2008 年度 卒業研究レポート

# OpenGL による 2 次元波動方程式の可視化

明治大学 理工学部 数学科
石橋　侑佳

2009 年 2 月 27 日

# 目 次

# はじめに

私は、OpenGL というグラフィックスライブラリィを利用して、長方形領域や円盤領域の波動方程式を可視化することを目標とし、研究してきた。数ある可視化の方法の中から OpenGL を選んだ理由は、CG に興味があったためである。OpenGL は、GLSC[1]のように数値計算の結果を可視化するために作られたものではなく、CG を描くためのライブラリィであるため、グラフを描くことは容易でなかったが、逆に数値解析用のものよりも精度の高い描画ができたと考えている。

さて、このレポートについて簡単に紹介すると、まず第 1 章「OpenGL について」では、OpenGL についての説明や、CG 特有の「座標系」の概念に触れている。第 2 章「OpenGL でグラフを描画するためのプログラム」では、私が作ったグラフを描画するためのライブラリィについて説明している。第 3 章「長方形領域上の波動方程式の初期値境界値問題」、第 4 章「円盤領域上の波動方程式の初期値境界値問題」では、長方形領域上と円盤領域上の波動方程式の差分法での解き方や、実際にそれを計算するプログラムをそれぞれ載せた。さらに付録として、描画した画像を JPEG ファイルに保存するためのプログラムについて簡単に触れた。

[1]Graphic Library for Scientific Computing の略。龍谷大学数理情報学科のグループが作成した、科学技術計算の結果を可視化するためのライブラリィである。

# 第1章　OpenGLについて

## 1.1　OpenGLとは

OpenGL（Open Graphics Library）とは、Silicon Graphics社が開発した3次元のグラフィックスライブラリィである。主な特長として、

- UNIX、Windows、MacOSなどの代表的なプラットフォームすべてに対応している
- ソースコードが公開されているため、広く一般に普及している
- 非常に高速に動作し、高精度な3次元CGを描画できる
- 標準C言語の知識のみでプログラミングが可能である

が挙げられる。また、陰面処理を容易に行うことができるなどの便利な機能も多数ある。

## 1.2　OpenGLを利用するにあたっての注意点

### 1.2.1　ワールド座標系とローカル座標系

3次元グラフィックスで用いられる、空間全体を表す座標系をワールド座標系という。物体を配置する最も簡単な方法は、ワールド座標系での座標値を指定して、その位置に物体の点やポリゴンを表示させることである。しかし、点やポリゴンを動かしてアニメーションにする場合、新しい座標値を毎回計算し、更新しなければならない。

そこで、ワールド座標系の中にある個別の物体それぞれにローカル座標系を設定する。これにより、容易に物体の配置を行うことができる。

### 1.2.2 幾何変換

ローカル座標系で表現された物体をワールド座標系の所定の場所に位置づけるために幾何変換を用いる。OpenGL では以下の幾何変換が可能である。

(1) 平行移動

```
glTranslatef(t_x, t_y, t_z);
```

$t_x, t_y, t_z$ は各軸方向への移動量

(2) 回転

```
glRotatef(θ, x, y, z);
```

$\theta$ は回転角度（ °）、$x, y, z$ は回転軸のベクトル

(3) 拡大・縮小

```
glScalef(s_x, s_y, s_z);
```

$s_x, s_y, s_z$ は各軸方向の拡大率

これらの幾何変換によりローカル座標系全体が変換される。

幾何変換で実際にどのような計算がされているかは、参考文献 [1] を見よ。

# 第2章　OpenGLで２変数関数のグラフを描画するためのライブラリィ

## 2.1　作成したプログラムの紹介

OpenGL によって２次元波動方程式を可視化するために、３つの C プログラムから成るライブラリィを作った。１つ目は、OpenGL を利用するために必要な関数をまとめた set_opengl.c である。後の２つは、数値計算の結果を３次元のグラフに描画する draw-graph.c と draw-graph-on-disk.c である。前者は長方形上で定義された関数、後者は円盤上で定義された関数に用いる。

## 2.2　使用方法

### 2.2.1　コンパイルの仕方

自分が書いたプログラムとともに、set-opengl.c、draw-graph.c（または draw-graph-on-disk.c）をコンパイルする。コンパイルの仕方は、

```
ccgl（自分が書いたプログラムの名前).c set-opengl.c draw-graph.c
```

である。

ccgl は、OpenGL と GLUT のライブラリィとコンパイル＆リンクするコマンドである (付録 A 参照)。

### 2.2.2　操作方法

キーボードとマウスを使って描画されたグラフを簡単に操作できる。以下に、操作方法を挙げる。

- r キー ··· 視点位置をリセットする
- w キー ··· 網目の ON/OFF を切り換える
- Space キー ··· アニメーションの再生/停止を切り替える
- Esc キー ··· プログラムを終了する
- マウス左ボタンのドラッグ ··· 視点を変更する
- マウス中ボタンのドラッグ ··· 視線回りに回転する
- マウス右ボタンのドラッグ ··· ズームする

## 2.3 プログラムの書き方

プログラムは、関数 main() 以外に、関数 display(), idle() が必要である。それぞれの関数の役割は以下のとおりである。

- display() ··· 画面に描画する処理を記述する。再描画が必要になる度に繰り返し呼び出される。
- idle() ··· 入力イベントがなくなった場合に、(適当な時間間隔で) 呼び出される。

また、時間変化によるアニメーションを作る場合は、idle() 内に時間ステップを進める処理と、`glutPostRedisplay()` の呼び出しを書くと良い。

### 2.3.1 サンプルプログラム１

時間変化によるアニメーションを作る例を示す。

$$f(x,y,t) = \sin(\sqrt{m^2+n^2}\pi t)\sin(m\pi x)\sin(n\pi y) \quad ((x,y) \in [-1,1]\times[-1,1], t>0)$$

のグラフを描くプログラムである。

```
/* sample-graph1.c --- 時間変化によるアニメーションを作る
 *                     ３変数関数のグラフ
 *
 *   ccgl sample-graph.c draw-graph.c set-opengl.c
```

```
 *
 *   入力の例:
 *     ./sample-graph
 *     Nx, Ny: 40 40
 *
 *   操作方法:
 *     d キー --- Dirichlet 境界条件
 *     n キー --- Neumann 境界条件
 *     r キー --- 視点位置のリセット
 *     w キー --- 網目の ON/OFF
 *     Space キー --- 再生/停止
 *     Esc キー --- 終了
 *     マウス左ボタン --- 視点の変更
 *     マウス左ボタン --- 視線回りの回転
 *     マウス左ボタン --- ズーム
 */

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "draw-graph.h"
#include "set-opengl.h"
#define PI M_PI
#define M 2
#define N 1

int i, j, n, nmax, skip;
/* x, y の区間 */
double xmin = -1.0, xmax = 1.0, ymin = -1.0, ymax = 1.0;
/* 区間の分割数 */
int Nx, Ny;
/* 区間の刻み幅 */
double hx, hy;
```

```
/* 時間の刻み幅 */
double tau;
/* 最終時刻 */
double Tmax;
/* 描画する時間間隔 */
double dt;
double **u;
/* グラフのｚ軸方向の範囲 */
double zmin = -1.0, zmax = 1.0;
/* 視点の位置 */
double distance = 5.0, twist = 0.0, elevation = 60.0, azimuth = 120.0;

void display(void)
{
  /* Δｔの整数倍の時刻ではグラフを描く */
  if(n % skip == 0){

    beginOpenGL();

    /* 鳥瞰図の描画 */
    drawGraph(Nx, Ny, u);

    endOpenGL();
  }
}

void next_step(void)
{
  double x, y;

  /* 時間に関するループ */
  if(n <= nmax){
    for(i=0;i<=Nx;i++){
      x = xmin + i*hx;
      for(j=0;j<=Ny;j++){
        y = ymin + j*hy;
```

```
        u[i][j] = sin(sqrt(M*M*N*N)*PI*n*tau) * sin(M*PI*x) * sin(N*PI*y);
      }
    }
  }
}

void idle(void)
{
  n++;
  next_step();
  glutPostRedisplay();}

int main(int argc, char **argv)
{
  printf("Nx, Ny: ");  scanf("%d %d", &Nx, &Ny);

  hx = (xmax - xmin) / Nx;
  hy = (ymax - ymin) / Ny;

  printf("τ: ");  scanf("%lf", &tau);

  printf("Tmax: ");  scanf("%lf", &Tmax);
  printf("Δt(>=%g): ", tau);  scanf("%lf", &dt);
  if(dt < tau){
    dt = tau;
  }
  skip = rint(dt / tau);
  n = 0;
  nmax = rint(Tmax / tau);

  /* uのメモリーを割り当てる */
  u = malloc(sizeof(double *) * (Nx+1));
  for(i=0;i<=Nx;i++){
    u[i] = malloc(sizeof(double) * (Ny+1));
  }
```

```
  setView(distance, twist, elevation, azimuth);
  setFrame(xmin, xmax, ymin, ymax, zmin, zmax);
  OpenGL(&argc, argv);
  return(0);
}
```

### 2.3.2 サンプルプログラム２

次に、時間変化によるアニメーションを作らない例を示す。

$$f(x,y) = x^2 - y^2 \quad ((x,y) \in [-1,1] \times [-1,1])$$

のグラフを描くプログラムである。

```
/* sample-graph2.c --- 時間変化によるアニメーションを作らない
 *                     (時間変数のない) 2 変数関数のグラフ
 *
 *   ccgl sample-graph.c draw-graph.c set-opengl.c
 *
 *   入力の例:
 *     ./sample-graph
 *     Nx, Ny: 40 40
 *
 *   操作方法:
 *     d キー --- Dirichlet 境界条件
 *     n キー --- Neumann 境界条件
 *     r キー --- 視点位置のリセット
 *     w キー --- 網目の ON/OFF
 *     Space キー --- 再生/停止
 *     Esc キー --- 終了
 *     マウス左ボタン --- 視点の変更
 *     マウス左ボタン --- 視線回りの回転
 *     マウス左ボタン --- ズーム
 */

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

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "draw-graph.h"
#include "set-opengl.h"

int i, j, n, nmax;
/* x, y の区間 */
double xmin = -1.0, xmax = 1.0, ymin = -1.0, ymax = 1.0;
/* 区間の分割数 */
int Nx, Ny;
/* 区間の刻み幅 */
double hx, hy;
double **u;
/* グラフの z 軸方向の範囲 */
double zmin = -1.0, zmax = 1.0;
/* 視点の位置 */
double distance = 5.0, twist = 0.0, elevation = 60.0, azimuth = 120.0;

void display(void)
{
  double x, y;

  beginOpenGL();

  for(i=0;i<=Nx;i++){
    x = xmin + i*hx;
    for(j=0;j<=Ny;j++){
      y = ymin + j*hy;
      u[i][j] = x*x - y*y;
    }
  }

  /* 鳥瞰図の描画 */
  drawGraph(Nx, Ny, u);
```

```
  endOpenGL();
}

void idle(void)
{
}

int main(int argc, char **argv)
{
  printf("Nx, Ny: ");  scanf("%d %d", &Nx, &Ny);

  hx = (xmax - xmin) / Nx;
  hy = (ymax - ymin) / Ny;

  /* u のメモリーを割り当てる */
  u = malloc(sizeof(double *) * (Nx+1));
  for(i=0;i<=Nx;i++){
    u[i] = malloc(sizeof(double) * (Ny+1));
  }

  setView(distance, twist, elevation, azimuth);
  setFrame(xmin, xmax, ymin, ymax, zmin, zmax);
  OpenGL(&argc, argv);
  return(0);
}
```

## 2.4 set-opengl.c

### 2.4.1 set-opengl.c とは

set-opengl.c は、OpenGL を利用するのに必要な関数を集めたプログラムである。以下に各関数の説明をする。

```
polarview()
```

図 2.1 で示すように、物体を常に視野の中心に置き、まわりからその物体を眺める場合に用いられる関数である。

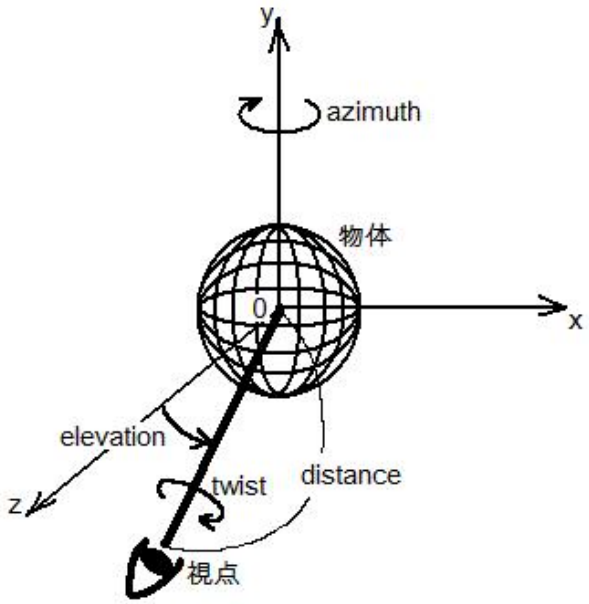


図 2.1: `polarview()` の説明図

- distance··· 視点から物体（原点）までの距離
- twist··· 視線周りの回転角度（首を傾げたときの角度）
- elevation··· 物体を見上げる角度
- azimuth··· 鉛直軸回りの角度

しかし実際は、物体が表現されているローカル座標系を幾何変換することにより、あたかも視点位置が変わったように見せているのである。例えば物体を $x$ 軸方向に $+5$ 平行移動させることは、視点位置を $x$ 軸方向に $-5$ 平行移動させることに相当する。

`resetview()`

polarview() で使用する変数 distance,twist,elevation,azimuth それぞれの初期値が入っている。つまり、実行すると視点位置が初期状態にリセットされる。

`myMotion(`$x, y$`)`

マウスをドラッグしている間、一定の時間間隔で呼び出される。引数の $x, y$ は、myMotion() が呼ばれたときのマウスのポインタの座標位置である。その $x, y$ と、前回 myMotion が呼ばれたときの座標位置 xBegin, yBegin との差をとり、マウスの移動距離 xDisp, yDisp を計算する。

xDisp, yDisp の値を、次々に polarview() で使用する変数 distance, twist, elevation, azimuth に足したり引いたりして、視点が動いたように見える。

### 2.4.2 ソースプログラム

```
/* set-opengl.c */

#include <stdio.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#define KEY_ESC 27
#define KEY_SPC 32
#define DBC 0
#define NBC 1

/* 再生/停止の切り替え */
static unsigned char moveFlag = GL_FALSE;
/* 境界条件の切り替え */
static int bc;

static int xBegin, yBegin;
static int mButton;
static double distance, twist, elevation, azimuth;
static double D = 10.0, T = 0.0, E = 60.0, A = 120.0;

void polarview(void);
void resetview(void);
void idle(void);
void display(void);
```

```
void setView(double d0, double t0, double e0, double a0)
{
  D = d0;  T = t0;  E = e0;  A = a0;
}

/* display 関数の中で最初に書くべき命令を集めた関数 */
void beginOpenGL(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_DEPTH_TEST);

  glPushMatrix();
  polarview();
}

/* display 関数の中で最後に書くべき命令を集めた関数 */
void endOpenGL(void)
{
  glPopMatrix();
  glDisable(GL_DEPTH_TEST);
  glutSwapBuffers();
}

#define MAXCOMMANDS (100)
typedef void vvfunc(void);
static char command_keys[MAXCOMMANDS];
vvfunc *command_funcs[MAXCOMMANDS];
static num_of_reg_commands = 0;

/* c という文字で f という関数を登録する（重複チェックをサボっている）*/
void register_command(char c, vvfunc f)
{
  if (num_of_reg_commands < MAXCOMMANDS) {
    command_keys[num_of_reg_commands] = c;
    command_funcs[num_of_reg_commands++] = f;
  }
```

```
  else {
    fprintf(stderr, "登録したコマンドが多すぎます。\n");
  }
}

void myKbd(unsigned char key, int x, int y)
{
  int i;

  switch(key){
  case 'r':
    resetview();
    break;
  case 'w':
    switchWireFlag();
    break;
  case KEY_SPC:
    moveFlag = !moveFlag;
    if(moveFlag == GL_TRUE)
      glutIdleFunc(idle);
    else
      glutIdleFunc(NULL);
    break;
  case KEY_ESC:
    exit(0);
  default:
    for(i=0;i<num_of_reg_commands;i++){
      if(key == command_keys[i]){
        command_funcs[i]();
        return;
      }
    }
  }
  glutPostRedisplay();
}
```

```
void myMouse(int button, int state, int x, int y)
{
  if(state == GLUT_DOWN){
    xBegin = x;
    yBegin = y;
    mButton = button;
  }
}

void myMotion(int x, int y)
{
  int xDisp, yDisp;

  xDisp = x - xBegin;
  yDisp = y - yBegin;

  switch(mButton){
  case GLUT_LEFT_BUTTON:
    azimuth -= (double)xDisp / 2.0;
    elevation -= (double)yDisp / 2.0;
    break;
  case GLUT_MIDDLE_BUTTON:
    twist = fmod(twist + xDisp/3.0, 360.0);
    break;
  case GLUT_RIGHT_BUTTON:
    distance += (double)yDisp / 60.0;
    break;
  }
  xBegin = x;
  yBegin = y;
  glutPostRedisplay();
}

void myInit(char *progname)
{
  glutInitWindowPosition(0, 0);
```

```
  glutInitWindowSize(400, 400);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutCreateWindow(progname);
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glutKeyboardFunc(myKbd);
  glutMouseFunc(myMouse);
  glutMotionFunc(myMotion);
  resetview();
}

void myReshape(int width, int height)
{
  double aspect = width/(double)height;
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, aspect, 1.0, 80.0);
  glMatrixMode(GL_MODELVIEW);
}

void polarview(void)
{
  glTranslatef(0.0, 0.0, -distance);
  glRotatef(-twist, 0.0, 0.0, 1.0);
  glRotatef(-elevation, 1.0, 0.0, 0.0);
  glRotatef(-azimuth, 0.0, 0.0, 1.0);
}

void resetview(void)
{
  distance = D;
  twist = T;
  elevation = E;
  azimuth = A;
}
```

```
void OpenGL(int *argc, char **argv)
{
  glutInit(argc, argv);
  myInit(argv[0]);
  glutReshapeFunc(myReshape);
  glutIdleFunc(NULL);
  glutDisplayFunc(display);
  glutMainLoop();
}
```

## 2.5 draw-graph.c

### 2.5.1 draw-graph.c とは

数値計算の結果を 3 次元のグラフに描画するプログラムである。

### 2.5.2 ソースプログラム

```
/* draw-graph.c */

#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>

static double xmin = -2.0, xmax = 2.0, ymin = -2.0, ymax = 2.0,
              zmin = -1.0, zmax =1.0;
static unsigned char wireFlag = GL_TRUE;

void setFrame(double x0, double x1, double y0, double y1, double z0, double z1)
{
  xmin = x0;  xmax = x1;
  ymin = y0;  ymax = y1;
  zmin = z0;  zmax = z1;
}
```

```
void setWireFlag(unsigned char wireFlag0)
{
  wireFlag = wireFlag0;
}

void switchWireFlag(void)
{
  wireFlag = !wireFlag;
}

/* 枠の描画 */
static void drawFrame(void)
{
  glColor3f(0.0, 0.0, 0.0);
  glBegin(GL_LINE_LOOP);
  glVertex3f(xmin,ymin,zmin);
  glVertex3f(xmax,ymin,zmin);
  glVertex3f(xmax,ymax,zmin);
  glVertex3f(xmin,ymax,zmin);
  glEnd();
  glBegin(GL_LINES);
  glVertex3f(xmin,ymin,zmin);  glVertex3f(xmin,ymin,zmax);
  glVertex3f(xmax,ymin,zmin);  glVertex3f(xmax,ymin,zmax);
  glVertex3f(xmax,ymax,zmin);  glVertex3f(xmax,ymax,zmax);
  glVertex3f(xmin,ymax,zmin);  glVertex3f(xmin,ymax,zmax);
  glEnd();
}

/* 網目のグラフの描画 */
static void drawWireGraph(int Nx, int Ny, double **u)
{
  int i, j;
  double x0, x1, y0, y1;
  double hx = (xmax - xmin) / Nx;
  double hy = (ymax - ymin) / Ny;
```

```
  for(i=0;i<Nx;i++){
    x0 = xmin + i * hx;
    x1 = xmin + (i+1) * hx;
    for(j=0;j<Ny;j++){
      y0 = ymin + j * hy;
      y1 = ymin + (j+1) * hy;
      glColor3f(0.0, 0.0, 0.0);
      glBegin(GL_LINE_LOOP);
      glVertex3f(x0, y0, u[i][j]);
      glVertex3f(x1, y0, u[i+1][j]);
      glVertex3f(x1, y1, u[i+1][j+1]);
      glVertex3f(x0, y1, u[i][j+1]);
      glEnd();
    }
  }
}

/* u[i][j] の値による色の R 成分 */
static double red(double v)
{
  if(v <= 0.4)
    return 0.0;
  else if(v <= 0.5)
    return (v - 0.4) / 0.1;
  else
    return 1.0;
}

/* u[i][j] の値による色の G 成分 */
static double green(double v)
{
  if(v <= 0.3)
    return v / 0.3;
  else if(v <= 0.6)
    return 1.0;
```

```
  else
    return -(v - 1) / 0.4;
}

/* u[i][j] の値による色の B 成分 */
static double blue(double v)
{
  if(v <= 0.3)
    return 1.0;
  else if(v <= 0.4)
    return -(v - 0.4) / 0.1;
  else
    return 0.0;
}

/* u[i][j] の値による色の塗り分け */
static void color(double v)
{
  double R, G, B;

  if(v < 0)  v = 0;
  if(v > 1)  v = 1;
  R = red(v);
  G = green(v);
  B = blue(v);
  glColor3f(R, G, B);
}

/* 塗りつぶしたグラフの描画 */
static void drawSolidGraph(int Nx, int Ny, double **u)
{
  int i, j;
  double x0, x1, y0;
  double hx = (xmax - xmin) / Nx;
  double hy = (ymax - ymin) / Ny;
  double v;
```

```
  glEnable(GL_POLYGON_OFFSET_FILL); /* ポリゴンオフセットの有効範囲の指
定 */
  glPolygonOffset(1.0, 1.0);        /* 面上に線を描いたときのちらつき
を防ぐ */

  for(i=0;i<Nx;i++){
    x0 = xmin + i * hx;
    x1 = xmin + (i+1) * hx;
    glBegin(GL_QUAD_STRIP);
    for(j=0;j<=Ny;j++){
      y0 = ymin + j * hy;

      v = (u[i][j] - zmin) / (zmax - zmin);
      color(v);

      glVertex3f(x0, y0, u[i][j]);
      glVertex3f(x1, y0, u[i+1][j]);
    }
    glEnd();
  }

  glDisable(GL_POLYGON_OFFSET_FILL);
}

/* グラフの描画 */
void drawGraph(int Nx, int Ny, double **u)
{
  /* 枠の描画 */
  drawFrame();

  /* 塗りつぶしたグラフの描画 */
  drawSolidGraph(Nx, Ny, u);

  if(wireFlag == GL_TRUE)
    /* 網目のグラフの描画 */
```

```
    drawWireGraph(Nx, Ny, u);
}
```

## 2.6 draw-graph-on-disk.c

### 2.6.1 draw-graph-on-disk.c とは

特に円盤上での数値計算の結果を３次元のグラフに描画するプログラムである。

### 2.6.2 ソースプログラム

```
/* draw-graph-on-disk.c */

#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#define PI M_PI

static double R = 1.0, zmin = -1.0, zmax = 1.0;
static unsigned char wireFlag = GL_TRUE;

void setFrame(double r0, double z0, double z1)
{
  R = r0;  zmin = z0;  zmax = z1;
}

void setWireFlag(unsigned char wireFlag0)
{
  wireFlag = wireFlag0;
}

void switchWireFlag(void)
{
  wireFlag = !wireFlag;
```

```
}

/* 枠の描画 */
static void drawFrame(void)
{
  int i, j;
  double x, y;

  glColor3f(0.0, 0.0, 0.0);
  glBegin(GL_LINES);
  for(i=0;i<360;i+=90){
    x = R * cos(PI / 180.0 * i);
    y = R * sin(PI / 180.0 * i);
    glVertex3f(x, y, zmin);  glVertex3f(x, y, zmax);
  }
  glEnd();
  glBegin(GL_LINE_LOOP);
  for(j=0;j<360;j++){
    x = R * cos(PI / 180.0 * j);
    y = R * sin(PI / 180.0 * j);
    glVertex3f(x, y, zmin);
  }
  glEnd();
}

/* 網目のグラフの描画 */
static void drawWireGraph(int Nr, int Np, double **u)
{
  int i, j;
  double r0, r1, p0, p1;
  double hr = R / Nr;
  double hp = 2*PI / Np;

  for(i=0;i<Nr;i++){
    r0 = i * hr;
    r1 = (i+1) * hr;
```

```
    for(j=0;j<Np;j++){
      p0 = j * hp;
      p1 = (j+1) * hp;
      glColor3f(0.0, 0.0, 0.0);
      glBegin(GL_LINE_LOOP);
      glVertex3f(r0*cos(p0), r0*sin(p0), u[i][j]);
      glVertex3f(r1*cos(p0), r1*sin(p0), u[i+1][j]);
      glVertex3f(r1*cos(p1), r1*sin(p1), u[i+1][j+1]);
      glVertex3f(r0*cos(p1), r0*sin(p1), u[i][j+1]);
      glEnd();
    }
  }
}

/* u[i][j] の値による色の R 成分 */
static double red(double v)
{
  if(v <= 0.4)
    return 0.0;
  else if(v <= 0.5)
    return (v - 0.4) / 0.1;
  else
    return 1.0;
}

/* u[i][j] の値による色の G 成分 */
static double green(double v)
{
  if(v <= 0.3)
    return v / 0.3;
  else if(v <= 0.6)
    return 1.0;
  else
    return -(v - 1) / 0.4;
}
```

```
/* u[i][j] の値による色の B 成分 */
static double blue(double v)
{
  if(v <= 0.3)
    return 1.0;
  else if(v <= 0.4)
    return -(v - 0.4) / 0.1;
  else
    return 0.0;
}

/* u[i][j] の値による色の塗り分け */
static void color(double v)
{
  double R, G, B;

  if(v < 0)  v = 0;
  if(v > 1)  v = 1;
  R = red(v);
  G = green(v);
  B = blue(v);
  glColor3f(R, G, B);
}

/* 塗りつぶしたグラフの描画 */
static void drawSolidGraph(int Nr, int Np, double **u)
{
  int i, j;
  double r0, r1, p0;
  double hr = R / Nr;
  double hp = 2*PI / Np;
  double v;

  glEnable(GL_POLYGON_OFFSET_FILL); /* ポリゴンオフセットの有効範囲の指
定 */
  glPolygonOffset(1.0, 1.0);        /* 面上に線を描いたときのちらつき
```

```
を防ぐ */

  for(i=0;i<Nr;i++){
    r0 = i * hr;
    r1 = (i+1) * hr;
    glBegin(GL_QUAD_STRIP);
    for(j=0;j<=Np;j++){
      p0 = j * hp;

      v = (u[i][j] - zmin) / (zmax - zmin);
      color(v);

      glVertex3f(r0*cos(p0), r0*sin(p0), u[i][j]);
      glVertex3f(r1*cos(p0), r1*sin(p0), u[i+1][j]);
    }
    glEnd();
  }

  glDisable(GL_POLYGON_OFFSET_FILL);
}

/* グラフの描画 */
void drawGraph(int Nr, int Np, double **u)
{
  /* 枠の描画 */
  drawFrame();

  /* 塗りつぶしたグラフの描画 */
  drawSolidGraph(Nr, Np, u);

  if(wireFlag == GL_TRUE)
    /* 網目のグラフの描画 */
    drawWireGraph(Nr, Np, u);
}
```

# 第3章　長方形領域上の波動方程式の初期値境界値問題

## 3.1　問題

波動方程式

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad ((x,y) \in \Omega, t > 0) \tag{3.1}$$

$$\Omega = (A,B) \times (C,D)$$

と初期条件

$$u(x,y,0) = \phi(x,y) \quad ((x,y) \in \overline{\Omega}) \tag{3.2}$$

$$\frac{\partial u}{\partial t}(x,y,0) = \psi(x,y) \quad ((x,y) \in \overline{\Omega}) \tag{3.3}$$

と次の境界条件のいずれか

$$\text{(Dirichlet 境界条件)} \qquad u(x,y,t)|_{\partial\Omega} = 0 \quad (t > 0) \tag{3.4}$$

$$\text{(Neumann 境界条件)} \qquad \left.\frac{\partial u}{\partial n}(x,y,t)\right|_{\partial\Omega} = 0 \quad (t > 0) \tag{3.5}$$

からなる初期値境界値問題を考える。

## 3.2　差分法

差分法（陽解法）で解く。

空間変数 $x$ の区間 $[A,B]$ を $N_x$ 等分、$y$ の区間 $[C,D]$ を $N_y$ 等分し、等分点をそれぞれ $x_i (i = 0, \cdots, N_x), y_j (j = 0, \cdots, N_y)$ とする。すなわち、刻み幅を

$$h_x = \frac{B-A}{N_x}, \quad h_y = \frac{D-C}{N_y}$$

として、

$$x_i = A + ih_x \quad (i = 0, 1, 2, \cdots, N_x),$$
$$y_j = C + jh_y \quad (j = 0, 1, 2, \cdots, N_y).$$

次に時間変数 $t$ に関する刻み幅 $\tau(> 0)$ を一つ定めて、

$$t_n = n\tau \quad (n = 0, 1, 2, \cdots)$$

とおく。

各格子点 $(x_i, y_j, t_n)$ における $u$ の値を $u^n_{i,j} \equiv u(x_i, y_j, t_n)$ とおく。格子点 $(x_i, y_j, t_n)$ において偏微分係数 $\dfrac{\partial^2 u}{\partial t^2}, \dfrac{\partial^2 u}{\partial x^2}, \dfrac{\partial^2 u}{\partial y^2}$ をそれぞれ 2 階中心差分近似すると、

$$\frac{\partial^2 u}{\partial t^2}(x_i, y_j, t_n) = \frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\tau^2} + O(\tau^2) \quad (\tau \to +0),$$
$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j, t_n) = \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{h_x^2} + O(h_x^2) \quad (h_x \to +0),$$
$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j, t_n) = \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{h_y^2} + O(h_y^2) \quad (h_y \to +0)$$

となる。$u$ は波動方程式 (3.1) を満たすので、$h_x, h_y, \tau$ が十分小さいとき、

$$\frac{1}{c^2}\frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\tau^2} \fallingdotseq \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{h_x^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{h_y^2}$$

が成り立つ。

そこで格子点 $(x_i, y_j, t_n)$ での $u$ の近似値 $U^n_{i,j}$ を決定する方程式として

$$\frac{1}{c^2}\frac{U^{n+1}_{i,j} - 2U^n_{i,j} + U^{n-1}_{i,j}}{\tau^2} = \frac{U^n_{i+1,j} - 2U^n_{i,j} + U^n_{i-1,j}}{h_x^2} + \frac{U^n_{i,j+1} - 2U^n_{i,j} + U^n_{i,j-1}}{h_y^2}$$

を考える。$\lambda_x = c\tau/h_x, \lambda_y = c\tau/h_y$ とおき、整理すると

$$U^{n+1}_{i,j} = 2(1 - \lambda_x^2 - \lambda_y^2)U^n_{i,j} + \lambda_x^2(U^n_{i-1,j} + U^n_{i+1,j}) + \lambda_y^2(U^n_{i,j-1} + U^n_{i,j+1}) - U^{n-1}_{i,j}$$
$$(i = 1, 2, \cdots, N_x - 1; j = 1, 2, \cdots, N_y - 1; n = 1, 2, \cdots) \qquad (3.6)$$

となる。

一方、初期条件 (3.2) より

$$U^0_{i,j} = \phi(x_i, y_j) \quad (i = 0, 1, 2, \cdots, N_x; j = 0, 1, 2, \cdots, N_y), \qquad (3.7)$$

(3.3) については、$u(x_i, y_j, \tau)$ を $t = 0$ の周りで Taylor 展開すると

$$\begin{aligned} u(x_i, y_j, \tau) &= u(x_i, y_j, 0) + \tau \frac{\partial u}{\partial t}(x_i, y_j, 0) + \frac{\tau^2}{2}\frac{\partial^2 u}{\partial t^2}(x_i, y_j, 0) + O(\tau^3) \\ &= u(x_i, y_j, 0) + \tau\psi(x_i, y_j) + \frac{\tau^2}{2}\frac{\partial^2 u}{\partial t^2}(x_i, y_j, 0) + O(\tau^3) \end{aligned}$$

となり、式 (3.1) が $t = 0$ のときも成り立つとすると

$$u(x_i, y_j, \tau) = u(x_i, y_j, 0) + \tau\psi(x_i, y_j) + \frac{c^2\tau^2}{2}\left(\frac{\partial^2 u}{\partial x^2}(x_i, y_j, 0) + \frac{\partial^2 u}{\partial y^2}(x_i, y_j, 0)\right) + O(\tau^3).$$

$\dfrac{\partial^2 u}{\partial x^2}, \dfrac{\partial^2 u}{\partial y^2}$ をそれぞれ 2 階中心差分近似すると

$$\begin{aligned} u(x_i, y_j, \tau) = u(x_i, y_j, 0) &+ \tau\psi(x_i, y_j) \\ &+ \frac{c^2\tau^2}{2}\left(\frac{u^0_{i+1,j} - 2u^0_{i,j} + u^0_{i-1,j}}{h_x^2} + \frac{u^0_{i,j+1} - 2u^0_{i,j} + u^0_{i,j-1}}{h_y^2}\right) \\ &+ O(h_x^2) + O(h_y^2) + O(\tau^3). \end{aligned}$$

ゆえに

$$U^1_{i,j} = U^0_{i,j} + \tau\psi(A + ih_x, C + jh_y) + \frac{c^2\tau^2}{2}\left(\frac{U^0_{i-1,j} - 2U^0_{i,j} + U^0_{i+1,j}}{h_x^2} + \frac{U^0_{i,j-1} - 2U^0_{i,j} + U^0_{i,j+1}}{h_y^2}\right)$$

と書ける。整理すると、

$$\begin{aligned} U^1_{i,j} = (1 - \lambda_x^2 - \lambda_y^2)U^0_{i,j} + \frac{\lambda_x^2}{2}(U^0_{i-1,j} + U^0_{i+1,j}) + \frac{\lambda_y^2}{2}(U^0_{i,j-1} + U^0_{i,j+1}) + \tau g(ih_r, jh_\theta) \\ (i = 1, 2, \cdots, N_x - 1; j = 1, 2, \cdots, N_y - 1) \end{aligned} \tag{3.8}$$

が得られる。

Dirichlet 境界条件 (3.4) より

$$\begin{aligned} U^n_{0,j} = U^n_{N_x,j} = 0 \quad (j = 0, 1, 2, \cdots, N_y; n = 1, 2, \cdots), \\ U^n_{i,0} = U^n_{i,N_y} = 0 \quad (i = 0, 1, 2, \cdots, N_x; n = 1, 2, \cdots). \end{aligned}$$

また、Neumann 境界条件 (3.5) は、$\dfrac{\partial u}{\partial x}(x_0, y_j, t_n), \dfrac{\partial u}{\partial y}(x_i, y_0, t_n)$ を前進差分近似、$\dfrac{\partial u}{\partial x}(x_{N_x}, y_j, t_n), \dfrac{\partial u}{\partial y}(x_i, y_{N_y}, t_n)$ を後退差分近似すると、

$$\begin{aligned} \frac{U^n_{1,j} - U^n_{0,j}}{h_x} = 0, \quad \frac{U^n_{N_x,j} - U^n_{N_x-1,j}}{h_x} = 0, \\ \frac{U^n_{i,1} - U^n_{i,0}}{h_y} = 0, \quad \frac{U^n_{i,N_y} - U^n_{i,N_y-1}}{h_y} = 0 \end{aligned}$$

と書ける。つまり

$$\begin{cases} U^n_{1,j} = U^n_{0,j}, \quad U^n_{N_x,j} = U^n_{N_x-1,j} \quad (j = 0, 1, 2, \cdots, N_y; n = 1, 2, \cdots) \\ U^n_{i,1} = U^n_{i,0}, \quad U^n_{i,N_y} = U^n_{i,N_y-1} \quad (i = 0, 1, 2, \cdots, N_x; n = 1, 2, \cdots). \end{cases}$$

## 3.3 ソースプログラム

```
/* wave2-opengl.c --- 波動方程式 (長方形領域)
 *
 *   ccgl wave2-opengl.c set-opengl.c draw-graph.c
 *
 *   入力の例:
 *     ./wave2-opengl
 *     Nx, Ny: 40 40
 *     τ : 0.01
 *     Tmax: 100(長く計算したければ大きい値を入れる)
 *     Δ t: 0.02
 *
 *   操作方法:
 *     d キー --- Dirichlet 境界条件
 *     n キー --- Neumann 境界条件
 *     r キー --- 視点位置のリセット
 *     w キー --- 網目の ON/OFF
 *     Space キー --- 再生/停止
 *     Esc キー --- 終了
 *     マウス左ボタン --- 視点の変更
 *     マウス中ボタン --- 視線回りの回転
 *     マウス右ボタン --- ズーム
 */

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
```

```
#include "draw-graph.h"
#include "set-opengl.h"
#define PI M_PI
#define DBC 0
#define NBC 1

int i, j, n, nmax, skip;
/* x, y の区間 */
double xmin = -2.0, xmax = 2.0, ymin = -2.0, ymax = 2.0;
/* 区間の分割数 */
int Nx, Ny;
/* 区間の刻み幅 */
double hx, hy;
/* 時間の刻み幅 */
double tau;
/* λ x, λ y, λ x^2, λ y^2 */
double lambdax, lambday, lambdax2, lambday2;
/* 最終時刻 */
double Tmax;
/* 描画する時間間隔 */
double dt;
/* u1[i][j]=u_{i,j}^{n-1}
   u2[i][j]=u_{i,j}^{n}
   u3[i][j]=u_{i,j}^{n+1} */
double **u1, **u2, **u3;
/* 境界条件の切り替え */
static int bc = DBC;
/* グラフの z 軸方向の範囲 */
double zmin = -1.0, zmax = 1.0;
/* 視点の位置 */
double distance = 10.0, twist = 0.0, elevation = 60.0, azimuth = 120.0;

void setBC(int bc0)
{
  bc = bc0;
}
```

```
void setDBC()
{
  setBC(DBC);
}

void setNBC()
{
  setBC(NBC);
}

void display(void)
{
  /* Δt の整数倍の時刻ではグラフを描く */
  if(n % skip == 0){

    beginOpenGL();

    /* t=0 のとき */
    if(n == 0){
      /* 鳥瞰図の描画 */
      drawGraph(Nx, Ny, u1);
    }

    /* t=n τ (n>=1) のとき */
    else{
      /* 鳥瞰図の描画 */
      drawGraph(Nx, Ny, u2);
    }

    endOpenGL();
  }
}

double phi(double x, double y)
{
```

```
  return sin(PI*x)/2.0 + sin(PI*y)/2.0;
}

double psi(double x, double y)
{
  return 0.0;
}

void first_step(void)
{
  /* 初期値 */

  /* u_{i,j}^0 = φ(xi,yj) */
  for(i=0;i<=Nx;i++){
    for(j=0;j<=Ny;j++){
      u1[i][j] = phi(xmin + i*hx, ymin + j*hy);
    }
  }

  /* u_{i,j}^1 = (1-λx^2-λy^2)... */
  for(i=1;i<Nx;i++){
    for(j=1;j<Ny;j++){
      u2[i][j] = (1.0 - lambdax2 - lambday2) * u1[i][j]
        + 0.5 * lambdax2 * (u1[i-1][j] + u1[i+1][j])
        + 0.5 * lambday2 * (u1[i][j-1] + u1[i][j+1])
        + tau * psi(xmin + i*hx, ymin + j*hy);
    }
  }
}

void next_step(void)
{
  /* 時間に関するループ */
  if(n <= nmax){
    for(i=1;i<Nx;i++){
      for(j=1;j<Ny;j++){
```

```
    u3[i][j] = 2.0 * (1.0 - lambdax2 - lambday2) * u2[i][j]
      + lambdax2 * (u2[i-1][j] + u2[i+1][j])
      + lambday2 * (u2[i][j-1] + u2[i][j+1])
      - u1[i][j];
  }
}

/* Dirichlet 境界条件 */
if(bc == DBC){
  for(i=0;i<=Nx;i++){
    u3[i][0] = u3[i][Ny] = 0.0;
  }
  for(j=0;j<=Ny;j++){
    u3[0][j] = u3[Nx][j] = 0.0;
  }
}

/* Neumann 境界条件 */
if(bc == NBC){
  for(i=0;i<=Nx;i++){
    u3[i][0] = u3[i][1];
    u3[i][Ny] = u3[i][Ny-1];
  }
  for(j=0;j<=Ny;j++){
    u3[0][j] = u3[1][j];
    u3[Nx][j] = u3[Nx-1][j];
  }
}

/* u1 <- u2, u2 <- u3 */
for(i=0;i<=Nx;i++){
  for(j=0;j<=Ny;j++){
    u1[i][j] = u2[i][j];
    u2[i][j] = u3[i][j];
  }
}
```

```
  }
}

void idle(void)
{
  n++;
  next_step();
  glutPostRedisplay();
}

int main(int argc, char **argv)
{
  printf("Nx, Ny: ");  scanf("%d %d", &Nx, &Ny);

  hx = (xmax - xmin) / Nx;
  hy = (ymax - ymin) / Ny;

  printf("τ: ");  scanf("%lf", &tau);

  lambdax = tau / hx;
  lambday = tau / hy;
  lambdax2 = lambdax * lambdax;
  lambday2 = lambday * lambday;

  printf("Tmax: ");  scanf("%lf", &Tmax);
  printf("Δ t(>=%g): ", tau);  scanf("%lf", &dt);
  if(dt < tau){
    dt = tau;
  }
  skip = rint(dt / tau);
  n = 0;
  nmax = rint(Tmax / tau);

  /* u1, u2, u3 のメモリーを割り当てる */
  u1 = malloc(sizeof(double *) * (Nx+1));
  u2 = malloc(sizeof(double *) * (Nx+1));
```

```
  u3 = malloc(sizeof(double *) * (Nx+1));
  for(i=0;i<=Nx;i++){
    u1[i] = malloc(sizeof(double) * (Ny+1));
    u2[i] = malloc(sizeof(double) * (Ny+1));
    u3[i] = malloc(sizeof(double) * (Ny+1));
  }

  register_command('d', setDBC);
  register_command('n', setNBC);

  setView(distance, twist, elevation, azimuth);
  setFrame(xmin, xmax, ymin, ymax, zmin, zmax);
  first_step();
  OpenGL(&argc, argv);
  return(0);
}
```

# 第4章　円盤領域上の波動方程式の初期値境界値問題

## 4.1　問題と厳密解

波動方程式

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad (t > 0, (x, y) \in \Omega) \tag{4.1}$$

$$\Omega = \{(x, y) \in \mathbf{R}^2 | x^2 + y^2 < 1\}$$

極座標

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$$

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} \quad ((r, \theta) \in (0, 1) \times [0, 2\pi), t > 0) \tag{4.2}$$

と初期条件

$$u(r, \theta, 0) = f(r, \theta) \quad ((r, \theta) \in [0, 1] \times [0, 2\pi]) \tag{4.3}$$

$$\frac{\partial u}{\partial t}(r, \theta, 0) = g(r, \theta) \quad ((r, \theta) \in [0, 1] \times [0, 2\pi]) \tag{4.4}$$

と次の境界条件のいずれか

$$\text{(Dirichlet 境界条件)} \qquad u(1, \theta, t) = 0 \quad (\theta \in [0, 2\pi), t > 0) \tag{4.5}$$

$$\text{(Neumann 境界条件)} \quad \frac{\partial u}{\partial r}(1, \theta, t) = 0 \quad (\theta \in [0, 2\pi), t > 0) \tag{4.6}$$

からなる初期値境界値問題を考える。Bessel 関数を利用し、Fourier の方法で解く。詳しくは参考文献 [5] を見よ。

## 4.2 差分法

差分法（陽解法）で解く。

空間変数 $r$ の区間 $[0, R]$ を $N_r$ 等分、$\theta$ の区間 $[0, 2\pi]$ を $N_\theta$ 等分し、等分点をそれぞれ $r_i (i = 0, \cdots, N_r), \theta_j (j = 0, \cdots, N_\theta)$ とする。すなわち、刻み幅を

$$h_r = \frac{R}{N_r}, \quad h_\theta = \frac{2\pi}{N_\theta}$$

として、

$$\begin{aligned} r_i &= ih_r \quad (i = 0, 1, 2, \cdots, N_r), \\ \theta_j &= jh_\theta \quad (j = 0, 1, 2, \cdots, N_\theta). \end{aligned}$$

次に時間変数 $t$ に関する刻み幅 $\tau (> 0)$ を一つ定めて、

$$t_n = n\tau \quad (n = 0, 1, 2, \cdots)$$

とおく。

各格子点 $(r_i, \theta_j, t_n)$ における $u$ の値を $u^n_{i,j} \equiv u(r_i, \theta_j, t_n)$ とおく。格子点 $(r_i, \theta_j, t_n)$ において偏微分係数 $\dfrac{\partial u}{\partial r}$ を 1 階中心差分近似、$\dfrac{\partial^2 u}{\partial t^2}, \dfrac{\partial^2 u}{\partial r^2}, \dfrac{\partial^2 u}{\partial \theta^2}$ をそれぞれ 2 階中心差分近似すると、

$$\begin{aligned} \frac{\partial u}{\partial r}(r_i, \theta_j, t_n) &= \frac{u^n_{i+1,j} - u^n_{i-1,j}}{2h_r} + O(h_r^2) \quad (h_r \to +0), \\ \frac{\partial^2 u}{\partial t^2}(r_i, \theta_j, t_n) &= \frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\tau^2} + O(\tau^2) \quad (\tau \to +0), \\ \frac{\partial^2 u}{\partial r^2}(r_i, \theta_j, t_n) &= \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{h_r^2} + O(h_r^2) \quad (h_r \to +0), \\ \frac{\partial^2 u}{\partial \theta^2}(r_i, \theta_j, t_n) &= \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{h_\theta^2} + O(h_\theta^2) \quad (h_\theta \to +0) \end{aligned}$$

となる。$u$ は波動方程式 (4.1) を満たすので、$h_r, h_\theta, \tau$ が十分小さいとき、

$$\frac{1}{c^2}\frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\tau^2} \fallingdotseq \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{h_r^2} + \frac{1}{r_i}\frac{u^n_{i+1,j} - u^n_{i-1,j}}{2h_r} + \frac{1}{r_i^2}\frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{h_\theta^2}$$

が成り立つ。

そこで格子点 $(r_i, \theta_j, t_n)$ での $u$ の近似値 $U^n_{i,j}$ を決定する方程式として

$$\frac{1}{c^2}\frac{U^{n+1}_{i,j} - 2U^n_{i,j} + U^{n-1}_{i,j}}{\tau^2} \fallingdotseq \frac{U^n_{i+1,j} - 2U^n_{i,j} + U^n_{i-1,j}}{h_r^2} + \frac{1}{r_i}\frac{U^n_{i+1,j} - U^n_{i-1,j}}{2h_r} + \frac{1}{r_i^2}\frac{U^n_{i,j+1} - 2U^n_{i,j} + U^n_{i,j-1}}{h_\theta^2}$$

を考える。$\lambda_r = c\tau/h_r, \lambda_\theta = c\tau/h_\theta$ とおき、整理すると

$$U_{i,j}^{n+1} = 2\left(1 - \lambda_r^2 - \frac{\lambda_\theta^2}{r_i^2}\right) U_{i,j}^n + \lambda_r^2 (U_{i-1,j}^n + U_{i+1,j}^n) + \frac{1}{r_i}\frac{c^2\tau^2}{2h_r}(U_{i+1,j}^n - U_{i-1,j}^n) + \frac{\lambda_\theta^2}{r_i^2}(U_{i,j-1}^n + U_{i,j+1}^n) - U_{i,j}^{n-1}$$
$$(i = 1, 2, \cdots, N_r - 1; j = 1, 2, \cdots, N_\theta - 1; n = 1, 2, \cdots) \qquad (4.7)$$

となる。

一方、初期条件 (4.3) より

$$U_{i,j}^0 = f(r_i, \theta_j) \quad (i = 0, 1, 2, \cdots, N_r; j = 0, 1, 2, \cdots, N_\theta), \qquad (4.8)$$

(4.4) については、$u(r, \theta, \tau)$ を $t = 0$ の周りで Taylor 展開すると

$$u(r,\theta,\tau) = u(r,\theta,0) + \tau\frac{\partial u}{\partial t}(r,\theta,0) + \frac{\tau^2}{2}\frac{\partial^2 u}{\partial t^2}(r,\theta,0) + O(\tau^3)$$
$$= u(r,\theta,0) + \tau g(r,\theta) + \frac{\tau^2}{2}\frac{\partial^2 u}{\partial t^2}(r,\theta,0) + O(\tau^3)$$

となり、式 (4.1) が $t = 0$ のときも成り立つとすると

$$u(r,\theta,\tau) = u(r,\theta,0) + \tau g(r,\theta) + \frac{c^2\tau^2}{2}\left(\frac{\partial^2 u}{\partial r^2}(r,\theta,0) + \frac{1}{r}\frac{\partial u}{\partial r}(r,\theta,0) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}(r,\theta,0)\right) + O(\tau^3).$$

$\dfrac{\partial u}{\partial r}$ を 1 階中心差分近似、$\dfrac{\partial^2 u}{\partial r^2}, \dfrac{\partial^2 u}{\partial \theta^2}$ をそれぞれ 2 階中心差分近似すると

$$u(r,\theta,\tau) = u(r,\theta,0) + \tau g(r,\theta) + \frac{c^2\tau^2}{2}\left(\frac{u_{i+1,j}^0 - 2u_{i,j}^0 + u_{i-1,j}^0}{h_r^2} + \frac{1}{r}\frac{u_{i+1,j}^0 - u_{i-1,j}^0}{2h_r} + \frac{1}{r^2}\frac{u_{i,j+1}^0 - 2u_{i,j}^0 + u_{i,j-1}^0}{h_\theta^2}\right) + 2O(h_r^2) + O(h_\theta^2) + O(\tau^3).$$

ゆえに

$$U_{i,j}^1 = U_{i,j}^0 + \tau g(ih_r, jh_\theta) + \frac{c^2\tau^2}{2}\left(\frac{U_{i+1,j}^0 - 2U_{i,j}^0 + U_{i-1,j}^0}{h_r^2} + \frac{1}{r_i}\frac{U_{i+1,j}^0 - U_{i-1,j}^0}{2h_r} + \frac{1}{r_i^2}\frac{U_{i,j+1}^0 - 2U_{i,j}^0 + U_{i,j-1}^0}{h_\theta^2}\right)$$

と書ける。整理すると、

$$U_{i,j}^1 = \left(1 - \lambda_r^2 - \frac{\lambda_\theta^2}{r_i^2}\right) U_{i,j}^0 + \frac{\lambda_r^2}{2}(U_{i-1,j}^0 + U_{i+1,j}^0) + \frac{1}{r_i^2}\frac{\tau^2}{2h_r}(U_{i+1,j}^0 - U_{i-1,j}^0) + \frac{\lambda_\theta^2}{r_i^2}(U_{i,j-1}^0 + U_{i,j+1}^0) + \tau g(ih_r, jh_\theta)$$
$$(i = 1, 2, \cdots, N_r - 1; j = 1, 2, \cdots, N_\theta - 1) \qquad (4.9)$$

が得られる。

Dirichlet 境界条件 (4.5) より

$$U^n_{N_r,j} = 0 \quad (j = 0, 1, 2, \cdots, N_\theta; n = 1, 2, \cdots)$$

また、Neumann 境界条件 (4.6) は、$\dfrac{\partial u}{\partial r}(r_{N_r}, \theta_j, t_n)$ を後退差分近似すると、

$$\frac{U^n_{N_r,j} - U^n_{N_r-1,j}}{h_r} = 0$$

と書ける。つまり

$$U^n_{N_r,j} = U^n_{N_r-1,j} \quad (j = 0, 1, 2, \cdots, N_\theta; n = 1, 2, \cdots).$$

## 4.3 差分解の安定性

差分解の安定性については、参考文献 [5] を見よ。

## 4.4 ソースプログラム

```
/* wave2d-disk-e-opengl.c --- 波動方程式 (円盤領域)
 *
 *   ccgl wave2d-disk-e-opengl.c set-opengl.c draw-graph-on-disk.c
 *
 *   入力の目安:
 *     ./wave2d-disk-e-opengl
 *     Nx, Ny: 40 100
 *     τ : 0.01
 *     Tmax: 100(長く計算したければ大きい値を入れる)
 *     Δ t: 0.02
 *
 *   操作方法:
 *     d キー --- Dirichlet 境界条件
 *     n キー --- Neumann 境界条件
 *     r キー --- 視点位置のリセット
 *     w キー --- 網目の ON/OFF
```

```
 *      Space キー --- 再生/停止
 *      Esc キー --- 終了
 *      マウス左ボタン --- 視点の変更
 *      マウス中ボタン --- 視線回りの回転
 *      マウス右ボタン --- ズーム
 */

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "draw-graph-on-disk.h"
#include "set-opengl.h"
#define PI M_PI
#define DBC 0
#define NBC 1

int i, j, n, nmax, skip;
/* r の区間 */
double R = 2.0;
/* 区間の分割数 */
int Nr, Np;
/* 区間の刻み幅 */
double hr, hp;
/* 時間の刻み幅 */
double tau;
/* λ r, λ θ , λ r^2, λ θ ^2 */
double lambdar, lambdap, lambdar2, lambdap2;
/* 最終時刻 */
double Tmax;
/* 描画する時間間隔 */
double dt;
/* u1[i][j]=u_{i,j}^{n-1}
   u2[i][j]=u_{i,j}^{n}
```

```
   u3[i][j]=u_{i,j}^{n+1} */
double **u1, **u2, **u3;
/* 境界条件の切り替え */
static int bc = DBC;
/* グラフのｚ軸方向の範囲 */
double zmin = -0.8, zmax = 0.8;
/* 視点の位置 */
double distance = 5.0, twist = 0.0, elevation = 60.0, azimuth = 120.0;

void setBC(int bc0)
{
  bc = bc0;
}

void display(void)
{
  /* Δｔの整数倍の時刻ではグラフを描く */
  if(n % skip == 0){

    beginOpenGL();

    /* t=0のとき */
    if(n == 0){
      /* 鳥瞰図の描画 */
      drawGraph(Nr, Np, u1);
    }

    /* t=nτ (n>=1)のとき */
    else{
      /* 鳥瞰図の描画 */
      drawGraph(Nr, Np, u2);
    }

    endOpenGL();
  }
}
```

```
double phi(double r, double p)
{
  return r*sin(p)*cos(p);
}

double psi(double r, double p)
{
  return 0.0;
}

void first_step(void)
{
  /* 初期値 */

  /* u_{i,j}^0 = φ(xi,yj) */
  for(i=0;i<=Nr;i++){
    for(j=0;j<=Np;j++){
      u1[i][j] = phi(i*hr, j*hp);
    }
  }

  /* u_{i,j}^1 = (1-λ r^2-λ θ^2/ri^2)... */
  for(i=1;i<Nr;i++){
    for(j=1;j<Np;j++){
      u2[i][j] = (1.0 - lambdar2 - (lambdap2 / (i*hr*i*hr))) * u1[i][j]
        + 0.5 * lambdar2 * (u1[i-1][j] + u1[i+1][j])
        + ((tau*tau) / (2.0*i*hr*hr)) * (u1[i+1][j] - u1[i-1][j])
        + (lambdap2 / (i*hr*i*hr)) * (u1[i][j-1] + u1[i][j+1])
        + tau * psi(i*hr,j*hp);
    }
  }
}

void next_step(void)
{
```

```
/* 時間に関するループ */
if(n <= nmax){
  for(i=1;i<Nr;i++){
    for(j=1;j<Np;j++){
      u3[i][j] = 2.0 * (1.0 - lambdar2 - (lambdap2 / (i*hr*i*hr))) * u2[i][j]
        + lambdar2 * (u2[i-1][j] + u2[i+1][j])
        + ((tau*tau) / (2.0*i*hr*hr)) * (u2[i+1][j] - u2[i-1][j])
        + (lambdap2 / (i*hr*i*hr)) * (u2[i][j-1]+u2[i][j+1])
        - u1[i][j];
    }
  }

  /* Dirichlet 境界条件 */
  if(bc == DBC){
    u3[Nr][j] = 0.0;
  }

  /* Neumann 境界条件 */
  if(bc == NBC){
    for(j=0;j<=Np;j++){
      u3[Nr][j] = u3[Nr-1][j];
    }
  }

  /* θ=2πとθ=0は重なるので、コピーする */
  for(i=0;i<=Nr;i++){
    u3[i][Np] = u3[i][0];
  }

  /* u1 <- u2, u2 <- u3 */
  for(i=0;i<=Nr;i++){
    for(j=0;j<=Np;j++){
      u1[i][j] = u2[i][j];
      u2[i][j] = u3[i][j];
    }
  }
```

```
  }
}

void idle(void)
{
  n++;
  next_step();
  glutPostRedisplay();
}

int main(int argc, char **argv)
{
  printf("Nr, Ntheta: ");  scanf("%d %d", &Nr, &Np);

  hr = R / Nr;
  hp = 2.0 * PI / Np;

  printf("τ: ");  scanf("%lf", &tau);

  lambdar = tau / hr;
  lambdap = tau / hp;
  lambdar2 = lambdar * lambdar;
  lambdap2 = lambdap * lambdap;

  printf("Tmax: ");  scanf("%lf", &Tmax);
  printf("Δ t(>=%g): ", tau);  scanf("%lf", &dt);
  if(dt < tau){
    dt = tau;
  }
  skip = rint(dt / tau);
  n = 0;
  nmax = rint(Tmax / tau);

  /* u1,u2,u3 のメモリーを割り当てる */
  u1 = malloc(sizeof(double *) * (Nr+1));
  u2 = malloc(sizeof(double *) * (Nr+1));
```

```
  u3 = malloc(sizeof(double *) * (Nr+1));
  for(i=0;i<=Nr;i++){
    u1[i] = malloc(sizeof(double) * (Np+1));
    u2[i] = malloc(sizeof(double) * (Np+1));
    u3[i] = malloc(sizeof(double) * (Np+1));
  }

  setView(distance, twist, elevation, azimuth);
  setFrame(R, zmin, zmax);
  first_step();
  OpenGL(&argc, argv);
  return(0);
}
```

# 付 録A　ccgl

OpenGL と GLUT のライブラリィとコンパイル&リンクするため、次のようなスクリプトを利用した。

Cygwin 用 ccgl

```
#!/bin/sh
name=‘basename $1 .c‘
gcc -finput-charset=cp932 -fexec-charset=cp932 \
  -Wl,--enable-auto-import \
  -I/usr/X11R6/include -o ${name} "$@" \
  -L/usr/X11R6/lib -lglut -lGL -lGLU -lX11
```

# 付 録B　描画した画像をJPEGファイルに保存する場合

## B.1　コンパイルの仕方

描画した画像を、Independent JPEG Group[1] が作成・公開しているライブラリィを用いて、JPEG ファイルに保存することができる。数値計算をするプログラムとともに、set-opengl-jpeg.c、draw-graph.c（または draw-graph-on-disk.c）、ijg-saveimage.c をコンパイルする。コンパイルの仕方は、

```
ccgl (数値計算をするプログラムの名前).c set-opengl-jpeg.c
draw-graph.c ijg-saveimage.c -ljpeg
```

である。

## B.2　set-opengl-jpeg.c

```
/* set-opengl-jpeg.c */

#include <stdio.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#define KEY_ESC 27
#define KEY_SPC 32
#define DBC 0
#define NBC 1
```

[1] http://www.ijg.org/

```
/* 再生/停止の切り替え */
static unsigned char moveFlag = GL_FALSE;
/* 境界条件の切り替え */
static int bc;

static int xBegin, yBegin;
static int mButton;
static double distance, twist, elevation, azimuth;
static double D = 10.0, T = 0.0, E = 60.0, A = 120.0;

void polarview(void);
void resetview(void);
void idle(void);
void display(void);

/* 画像を保存するフォルダを作る */
int id = 0;
char fname[100];
#define DIRNAME "wave2-images"

void setView(double d0, double t0, double e0, double a0)
{
  D = d0;  T = t0;  E = e0;  A = a0;
}

/* display 関数の中で最初に書くべき命令を集めた関数 */
void beginOpenGL(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_DEPTH_TEST);

  glPushMatrix();
  polarview();
}

/* display 関数の中で最後に書くべき命令を集めた関数 */
```

```
void endOpenGL(void)
{
  glPopMatrix();
  glDisable(GL_DEPTH_TEST);
  glutSwapBuffers();

  /* 最大1万ステップまでの図を10ステップおきに記録(最大1000枚記録) */
  if(id < 10000){
    if(id % 10 == 0){                /* 10ステップおき */
      snap_ijg_image();
      sprintf(fname, "%s/image%03d.jpg", DIRNAME, id/10);
      save_ijg_image(fname);
    }
    id++;
  }
}

#define MAXCOMMANDS (100)
typedef void vvfunc(void);
static char command_keys[MAXCOMMANDS];
vvfunc *command_funcs[MAXCOMMANDS];
static num_of_reg_commands = 0;

/* c という文字で f という関数を登録する(重複チェックをサボっている) */
void register_command(char c, vvfunc f)
{
  if (num_of_reg_commands < MAXCOMMANDS) {
    command_keys[num_of_reg_commands] = c;
    command_funcs[num_of_reg_commands++] = f;
  }
  else {
    fprintf(stderr, "登録したコマンドが多すぎます。\n");
  }
}

void myKbd(unsigned char key, int x, int y)
```

```
{
  int i;

  switch(key){
  case 'r':
    resetview();
    break;
  case 'w':
    switchWireFlag();
    break;
  case KEY_SPC:
    moveFlag = !moveFlag;
    if(moveFlag == GL_TRUE)
      glutIdleFunc(idle);
    else
      glutIdleFunc(NULL);
    break;
  case KEY_ESC:
    exit(0);
  default:
    for(i=0;i<num_of_reg_commands;i++){
      if(key == command_keys[i]){
        command_funcs[i]();
        return;
      }
    }
  }
  glutPostRedisplay();
}

void myMouse(int button, int state, int x, int y)
{
  if(state == GLUT_DOWN){
    xBegin = x;
    yBegin = y;
    mButton = button;
```

```
  }
}

void myMotion(int x, int y)
{
  int xDisp, yDisp;

  xDisp = x - xBegin;
  yDisp = y - yBegin;

  switch(mButton){
  case GLUT_LEFT_BUTTON:
    azimuth -= (double)xDisp / 2.0;
    elevation -= (double)yDisp / 2.0;
    break;
  case GLUT_MIDDLE_BUTTON:
    twist = fmod(twist + xDisp/3.0, 360.0);
    break;
  case GLUT_RIGHT_BUTTON:
    distance += (double)yDisp / 60.0;
    break;
  }
  xBegin = x;
  yBegin = y;
  glutPostRedisplay();
}

void myInit(char *progname)
{
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(400, 400);

  prepare_ijg_buffer(400, 400);     /* 取得画像エリアの確保 */

  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutCreateWindow(progname);
```

```
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glutKeyboardFunc(myKbd);
  glutMouseFunc(myMouse);
  glutMotionFunc(myMotion);
  resetview();
}

void myReshape(int width, int height)
{
  double aspect = width/(double)height;
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, aspect, 1.0, 80.0);
  glMatrixMode(GL_MODELVIEW);
}

void polarview(void)
{
  glTranslatef(0.0, 0.0, -distance);
  glRotatef(-twist, 0.0, 1.0, 0.0);
  glRotatef(-elevation, 1.0, 0.0, 0.0);
  glRotatef(-azimuth, 0.0, 0.0, 1.0);
}

void resetview(void)
{
  distance = D;
  twist = T;
  elevation = E;
  azimuth = A;
}

void OpenGL(int *argc, char **argv)
{
  /* DIRNAME というフォルダを作る */
```

```
  char cmd[1024];
  sprintf(cmd, "mkdir %s", DIRNAME);
  system(cmd);

  glutInit(argc, argv);
  myInit(argv[0]);
  glutReshapeFunc(myReshape);
  glutIdleFunc(NULL);
  glutDisplayFunc(display);
  glutMainLoop();
}
```

## B.3 ijg-saveimage.c

```
/*
 * ijg-saveimage.c --- Independent JPEG Group のライブラリィを用いて
 *                     OpenGL で描いた画像を保存する (2008/8/16, by mk)
 *                     正方形でない画像が保存できないバグを取る (2008/8/30)
 *                     メモリー・リークしないよう img を固定長配列にす
る (同上)
 *                     ijg_buffer も重複して確保しないようにした (2008/11/2)
 *
 *  使い方:
 *    prepare_ijg_buffer(幅, 高さ);
 *    snape_ijg_image();
 *    save_ijg_image(ファイル名);
 *
 *  参考: http://www.syuhitu.org/other/jpeg/jpeg.html
 */

#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <jpeglib.h>
```

```
#define MAX_HEIGHT (2048)

static int ijg_width = 0, ijg_height = 0;
static JSAMPLE *ijg_buffer = NULL;

void prepare_ijg_buffer(int w, int h)
{
  /* 前回と同じサイズならば何もしない */
  if (w == ijg_width && h == ijg_height)
    return;
  /* 大きすぎるものは拒否 */
  if (h > MAX_HEIGHT) {
    fprintf(stderr, "h=%d is too large (must be <= %d)\n", h, MAX_HEIGHT);
    return;
  }
  ijg_width = w; ijg_height = h;
  if (ijg_buffer != NULL)
    free(ijg_buffer);
  ijg_buffer = malloc(w * h * 3);
}

void snap_ijg_image()
{
  /* フロント・バッファーを読み込むように設定する */
  glReadBuffer(GL_FRONT);
  /* ピクセルの内容を buffer に保存 */
  glReadPixels(0, 0, ijg_width, ijg_height, GL_RGB, GL_UNSIGNED_BYTE, ijg_buffer);
}

void save_ijg_image(char *fname)
{
  struct jpeg_compress_struct cinfo;
  struct jpeg_error_mgr jerr;
  FILE *outfile;
  JSAMPROW img[2048]; // JSAMPARRAY *img; だった。
```

```
  int i,j;

  // JPEG オブジェクトの初期化
  cinfo.err = jpeg_std_error(&jerr);
  jpeg_create_compress(&cinfo);

  // ファイルを開く
  outfile = fopen(fname, "wb");
  jpeg_stdio_dest(&cinfo, outfile);

  // パラメータの設定
  cinfo.image_width = ijg_width;
  cinfo.image_height = ijg_height;
  cinfo.input_components = 3;
  cinfo.in_color_space = JCS_RGB;

  // デフォルト値の設定
  jpeg_set_defaults(&cinfo);

  jpeg_set_quality(&cinfo, 100, TRUE);
  // 圧縮の開始
  jpeg_start_compress(&cinfo, TRUE);
  // 全イメージデータを出力
  for (i = 0; i < ijg_height; i++) {
    img[i] = ijg_buffer + (ijg_height - i) * 3 * ijg_width;
  }
  jpeg_write_scanlines(&cinfo, img, ijg_height);
  // 圧縮の終了
  jpeg_finish_compress(&cinfo);
  // JPEG オブジェクトの破棄
  jpeg_destroy_compress(&cinfo);
  // ファイルを閉じる
  fclose(outfile);
}
```

# 参考文献


[1] 林　武文・加藤　清敬, OpenGL による 3 次元 CG プログラミング, コロナ社 (2003).

[2] 桂田　祐史, 波動方程式に対する差分法, http://www.math.meiji.ac.jp/~mk/labo/text/wave.pdf (2008).

[3] 桂田　祐史, 発展系の数値解析, http://www.math.meiji.ac.jp/~mk/labo/text/heat-fdm-0.pdf (2008).

[4] 三井　康之, Java による波動方程式の数値解析, 2001 年度桂田研卒業研究レポート, http://www.math.meiji.ac.jp/~mk/labo/report/pdf/2001-mitsui.pdf (2002).

[5] 中西　謙太, 2 次元円盤領域における波動方程式の研究, 2004 年度桂田研卒業研究レポート, http://www.math.meiji.ac.jp/~mk/labo/report/open/2004-nakanishi.pdf (2005).

[6] James D. Foley, Andres van Dam, Steven K. Feiner, John F. Hughes 共著, 佐藤義雄監訳, コンピュータグラフィックス理論と実践, オーム社 (2001).