5 (cross2 p8 p4 (diff2 p8 p6)
                    (rot (diff2 p8 p6) (degree 50)))))
  `((angle .,p4)
    (bezier .,(inter2 p4 p5 0.9))
    (bezier .,(inter2 p8 p5 0.9))
    (angle .,p8)
    (angle .,p6)
    (angle .,p1))))
```

図 6 接続のセリフ肉付け
Fig. 6 Rendering joint point.

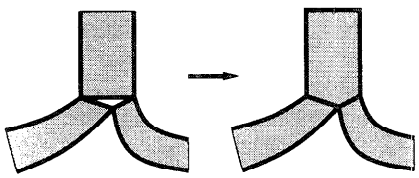


図 7 3つのエレメントの接続
Fig. 7 Joint of three elements.

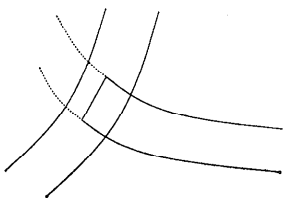


図 8 中途接続
Fig. 8 Joint at the middle of element.

共有できるので，一部を定義し直すだけでよい．これをオブジェクト指向言語に当てはめると，書体がクラスに当たり，肉付けプログラムの共有はメソッドの継承と考えることができる．

書体間のスケルトンデータの相違を吸収したり，肉付けに関するヒントを自動生成するために，書体ごとのフックプログラムを定義する機能も用意されている．このフックプログラムは肉付け前のスケルトンデータを引数として起動される．

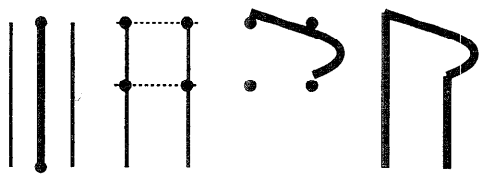


図 9 孤立点のセリフ
Fig. 9 Serif of isolated point.

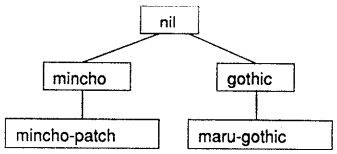


図 10 書体の階層構造
Fig. 10 Layer of type faces.

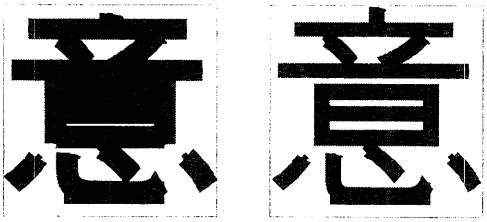


図 11 フックプログラム
Fig. 11 Effect of Hook programs.

フックプログラムはスケルトンデータと大域変数の値を変更する．たとえば標準スケルトンデータにゴシック体の肉付けをするために以下のような変更が必要となる．

1. 横棒や縦棒の間の最小間隔を求め，それに応じて太さパラメータを調節
2. エレメントの始点や終点が他のエレメントの肉付け内部に隠れないように座標を変更
3. 不要なゲタ（縦棒の終点と横棒とでできるセリフ）の削除

図 11 の左は与えられたスケルトンデータとパラメータを変更なしに肉付けしたもので，右がフックプログラムによって変更後に肉付けしたものである．ただし，エレメントの種類や制御点の座標などが大幅に変化するためにフックによる修正の難しい字は書体ごとにデザインし直す必要がある．

エレメントの肉付けをプログラムで表現することによって，肉付けの自由度が上がった例として左払いの肉付けがある．明朝体の左払いは，長さや始点終点での角度の差などによって肉付けが大幅に変化するエレメントである．そのため，見本エレメントの変形によって肉付けをおこなう文献5)では，8種類の左払いを必要としている．肉付けをプログラムで実現した結果，本研究のシステムでは1種類の左払いを定義するだけで，**図 12** のような多様な肉付けを実現している．

同様のことはセリフの肉付けでも言える．縦棒の肉付けが太い時は，横棒の始点と縦棒等の始点との接続は，**図 13** の左側のように縦棒の範囲内でセリフをつけるが，細くなると，図の右のように縦棒の幅以上のセリフをつける補正をおこなう．

5. 評　価

以上の方針にしたがって，JIS X 0208, X 0212 に含まれる漢字 12156 文字について明朝体，角ゴシック体，丸ゴシック体の試作をおこなった．**図 14** はその一部である．見本とする漢字の選択は文献13)にならった．

まず，スケルトンフォントの採用によるデザインコストの低減の目安としてスケルトンフォントとアウトラインフォントのデータの量の比較を**表 1** に示す．アウトラインのデータとしては，スケルトンフォントに肉付けしたものを用いた．アウトラインフォントではエレメントという概念がないので，アウトラインを構成する直線，Bezier 曲線の数をエレメント数とした．

アウトラインと比較して明朝体の場合，エレメント数で約6の1，制御点数で，約5分の1のデータ量ですむことがわかる．エレメントの種類に違いはあるものの，エレメント数はほぼデザインコストに比例すると考えてよいので，書体によるスケルトンデータの共有を別にしてデザインコストが大幅に減らせたことが

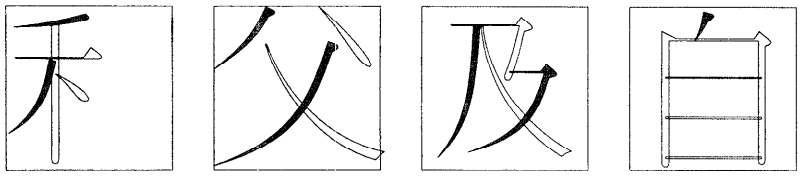


図 12 さまざまな左払い
Fig. 12 Variations of “hidari-harai.”

わかる．

明朝体用のスケルトンデータをゴシック体でそのまま利用することができなかった文字は，12156文字中

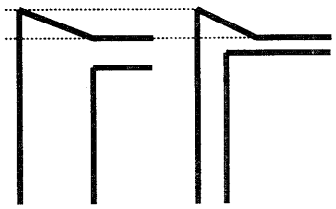


図 13 横棒の始点の接続
Fig. 13 Joint between vertical element and horizontal element.

表 1 12156文字のデータ量
Table 1 Data amount of 12156 characters.

	アウトライン（細明朝体）	アウトライン（丸ゴシック体）	スケルトン（明朝体）
エレメントの種類	2	2	16
制御点数	2030175	2404951	359811
（平均）	167.0	197.8	29.6
エレメント数	927239	1055835	169935
（平均）	76.3	86.9	14.0

の2702文字だった．ただ，その場合も一部のエレメントの種類や位置，接続を変更するだけで利用できた．

16種類用意したエレメントがどれだけの頻度で使われたかをまとめたが，**表2**である．あまり使われないエレメントもあるが，よく使われるエレメントは一文字あたり平均4回ほど使われていることがわかる．

次に，各エレメントの接続相手を**表3**に示す．非接続（始点終点）のセリフが32種類，接続によるセリフが25種類あることがわかる．しかし，本研究ではセリフ生成プログラムを共有できるようにしたため，**表4**に見るように書くプロラグムの種類を抑えることができた．

書体ごとの肉付けプログラムの製作コストを大まかに見積もるために，プログラムの行数で評価する．書体に依存しないシステムの各部分は4000行余りである．一方，書体の作成に伴う肉付けプログラムの記述は明朝体については862行，角ゴシックについては329行，丸ゴシックについては，角ゴシックの肉付けの変更部分だけを記述するため，84行で済んでいる．Lisp に関する基礎知識は必要なものの，割合少ないプログラム量ですんでいることがわかる．

図 14 試作したフォントの一部
Fig. 14 Samples.

6. まとめ

本研究では，多様な漢字書体に対応するスケルトンフォントを実現するために，肉付けアルゴリズムのうち，各書体で共通する部分だけを切り出し，

表 2　エレメントの出現頻度（JIS X 0208, X 0212 の漢字 12156 文字中）
Table 2　How many elements are used.

エレメントの種類	制御点の数	出現回数
点	2	19683
縦　棒	2	41099
横　棒	2	61344
上はね	3	4999
左払い	3	28368
縦棒＋左払い	4	306
右払い	3	3733
こざとへんの一部	4	516
縦棒＋はね	3	3667
つくりのはね	4	1335
さんずいの下	2	704
心の一部	4	2345
たすき	4	437
まがり縦棒	3	992
か　ぎ	3	76
しんにょうの一部	3	331

書体ごとに特徴的な肉付けはプログラムで記述することによって，新たな書体の作成を容易におこなえるシステムを作成した．

このシステムによって作成されたアウトラインフォントは一文字一文字を見たら，最初から人手でデザインした文字に匹敵するほどの水準に達するものも多数存在する．しかし，人間のデザインにおける，文字ごとに太さを変えて全体の黒さを調節したり，あるいは一文字の中でも同種のストロークでも太さを変えたりなどの細かい調節はおこなっていないので，不自然に見える字もある．

本システムでも，書体ごとにフックプログラムを指定することによって，スケルトンデータに変更を加えて座標を動かしたり，ヒント情報をつけることができるようにしている．しかし，実際に各書体に関してどのような規則に従って変更を加えれば良いかは，今後の課題である．

なお，本システムによって作られた JIS X 0208, X 0212 のアウトラインフォントは ftp.ipl.t.u-tokyo.

表 3　エレメントの接続
Table 3　How many times elements are jointed.

エレメント，始点終点	非接続	中途接続	接続相手
点始点	17072	2389	左払い終点(222)
点終点	19683	0	
縦棒始点	12559	7535	横棒始点(10673), 横棒終点(10243), 左払い終点(89)
縦棒終点	16241	11426	横棒始点(7587), 横棒終点(5772), 上はね終点＋しんにょうの一部始点(73)
横棒始点	25067	17783	縦棒始点(10673), 縦棒終点(5772), 左払い始点(1427), 心の一部始点(239), まがり縦棒始点(196), まがり縦棒終点(159), 縦棒左払い始点(16), かぎ始点(12)
横棒終点	26503	9606	縦棒始点(10243), 縦棒終点(7587), 左払い始点(3608), つくりはね始点(1848), 縦棒はね始点(1334), 心の一部始点(246), 右払い始点(174), まがり縦棒始点(141), たすき始点(54)
上はね始点	4999	0	
上はね終点	3444	1423	縦棒終点＋しんにょうの一部始点(73), 右払い始点(33), しんにょうの一部始点(26)
左払い始点	17769	5448	横棒終点(3608), 横棒始点(1427), 左払い終点(112), 左払い終点(4)
左払い終点	24828	2851	こざとへんの一部始点(262), 点始点(222), 左払い始点(112), 縦棒始点(89), 左払い始点(4)
縦棒左払い始点	119	171	横棒始点(16)
縦棒左払い終点	306	0	
右払い始点	338	3188	横棒終点(174), 上はね終点(33)
右払い終点	3733	0	
こざとへんの一部始点	120	134	左払い終点(262)
こざとへんの一部終点	516	0	
縦棒はね始点	1848	1189	横棒終点(630)
縦棒はね終点	3677	0	
つくりはね始点	1	0	横棒終点(1334)
つくりはね終点	1335	0	
さんずいの一部始点	704	0	
さんずいの一部終点	704	0	
心の一部始点	1191	669	横棒終点(246), 横棒始点(239)
心の一部終点	2345	0	
たすき始点	312	71	横棒終点(54)
たすき終点	437	0	
まがり縦棒始点	503	152	横棒始点(196), 横棒終点(141)
まがり縦棒終点	644	189	横棒始点(159)
かぎ始点	11	53	横棒始点(12)
かぎ終点	76	0	
しんにょうの一部始点	168	64	縦棒終点＋上はね終点(73), 上はね終点(26)
しんにょうの一部終点	331	0	

表 4 セリフプログラムの種類
Table 4 Number of serif programs.

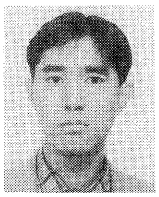
書 体	非接続のセリフ (32種類)	接続によるセリフ (25種類)
mincho	9	5
gothic	5	2
maru-gothic	7	8

ac.jp[130.69.168.19] から anonymous FTP で入手することができるようになっている.

参 考 文 献

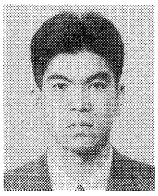
1) 荻原博也, 中沢和夫, 油田信一, 中島真人: 内外挿を用いた高品位なゴシック体日本語超大形文字の自動生成, 信学論(D), Vol. J 72-D-II, No. 9, pp. 1388-1396 (1989).
2) 上原徹三, 国西元英, 下位憲司, 鍵政秀子, 菊池純男: ストローク種別に基づく漢字形状生成方式, 情報処理学会論文誌, Vol. 31, No. 2, pp. 209-218 (1990).
3) 荻原博也, 中沢和夫, 油田信一, 中島真人: ゴシック体超大形日本語文字の自動生成, 信学論(D), Vol. J 72-D-II, No. 7, pp. 1048-1055 (1989).
4) 菊池純男, 大山恵三子, 高橋 栄: 字体のパラメトリック基本エレメント貼付け方式による高品質文字形状生成方式, 第29回情報処理学会全国大会論文集, 3 J-7 (1984).
5) 陳 和明, 小沢慎治: 多様な明朝体文字の規則的な生成, 信学論(D), Vol. J 72-D-II, No. 9, pp. 1423-1431 (1989).
6) 近山 隆: UtiLisp システムの開発, 情報処理学会論文誌, Vol. 24, No. 5, pp. 599-604 (1983).
7) 田中哲朗: SPARC の特徴を生かした UtiLisp/C の実現法, 情報処理学会論文誌, Vol. 32, No. 5, pp. 684-690 (1991).
8) Hobby, J. and Guoan, G.: A Chinese Meta-Font, Stanford Univ. STAN-CS-83-974 (1983).
9) 坂元宗和, 高木幹雄: 高品質明朝体ひらがな・カタカナフォントの計算機による生成, 信学論(D), Vol. J 68-D, No. 4, pp. 702-709 (1984).
10) 張 憲栄, 真田英彦, 手塚慶一: 漢字楷書毛筆字体の計算機による生成, 信学論(D), Vol. J 67-D, No. 5, pp. 599-606 (1984).
11) 下位憲司, 上原徹三: 骨格ベクトル文字からアウトライン文字への変換方式, 情報処理学会論文誌, Vol. 30, No. 9, pp. 1111-1118 (1989).
12) 奥村彰二, 前田正弘: 漢字画像から文字要素の自動抽出, 情報処理学会論文誌, Vol. 32, No. 1, pp. 50-61 (1991).
13) 佐藤敬之輔: 漢字 (上, 下), 丸善 (1973).
14) 上原徹三, 国西元英, 下位憲司, 鍵政秀子: 骨格ベクトル方式による文字形状の表現と生成, 信学論(D), Vol. J 74-D-II, No. 8, pp. 1020-1031 (1991).
15) 田中哲朗, 石井裕一郎, 長橋賢児, 竹内幹雄, 岩崎英哉, 和田英一: 漢字スケルトンフォントの生成支援システム, 第 32 回プログラミングシンポジウム報告集, pp. 1-8 (1991).
16) 石井裕一郎: Lisp による漢字フォント作成支援ツールの開発, 情報処理学会記号処理研究会資料, SYM 61-4 (1991).

(平成6年 4 月 11 日受付)
(平成6年 10 月 13 日採録)



田中 哲朗 (正会員)

1965 年生まれ. 1987 年東京大学工学部計数工学科卒業. 1992 年同大学院博士課程修了. 博士(工学). 東京大学工学部助手. 記号処理言語. 漢字フォント, ゲームプログラミングに興味を持つ. ACM, 日本ソフトウェア科学会各会員.



石井裕一郎 (正会員)

1967 年生まれ. 1990 年東京大学工学部航空工学科卒業. 現在同大学大学院工学系研究科情報工学博士課程在学中. 関数型言語と非決定性の関係等について興味を持つ. 日本ソフトウェア科学会, ACM 各会員.

竹内 幹雄 (正会員)

1966年生まれ. 1990年東京大学工学部計数工学科卒業. 1993年同大学大学院工学系研究科計数工学専攻修士課程修了. 同年日本アイ・ビー・エム(株)入社. 現在東京基礎研究所勤務. プログラミング言語処理系, プログラミング環境, 分散実行環境, オブジェクト指向方法論等の研究に従事. 日本ソフトウェア科学会会員.

和田 英一 (正会員)

昭和6年生まれ. 昭和30年東京大学理学部物理学科卒業, 32年同大学院修士課程修了, 小野田セメント(株)調査部統計課を経て昭和 39 年から東京大学工学部計数工学科助教授, 52 年同教授. この間昭和 48 年から 49 年まで M. I. T. 電気工学科助教授併任. 平成4年東京大学退官, 現在富士通研究所常任顧問, 東京大学名誉教授, 工学博士. 計量国語学会, ACM 各会員.