イベント駆動型プログラムと座標系

前回は GLUT を使った簡単な 2 次元グラフィックスについて習った。今週はイベント駆動型プログラムの動作とコンピュータグラフィックスの座標系について学ぶ。

イベント駆動型プログラム

これまでに学習してきたプログラムは上から下に順次実行され、条件分岐や繰り返し処理によって、プログラムの流れ（flow:フロー）に従って実行される。これを特にフロー駆動型プログラムと呼ぶ。それに対して、イベント（event:出来事）の発生に対して、実行される内容が変化するプログラムのことをイベント駆動型プログラムと呼ぶ。例えば、コントローラーからの入力に反応してキャラクタが行動するゲームプログラムはまさにこれに当たる。この場合、プログラムはイベントを監視し、発生したイベントに応じた動作を記述することになる。このイベントの監視と対応する動作をあわせて、イベントハンドラと呼ぶ。glut を使ったプログラムはこのイベント駆動型のプログラムである。作成したウィンドウの縮小・拡大や、マウスやキーボードからの入力といったイベントに対応してプログラムが実行される。

イベントはイベントキュー（event queue）と呼ばれるデータ構造に格納される。キューに格納されたイベントは格納された順に取り出され（先入れ先出し[FIFO：First In, First Out]）、それに対応する処理を行う。イベントに対応する処理はコールバック関数（callback function）を使って記述する。プログラムは実行時にイベントループと呼ばれる繰り返し処理が行われ、これによってイベントの監視とそれに対応する処理を行い続ける。前回のテキストのプログラム例 1 では display()がコールバック関数にあたり、glutMainLoop()がイベントループの実行にあたる。

座標系について

コンピュータを使って図を描く場合、以下に示す座標系を区別する必要がある。

ワールド座標系（world coordinates）またはオブジェクト座標系（object coordinates）

空間上に物体や図形を配置するために用いられる座標系。いわゆる数学で学んでいる座標だと思ってよい。単位（[m]や[mm]）についてはプログラマが自由に決めて、対応するようにプログラムを記述することになる。

ディスプレイ座標系（window coordinates）または画面座標系（screen coordinates）

ディスプレイ上の座標系。実際に表示するのに使う。この座標系の単位はピクセル（pixel）となる。ピクセルは画素とも呼ばれる。

OpenGL では表示処理の一部として、ワールド座標系からディスプレイ座標系への変換を行っている。この変換に必要な情報は「表示するウィンドウのサイズ」と「ワールド座標系でどの範囲を表示したいのか」、「ウィンドウのどの位置に表示したいのか」である。

「表示するウィンドウのサイズ」は glutInitWindowSize()で指定する。プロトタイプ宣言は以下である。

```
void glutInitWindowSize(int width, int height);
```

この関数は glCreateWindow()で作成するウィンドウの初期サイズを指定する。width は幅で、height は高さであり、単位はピクセルで指定する（図１参照）。

「ワールド座標系でどの範囲を表示したいのか」は gluOrtho2D()で指定する。プロトタイプ宣言は以下である。

```
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
```

GLdouble は OpenGL 用の変数の型であり、通常の double と同じと考えて良い。この関数の引数 left、right、bottom、top によって、ディスプレイに描くワールド座標系の四角形領域を指定する（図１参照)。前回のプログラムではこの処理を省略していた。その場合にはウィンドウの座標は左下が(-1,-1)、右上が(1,1)で設定された状態となっている。

「ウィンドウのどの位置に表示したいのか」は glViewport()で指定する。プロトタイプ宣言は以下である。

```
void glViewport(GLint x, GLint y, GLint w, GLint h);
```

GLint は OpenGL 用の変数の型であり、通常の int と同じと考えて良い。この関数の引数 x と y はウィンドウに描くワールド座標系の左下の位置を指定し、w と h はサイズを示している（図１参照)。単位はピクセルである。

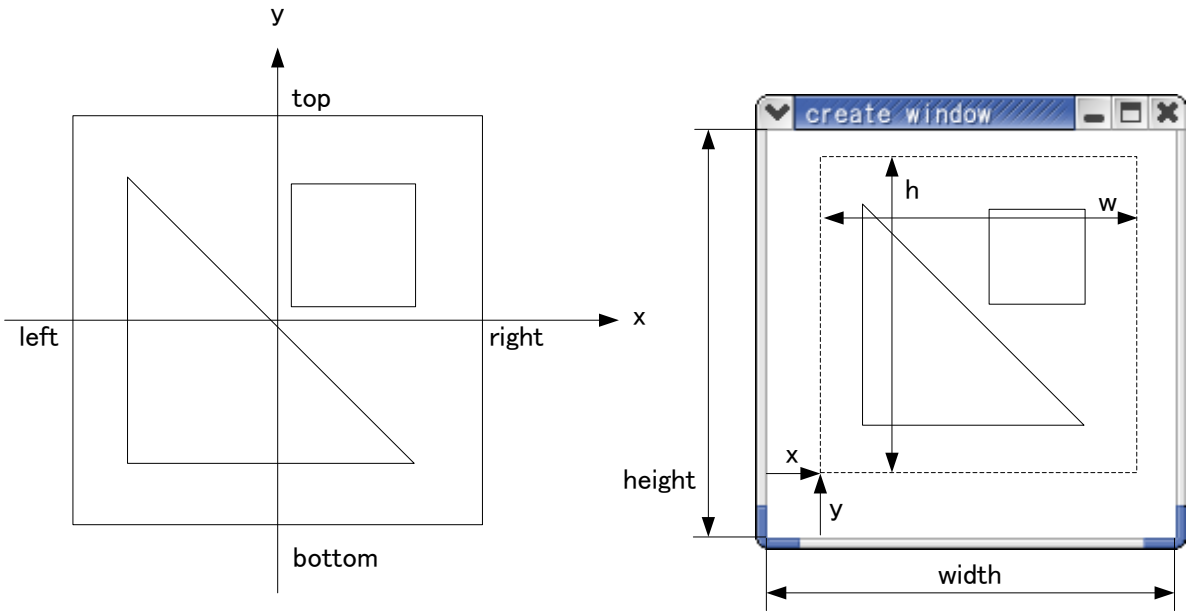

図１　ワールド座標系とディスプレイ座標系の関係

座標系を指定したプログラムの例

前回までのプログラムではウィンドウを変形するとそれに応じて図２のように描いた図形が変形した。これはワールド座標系とウィンドウ座標系の対応がくずれたためである。

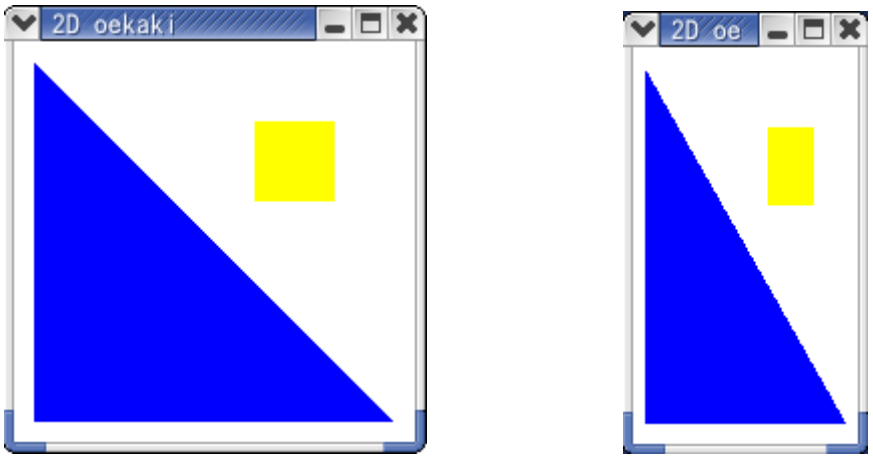

図２　描画図形の変形

そこで、ワールド座標系とウィンドウ座標系を指定し、ウィンドウサイズを変化させても形が変わらないプログラム例を示す。このプログラムは前回のテキストのプログラム例を変更したものである。変更部分は太字で示した。

プログラム例１

```
#include <stdio.h>
#include <GL/glut.h>

void init_opengl(void);                     // OpenGLの初期化
void display(void);                         // コールバック関数glutDisplayFunc()用
void resize(int w, int h);                  // コールバック関数glutReshapeFunc()用

int main(int argc, char *argv[])
{
        glutInitWindowPosition(100, 100); // ウィンドウの表示位置の指定
        glutInitWindowSize(200, 200);     // ウィンドウサイズの指定
        glutInit(&argc, argv);            // GLUTの初期化
        glutInitDisplayMode(GLUT_RGBA);   // 表示モードの指定
        glutCreateWindow("2D oekaki");    // ウィンドウを生成
        glutDisplayFunc(display);         // 描画イベント時のコールバック関数の設定
        glutReshapeFunc(resize); // ウィンドウサイズ変更イベント時のコールバック関数の設定

        init_opengl();                    // OpenGLに関する初期化　一度だけ呼ばれる

        glutMainLoop();                   // GLUTに関する無限ループ

        return 0;
}

void init_opengl(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);                   // ウィンドウを白で描画
}

void display(void)
```

```
{
        glClear(GL_COLOR_BUFFER_BIT);

        glColor3f(0.0, 0.0, 1.0);                   // 色をRGBで指定　この場合は青
        glBegin(GL_TRIANGLES);                      // 開始　三角形を描く
        glVertex2f(-0.9, -0.9);                     // 頂点を指定
        glVertex2f( 0.9, -0.9);
        glVertex2f(-0.9,  0.9);
        glEnd();                                    // 終了

        glColor3f(1.0, 1.0, 0.0);                   // 色をRGBで指定　この場合は黄色
        glBegin(GL_QUADS);                          // 開始　四角形を描く
        glVertex2f( 0.2,  0.2);                     // 頂点を指定
        glVertex2f( 0.2,  0.6);
        glVertex2f( 0.6,  0.6);
        glVertex2f( 0.6,  0.2);
        glEnd();                                    // 終了

        glFlush();
}

void resize(int w, int h)
{
        glLoadIdentity();                           // 変換行列を単位行列に設定

        // 描画するワールド座標系の範囲を指定
        gluOrtho2D(-w / 200.0, w / 200.0, -h / 200.0, h / 200.0);

        glViewport(0, 0, w, h);                     // ウィンドウの描画領域を指定
}
```

着目すべきはコールバック関数 resize()内でのプログラムの記述である。

glLoadIdentity()は変換行列を単位行列に設定している。説明の詳細は省略するが、resize 関数内の gluOrtho2D()と glViewport()と併せて使用することで、ウィンドウにどのように図形を描画するかを変更できる。gluOrtho2D()は描画するワールド座標系の範囲を指定している。glutInitWindowSize で設定したウィンドウサイズ幅 200、高さ 200 の情報を利用して、1 ピクセルがワールド座標の 0.01 になるように調整している。glViewport()は作成したウィンドウの全領域を描画領域とするように指定している。　図３にこのプログラムの実行結果を示す。ウィンドウを変形しても、図形の大きさが変更されないことがわかる。

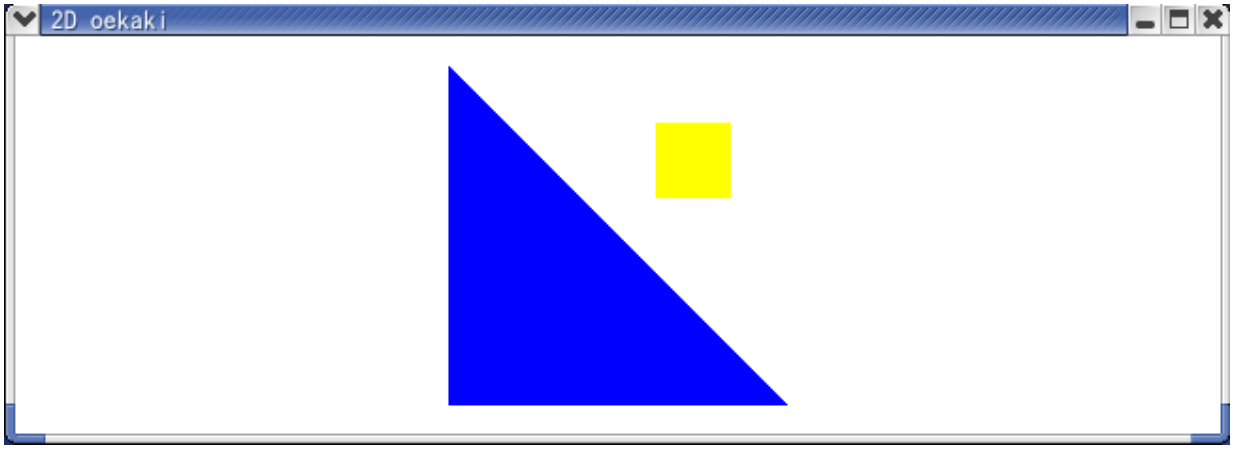

図３　座標系を指定したプログラム：ウィンドウを変形しても描画した図形のサイズが変更されていないことがわかる。

演習

演習１　プログラム例１を作成し、ウィンドウを拡大・縮小しても描画する図形が変化しないことを確認しなさい。また、glutInitWindowSize()、gluOrtho2D()、glViewport()の引数に与える値を変更し、図１に示したワールド座標系とウィンドウ座標系の関係を理解しなさい。

演習２　いろいろな図形を組み合わせて、描画する。

補足：円を描く

OpenGL で円を描く場合には多角形の近似として描く。具体的には以下のようなプログラムになる。今回のプログラム例１の display()を以下に置き換えると円が描画される。以下のプログラムにおいて、#define _USE_MATH_DEFINES は VisualStudio において、math.h の円周率 M_PI などのマクロ定義を使うために必要となる。M_PI の定義は ANSI-C 標準ではないため、この文を math.h のインクルードの前にマクロ定義する必要がある。

円を描く　glBegin(GL_POLYGON);を glBegin(GL_LINE_LOOP);に変更すると線画になる

```
#define _USE_MATH_DEFINES  // プログラムの１行目に書く
void display(void){
        int i;
        float rad;

        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 1.0, 0.0);                       // 色をRGBで指定　この場合は緑
        glBegin(GL_POLYGON);
        for(i = 0; i < 360; i++){
                rad = M_PI * (i / 180.0);
                glVertex2f(0.9 * sin(rad), 0.9 * cos(rad));     // 頂点を指定
        }
        glEnd();                                                // 終了
        glFlush();
}
```