2008年度　卒　業　論　文

音喩利用を念頭に置いた書体デザインに関する研究

指導教員：渡辺 大地講師

メディア学部 ゲームサイエンス(企画・デザイン)
学籍番号　M0105143
川端　恵

2008 年度　　卒　業　論　文　概　要

論文題目

音喩利用を念頭に置いた
書体デザインに関する研究

メディア学部 **学籍番号：M0105143**	**氏名**	川端　恵	**指導教員**	渡辺 大地講師

キーワード	書体、フォント、文字、漫画、音喩

漫画は世界に広く認められた日本の文化の一つである。漫画の中には漫画独特の表現が多く、近年そのような表現を CG に応用する研究が行われている。日本の漫画の中によくある表現として音喩がある。音喩とは、漫画において書き文字として描かれたオノマトペ（擬音語・擬態語）を指す夏目房之介による造語であり、音喩は効果音から人の心情まで幅広く表現する。日本はオノマトペが豊富な言語であり、音喩の表現も多岐にわたる。通常の書体は文章として読みやすいという機能性が重要であるが、音喩は効果を強めるために様々な工夫がなされている。そのため音喩は独特の形が多く、従来の書体の研究では不可能な形が数多くある。これは特に端にある音喩の特徴を表現するために頂点の数が従来よりも多く必要で、局所的な操作ができないからである。また既存のフォント作成ソフトでは編集する頂点の数が多くなり、作成に時間がかかる。そこで本研究では、音喩で多く使われるカタカナを対象とし、音喩のような形状の文字の作成の支援に関する研究を行った。本手法で扱う文字データは、より音喩らしい特徴を持った文字に変形することが容易になるように、領域とよばれる文字の部品を任意に分け、また頂点の数を任意に変更できるようにした。その用意した文字データに、拡大・縮小といった基本的な変形、中心の軸を移動させ文字の形を変える変形、さらに音喩独特の領域の端々の変形を加え、音喩らしい文字を生成する手法を提案した。さらに実際に使われているような音喩と比較し、本研究で提案した手法の検証をした。実際に漫画で使われている音喩と、本手法を用いて作成した文字を比較し、その有用性を確認した。

目 次

図 目 次

第 1 章

はじめに

1.1 研究背景と目的

漫画は世界に広く認められた日本の文化の一つである。漫画の中には漫画独特の表現が多く、近年そのような表現を CG に応用する研究 [1][2] が行われている。日本の漫画の中によくある表現として音喩 [3] がある。音喩とは、漫画において書き文字として描かれたオノマトペ（擬音語・擬態語）を指す夏目房之介による造語であり、音喩は効果音から人の心情まで幅広く表現する。図 1.1 に音喩の例を示す。

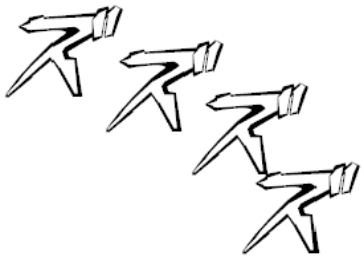

図 1.1: 音喩の例

このように、音喩にはその音を発した物や人の状態・心情といったものをあらわすために、文章などに使われる文字にくらべ複雑な形状をしている。戦後の漫画において音喩は多用されており、独自の進化を遂げ、現在では発音すら難しい音喩や漫画家オリジナルの音喩なども生まれている。より音喩の効果を強めるた

めに様々な工夫がなされることが一般的で、漫画家の個性が表れる部分の 1 つとなっている。

音喩にはデザインした人の個性と感性が大きく反映されているが、 このように優れたデザインの文字はデザインに慣れていない人が作るのは難しい。また音喩は読みやすい文字を目指したものではなく、迫力を出し、その場の雰囲気をより豊かに表現するためのものである。読めないほど大きく文字を崩したような音喩もある。

このような文字を他の映像作品やゲームといった他のコンテンツにも利用することでより表現の幅が広がるだろう。本研究では、日本のマンガに見られる音喩のデザイン的な特徴を持った文字を作る支援を目的として行った。今回の研究では、音喩でよく使われているカタカタを対象とし、音喩の形状について扱う。

文字の作成支援に関する先行研究として、ユーザーが冷たい、洗練された、重厚な、というような感性を入力して、それに応える最適なパラメータを求め、それを元に書体を生成する研究 [4][5][6][7] や、手書きでいくつかの文字を入力し、そこからそのユーザーの個性を求めて、手書き風の書体を生成する研究 [8][9] といったものがある。また、古印体の特徴を持った文字を自動生成する研究 [10] や、スケルトン構造から明朝体の特徴を持った文字を生成する研究 [11][12][13] がある。質感を重視した研究として、毛筆のかすれやにじみを表現する研究 [14][15][16][17][18] や、それに加えて文字を崩した連筆文字を生成する研究 [19] がある。しかしこれらの研究では音喩のような特殊な形状の文字を生成するすることはできない。文字の端々の形状や文字のゆがみといった音喩らしい形状を表現するためには、文字の一画一画に拡大縮小や回転等それぞれ違う変形を加える必要がある。従来の研究では、文字全体に対する変形しかできないため、音喩を作成する際には問題がある。また、文字を形作る頂点の数が通常のアウトラインフォントより多く必要である。図 1.2 は文字の端を音喩らしい形状に変形した文字の例である。

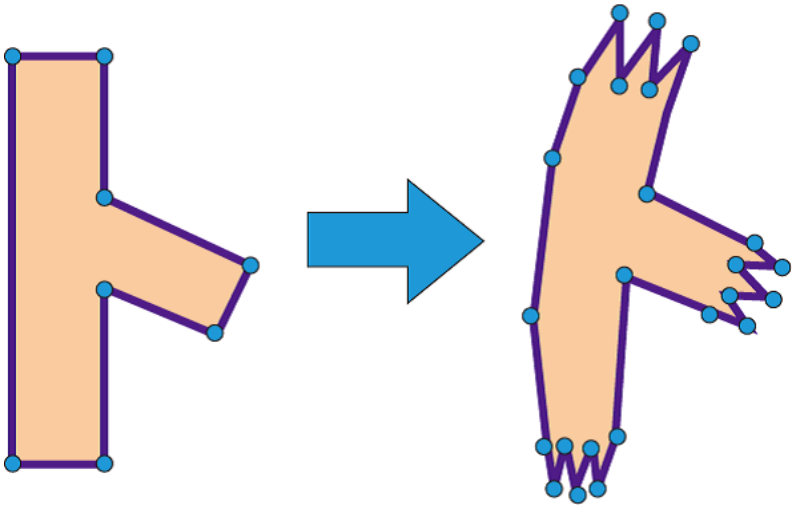

図 1.2: 音喩のように変形した例

このように音喩の特徴を持った文字に変形するためには通常より多くの頂点が必要となる。従来の研究では頂点の数が足りず、このような局所的な変形ができない。既存のフォント作成ソフトの場合は音喩を作成することは可能である。しかし先述のとおり通常より頂点の数が多いため、文字を編集し音喩のような形を作るためには非常に手間と時間がかかってしまう。

本研究ではこれらの問題点に対応できるように、音喩の独特な形状に対応するために独自の文字データを用意した。領域と呼ばれる文字の部品に任意で分け、頂点の数もユーザーの入力により決めることが出来るようにした。また領域ごとにその領域の軸を用意し、軸の動きに合わせて領域を変形出来るようにした。このデータを使うことにより、よりダイナミックに文字を変形でき、それによって音喩のような大きく崩れた字を作ることを可能になった。用意した文字データに、拡大・縮小といった基本的な変形、中心の軸を移動させ領域の形を変える変形、さらに音喩独特の領域の端々への変形を加え、音喩らしい文字を生成する。さらに増えた頂点をより楽に編集できるように、１つの操作で複数の頂点を一度に動かすことができるようにすることで効率的に編集することができるようにした。

1.2　本論文の構成

本論文は 4 章で構成する。第 2 章では、本研究で利用した文字データの構成と、その文字データに施す変形について述べる。第 3 章では、第 2 章で述べた手法で実装し、その結果から考察を行う。第 4 章で、本研究についてのまとめと今後の展望について述べる。

第 2 章

文字の構造と変形

2.1　文字の構造

本研究では、様々な変形に対応できるように、文書などで使われる通常のアウトラインフォントとは違う文字データを用意した。図 2.1 は、本研究で扱う文字データと通常の文字データの図である。

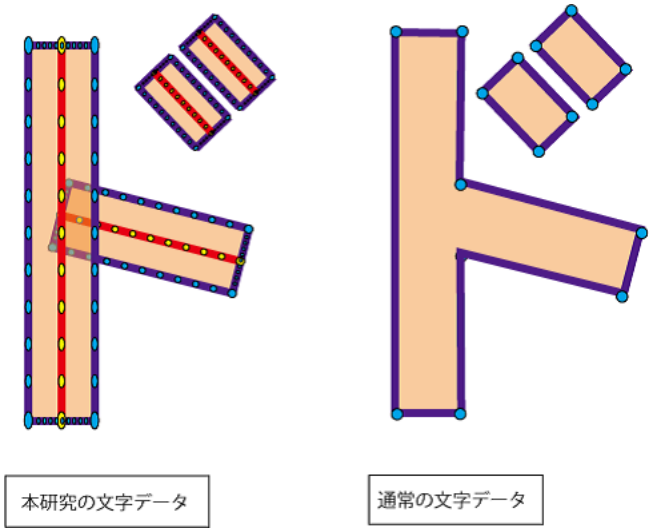

図 2.1: 文字データの違い

通常のアウトラインフォントの文字データは文字を形成する制御点、制御点を結ぶストローク、ストロークに囲まれた領域で構成している。本研究で扱う文字

データは、基本となる中心軸、中心軸を形成する中心軸点情報、文字を形成する文字制御点情報、制御点を結ぶストローク、そしてストロークに囲まれた領域、そしてこれらを囲むバウンダリボックスで構成している。図 2.2 に本研究の 1 つの領域の文字データを示す。

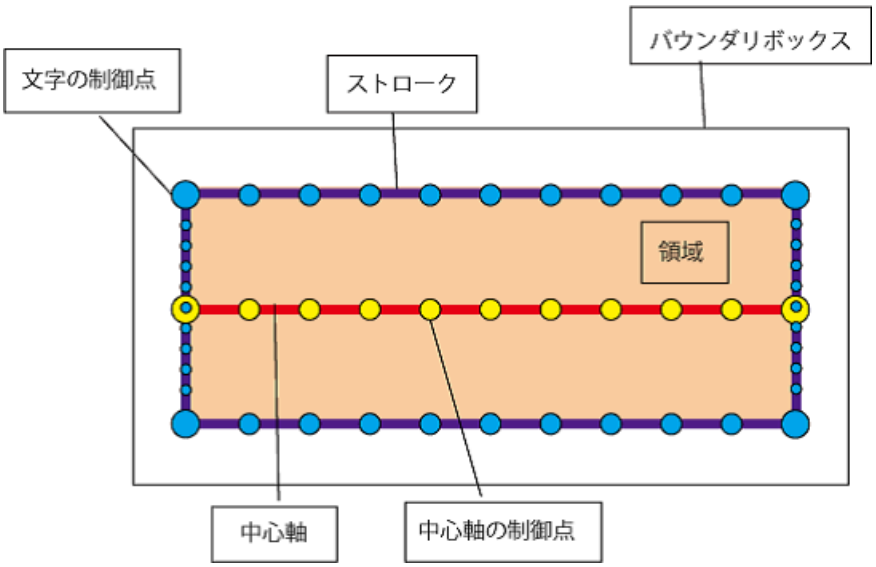

図 2.2: 1 つの領域の文字データ

通常の文字データでは、重なっている部分は 1 つの領域として扱う。本研究で扱う文字データは 1 画ごとに領域を分けた。例えば「ド」の文字の場合、1 画目の縦の棒と 2 画目の横の棒は重なっている。そのため、通常のアウトラインフォントでは図 2.1 の右側のように 1 つの領域として扱う。これを本研究では別の 2 つの領域として扱う。また、1 画でも「ム」のように途中で線の向きが大きく変わる場合は別の領域として扱う。領域の設定を細かくすることで、それぞれ領域別に変形を加えることができ、よりダイナミックな変形が可能になる。また、領域ごとに骨となる中心軸を作る。中心軸上にある中心軸制御点の個数はユーザーの入力によって決める。そうすることで無駄な制御点を作らず、その領域ごとに適した中心軸を作ることができる。さらに、その中心軸を基にして、領域を形作る文字制御点を作る。文字制御点は、中心軸制御点と関連付けする。中心軸制御点を動かすと、その点に連動して関連付けした文字制御点も動く。このような文字データ

にすることにより、ダイナミックな変形が可能となり、幅広い音喩の種類に対応することができるようになった。

1 つの領域を生成する手順は以下の通りである。

1. ユーザーが文字を入力する。
2. その文字の領域ごとに中心軸の両端点の位置を決める。領域の縦の長さと横の長さを比べ、長い辺に平行に軸を作る。
3. その両端点を結ぶ線分を任意の数だけ等分するようにさらに中心軸制御点を生成し、中心軸を作る。$(1-t) : t$ 等分する式は以下の式 (2.1) で表される。線分の始点を A、終点を B とし、それぞれの位置ベクトルを $\mathbf{a}$、$\mathbf{b}$ とする。線分 AB 上の点 P の位置ベクトルを $\mathbf{P}$ とする。

$$\mathbf{P} = (1-t)\mathbf{a} + t\mathbf{b} \qquad (0 \leq t \leq 1) \tag{2.1}$$

4. 中心軸の法線方向の 2 方向にそれぞれの中心軸制御点から任意の距離離れた文字制御点を生成する。
5. 生成した文字制御点の端と端を結ぶ線分を任意の数だけ等分するようにさらに文字制御点を生成する。等分する式は (2.1) と同じように表される。
6. 点同士をストロークで結び、文字の領域を形作る。
7. 文字を囲むバウンダリボックスをを生成する。このとき、バウンダリボックスの各辺は x 軸や y 軸と平行である。

2.2 文字の変形

ここでは 2.1 節で提案した手法で用意した文字データに加える変形について述べる。本研究で扱う変形は大きく分けて 3 つある。

- 基本変形

- 領域の曲線変形
- 音喩の特徴的な変形

以下でそれぞれの変形の処理についての説明を述べる。

2.2.1 基本変形

本手法で扱った領域の変形とは以下の 5 つの変形 [20][21] である。

1. 拡大・縮小
2. 回転
3. 平行移動
4. せん断（スキュー）
5. 台形変形

x 軸・y 軸方向の拡大・縮小、せん断、台形変形の場合、文字制御点にも変形を加えると、中心軸制御点と文字制御点の関係が崩れてしまう。そのため、今回の研究ではこれらの変形は中心軸制御点のみに行い、文字制御点との関係を保つようにした。また、これらの変形は文字全体に加えることも領域ごとにそれぞれ変形を加えることもできる。中心軸の一部にだけ変形を加えることはできない。以下でそれぞれの変形について述べる。また、ある点 $A(x, y)$ の移動後の頂点 $A'(x', y')$ を求める式を示す。

・拡大・縮小

拡大・縮小は、以下の式 (2.2) で表す。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 1 \\ 0 & s_y & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & -q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.2}$$

ここで、s_x、s_y は、それぞれ x 方向、y 方向の拡大・縮小率で、その値が 1 より大きいときは拡大、1 より小さいときは縮小となる。

・回転

バウンダリボックスから、その文字や領域の中心となる座標 (p, q) を求める。その座標を中心に、半時計回りに角度 θ だけ回転する変換は、以下の式 (2.3) で表す。

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & -q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.3}$$

・平行移動

x 軸方向の移動量を t_x、y 軸方向の移動量を t_y としたとき、平行移動は以下の式 (2.4)、式 (2.5) で表す。

$$x' = x + t_x \tag{2.4}$$

$$y' = y + t_y \tag{2.5}$$

・せん断（スキュー）

長方形を傾けて平行四辺形にするような変換をせん断（スキュー）という。この変形は中心軸に対して行う。図 2.3 はそれぞれ x 軸方向、y 軸方向に θ だけ傾けたせん断の例である。

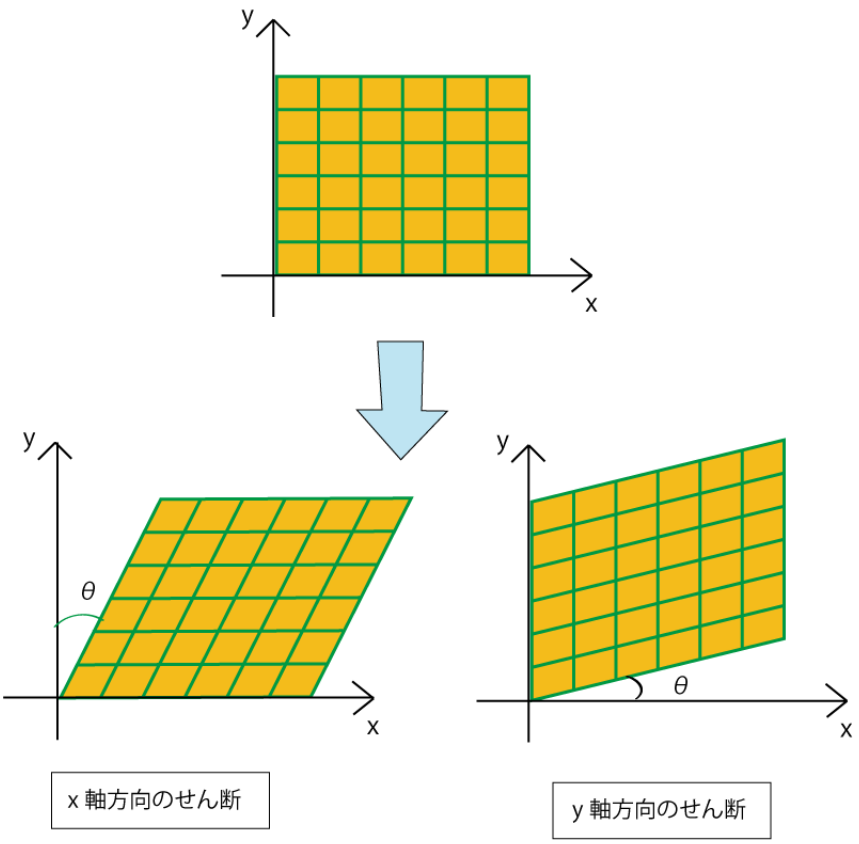

図 2.3: せん断

x 軸方向に角度 θ だけ傾けるせん断は以下の式 (2.6) で表す。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & tan\theta \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.6}$$

また、y 軸方向に角度 θ だけ傾けるせん断は次の式 (2.7) で表す。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ tan\theta & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.7}$$

・台形変形

長方形の辺の長さを変えて台形にするような変換を台形変形という。この変形は中心軸に対して行う。図 2.4 に台形変形の例を示す。

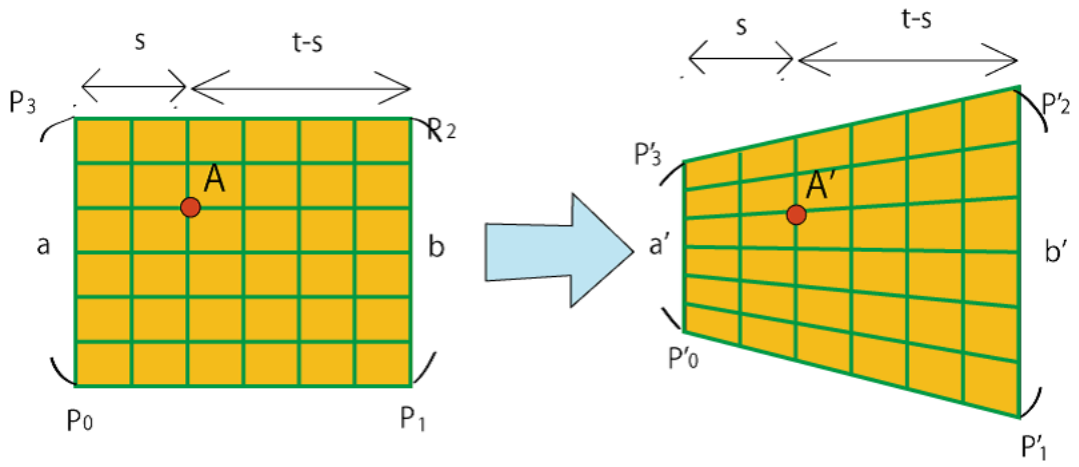

図 2.4: 台形変形

バウンダリボックスの頂点をある頂点から反時計回りに順に P_0、P_1、P_2、P_3 とする。y 軸方向にバウンダリボックスの頂点を動かした場合を考える。P_0 と P_3 の距離を a、P_1 と P_2 の距離を b と置く。P_2 と P_3 の距離を t と置いたとき、P_3 の x 成分とある制御点 A の x 成分の差を s とする。a の長さが a'、b の長さが b' に変形したとき、そのバウンダリボックスの中に位置するある制御点 $A(x, y)$ の移動後の制御点 $A(x', y')$ は以下の式 (2.8) と式 (2.9) で求まる。

$$x' = \frac{(t-s)a' + sb'}{(t-s)a + sb}x \tag{2.8}$$

$$y' = \frac{(t-s)a' + sb'}{(t-s)a + sb}y \tag{2.9}$$

x 軸に対してバウンダリボックスの頂点を動かした場合も同様に求められる。

2.2.2 領域の曲線変形

領域を曲げる表現や段を付ける表現は中心軸と、中心軸制御点それぞれに関連付けされた文字制御点を用いて行う。中心軸点を動かし、その中心軸点と関連付

けしている文字制御点を同じ距離を保つように動かす。また、中心軸制御点と文字制御点の距離を変えることによって、太さの変形ができる。この 2 つを組み合わせることで、領域の様々な変形に対応することができる。図 2.5 に中心軸を用いた変形の例を示す。

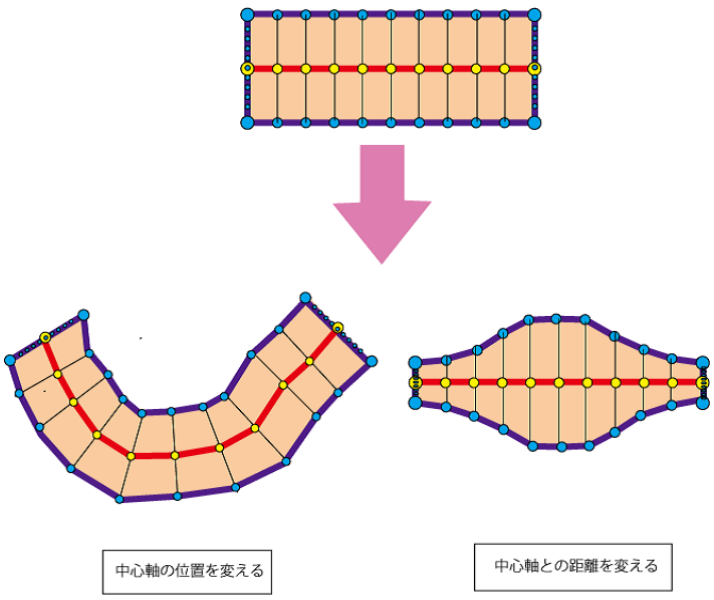

図 2.5: 中心軸を用いた変形

中心軸制御点を動かした場合、2.2.3 節で述べる中心軸に垂直なストローク上にある文字制御点に加えた変形を保ったまま移動する。アルゴリズムは以下の通りである。

1. 初期状態の中心軸に垂直なストロークの長さに対する、それぞれの文字制御点と中心軸との距離の割合を算出する。

2. 動かした中心軸制御点の両隣にある中心軸制御点同士を結ぶ線分の法線ベクトルを求める。

3. 動かした中心軸制御点に対応した文字制御点の 2 点を、動かした中心軸制御点から求めた法線方向へ、関連付けされた中心軸制御点との距離を保ったまま移動する。

4. 中心軸に垂直なストローク上にある文字制御点を、動かした中心軸制御点とその隣の中心軸制御点を通る中心軸に平行に、それぞれの文字制御点の移動量だけ移動する。

図 2.5 の右のように領域の太さを変更する場合は、中心軸制御点と文字制御点の距離を変更することで実現できる。

2.2.3 音喩の特徴的な変形

より音喩らしさを出すために、領域の端にあたる中心軸に垂直なストローク上にある文字制御点を用いて変形する。本研究で行った、中心軸に垂直なストローク上にある文字制御点を用いた変形は以下の 3 つである。

1. 刻み目を付ける
2. 先を鋭利にする
3. 先に丸みを付ける

以下で個々の処理について説明する。文字の端の両端点 A と B の座標をそれぞれ (x_a, y_a)、(x_b, y_b)、ある文字制御点 (i) の座標を (x_i, y_i)、移動後の (i) の座標を (x'_i, y'_i) と置く。

・刻み目を付ける

図 2.6 は刻み目のある音喩の例である。

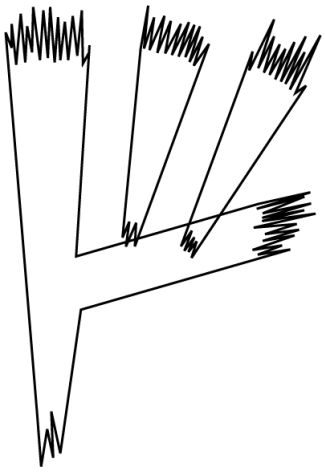

図 2.6: 先に刻み目を付けた音喩の例

このように文字の端にのこぎりの歯のような刻み目を付ける手法を述べる。文字の端の両端点を結んだ線分に対して垂直に中央軸に垂直なストローク上にある文字制御点群を動かす。このとき、隣り合った文字制御点同士は両端点を結んだ線分に対して反対側へ動くように設定する。そうすることでギザギザとした刻み目の表現ができる。

図 2.7 は、文字制御点を刻み目状に動かした状態の図である。

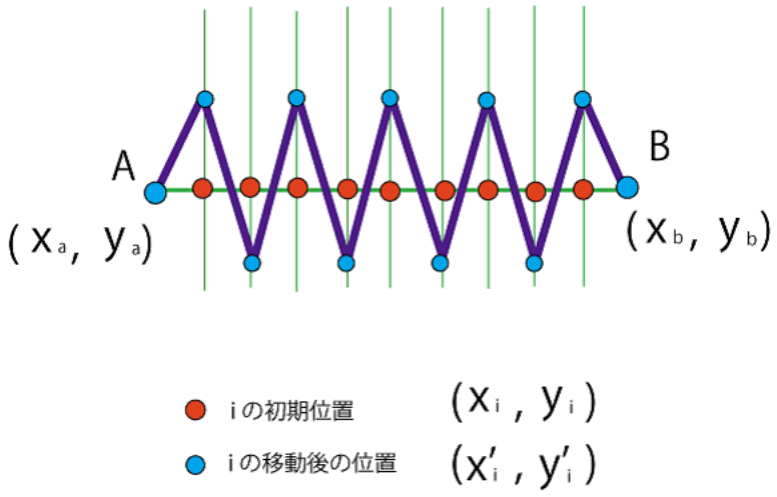

図 2.7: 刻み目を付けたときの制御点

文字の端に刻み目を付ける手順は以下の通りである。

1. ベクトル $\overrightarrow{AB}$ の法線ベクトルを求める。

2. 両端点以外の中央軸に対して垂直なストローク上にある文字制御点を任意の距離とそれにランダムな値を加えて法線方向へ動かす。このとき、隣り合った文字制御点は逆の法線方向へ動くようにする。

3. 動かした文字制御点を端から順番に線で結ぶ。

・先を鋭利にする

図 2.8 は先を尖らせた音喩の例である。

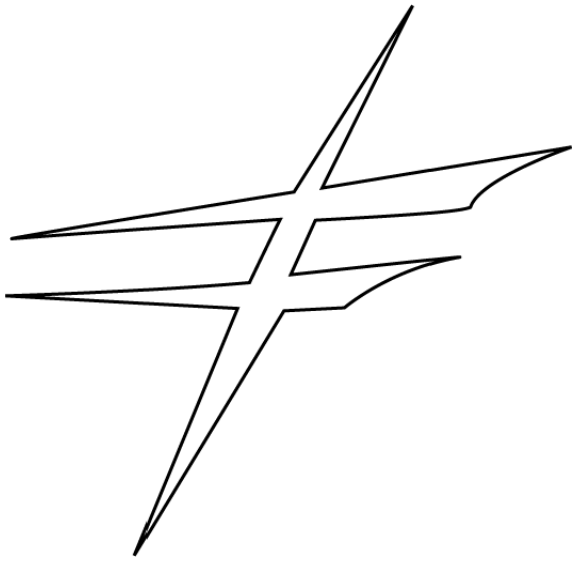

図 2.8: 先を尖らせた音喩の例

このように文字の端を鋭利にする方法を述べる。文字の端の両端点を結んだ線分に対して垂直に、中央軸に垂直なストローク上にある文字制御点群中の任意の 1 点を動かす。その動かした 1 点と両端の点 A と点 B をそれぞれ結んだ線分上に他の文字制御点も動かす。こうして先のとがった表現ができる。

図 2.9 は先を尖らせるように制御点を移動させた状態の図である。

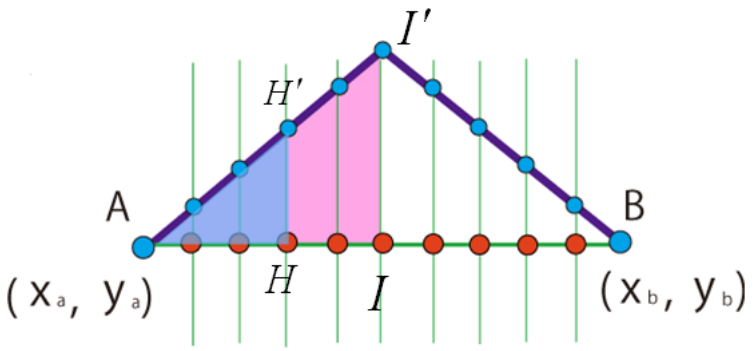

図 2.9: 先を尖らせたときの制御点

文字の端を鋭利にする手順は以下の通りである。

1. ベクトル $\overrightarrow{AB}$ の法線ベクトルを求める。

2. 中央軸に対して垂直なストローク上にある文字制御点のうち、ある点 I を任意の距離だけ法線方向へ動かす。

3. 動かした 1 点と両端の点 A と点 B をそれぞれ結んだ線分上に他の文字制御点も動かす。I の移動後の点を I' とする。このとき、他の制御点 H とその移動後の点 H' の距離 $\overline{HH'}$ は以下の式 (2.10) で求められる。$\overline{II'}$ は I と I' の距離、$\overline{AH}$、$\overline{AI}$ はそれぞれ点 A と H、I との距離である。

$$\overline{HH'} = \frac{\overline{II'} \cdot \overline{AH}}{\overline{AI}} \tag{2.10}$$

   H が I に対して点 B 側にある場合は点 B との距離を使って求める。

4. 動かした文字制御点を端から順番にストロークで結ぶ。

なお、刻み目をつける変形と先を尖らせる変形は同時に利用できるようにした。これは、刻み目にメリハリをつけ単調にならないようにするためである。

・先に丸みを付ける

図 2.10 は先に丸みがある音喩の例である。

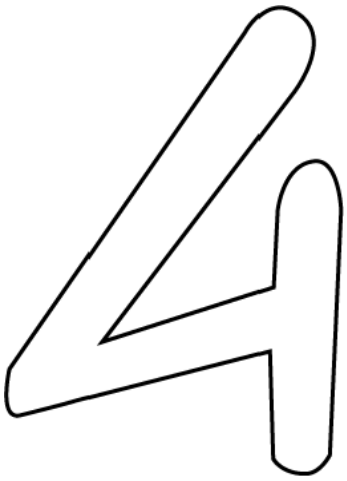

図 2.10: 先に丸みのある音喩の例

このように文字の端を丸くする方法を述べる。文字の端を丸くする場合、中心軸に対して垂直な方向へ半楕円を描く。楕円の長軸はユーザーの入力によって任意に決められ、短軸の長さは点 A と点 B の距離にし、半楕円を描く。ストロークを滑らかに見せるため、楕円では中心軸に垂直なストローク上にある文字制御点は利用しない。

図 2.11 は先を丸めるように制御点を移動させた状態の図である。

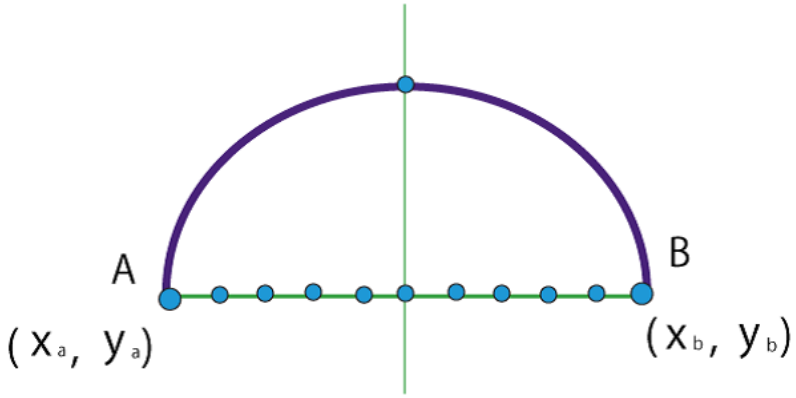

図 2.11: 先を丸めたときの制御点

文字の端を丸める手順は以下の通りである。

1. ベクトル $\overrightarrow{AB}$ の法線ベクトルを求める。

2. 短軸のベクトル $\overrightarrow{AB}$ の長さ a を求める。長軸の長さ b を求める。

3. 楕円を描く。楕円は以下の式 (2.9)、式 (2.10) で表す。

$$x = acos\theta \tag{2.11}$$

$$y = bsin\theta \tag{2.12}$$

4. 中心軸の傾きの角度に応じて楕円を回転する。

第 3 章

結果と考察

3.1 実装

第 2 章と第 3 章で説明した内容で実装を行った。実装にはグラフィックスツールキットである FK Tool Kit System[22] を使用した。また、文字の領域をフォント作成ツールである TTEdit[23] を利用し設定した。文字の読み込みには Free type[24] を使用した。図 3.1 は本手法を用いて実装したプログラムの実行画面である。

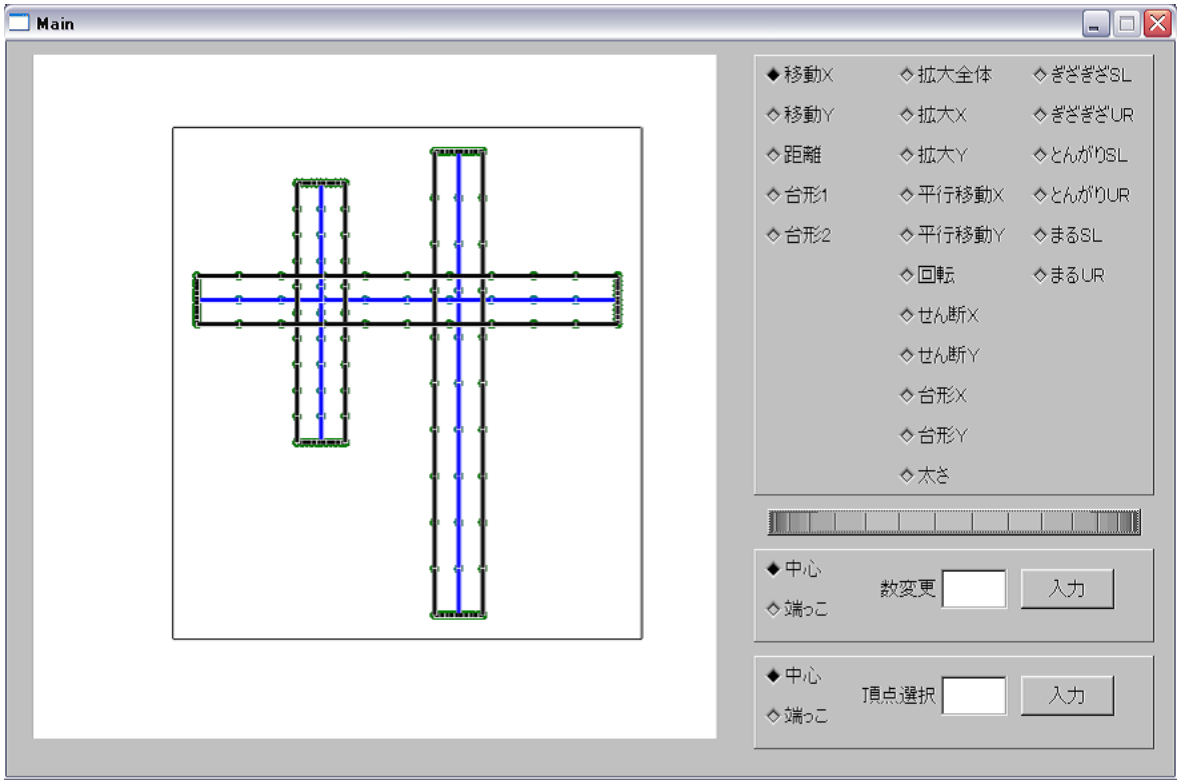

図 3.1: 実行画面

これは左の画面に入力した文字を表示する。そして、一番上のメニューから加えたい操作を選び、中心のスクロールバーを動かす。また、二つ目のメニューでは中心軸と端にあたる部分の制御点の数を変更できる。一番下のメニューでは中心軸と端にあたる部分にある制御点のうち、一番上のメニューで行う動作を加える制御点を選ぶことができる。また、マウスの操作で中心軸の制御点を選択し、移動することができる。

3.2 実行結果

図 3.2 は本研究の手法で生成した文字「ビ」の図である。

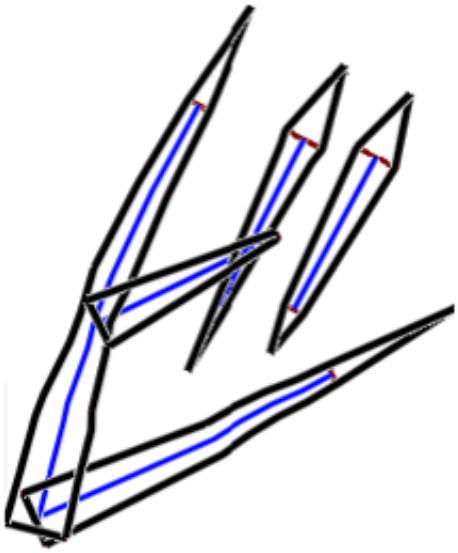

図 3.2: 実行例 1

図 3.2 の作成方法として、領域ごとに基本変形を加え位置を調整した。さらに中心軸制御点の位置や、その中心軸制御点に関連付けされた文字制御点との距離を調整し、領域にゆがみを加えた。最後に領域の端を鋭利にし、図 3.2 のような文字を作成した。これら操作にかかった時間は 7 分であった。このようにして線の鋭さやゆがみの表現ができた。

図 3.3 は本研究の手法で生成した文字「ド」の図である。

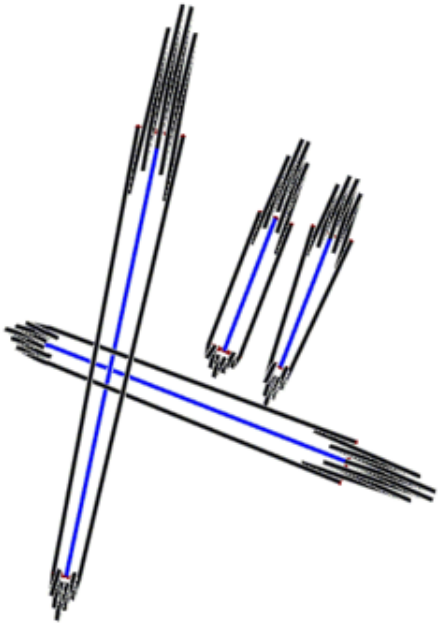

図 3.3: 実行例 2

図 3.3 の作成方法として、領域ごとに基本変形を加え位置を調整した。中心軸制御点と、それに関連付けされた文字制御点との距離を変更した。領域の端を鋭利にし、そこにさらに刻み目を加え、図 3.3 のような文字を作成した。これら操作にかかった時間は 5 分であった。「ド」の横棒が縦棒を突き抜ける表現や、端の刻み目の表現を実現することができた。

図 3.4 は本研究の手法で生成した文字「ム」の図である。

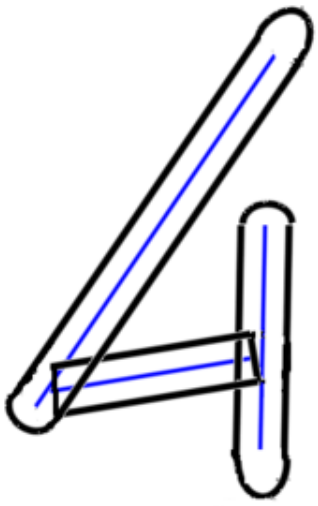

図 3.4: 実行例 3

領域ごとに基本変形を加え位置を調整した。さらに領域の端に丸みを加え、図 3.4 のような文字を作成した。これら操作にかかった時間は 5 分であった。丸みのある文字の表現は出来たが、領域が重なっている部分がうまく表現できなかった。

本研究でのソフトを用いることで音喩と特徴を持った文字を生成することができた。時間も 1 文字につき 5 分前後と短時間で作成することができた。イラストレーターなどの画像処理ソフトでは難しい音喩らしい特徴である先端の表現が本手法を使うことにより簡単にできた。ツールで行うことができる先端の表現をさらに増やすことでより多くの音喩の表現ができるだろう。また、今回考慮していない点で、領域を自由につなぐ・離すという点がある。これをできるようにすることでよりクオリティの高いものが作ることができるだろう。

3.3 検証と考察

検証としてフォントの作成をしたことがない学生 4 人に実際に音喩を作ってもらった。2 人がある程度デザインやイラストレータなどの画像処理のソフトを使ったことがある学生で、もう 2 人がデザインや画像処理ソフトに慣れていない学生

である。同じデザインの「ド」の音喩を参考として作成してもらった。以下の図3.5、図3.6、図3.7、図3.8はそれぞれの学生に、フォント作成ソフトと本研究で作成したソフトの両方で作成した図である。

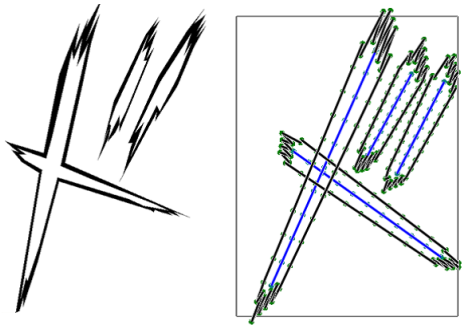

図3.5: 検証1

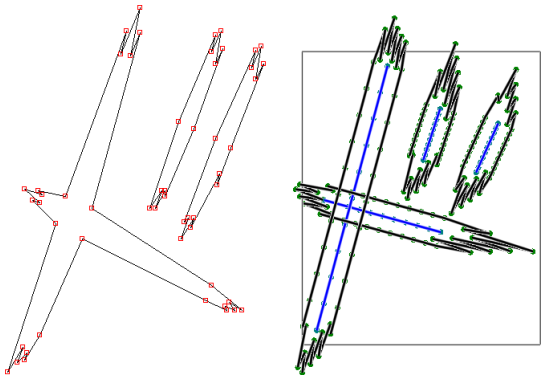

図3.6: 検証2

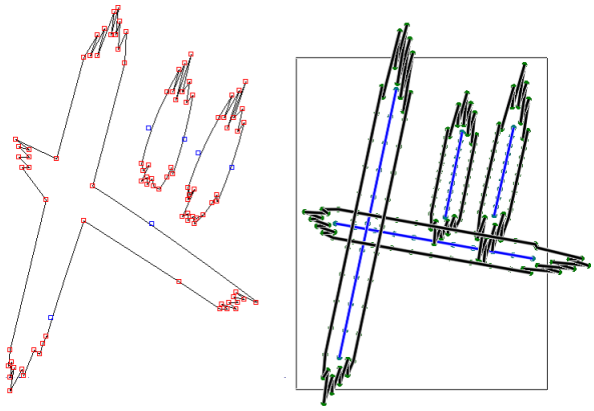

図 3.7: 検証 3

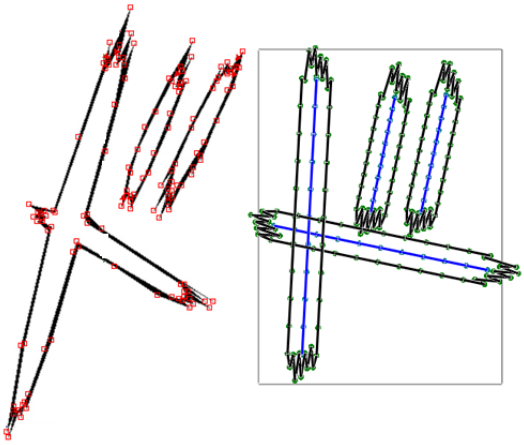

図 3.8: 検証 4

それぞれのソフトの学習は 10 分程行った。検証 1 、検証 2 の学生はデザインに慣れている学生である。検証 1 の学生の場合、フォント作成ソフトでの時間が 30 分、本研究のソフトでの時間が 7 分であった。検証 2 の学生の場合、フォント作成ソフトでの時間が 19 分、本研究のソフトでの時間が 4 分であった。検証 3、検証 4 の学生はデザインに慣れている学生である。検証 3 の学生の場合、フォント作成ソフトでの時間が 18 分、本研究のソフトでの時間が 5 分であった。検証 4 の学生の場合、フォント作成ソフトでの時間が 60 分、本研究のソフトでの時間が 10 分であった。このように両者において時間を大きく軽減することができた。このソ

フトを利用して作成した場合、デザインに不慣れかどうかに関係なく、目標とする文字を同じように作成することができた。また、フォント作成ソフトでは制御点をたくさん置いていくのに手間がかかる、刻み目の表現が難しいといった問題があったが、今回の研究のソフトの場合はそのような点が楽にできたという感想を得られた。しかし、インターフェースが分かりにくいという感想もあり、この問題点を解決し、より直感的な操作ができるよう改良する必要がある。

第 4 章

まとめ

本論文の締めくくりとして、まとめと今後の展望に関して述べる。本研究では、書体として漫画の中に手書きされている文字である音喩に着目した。独特な形を持つ音喩のために本研究独自の文字データを用意した。この文字データは一画ごとに文字の領域を持ち、その領域ごとに中心となる軸を作り、その軸の制御点と文字のストロークを描く制御点が対応している。中心軸の制御点を動かしたり、中心軸制御点と関連付けされた文字制御点との距離を変えることにより、文字の領域を変形させることができる。それに、拡大・縮小といった基本的な変形や、刻み目を付ける、鋭利にする、丸みを付けるという音喩独特の変形の 3 つの変形を加え、音喩らしい文字を生成する手法を提案した。これらの変形を組み合わせて音喩らしい特徴を持った文字を生成することができた。

今回は音喩の形状に注目した手法を提案した。先端の変形のバリエーションを増やすことで、より多くの音喩を簡単に表現できるようになる。今後の展望として、ストロークの太さの強弱を加えることでより音喩らしい文字を生成できるであろう。また、今回着手していないものとしてスクリーントーン等領域の質感の表現がある。スクリーントーン（Screen Tone） とは、マンガ・デザイン用画材のひとつで、糊のついた透明なフィルムに模様 （トーン） が印刷されたもののことである。スクリーントーンを多数用意すれば、より音喩らしさを表現することができるであろう。さらに、ストロークそのものの質感も、例えば毛筆の筆跡のかす

れた表現や鉛筆の硬い筆跡の表現というように様々なものを用意することで、より幅広い表現が可能になる。

謝辞

本論分を締めくくるにあたり、研究手法や論文執筆についてご教授を頂きました渡辺大地講師及び、ゲームサイエンスプロジェクトのスタッフの三上浩司講師、小澤賢侍氏、中村太戯留氏に心より感謝いたします。また、研究において大変お世話になりましたゲームサイエンスプロジェクト研究室の先輩方に感謝いたします。そして、同じ時間を戦い抜き、つらいときにも支えてくれたゲームサイエンスプロジェクト研究室の仲間たちに深く感謝いたします。

参考文献

[1] 丹治 宏文, ”リアルタイム 3DCG ツールキット上での絵画調レンダリングの実装とその効果に関する研究”, 東京工科大学メディア学部卒業論文, 2002.

[2] 川崎 真吾, ”漫画的動線によるカメラの位置を考慮したスピード表現の研究”, 東京工科大学メディア学部卒業論文, 2007.

[3] 夏目房之介, ”マンガの力-成熟する戦後マンガ ”, 晶文社, 1999 年.

[4] 和田章男, 堀田創, 萩原将文, ”多様なデザイン機能を備えた感性を反映する日本語フォント自動生成システム”, デザイン学研究,52(6), pp.9-16, 2006.

[5] 和田章男, 萩原将文, ”感性を反映できる日本語フォント自動作成システム”, 感性工学研究論文集, 5(2), pp.1〜8, 2005 .

[6] 堀田創, 萩原将文,”感性ルールベースを用いた日本語フォント自動作成システム”, 情処学論 vol.48, no.3, pp.1491-1501, 2007.

[7] 野澤 貴, 堀田 創, 萩原 将文,”パラメータ拡張可能な書体定義法とフォント自動作成システムへの応用”, 電子情報通信学会論文誌 D Vol.J90-D No.10 pp.2755-2764, 2007.

[8] 安本護, 池田尚志, 堀井洋, ”大域的個人性と局所的個人性に基づく手書き風フォントの生成”, 電子情報通信学会論文誌, D-II, 情報・システム,I I-情報処理, J80-D-2(11), pp.2930-2939, 1997.

[9] 安本護, 池田尚志, 豊倉 完治, ”部分字形組合せを用いた手書き風フォントの自動生成”, 電子情報通信学会技術研究報告. PRU, パターン認識・理解, Vol.94, No.423, 1994.

[10] 安居院猛, 井手賢一, 長橋宏, 長尾智晴, ”文字輪郭線の変形による古印体フォントの自動生成”, 電子情報通信学会論文誌, ’93/12 Vol.J76-D-II, No.12, pp.2652-2656,1993.

[11] 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 .

[12] 田中 哲朗, 石井 裕一郎, 竹内 幹雄, 和田 英一 ”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” 情報処理学会論文誌, Vol. 36, No. 1, pp. 177 – 187,1995.

[13] 田中 哲朗, 岩崎 英哉, 長橋 賢児, 和田 英一 ”部品合成による漢字スケルトンフォントの作成” 情報処理学会論文誌, Vol. 36. No. 9, pp. 2122-2131, 1995.

[14] 中村 剛士, 真野 淳治, 世木 博久, 伊藤 英則, ”毛筆フォントの掠れ・滲み処理システムについて”, 情報処理学会論文誌, 38(5), pp.1008-1015, 1997.

[15] 真野 淳治, 中村 剛士, 世木 博久, 伊藤, 英則, ”毛筆書体におけるくりこみ群を用いたかすれ・にじみ表現”, 情報処理学会論文誌, 38(4), pp.806-814, 1997.

[16] 中村 剛士, ”フラクタルを用いた毛筆文字のかすれ表現について”, 日本ファジィ学会誌, 8(3), pp.558-566, 1996.

[17] 中村 剛士, ”筆記速度のファジィ評価方法を導入した毛筆文字生成システムについて”, 日本ファジィ学会誌, 7(2), pp.371-379, 1995.

[18] 朝倉 幹人, 中村 剛士, 伊藤 英則, ”5D-1 文字構造データベースを用いた毛筆フォントの掠れ・滲み処理について”, 全国大会講演論文集, 第 58 回平成 11 年前期 (2), pp.”2-37”-”2-38”, 1999.

[19] 山田 晃嗣, 江野脇 宏, 中村 剛士, 何 立風, 伊藤 英則, ”楷書オンライン入力からの連筆文字生成法について”, 日本ファジィ学会誌, 13(5), pp.66-75, 2001.

[20] ”ディジタル画像処理”, CG-ARTS 協会, 2007.

[21] 山北篤, ”コンピュータゲームの数学”, 新紀元社, 2006.

[22] 渡辺大地, FK Tool Kit System, ⟨ http://fktoolkit.sourceforge.jp/ ⟩.

[23] 武蔵システム, ⟨ http://musashi.or.tv/ttedit.htm ⟩.

[24] David Turner, Robert Wilhelm, Werne Lemberg, Free Type⟨ http://savannah.nongnu.org/projects/freetype/ ⟩.