〈研究ノート〉

X−Windowにおける変形文字列の生成手法とその実装

小山職業能力開発短期大学校　　森　下　　茂

A Technique Generating Transformed Text for X-Window and Its Implement

Shigeru　MORISHITA

要約　X−Window上でのアプリケーションプログラミングにおいて変形文字列を生成するための一手法を提案する。ここで変形文字列とは、回転や拡大、縮小などの変形操作を施された文字列のことを言う。

X−Windowには、文字列単位で変形操作を施すためのAPIがサポートされていない。このため図形処理アプリケーションなどの開発において、制約を生じることが多い。

本手法は、文字列データを画像データとしてメモリ上に展開し、画像処理的な方法を用いて、文字列単位の変形操作を実現するものである。このため、X−Windowで提供される豊富なフォントデータを、そのまま利用して変形文字列を生成することが可能となる。フォントデータとしては漢字データも使用できる。

本手法の有効性を確認するため、X−Window上でC言語による実装を行い、その実行例を示した。実装は、アプリケーション開発での使用を前提とし、ライブラリ形式とした。Xlibと併用するため、APIはXlibの形式に合わせた。この結果、アプリケーションプログラミングレベルで手軽に使え、実用性のある手法であることが確認できた。X−Windowにおける図形処理技法の基礎技術として、実務上、あるいは教育訓練の分野において広範囲な活用が期待できると考える。

1　はじめに

X−Window上でのアプリケーション開発、特にCADシステムなどに代表される図形処理アプリケーションなどの開発において、変形文字列を使用したいという要望は多い。ここで、“変形文字列”とは、回転や拡大・縮小、反転などの変形操作を文字列単位で施された文字列のことを言う。

X−Windowでは、文字列表示を行うために豊富なフォントデータをサポートしており、C言語インターフェースであるXlibを通して、アプリケーションプログラミングレベルでも利用可能となっている。しかしながら、文字列単位の変形操作については具体的に、API（アプリケーションプログラミングインターフェース）として与えられていない。X11R4においてはアウトラインフォントに対して、またX11R5からはビットマップフォントに対しても、ポイントサイズの選択という形で、フォントファイル単位での拡大、縮小機能がサポートされているが、回転および反転についてはサポートされていない、また最新のリリースであるX11R6においては、フォント名を拡張する形でフォントデータへのアフィン変換が可能となったが、これも基本的にはフォントファイル単位での変換であり、処理時間に難のあること、またプログラマにアフィン変換に対する知識が必要とされるため、アプリケーションプログラミングレベルからの使い勝手も必ずしも良いとはいえない。これらの変形操作がX−Windowの仕様上提供されても、それが実現されるかどうかは、基本的にはベンダやフォントの種類に依存し、すべてのフォントデータに対して変形操作が可能とは限らな

論文受付日　H6．11．17

い。このため、図形処理アプリケーションなどの開発において、制約を生じることも多い。

このような現状をふまえ、アプリケーションプログラミングレベルからX－Windowの豊富なフォントデータをそのまま利用し、変形文字列を生成する手法を考案した。また、本手法の有効性を確認するため、X－Window上で実装を行ない、その実行例を示した。実装は、アプリケーション開発での使用を前提に、手軽に利用できることを目的として、利用者から見た場合に、本手法がブラックボックスとなるように、ライブラリ形式とした。Xlibとの併用が必要であるため、APIもXlibの形式に合わせた。この結果、アプリケーションプログラミングレベルで手軽に使え、実用性のある手法であることが確認できた。

本手法の適用により、次のような効果が期待できる。

(1) フォントファイル単位の変換でなく、文字列単位の変換が可能となる。

(2) ライブラリ化により、アプリケーションプログラミングにおいて、アフィン変換の知識がなくても変形文字列を手軽に生成、利用できる。

(3) Xlibで表示できるフォントデータであれば、その種類に依存せず、変形文字列を生成できる。したがって、X－Window上でサポートされる豊富なフォントデータを利用できる。

(4) X－Windowのリリースバージョンに関係なく変形文字列が生成可能となる。

II　変形文字列の生成手法

1．手法の概要

本手法は、表示文字列を一種の画像データとしてとらえ、これに、画像処理的な手法でアフィン変換を施すことにより、変形文字列を生成する方法である。具体的には、対象文字列を直接画面上に描画せずに、一度、画像データとしてメモリ上に展開したあとで、ピクセル単位に必要な変換を施して、別に用意したメモリ上に出力画像を生成する。この後、出力画像の背景部を透明化して表示を行う。

2．変形操作の仕様

本手法で対象とする文字列の変形操作のための仕様を示す。

(1) 文字列の表示および変換操作は、スクリーン座標系で行う。スクリーン座標系は、描画ウィンドウの左上隅を原点とし、右方向がx正方向、下向きがy正方向の座標系である。文字列の原点は、スクリーン座標系で与えられる。

(2) 文字列データのメトリクスはX－Windowでの定義に従う。メトリクス情報から文字枠を算出する。

(3) 変換操作としては、拡大・縮小、反転、回転を可能とする。

(4) 変換中心は、文字列の原点とする。また回転方向

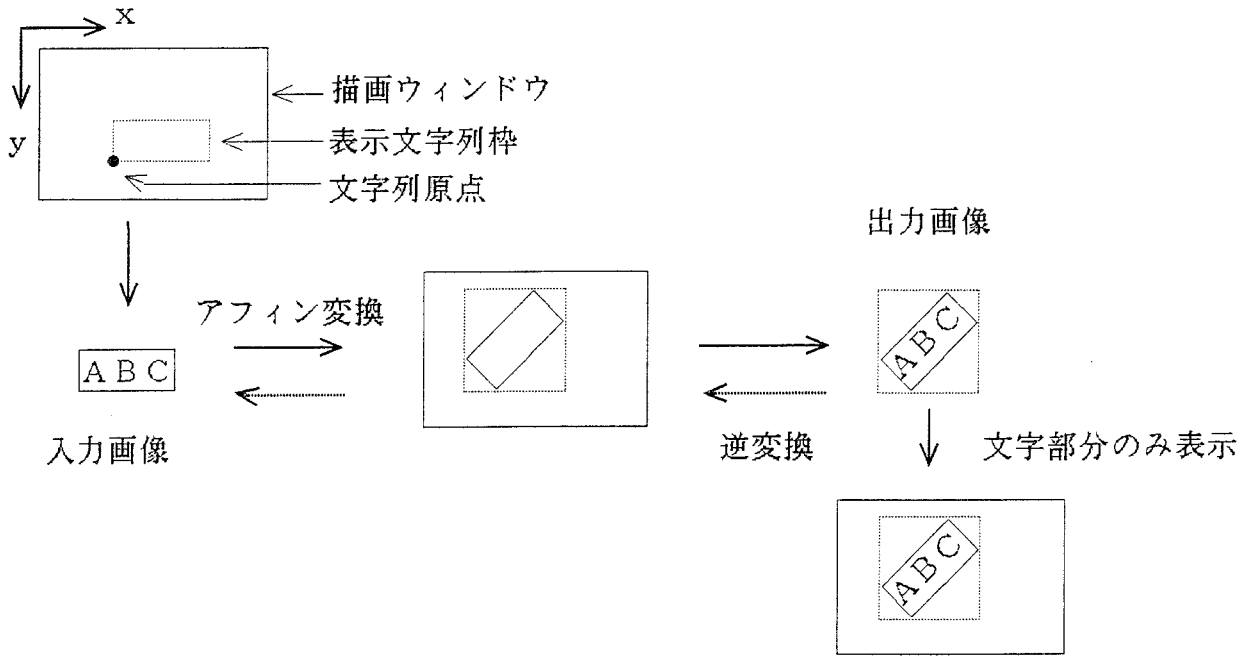

図1　変形操作の方法

は、スクリーン座標系においては時計回りが正となる。

3．変形操作の方法

変換操作の概要を図１に示す。

具体的な変形操作の手順は次のようになる。

(1) 表示文字列枠（対象文字列を含む最小の矩形）と同じサイズのメモリ領域を用意し、そこに、文字列を二値画像として書き込む。これを、ここでは入力画像と呼ぶ。入力画像は、画面に表示されず、目に見えない領域である。

(2) 表示文字列枠の４頂点に、変形操作と同じアフィン変換を施し、変換後の文字列全体を含む最小の矩形領域と同じサイズのメモリ領域を用意する。これを、出力画像と呼ぶことにする。出力画像も二値画像でよい。この時点で、出力画像は、白紙の状態である。

(3) 次に、出力画像に変形文字列の生成を行う。生成の仕方は変形操作の内容によって、以下の３通りに分ける。

① x、y両方向の縮小を同時に含む場合（いずれか一方のスケーリングファクタが１となる場合を含める。）

② x、y両方向の拡大を同時に含む場合（両方向ともにスケーリングファクタが1となる場合を含める。）

③ x、yいずれか一方が拡大で、一方が縮小の場合

①の場合は、入力画像を構成する各ピクセルに対して与えられた変形に対応するアフィン変換を施し、出力画像に書き込む。このときアフィン変換を施すピクセルは、文字の書かれたピクセルだけとすることにより、高速化を図る。なお、このような入力画像から出力画像への変換を、ここでは“順変換”と呼ぶことにする。

②の場合は、上のような順変換では、一般に変換後の文字に不要な隙間（ドット抜け）を生じ、文字品質が著しく劣化することがある。これを避けるため、①の場合とは逆方向の変換操作を行う。すなわち、出力画像を構成する各ピクセルに対し、本来適用すべきアフィン変換の逆変換を施し、対応する入力画像側のピクセルを求めこれを出力画像にマッピングする。このような逆方向の変換を単に“逆変換”と呼ぶことにする。

③の場合、①、②の複合体と考えられるため、まず順変換を行った後、さらに逆変換を施す。これは、いずれか一方の変換だけでは、文字品質に劣化が見られるためである。

(4) (3)で生成した出力画像を表示文字列として、アプリケーション画面の指定位置に表示する。この際、単純に出力すると出力画像の文字以外の背景部分もそのまま出力されてしまうため、アプリケーション画面の対応する部分が上書きされて消えてしまう。これを避けるために、出力画像を背景が透明な文字、すなわち透過型のマスクパターンとして、文字部分のみを出力する。

(3)における入力画像から出力画像へのアフィン変換は、スクリーン座標系において、変換前の点を(x , y)、変換後の点を(X , Y)、文字列原点を（x_t , y_t)、x、y方向のスケーリングファクタをそれぞれs_x、s_y、回転角をθとすると、次のようになる。

$$[X , Y , 1] = [x , y , 1]\, T_1 S R T_2 \quad \cdots (a)$$

ここで、

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_t & -y_t & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_t & y_t & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

また、出力画像から入力画像への逆変換は、同様に、次式となる。

$$[x , y , 1] = [X , Y , 1]\, T_1 R_r S_r T_2 \quad \cdots (b)$$

ここで、

$$S_r = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_r = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

なお(3)の場合分けは、実験の結果に基づいている。

x，y方向それぞれについて、縮小、等倍（1.0）、拡大の9通りの各組み合わせについて、順変換のみ、逆変換のみ、両変換併用の各ケースについて実際に文字列を生成して比較検討し、その中で変形後の文字列が最も元の文字形状を維持していると考えられるものを、その組み合わせの変形方法として採用した。場合分けはそれをまとめた結果である。

III 実装

上で述べた方法に基づき、X－Window上でC言語による実装を行った。実装は、Xlibとの併用を前提に、XlibのAPI形式に合わせたライブラリとして実現した。また1バイト系文字列と漢字データを対象とした。これらは、データのコード形式が異なるため、それぞれ独立したAPIに分けたが、実装の基本的な原理はほぼ同じである。以下では、主として1バイト系文字列について述べ、適宜、漢字対応について補足する。なお実装の環境は次のとおりである。

・X－Windowバージョン
　X11R4（OpenWindows 環境）
・使用マシン
　SUN4/2

1．入力画像と出力画像

入力画像と出力画像は、Pixmap構造体とXImage構造体で構成する。（以下、構造体は省略する。）Pixmapは、画面に表示されないメモリ上の領域である。

入力画像のサイズは、文字枠の大きさから求める。ここに、バックグランドを黒として、文字列を白で書き込む。文字枠のサイズは、XTextExtents 関数をもとに算出する。出力画像のサイズは、アフィン変換後の文字枠から計算する。入出力画像のサイズは、計算誤差による文字切れを防ぐため、実装においては、多少、大きめにとっておいた方が良い。なお漢字の場合、文字枠のサイズは、XTextExtents16関数を使用する。このため、漢字データのコード変換（EUCからJIS）が必要となる。また漢字の入力画像への書き込みはXDrawString16関数を使う。

このPixmapをピクセル単位の画像データとして操作するため、XImageに変換する。XImageへの変換は、XGetImage関数、ピクセル単位の操作は、XPut Pixel 関数、XGetPixel関数で行う。なお、入力画像領域および出力画像領域の座標系は、Pixmapに合わせる。すなわち、左上隅が原点、x軸は右向きが正、y軸は下向きが正となる。

2．出力画像の表示

出力画像をそのまま画面に表示すると、出力画像の下になるもともとの画面が上書きされ、消されてしまう。これを避けるために、XPutImage関数を使用せず、出力画像を透過型のスティプルパターンに変換し、このパターンを使って、画面上の出力画像領域をフォアグランドカラーで塗りつぶすという手順を取る。スティプルパターン化するには、出力画像のビットマップデータ（モノクロ二値画像）が必要となる。XlibではPixmapから直接Bitmapを生成する機能が提供されていない。（XCopyPlane 関数では、期待した結果が得られない。）しかしながら、幸いBitmapのフォーマットが公開されているので、変換ルーチンを作成することにより、出力画像のビットパターンから直接ビットマップデータを生成し、スティプルパターンとすることができる。なお、入力、出力画像のPixmapを最初からBitmapとして使用する方法も試みたが、これはXImage化する段階でエラーが発生した。

IV 実行例

以下実装したプログラムによる実行例を示す。実行例はいずれも画面のハードコピー出力である。なお、付録に、実装したライブラリの使用例として、以下の(2)の例のプログラムリストを示しておく。

(1) 拡大／縮小

図2に拡大／縮小の例を示す。数値は、それぞれx, y方向のスケーリングファクタを示している。

(2) 反転

図3に反転の例を示す。反転は負のスケーリングファクタを与える。ここでは、$s_x=-1.0$（y軸反転）と$s_y=-2.0$（x軸反転）の例を示した。

(3) 回転

$s_x=1.0$、$s_y=2.0$、45度単位の回転例を図4に示す。

(4) フォントを変えた例

図5にフォントの種類を変えた例を示す。$s_x=1.0$、$s_y=2.0$で、15度単位に回転を加えた。

(5) 図6に漢字の出力例を示す。

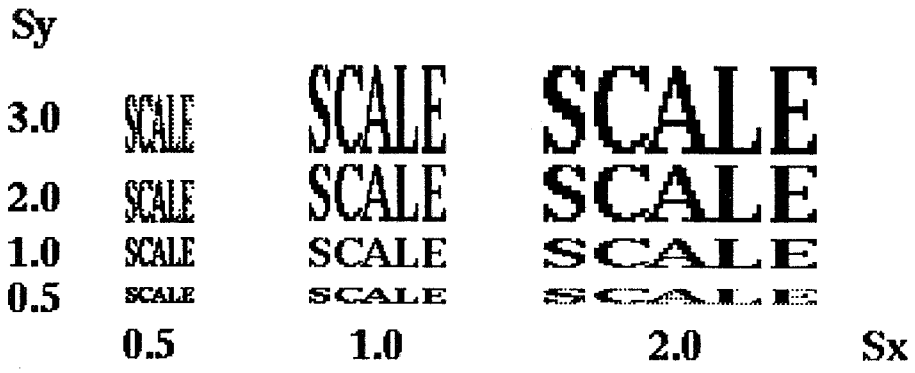

図２　拡大／縮小の例

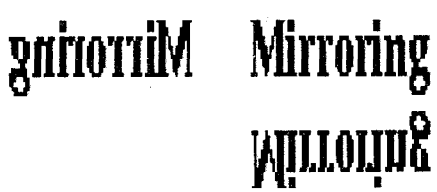

図３　反転の例

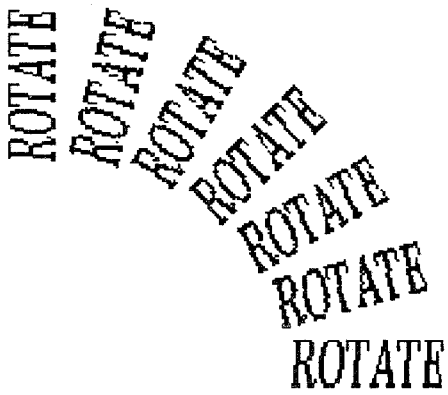

図５　フォントを変えた例

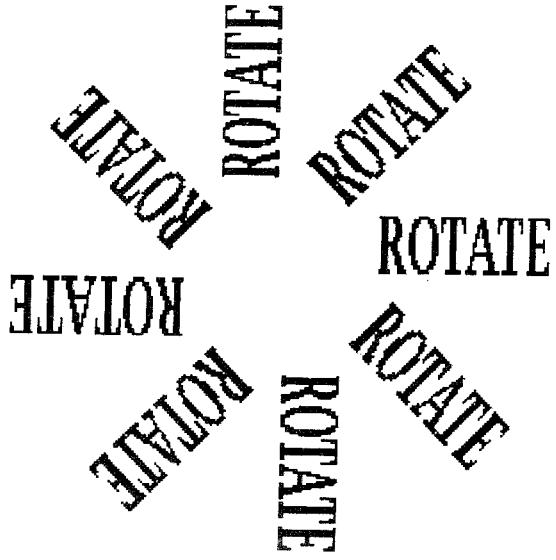

図４　回転の例

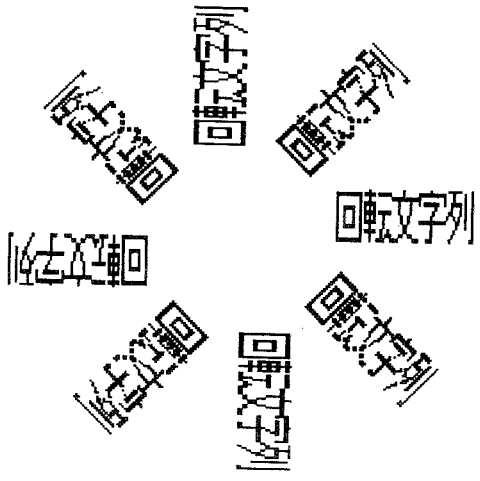

図６　漢字の例

V　検討

1．文字の品質

変形後の文字品質について評価する。評価の基準を、判読可能かどうか、元の文字列形状に近いかどうかに置く。

本手法との比較のため、図4と同じ例で、単純にアフィン変換のみを適用した場合の例を図7に示す。図7に比べ、図4では文字品質が改善されており、本手法の有効性が確認できる。ただし、本手法は、本質的には画像データの変形操作であるため、変形後の文字品質については自ずと限界がある点は否めない。変形操作の組み合わせによっては、単純な方法に比べ改善されてはいるものの、まだ若干の乱れが見られる場合がある。特にx方向の縮小とy方向の拡大を組み合わせた場合（図2のs_x=0.5、s_y=2.0，3.0）に顕著である。この場合、元となるフォントの品質によっては判読が難しくなる場合もある。これは、逆変換時の入力画像側の対応点が、必ずしも格子点にのらないために発生すると考えられる。通常の画像処理では、この点の濃度を、周囲の格子点の濃度を用いて線形補間法あるいは最近傍法によって決定する。文字画像の場合、線形補間は適さないと思われる。二値画像なので、中間的な値が意味を持たないからである。今回は最近傍法を用いたが、計算誤差による文字品質への影響という点で、まだ検討の余地が残る。しかしながら、実行例で示したように、実用上は、おおむね差し支えない程度の文字品質が得られていると考える。

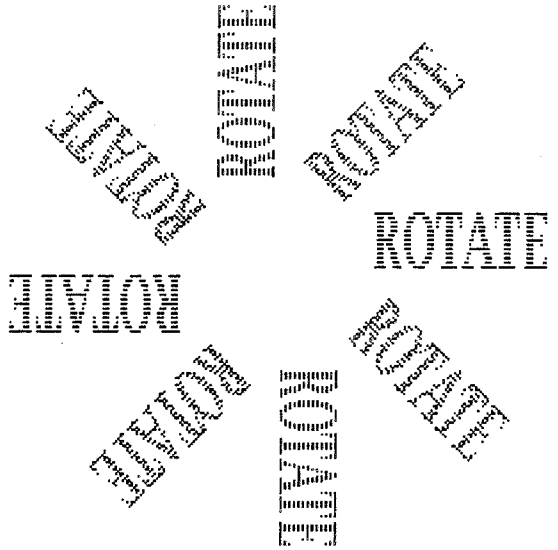

図7　順変換だけの例

2．処理速度

次に処理速度について考察する。実行例の処理時間（実行開始から描画終了までの時間）は、いずれも実測で2、3秒であった。これには、サーバとの接続時間なども含まれているので、実際の描画時間は、もっと少ないと思われる。また、文字列長12、x，y方向のスケーリングファクタを10、回転角45度とした（実用上ほとんど起こり得ないという意味で）過負荷なデータでの処理時間を測定してみると、約9秒であった。この程度であれば、グラフィックス処理などのように、文字を補助的に使用するアプリケーションであれば十分実用になると考えられるが、ここではさらに、処理速度の向上という観点から検討を加える。

本手法においては、入出力画像のピクセル単位の走査が必要になる。したがって、処理速度は、これらの画像の大きさに依存する。画像の大きさ自体は、文字列の長さ、拡大率、回転角に依存する。したがって、使用状況によっては、運用上、これらの点に配慮することが処理速度の向上につながる。

実装上は、処理対象となるピクセル数をなるべく減らすことも高速化に効果がある。本手法では、変形操作の内容により、変換方法の場合分けを行っているが、特に順変換が適用される場合には、画像の走査時に、文字データを構成するピクセルに対してのみ、変換処理を施すようにして高速化を図っている。今回の実装では、原理の確認が主な目的であるため、これ以上の考慮はしていないが、このほかにもアフィン変換の計算処理などにおいて、繰り返し処理内での計算量を減らすなどの実装上の工夫により、ある程度の処理速度の向上も可能と考える。

VI　おわりに

X－Windowのフォントデータを利用して、画像処理的な手法により変形文字列を生成するための一手法を考案した。またC言語での実装を行い、その実行例を示した。

この結果、文字品質、処理速度の点から見て、実用性のある手法であることが確認できた。また、ライブラリ形式で実装することにより、アフィン変換の知識なくして、簡単に変形文字列を生成できること、文字列単位に使用できること、既存の豊富なフォントデータをそのまま利用できることから、X－Window上におけるアプリケーション開発、特に図形処理分野での基礎技術として、実務分野で、あるいはまた教育訓練の分野においても広範囲な活用が期待できると考える。

【参考文献】
(1) 木下凌一、林秀幸、X-Window Ver.11 プログラミング、日刊工業新聞社、1993
(2) Oliver Jones、X Windowハンドブック，アスキー出版局、1992
(3) Adrian Nye,Xlib Reference Manual,O'Reilly & Associates,Inc、1990
(4) 遠藤知宏、Xウィンドウプログラミング8　フォント、bit、Vol.26 No.11、1994
(5) 林　秀幸、X−Window　OSF／MOTIFツールキットプログラミングII、日刊工業新聞社、1995
(6) 山口富士夫、コンピュータディスプレイによる図形処理工学、日刊工業新聞社、1983
(7) 八木伸行、井上誠喜他、C言語で学ぶ実践画像処理、オーム社、1992
(8) 柴山守、X11による画像処理基礎プログラミング、技術評論社、1994

[付録]

実装したライブラリの使用例として、Ⅳの(2)の反転の例のプログラムリストを図8に示す。図中、左端の数字は行番号である。

行3は、本ライブラリを使用するためのヘッダファイルのインクルード指示である。X_open関数およびX_close関数は、X−Windowを利用するための処理であり、本手法と本質的には関係がない。X_open関数内の関数とマクロおよび行17の関数は、XlibのAPIである。G_で始まる関数が本ライブラリのAPIであり、行20から行22のG_Xtext_af関数が、1バイト系文字列のための変形文字列生成用APIである。以下に仕様を示しておく。

```
void G_Xtext_af (d, w, gc, fs, x, y, text, x_scale,
                 y_scale, rot)
Display     *d;
  ディスプレイid
Window      w;
  ウィンドウid
GC          gc;
  グラフィックスコンテキスト
XFontStruct *fs;
  フォント情報。システムのデフォルトフォントを
  使う場合はＮＵＬＬを指定する。
int         x,y;
  文字列の原点位置（スクリーン座標系で指定）
char        *text;
  変形対象の文字列
float       x_scale,y_scale;
  x、y方向のスケーリングファクタ。反転の場合
  は、マイナス値を指定。
float       rot;
  回転角
```

```
1   #include <stdio.h>
2   #include <X11/Xlib.h>
3   #include "xlib_plus.h"
4
5   #define  TEXT "Mirroring"
6
7   Display *d;
8   Window  w;
9   GC      gc;
10  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++main*/
11  main()
12  {
13    XFontStruct *fs;
14
15    X_open(100,100,500,400);
16
17    fs = XLoadQueryFont(d,
18              "-adobe-times-bold-r-normal--18-180-75-75-p-99-iso8859-1");
19
20    G_Xtext_af(d,w,gc,fs,200,250,TEXT, 1.0, 2.0,0.0);
21    G_Xtext_af(d,w,gc,fs,200,270,TEXT, 1.0,-2.0,0.0);
22    G_Xtext_af(d,w,gc,fs,180,250,TEXT,-1.0, 2.0,0.0);
23
24    X_close();
25  }
26  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++X_open*/
27  X_open(x,y,width,height)
28  int x,y;
29  unsigned int width,height;
30  {
31    XSetWindowAttributes attr;
32    unsigned long        white,black;
33
34    d = XOpenDisplay(NULL);
35    white = WhitePixel(d,0);
36    black = BlackPixel(d,0);
37    w = XCreateSimpleWindow(d,RootWindow(d,0),x,y,width,height,1,black,white);
38    attr.override_redirect = True;
39    XChangeWindowAttributes(d,w,CWOverrideRedirect,&attr);
40    XMapWindow(d,w);
41    gc = XCreateGC(d,w,0,0);
42    XSetForeground(d,gc,black);
43    XFlush(d);
44  }
45  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++X_close*/
46  X_close()
47  {
48    printf("Type any character + return to quit.\n");
49    getchar();
50  }
```

図8 使用例