GLSC3D の勧め

桂田 祐史

2017 年 12 月 2 日, 2017 年 12 月 10 日

1 はじめに

GLSC3D[1]は、北海道大学の秋山正和[2] 氏を中心とするチームによって開発・保守がされているグラフィックス・ライブラリィである。長い間、使われて来た GLSC (Graphic Library for Scientific Computing) の後継というべきソフトウェアである。

マニュアル [1] は WWW 上に置かれている。

特徴を (独断で) 述べると

- OpenGL 3.3+GLSL に基づく 3D 描画機能
- 最近の環境を基礎にした設計
  (GLSC は、X Window System を利用していて、オリジナルの龍谷大学版の最終更新日時が 1995 年)
- 動画の作成も快適に行える
  (GLSC のやや弱かった面が大幅に改善された)
- GLSC と使い方が似ているので、GLSC に慣れている人にはとっつきやすい
- GLSC と共存が可能
  (似ているが別物で、一つのシステムに両方インストールしても何ら問題がない)

コンピューター・システムの性能向上により、空間 3 次元のシミュレーションが比較的気軽に出来るようになってきている。その可視化の手段として GLSC3D は真っ先にあげられる選択肢になる。

2 Mac へのインストール

2.1 インストール手段の選択理由

GLSC3D では、Mac, Windows, Linux 環境がサポートされているが、Mac はメインの扱いのようで (？)、現象数理学科のメンバーとしてはとても心強い。

二つのインストール手段が提供されている。

1. 「最小限でインストール」
   いわゆるバイナリー・インストールであるが、(良くあるように) OS のバージョンごとにバイナリーが用意されていないので、しばしば警告が出る。

[1] http://www-mmc.es.hokudai.ac.jp/~masakazu/\#section14
[2] http://www-mmc.es.hokudai.ac.jp/~masakazu/

2. 「フルインストール」
   ソースプログラムからコンパイルしてライブラリィを生成する。Xcode と MacPorts を利用する。

ゼミの学生が持っている Mac (現象数理学科 Mac) の状況を考えると (Xcode と MacPorts は入っている、結構古い macOS を使っている人がいる)、フルインストールが適当であると考える。

2.2 準備

(1) (これはオプションで、やらなくても多分大丈夫) macOS のアップグレード — これは必要ではないけれど、前からアップグレードしようかどうか迷っていた人には、この機会にやってみることを勧める。アップグレード自身に長い時間がかかり、アップグレードすると Xcode, MacPorts の再インストールも必要になるので、かなりの時間がかかる。でも、どうせ (2) をするならば、この機会に macOS のアップグレードをするのはトータルでトクになるかもしれない。

(2) Xcode のアップデート

- 最初に Xcode を起動できるかどうか確認すること。macOS をアップグレードした人の中には、Xcode がなくなっている人もいる。Xcode がなくなっている人は、App Store でインストールすること。
- Xcode がある人の場合、左上の林檎マーク→「この Mac について」→ [ソフトウェア・アップデート] で表示されるところに Xcode があれば、アップデートする。
- アップデートするときに、大量のファイルをダウンロードする必要があるので、この作業は、時間に余裕があるときに、高速なネット接続があるところで行うこと。大学でするのは (日曜日でもない限り) 勧められない。

(3) MacPorts のアップデート
ターミナルで

```
sudo port selfupdate
sudo port upgrade outdated
```

を実行する。
(sudo を知らない人に: コマンドを管理者権限で実行するためのコマンドである。最初に sudo を用いるとき、その意味を理解しているかどうか警告される。パスワード入力を求められる。パスワードの後に enter(return) を入力する。パスワードを打っても (盗み見されないよう) 画面には表示されないことに注意する。)
もしかすると、最新の MacPorts をインストールするように促されるかもしれない (がんばって英語を読むこと)。その場合は https://www.macports.org/install.php から自分の macOS に適合するバージョンをダウンロードしてからインストール。
参考: https://trac.macports.org/wiki/Migration, http://nalab.mind.meiji.ac.jp/~mk/knowhow-2017/node32.html

(4) フルインストール用ファイル `Script_on_mac.zip` の入手
http://www-mmc.es.hokudai.ac.jp/~masakazu/Software/Script_on_mac.zip を適当な場所 (ターミナルで作業しやすい場所) に保存する。

本当は「現象数理学科 Mac の環境の更新マニュアル」みたいなのを作るべきかもしれない (素材は既にあちこちに書き散らしてある)。

2.3 フル・インストール手順

```
unzip Script_on_mac.zip
cd Script_on_mac
./1_Install_macports_mac
./2_Install_dependency_library_mac
./3_Test_GLSC3D_on_mac

途中で “Do you want to run the Sample Program?  (YES=1, NO=0)”, “Do you want
to run the Advanced Program?  (YES=1, NO=0)” と尋ねられる。

./4_Install_GLSC3D_on_your_mac
```

コマンドの名前が長目であるが、全部タイプする必要はない。例えば 1_Install_macports_mac を入力するためには、./1 をタイプした後に tab キーを押すと良い。

GLSC_Working_Directory というディレクトリィが残り、「消しても構わない」という意味のメッセージが出るが、サブディレクトリィ GLSC3D にソース・プログラムが残っているので、残しておく方が良い場合もある。

2.4 「最小限」から「フル・インストール」への切り替え手順

「最小限」でインストールしたが、「フル・インストール」に切り替えたい場合の手順を以下に述べる。

「最小限」のファイルのディレクトリィ[3]に UninstallGLSC3D というスクリプトが残っている[4]。これを実行してから、前項の手順を踏む、が大まかな内容であるが、途中で sudo port install libpng が必要のようだ。

```
cd GLSC3D_mac_minimum/
./UninstallGLSC3D
unzip Script_on_mac.zip
cd ../Script_on_mac
./1_Install_macports_mac
./2_Install_dependency_library_mac
sudo port install libpng
./3_Test_GLSC3D_on_mac
./4_Install_GLSC3D_on_your_mac
```

無事インストール出来たかどうか確認するには、例えば

[3] もう消してしまった人は、もう一度ファイルを入手すれば良い。

[4] /usr/local/lib と /usr/local/include にインストールしたファイルと /usr/local/bin/ccg を削除する、という内容。

```
cd ~/GLSC3D_Working_Directory
ls
ccg Hello_GLSC3D.c
./Hello_GLSC3D.c

(HEllo_GLSC3D プログラムを、インストールした ccg でコンパイルして実行できるかどうか)
```

~/GLSC3D_Working_Directory/ の下に GLSC3D というディレクトリィがあり、そこに C プログラムがある (例えばサンプル・プログラムのソースプログラムは Src の下にある。)。

```
cd GLSC3D/Samples
ls
ccg Sample_g_pyramid.c
./Sample_g_pyramid
```

2.5 ドキュメント

マニュアル [1] の PDF ファイルが用意されている。

この手のものは、ソースプログラムもドキュメントの一部かもしれない (読んで勉強になるから)。フルインストール後には、ホームディレクトリィの下に GLSC3D_Working_Directory/GLSC3D というディレクトリィが残り、その下の Src にソース・プログラムが置かれている。

3 C, C++プログラムのコンパイル

大抵の場合、スクリプト ccg を使う、で OK.

```
ccg なんとか.c
```

あるいは

```
ccg なんとか.cpp
```

GLSC3D で提供される ccg には二種類ある。

1. 「フルインストール」でインストールされるもの
   普通、~/bin にコピーされる。インクルードファイル (glsc3d_3.h, glsc3d_3_math.h) は ~/include に、ライブラリィ・ファイル (GLSC3D の libglsc3d.a) は、~/lib に置いてあると想定してある。

   ccg
   ```
   if [ ${1##*.} = cpp ]
   then
   c++ ${1} -W -Wall -O2 -I ~/include -L ~/lib -lglsc3d_3 -framework OpenGL -L/opt/local/lib -lsdl
   else
   cc ${1} -W -Wall -O2 -I ~/include -L ~/lib -lglsc3d_3 -framework OpenGL -L/opt/local/lib -lsdl2
   fi
   ```

2. 「最小限」でインストールされるもの
   普通、/usr/local/bin にコピーされる。インクルードファイルは /usr/local/include に、ライブラリィ・ファイル (GLSC3D の libglsc3d.a 以外に、libpng, libfreetype など) は、/usr/local/lib に置いてあると想定してある。

自分がどちらを実行しているか知りたいときは

```
which ccg
```

とすると良い (/usr/local/bin/ccg ならば、おそらくは「最小限」の方)。

Mac の場合は、インクルード・ファイル、ライブラリィ・ファイルを移動しても問題がない。その場合、スクリプト・ファイルを適当に書き換えれば良い。

4 GLSC ユーザーのための情報

4.1 公式ドキュメントから引用

0. #include <glsc.h> を #include <glsc3d_3.h> に変更。
1. g_init の ウィンドウサイズ指定は int でピクセル単位にする。
2. g_device, g_term は消す。
3. g_def_scale は g_def_scale_2D に変更。std 座標はピクセル単位になっているので変更 (3 倍ぐらい?) する必要あり。
4. 描画終わりに g_finish を挿入。
5. 全ての色指定は (r, g, b, a) に変更。
6. g_move, g_plot, g_box, g_circle の後ろに _2D をつける。
7. g_polyline, g_polygon, g_data_plot, g_contln の後ろに _2D をつける。
8. g_text は g_text_standard に変更。std 座標の変更。
9. g_bird_view, g_hidden は g_bird_view_3D に変更。引数も変更。
10. g_sleep(G_STOP); の G_STOP マクロはないので、-1 などとおく。
11. g_def_scale で座標を決めるのに使った長方形の枠を描くときは、従来は g_box で描いていたが、glsc_3d では g_box_2d ではなく g_boundary を使う。
12. g_sel_scale が呼ばれた時点では g_clipping(0) になっているので、g_boundary で描かれる枠の外にも絵を描きたいときは、g_sel_scale を 呼んだ直後に g_clipping(1) としておく。
13. アニメを作るとき、動くオブジェクトを部分的に塗りつぶして、再描画するという方法を取っていた場合、うまくいかなくなっている。g_cls でクリアして、全体を再描画するように変更する。

文責:小林亮

4.2 プログラム例 1: 1 変数関数のグラフ

これは以前書いた「GLSC の紹介」4 節[5] のプログラムを書き直したもの。

```
 1  /*
 2   * draw-graph-GLSC3D.c -- 1 変数関数のグラフを描く
 3   *   コンパイル:  ccg draw-grap-GLSC3D.c
 4   */
 5
 6  #include <stdio.h>
 7  #include <math.h>
 8
 9  #ifdef OLD
10  #define G_DOUBLE
11  #include <glsc.h>
12  #else
13  #include <glsc3d_3.h>
14  #endif
15
16  double pi;
17
18  int main()
19  {
20    int i, n;
21    double a, b, c, d;
22    double h, x;
23    double f(double);
24  #ifdef OLD
25    char title[100];
26  #endif
27    double win_width, win_height, w_margin, h_margin;
28
29    pi = 4 * atan(1.0);
30
31    /* 表示する範囲 [a,b] × [c,d] を決定 */
32    a = - 10 * pi; b = 10 * pi; c = - 2.0; d = 2.0;
33
34    /* 区間の分割数 n */
35    n = 200;
36
37    /* GLSC の開始
38       メタファイル名、ウィンドウ・サイズの決定 */
39  #ifdef OLD
40    win_width = 200.0; win_height = 200.0; w_margin = 10.0; h_margin = 10.0;
41  #else
42    win_width = 600.0; win_height = 600.0; w_margin = 30.0; h_margin = 30.0;
43  #endif
44    g_init("GRAPH", win_width  + 2 * w_margin, win_height + 2 * h_margin);
45
46  #ifdef OLD
47    /* 出力デバイスの決定 */
48    g_device(G_BOTH);
49  #endif
50
51    /* 座標系の定義: [a,b] × [c,d] という閉領域を表示する */
52  #ifdef OLD
53    g_def_scale(0,
54                a, b, c, d,
55                w_margin, h_margin, win_width, win_height);
56  #else
57    g_def_scale_2D(0,
```

[5] http://nalab.mind.meiji.ac.jp/~mk/labo/howto/intro-glsc/node6.html

```
 58   a, b, c, d,
 59   w_margin, h_margin, win_width, win_height);
 60    g_cls(); // ないとマズイ？
 61  #endif
 62
 63    /* 線を二種類用意する */
 64  #ifdef OLD
 65    g_def_line(0, G_BLACK, 2, G_LINE_SOLID);
 66    g_def_line(1, G_RED,   0, G_LINE_SOLID);
 67  #else
 68    // 線の type は用意されていない
 69  #define G_LINE_SOLID (0)
 70    g_def_line(0, 0, 0, 0, 1, 2, G_LINE_SOLID);
 71    g_def_line(1, 1, 0, 0, 1, 2, G_LINE_SOLID);// 太くしないと点線のように見える
 72  #endif
 73    /* 表示するための文字列の属性を定義する */
 74  #ifdef OLD
 75    g_def_text(0, G_BLACK, 3);
 76  #else
 77    g_def_text(0, 0, 0, 0, 1, 24); // 文字のサイズの単位が全然違う
 78  #endif
 79    /* 定義したものを選択する */
 80    g_sel_scale(0); g_sel_line(0); g_sel_text(0);
 81
 82    /* 座標軸を描く */
 83  #ifdef OLD
 84    g_move(a, 0.0); g_plot(b, 0.0);
 85    g_move(0.0, c); g_plot(0.0, d);
 86  #else
 87    g_move_2D(a, 0.0); g_plot_2D(b, 0.0);
 88    g_move_2D(0.0, c); g_plot_2D(0.0, d);
 89  #endif
 90    /* タイトルを表示する */
 91  #ifdef OLD
 92    sprintf(title, "Bessel function J0(x)  (%g<=x<=%g)", a, b);
 93    g_text(20.0, 10.0, title);
 94  #else
 95    g_text_standard(60.0, 30.0, "Bessel function J0(x)  (%f<=x<=%f)", a, b);
 96  #endif
 97    /* 刻み幅 */
 98    h = (b - a) / n;
 99    /* グラフを描くための線種の選択 */
100    g_sel_line(1);
101    /* 折れ線でグラフを描く */
102  #ifdef OLD
103    g_move(a, f(a));
104  #else
105    g_move_2D(a, f(a));
106  #endif
107    for (i = 1; i <= n; i++) {
108      x = a + i * h;
109  #ifdef OLD
110      g_plot(x, f(x));
111  #else
112      g_plot_2D(x, f(x));
113  #endif
114    }
115  #ifndef OLD
116    g_finish();
117  #endif
118
119    /* ユーザーのマウス入力を待つ */
120    printf("終りました。X の場合はウィンドウをクリックして下さい。\n");
```

```
121    g_sleep(-1.0);
122    /* ウィンドウを閉じる */
123  #ifdef OLD
124    g_term();
125  #endif
126    return 0;
127  }
128
129  double f(double x)
130  {
131    /* 0 次 Bessel 関数 */
132    return j0(x);
133  }
```

- glsc.h のかわりに glsc3d.h をインクルードする。以前のように G_REAL はなくなったので、G_DOUBLE を定義したりする必要はない。

- g_text() で printf() のような書式が使えるようになったので、char 型の配列を準備して、sprintf() を呼び出す必要はなくなった。

- g_init() でウィンドウ・サイズを指定するとき、以前はプリントしたときの長さ (mm) が単位だったのを、ドット数 (ピクセル数, ただし float 型) が単位に変更された。こういう名前が同じで仕様が異なる関数が導入されるのは困ったことだが、まあ受け入れ可能なレベルである。3 倍程度にするというのには、同意 (例えば 200 mm = 20 cm を 600 ピクセルにする)。

- g_{def,sel}_scale() でも単位が長さ (mm) からドット数に変更されたので、g_init() での変更に応じて、数値を大きくする必要がある。

- g_device() は不要であると。確かにファイル出力のしかけは変わったからな。

- 以前は最初に黒板を消す必要、もとい g_cls() を実行する必要はなかったが、それが必須になった？ゴミが表示される (勘違いでなければ)。

- 線のタイプを名前では指定できないようになった。これは少しまずいんじゃないかな。

- 実線を指定したつもりで、線の太さが小さいと、点線のように見えることがある。バグ？

- 線の色を名前でなく、R, G, B で指定するようになった。同じ名前の関数の仕様を変えてしまうのは、率直に言って賛成できない変更。関数の名前を変えるべきであった (C++ ではないので、多重定義というわけにはいかないのか)。

- g_def_text() で文字のサイズを指定する方法も変わったけれど、この辺はオリジナルでは単位もあやふやだったので仕方がないと考える。…昔は文字を描くのは大変で、色々なことが思い出されるけれど、今は freetype もあるし、スマートに解決できた、と言っていいのかも。ぱちぱちぱち。

4.3 プログラム例 2: 2 変数関数のグラフの等高線＆鳥瞰図

まず GLSC バージョンのプログラムを掲げる。

```
/*
 * test-contln.c --- 左側に等高線、右側にグラフの bird view を描く
 *   cglsc test-contln.c
 */
```

```
#include <stdio.h>

/* 動的に確保できる行列 matrix */
#include <stdlib.h>
typedef double **matrix;
matrix new_matrix(int, int);
void delete_matrix(matrix);

#include <math.h>
#ifndef G_DOUBLE
#define G_DOUBLE
#endif
#include "glsc.h"

#define W0 80.0
#define H0 80.0
#define W1 80.0
#define H1 80.0
#define W_MARGIN 10.0
#define H_MARGIN 10.0

double pi;

void compute(double (*)(double, double), matrix,
     double, double, double, double,
     int, int);
double f(double, double);

double max(double x, double y) { return (x > y) ? x : y; }

int main()
{
    int m, n, k;
    double xmin, xmax, ymin, ymax;
    matrix u;

    pi = 4 * atan(1.0);

    /* 分割数 */
    m = 100; n = 100;
    /* 定義域は [-π, π] × [-π, π] */
    xmin = - pi; xmax = pi; ymin = - pi; ymax = pi;

    /* 格子点における数値を納める変数 */
    if ((u = new_matrix(m+1,n+1)) == NULL) {
        fprintf(stderr, " 行列のためのメモリーが確保できませんでした。\n");
        return 1;
    }
    /* GLSC */
    g_init("Meta", W0 + W1 + 3 * W_MARGIN, max(H0, H1) + 2 * H_MARGIN);
    g_device(G_BOTH);
    /* ウィンドウ 0 */
    g_def_scale(0,
                xmin, xmax, ymin, ymax,
                W_MARGIN, H_MARGIN, W0, H0);
    g_def_scale(1,
                xmin, xmax, ymin, ymax,
                W_MARGIN + W0 + W_MARGIN, H_MARGIN, W1, H1);
    /* */
    g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
    g_def_text(0, G_BLACK, 2);
    /* 定義したものを呼び出す */
```

```
    g_sel_scale(0);
    g_sel_line(0);
    g_sel_text(0);
    /* title */
    g_text(W_MARGIN + W0 * 0.6, H_MARGIN / 2, "contour and bird view");
    /* 格子点上での関数値を計算する */
    compute(f, u, xmin, xmax, ymin, ymax, m, n);
    /* 等高線 */
    for (k = -10; k <= 10; k++)
        g_contln(xmin, xmax, ymin, ymax, &u[0][0], m+1, n+1, 0.1 * k);
    /* 鳥瞰図 */
    g_hidden(1.0, 1.0, 0.4,
     -1.0, 1.0,
     /* 視点 (距離, 方角を表わす (θ, φ)) */
     5.0, 30.0, 30.0,
     W_MARGIN + W0 + W_MARGIN, H_MARGIN,
     W1, H1,
     &u[0][0], m + 1, n + 1, 1,
     G_SIDE_NONE, 2, 1);

    printf("終了したらウィンドウをクリックして終了してください。\n");
    g_sleep(-1.0);
    g_term();

    return 0;
}

/*
 * [xmin,xmax] × [ymin,ymax] を x 軸方向に m 等分、y 軸方向に n 等分して
 * 各格子点上の f の値を u に格納する。
 */
void compute(double (*f)(), matrix u,
        double xmin, double xmax, double ymin, double ymax,
        int m, int n)
{
    int i, j;
    double dx, dy, x, y;

    dx = (xmax - xmin) / m;
    dy = (ymax - ymin) / n;
    for (i = 0; i <= m; i++) {
        x = xmin + i * dx;
        for (j = 0; j <= n; j++) {
            y = ymin + j * dy;
            u[i][j] = f(x, y);
        }
    }
}

double f(double x, double y)
{
    return sin(3 * x) * sin(y);
}

matrix new_matrix(int m, int n)
{
  int i;
  double *dp;
  matrix p;
  if ((p = malloc(sizeof(double *) * m)) == NULL) {
    fprintf(stderr, "new_matrix(): cannot allocate memory\n");
    return NULL;
  }
```

```
  if ((dp = malloc(sizeof(double *) * m * n)) == NULL) {
    fprintf(stderr, "new_matrix(): cannot allocate memory\n");
    free(p);
    return NULL;
  }
  for (i = 0; i < m; i++)
    p[i] = dp + i * n;
  return p;
}

void delete_matrix(matrix a)
{
  free(a[0]);
  free(a);
}
```

次に GLSC3D バージョンのプログラム。

```
/*
 * test-contln-GLSC3D.c --- 左側に等高線、右側にグラフの bird view を描く
 *   ccg test-contln-GLSC3D.c
 */

#include <stdio.h>

/* 動的に確保できる行列 matrix */
#include <stdlib.h>
typedef double **matrix;
matrix new_matrix(int, int);
void delete_matrix(matrix);

#include <math.h>
#include "glsc3d_3.h"
double pi;

void compute(double (*)(double, double), matrix,
     double, double, double, double,
     int, int);
double f(double, double);
double max(double x, double y) { return (x > y) ? x : y; }

#define W0 480.0
#define H0 480.0
#define W1 480.0
#define H1 480.0
#define W_MARGIN 50.0
#define H_MARGIN 50.0

int main()
{
    int m, n, k;
    double xmin, xmax, ymin, ymax;
    matrix u;

    pi = 4 * atan(1.0);

    /* 分割数 */
    m = 100; n = 100;
    /* 定義域は [-π, π] × [-π, π] */
    xmin = - pi; xmax = pi; ymin = - pi; ymax = pi;

    /* 格子点における数値を納める変数 */
    if ((u = new_matrix(m+1,n+1)) == NULL) {
        fprintf(stderr, " 行列のためのメモリーが確保できませんでした。\n");
```

```
        return 1;
    }
    g_init("Meta", W0 + W1 + 3 * W_MARGIN, max(H0, H1) + 2 * H_MARGIN);
    g_def_scale_2D(0,
                   xmin - 0.1, xmax + 0.1, ymin - 0.1, ymax + 0.1,
                   W_MARGIN, H_MARGIN, W0, H0);
    g_def_scale_3D(1,
                   xmin, xmax, ymin, ymax, -1.2, 1.2,
                   xmin, xmax, ymin, ymax, -1.2, 1.2,
                   W_MARGIN + W0 + W_MARGIN, H_MARGIN, W1, H1);
    /* */
    g_def_line(0, 0, 0, 0, 1, 1, 0);
    g_def_text(0, 0, 0, 0, 1, 24);
    g_sel_line(0);
    g_sel_text(0);
    /* 定義したものを呼び出す */
    g_sel_scale(0);
    g_move_2D(xmin, ymin); g_plot_2D(xmax, ymin); g_plot_2D(xmax, ymax);
    g_plot_2D(xmin, ymax); g_plot_2D(xmin, ymin);
    /* */
    g_cls();
    /* title */
    g_text_standard(W_MARGIN + W0 * 0.8, H_MARGIN / 2, "contour and bird view");
    /* 格子点上での関数値を計算する */
    compute(f, u, xmin, xmax, ymin, ymax, m, n);
    /* 等高線 */
    for (k = -10; k <= 10; k++) {
        g_contln_f_2D(xmin, xmax, ymin, ymax, m+1, n+1, &u[0][0], 0.1 * k);
//g_finish(); // 毎回呼ぶとおかしくなる。
    }
    //g_finish(); // ここで呼ぶと、次の鳥瞰図を描いた後消えてしまう
    /* 鳥瞰図 */
    g_sel_scale(1);
    g_bird_view_f_3D(xmin, xmax, ymin, ymax,
                     m + 1, n + 1,
            &u[0][0],
            0, 1);
    g_finish();
    printf("終了したらウィンドウをクリックして終了してください。\n");
    g_sleep(-1.0);

    return 0;
}

/*
 * [xmin,xmax] × [ymin,ymax] を x 軸方向に m 等分、y 軸方向に n 等分して
 * 各格子点上の f の値を u に格納する。
 */
void compute(double (*f)(), matrix u,
        double xmin, double xmax, double ymin, double ymax,
        int m, int n)
{
    int i, j;
    double dx, dy, x, y;

    dx = (xmax - xmin) / m;
    dy = (ymax - ymin) / n;
    for (i = 0; i <= m; i++) {
        x = xmin + i * dx;
        for (j = 0; j <= n; j++) {
            y = ymin + j * dy;
            u[i][j] = f(x, y);
        }
```

```
    }
}

double f(double x, double y)
{
    return sin(3 * x) * sin(y);
}

matrix new_matrix(int m, int n)
{
  int i;
  double *dp;
  matrix p;
  if ((p = malloc(sizeof(double *) * m)) == NULL) {
    fprintf(stderr, "new_matrix(): cannot allocate memory\n");
    return NULL;
  }
  if ((dp = malloc(sizeof(double *) * m * n)) == NULL) {
    fprintf(stderr, "new_matrix(): cannot allocate memory\n");
    free(p);
    return NULL;
  }
  for (i = 0; i < m; i++)
    p[i] = dp + i * n;
  return p;
}

void delete_matrix(matrix a)
{
  free(a[0]);
  free(a);
}
```

g_finish() が良く分からない。

A　講習会のサンプル・プログラムを読む

A.1　はじめに

以下のプログラムは 2017/11/？ の講習会で解説されたものである。

例えば 1_CreateWindows.c をコンパイル・実行するには

```
ccg 1_CreateWindow.c
./1_CreateWindow
```

とすれば良い。

GLSC のグラフィックスのウィンドウで esc を入力するとウィンドウが閉じられる。

A.2 1_CreateWindow.c

1_CreateWindow.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3
 4  int main()
 5  {
 6      g_init("Window", 600, 600); //Pixel Size
 7      g_def_scale_2D(0,           //ID
 8                     -1, 1,       //xmin,xmax
 9                     -1, 1,       //ymin,ymax
10                     20.0, 20.0,  //Window (Left, Top) Position
11                     560, 560);   //Window Size (x,y)
12      g_cls();                    //Clear window
13      g_sel_scale(0);             //Select Virtual scale
14      g_boundary();               //Draw Boundary
15      g_finish();                 //flush Draw buffer
16      g_sleep(10.0);              //Sleep 10 sec
17
18      return 0;
19  }
```

- 2行目 glsc.h でなく glsc_3d.h をインクルード。
- 6行目 ウィンドウのサイズの単位はピクセルであることに注意。
- 7行目 g_def_scale() でなくて g_def_scale_2D() という名前であるが、引数の意味は g_def_scale() と同様。
- 14行目
- 15行目 g_finish() これを適当なタイミングで実行するのが、GLSC とは違うところ。

A.3 2_VectorField.c

VectorField.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3
 4  int main()
 5  {
 6      g_init("Window", 600, 600); //Pixel Size
 7      g_def_scale_2D(0,           //ID
 8                     -1, 1,       //xmin,xmax
 9                     -1, 1,       //ymin,ymax
10                     20.0, 20.0,  //Window (Left, Top) Position
11                     560, 560);   //Window Size (x,y)
12      g_cls();                    //Clear window
13      g_sel_scale(0);             //Select Virtual scale
14      g_boundary();               //Draw Boundary
15      int Imax = 10;
16      int Jmax = 10;
17      double VecX; double VecY;
18      for(int i = 1; i < Imax; i ++)
19      {
20          for(int j = 1; j < Jmax; j ++)
21          {
22              double x = i*0.2 - 1.0, y = j*0.2 - 1.0;
23              VecX = x; VecY = y;     // Divergence
24              //VecX = -y; VecY = x;  // Rotation
25
26              g_arrow_2D(x, y,        // Base Point
27                         VecX, VecY,  // Direction
28                         0.1,         // Size of Arrow Length
29                         0.05,        // Size of Arrow Head
30                         2);          // Arrow kinds
31          }
32      }
33      g_finish();                 //flush Draw buffer
34      g_sleep(10.0);              //Sleep 10 sec
35      return 0;
36  }
```

- 26 行目 g_arrow() というのは、オリジナルの GLSC にはなく、後から導入されたコマンドで、g_arrow_2D() は GLSC3D における 2 次元バージョン。注釈のおかげで意味はほぼ分かる。最後の引数は Arrow kinds だそうだけど、マジック・ナンバーなのはちょっと嫌。

A.4 3_Animation.c

Animation.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3
 4  int main()
 5  {
 6      int INTV = 100;
 7      g_init("Window", 600, 600);                 //Pixel Size
 8      g_def_scale_2D(0,                           //ID
 9                     -1, 1,                       //xmin,xmax
10                     -1, 1,                       //ymin,ymax
11                     20.0, 20.0,                  //Window (Left, Top) Position
12                     560, 560);                   //Window Size (x,y)
13
14      ////////////// Start time loop //////////////
15      for (int i_time = 0; ; i_time++) {
16          double dt = 0.0001;                     //Time Discritization
17          double t = i_time * dt;                 //Time
18          ////////////// Calculation Part //////////////
19          double x = 0.5 * cos(2.0*M_PI * t);     //X Coordinate
20          double y = 0.5 * sin(2.0*M_PI * t);     //Y Coordinate
21          ////////////// Draw Part //////////////
22          if (i_time%INTV == 0) {
23              g_cls();                            //Clear window
24              g_sel_scale(0);                     //Select Virtual scale
25              g_boundary();                       //Draw Boundary
26              g_area_color(1,0,0,1);              //Area Color
27              g_circle_2D(x, y, 0.1, G_NO, G_YES);//g_circle_2D
28              g_finish();                         //flush Draw buffer
29              g_sleep(0.01);                      //Sleep 0.01 sec
30          }
31      }
32      return 0;
33  }
```

- 19 行目円周率を意味する M_PI は、C 言語の規格には入っていないけれど (だから GCC, LLVM にはあるけれど、Visual Studio の C コンパイラーでは定義されていない？)、定義されていないときは GLSC3D で定義するのだそうです。

- 26 行目 g_area_color() という名前の関数は、GLSC にもあるけれど、引数が変わっている。R, G, B, と不透明度を並べるのだとか。

- 27 行目 g_circle() ...g_circle_2D()

A.5 4_Wave_2D_Contln.c

Wave_2D_Contln.c

```
/*****************************************************************************************

    This program solves a wave equation using the explicit method.

    A wave equation is utt = c * c * uxx.
    The function f is the initial shape of wave.

    Time loop in this program consists of
    "Calculation part" and "Draw Part".

    Calculation part is controlled by the interval parameter INTV.
    This numerical result is visualized by using "g_contln_2D" of GLSC3D.

*****************************************************************************************/

#include<stdio.h>
#include<glsc3d_3.h>

#define N (100)
#define M (100)

double f(double x,double y)
{
    return 3 * exp(-((x - M_PI / 2)*(x - M_PI / 2) + (y  - 3 * M_PI / 2)*(y  - 3 * M_PI / 2))/ 0.
}

int main()
{
    double  x, y;

    double  Lx = 8 * M_PI;
    double  Ly = 8 * M_PI;
    double  dx = Lx / N;
    double  dy = Ly / M;
    double  dt = 0.002;
    double  c = 1.0;

    double  u0[N + 2][M + 2];
    double  u1[N + 2][M + 2];
    double  u2[N + 2][M + 2];

    double  lambda_x = c * dt / dx;
    double  lambda_y = c * dt / dy;

    int     INTV = 100;

    g_init("Window", 600, 600); //Pixel Size

    g_def_scale_2D(0,                   //ID
                   -Lx*0.5, Lx*0.5,     //xmin,xmax
                   -Ly*0.5, Ly*0.5,     //ymin,ymax
                   20.0, 20.0,          //Window (Left, Top) Position
                   560, 560);           //Window Size (x,y)

    //Set initial calculation
    {
        //Step1
        for(int j = 1;j < M + 1; j ++)
        {
            y = (j - 0.5) * dy;
            for(int i = 1;i < N + 1; i ++)
            {
                x = (i - 0.5) * dx;
                u0[i][j] = f(x,y);
```

- 長方形領域上の 2 次元波動方程式 $\frac{1}{c^2}u_{tt} = u_{xx} + u_{yy}$ (境界条件は同次 Neumann なのかな) を差分法で解いた解を可視化する (等高線を描く) プログラム。

- `g_line_color()` も `g_area_color()` とおなじく、R, G, B, 不透明度を引数に取る。

- 長方形 $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ で定義された関数 u があるとき、定義域を x 軸方向に N_x 等分, y 軸方向に N_y 等分して格子を作り、格子点での値 $u(x_i, y_j)$ $(0 \leq i \leq N_x, 0 \leq j \leq N_y)$ を 2 次元配列 `u[nx+1,ny+1]` に記録してあるとする。このとき関数値が `level` である等高線を描くには、

```
g_contln_2D(xmin, xmax, ymin, ymax, nx+1, ny+1, u, level);
```

とすれば良い。

A.6 5_Turing_2D_Contln.c

5_Turing_2D_Contln.c

```
/*****************************************************************************

 This program solves reacrtion-diffusion equations using the explicit method.

 Reacrtion-diffusion equations are ut = du uxx + f(u,v) and vt = dv vxx + g(u,v).

 Time loop in this program consists of
 "Calculation part" and "Draw Part".

 Calculation part is controlled by the interval parameter INTV.
 This numerical result is visualized by using "g_contln_2D" of GLSC3D.

*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<glsc3d_3.h>

#define N (100)
#define M (100)

double f(double u, double v)
{
    return u * (1 - u * u) - v;
}
double g(double u, double v)
{
    return 3.0 * u - 2.0 * v;
//    return 3.0 * (u + 0.05) - 2.0 * v;
//    return 3.0 * (u - 0.05) - 2.0 * v;
}

int main()
{
    double  Lx = 100.0;
    double  Ly = 100.0;
    double  dx = Lx / N;
    double  dy = Ly / M;
    double  dt = 0.001;
    double  du = 1.0;
    double  dv = 10.0;

    double  lambda_u = du * dt / (dx * dx);
    double  lambda_v = dv * dt / (dy * dy);

    double  u0[N + 2][M + 2];
    double  u1[N + 2][M + 2];
    double  v0[N + 2][M + 2];
    double  v1[N + 2][M + 2];

    int INTV = 200;

    g_init("Window", 600, 600); //Pixel Size

    g_def_scale_2D(0,                   //ID
                   -Lx*0.5, Lx*0.5,//xmin,xmax
                   -Ly*0.5, Ly*0.5,     //ymin,ymax
                   20.0, 20.0,          //Window (Left, Top) Position
                   560, 560);           //Window Size (x,y)
    //Set initial calculation
    {
        //Step1
        double u_Init_Sum = 0.0;
        double v_Init_Sum = 0.0;
```

- 反応拡散方程式 (reaction diffusion equations) を差分法で解いて、等高線を描くプログラム。

A.7 6_Turing_2D_ColorMap.c

6_Turing_2D_ColorMap.c

```
/*****************************************************************************

 This program solves reacrtion-diffusion equations using the explicit method.

 Reacrtion-diffusion equations are ut = du uxx + f(u,v) and vt = dv vxx + g(u,v).

 Time loop in this program consists of
 "Calculation part" and "Draw Part".

 Calculation part is controlled by the interval parameter INTV.
 This numerical result is visualized by using "g_color_map" made in thie program.

*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<glsc3d_3.h>

#define N (100)
#define M (100)

////////////// Color map: g_color_map ////////////
void g_color_map(double Array[][M+2],
                 double dx_width, double dy_width,
                 double L_bottom_x, double L_bottom_y,
                 double max, double min
)
{
    int Fine_Grid = 1;
    for (int j=0; j < M+2; j++) {
        for (int i=0; i < N+2; i++) {
            if (Fine_Grid == 0) {
                g_area_color((Array[i][j]-min)/(max-min), 0, 1.0-(Array[i][j]-min)/(max-min), 0.5
                g_box_center_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y, dx_widt
            }

            if (Fine_Grid == 1) {
                double tmpColor;
                if (j-1<0) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i][j-1]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x - dx_width*0.5, dy_width * (j) + L_bott
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bott
                              G_NO, G_YES);

                if (i+1>=N+2) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i+1][j]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bott
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bott
                              G_NO, G_YES);

                if (j+1>=M+2) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i][j+1]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bott
                              dx_width * (i) + L_bottom_x - dx_width*0.5, dy_width * (j) + L_bott
                              G_NO, G_YES);

                if (i-1<0) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i-1][j]*0.5;
```

- 反応拡散方程式 (reaction diffusion equations) を差分法で解いて、レベルを色で表したプログラム。

A.8 7_Wave_2D_ColorMap.c

7_Wave_2D_ColorMap.c

```
/*****************************************************************************

    This program solves a wave equation using the explicit method.

    A wave equation is utt = c * c * uxx.
    The function f is the initial shape of wave.

    Time loop in this program consists of
    "Calculation part" and "Draw Part".

    Calculation part is controlled by the interval parameter INTV.
    This numerical result is visualized by using "g_color_map".

*****************************************************************************/

#include<stdio.h>
#include<glsc3d_3.h>

#define N (100)
#define M (100)

////////////// Color map: g_color_map //////////////
void g_color_map(double Array[][M+2],
                 double dx_width, double dy_width,
                 double L_bottom_x, double L_bottom_y,
                 double max, double min
                 )
{
    int Fine_Grid = 1;
    for (int j=0; j < M+2; j++) {
        for (int i=0; i < N+2; i++) {
            if (Fine_Grid == 0) {
                g_area_color((Array[i][j]-min)/(max-min), 0, 1.0-(Array[i][j]-min)/(max-min), 0.5);
                g_box_center_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y, dx_width, d
            }

            if (Fine_Grid == 1) {
                double tmpColor;
                if (j-1<0) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i][j-1]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x - dx_width*0.5, dy_width * (j) + L_bottom_y
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bottom_y
                              G_NO, G_YES);

                if (i+1>=N+2) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i+1][j]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bottom_y
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bottom_y
                              G_NO, G_YES);

                if (j+1>=M+2) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i][j+1]*0.5;
                g_area_color((tmpColor-min)/(max-min), 0, 1.0-(tmpColor-min)/(max-min), 1);
                g_triangle_2D(dx_width * (i) + L_bottom_x, dy_width * (j) + L_bottom_y,
                              dx_width * (i) + L_bottom_x + dx_width*0.5, dy_width * (j) + L_bottom_y
                              dx_width * (i) + L_bottom_x - dx_width*0.5, dy_width * (j) + L_bottom_y
                              G_NO, G_YES);

                if (i-1<0) tmpColor = Array[i][j];
                else tmpColor = Array[i][j]*0.5 + Array[i-1][j]*0.5;
```

- 2 次元波動方程式 $\frac{1}{c^2}u_{tt} = u_{xx} + u_{yy}$ を差分法で解いた解を値を色で表して可視化するプログラム。

A.9 8_Wave_2D_BirdView.c

8_Wave_2D_BirdView.c

```
/*****************************************************************************************

    This program solves a wave equation using the explicit method.

    A wave equation is utt = c * c * uxx.
    The function f is the initial shape of wave.

    Time loop in this program consists of
    "Calculation part" and "Draw Part".

    Calculation part is controlled by the interval parameter INTV.
    This numerical result is visualized by using "g_bird_view_3D" of GLSC3D.
    Please note that this program set "g_def_scale_3D".

*****************************************************************************************/

#include<stdio.h>
#include<glsc3d_3.h>

#define N (100)
#define M (100)

double f(double x,double y)
{
    return 3 * exp(-((x - M_PI / 2)*(x - M_PI / 2) + (y  - 3 * M_PI / 2)*(y  - 3 * M_PI / 2))/ 0.
}

int main()
{
    double  x, y;

    double  Lx = 8 * M_PI;
    double  Ly = 8 * M_PI;
    double  Lz = 8 * M_PI;
    double  dx = Lx / N;
    double  dy = Ly / M;
    double  dt = 0.002;
    double  c = 1.0;

    double  u0[N + 2][M + 2];
    double  u1[N + 2][M + 2];
    double  u2[N + 2][M + 2];

    double  lambda_x = c * dt / dx;
    double  lambda_y = c * dt / dy;

    int     INTV = 100;

    g_init("Window", 600, 600); //Pixel Size

    g_def_scale_3D_fix(0,                   //ID
                   -Lx*0.5, Lx*0.5,//xmin,xmax
                   -Ly*0.5, Ly*0.5,     //ymin,ymax
                   -Lz*0.5, Lz*0.5,     //zmin,zmax
                   20.0, 20.0,          //Window (Left, Top) Position
                   560, 560);           //Window Size (x,y)

//    g_vision(0,                         //ID
//             0.0, -Ly, Lz,              //eye
//             0.0, 0.0, 1.0,             //up
//             1.25                       //zoom
//             );
    //Set initial calculation
    {
```

- 2 次元波動方程式 $\frac{1}{c^2}u_{tt} = u_{xx} + u_{yy}$ を差分法で解いた解をのグラフの鳥瞰図 (bird view) を描くプログラム。

A.10 9_Diffusion_1D.c

9_Diffusion_1D.c

```
#include<stdio.h>
#include<glsc3d_3.h>

#define N (200)

int main()
{
    double  x;
    double  L = 5;
    double  dx = L / N;
    double  t = 0.0;
    double  dt = 0.0001;
    double  u[N + 2];
    double  tmpu[N + 2];
    double  D = 1.0;
    int     INTV = 100;
    g_init("Window", 600, 320); //Pixel Size
    g_def_scale_2D(0,                   //ID
                   -L*0.5, L*0.5,       //xmin,xmax
                   -0.2, 1.2,     //ymin,ymax
                   20.0, 20.0,          //Window (Left, Top) Position
                   560, 280);           //Window Size (x,y)
    //Set initial calculation
    {
        //Step1
        for(int i = 1;i < N + 1; i ++)
        {
            x = i*dx - L*0.5;
            u[i] = exp(-(x*x));
        }
        //Step2
        u[0] = u[1];
        u[N + 1] = u[N];
    }
    ////////////// Start time loop ////////////
    for (int i_time = 0;t < 10 ; i_time++) {

        t = i_time * dt;

        ////////////// Draw Part ////////////
        if (i_time%INTV == 0) {
            g_cls();                    //Clear window
            g_sel_scale(0);             //Select Virtual scale
            g_line_color(0.0, 0.0, 0.0, 1.0);
            g_boundary();               //Draw Boundary
            g_line_width(1.0);
            g_line_color(0.0, 0.0, 0.0, 0.5);
            g_move_2D(-L*0.5, 0.0); g_plot_2D(L*0.5, 0.0);  //X Axis
            g_move_2D(0.0, -1.2); g_plot_2D(0.0, 1.2);      //Y Axis
            //Profile of u
            g_line_width(2.0);
            g_line_color(1.0, 0.0, 0.0, 1.0);
            for (int i=0; i<N-1; i++) {
                g_move_2D(i*dx-L*0.5, u[i]);
                g_plot_2D((i+1)*dx-L*0.5, u[i+1]);
            }

            g_text_size(18);
            g_text_color(1.0, 0.0, 0.0, 1.0);
            g_text_2D_virtual(L*0.5 - 0.4, 1.0, "u: -");

            g_finish();                 //flush Draw buffer
            g_sleep(0.0);               //Sleep 0.0 sec
        }
        ////////////// Calculation Part ////////////
```

- 空間１次元の拡散方程式のグラフを描くプログラム。

A.11 10_Diffusion_1D_Text.c

10_Diffusion_1D_Text.c

```
#include<stdio.h>
#include<glsc3d_3.h>

#define N (200)

int main()
{
    double  x;
    double  L = 5;
    double  dx = L / N;
    double  t = 0.0;
    double  dt = 0.0001;
    double  u[N + 2];
    double  tmpu[N + 2];
    double  D = 1.0;
    int     INTV = 100;
    g_init("Window", 900, 320); //Pixel Size
    g_def_scale_2D(0,                   //ID
                   -L*0.5, L*0.5,       //xmin,xmax
                   -0.2, 1.2,     //ymin,ymax
                   20.0, 20.0,          //Window (Left, Top) Position
                   560, 280);           //Window Size (x,y)

    g_def_scale_2D(1,                   //ID
                   -L*0.5, L*0.5,       //xmin,xmax
                   -0.2, 1.2,     //ymin,ymax
                   620.0, 20.0,          //Window (Left, Top) Position
                   260, 280);           //Window Size (x,y)

    //Set initial calculation
    {
        //Step1
        for(int i = 1;i < N + 1; i ++)
        {
            x = i*dx - L*0.5;
            u[i] = exp(-(x*x));
        }
        //Step2
        u[0] = u[1];
        u[N + 1] = u[N];
    }
    ////////////// Start time loop ////////////
    for (int i_time = 0; t < 10; i_time++) {
        t = i_time * dt;
        ////////////// Draw Part ////////////
        if (i_time%INTV == 0) {
            g_cls();                    //Clear window
            ////////////// Graph Part ////////////
            {
                g_sel_scale(0);             //Select Virtual scale
                g_line_color(0.0, 0.0, 0.0, 1.0);
                g_boundary();               //Draw Boundary

                g_line_width(1.0);
                g_line_color(0.0, 0.0, 0.0, 0.5);
                g_move_2D(-L*0.5, 0.0); g_plot_2D(L*0.5, 0.0);  //X Axis
                g_move_2D(0.0, -1.2); g_plot_2D(0.0, 1.2);      //Y Axis

                //Profile of u
                g_line_width(2.0);
                g_line_color(1.0, 0.0, 0.0, 1.0);
                for (int i=0; i<N-1; i++) {
                    g_move_2D(i*dx-L*0.5, u[i]);
                    g_plot_2D((i+1)*dx-L*0.5, u[i+1]);
```

- 空間1次元の拡散方程式のグラフを描くプログラムという点では、9_Diffusion_1D.c と同じ。凡例というのか、パラメーターや現在時刻を表示する。そのために、二つの scale を定義している。

A.12 11_g_def_scale_2D.c

11_g_def_scale_2D.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WINDOW_SIZE_X     (600)
 4  #define WINDOW_SIZE_Y     (300)
 5  int main()
 6  {
 7      g_init("Window", WINDOW_SIZE_X, WINDOW_SIZE_Y);
 8      g_def_scale_2D(0, -1, 1, -1, 1,
 9                     20.0, 20.0,
10                     260.0, 260.0);
11      g_def_scale_2D(1, -1, 1, -1, 1,
12                     //   g_def_scale_2D(1, -2, 2, -2, 2,
13                     320.0, 20.0,
14                     260.0, 260.0);
15      for (int i_time = 0;; i_time++)
16      {
17          g_cls();
18          g_sel_scale(0);
19          g_boundary();
20          g_circle_2D(0.0, 0.0, 1.0, 1, 1);
21
22          g_sel_scale(1);
23          g_boundary();
24          //      g_area_color(0, 1, 0, 1);
25          //      g_circle_2D(0.0, 0.0, 1.0, 1, 1);
26          g_finish();
27      }
28      return 0;
29  }
```

-

A.13 12_g_def_scale_3D_fix.c

12_g_def_scale_3D_fix.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WX     (600)
 4  #define WY     (600)
 5  int main()
 6  {
 7      g_init("Window", WX, WY);
 8      g_def_scale_3D_fix(0,
 9                         -1, 1,
10                         -1, 1,
11                         -1, 1,
12                         20.0, 20.0,
13                         WX - 40.0, WY - 40.0);
14
15      for (int i_time = 0;; i_time++)
16      {
17          g_cls();
18          g_sel_scale(0);
19          g_boundary();
20          g_sphere_3D(0.0, 0.0, 0.0,
21                      1.0, 0, 1);
22
23          g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
24          g_finish();
25      }
26      return 0;
27  }
```

-

A.14 13_g_vision.c

13_g_vision.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WX     (600)
 4  #define WY     (600)
 5  int main()
 6  {
 7      g_init("Window", WX, WY);
 8      g_def_scale_3D_fix(0,
 9                         -1, 1,
10                         -1, 1,
11                         -1, 1,
12                         20.0, 20.0,
13                         WX - 40.0, WY - 40.0);
14
15      for (int i_time = 0;; i_time++)
16      {
17          double a = -1;
18          g_vision(0,
19                   0, a, 4,
20                   0, 0, 1,
21                   1.0);
22          g_cls();
23          g_sel_scale(0);
24          g_boundary();
25          g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
26          //X-Axes
27          g_area_color(1, 0, 0, 1);
28          g_arrow_3D(-1, -1, -1,
29                     1.0, 0.0, 0.0,
30                     2.0, 0.25,
31                     0, 1);
32          //Y-Axes
33          g_area_color(0, 1, 0, 1);
34          g_arrow_3D(-1, -1, -1,
35                     0.0, 1.0, 0.0,
36                     2.0, 0.25,
37                     0, 1);
38          //Z-Axes
39          g_area_color(0, 0, 1, 1);
40          g_arrow_3D(-1, -1, -1,
41                     0.0, 0.0, 1.0,
42                     2.0, 0.25,
43                     0, 1);
44
45          g_area_color(1, 1, 0, 1);
46          g_cone_3D(0, 0, -1,
47                    0, 0, 1,
```

•

A.15 14_g_def_scale_3D_1.c

14_g_def_scale_3D_1.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WX     (600)
 4  #define WY     (600)
 5
 6  #define XLEN    (2.0 * M_PI)
 7  #define YLEN    (2.0 * M_PI)
 8  #define ZLEN    (2.0 * M_PI)
 9  #define Imax    (100)
10  #define Jmax    (100)
11
12  int main()
13  {
14      g_init("Window", WX, WY);
15      g_def_scale_3D_fix(0,
16                         -XLEN / 2, XLEN / 2,
17                         -YLEN / 2, YLEN / 2,
18                         -ZLEN / 2, ZLEN / 2,
19                         20.0, 20.0,
20                         WX - 40.0, WY - 40.0);
21      double u[Imax][Jmax];
22      double dx = XLEN / Imax, dy = YLEN / Jmax,rx,ry;
23
24      for(int i = 0;i < Imax;i ++)
25      {
26          rx = (i + 0.5) * dx - XLEN / 2;
27          for(int j = 0;j < Jmax;j ++)
28          {
29              ry = (j + 0.5) * dy - YLEN / 2;
30              u[i][j] = sin(rx * ry) * 0.5;
31          }
32      }
33      for (int i_time = 0;; i_time++)
34      {
35          g_cls();
36          g_sel_scale(0);
37          g_boundary();
38          g_box_center_3D_core(0, 0, 0, XLEN, YLEN, ZLEN*0.5, 0, 1, 0);
39          g_bird_view_3D(-XLEN / 2, XLEN / 2,
40                         -YLEN / 2, YLEN / 2,
41                         Imax, Jmax,
42                         u, 0, 1);
43          g_finish();
44      }
45      return 0;
46  }
```

•

A.16 15_g_def_scale_3D_2.c

15_g_def_scale_3D_2.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WX     (600)
 4  #define WY     (600)
 5
 6  #define XLEN    (2.0 * M_PI)
 7  #define YLEN    (2.0 * M_PI)
 8  #define ZLEN    (2.0 * M_PI)
 9  #define Imax    (100)
10  #define Jmax    (100)
11
12  int main()
13  {
14      g_init("Window", WX, WY);
15      //  g_def_scale_3D_fix(0,
16      //                     -XLEN / 2, XLEN / 2,
17      //                     -YLEN / 2, YLEN / 2,
18      //                     -ZLEN / 2, ZLEN / 2,
19      //                     20.0, 20.0,
20      //                     WX - 40.0, WY - 40.0);
21      g_def_scale_3D(0,
22                     -XLEN / 2, XLEN / 2,
23                     -YLEN / 2, YLEN / 2,
24                     -ZLEN / 4, ZLEN / 4,
25                     -XLEN / 2, XLEN / 2,
26                     -YLEN / 2, YLEN / 2,
27                     -ZLEN / 2, ZLEN / 2,
28                     20.0, 20.0,
29                     WX - 40.0, WY - 40.0);
30
31      double u[Imax][Jmax];
32      double dx = XLEN / Imax, dy = YLEN / Jmax,rx,ry;
33
34      for(int i = 0;i < Imax;i ++)
35      {
36          rx = (i + 0.5) * dx - XLEN / 2;
37          for(int j = 0;j < Jmax;j ++)
38          {
39              ry = (j + 0.5) * dy - YLEN / 2;
40              u[i][j] = sin(rx * ry) * 0.5;
41          }
42      }
43      for (int i_time = 0;; i_time++)
44      {
45          g_cls();
46          g_sel_scale(0);
47          g_boundary();
```

•

A.17 16_NewFunction1.c

16_NewFunction1.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WX     (600)
 4  #define WY     (600)
 5  int main()
 6  {
 7      g_init("Window", WX, WY);
 8      g_def_scale_3D_fix(0,
 9                         -1, 1,
10                         -1, 1,
11                         -1, 1,
12                         20.0, 20.0,
13                         WX - 40.0, WY - 40.0);
14      for (int i_time = 0;; i_time++)
15      {
16          double dt = 0.01;
17          double t = i_time * dt;
18          g_vision(0,
19                   3 * cos(t), 3 * sin(t), 3,
20                   0, 0, 1,
21                   1.0);
22          g_cls();
23          g_sel_scale(0);
24          g_boundary();
25          g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
26          //X-Axes
27          g_area_color(1, 0, 0, 1);
28          g_arrow_3D(-1, -1, -1,
29                     1.0, 0.0, 0.0,
30                     2.0, 0.25,
31                     0, 1);
32          //Y-Axes
33          g_area_color(0, 1, 0, 1);
34          g_arrow_3D(-1, -1, -1,
35                     0.0, 1.0, 0.0,
36                     2.0, 0.25,
37                     0, 1);
38          //Z-Axes
39          g_area_color(0, 0, 1, 1);
40          g_arrow_3D(-1, -1, -1,
41                     0.0, 0.0, 1.0,
42                     2.0, 0.25,
43                     0, 1);
44
45          g_area_color(1, 1, 0, 1);
46          g_cone_3D(0, 0, -1,
47                    0, 0, 1,
```

•

A.18 17_NewFunction2.c

17_NewFunction2.c

```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX     (600)
#define WY     (600)
int main()
{
    g_enable_highdpi();
    g_set_antialiasing(4);
    g_init("Window", WX, WY);
    g_def_scale_3D_fix(0,
                       -1, 1,
                       -1, 1,
                       -1, 1,
                       20.0, 20.0,
                       WX - 40.0, WY - 40.0);


    for (int i_time = 0;; i_time++)
    {
        double dt = 0.01;
        double t = i_time * dt;
        g_vision(0,
                 3 * cos(t), 3 * sin(t), 3,
                 0, 0, 1,
                 1.0);
        g_cls();
        g_sel_scale(0);
        g_boundary();
        g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
        //X-Axes
        g_area_color(1, 0, 0, 1);
        g_arrow_3D(-1, -1, -1,
                   1.0, 0.0, 0.0,
                   2.0, 0.25,
                   0, 1);
        //Y-Axes
        g_area_color(0, 1, 0, 1);
        g_arrow_3D(-1, -1, -1,
                   0.0, 1.0, 0.0,
                   2.0, 0.25,
                   0, 1);
        //Z-Axes
        g_area_color(0, 0, 1, 1);
        g_arrow_3D(-1, -1, -1,
                   0.0, 0.0, 1.0,
                   2.0, 0.25,
                   0, 1);
```

-

A.19 18_Hinode_Tuki_1.c

18_Hinode_Tuki_1.c

```
 1  #include<stdio.h>
 2  #include<glsc3d_3.h>
 3  #define WINDOW_SIZE_X     (800)
 4  #define WINDOW_SIZE_Y     (400)
 5  int main()
 6  {
 7      g_init("Hinode_Tuki", WINDOW_SIZE_X, WINDOW_SIZE_Y);
 8      g_def_scale_2D(0, -4, 4, -2, 2,
 9                     0.0, 0.0,
10                     WINDOW_SIZE_X, WINDOW_SIZE_Y);
11
12      for (int i_time = 0;; i_time++)
13      {
14          g_cls();
15          g_sel_scale(0);
16          g_boundary();
17          g_finish();
18      }
19      return 0;
20  }
```

-

A.20 19_Hinode_Tuki_2.c

19_Hinode_Tuki_2.c

```
#include<stdio.h>
#include<glsc3d_3.h>
#define WINDOW_SIZE_X     (800)
#define WINDOW_SIZE_Y     (400)
int main()
{
    g_init("Hinode_Tuki", WINDOW_SIZE_X, WINDOW_SIZE_Y);
    g_def_scale_2D(0, -4, 4, -2, 2,
                   0.0, 0.0,
                   WINDOW_SIZE_X, WINDOW_SIZE_Y);

    g_capture_set("");
    double dt = 0.005, t,tmax = 5;

    for (int i_time = 0;; i_time++)
    {
        t = i_time * dt;
        g_cls();
        g_sel_scale(0);
        g_boundary();

        //Sora
        g_area_color(0.6, 0.6, 0.6, 1);
        g_box_2D(-4,4,-0.5,2,G_NO,G_YES);

        //Tuki
        g_area_color(1, 1, 0, 1);
        g_circle_2D(2.0, 1.0, 0.2, G_NO, G_YES);
        g_area_color(0.6, 0.6, 0.6, 1);
        g_circle_2D(1.8, 1.0, 0.2, G_NO, G_YES);

        //Taiyo
        g_area_color(1, 0, 0, 1);
        g_circle_2D(0.0, tanh(t) - 1, 0.5, G_NO, G_YES);

        //Umi
        g_area_color(0.2, 0.6, 1, 1);
        g_box_2D(-4,4,-2,-0.5,G_NO,G_YES);

        //Moji
        if(t > tmax)
        {
            g_text_standard(350, 100, "日本一!!");
        }

        g_finish();
        g_capture();
```

- Frames.2017.月.日.時.分.秒/ というディレクトリィを作り、000000.png, 000001.png, ··· という 6 桁の数 + .png という名前で画像ファイルを出力する。

```
ffmpeg -i Frames.2017.月.日.時.分.秒/%06d.png なんとか.mpg
```

とすると (%06d というのは printf() で用いる書式の流用である)、それら PNG ファイルから なんとか.mpg という名前の動画ファイルを作る。

B なぞ

- g_cls() を実行しないとゴミが表示される？
- g_finish() を連続して実行するとおかしい。バッファーが空のときの動作が変、ということらしい。
- g_contln_f_2D() の動作が不安定。一度描いた等高線が消されてしまう？なぜ？
- マニュアルの関数の説明で引数の意味が書いていないことが多い。実験精神を発露させるか、ソースを読むか。

参考文献

[1] 秋山正和, 舘入数磨, 舘入数磨, 小林亮：GLSC3D (Ver. 3.0.0) Manual, http://www-mmc.es.hokudai.ac.jp/~masakazu/Software/Manual/GLSC3D_Ver3.0.0Manual/GLSC3D_Manual.pdf (2017/11/22).