Handy Graphic ユーザーズガイド

for Handy Graphic Version 0.5 2008-06-09 荻原剛志

1 Handy Graphic とは何か

Handy Graphic とは、簡単な作図を行うための機能を C プログラムから使えるようにまとめたものです。プログラムから使うための機能のまとまりを**ライブラリ**と呼びますが、Handy Graphic は作図を行うためのライブラリですので**グラフィックライブラリ**と呼ばれます。

Handy Graphic を使ったプログラムでは、画面上にウィンドウを開いて、自分の思うように直線や円、長方形を描くことができます。単純な機能しか持っていませんが、逆に、覚えることは少なくなっています。これらの機能を組み合わせて幾何学的な模様を描いたり、簡単なグラフを描いたりすることができます。パソコンや携帯電話のゲームのような高度で複雑な機能は持っていませんが、工夫次第では簡単なゲームやパズルも作成できるでしょう。

Mac OS X で Handy Graphic を使ったプログラムで図形を表示するためには、HgDisplayer というアプリケーションを使用する必要があります。Handy Graphic および HgDisplayer のインストールは、パッケージファイルをダブルクリックしてインストーラを起動するだけで行えます。

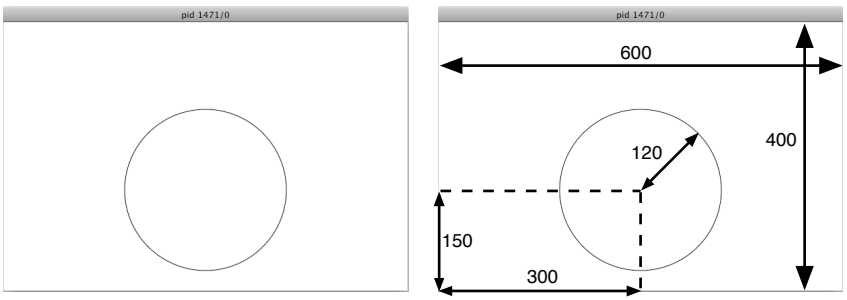
2 Handy Graphic を使ったプログラム例

図 1 は Handy Graphic を使って円を描くプログラムです。

```
#include <stdio.h>
#include <handy.h>                  /* Handy Graphic を使うために必要 */

int main(void)
{
    HgOpen(600, 400);               /* 描画用ウィンドウを開く */
    HgCircle(300, 150, 120);        /* 中心 (300, 150)、半径 120 の円を描く */
    getchar();                      /* プログラムを待たせる */
    return 0;                       /* 改行キーを入力すると終了する。 */
}
```

図 1: 円を描くプログラム



(a) ウィンドウと描かれた図形　　(b) 図形と座標の関係

図 2: 円を描くプログラムの実行結果

このプログラムはまず、画面に横 600 画素、縦 400 画素の大きさの表示領域を持つウィンドウを開きます。この表示領域の左下を座標の原点と考え、(300, 150) の点を中心とした半径 120 の円を描きます。それだけではプログラムがすぐに終了してしまうため、getchar() を呼び出してターミナルからの入力を待ちます。描画した結果は図 2 のようになります。

プログラムで図形を描けますので、繰り返したり大きさを変化させたりもできます。図 3 のプログラムは図 4 のように、いくつもの長方形を描きます。HgBox(x, y, w, h) は、左下が (x, y), 幅 w, 高さ h の長方形を描く関数です。

```
#include <stdio.h>
#include <handy.h>

int main(void)
{
    int i;
    double w, h;

    HgOpen(500, 400);
    w = 100;
    h = 40;
    for (i = 1; i <= 15; i++) {
        w -= 4;
        h += 4;
        HgBox(i * 28, i * 20, w, h);
    }
    getchar();
    return 0;
}
```

図 3: 長方形を描くプログラム

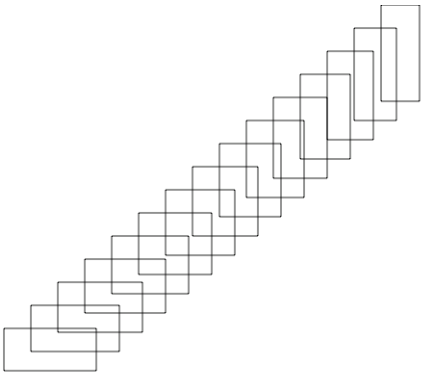


図 4: 長方形を描くプログラムの実行結果

3 コンパイルと実行の方法

図 1 や図 3 のようなプログラムをテキストエディタで作成します。ここでは draw.c という名前のファイルだとして説明します。

Handy Graphic がインストールされていれば hgcc というコマンドでコンパイルできます（以下の説明で冒頭の $ はターミナルのプロンプトです)。次のコマンドでは実行ファイルとして a.out が生成されます。

```
$ hgcc draw.c
```

次のように、cc（あるいは gcc）と同様なオプションが指定できます。

```
$ hgcc -Wall -o draw draw.c
```

プログラムの実行は通常と同じです。すなわち、実行ファイル名が draw ならばターミナルから次のコマンドを入力します。

```
$ ./draw
```

このプログラムを実行する前に、HgDisplayer を起動しておいた方がよいでしょう。以前に HgDisplayer を使ったことがあれば、ターミナルからプログラムを実行すると自動的に HgDisplayer が起動しますが、多少時間がかかることがあります。

HgDisplayer は自作のプログラムを終了させるたびに終了させる必要はなく、いったん起動したらそのままにしておいて構いません。また、ターミナルでプログラムを動かしたまま HgDisplayer を終了させるとエラーとなります。

HgDisplayer は、Handy Graphic を使った複数のプログラムの表示を同時に行うことができます。

4 描画のための設定

4.1 ウィンドウの作成

```
int HgOpen(double w, double h)
```
引数: w,h: 幅と高さ
返り値: 0: 正常、 -1: 異常

描画を行うウィンドウを作成し、画面の中央に表示します。引数 w, h は、ウィンドウ内の描画可能な領域の幅と高さを指定します。

なお、ファイルを保存する関数などを除き、ほとんどの関数はめったに異常値を返しません。従って、これ以降の大部分の関数については、返り値が正常かどうかをチェックしなくても問題はありません。

4.2 線の太さ

何も指定しない場合、図形の線の太さは 1 画素分ですが、これを変更することができます。いったん太さを変更すると、別の指定があるまではその値が使われます。

```
int HgSetWidth(double t)
```
引数: t: 線の太さ
返り値: 0: 正常、 -1: 異常

線の太さの指定は、直線、円、長方形などの図形に共通して有効です。

4.3 図形の色の指定

図形を描くのに使われる色を指定することができます。色は hgcolor 型というデータ型で指定しますが、よく使われる色はマクロで定義されています（表 1)。

表 1: 色のマクロ定義

色	マクロ名	色	マクロ名	色	マクロ名
白	HG_WHITE	赤	HG_RED	シアン	HG_CYAN
黒	HG_BLACK	緑	HG_GREEN	オレンジ	HG_ORANGE
灰色	HG_GRAY	青	HG_BLUE	ピンク	HG_PINK
淡灰色	HG_LGRAY	黄	HG_YELLOW	マゼンタ	HG_MAGENTA
濃灰色	HG_DGRAY	紫	HG_PURPLE	茶	HG_BROWN

描かれる線の色を指定するには次の関数を使います。いったん色を指定すると、別の色を指定するまで同じ色が使われます。一度も指定しない場合は黒で描かれます。

```
int HgSetColor(hgcolor clr)
```
引数:　clr:色の指定
返り値:　0:　正常、　-1:　異常

下で説明する円や長方形では内部を塗りつぶすことができます。このような図形を**塗りつぶし図形**と呼びますが、塗りつぶしに使う色は線を描くための色とは別に、次の関数を使って指定します。いったん色を指定すると、別の色を指定するまで同じ色が使われます。一度も指定しない場合は白で塗りつぶされます。

```
int HgSetPaintColor(hgcolor clr)
```
引数:　clr:色の指定
返り値:　0:　正常、　-1:　異常

4.4　文字コードとフォントの指定

日本語を含む文字列を描画する場合、ソースプログラムの日本語がどんな文字コードで表現されているのかを HgDisplayer に伝える必要があります。そのために次の関数を使います。引数には表 2 のマクロのいずれかを指定します。

```
int HgEncoding(int coding)
```
引数:　coding:　文字コードを指定する整数値
返り値:　0:　正常、　-1:　異常

文字列を描画する場合にフォント（字体）とその大きさを指定できます。フォント名には表 3 のマクロのいずれかを指定します。いったん指定すると、別の指定があるまで同じ字体と大きさが使われます。

```
int HgSetFont(hgfont font, double size)
```
引数:　font:　フォントを指定する定数名　size:　文字サイズ
返り値:　0:　正常、　-1:　異常

表 2: 文字コードのマクロ定義

マクロ名	文字コード
HG_UTF8_CODE	UTF-8
HG_JIS_CODE	JIS
HG_SJIS_CODE	シフト JIS
HG_EUCJP_CODE	日本語 EUC

表 3: フォントのマクロ定義

フォント	細字体	斜体	太字体	太字斜体
Times	HG_T	HG_TI	HG_TB	HG_TBI
Helvetica	HG_H	HG_HI	HG_HB	HG_HBI
Courier	HG_C	HG_CI	HG_CB	HG_CBI
明朝	HG_M		HG_MB	
ゴシック	HG_G		HG_GB	

5　主な描画関数

5.1　直線

関数 HgLine() は座標 (x_0, y_0) と (x_1, y_1) を結ぶ線分を描きます。これらの点はウィンドウの外部の点でも構いません。

```
int HgLine(double x0, double y0, double x1, double y1)
引数:  x0,y0:  線分の始点  x1,y1:  線分の終点
返り値:  0:  正常、  -1:  異常
```

5.2 円

円周だけを描く関数と、塗りつぶした円を描く関数があります。

```
int HgCircle(double x, double y, double r)
引数:  x,y:  円の中心  r:  半径
返り値:  0:  正常、  -1:  異常

int HgCircleFill(double x, double y, double r, int stroke)
引数:  x,y:  円の中心  r:  半径  stroke:  円周を描くかどうか
返り値:  0:  正常、  -1:  異常
```

HgCircle() は、座標 (x, y) を中心とした半径 r の円を描きます。

HgCircleFill() は、座標 (x, y) を中心とした半径 r の塗りつぶされた円を描きます。塗りつぶしには HgSetPaintColor() で指定した色が使われます。引数 stroke が 0 の場合は円周を描きません。0 以外の値（例えば 1）の場合、他の線と同じ太さ、同じ色の線で円周を描きます。

中心点はウィンドウの外部の点でも構いません。半径は $r \geq 0.0$ の実数値です。

5.3 長方形

```
int HgBox(double x, double y, double w, double h)
引数:  x,y:  左下隅の座標  w,h:  幅と高さ
返り値:  0:  正常、  -1:  異常

int HgBoxFill(double x, double y, double w, double h, int stroke)
引数:  x,y:  左下隅の座標  w,h:  幅と高さ  stroke:  周囲を描くかどうか
返り値:  0:  正常、  -1:  異常
```

HgBox() は、座標 (x, y) を左下隅とする幅 w、高さ h の長方形を描きます。

HgBoxFill() は、座標 (x, y) を左下隅とする幅 w、高さ h の塗りつぶされた長方形を描きます。塗りつぶしには HgSetPaintColor() で指定した色が使われます。引数 stroke が 0 の場合は周囲に長方形を描きません。0 以外の値（例えば 1）の場合、他の線図形と同じ太さ、同じ色の線で長方形を描きます。

左下隅の座標はウィンドウの外部の点でも構いません。幅と高さは $w \geq 0.0, h \geq 0.0$ の実数値です。

5.4 文字列

```
int HgText(double x, double y, const char *str, ...)
引数:  x,y:  左下隅の座標  str:  書式文字列
返り値:  0:  正常、  -1:  異常
```

HgText() は、座標 (x, y) を左下隅として、指定された文字列を描きます。文字列 `str` 以降は C の標準関数 printf() と同様に書式を使った記述が可能です。

左下隅の座標はウィンドウの外部の点でも構いません。

5.5 ウィンドウ内の全消去

```
int HgClear(void)
返り値:  0:  正常、  -1:  異常
```

ウィンドウ内に描かれたすべての図形や文字を消去します。

6 Handy Graphic をもう少し使ってみる

6.1 色を自分で作るには

マクロで定められた色以外に、自分でさまざまな色を詳細に指定することも可能です。hgcolor 型の値の決め方には、グレーの濃度を指定する方法と、赤、緑、青 (RGB) の三原色の濃度で指定する方法があります。

```
hgcolor HgGray(double g)
引数:  g:  グレーの濃度
返り値:  色を表す値
```

```
hgcolor HgRGB(double r, double g, double b)
引数:  r:  赤の濃度  g:  緑の濃度  b:  青の濃度
返り値:  色を表す値
```

HgGray() は白から黒までの範囲でグレーの濃度を指定します。引数の `g` は $0.0 \leq g \leq 1.0$ の範囲の実数値で、0.0 が黒、1.0 が白です。

HgRGB() の引数も $0.0 \leq x \leq 1.0$ の範囲の実数値で、`r`、`g`、`b` がそれぞれ赤、緑、青の濃度を表します。例えば、`HgRGB(1.0, 0.0, 0.0)` で赤、`HgRGB(0.8, 0.8, 1.0)` で薄い青になります。

6.2 カレントポイントを使って線分を描く

関数 HgLine() では、線分の両端の座標を指定してその間を結んでいました。このような描き方のほかに、**カレントポイント**と呼ばれる点から指定した座標までを結ぶ描き方があります。

カレントポイントとは、紙の上に置いたペン先の位置と考えればよいでしょう。ペンを紙に置いたまま別の点まで動かせば、その間に直線を描くことができます。また、ペン先を紙から放して位置だけ移動させることもできます。この場合、線は描かれません。

カレントポイントだけを設定する（図形は描かない）ためには次の関数を使います。

```
int HgMoveTo(double x, double y)
引数:  x,y:  新しいカレントポイントの座標
返り値:  0:  正常、  -1:  異常
```

現在のカレントポイントから別の点まで直線を描くには次の関数を使います。カレントポイントは新しく指定した点になります。

```
int HgLineTo(double x, double y)
引数:   x,y:  線分の終点
返り値:   0:  正常、   -1:  異常
```

なお、2点を指定して直線を描く関数 HgLine() で描画すると、2つめの点 (x_1, y_1) が新たなカレントポイントになります。

6.3 折れ線と多角形

複数の点の間を結ぶ折れ線を一度に描画することができます。複数の点の座標は配列に用意しておきます。

```
int HgLines(int n, const double *xp, const double *yp)
引数:   n:  頂点数   xp,yp:  頂点の x 座標、y 座標を格納した配列へのポインタ
返り値:   0:  正常、   -1:  異常
```

n は折れ線で結ぶ点の個数です。これらの点の座標値は、x 軸と y 軸の値に分かれて、ポインタ xp と yp で参照される配列に入れられているものとします。点の座標はウィンドウの外部でも構いません。最後の座標値が新しいカレントポイントになります。

複数の点を同様な方法で指定して、多角形を描くことができます。折れ線と異なり、最初の点と最後の点の間が結ばれます。

```
int HgPolygon(int n, const double *xp, const double *yp)
引数:   n:  頂点数   xp,yp:  頂点の x 座標、y 座標を格納した配列へのポインタ
返り値:   0:  正常、   -1:  異常

int HgPolygonFill(int n, const double *xp, const double *yp, int stroke)
引数:   n:  頂点数   xp,yp:  頂点の x 座標、y 座標を格納した配列へのポインタ
        stroke:  周囲を描くかどうか
返り値:   0:  正常、   -1:  異常
```

関数 HgPolygon() は線を描くだけですが、関数 HgPolygonFill() は描かれた図形を塗りつぶします。引数 stroke が 0 以外の値の場合、周囲の線も描きます。

6.4 円弧と扇型

```
int HgArc(double x, double y, double r, double a0, double a1)
引数:   x,y:  円の中心   r:  半径   a0:  始点角度   a1:  終点角度
返り値:   0:  正常、   -1:  異常
```

HgArc() は、座標 (x, y) を中心とした半径 r の円について、開始角度 $a0$ と終了角度 $a1$ を指定して円弧を描きます。円弧は始点角度から終点角度までを反時計回りに結びます。中心座標はウィンドウの外部でも構いません。

ここでいう角度とは、弧の端点と中心を結ぶ直線が x 軸となす角度で、弧度法で指定します。従って、例えば 90 °ではなく $\pi/2$ を使います。これは、C 言語の標準の算術関数が弧度法を用いているのに合わせています。

同様な指定で、扇形を描くことができます。

```
int HgFan(double x, double y, double r, double a0, double a1)
引数:   x,y:  円の中心   r:  半径   a0,a1:  始点、終点角度
返り値:   0:  正常、   -1:  異常

int HgFanFill(double x, double y, double r, double a0, double a1, int stroke)
引数:   x,y:  円の中心   r:  半径   a0,a1:  始点、終点角度
       stroke:  周囲を描くかどうか
返り値:   0:  正常、   -1:  異常
```

扇型の弧は円弧と同様、始点角度から終点角度までを反時計回りに結び、円の中心との間に線分を描きます。

HgFanFill() は塗りつぶされた扇型を描きます。引数 stroke が 0 以外の値の場合、周囲の線も描きます。

6.5 座標原点の移動と縮尺の指定

これまでの例題のプログラムでは、ウィンドウの左下を座標原点 (0, 0) として図形の位置を決めていましたが、これを別の点にするように指定できます。また、図形の位置決めを 1 画素を単位する長さで行っていましたが、この縮尺を 1 倍として何倍の縮尺を使うかを指定することもできます。このようにして定めた座標を**アプリケーション座標**と呼びます。

```
int HgCoordinate(double sx, double sy, double scale)
引数:   sx, sy:  原点の指定   scale:  縮尺
返り値:   0:  正常、   -1:  異常
```

sx, sy は新しい座標で原点とする位置を、ウィンドウの左下を原点とした場合の座標で指定します。ウィンドウ上に表示されていない点でもかまいません。

引数 scale は縮尺で、正の実数値を指定します。2.0 は 2 倍の拡大で 2 画素が長さの 1 単位に相当し、逆に 0.5 は 1/2 の縮小で 2 単位が 1 画素に相当します。地図の縮尺と同じで、縮尺を 1/10000 にすれば半径が 100000 の円も小さく描画できます。ただし、縮尺は線の太さや文字の大きさには影響しません。

この関数を用いると、それまでにウィンドウ内に描かれたすべての図形や文字は消されます。

6.6 描いた結果を画像として保存する

描画した結果を PDF 形式のファイルとして保存することができます。ただし現状では画素の集合として保存されますので、拡大すると画像が粗くなります。

HgDisplayer のメニューから「描画 → 画像を保存」を選択すると、ファイル名と保存場所を指定して、その時に描かれている内容を保存できます。

プログラムから保存を指定することもできます。これには次の関数を使います。

```
int HgSave(const char *str)
引数:   str:  ファイル名
返り値:   0:  正常、   -1:  異常
```

呼び出された時に描かれている内容を画像ファイルに保存します。ファイルは文字列 str で指定されたファイル名で作成され、必要なら拡張子（.pdf）が付けられます。書き出しが正常に行えなかった場合はエラーになります。

7　複数のウィンドウを使うプログラム

7.1　複数のウィンドウを作成するには

関数 HgOpen() で作成したウィンドウは**標準ウィンドウ**と呼びます。標準ウィンドウは１つしかありません。

Handy Graphic では複数のウィンドウを同時に表示し、それぞれに異なる描画を行うことができます。複数のウィンドウを作成するには次の関数を使います。この関数を呼び出すたびに、新しいウィンドウが作成されます。

```
int HgWOpen(double x, double y, double w, double h)
```
引数:　x,y:　左下隅の座標　w,h:　幅と高さ
返り値:　0 あるいは正整数:　ウィンドウ id、　-1:　異常

この関数の返す値は**ウィンドウ id** と呼ばれる整数で、ウィンドウごとに異なります。この値を使って、どのウィンドウに描画するかを指定します。標準ウィンドウのウィンドウ id は常に 0 と定められています。なお、ウィンドウが何も表示されていない時に関数 HgWOpen() で新しいウィンドウを作成すると、ウィンドウ id は 0、つまり標準ウィンドウになります。

関数 HgWOpen() ではウィンドウの表示される位置を指定しなければなりませんが、画面の大きさは使うコンピュータによっては違うこともあります。そこで、プログラムを実行しているコンピュータの画面の大きさを調べる関数が用意されています。

```
void HgScreenSize(double *width, double *height)
```
引数:　width, height:　ディスプレイ装置の幅と高さを格納するためのポインタ

ディスプレイ装置の幅と高さを、ポインタ width と height が指す変数へ書き込みます。長さの単位は画素です。

7.2　指定したウィンドウに描画するには

これまでに示した描画用の関数はすべて標準ウィンドウを対象とするものでした。HgWOpen() で作成したウィンドウに描画を行うには、ウィンドウ id を引数とする描画用関数を使います。

例えば、標準ウィンドウに線分を描く関数 HgLine() に対し、指定したウィンドウに描画を行う関数は HgWLine() という名前で定義されています。関数 HgWLine() は第１引数にウィンドウ id を指定します。両者を下に示します。

```
int HgLine(double x0, double y0, double x1, double y1)

int HgWLine(int wid, double x0, double y0, double x1, double y1)
```

この例のように、標準ウィンドウを対象とした関数が Hg… という名前なのに対し、ウィンドウを指定できる関数は HgW… という名前で、第１引数にウィンドウ id を指定します。

この文書の最後に付録として関数一覧を掲載しています。ここには、標準ウィンドウを対象とする関数と、ウィンドウ id を指定する関数が掲載されています。

ウィンドウ id として 0 を指定すると標準ウィンドウを描画の対象にできます。また、図形の色、線の太さ、フォント、および座標の指定はウィンドウごとに個別の設定が保たれます。

7.3 ウィンドウを閉じる

標準ウィンドウを画面上から消去します。

```
int HgClose(void)
返り値:  0:  正常、  -1:  異常
```

指定したウィンドウを閉じるには次の関数 HgWClose() を使います。また、関数 HgCloseAll() は、表示されているウィンドウがあればそれらをすべて閉じます。

```
int HgWClose(int wid)
引数:  wid:  ウィンドウ id
返り値:  0:  正常、  -1:  異常

void HgCloseAll(void)
```

プログラムの終了時には自動的にウィンドウは閉じますので、これらの関数の主な用途は、プログラムの実行中にウィンドウを閉じることと言えます。

7.4 ウィンドウにタイトルを付ける

何も指定しない場合、HgDisplayer は実行しているプログラムのプロセス番号をウィンドウの上部に表示します。この部分に、タイトルとして任意の文字列を表示させることができます。関数 HgSetTitle() は標準ウィンドウ用、関数 HgWSetTitle() はウィンドウを指定して利用します。

文字列 `str` 以降は C の標準関数 printf() と同様に書式を使った記述が可能です。

```
int HgSetTitle(const char *str, ...)
引数:  str:  書式文字列
返り値:  0:  正常、  -1:  異常

int HgWSetTitle(int wid, const char *str, ...)
引数:  wid:  ウィンドウ id  str:  書式文字列
返り値:  0:  正常、  -1:  異常
```

8 イベントに関する機能

描画部分をマウスでクリックしたり、キーボードから何かを入力した場合にその情報を取得することができます。このようにプログラム外部からやってくる情報を**イベント**と呼びます。通常の GUI システムではイベントを受け取ったことをきっかけとしてプログラムが動くようになっていますが、Handy Graphic では何かのイベントが来るまで待つという形でイベントを取得します。

なお、HgDisplayer は、描画のみのプログラムは並列して複数個を動作させることができますが、イベントを取得して動作するプログラムはひとつだけしか動作させることができません。

8.1 キー入力を得る

次の関数を呼び出すことで、標準ウィンドウへのキー入力（文字）を得ることができます。標準ウィンドウが前面に出ている必要があります。

```
int HgGetChar(void)
返り値:  0 以上:  入力された文字、  -1:  異常
```

例えば A のキーを押すと小文字の 'a' の文字コードに相当する 97 が返り、シフトキーを押しながら A を押すと 'A' に相当する 65 が返ります。

矢印キーを押した場合には、表 4 のマクロで表される整数値が返ります。

また、標準ウィンドウ以外のウィンドウへのキー入力を得るために次の関数を利用できます。

```
int HgWGetChar(int wid)
引数:  wid:  ウィンドウ id
返り値:  0 以上:  入力された文字、  -1:  異常
```

ただし、あるウィンドウへのキー入力を待っている間、他のウィンドウに対する入力はすべて捨てられてしまいます。HgGetChar() または HgWGetChar() を使う場合、どれかひとつのウィンドウだけからキー入力を得るようにプログラムする必要があります。

関数 HgGetChar() および HgWGetChar() は、下で述べるイベントの取得のための関数（8.2）を利用して、キー入力だけが簡単に得られるようにしています。複数のウィンドウへのキー入力やマウスのクリックなどのイベントも取得するためには HgEventMask() と HgEvent() だけを使い、HgGetChar() または HgWGetChar() は同時には使わないようにして下さい。

表 4: 矢印キーに相当するマクロ名

矢印キー	マクロ名
上向き（↑）	HG_U_ARROW
下向き（↓）	HG_D_ARROW
左向き（←）	HG_L_ARROW
右向き（→）	HG_R_ARROW

表 5: イベントマスクとマクロ名

イベントの種類	マクロ名
マウスボタンが押された	HG_MOUSE_DOWN
キーボードが押された	HG_KEY_DOWN

8.2 イベントの取得

ここではマウスのクリックとキーボードからの入力についてのみ説明します。

まず、どのようなイベントを扱うのかを設定する必要があります。この設定はウィンドウごとに行います。下の関数 HgEventMask() は標準ウィンドウに対する設定を行いますが、指定したウィンドウに設定を行う関数 HgWEventMask() もあります。

```
int HgEventMask(unsigned int mask)
引数:  mask:  イベントマスク
返り値:  0:  正常、  -1:  異常
```

イベントマスクは、表 5 のマクロのどちらか、あるいはこれらをビット OR で結合したものです。イベントマスクで指定したイベントを取得できるようになりますので、マウスのクリックだけ取得する場合には

HG_MOUSE_DOWN を指定し、マウスクリックもキー入力も取得する場合は (HG_MOUSE_DOWN | HG_KEY_DOWN) を指定することになります。

イベントを取得するには次の関数を使います。

```
hgevent *HgEvent(void)
返り値:  NULL: 異常、  NULL 以外:  hgevent 構造体へのポインタ
```

この関数は、HgEventMask() で指定したイベントが来るまで待ちます。複数のウィンドウがあっても、すべてこの関数で扱います。ウィンドウがひとつも表示されていないか、どのウィンドウもイベントマスクの設定を行っていないか、あるいは他の何らかのエラーが生じた場合に NULL が返されます。

何かイベントがあると、次のような構造体へのポインタが返されます。

```
typedef struct {
    unsigned long   type;       /* 発生したイベントを表す */
    int             wid;        /* イベントのあったウィンドウの id */
    double          x;          /* (x, y) = マウスイベントが発生した位置 */
    double          y;
    int             count;
    unsigned int    modkey;
    unsigned int    ch;         /* 入力されたキーを示す文字 */
} hgevent;
```

type には、イベントの種類に応じて、表 5 のどれかの値が入れられています。wid はイベントの発生したウィンドウ id を表します。

マウスがクリックされた場合、(x, y) がその位置を表しますが、この座標はウィンドウの左下を原点と考えた時の位置になります。

キーボード上のキーが押された場合、ch にそのキーの文字を表す整数値が入れられます。

HgEventMask() が返すポインタが指すメモリ領域を解放する必要はありません。ただし、あるイベントによって返された情報は、その後に複数個のイベントが発生すると上書きされることがあります。イベントの情報を保存したい場合は、別の変数を用意して必要な情報をコピーしておいて下さい。

8.3 座標変換

```
void HgTransWtoA(double wx, double wy, double *ax, double *ay)
引数:   wx, wy:  ウィンドウ座標での点の座標
        ax, ay:  アプリケーション座標に変換された値を格納するためのポインタ

void HgTransAtoW(double ax, double ay, double *wx, double *wy)
引数:   wx, wy:  アプリケーション座標での点の座標
        ax, ay:  ウィンドウ座標に変換された値を格納するためのポインタ
```

関数 HgCoordinate() でアプリケーション座標の設定をした場合に、ウィンドウ上の物理的な位置（ウィンドウ座標）とアプリケーション座標の間で、位置の座標変換を行います。

関数 HgTransWtoA() はウィンドウ座標からアプリケーション座標へ変換し、関数 HgTransAtoW() はアプリケーション座標からウィンドウ座標へ変換します。たとえば、マウスクリックのイベントで返される位置情報はウィンドウ座標ですので、必要ならば関数 HgTransWtoA() を使ってアプリケーション座標に変換します。

上記の２つの関数は標準ウィンドウを対象としていますが、ウィンドウ id でウィンドウを指定して座標変換を行う関数、HgWTransWtoA()、HgWTransAtoW() も用意されています。

8.4 その他

Handy Graphic の仕様ではありませんが、１秒以下の短時間だけ実行を停止させる関数を用意しています。ゆっくり描かせたい場合などに、描画関数の間にはさむとよいでしょう。

単位は秒で、例えば引数に 0.5 を指定すると 0.5 秒だけ実行を一時停止します。ただし、時間の厳密な正確さは保障されません。

```
void msleep(double msec)
引数:   msec:  時間間隔（秒）
```

付録　関数の一覧

関数の役割	関数	指定ウィンドウ用の関数	参照
ウィンドウを作成	HgOpen(wd,hg)	HgWOpen(x,y,wd,hg)	4.1,7.1
ウィンドウを閉じる	HgClose()	HgWClose(w)	7.3
全ウィンドウを閉じる	HgCloseAll()		7.3
ウィンドウタイトル	HgSetTitle(str,...)	HgWSetTitle(w,str,...)	7.4
画面の大きさを得る	HgScreenSize(wp,hp)		7.1
線の太さを指定	HgSetWidth(t)	HgWSetWidth(w,t)	4.2
灰色を作る	HgGray(g)		6.1
色を RGB で作る	HgRGB(r,g,b)		6.1
線の色を指定	HgSetColor(c)	HgWSetColor(w,c)	4.3
塗りつぶし色を指定	HgSetPaintColor(c)	HgWSetPaintColor(w,c)	4.3
文字コードを指定	HgEncoding(code)		4.4
フォントを指定	HgSetFont(f,sz)	HgWSetFont(w,f,sz)	4.4
２点間に線分を描く	HgLine(x0,y0,x1,y1)	HgWLine(w,x0,y0,x1,y1)	5.1
カレントポイントを移動	HgMoveTo(x,y)	HgWMoveTo(w,x,y)	6.2
指定点まで線分を描く	HgLineTo(x,y)	HgWLineTo(w,x,y)	6.2
円を描く	HgCircle(x,y,r)	HgWCircle(w,x,y,r)	5.2
円を塗りつぶす	HgCircleFill(x,y,r,s)	HgWCircleFill(w,x,y,r,s)	5.2
円弧を描く	HgArc(x,y,r,a0,a1)	HgWArc(w,x,y,r,a0,a1)	6.4
扇型を描く	HgFan(x,y,r,a0,a1)	HgWFan(w,x,y,r,a0,a1)	6.4
扇型を塗りつぶす	HgFanFill(x,y,r,a0,a1,s)	HgWFanFill(w,x,y,r,a0,a1,s)	6.4
長方形を描く	HgBox(x,y,wd,hg)	HgWBox(w,x,y,wd,hg)	5.3
長方形を塗りつぶす	HgBoxFill(x,y,w,h,s)	HgWBoxFill(w,x,y,wd,hg,s)	5.3
文字列を描く	HgText(x,y,str,...)	HgWText(w,x,y,str,...)	5.4
折れ線を描く	HgLines(n,x,y)	HgWLines(w,n,x,y)	6.3
多角形を描く	HgPolygon(n,x,y)	HgWPolygon(w,n,x,y)	6.3
多角形を塗りつぶす	HgPolygonFill(n,x,y,s)	HgWPolygonFill(w,n,x,y,s)	6.3
全消去	HgClear()	HgWClear(w)	5.5
画像を保存	HgSave(file)	HgWSave(w,file)	6.6
アプリケーション座標	HgCoordinate(x,y,sc)	HgWCoordinate(w,x,y,sc)	6.5
座標変換	HgTransWtoA(wx,wy,ax,ay)	HgWTransWtoA(w,wx,wy,ax,ay)	8.3
座標変換	HgTransAtoW(ax,ay,wx,wy)	HgWTransAtoW(w,ax,ay,wx,wy)	8.3
キー入力を得る	HgGetChar()	HgWGetChar(w)	8.1
イベントマスクを指定	HgEventMask(m)	HgWEventMask(w,m)	8.2
イベントを取得	HgEvent()		8.2