

G L S C (ver. 3.3) 解説書

龍谷大学 理工学部 数理情報学科

小林 亮・高橋 大輔・中野 浩
松木平 淳太

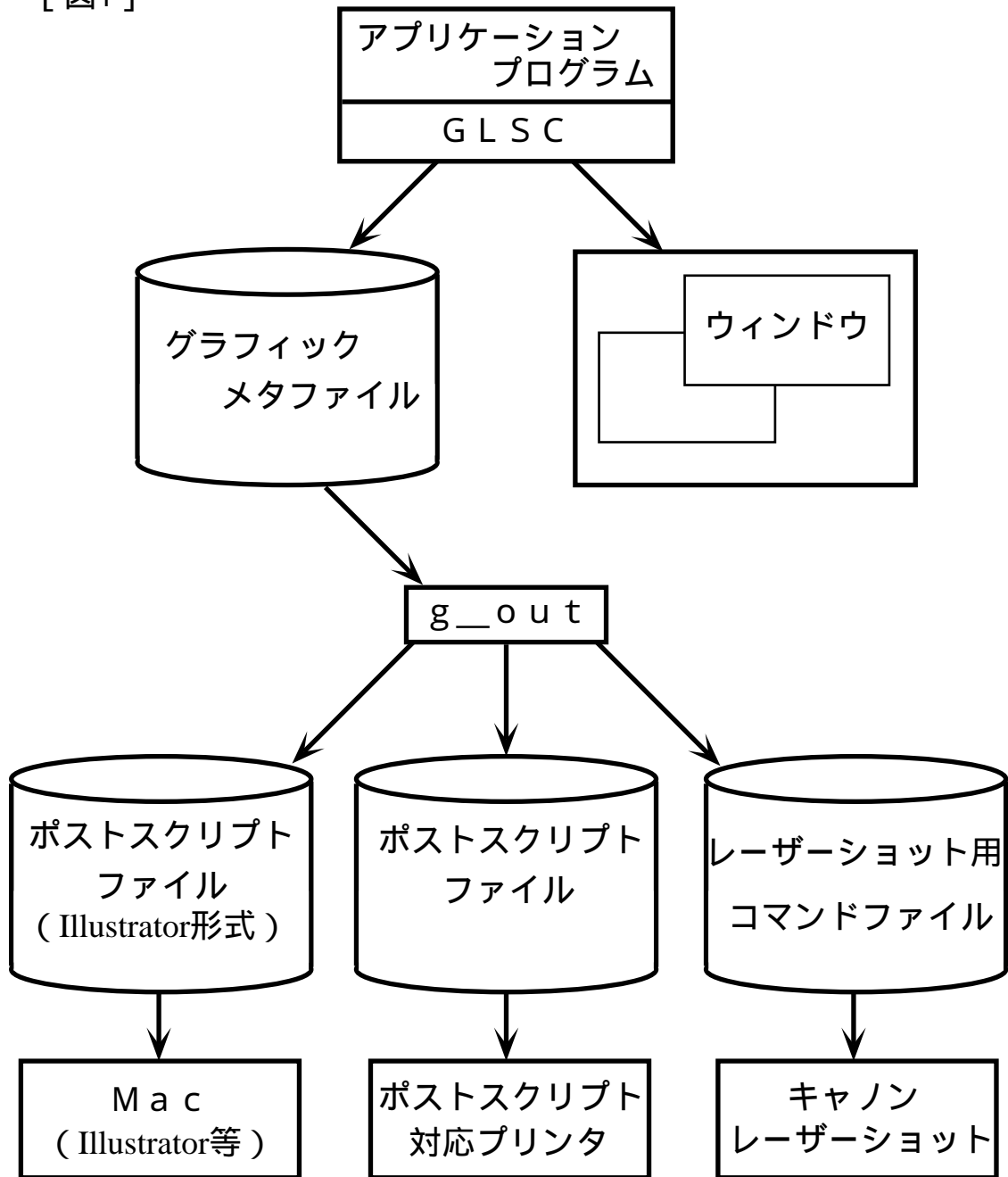
概説

GLSCとは、Graphic Library for Scientific Computing の略で、科学技術計算の結果をディスプレイ上に表示したり、プリンタに出力したりするための簡単なグラフィックライブラリです。このライブラリは、C言語とFORTRANから呼び出して使うことができます。以下の各関数(サブルーチン)の解説においては、書式はすべてC言語での書き方を記してあります。特にC言語とFORTRANで呼ぶ時の引数が異なる場合のみ2通りの書き方を記してあります。GLSCでどのようなことができるかをおおまかに述べておきましょう。

GLSCは基本的にウインドウに絵を表示するためのものですが、それだけでなくグラフィックデータをメタファイルの形で保存することができます。このメタファイルの中でのコマンド形式はGLSC独自のものです。このメタファイルはg_outというコマンドを用いることによりPostscript形式のファイルに変換することができます。ということはPostscript対応のプリンタがあれば、絵を出力することができます。また、CannonのLaser Shot用のコマンドファイルに変換することもできますので、Laser Shotにも絵が出せます。現段階ではプリンタは白黒のもののみを考えています。このg_outというコマンドはいろいろな機能を持っていて、メタファイルの変換をすると同時に、簡単なレイアウトをしたり、絵の拡大・縮小等もできます。さらに、g_outはメタファイルをIllustrator形式のファイルにも変換でき、Macに取り込んで編集が可能です。詳しくはg_outコマンド解説(p.44)を参照して下さい。

これらの作業の流れを次ページの [図1] に示しておきます。

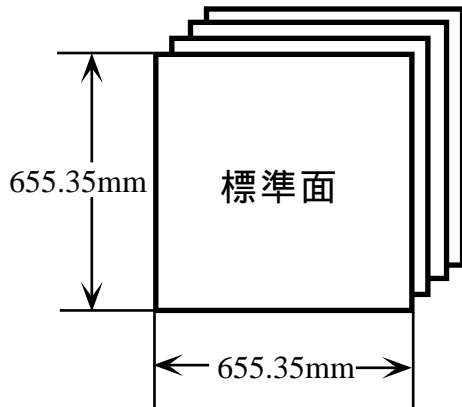
[図1]



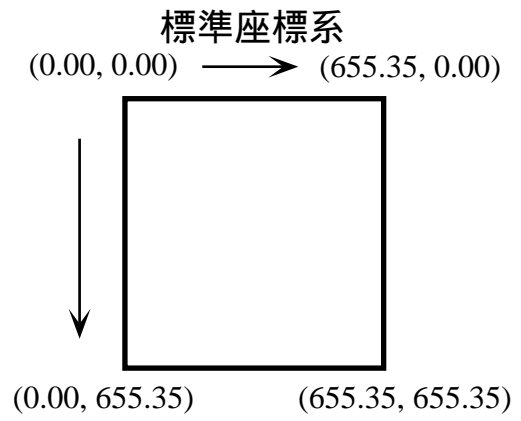
GLSCにおいては、描画は標準面というものに対して行なわれます。標準面というのは、1辺が655.35mmの正方形の白い紙だと思って下さい。そしてこの紙が好きなだけ用意されています [図2]。以下ではこの白い紙1ページのことを1フレームと呼ぶことにしましょう。このフレームを用意するための関数が `g_cls` です。 `g_cls` をコールすると、1フレームが用意され、描画可能状態になります。そのフレームに対する描画が終わって、次のフレームに行くときにまた `g_cls` をコールします。このようにして何枚でも好きな数のフレームに描画を行なうことができます。これらのフレームには標準座標系というものが導入されています。この座標系は [図3] のように、左上頂点を (0.00, 0.00) とし、右下頂点を (655.35, 655.35) とするようになっています。y座標の方は通常の座標系と上下が逆になっていることと、この座標系が実数を用いて書かれていることに注意して下さい。GLSCでは通常、絵を描く時はこの標準座標系を用いることはありません。(`g_text` だけは例外で標準座標系を用います。) 描画するときには使うのは仮想座標系です。これは標準面上にユーザーが定義する座標系で同時に50個まで定義できます [図4]。具体的には `g_def_scale` を用いて、必要なだけ座標系を導入しておいて、 `g_sel_scale` によって、使用したい仮想座標系を選択するという形をとります。以下、描画時に選択されている仮想座標系のことをカレント座標系と呼ぶことにします。これらの仮想座標系は、 `g_cls` がコールされてフレームが変わっても影響を受けません。

この標準面に描かれた絵は、グラフィックメタファイルとして保存したり、ウィンドウ上に表示することができます。標準面の中で、ウィンドウに表示される領域は、標準面の左上頂点とウィンドウの左上頂点を一致させたときウィンドウと重なった領域です [図5]。そして標準面上の絵は、実寸(要するに標準面上での大きさそのもの)で表示されます。ですから、よほど大きな画面のディスプレイでもない限り、標準面いっぱい絵を描いたりすると、ウィンドウ上に表示しきれないこととなります(メタファイルには保存されています)。よって、普通は標準面左上隅のディスプレイにおさまるくらいの領域の中に絵を描いて下さい。

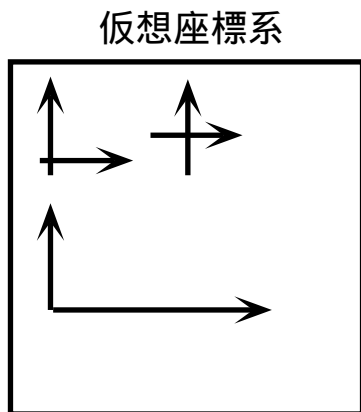
[図2]



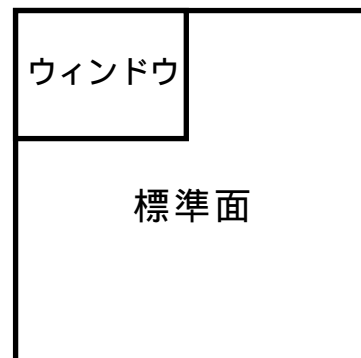
[図3]



[図4]



[図5]



GLSCには、線、ぬりつぶし、マーカー、テキストの4通りの描画単位があります。これらの描画単位はそれぞれ次のような属性を持っています。

線	色、太さ、種類
ぬりつぶし	色 (ディスプレイ上では色ですが、プリンタ上ではぬりつぶしのパターンに対応します。)
マーカー	色、大きさ、種類
テキスト	色、フォント (現在のところ times でサイズだけがちがいます。プリンタ上では1通りだけです。)

描画単位は、それが描かれる時点での属性に従って描かれます。この描画単位が描かれる時点での属性のことをカレント属性と呼びます。このカレント属性を変化させながら描画することにより、色々な絵が描けます。描画を行なうための関数は次のようなものです。

線	g_move と g_plot のペア g_polyline, g_data_plot, g_contln, g_box, g_circle, g_polygon において edge = G_YES のとき。
ぬりつぶし	g_box, g_circle, g_polygon において fill = G_YES のとき (注: basic における paint にあたるものではありません。)
マーカー	g_marker
テキスト	g_text

これらの関数は、コールされる時点でのカレント属性にしたがって描画を行ないます。ゆえに、このカレント属性をコントロールするための関数が用意されています。それは次のようなものです。

g_line_color, g_line_width, g_line_type, g_area_color, g_marker_color,
g_marker_size, g_marker_type, g_text_color, g_text_font

それぞれの機能は名前を見ればわかるでしょう。

これらの属性コントロール関数を描画関数をコールする前にコールしてやれば自分の望む属性で描画を行なうことができます。また、これらの属性はすべて整数値で指定されますが、そのうちのいくつかは /usr/include/glsc.h の中で定義されているマクロを用いて書くこともできます。

カレント属性をコントロールする方法としては、この方法を用いる以外に、もう一つのやり方があります。例えば、線を例にとりますと、線の属性のセット（色、大きさ、種類）を1度に定義し（例えば、赤くて太い点線）、それに属性番号をつけておいて、あとでその属性番号を呼び出すことにより、一挙にカレント属性を変更するという方法です。それは次のようなパターンです。

```
g_def_line (3, G_RED, 3, G_LINE_DASHED);
    :
g_sel_line (3);
```

どちらのやり方をとるかは人の好き好きですが、度々、属性変更をする場合には、はじめにまとめて属性定義をしておいて後で呼び出す方法の方がよいと思います。もちろん、すべてのカレント属性は、それを換えようとしなない限りは、そのまま持続します。また g_init をコールした時点でカレント属性は次のようなデフォルト値にセットされます。

```
線          : 色…黒、 太さ…0、 種類…0（実線）
ぬりつぶし  : 色…白
マーカー    : 色…黒、 大きさ…5、 種類…0（*）
テキスト    : 色…黒、 フォント…2（times 18）
```

以上に説明したように、GLSCではカレント属性のコントロールと描画単位を描くということの組み合わせですべての絵を描いて行きます。ただし、g_bird_view と g_hidden 及び g_fake_bird_view と g_fake_hidden は例外です。これらは、 $z = f(x,y)$ のグラフの鳥瞰図（及び鳥瞰図もどき）を描く関数なのですが、これらはすべてのカレント属性を無視しますし、またカレント属性に影響を与えません。

GLSCにおいて、各関数が呼ばれる順番の典型的な例は次のようなものです。

<code>g_init (...);</code>	初期化
<code>g_device (...);</code>	出力先指定
<code>g_def_scale (...);</code>	
<code>:</code>	スケール定義
<code>g_def_scale (...);</code>	
<code>g_def_line (...);</code>	
<code>:</code>	属性セット定義
<code>g_def_text (...);</code>	
 <code>g_cls ();</code>	
<div style="border: 1px solid black; padding: 5px; display: inline-block;">1 フレーム 描画</div>	1 フレーム目 描画
 <code>g_cls ();</code>	
<div style="border: 1px solid black; padding: 5px; display: inline-block;">1 フレーム 描画</div>	2 フレーム目 描画
 <code>:</code>	
 <code>g_cls ();</code>	
<div style="border: 1px solid black; padding: 5px; display: inline-block;">1 フレーム 描画</div>	最終フレーム 描画
 <code>g_term ();</code>	終了

大事なことは、`g_init` ではじまり、`g_term` で終わる一連のコーリングシーケンスの中では、アクセスできるメタファイルは1つだけで、開くことのできるウィンドウも1つだけだということです。ただし、このメタファイルやウィンドウへの出力モードは `g_device` を用いて、途中でもいろいろ変えることはできます。（例えば、最終フレーム以外はウィンドウでモニターするのみで、最終フレームだけはウィンドウにもメタファイルにも出力する等。）

テストプログラムのソースファイル（C 言語 と FORTRAN）と実行結果を巻末に載せておきますので、テストプログラムを読んで、GLSC が実際にどのように使われているかを見て下さい。

GLSC を使用するときには、次のライブラリをリンクして下さい。

単精度実数を使用する場合 : `-lglscs -lX11 -lm`
倍精度実数を使用する場合 : `-lglscd -lX11 -lm`

例

```
cc abc.c def.c -lglscs -lX11 -lm
f77 -o xyz pqr.f stu.f -lglscd -lX11 -lm
```

注意：GLSC を使用するときには、実数は単精度か倍精度かのいずれか一方に統一して下さい。

目次（機能別）

1) 制御関数

g_init	グラフィックの初期化をする	P.14
g_device	出力先を指定・変更する	P.15
g_cls	新しいフレームを用意する	P.15
g_term	グラフィックスを終了する	P.15

2) スケール関数

g_def_scale	標準面上に仮想座標系を定義する	P.16
g_sel_scale	仮想座標系を選択する	P.17

3) 属性コントロール関数

g_line_color	線の色を変更する	P.18
g_line_width	線の太さを変更する	P.19
g_line_type	線の種類を変更する	P.20
g_area_color	ぬりつぶしの色（パターン）を変更する	P.21
g_marker_color	マーカーの色を変更する	P.22
g_marker_size	マーカーの大きさを変更する	P.22
g_marker_type	マーカーの種類を変更する	P.23
g_text_color	テキストの色を変更する	P.24
g_text_font	テキストのフォントを変更する	P.24
g_def_line	線の属性セットを定義する	P.25
g_def_area	ぬりつぶしの属性セットを定義する	P.25
g_def_marker	マーカーの属性セットを定義する	P.26
g_def_text	テキストの属性セットを定義する	P.26
g_sel_line	線の属性セットを選択する	P.27
g_sel_area	ぬりつぶしの属性セットを選択する	P.27
g_sel_marker	マーカーの属性セットを選択する	P.28
g_sel_text	テキストの属性セットを選択する	P.28

4) 描画関数

<code>g_move</code>	カレントポジションを移動する	P.29
<code>g_plot</code>	カレントポジションから指定された点まで線分をひく	P.29
<code>g_box</code>	長方形を描いたり、中をぬりつぶしたりする	P.30
<code>g_circle</code>	円を描いたり、中をぬりつぶしたりする	P.30
<code>g_polygon</code>	多角形を描いたりぬりつぶしたりする	P.31
<code>g_polyline</code>	与えられた点列を線分で連続的に結ぶ	P.31
<code>g_data_plot</code>	与えられた配列を用いてグラフをかく	P.32
<code>g_marker</code>	マーカーを描く	P.33
<code>g_text</code>	文字列をかく	P.33

5) 上位関数

<code>g_contln</code>	等高線を描く	P.34
<code>g_bird_view</code>	$z = f(x,y)$ のグラフの鳥瞰図を隠線処理をせずに描く	P.35
<code>g_hidden</code>	$z = f(x,y)$ のグラフの鳥瞰図を隠線処理をして描く	P.36
<code>g_fake_bird_view</code>	$z = f(x,y)$ のグラフの鳥瞰図もどきを隠線処理をせずに描く	P.38
<code>g_fake_hidden</code>	$z = f(x,y)$ のグラフの鳥瞰図もどきを隠線処理をして描く	P.39

6) 補助関数

<code>g_sleep</code>	描画後、絵を消さずに表示する	P.41
<code>g_sformat</code>	変数を指定された書式で表わし文字列として返す	P.42

目次 (アルファベット順)

1.	<code>g_area_color</code>	ぬりつぶしの色 (パターン) を変更する	P.21
2.	<code>g_bird_view</code>	$z = f(x,y)$ のグラフの鳥瞰図を隠線処理をせずに描く	P.35
3.	<code>g_box</code>	長方形を描いたり、中をぬりつぶしたりする	P.30
4.	<code>g_circle</code>	円を描いたり、中をぬりつぶしたりする	P.30
5.	<code>g_cls</code>	新しいフレームを用意する	P.15
6.	<code>g_contln</code>	等高線を描く	P.34
7.	<code>g_data_plot</code>	与えられた配列を用いてグラフをかく	P.32
8.	<code>g_def_area</code>	ぬりつぶしの属性を定義する	P.25
9.	<code>g_def_line</code>	線の属性セットを定義する	P.25
10.	<code>g_def_marker</code>	マーカーの属性セットを定義する	P.26
11.	<code>g_def_scale</code>	標準面上に仮想座標系を定義する	P.16
12.	<code>g_def_text</code>	テキストの属性セットを定義する	P.26
13.	<code>g_device</code>	出力先を指定・変更する	P.15
14.	<code>g_fake_bird_view</code>	$z = f(x,y)$ のグラフの鳥瞰図もどきを隠線処理をせずに描く	P.38
15.	<code>g_fake_hidden</code>	$z = f(x,y)$ のグラフの鳥瞰図もどきを隠線処理をして描く	P.39
16.	<code>g_hidden</code>	$z = f(x,y)$ のグラフの鳥瞰図を隠線処理をして描く	P.36
17.	<code>g_init</code>	グラフィックの初期化をする	P.14
18.	<code>g_line_color</code>	線の色を変更する	P.18
19.	<code>g_line_type</code>	線の種類を変更する	P.20
20.	<code>g_line_width</code>	線の太さを変更する	P.19

21.	<code>g_marker</code>	マーカーを描く	P.33
22.	<code>g_marker_color</code>	マーカーの色を変更する	P.22
23.	<code>g_marker_size</code>	マーカーの大きさを変更する	P.22
24.	<code>g_marker_type</code>	マーカーの種類を変更する	P.23
25.	<code>g_move</code>	カレントポジションを移動する	P.29
26.	<code>g_plot</code>	カレントポジションから指定された点まで線分をひく	P.29
27.	<code>g_polygon</code>	多角形を描いたりぬりつぶしたりする	P.31
28.	<code>g_polyline</code>	与えられた点列を線分で連続的に結ぶ	P.31
29.	<code>g_sel_area</code>	ぬりつぶしの属性セットを選択する	P.27
30.	<code>g_sel_line</code>	線の属性セットを選択する	P.27
31.	<code>g_sel_marker</code>	マーカーの属性セットを選択する	P.28
32.	<code>g_sel_scale</code>	仮想座標系を選択する	P.17
33.	<code>g_sel_text</code>	テキストの属性セットを選択する	P.28
34.	<code>g_sformat</code>	変数を指定された書式で表わし文字列として返す	P.42
35.	<code>g_sleep</code>	描画後、絵を消さずに表示する	P.41
36.	<code>g_term</code>	グラフィックスを終了する	P.15
37.	<code>g_text</code>	文字列をかく	P.33
38.	<code>g_text_color</code>	テキストの色を変更する	P.24
39.	<code>g_text_font</code>	テキストのフォントを変更する	P.24

`g_init` グラフィックの初期化をする

書式 `g_init (file_name, size_x, size_y);`
`call g_init (file_name, length, size_x, size_y)`

入力パラメータ

<code>file_name</code>	文字列	メタファイル名
<code>length</code>	整数	<code>file_name</code> の文字数
<code>size_x</code>	実数	ディスプレイ上に開くウインドウの横方向のサイズ (mm)
<code>size_y</code>	実数	ディスプレイ上に開くウインドウの縦方向のサイズ (mm)

説明

メタファイルに出力しない場合でもなんらかのダミーのファイル名をかくこと。ウインドウに出力しない場合でもなんらかのダミーのウインドウサイズをかくこと。C言語からコールする場合は `length` は不要だが、FORTRAN からコールする場合は `length` が必要である。

`g_device` 出力先を指定・変更する

書式 `g_device (device);`

入力パラメータ

`device` 整数 グラフィックデータの出力先を指定する番号

device		出力先
番号	マクロ	
0	G_NONE	なし
1	G_META	メタファイルのみ
2	G_DISP	ディスプレイのみ
3	G_BOTH	メタファイルとディスプレイ

`g_cls` 新しいフレームを用意する

書式 `g_cls ();`

`g_term` グラフィクスを終了する

書式 `g_term ();`

`g_def_scale` 標準面上に仮想座標系を定義する

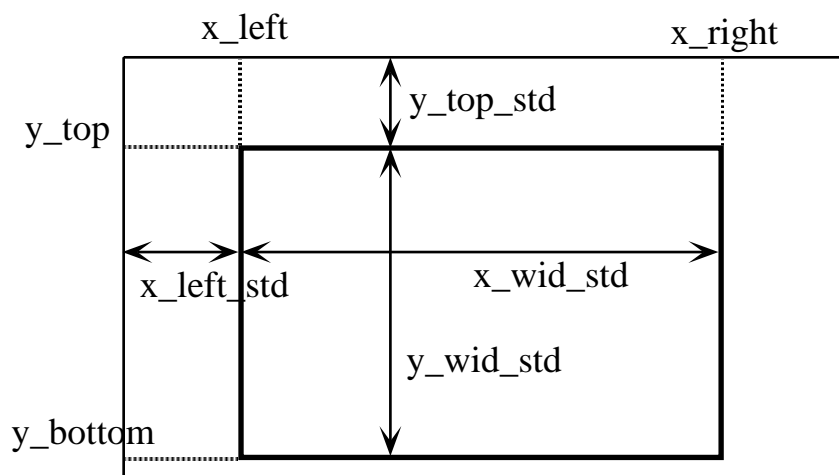
書式 `g_def_scale (scale, x_left, x_right, y_bottom, y_top, x_left_std, y_top_std, x_wid_std, y_wid_std);`

入力パラメータ

<code>scale</code>	整数	スケール番号。50 個まで仮想座標系を導入することができる。(0 <code>scale</code> 49)
<code>x_left</code>	実数	長方形の左辺の仮想座標系における x 座標
<code>x_right</code>	実数	長方形の右辺の仮想座標系における x 座標
<code>y_bottom</code>	実数	長方形の下辺の仮想座標系における y 座標
<code>y_top</code>	実数	長方形の上辺の仮想座標系における y 座標
<code>x_left_std</code>	実数	長方形の左辺の標準 x 座標
<code>y_top_std</code>	実数	長方形の上辺の標準 y 座標
<code>x_wid_std</code>	実数	長方形の標準面上での x 軸方向の幅
<code>y_wid_std</code>	実数	長方形の標準面上での y 軸方向の幅

説明

下図のように適当な長方形を考えて、上記各パラメータを定めることにより仮想座標系を決める。



`g_sel_scale` 仮想座標系を選択する

書式 `g_sel_scale (scale);`

入力パラメータ

`scale` 整数 スケール番号

`g_line_color` 線の色を変更する

書式 `g_line_color (color);`

入力パラメータ

`color` 整数 色番号

説明

カラーディスプレイの場合、下記の8色が使える。

color		カラーディスプレイ	白黒ディスプレイ
番号	マクロ		
0	G_BLACK	黒	黒
1	G_RED	赤	黒
2	G_GREEN	緑	黒
3	G_BLUE	青	黒
4	G_MAGENTA	マゼンタ	黒
5	G_YELLOW	黄	黒
6	G_CYAN	シアン	黒
7	G_WHITE	白	白

`g_line_width` 線の太さを変更する

書式 `g_line_width (width);`

入力パラメータ

`width` 整数 線の太さを表わす番号

説明

番号と線の太さの対応はシステムにより多少違うと思われる。通常は0から3までの整数を用いる。あまり太すぎる線は描画に時間がかかる。0と1では線の太さは変わらないが、0が最も描画が速い。

`g_line_type` 線の種類を変更する

書式 `g_line_type (type);`

入力パラメータ

`type` 整数 線種を表わす番号

説明

type		線
番号	マクロ	
0	G_LINE_SOLID	—————
1	G_LINE_DOTS
2	G_LINE_DASHED
3	G_LINE_LONG_DASHED	- - - - -
4	G_LINE_THIN_DOTS
5	G_LINE_DOT_DASHED
6	G_LINE_D_DOT_DASHED

`g_area_color` ぬりつぶしの色 (パターン) を変更する

書式 `g_area_color (color);`

入力パラメータ

`color` 整数 色 (パターン) 番号

説明

ウインドウへの出力に際しては色を指定するが、それぞれの色はプリンタ出力時にはなんらかのパターンに対応している。色番号と色との対応は線の場合と同じである。

`g_marker_color` マーカーの色を変更する

書式 `g_marker_color (color);`

入力パラメータ

`color` 整数 色番号

説明

色番号と色との対応は線の場合と同じである。

`g_marker_size` マーカーの大きさを変更する

書式 `g_marker_size (size);`

入力パラメータ

`size` 整数 マーカーの大きさを示す数

説明

マーカーの大きさはだいたい直径 `size` mm ぐらいである。
`size = 0` のときマーカーはポイントになる。

`g_marker_type` マーカーの種類を変更する

書式 `g_marker_type (type);`

入力パラメータ

`type` 整数 マーカー番号

説明

type		マーカー
番号	マクロ	
0	G_MARKER_STAR	*
1	G_MARKER_CIRC	
- 1	G_MARKER_F_CIRC	
2	G_MARKER_BOX	
- 2	G_MARKER_F_BOX	
3	G_MARKER_TRIANGLE	
- 3	G_MARKER_F_TRIANGLE	
4	G_MARKER_PLUS	+
- 4	G_MARKER_X	×

注意

マーカーのサイズが0のときは、マーカーはポイントになるので、マーカー番号は意味を持たない。

`g_text_color` テキストの色を変更する

書式 `g_text_color (color);`

入力パラメータ

`color` 整数 色番号

説明

色番号との対応は線の場合と同じである。

`g_text_font` テキストのフォントを変更する

書式 `g_text_font (font);`

入力パラメータ

`font` 整数 フォント番号

説明

font		フォント
番号	マクロ	
0	<code>G_FONT_TIMES_8</code>	times 8
1	<code>G_FONT_TIMES_12</code>	times 12
2	<code>G_FONT_TIMES_18</code>	times 18
3	<code>G_FONT_TIMES_24</code>	times 24

現在のところ、メタファイルにはフォントの区別は出力されない。

`g_def_line` 線の属性セットを定義する

書式 `g_def_line (line, color, width, type);`

入力パラメータ

<code>line</code>	整数	属性セット番号
<code>color</code>	整数	色番号
<code>width</code>	整数	線の太さを示す数
<code>type</code>	整数	線種番号

`g_def_area` ぬりつぶしの属性セットを定義する

書式 `g_def_area (area, color);`

入力パラメータ

<code>area</code>	整数	属性セット番号
<code>color</code>	整数	色 (パターン) 番号

説明

属性セットといっても、この場合は属性は1つだけである。

`g_def_marker` マーカーの属性セットを定義する

書式 `g_def_marker (marker, color, size, type);`

入力パラメータ

<code>marker</code>	整数	属性セット番号
<code>color</code>	整数	色番号
<code>size</code>	整数	マーカーのだいたいの直径 (mm)
<code>type</code>	整数	マーカー番号

`g_def_text` テキストの属性セットを定義する

書式 `g_def_text (text, color, font);`

入力パラメータ

<code>text</code>	整数	属性セット番号
<code>color</code>	整数	色番号
<code>font</code>	整数	フォント番号

`g_sel_line` 線の属性セットを選択する

書式 `g_sel_line (line);`

入力パラメータ

`line` 整数 線の属性セット番号

`g_sel_area` ぬりつぶしの属性セットを選択する

書式 `g_sel_area (area);`

入力パラメータ

`area` 整数 ぬりつぶしの属性セット番号

`g_sel_marker` マーカーの属性セットを選択する

書式 `g_sel_marker (marker);`

入力パラメータ

`marker` 整数 マーカーの属性セット番号

`g_sel_text` テキストの属性セットを選択する

書式 `g_sel_text (text);`

入力パラメータ

`text` 整数 テキストの属性セット番号

`g_move` カレントポジションを移動する

書式 `g_move (x, y);`

入力パラメータ

<code>x</code>	実数	移動先の点のカレント座標系における <code>x</code> 座標を表わす。
<code>y</code>	実数	移動先の点のカレント座標系における <code>y</code> 座標を表わす。

`g_plot` カレントポジションから指定された点まで線分をひく

書式 `g_plot (x, y);`

入力パラメータ

<code>x</code>	実数	指定点のカレント座標系における <code>x</code> 座標を表わす。
<code>y</code>	実数	指定点のカレント座標系における <code>y</code> 座標を表わす。

説明

ここでいうカレントポジションという概念は、`g_move` と `g_plot` だけに関与したものであり、他の描画関数には影響を受けない。

`g_box` 長方形を描いたり、中をぬりつぶしたりする

書式 `g_box (x_left, x_right, y_bottom, y_top, edge, fill);`

入力パラメータ

<code>x_left</code>	実数	長方形の左辺のカレント座標系における x 座標
<code>x_right</code>	実数	長方形の右辺のカレント座標系における x 座標
<code>y_bottom</code>	実数	長方形の下辺のカレント座標系における y 座標
<code>y_top</code>	実数	長方形の上辺のカレント座標系における y 座標
<code>edge</code>	整数	G_YES のときふちを描き、G_NO のときは描かない。
<code>fill</code>	整数	G_YES のとき中をぬり、G_NO のときはぬらない。

説明

数値	マクロ	意味	
		edge	fill
1	G_YES	ふちを描く	中をぬる
0	G_NO	ふちを描かない	中をぬらない

`g_circle` 円を描いたり、中をぬりつぶしたりする

書式 `g_circle (x, y, radius, edge, fill);`

入力パラメータ

<code>x</code>	実数	円の中心のカレント座標系における x 座標
<code>y</code>	実数	円の中心のカレント座標系における y 座標
<code>radius</code>	実数	円の半径を x 軸方向のスケールで表わしたもの
<code>edge</code>	整数	G_YES のときふちの描き、G_NO のときは描かない。
<code>fill</code>	整数	G_YES のとき中をぬり、G_NO のときはぬらない。

`g_polygon` 多角形を描いたりぬりつぶしたりする

書式 `g_polygon (x, y, n, edge, fill);`

入力パラメータ

x	実数列	多角形の頂点のカレント座標系におけるx座標からなる実数値配列
y	実数列	多角形の頂点のカレント座標系におけるy座標からなる実数値配列
n	整数	多角形の頂点の数 (n 2 5 6)
edge	整数	G_YES のときふちを描き、G_NO のときは描かない。
fill	整数	G_YES のとき中をぬり、G_NO のときはぬらない。

説明

配列 x , y のサイズは n 以上でなくてはならない。

`g_polyline` 与えられた点列を線分で連続的に結ぶ

書式 `g_polyline (x, y, n);`

入力パラメータ

x	実数列	カレント座標系におけるx座標の実数値配列
y	実数列	カレント座標系におけるy座標の実数値配列
n	実数	点列を構成する点の数

説明

配列 x , y のサイズは n 以上でなくてはならない。

`g_data_plot` 与えられた配列を用いてグラフをかく

書式 `g_data_plot (x_left, x_right, y, n);`

入力パラメータ

<code>x_left</code>	実数	カレント座標系における x 軸の左端の座標
<code>x_right</code>	実数	カレント座標系における x 軸の右端の座標
<code>y</code>	実数列	カレント座標系における y 座標の実数値配列
<code>n</code>	整数	データ数

説明

配列 `y` のサイズは `n` 以上でなくてはならない。この関数は `x` 方向の分割幅が一定の時としてグラフの描画を行なう。

`g_m a r k e r` マーカーを描く

書式 `g_marker (x, y);`

入力パラメータ

<code>x</code>	実数	マーカーの中心点のカレント座標系における <code>x</code> 座標
<code>y</code>	実数	マーカーの中心点のカレント座標系における <code>y</code> 座標

`g_t e x t` 文字列をかく

書式 `g_text (x_std, y_std, string);`
`call g_text (x_std, y_std, string, length)`

入力パラメータ

<code>x_std</code>	実数	文字列の先頭文字の左下端の位置の標準 <code>x</code> 座標
<code>y_std</code>	実数	文字列の先頭文字の左下端の位置の標準 <code>y</code> 座標
<code>string</code>	文字列	描こうとするテキストのはいった文字列
<code>length</code>	整数	<code>string</code> の文字数

説明

描画関数の中でこれだけは、標準座標によって位置を指定する。
C 言語からコールする場合は `length` は不要だが、FORTRAN からコールする場合は `length` が必要である。

`g_contln` 等高線を描く

書式 `g_contln (x_left, x_right, y_bottom, y_top, array,
number_x, number_y, height);`

入力パラメータ

<code>x_left</code>	実数	枠となる長方形の左辺のカレント座標系における x 座標
<code>x_right</code>	実数	枠となる長方形の右辺のカレント座標系における x 座標
<code>y_bottom</code>	実数	枠となる長方形の底辺のカレント座標系における y 座標
<code>y_top</code>	実数	枠となる長方形の上辺のカレント座標系における y 座標
<code>array</code>	実数列	サイズ <code>number_x · number_y</code> の 2 次元実数値配列。 等高線を描くためのデータを入力。
<code>number_x</code>	整数	2 次元配列 <code>array</code> の x 方向のサイズ
<code>number_y</code>	整数	2 次元配列 <code>array</code> の y 方向のサイズ
<code>height</code>	実数	描きたい等高線のレベル

`g_bird_view` $z = f(x, y)$ のグラフの鳥瞰図を隠線処理をせずに描く

書式 `g_bird_view` (`x_wid`, `y_wid`, `z_wid`, `z_bottom`, `z_top`, `distance`, `theta`, `phi`, `x_left_std`, `y_top_std`, `x_wid_std`, `y_wid_std`, `array`, `number_x`, `number_y`, `direction`);

入力パラメータ

<code>x_wid</code>	実数	グラフがおさまる直方体の x 方向の幅
<code>y_wid</code>	実数	グラフがおさまる直方体の y 方向の幅
<code>z_wid</code>	実数	グラフがおさまる直方体の z 方向の幅
<code>z_bottom</code>	実数	直方体の底面に対応するデータ値
<code>z_top</code>	実数	直方体の上面に対応するデータ値
<code>distance</code>	実数	直方体の中心と視点との距離
<code>theta</code>	実数	x y 平面において、視線を x y 平面に射影した直線と、 y 軸とのなす角 (単位は度)
<code>phi</code>	実数	x y 平面と視線とのなす角 (単位は度)
<code>x_left_std</code>	実数	点 A の標準 x 座標
<code>y_top_std</code>	実数	点 A の標準 y 座標
<code>x_wid_std</code>	実数	標準面上における AD の長さ
<code>y_wid_std</code>	実数	標準面上における AB の長さ
<code>array</code>	実数列	サイズ <code>number_x</code> \cdot <code>number_y</code> の実数値 2 次元配列 グラフを描くためのデータを入力
<code>number_x</code>	整数	2 次元配列 <code>array</code> の x 方向のサイズ
<code>number_y</code>	整数	2 次元配列 <code>array</code> の y 方向のサイズ
<code>direction</code>	整数	正のとき y 軸正方向は遠ざかる方向 負のとき y 軸正方向は近づく方向

説明

37 ページの図を参照

<p><code>g_hidden</code> $z = f(x, y)$ のグラフの鳥瞰図を隠線処理をして描く</p>
--

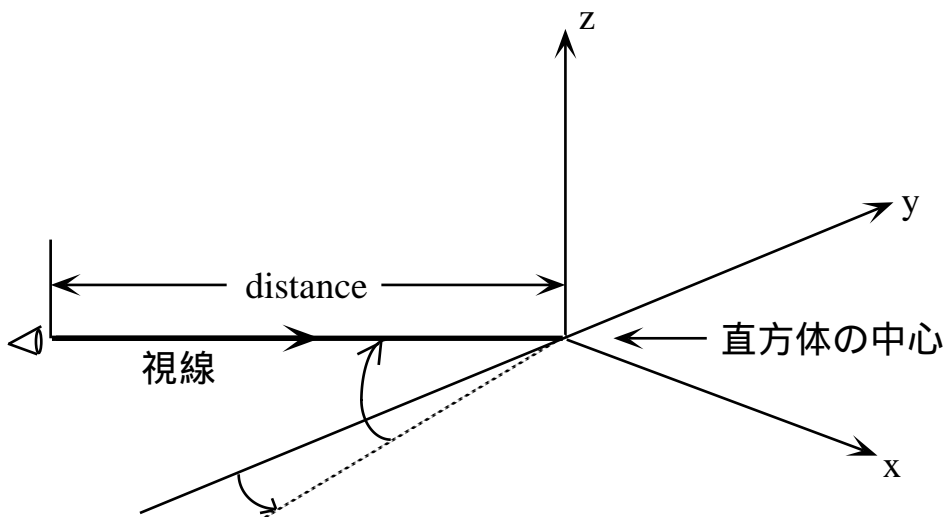
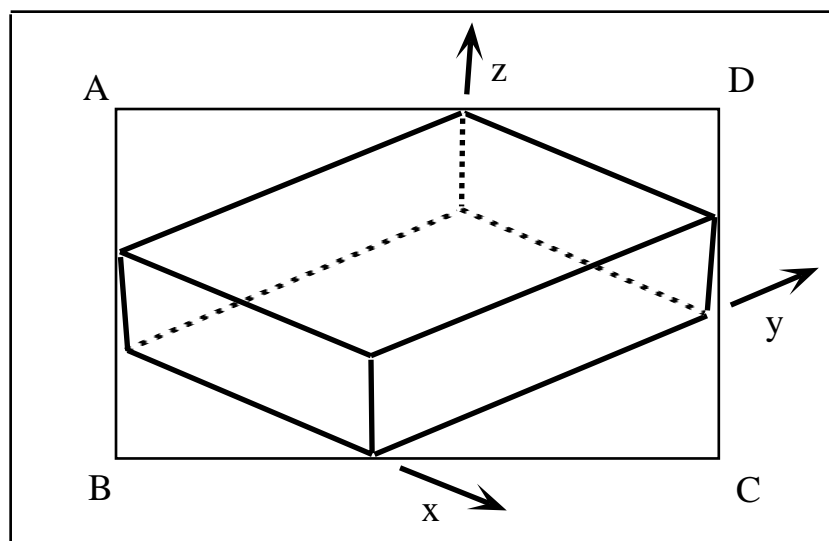
書式 `g_hidden (x_wid, y_wid, z_wid, z_bottom, z_top, distance, theta, phi, x_left_std, y_top_std, x_wid_std, y_wid_std, array, number_x, number_y, direction, side_paint, x_intv, y_intv);`

入力パラメータ

<code>x_wid</code>	実数	グラフがおさまる直方体の x 方向の幅
<code>y_wid</code>	実数	グラフがおさまる直方体の y 方向の幅
<code>z_wid</code>	実数	グラフがおさまる直方体の z 方向の幅
<code>z_bottom</code>	実数	直方体の底面に対応するデータ値
<code>z_top</code>	実数	直方体の上面に対応するデータ値
<code>distance</code>	実数	直方体の中心と視点との距離
<code>theta</code>	実数	x y 平面において、視線を x y 平面に射影した直線と、y 軸とのなす角 (単位は度)
<code>phi</code>	実数	x y 平面と視線とのなす角 (単位は度)
<code>x_left_std</code>	実数	点 A の標準 x 座標
<code>y_top_std</code>	実数	点 A の標準 y 座標
<code>x_wid_std</code>	実数	標準面上における AD の長さ
<code>y_wid_std</code>	実数	標準面上における AB の長さ
<code>array</code>	実数列	サイズ <code>number_x · number_y</code> の実数値 2 次元配列 グラフを描くためのデータを入力
<code>number_x</code>	整数	2 次元配列 <code>array</code> の x 方向のサイズ
<code>number_y</code>	整数	2 次元配列 <code>array</code> の y 方向のサイズ
<code>direction</code>	整数	正のとき y 軸正方向に向かってインデックスが増える 負のとき y 軸正方向に向かってインデックスが減る
<code>side_paint</code>	整数	1 と 3 のとき両側面をぬり、2 と 3 のとき前後面をぬる 0 のときぬらない
<code>x_intv</code>	整数	y 軸方向の描線の間隔
<code>y_intv</code>	整数	x 軸方向の描線の間隔

説明

37 ページの図を参照



`g_fake_bird_view` $z = f(x, y)$ のグラフの
鳥瞰図もどきを隠線処理をせずに描く

書式 `g_fake_bird_view` (`z_bottom`, `z_top`, `x_ratio`, `y_ratio`,
`x_left_std`, `y_top_std`, `x_wid_std`, `y_wid_std`,
`array`, `number_x`, `number_y`, `direction`);

入力パラメータ

<code>z_bottom</code>	実数	z 軸の下限の座標
<code>z_top</code>	実数	z 軸の上限の座標
<code>x_ratio</code>	実数	$B F / B C$
<code>y_ratio</code>	実数	$B E / B A$
<code>x_left_std</code>	実数	点 A の標準 x 座標
<code>y_top_std</code>	実数	点 A の標準 y 座標
<code>x_wid_std</code>	実数	標準面上における A D の長さ
<code>y_wid_std</code>	実数	標準面上における A B の長さ
<code>array</code>	実数列	サイズ <code>number_x</code> ・ <code>number_y</code> の実数値 2 次元配列。 グラフを描くためのデータを入力
<code>number_x</code>	整数	2 次元配列 <code>array</code> の x 方向のサイズ
<code>number_y</code>	整数	2 次元配列 <code>array</code> の y 方向のサイズ
<code>direction</code>	整数	正のとき y 軸正方向は遠ざかる方向 負のとき y 軸正方向は近づく方向

説明

40 ページの図を参照

`g_fake_hidden` $z = f(x, y)$ のグラフの鳥瞰図もどきを
隠線処理をして描く

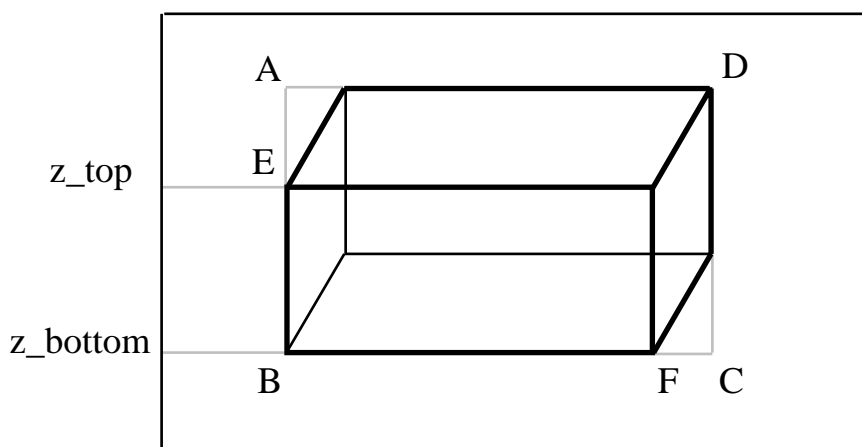
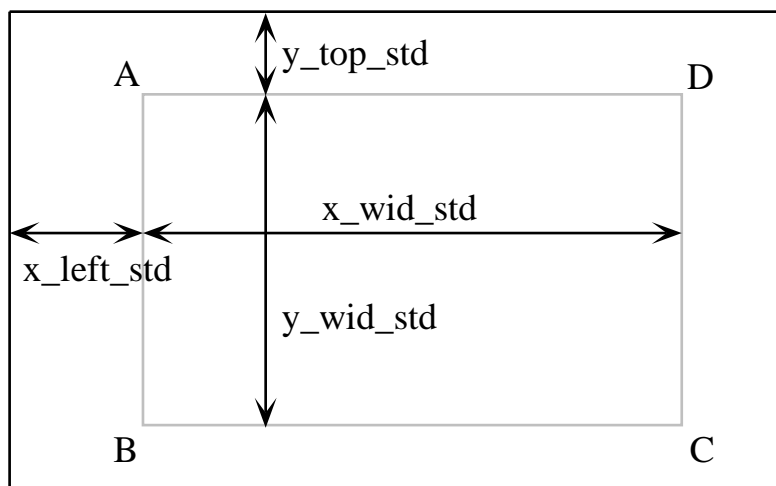
書式 `g_fake_hidden (z_bottom, z_top, x_ratio, y_ratio,
x_left_std, y_top_std, x_wid_std, y_wid_std,
array, number_x, number_y,
direction, side_paint, x_intv, y_intv);`

入力パラメータ

<code>z_bottom</code>	実数	z 軸の下限の座標
<code>z_top</code>	実数	z 軸の上限の座標
<code>x_ratio</code>	実数	$B F / B C$
<code>y_ratio</code>	実数	$B E / B A$
<code>x_left_std</code>	実数	点 A の標準 x 座標
<code>y_top_std</code>	実数	点 A の標準 y 座標
<code>x_wid_std</code>	実数	標準面上における A D の長さ
<code>y_wid_std</code>	実数	標準面上における A B の長さ
<code>array</code>	実数列	サイズ $\text{number}_x \cdot \text{number}_y$ の実数値 2 次元配列。 グラフを描くためのデータを入力
<code>number_x</code>	整数	2 次元配列 <code>array</code> の x 方向のサイズ
<code>number_y</code>	整数	2 次元配列 <code>array</code> の y 方向のサイズ
<code>direction</code>	整数	正のとき y 軸正方向は遠ざかる方向。 負のとき y 軸正方向は近づく方向。
<code>side_paint</code>	整数	1 と 3 のとき両側面をぬり、2 と 3 のとき前後面をぬる 0 のときぬらない
<code>x_intv</code>	整数	y 軸方向の描線の間隔
<code>y_intv</code>	整数	x 軸方向の描線の間隔

説明

40 ページの図を参照



上図の直方体の中に $z=f(x,y)$ のグラフが描かれる

`g_sleep` 描画後、絵を消さずに表示する

書式 `g_sleep (time);`

入力パラメータ

`time` 実数 絵を消さずに表示することを保証する時間 (秒)

説明

この関数がコールされるまでに描かれるべき絵の描画が完了してから、`time` 秒間絵を表示する。もちろん、`g_cls` や `g_term` をコールすることにより、描かれた絵を消さない限りは `time` 秒間を過ぎても、絵は表示され続ける。

`time` が負の数の場合は絵が表示されているウインドウ内でマウスボタンがクリックされるまで絵は表示され続ける。

`g_sformat` 変数を指定された書式で表わし文字列として返す

書式 `call g_sformat (string, length, format, variable)`

出力パラメータ

<code>string</code>	文字列	<code>variable</code> を <code>format</code> に従って書き表した結果が返される文字列
<code>length</code>	整数	上記文字列の長さが返される

入力パラメータ

<code>format</code>	文字列	<code>variable</code> を表現するための書式からなる文字列 (書式指定はC言語に従う)
<code>variable</code>	不定	変数(型は特に指定しない)

説明

これは FORTRAN にのみ必要な関数で、C言語であれば `sprintf` を使えばよい。GLSC においては通常 `g_text` とともに用いられる。

例

```
call g_sformat (string, length, 'x = %f', x)
call g_text (50.0, 50.0, string, length)
```

C言語では、次のようになる。

```
sprintf (string, "x = %f", x);
g_text (50.0, 50.0, string);
```

注意

`format` の中で%は必ず1つあり、かつ2つ以上あってはならない

g_out メタファイル図形出力プログラム

<形 式>

g_out [-clvfwtm] [縦フレーム数, 横フレーム数] [枠の線幅]
[出力倍率] 入力ファイル名

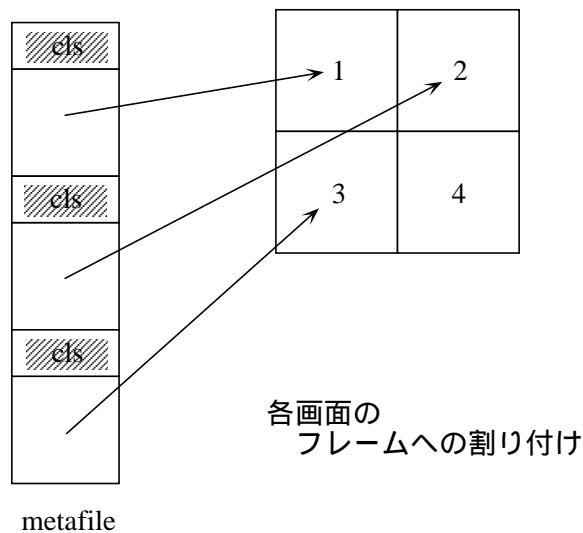
<機 能>

指定されたメタファイルの内容をレーザーショット・コマンド形式、ポスト・スクリプト形式、Illustrator 形式のいずれかで、ファイルに出力する。出力ファイルの名称は、出力形式に応じて各々'.ls'、'.ps'、'.ptnn' (nn は 00 ~ 99) を入力メタファイル名に付加したものとなる。

出力紙面の書式は、A 4 ヨコまたは A 4 タテのどちらかとする。メタファイルに複数の図面が出力されているとき、出力紙面をフレームに区切って、指定された方向で、順次図面の割り付けを行うことができる(図 1)。このとき、図面の原点とフレームの原点を一致させ、(指定された変倍を行って)出力するが、フレームからはみ出し部分については、クリッピングを行わない。

フレーム数の指定がないときは、紙面全体をひとつのフレームとみなし、複数の図面は改頁して順次出力される。

(図 1)



<オプション>

* 出力形式の指定

-i ... Illustrator 形式で出力する。メタファイルの内容を各フレームごとに、~.i00 から順に番号をつけて、出力ファイルとして生成する。

-l ... レーザーショット・コマンド形式で出力する。

-i, -l の指定がなければ、ポスト・スクリプト形式で出力する。

* 書式の指定

-v ... A 4 の紙面を縦長に使用することを指示する。

この指定がなければ横長に使用する。

このオプションを指定したとき、ピクセル・フォントを用いるテキスト・パスが正しい方向に出力されない。

* フレーム数の指定

-f ... このオプション指定の後に、縦横のフレーム数の指定が続くことを示す。

フレーム数の指定がないときは、紙全体がひとつのフレームとみなされる。

縦または横のフレーム数は、1 以上でなければならない。さらに、紙面上のフレーム総数(縦 × 横)は、16 以下でなければならない。

* 枠の有無とその線幅の指定

-w ... フレーム枠の線幅を指定することを示す。

この指定がなければフレーム枠を描かない。

線幅は0以上でなければならない。0を指定したときは、出力装置の持つ最も細い線で出力する。線幅として、負の値が指定された時は、線幅指定がなかったものとみなし、枠を描かない。

* フレーム割り付けの方向

-t ... メタファイル中の図形をフレームに割り付けるとき、割り付けを左上の枠から下に向かって順に行うことを指定する。

この指定がないときは、左上から右に向かって順に割り付けを行う。

(例 参照)

(例)

1	3	5
2	4	6

- t のあるとき

1	2	3
4	5	6

- t のないとき

* 変倍率の指定

-m ... 変倍率の指定が続くことを示す。

変倍率は0.0より大きく50.0以下の値で指定しなければならない。

この指定がないときは、変倍率は1.0とみなされる。

レーザーショットではテキストは変倍されないが、マーカは変倍される。

<使用例>

```
sample% g_out -lfwm 2,3 0 1.2 metafile
```

- ・出力装置 レーザーショット
- ・フレーム数 縦2 x 横3
- ・フレーム枠 最も細い線（線幅0）で描く
- ・割り付け方向 横方向
- ・変倍率 1.2倍
- ・入力メタファイル metafile
- ・出力ファイル metafile.ls

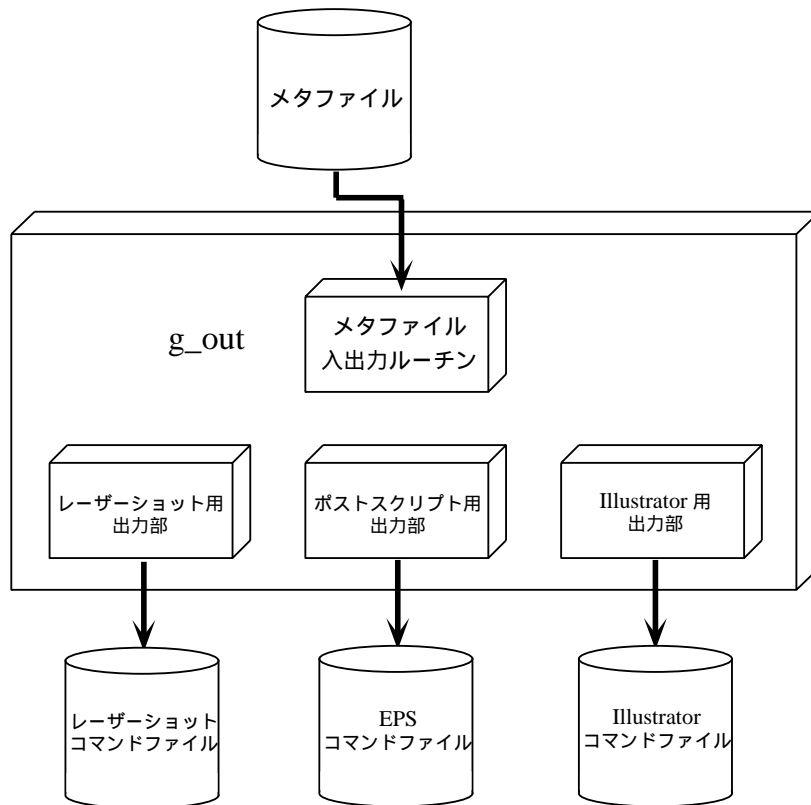
フレームの割り付け

1	2	3
4	5	6

* 処理条件の指定形式に誤りがあるときは、下記のメッセージを出力し、正しい形式の指定を促す。

```
Usage : g_out [-ilvfwtm] [frame_y, frame_x][width] [mag] metafile_name
```

<概要図>



<レーザーショットでの属性の表現と対応>

属性指定ルーチン	メタファイルでの入力	レーザーショットでの出力
m_line_width(n)	n < 0 (エラー) n = 0, 1 n = 2 n = 3 n ≥ 4	細線 (1 ドット幅) (デフォルト値) 中細線 (3 ドット幅) 中太線 (5 ドット幅) 太線 (7 ドット幅)
m_line_type(n)	n = 0, 7 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6	実線 (デフォルト値) 短点線 短鎖線 長鎖線 長点線 一点鎖線 二点鎖線
m_marker_type(n)	n = - 4 n = - 3 n = - 2 n = - 1 n = 0 n = 1 n = 2 n = 3 n = 4	× * (デフォルト値) +
m_marker_size(s)	s < 0.0 (エラー) 0.0 ≤ s < 3.0 s ≤ 64.0	最小サイズ (9 ドット幅) (デフォルト値) 3 * S ドット幅
m_line_color(n)		表現できない
m_marker_color(n)		表現できない
m_text_color(n)		表現できない
m_area_color(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7	ベタ黒 左斜線 右斜線 淡いハーフトーン 濃いハーフトーン 縦線 横線 ベタ白 (デフォルト値)

- * m_line_color, m_marker_color, m_text_color は、レーザーショットで表現できないため、g_out.c では、データの読み込みを行うだけで何もしない。
- * m_area_color については、塗りつぶしパターンで代用する。
- * 上記の表以外の値で指定されたときは、各々の属性のデフォルト値にセットされる。

< E P S ファイル形式での属性の表現と対応 >

属性指定ルーチン	メタファイルでの入力	レーザーライターでの出力
m_line_width (n)	n < 0 (エラー) n = 0, 1 n = 2 n = 3 n ≥ 4	細線 (0.2point 幅) (デフォルト値) 中細線 (0.6point 幅) 中太線 (1.0point 幅) 太線 (1.4point 幅)
m_line_type(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6	実線 (デフォルト値) 短点線 短鎖線 長鎖線 長点線 一点鎖線 二点鎖線
m_line_color(n)		表現できない
m_text_color(n)		表現できない
m_area_color(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7	ベタ黒 (0) (濃い) (0.14) ⋮ (0.28) グレイの濃淡 (0.42) ⋮ (0.56) ⋮ (0.71) (薄い) (0.85) ベタ白 (1) (デフォルト値)

- * m_line_color, m_text_color は、初版では表現できない。
- * 初版では m_area_color については、setgray で代用する。
- * 上記の表以外の値で指定されたときは、各々の属性のデフォルト値にセットされる。

< Illustrator88 ファイル形式での属性の表現と対応 >

属性指定ルーチン	メタファイルでの入力	Illustrator での出力
m_line_width (n)	n < 0 (エラー) n = 0, 1 n = 2 n = 3 n ≥ 4	ペン幅 細線 (0.2 point 幅) (デフォルト値) 中細線 (0.6 point 幅) 中太線 (1.0 point 幅) 太線 (1.4 point 幅)
m_line_type(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6	実線 (デフォルト値) 短点線 短鎖線 長鎖線 長点線 一点鎖線 二点鎖線
m_line_color(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7	BLACK (デフォルト値) RED GREEN BLUE MAGENTA YELLOW CYAN WHITE
m_text_color(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7	BLACK (デフォルト値) RED GREEN BLUE MAGENTA YELLOW CYAN WHITE
m_area_color(n)	n = 0 n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7	BLACK (デフォルト値) RED GREEN BLUE MAGENTA YELLOW CYAN WHITE

* 上記の表以外の値で指定された時は、各々の属性のデフォルト値にセットされる。

Test Programs

and

Drawn Graphs

```

/*
 * Test program for GLSC library
 *
 * C language version
 */

#include <math.h>
#include <glsc.h>

#define MAX (5000)
#define MAX_X (40)
#define MAX_Y (40)
#define PI (3.1415926545)
#define N (30)

main ()
{
    char text[256];
    int i, j;
    G_REAL x[MAX + 1], y[MAX + 1], a, dt, t;
    G_REAL array[MAX_X + 1][MAX_Y + 1], dx, dy, xx, yy;

    /* Initialization and definitions */
    g_init ("Meta", 250.0, 160.0);
    g_device (G_BOTH);
    g_def_scale (0, 0.0, 1.0, 0.0, 1.0, 30.0, 70.0, 100.0, 72.0);
    g_def_scale (1, 0.0, 1.0, 0.0, 1.0, 30.0, 30.0, 100.0, 100.0);
    g_def_scale (2, -1.0, 1.0, -1.0, 1.0, 20.0, 30.0, 80.0, 80.0);
    g_def_scale (3, 0.0, N * PI, -1.0, 1.0, 110.0, 30.0, 130.0, 80.0);
    g_def_scale (4, -1.0, 1.0, -1.0, 1.0, 30.0, 30.0, 100.0, 100.0);
    g_def_line (0, G_BLACK, 0, G_LINE_SOLID);
    g_def_line (1, G_RED, 0, G_LINE_SOLID);
    g_def_line (2, G_GREEN, 0, G_LINE_DASHED);
    g_def_line (3, G_BLUE, 0, G_LINE_SOLID);
    g_def_text (0, G_BLACK, G_FONT_TIMES_8);
    g_def_text (1, G_RED, G_FONT_TIMES_12);
    g_def_text (2, G_GREEN, G_FONT_TIMES_18);
    g_def_text (3, G_BLUE, G_FONT_TIMES_24);

```

```

/* Hinomaru and text */
g_cls ();
for (i = 0; i < 4; i ++)
{
    g_sel_text (i);
    g_text (30.0, 20.0 + 10.0 * i, "Test Hinomaru no Hata.");
}
g_sel_scale (0);
g_area_color (G_WHITE);
g_box (0.0, 1.0, 0.0, 1.0, G_YES, G_YES);
g_area_color (G_RED);
g_circle (0.5, 0.5, 0.18, G_NO, G_YES);

for (i = 0; i < 10; i ++)
{
    a = i * 0.2;
    sprintf (text, "a = %f", a);
    g_text (160.0, 20.0 + 10.0 * i, text);
}
g_sleep (3.0);

/* Various lines and boxes */
g_cls ();
g_sel_scale (1);
g_box (0.0, 1.0, 0.0, 1.0, G_YES, G_NO);
g_area_color (G_GREEN);
g_box (0.0, 0.5, 0.5, 1.0, G_YES, G_YES);
g_area_color (G_RED);
g_box (0.5, 1.0, 0.0, 0.5, G_YES, G_YES);
g_area_color (G_BLUE);
g_box (0.5, 1.0, 0.5, 1.0, G_YES, G_YES);

for (i = 0; i < 8; i ++)
{
    g_line_width (2 * (i + 1));
    g_line_type (i);
    g_line_color (i);
    g_move (0.1, 0.1 + i / 7.0 * 0.8);
    g_plot (0.9, 0.1 + i / 7.0 * 0.8);
}

```

```

}
for (i = 0; i < 8; i ++)
{
    g_line_width (2 * (7 - i + 1));
    g_line_type (7 - i);
    g_line_color (7 - i);
    g_move (0.1 + (7 - i) / 7.0 * 0.8, 0.1);
    g_plot (0.1 + (7 - i) / 7.0 * 0.8, 0.9);
}
g_sleep (3.0);

/* Graphs */
a = -0.04;
dt = N * PI / MAX;

for (i = 0; i <= MAX; i ++)
{
    t = dt * i;
    x[i] = exp (a * t) * cos (t);
    y[i] = exp (a * t) * sin (t);
}

g_cls ();
g_sel_scale (2);
g_sel_line (0);
g_box (-1.0, 1.0, -1.0, 1.0, G_YES, G_NO);
g_move (-1.0, 0.0);
g_plot (1.0, 0.0);
g_move (0.0, -1.0);
g_plot (0.0, 1.0);
g_sel_line (1);
g_polyline (x, y, MAX + 1);
g_sel_scale (3);
g_sel_line (0);
g_box (0.0, N * PI, -1.0, 1.0, G_YES, G_NO);
g_move (0.0, 0.0);
g_plot (N * PI, 0.0);
g_sel_line (2);
g_data_plot (0.0, N * PI, x, MAX + 1);

```

```

g_sel_line (3);
g_data_plot (0.0, N * PI, y, MAX + 1);

g_marker_size (5);
g_marker_color (G_RED);
g_marker_type (2);
for (i = 0; i <= 10; i ++)
{
    g_marker (i * dt * (MAX / 10),
              (x[i * (MAX / 10)] + y[i * (MAX / 10)]) / 2);
}
g_sleep (3.0);

/* Markers */
dt = 2 * PI / 6;
for (i = 0; i < 6; i ++)
{
    t = dt * i;
    x[i] = cos (t);
    y[i] = sin (t);
}

g_cls ();
g_sel_scale (4);
g_area_color (G_WHITE);
g_line_width (3);
g_line_color (G_RED);
g_circle (0.0, 0.0, 1.2, G_YES, G_YES);
g_line_color (G_CYAN);
g_polygon (x, y, 6, G_YES, G_YES);

g_marker_color (G_BLACK);
g_marker_size (10);
for (i = -4; i <= 4; i ++)
{
    g_marker_type (i);
    g_marker (0.5 * cos (2 * PI / 9 * i), 0.5 * sin (2 * PI / 9 * i));
    g_marker (cos (2 * PI / 9 * i), sin (2 * PI / 9 * i));
}

```

```

g_sleep (3.0);

/* Contour lines and bird eye's view */
dx = PI * 4 / MAX_X;
dy = PI * 3 / MAX_Y;
for (j = 0; j <= MAX_Y; j ++)
{
    yy = dy * (j - MAX_Y / 2);
    for (i = 0; i <= MAX_X; i ++)
    {
        xx = dx * (i - MAX_X / 2);
        array[i][j] = sin (xx - 0.3) * cos (yy - 0.2);
    }
}

g_cls ();
g_sel_scale (4);
g_sel_line (0);
g_line_width (1);
g_box (-1.0, 1.0, -1.0, 1.0, G_YES, G_NO);
for (i = 1; i < 7; i ++)
{
    g_line_color (i);
    g_contln (-1.0, 1.0, -1.0, 1.0, (G_REAL *) array,
              MAX_X + 1, MAX_Y + 1, 0.32 * i - 1.12);
}
g_sleep (3.0);

g_cls ();
g_bird_view (1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,
             150.0, 100.0, (G_REAL *) array, MAX_X + 1, MAX_Y + 1, 1);
g_sleep (3.0);

g_cls ();
g_fake_bird_view (-1.0, 1.0, 0.8, 0.6, 20.0, 20.0, 150.0, 100.0,
                  (G_REAL *) array, MAX_X + 1, MAX_Y + 1, 1);
g_sleep (3.0);

g_cls ();

```



```
g_hidden (1.0, 1.0, 0.4, -1.0, 1.0, 5.0, 25.0, 20.0, 20.0, 20.0,  
          150.0, 100.0, (G_REAL *) array, MAX_X + 1, MAX_Y + 1,  
          1, G_SIDE_NONE, 2, 1);  
g_sleep (3.0);  
  
g_cls ();  
g_fake_hidden (-1.0, 1.0, 0.8, 0.6, 20.0, 20.0, 150.0, 100.0,  
              (G_REAL *) array, MAX_X + 1, MAX_Y + 1,  
              1, G_SIDE_NONE, 2, 1);  
g_sleep (3.0);  
  
/* Termination */  
g_term ();  
}
```

```

c
c   Test program for GLSC library
c
c           F77 version
c
c-----
c           program ftest
c-----

include '/usr/include/glsc_ftn.h'

parameter (mmax = 5000, imax = 40, jmax = 40)

dimension x(0:mmax), y(0:mmax)
character text(256)
dimension array(0:imax,0:jmax)

data PI /3.141592654/, N /30/

c Initialization and definitions
  call g_init ('Meta', 4, 250.0, 160.0)
  call g_device (G_BOTH)
  call g_def_scale (0, 0.0, 1.0, 0.0, 1.0,
1                 30.0, 70.0, 100.0, 72.0)
  call g_def_scale (1, 0.0, 1.0, 0.0, 1.0,
1                 30.0, 30.0, 100.0, 100.0)
  call g_def_scale (2, -1.0, 1.0, -1.0, 1.0,
1                 20.0, 30.0, 80.0, 80.0)
  call g_def_scale (3, 0.0, N*PI, -1.0, 1.0,
1                 110.0, 30.0, 130.0, 80.0)
  call g_def_scale (4, -1.0, 1.0, -1.0, 1.0,
1                 30.0, 30.0, 100.0, 100.0)
  call g_def_line (0, G_BLACK, 0, G_LINE_SOLID)
  call g_def_line (1, G_RED, 0, G_LINE_SOLID)
  call g_def_line (2, G_GREEN, 0, G_LINE_DASHED)
  call g_def_line (3, G_BLUE, 0, G_LINE_SOLID)
  call g_def_text (0, G_BLACK, G_FONT_TIMES_8)
  call g_def_text (1, G_RED, G_FONT_TIMES_12)

```

```

    call g_def_text (2, G_GREEN, G_FONT_TIMES_18)
    call g_def_text (3, G_BLUE, G_FONT_TIMES_24)

c   Hinomaru and text
    call g_cls ()
    do 100 i = 0, 3
        call g_sel_text (i)
        call g_text (30.0, 20.0+(10.0*i),
1           'Test Hinomaru no Hata.', 22)
100    continue
    call g_sel_scale (0)
    call g_area_color (G_WHITE)
    call g_box (0.0, 1.0, 0.0, 1.0, G_YES, G_YES)
    call g_area_color (G_RED)
    call g_circle (0.5, 0.5, 0.18, G_NO, G_YES)

    do 200 i = 0, 9
        a = i*0.2
        call g_sformat (text, length, 'a = %f', a)
        call g_text (160.0, 20.0+10.0*i, text, length)
200    continue
    call g_sleep (3.0)

c   Various lines and boxes
    call g_cls ()
    call g_sel_scale (1)
    call g_box (0.0, 1.0, 0.0, 1.0, G_YES, G_NO)
    call g_area_color (G_GREEN)
    call g_box (0.0, 0.5, 0.5, 1.0, G_YES, G_YES)
    call g_area_color (G_RED)
    call g_box (0.5, 1.0, 0.0, 0.5, G_YES, G_YES)
    call g_area_color (G_BLUE)
    call g_box (0.5, 1.0, 0.5, 1.0, G_YES, G_YES)

    do 300 i = 0, 7
        call g_line_width (2*(i+1))
        call g_line_type (i)
        call g_line_color (i)
        call g_move (0.1, 0.1+i/7.0*0.8)

```

```

        call g_plot (0.9, 0.1+i/7.0*0.8)
300    continue

do 400 i = 0, 7
    call g_line_width (2*(7-i+1))
    call g_line_type (7-i)
    call g_line_color (7-i)
    call g_move (0.1+(7-i)/7.0*0.8, 0.1)
    call g_plot (0.1+(7-i)/7.0*0.8, 0.9)
400    continue
    call g_sleep (3.0)

c    Graphs
    a =-0.04
    dt = N*PI/mmax

do 500 i = 0, mmax
    t = dt*i
    x(i) = exp(a*t)*cos(t)
    y(i) = exp(a*t)*sin(t)
500    continue

call g_sel_scale (2)
call g_cls ()
call g_sel_line (0)
call g_box (-1.0, 1.0, -1.0, 1.0, G_YES, G_NO)
call g_move (-1.0, 0.0)
call g_plot ( 1.0, 0.0)
call g_move ( 0.0, -1.0)
call g_plot ( 0.0, 1.0)
call g_sel_line (1)
call g_polyline (x, y, mmax+1)
call g_sel_scale (3)
call g_sel_line (0)
call g_box (0.0, N*PI, -1.0, 1.0, G_YES, G_NO)
call g_move (0.0, 0.0)
call g_plot (N*PI, 0.0)
call g_sel_line (2)
call g_data_plot (0.0, N*PI, x, mmax+1)

```

```

call g_sel_line (3)
call g_data_plot (0.0, N*PI, y, mmax+1)
call g_marker_size (5)
call g_marker_color (G_RED)
call g_marker_type (2)
do 600 i = 0, 10
    call g_marker (i*dt*(mmax/10),
1          (x(i*(mmax/10)) +y(i*(mmax/10)))/2)
600 continue
call g_sleep (3.0)

c Markers
dt = 2*PI/6
do 700 i = 0, 5
    t = dt*i
    x(i) = cos(t)
    y(i) = sin(t)
700 continue

call g_cls ()
call g_sel_scale (4)
call g_area_color (G_WHITE)
call g_line_width (3)
call g_line_color (G_RED)
call g_circle (0.0, 0.0, 1.2, G_YES, G_YES)
call g_line_color (G_CYAN)
call g_polygon (x, y, 6, G_YES, G_YES)

call g_marker_color (G_BLACK)
call g_marker_size (10)
do 800 i = -4, 4
    call g_marker_type (i)
    call g_marker (0.5*cos(2*PI/9*i), 0.5*sin(2*PI/9*i))
    call g_marker (cos(2*PI/9*i), sin(2*PI/9*i))
800 continue
call g_sleep (3.0)

c Contour lines and bird eyes view
dx = PI*4/imax

```

```

dy = PI*3/jmax
do 1000 j = 0, jmax
    yy = dy*(j-jmax/2)
    do 900 i = 0, imax
        xx = dx*(i-imax/2)
        array(i,j) = sin(xx-0.3)*cos(yy-0.2)
900         continue
1000    continue

call g_cls ()
call g_sel_scale (4)
call g_sel_line (0)
call g_box (-1.0, 1.0, -1.0, 1.0, G_YES, G_NO)
do 1100 i = 1, 6
    call g_line_color (i)
    call g_contln (-1.0, 1.0, -1.0, 1.0,
1           array, imax+1, jmax+1, 0.32*i-1.12)
1100    continue
call g_sleep (3.0)

call g_cls ()
call g_bird_view (1.0, 1.0, 0.4, -1.0, 1.0,
1           5.0, 25.0, 20.0, 20.0, 20.0,
2           150.0, 100.0, array, imax+1, jmax+1, 1)
call g_sleep (3.0)

call g_cls ()
call g_fake_bird_view (-1.0, 1.0, 0.8, 0.6,
1           20.0, 20.0, 150.0, 100.0,
2           array, imax+1, jmax+1, 1)
call g_sleep (3.0)

call g_cls ()
call g_hidden (1.0, 1.0, 0.4, -1.0, 1.0,
1           5.0, 25.0, 20.0, 20.0, 20.0,
2           150.0, 100.0,
3           array, imax+1, jmax+1, 1, G_SIDE_NONE, 2, 1)
call g_sleep (3.0)

```

```
    call g_cls ()
    call g_fake_hidden (-1.0, 1.0, 0.8, 0.6,
1      20.0, 20.0, 150.0, 100.0,
2      array, imax+1, jmax+1, 1, G_SIDE_NONE, 2, 1)
    call g_sleep (3.0)

c  Termination
    call g_term ()

    stop
    end

c  End of file
```


Test Hinomaru no Hata.

a = 0.000000

Test Hinomaru no Hata.

a = 0.200000

Test Hinomaru no Hata.

a = 0.400000

Test Hinomaru no Hata.

a = 0.600000

a = 0.800000

a = 1.000000

a = 1.200000

a = 1.400000

a = 1.600000

a = 1.800000

