Xウィンドウ用グラフィックスライブラリ「xtgl」の作成

佐賀　孝博

1　はじめに

Xウィンドウ・システム（以下単にX）はUNIX上のウィンドウシステムとして標準的なものである。本学ではSUNマイクロシステムズ社（以下SUN社）のワークステーションを使用して、授業をおこなっており、SUN社の提供するOSを含めた統合環境であるSolarisもUNIXとXをベースとした環境になっている。

X上で動作するプログラムを作製する場合C言語を用いて、それにX用のプログラミングインターフェースであるXlibというXライブラリをリンクして使用するのが一般的である。Xで実行できる形式のプログラムを書くためにはある程度の知識を要求されるが、本学に入学したばかりの学生でも「グラフィックスを扱ったプログラムを作成してみたい」という希望は少なからずあった（ここでいうグラフィックスとは凝った3DCGなどではなく、「自分で作成したキャラクタが画面上で動くようなプログラムを作成してみたい」ということである）。そのような希望を持つ学生に対して、単に参考書を指定するだけでは知識は深まらず、プログラミングについてまだそれほど知識や経験がない学生に直接多くのことを教授しなければならない。しかし、直接教授したとしても、最初に行わなくてはならない「画面にグラフィックスを出す」というところまでの敷居が高いため、実際に自分の描いているプログラムを作る段階になる前に興味が失われるという事もあった。

Xlibをより扱い易くした「ツールキット」というものがあり、それを使用する事である程度Xのプログラミングは楽になる。しかし、ツールキットはアプリケーションプログラムに必須の機能（ボタンやメニューなど）を簡略して書くことはできても、グラフィックスに関してはXlibの機能を使用するという場合が多いため、上述のような「(簡単に) 画面にグラフィックス（キャラクタ）を出したい」という要望はかなえられにくい。

本学で過去にPCで授業を行っていたときには、PC上でBASICライクに使用できるグラフィックスライブラリがあり、それを使用して見栄えのするプログラムを作る学生もいた。そのような学生はそれを他に良い評価される事によって自信を持ち、さらにプログラミング技術が向上していった。

このように、初期にプログラミングに興味を持つ学生に対しては、比較的簡単にグラフィックス処理が出来るようなものが望まれていると思われる。今回開発したxtglはそのことを考慮しており、このライブラリを用いる事によりプログラミングへの関心も高まる事を期待している。

2　xtglとは

xtglとはX window system Tiny Graphics libraryの略であり、X上のライブラリXlibを利用したグラフィックス関数群である。なお、実際には音声などの扱いも含めて、SUN社のSolarisで使用されることを前提としている。

Xlibを用いればさまざまな事が実現できるが、それゆえにプログラミングをするのは簡単ではなく、自分で描いたキャラクタ（グラフィックス）を画面に出すだけでも少なからず知識が要求される。xtglでは

Xlib の知識がなくても画面上にグラフィックスを表示させたり、そのグラフィックスを移動させたりする事がなるべく簡単にできることを目標に開発された。

3 xtgl の特徴

xtgl は名前の通り、グラフィックスに関してそれほど多くの機能は持っていない。あくまでも、学生が手軽に使用できる事を重点において作成した。これ以上の機能が必要となるプログラムを作成しようという者は Xlib そのものを学習した方がいいというのが筆者の考えである。

なお、命令体系は一般的に使われている BASIC やパソコン用 C 言語のグラフィックスライブラリを参考にした。

xtgl の特徴としては以下のことが挙げられる。

- Xlib の細かい文法事項を知らなくても良い

  当然のことながら、xtgl でできる事はすべて Xlib でも実現可能であるが、なるべく手間を少なくして（その結果、制約も多くなってしまったが)、「自分がプログラムしたい事」と「実際に使用しなければならない関数」とを 1 対 1 の関係で実現できるように配慮した。

- BASIC のグラフィックス命令と似ている

  少し前まで個人でプログラムを作成するためのプログラミング言語は BASIC であった。そのため、なるべくその命令体系を崩さないようにした。また、個人用に販売されている C 言語のグラフィックスライブラリに関しても基本的に BASIC での命令体系と変わりはないようなので、もし、入学前にこれらの言語を学習していても違和感は少ないと思われる。

- 日本語を簡単に扱える

  Xlib に関しては日本語についても、扱いが面倒な面があったので、簡略化してフォントの設定なども事前にしてある（フォントの変更も可能)。また、自分の希望の位置に日本語が出力できるようになっている。

- エラーは日本語で出力（Warning 扱いとして処理を中断しない）

  エラーが発生した場合の処理としては、警告のみを出力し、処理は続ける方式を採用した。これは、グラフィックスを扱うプログラムの場合、「色を塗る範囲が間違っている」、「色のコード番号自体が不正だ」など、様々なエラーがでる可能性が高く、細かいエラーでプログラム全てが停止するのを避けるためである。グラフィックスが動くプログラムを作成するためにこのライブラリを使用するのであるから、ある程度おかしいところはプログラマが自分の目で見て、気づくはずである。このように自分で「このプログラムの動作は本当に正しいのか」ということをプログラムする本人が考える事は非常に重要なことである。そのような理由からプログラマ本人が気づけば直せるようなエラーは警告扱いとしてある。

- 音声を出力できる

  簡単な音量調節、と au 形式のファイルを音声として出力できるようにしてある。この機能をグラフィックス機能と組み合わせて使う事で、プログラミングの幅が広がると思われる。

- 乱数を簡単に使える

  初心者にとっては、「乱数」を使用するのも大変な事のようなので、これも事前に「初期化」した乱数を発生するようにした。

- 座標上の色のコードを調べる事ができる
  ゲームづくりをしようとした場合、ある物体との接触をチェックするプログラムが最も大変であるように思う。xtgl では、そのチェックをある程度簡単に行えるように、色コードを抽出できるようにしてある。

- キー入力やマウス入力を簡単に取り出すことができる
  キーボードからの入力やマウスからの入力をチェックする関数もあるので、これらの入力デバイスを利用したプログラムの作成も容易に行える。

以上のような特徴から、学生が自分でプログラムを作る際に最も動機付けしやすいと考えられる、グラフィックスを用いた単純なゲーム程度なら楽に作成する事ができる。

4 xtgl の仕様

xtgl の仕様は以下の通りである。

書式：
```
void ginit(window_name, win_width, win_height)
char *window_name;     /* ウィンドウのフレームにつく名前 */
int  win_width;        /* ウィンドウフレームの幅 */
int  win_height;       /* ウィンドウフレームの高さ */
```

機能：　グラフィックス関数群の初期化 & ウィンドウの表示
（X のウィンドウの生成、フォントのロードなど）
（音声出力の初期化、乱数の種生成）

文例：
```
ginit("サンプル", 640, 480);
```
フレーム名を「サンプル」とし、640x480 のウィンドウを生成
（&様々なものを初期化）

書式：
```
void gterm(void)
```

機能：　作成したウィンドウを終了する（使用終了）

文例：
```
gterm();
```

書式：
```
void gcolor(fore, back)
int fore;              /* フォアグラウンドの色番号 */
int back;              /* バックグラウンドの色番号 */
                       /* 色番号については、注1を参照 */
```

機能：　色の設定（あまり、使用する必要はないかも）

文例：
```
gcolor(GREEN, YELLOW);
```
フォアグラウンド「緑」、バックグラウンド「黄色」とする

書式：
```
void gpset(x, y, color)
int  x, y;             /* 座標 */
int  color;            /* 色番号 */
                       /* 色番号については、注1を参照 */
```

機能：　点を描画

文例：
```
gpset(200, 300, BLACK);
```
座標(200,300)に「黒」で点を描画

書式：
```
void gline(x1, y1, x2, y2, color, type)
int  x1, y1;          /* 描画始点の座標 */
int  x2, y2;          /* 描画終点の座標 */
int  color;           /* 色番号 */
                      /* 色番号については、注1を参照 */
int  type;            /* 描画タイプ */
                      /* 0(LINE) - 直線描画          */
                      /* 1(FILL) - 塗りつぶし矩形描画 */
                      /* 2(BOX)  - 矩形描画          */
```

機能：　直線（矩形）の描画

文例：
```
gline(300, 200, 350, 300, YELLOW, LINE);
```
(300,200)-(350,300) に「黄色」で直線を描画

```
gline(0, 0, 20, 20, RED, BOX);
```
(0, 0) - (20, 20) を対角線とする四角形を「赤」で描画

書式：
```
void gcircle(cx, cy, rx, ry, color, type)
int  cx, cy;          /* 円の中心座標 */
int  rx, ry;          /* 円の半径 */
int  color;           /* 色番号 */
                      /* 色番号については、 注1を参照 */
int  type;            /* 描画タイプ */
                      /* 0(NOMAL) - 円描画          */
                      /* 1(FILL)  - 塗りつぶし円描画 */
```

機能：　円の描画

文例：
```
gcircle(200, 200, 80, 100, GREEN, FILL);
```
(200,200) を中心としたX方向半径80、Y方向半径100 の円を「緑」で塗りつぶし描画

書式：
```
void gline_style(width, line_style, cap_style, join_style)
int  width;          /* 線の太さ */
int  line_style;     /* 破線の処理 */
                     /* 0(NOMAL) - 破線なし */
                     /* 1(DASH)  - 破線にする */
                     /* 詳しくは、注２を参照 */
                     /* デフォルト - NOMAL */
int  cap_style;      /* 線端の処理 */
                     /* 0(NOMAL)   - 線端の処理は特になし */
                     /* 1(OPTION1) - 線端をまるめる */
                     /* 2(OPTION2) - 線端を少しはみ出させる */
                     /* 詳しくは、注２を参照 */
                     /* デフォルト - NOMAL */
int  join_style;     /* 折れ曲がり部分の形状 */
                     /* 0(NOMAL)   - 外側の線を延長交差 */
                     /* 1(OPTION1) - 交差点をまるめる */
                     /* 2(OPTION2) - 外側のコーナーを直線で結ぶ */
                     /* 詳しくは、注２を参照 */
                     /* デフォルト - NOMAL */
```

機能：　ラインスタイルの変更

文例：
```
gline_style(20, DASH1, OPTION1, NOMAL);
```
次回、gline などを使用するときは、「太さ 20」、「破線」、
「線端はまるく」し、「折れ曲がり部分は、外側の線を延長交差」
するような、線になるように指定

書式：
```
void gprintf(x, y, color, format, ...)
int  x, y;           /* 出力座標 */
int  color;          /* 色番号 */
                     /* 色番号については、注１を参照 */
char *format;        /* フォーマットされた文字列 */
                     /* フォーマットは printf に準拠だが簡易版 */
```

機能：　画面に文字列を出力

文例：
```
gprintf(100, 100, RED, "%d 番は %s です", no, name);
```
座標 (100,100) から「1 番は 中森 です」など (int no, char *name) と表示

書式：
```
void gget(file_name, pixmap_name)
char *file_name;        /* Pixmap 形式のファイル名 */
                        /* Pixmap については、注 3 を参照 */
char *pixmap_name;      /* Pixmap につける名前 */
```

機能：　Pixmap 形式のファイルを Pixmap に変換する

文例：
```
gget("dog.xpm", "dog");
```
Pixmap 形式のファイル「dog.xpm」を Pixmap に変換して、
以後、「dog」で呼び出せる

書式：
```
void gput(pixmap_name, x, y, [END_ARG] or [x0, y0, width, height])
char *pixmap_name;      /* Pixmap についている名前 */
                        /* Pixmap については、注 3 を参照 */
int x, y;               /* 呼びだした Pixmap を表示する座標 */
int x0, y0;             /* Pixmap 左上点の座標 */
int width, height;      /* Pixmap 左上点からの幅と高さ */
```

機能：　名前のつけられている Pixmap を任意の大きさで表示

文例：
```
gput("dog", 50, 50, END_ARG);
```
Pixmap「dog」を (50,50) に表示
(引数の最後の END_ARG(-1) を忘れないように注意)

```
gput("dog", 50, 50, 0, 32, 64, 64);
```
Pixmap「dog」の (0,32)-(0+64,32+64) の範囲を、
(50,50) に表示

書式：　int gpoint(x, y)
　　int　x, y;　　/* 点座標 */

戻り値：　色番号　　/* 色番号については、注1を参照 */

機能：　ドットに対応する色番号の通知

文例：　color = gpoint(100, 100);
　　座標(100,100)の色番号を変数colorに代入

注意：　この関数は「gcls」「gline」「gcircle」など、色番号を指定して
　　グラフィックスを描いた色のみ有効

書式：　char *gpoint_rgb(x, y)
　　int　x, y;　　/* 点座標 */

戻り値：　カラーPixmap形式のRGB（文字列）値
　　/* Pixmapについては、注3を参照 */

機能：　ドットに対応するカラーPixmapのRGB（文字列）値の通知

文例：　if (strcmp(gpoint_rgb(100, 100), "#FFFFFFFFFFFF") == 0)　exit(1);
　　座標(100,100)の色番号が"#FFFFFFFFFFFF"（黒）なら、プログラム終了

注意：　この関数は「gget」を用いて、グラフィックスを描いた色のみ有効
　　RGB値については、カラーPixmap形式ファイルの3行目以降を参照

書式：　int gkey(void)

戻り値：　キーボードの文字
　　エラーのときには、-1(FALSE)を返す

機能：　入力された文字を検出する

文例：　if (gkey() == 'q') exit(1);
　　「q」キーが押されたら、プログラムを終了する

書式：
```
struct mouse_point gmouse(void)
struct mouse_point {
        int button;     /* マウスボタン */
                        /* 1(MOUSE_LEFT)   - 左ボタン */
                        /* 2(MOUSE_MIDDLE) - 中ボタン */
                        /* 3(MOUSE_RIGHT)  - 右ボタン */
        int x, y;       /* 座標 */
};
```

戻り値：押されたときのボタンと座標
エラーのときにはすべてに 、-1(FALSE) をいれて返す

機能：マウスのどのボタンがどの座標で押されたかを検出する

文例：
```
(struct mouse_point)point = gmouse();
if (point.button==MOUSE_LEFT && point.x==0 && point.y==0)
    exit(1);
```
マウスの左ボタンが (0,0) のところで押されたら、プログラムを終了

書式：
```
void audio_play(file_name)
char *file_name;        /* 「音声形式」のオーディオファイル名 */
```

機能：「音声形式」のオーディオファイルを再生する

文例：
```
audio_play("song.au");
```
「音声形式」のオーディオファイル「song.au」を再生

書式：
```
void audio_volume(volume)
int  volume;            /* 音量の値 */
```

機能：audio_play() の音量の調節
デフォルトは 20

文例：
```
audio_volume(100);
```
audio_play() の音量を 100 に設定する

書式：　int rnd(range)
　　int　range;　　/* 乱数の発生する範囲 */

戻り値：　発生した乱数の値

機能：　0～(range-1) の範囲で乱数を発生させる

文例：　printf("%d\n", rnd(100));
　　0～99 の間の数値が表示される

5 xtgl を利用してみて

xtgl を利用した場合のソースコードの長さは Xlib そのものを利用した場合に比べてかなり短くなる。これは、プログラムの流れを追う上でも重要なことである。また、「させたい命令」と「関数」がほぼ1対1で対応しているため、プログラムの初心者でも戸惑うことは少ないと思われる。また、Xウィンドウのプログラムを作成する場合、命令のどこからどこまでを繰り返すかを意識してループをつくらなくてはいけないが、xtgl の場合はそのループを気にしなくてもよいのでプログラミングしやすくなる。

実際に xtgl を使用したサンプルを例示してみる。

```
#include <stdio.h>
#include "fg.h"       /* xtgl のインクルードファイル。必須 */

int main(void)
{
    struct mouse_point mouse;
    int x, y, oldx, oldy;
    int key, end_flag=0;
    x=250; y=200;

    ginit("サンプル", 320, 240);  /* xtgl を使用。必須 */

    gget("man.xpm", "man");      /* ピックスマップデータを読み込み */
    gget("erase.xpm", "kesu");

    gcls(WHITE);                  /* 白で画面クリア */

    audio_play("go.au");          /* go.au をオーディオ出力 */
    gprintf(150, 100, BLUE, "ＧＯ!!");
                                  /* ＧＯ!! を表示 */
    sleep(1);
    gcls(WHITE);

    gcircle(rnd(200), rnd(150), 20, 20, BLACK, FILL);
                                 /* ランダムな中心座標に黒い●を表示 */

    oldx=x; oldy=y;
    gput("man", x, y, END_ARG); /* 読み込んだピックスマップデータを表示 */

    while (1) {
        key = 1;
        mouse = gmouse();        /* マウスを使用 */
```

＜次ページへ続く＞

＜前ページの続き＞

```
        switch(mouse.button) {    /* マウスボタンが押されたときの処理 */
        case MOUSE_LEFT:          /* 左ボタンの場合 */
        case MOUSE_RIGHT:         /* 右ボタンの場合 */
            end_flag = 1;
            break;
        }
        switch (gkey()) {         /* キー入力のチェック */
        case 'h':                 /* 左に移動するときは [h] キー */
            x-=10;  break;
        case 'l':                 /* 右に移動するときは [l] キー */
            x+=10;  break;
        case 'k':                 /* 上に移動するときは [k] キー */
            y-=10;  break;
        case 'j':                 /* 下に移動するときは [j] キー */
            y+=10;  break;
        default:                  /* その他はキーが押されてないとする */
            key=0;  break;
        }

        if ( gpoint(x, y) == BLACK ) {
                                  /* 自キャラクタが「黒」の座標と一致したら終わり*/
            gput("kesu", oldx, oldy, END_ARG);
            gput("man", x, y, END_ARG);
            break;
        } else if ( end_flag == 1) {
                                  /* マウスのボタンが押されたら終わり*/
            break;
        }

        if (key == 1) {
            gput("kesu", oldx, oldy, END_ARG);
            gput("man", x, y, END_ARG);
            oldx=x; oldy=y;
        }
    }

    gprintf(125, 100, RED, "おしまい!!");
    sleep(3);
    gterm();                      /* xtgl 終了。必須 */
}
```

これは自キャラクタを上下左右に動かして（図１）、●にぶつかると終わりになる（図２）というプログ

ラムであるが、Xlib でプログラミングするよりはかなり短くなっている。

また、このプログラムでは、自キャラクタの動作時の「ちらつき」を回避したりしているので、多少無駄の多いプログラムになってはいるが、プログラムの流れを追う事は容易だと思われる。

図1

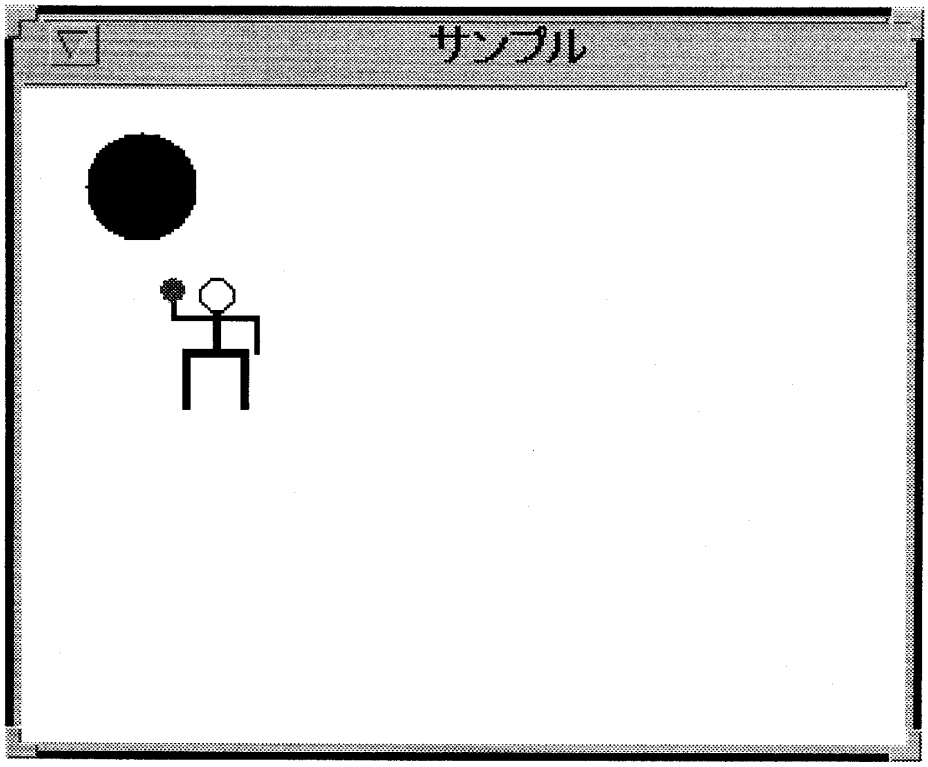
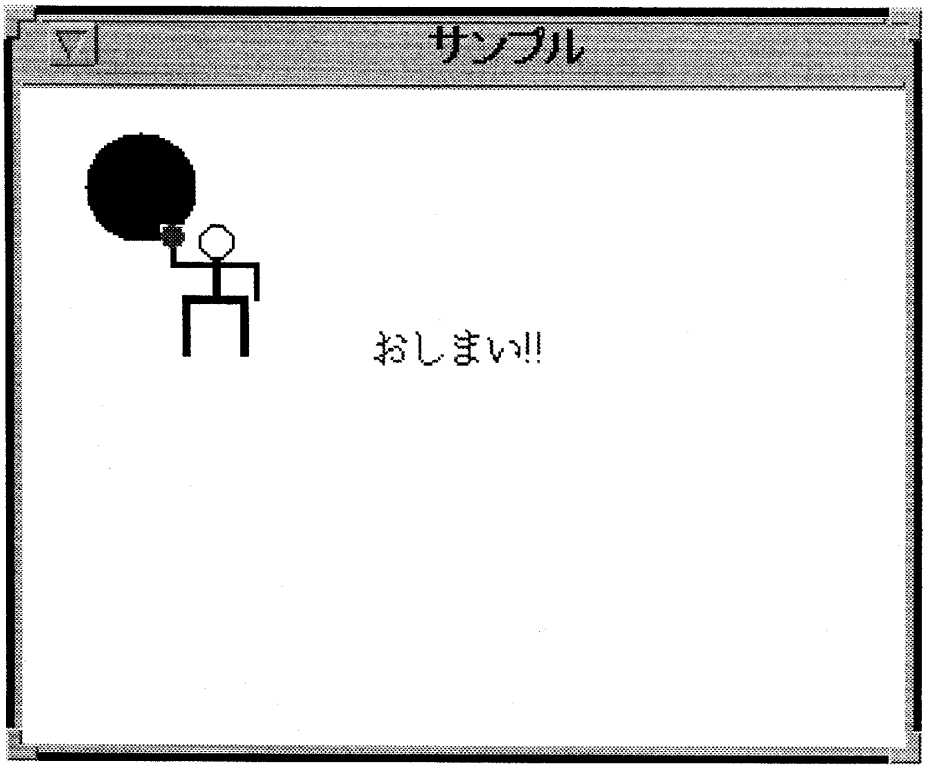


図2



6 xtgl の課題

- スプライト（グラフィックスの重ね合わせ）機能がない。
- グラフィックスの数、形式が制限されている。

以上のような問題点があり、本格的なゲームやアプリケーションを作成するのは困難だと思われる。

7 おわりに

xtgl を用いてプログラミングする事により、比較的簡単にグラフィックスを使用した「動くプログラム」を作成できるので、プログラミングの初心者であっても興味を持ってプログラムする事が可能となった。このことは、プログラミングの動機付けとして有効であると考える。ただ、このライブラリを導入以上に本格的に使用しようとすると、少なからず問題点が出ると思われるので、今回のライブラリを実際に学生に使用・評価してもらい、より扱い易いライブラリになるよう改善を計りたい。

8 注意

8.1 注1（色番号）

色は数値として扱う (0〜436) が、define された文字列で扱うことも可能

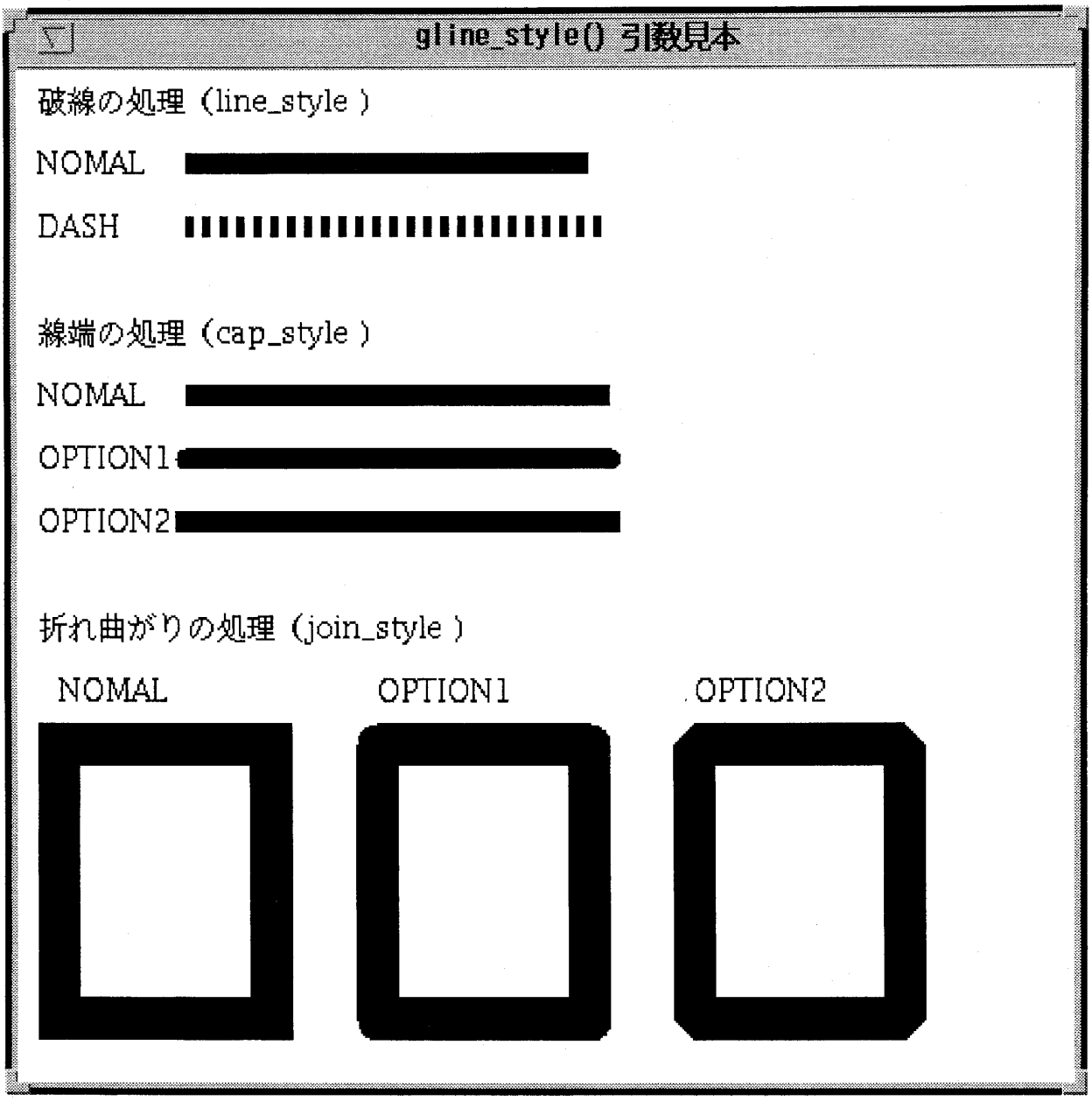
```
SNOW, GHOST_WHITE, WHITE_SMOKE, GAINSBORO, FLORAL_WHITE,
OLD_LACE, LINEN, ANTIQUE_WHITE, PAPAYA_WHIP, BLANCHED_ALMOND,
BISQUE, PEACH_PUFF, NAVAJO_WHITE, MOCCASIN, CORNSILK,
IVORY, LEMON_CHIFFON, SEASHELL, HONEYDEW, MINT_CREAM,
AZURE, ALICE_BLUE, LAVENDER, LAVENDER_BLUSH, MISTY_ROSE,
WHITE, ROYAL_BLUE, BLUE, DODGER_BLUE, DEEP_SKY_BLUE,
SKY_BLUE, LIGHT_SKY_BLUE, STEEL_BLUE, LIGHT_STEEL_BLUE, LIGHT_BLUE,
POWDER_BLUE, PALE_TURQUOISE, DARK_TURQUOISE, TURQUOISE, CYAN,
LIGHT_CYAN, CADET_BLUE, MEDIUM_AQUAMARINE, AQUAMARINE, DARK_GREEN,
DARK_OLIVE_GREEN, DARK_SEA_GREEN, DARK_GOLDENROD, BLACK, DARK_SLATE_GRAY,
DIM_GRAY, SLATE_GRAY, GRAY, LIGHT_GRAY, MIDNIGHT_BLUE,
NAVY, NAVY_BLUE, CORNFLOWER_BLUE, DARK_SLATE_BLUE, SLATE_BLUE,
MEDIUM_SLATE_BLUE, LIGHT_SLATE_BLUE, MEDIUM_BLUE, SEA_GREEN, MEDIUM_SEA_GREEN,
LIGHT_SEA_GREEN, PALE_GREEN, LAWN_GREEN, GREEN, CHARTREUSE,
MEDIUM_SPRING_GREEN, GREEN_YELLOW, LIME_GREEN, YELLOW_GREEN, FOREST_GREEN,
OLIVE_DRAB, DARK_KHAKI, KHAKI, PALE_GOLDENROD, LIGHT_GOLDENROD_YELLOW,
LIGHT_YELLOW, YELLOW, GOLD, LIGHT_GOLDENROD, GOLDENROD,
PALE_VIOLET_RED, MAROON, MEDIUM_VIOLET_RED, VIOLET_RED, MAGENTA,
VIOLET, PLUM, ORCHID, MEDIUM_ORCHID, DARK_ORCHID,
DARK_VIOLET, BLUE_VIOLET, PURPLE, MEDIUM_PURPLE, THISTLE,
SNOW1, SNOW2, SNOW3, SNOW4, SEASHELL1,
SEASHELL2, SEASHELL3, SEASHELL4, ANTIQUE_WHITE1, ANTIQUE_WHITE2,
ANTIQUE_WHITE3, ANTIQUE_WHITE4, BISQUE1, BISQUE2, BISQUE3,
BISQUE4, ROYAL_BLUE1, ROYAL_BLUE2, ROYAL_BLUE3, ROYAL_BLUE4,
```

BISQUE4, ROYAL_BLUE1, ROYAL_BLUE2, ROYAL_BLUE3, ROYAL_BLUE4,
BLUE1, BLUE2, BLUE3, BLUE4, DODGER_BLUE1,
DODGER_BLUE2, DODGER_BLUE3, DODGER_BLUE4, STEEL_BLUE1, STEEL_BLUE2,
STEEL_BLUE3, STEEL_BLUE4, DEEP_SKY_BLUE1, DEEP_SKY_BLUE2, DEEP_SKY_BLUE3,
DEEP_SKY_BLUE4, SKY_BLUE1, SKY_BLUE2, SKY_BLUE3, SKY_BLUE4,
LIGHT_SKY_BLUE1, LIGHT_SKY_BLUE2, LIGHT_SKY_BLUE3, LIGHT_SKY_BLUE4, SLATE_GRAY1,
SLATE_GRAY2, SLATE_GRAY3, SLATE_GRAY4, LIGHT_STEEL_BLUE1, LIGHT_STEEL_BLUE2,
LIGHT_STEEL_BLUE3, LIGHT_STEEL_BLUE4, LIGHT_BLUE1, LIGHT_BLUE2, LIGHT_BLUE3,
LIGHT_BLUE4, SPRING_GREEN1, SPRING_GREEN2, SPRING_GREEN3, SPRING_GREEN4,
GREEN1, GREEN2, GREEN3, GREEN4, CHARTREUSE1,
CHARTREUSE2, CHARTREUSE3, CHARTREUSE4, OLIVE_DRAB1, OLIVE_DRAB2,
OLIVE_DRAB3, OLIVE_DRAB4, DARK_OLIVE_GREEN1, DARK_OLIVE_GREEN2, DARK_OLIVE_GREEN3,
DARK_OLIVE_GREEN4, KHAKI1, KHAKI2, KHAKI3, KHAKI4,
LIGHT_GOLDENROD1, LIGHT_GOLDENROD2, LIGHT_GOLDENROD3, LIGHT_GOLDENROD4, LIGHT_YELLOW1,
LIGHT_YELLOW2, LIGHT_YELLOW3, LIGHT_YELLOW4, YELLOW1, YELLOW2,
YELLOW3, YELLOW4, GOLD1, GOLD2, GOLD3,
GOLD4, BROWN1, BROWN2, BROWN3, BROWN4,
ROSY_BROWN, INDIAN_RED, SADDLE_BROWN, SIENNA, PERU,
BURLYWOOD, SANDY_BROWN, TAN, CHOCOLATE, FIREBRICK,
BROWN, DARK_SALMON, SALMON, LIGHT_SALMON, ORANGE,
DARK_ORANGE, CORAL, LIGHT_CORAL, TOMATO, ORANGE_RED,
RED, HOT_PINK, DEEP_PINK, PINK, LIGHT_PINK,
PEACH_PUFF1, PEACH_PUFF2, PEACH_PUFF3, PEACH_PUFF4, NAVAJO_WHITE1,
NAVAJO_WHITE2, NAVAJO_WHITE3, NAVAJO_WHITE4, LEMON_CHIFFON1, LEMON_CHIFFON2,
LEMON_CHIFFON3, LEMON_CHIFFON4, CORNSILK1, CORNSILK2, CORNSILK3,
CORNSILK4, IVORY1, IVORY2, IVORY3, IVORY4,
HONEYDEW1, HONEYDEW2, HONEYDEW3, HONEYDEW4, LAVENDER_BLUSH1,
LAVENDER_BLUSH2, LAVENDER_BLUSH3, LAVENDER_BLUSH4, MISTY_ROSE1, MISTY_ROSE2,
MISTY_ROSE3, MISTY_ROSE4, AZURE1, AZURE2, AZURE3,
AZURE4, SLATE_BLUE1, SLATE_BLUE2, SLATE_BLUE3, SLATE_BLUE4,
LIGHT_CYAN1, LIGHT_CYAN2, LIGHT_CYAN3, LIGHT_CYAN4, PALE_TURQUOISE1,
PALE_TURQUOISE2, PALE_TURQUOISE3, PALE_TURQUOISE4, CADET_BLUE1, CADET_BLUE2,
CADET_BLUE3, CADET_BLUE4, TURQUOISE1, TURQUOISE2, TURQUOISE3,
TURQUOISE4, CYAN1, CYAN2, CYAN3, CYAN4,
DARK_SLATE_GRAY1, DARK_SLATE_GRAY2, DARK_SLATE_GRAY3, DARK_SLATE_GRAY4, AQUAMARINE1,
AQUAMARINE2, AQUAMARINE3, AQUAMARINE4, DARK_SEA_GREEN1, DARK_SEA_GREEN2,
DARK_SEA_GREEN3, DARK_SEA_GREEN4, SEA_GREEN1, SEA_GREEN2, SEA_GREEN3,
SEA_GREEN4, PALE_GREEN1, PALE_GREEN2, PALE_GREEN3, PALE_GREEN4,
GOLDENROD1, GOLDENROD2, GOLDENROD3, GOLDENROD4, DARK_GOLDENROD1,
DARK_GOLDENROD2, DARK_GOLDENROD3, DARK_GOLDENROD4, ROSY_BROWN1, ROSY_BROWN2,
ROSY_BROWN3, ROSY_BROWN4, INDIAN_RED1, INDIAN_RED2, INDIAN_RED3,
INDIAN_RED4, SIENNA1, SIENNA2, SIENNA3, SIENNA4,
BURLYWOOD1, BURLYWOOD2, BURLYWOOD3, BURLYWOOD4, WHEAT1,
WHEAT2, WHEAT3, WHEAT4, TAN1, TAN2,

TAN3, TAN4, CHOCOLATE1, CHOCOLATE2, CHOCOLATE3,
CHOCOLATE4, FIREBRICK1, FIREBRICK2, FIREBRICK3, FIREBRICK4,
HOT_PINK1, HOT_PINK2, HOT_PINK3, HOT_PINK4, SALMON1,
SALMON2, SALMON3, SALMON4, LIGHT_SALMON1, LIGHT_SALMON2,
LIGHT_SALMON3, LIGHT_SALMON4, ORANGE1, ORANGE2, ORANGE3,
ORANGE4, DARK_ORANGE1, DARK_ORANGE2, DARK_ORANGE3, DARK_ORANGE4,
CORAL1, CORAL2, CORAL3, CORAL4, TOMATO1,
TOMATO2, TOMATO3, TOMATO4, ORANGE_RED1, ORANGE_RED2,
ORANGE_RED3, ORANGE_RED4, RED1, RED2, RED3,
RED4, DEEP_PINK1, DEEP_PINK2, DEEP_PINK3, DEEP_PINK4,
DARK_ORCHID1, DARK_ORCHID2, DARK_ORCHID3, DARK_ORCHID4, PINK1,
PINK2, PINK3, PINK4, LIGHT_PINK1, LIGHT_PINK2,
LIGHT_PINK3, LIGHT_PINK4, PALE_VIOLET_RED1, PALE_VIOLET_RED2, PALE_VIOLET_RED3,
PALE_VIOLET_RED4, MAROON1, MAROON2, MAROON3, MAROON4,
VIOLET_RED1, VIOLET_RED2, VIOLET_RED3, VIOLET_RED4, MAGENTA1,
MAGENTA2, MAGENTA3, MAGENTA4, ORCHID1, ORCHID2,
ORCHID3, ORCHID4, PLUM1, PLUM2, PLUM3,
PLUM4, MEDIUM_ORCHID1, MEDIUM_ORCHID2, MEDIUM_ORCHID3, MEDIUM_ORCHID4,
PURPLE1, PURPLE2, PURPLE3, PURPLE4, MEDIUM_PURPLE1,
MEDIUM_PURPLE2, MEDIUM_PURPLE3, MEDIUM_PURPLE4, THISTLE1, THISTLE2,
THISTLE3, THISTLE4, , COLOR_NUMBER,

8.2 注 2 （gline_style()）

具体的な図は以下の通り



8.3 注3（Pixmap形式ファイル）

具体的なファイル内容は以下の通り

```
! XPM2
16 16 8 1
  c #000000000000
. c #E5E5BEBE5B5B
X c #26264C4CBFBF
o c #B7B75B5BE5E5
O c #DFDFE5E58989
+ c #E5E55B5B5B5B
@ c #2D2DE5E55B5B
# c #FFFFFFFFFFFF

 ................
 .XXXXXXXXXXXX.
 .Xoooooooooo.
 .XoOOOOOOOOoX.
 .XoO++++++OoX.
 .XoO+@@@@+OoX.
 .XoO+@##@+OoX.
 .XoO+@##@+OoX.
 .XoO+@@@@+OoX.
 .XoO++++++OoX.
 .XoOOOOOOOOoX.
 .Xoooooooooo.
 .XXXXXXXXXXX.
 ................
```

現在のXでは色の指定は、色コードではなく色名で指定するのが標準で、データ構造自体ももインクルードするような形式になっているが、Solaris に付属のキャラクタデザインツール「Icon Editor」が上記のような、形式で保存するのでそれに合わせたデータ構造を解釈するようにしてある。

9 参考文献

「X-Window Ver.11 プログラミング [第2版]」木下凌一・林秀幸著　日刊工業新聞社刊

「Xlib プログラミング・マニュアル 日本語版」Adrian Nye 著　監訳　坂下秀、荒井美千子、西垣内昌喜、藤井裕史　ソフトバンク刊

「Xlib リファレンス・マニュアル 日本語版」Adrian Nye 著　監訳　坂下秀　ソフトバンク刊