OpenGL

- typedef unsigned int GLenum
- typedef unsigned int GLbitfield
- typedef int GLint
- typedef int GLsizei
- typedef unsigned int GLuint
- typedef float GLfloat
- typedef float GLclampf
- typedef double GLdouble
- typedef void GLvoid
- void glutInit(int *argcp, char **argv)
  - GLUTライブラリを初期化します.
  - 「argc」と「argv」はmain関数の引数，すなわちコマンドライン引数を渡します．これらの引数は，コマンドラインのオプション指定時に用いられます.
- void glutInitDisplayMode(unsigned int mode)
  - ディスプレイの表示モードを設定します.
  - 「glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE | GLUT_DEPTH)」のように書くと，「RGBAカラーモデル」で「ダブルバッファ」を使い，「デプスバッファ」も使うという指定になります.
- void glutInitWindowSize(int width, int height)
  - ウィンドウの初期サイズを設定します.
  - 「width」はウィンドウの幅，「height」はウィンドウの高さになります.
- void glutInitWindowPosition(int x, int y)
  - ウィンドウの左上の位置を指定する．引数は共にピクセル値.
- int glutCreateWindow(char *title)
  - ウィンドウを生成する．引数はそのウィンドウの名前となる.
- void glutMainLoop(void)
  - GLUTのイベントが発生するまで，待機状態になります.
- void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
  - 「glClear(GL_COLOR_BUFFER_BIT)」でウィンドウを塗りつぶす際の色を指定します.
  - 「red」「green」「blue」はそれぞれ「赤」「緑」「青色」の成分の強さを示すGLclampf型（float型と等価）の値で，0～1の間の値をもちます．1が最も明るく，この3つに(0,0,0)を指定すれば「黒色」になり，(1,1,1)を指定すれば「白色」になります.
  - 最後の「alpha」は「α値」と呼ばれ，OpenGLでは不透明度として扱われます（0で透明，1で不透明）．ここではとりあえず「1」にしておいてください.
- void glClear(GLbitfield mask)
  - 「mask」に指定したバッファのビットを初期化します.
  - 「glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)」と指定すると「カラーバッファ」と「Zバッファ」が初期化されます.
- void glutSwapBuffers(void)
  - 描画の最後で記述する．この関数が実行されると，バックバッファの内容がフロントバッファに転送される.

- void glutDisplayFunc(void (*)(void))
  - 引数は開いたウィンドウ内に描画する関数へのポインタです．ウィンドウが開かれたり，他のウィンドウによって隠されたウィンドウが再び現われたりしてウィンドウを再描画する必要があるときに，この関数が実行されます．したがって，この関数内で図形表示を行います．
- void glutTimerFunc(unsigned int millis, void (*)(int value), int value)
  - 指定された時間に呼び出されるコールバック関数を登録します．異なる時間のコールバック関数を複数用意できます．
  - 「millis」は呼び出される時間をミリ秒で指定します．少なくとも「millis」ミリ秒後にコールされるようになります．
  - 第3引数の「value」は登録したタイマーコールバック関数に渡されます．
- void glutKeyboardFunc(void (*)(unsigned char key, int x, int y))
  - 引数には，キーがタイプされたときに実行する関数のポインタを与えます．この関数の引数「key」には，タイプされたキーのASCIIコードが渡されます．また，「x」と「y」にはキーがタイプされたときのマウスの位置が渡されます．
- void glutPostRedisplay(void)
  - ウィンドウを再描画します．glutDisplayFunc()で登録したコールバック関数が呼び出されます．

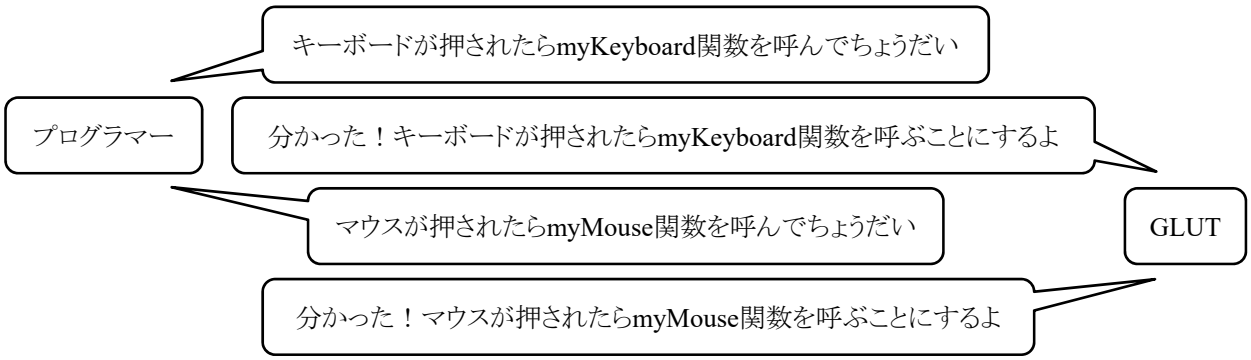
OpenGL

- void glBegin(GLenum mode)
  void glEnd(void)
  - 「glBegin」と「glEnd」の間に指定した頂点座標を使って，描画を行います．
  - 描画内容は「mode」に指定します．「mode」には「GL_TRIANGLES」などが指定できます．
- void glVertex2d(GLdouble x, GLdouble y)
  - 2次元の座標値を設定します．
  - 引数はGLdouble型の(x, y) で指定します．
- void glColor3d(GLdouble red, GLdouble green, GLdouble blue)
  - これから描画するものの色を指定します．
  - 引数の型はGLdouble型で，「red」「green」「blue」にはそれぞれ「赤」「緑」「青」の強さを「0～1」の範囲で指定します．

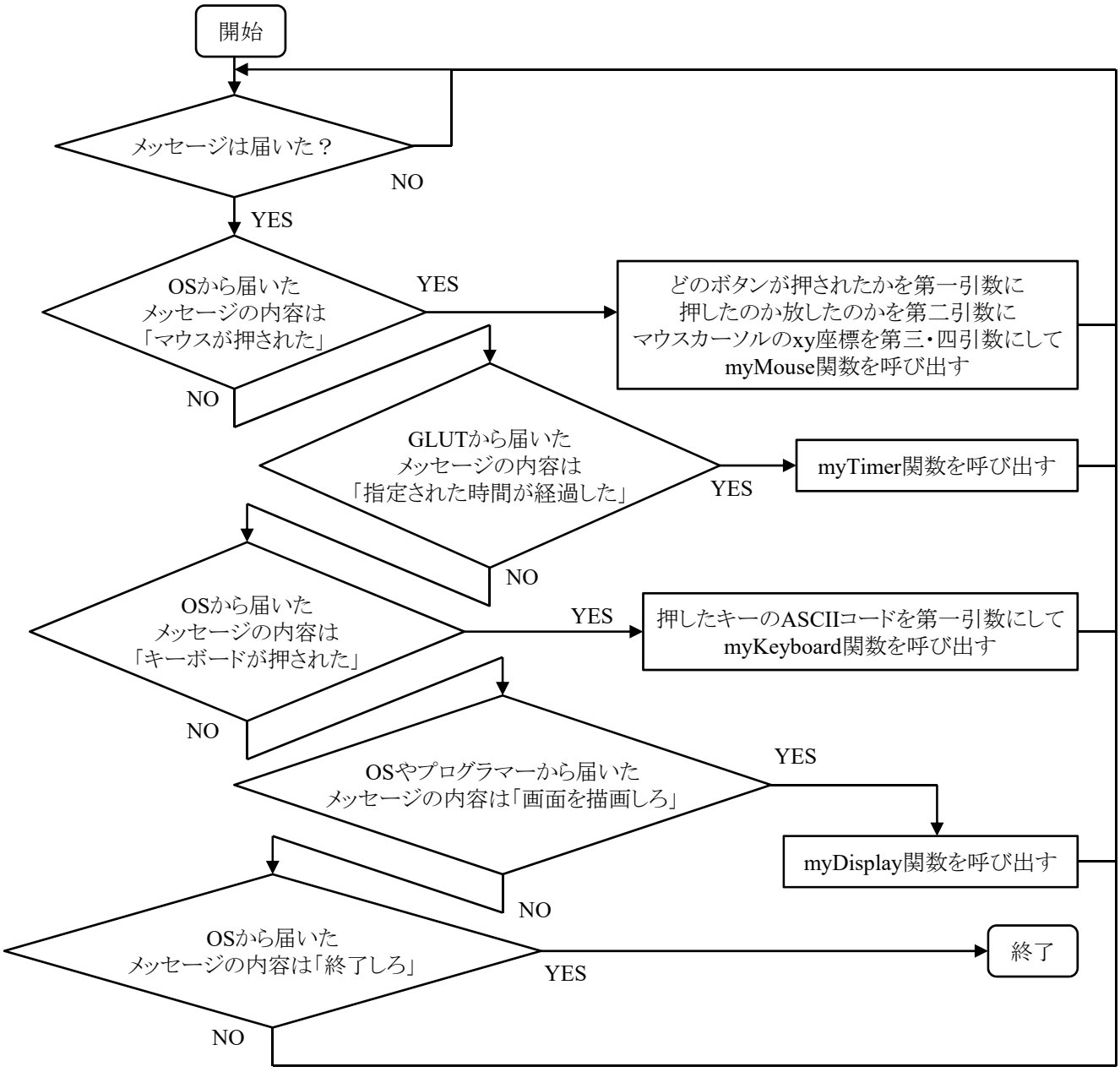
イベントドリブン型プログラミング

- glutMainLoopは無限ループしているだけで，メッセージが届くのをひたすら待ち続けます
- メッセージを受け取ったら，メッセージに応じた処理をして，また再び無限に待機します

コールバック関数の指定



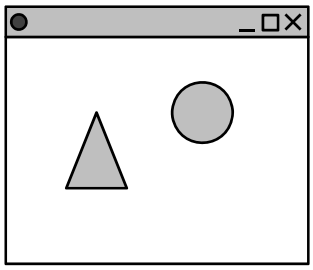
ループ中の動作の例



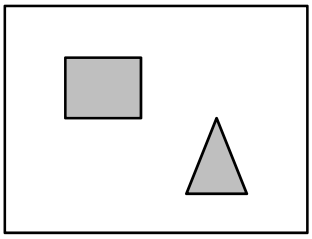
画面の描画

- glClearで画面を消去してglutSwapBuffersで描画します

描画の流れ

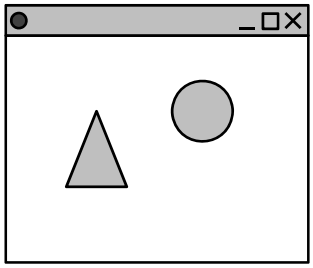


ウィンドウ表示用のバッファ



メモリ内にあるバッファ

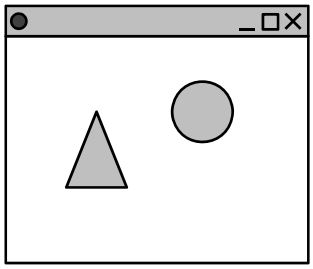
glClear(GL_COLOR_BUFFER_BIT);
画面消去



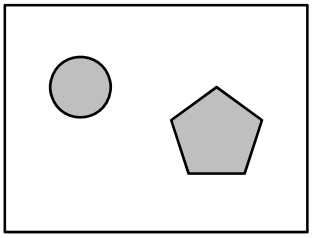
ウィンドウ表示用のバッファ

メモリ内にあるバッファ

glBegin～glEnd
図形の描画

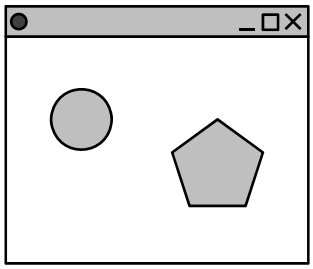


ウィンドウ表示用のバッファ

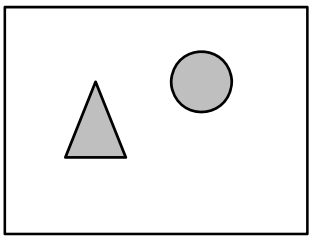


メモリ内にあるバッファ

glutSwapBuffers();
ウィンドウに表示



ウィンドウ表示用のバッファ



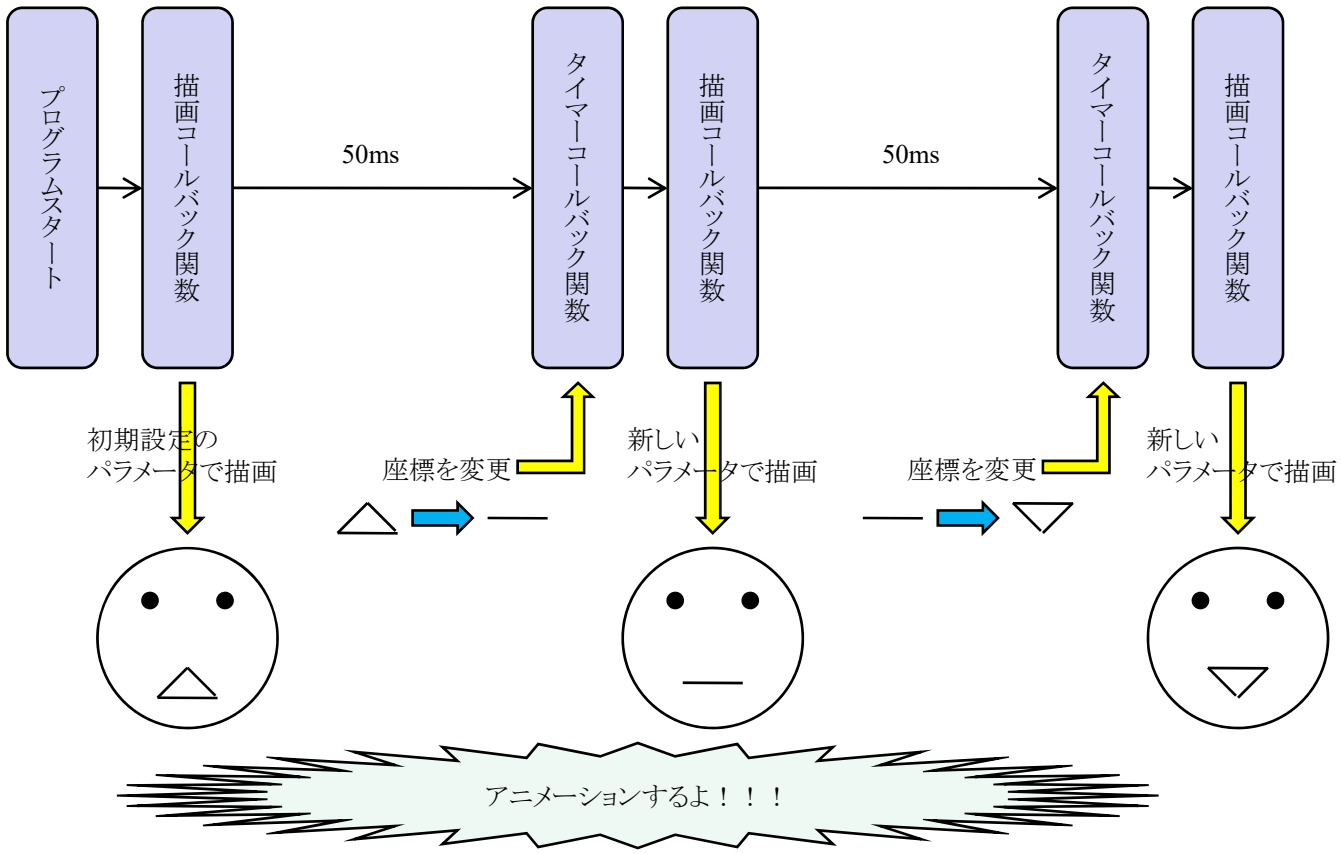
メモリ内にあるバッファ

```
void ディスプレイコールバック関数()
{
 // 画面消去
 glClear(GL_COLOR_BUFFER_BIT);

 // 図形の描画
 glBegin(GL_QUADS);
 glVertex3d(-1.0, -1.0, 0.0);
 glVertex3d(1.0, -1.0, 0.0);
 glVertex3d(1.0, 1.0, 0.0);
 glVertex3d(-1.0, 1.0, 0.0);
 glEnd();

 // ウィンドウに表示
 glutSwapBuffers();
}
```

タイマー

- void glutTimerFunc(unsigned int msecs, void (*func)(int value), int value)
  - 指定された時間に呼び出されるコールバック関数を登録します．異なる時間のコールバック関数を複数用意できます．
  - 「msecs」は呼び出される時間をミリ秒で指定します．少なくとも「msecs」ミリ秒後にコールされるようになります．
  - 第3引数の「value」は登録したタイマーコールバック関数に渡されます．
- void glutPostRedisplay(void)
  - ウィンドウ内の再描画を行います．より正確には，現在のウィンドウをマークして，「display関数」を呼び出します．

```
glutTimerFunc(50,myTimer,1); // 50ミリ秒後にmyTimer関数を呼べ
void myTimer(int value) {
    // 描画オブジェクトの座標や各種パラメータを変更する処理を行う
    glutTimerFunc(50,myTimer,1); // 再び50ミリ秒後にmyTimer関数を呼べ
    glutPostRedisplay(); // ウィンドウの内容を再描画しろ（新しい座標やパラメータで描画）
}
```



点，線，ポリゴンの描画

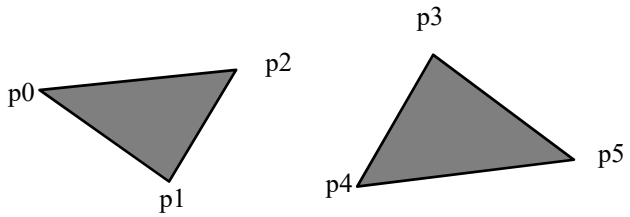
- 点，線，ポリゴンを描くのに，次のようにglBegin()とglEnd()および，glVertex*()を用いる．
- glBegin(mode);
  glVertex*(p0);
  glVertex*(p1);
  .....
  glVertex*(pn);
  glEnd();
- modeは描く図形の種類を指定し，p0，p1，…，pnは座標位置を意味する．

modeの種類

- GL_TRIANGLES　　三つ一組の頂点を，それぞれ独立した三角形として描画する．

ポリゴン描写の注意点

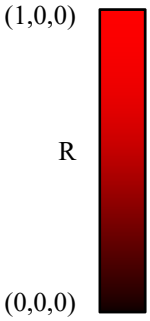
- ポリゴン（polygon）とは多角形の意味
- 頂点座標は反時計回りに設定すること
  - ポリゴンの表面と裏面を区別するため
  - 視点から見て頂点座標が反時計回りに配置されているポリゴンは表面，時計回りの場合は裏面と約束されている

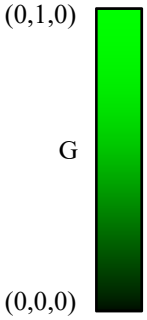


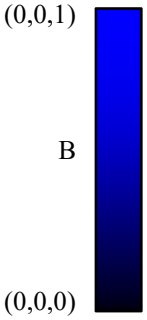
GL_TRIANGLES

色の表現

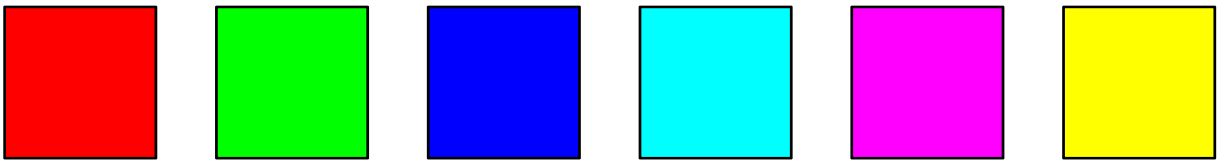
- 光の三原色で表す
- R: Red 赤, G: Green 緑, B: Blue 青
- glClearColor関数で背景色を指定，glColor3d関数で描画する図形の色を指定します
- OpenGLのこれらの関数の引数は，浮動小数点の0～1の値で表します
  - ちなみに，画面や画像ファイルなどの画素は通常，RGBそれぞれ8bitずつ，合計24bitで表します．整数で0～255の値で表します．OpenGLは0～1ですのでご注意ください．
- 0が暗くて，1が明るいです







(R,G,B)=(0,0,0)　　(R,G,B)=(0.5,0.5,0.5)　　(R,G,B)=(1,1,1)



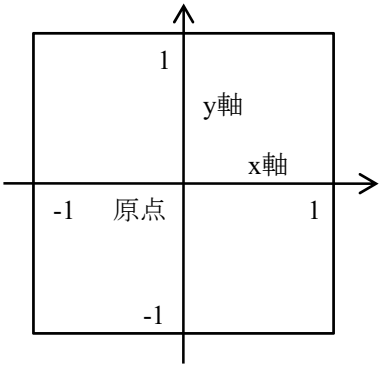
(1,0,0)　(0,1,0)　(0,0,1)　(0,1,1)　(1,0,1)　(1,1,0)

(1,0,0)　(1,0.5,0.5)　(1,1,1)　　(0,1,0)　(0.5,1,0.5)　(1,1,1)　　(0,0,1)　(0.5,0.5,1)　(1,1,1)

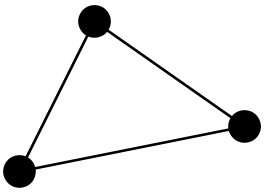
(1,0,0)　(1,1,0)　(0,1,0)　(0,1,1)　(0,0,1)　(1,0,1)　(1,0,0)

二次元図形の三角形のアニメーション

- ウィンドウ内を二次元図形の三角形がアニメーションをするプログラムを書け．三角形は刻々とその形状を変化させること．三角形は刻々とその位置を変化させること．三角形は刻々とその色を変化させること．三角形は画面の外へ出ないこと．



この正方形はウィンドウを表す
x座標とy座標は-1～1の範囲であればウィンドウ内に収まる
ウィンドウの大きさを変えようが長方形にしようが，-1～1という座標は変わらない



三角形の頂点1の(x,y)座標
三角形の頂点2の(x,y)座標
三角形の頂点3の(x,y)座標

この6個分（2×3個分）の配列を
一つの三角形ごとに確保する

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} \leftarrow \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$ 式(1)

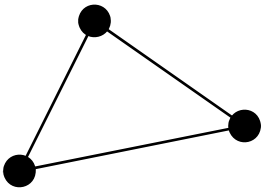
$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} \leftarrow \begin{pmatrix} v_x \\ v_y \end{pmatrix} + \begin{pmatrix} a_x \\ a_y \end{pmatrix}$$ 式(2)

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} \leftarrow \begin{pmatrix} a_x \\ a_y \end{pmatrix} + \begin{pmatrix} m_x \\ m_y \end{pmatrix}$$ 式(3)

を一定時間ごとに計算する．ただし，(p_x,p_y)は位置，(v_x,v_y)は速度，(a_x,a_y)は加速度

また，(m_x,m_y)は-M～Mの一様乱数（例えばMとして0.0001を使う）

乱数に関するヒント：　ヒント1：rand()は0～RAND_MAXの整数値をとる
ヒント2：rand()をRAND_MAXで割れば0～1の実数値になる
ヒント3：rand()もRAND_MAXも整数型の値なので，浮動小数点型に変換すること



三角形の(R,G,B)の色

この3個分の配列を
一つの三角形ごとに確保する

$$\begin{pmatrix} p_R \\ p_G \\ p_B \end{pmatrix} \leftarrow \begin{pmatrix} p_R \\ p_G \\ p_B \end{pmatrix} + \begin{pmatrix} v_R \\ v_G \\ v_B \end{pmatrix}$$ 式(4)

$$\begin{pmatrix} v_R \\ v_G \\ v_B \end{pmatrix} \leftarrow \begin{pmatrix} v_R \\ v_G \\ v_B \end{pmatrix} + \begin{pmatrix} n_R \\ n_G \\ n_B \end{pmatrix}$$ 式(5)

を一定時間ごとに計算する．ただし，(p_R,p_G,p_B)は色，(v_R,v_G,v_B)は色の変化速度

また，(n_R,n_G,n_B)は-N～Nの一様乱数（例えばNとして0.001を使う）

プロトタイプ

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

const int delay = 50;
const int OBJNUM = 10;
double objp[OBJNUM][3][2];
double objv[OBJNUM][3][2];
double obja[OBJNUM][3][2];
double colp[OBJNUM][3];
double colv[OBJNUM][3];

void myDisplay()
{
    int i;

    glClearColor(1.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    for (i = 0; i < OBJNUM; i++) {
        glBegin(GL_TRIANGLES);
        glColor3d(colp[i][0], colp[i][1], colp[i][2]);
        glVertex2d(objp[i][0][0], objp[i][0][1]);
        glVertex2d(objp[i][1][0], objp[i][1][1]);
        glVertex2d(objp[i][2][0], objp[i][2][1]);
        glEnd();
    }
    glutSwapBuffers();
}

void myTimer(int value)
{
    int i, j, k;

    if (value == 1) {
        glutTimerFunc(delay, myTimer, 1);
        for (i = 0; i < OBJNUM; i++) {
            for (j = 0; j < 3; j++) {
                for (k = 0; k < 2; k++) {
                    // [課題] objp[i][j][k]とobjv[i][j][k]と
                    // obja[i][j][k]に対する処理として適切な
                    // ソースコードを書け
                }
                colp[i][j] += colv[i][j];
                colv[i][j] += 0.002 * (double)rand() / (double)RAND_MAX - 0.001;
                if (colp[i][j] < 0.0) {
                    colp[i][j] = 0.0;
                    colv[i][j] = -colv[i][j];
                }
                if (colp[i][j] > 1.0) {
                    colp[i][j] = 1.0;
                    colv[i][j] = -colv[i][j];
                }
                if (colv[i][j] < -0.02) colv[i][j] = -0.02;
                if (colv[i][j] > 0.02) colv[i][j] = 0.02;
            }
        }
        glutPostRedisplay();
    }
}

void myKeyboard(unsigned char key, int x, int y)
{
    if (key == 0x1B) exit(0);
}
```

プロトタイプ

```
int main(int argc, char* argv[])
{
    int i, j, k;
    for (i = 0; i < OBJNUM; i++) {
        for (j = 0; j < 3; j++) {
            for (k = 0; k < 2; k++) {
                objp[i][j][k] = 2.0 * (double)rand() / (double)RAND_MAX - 1.0;
                objv[i][j][k] = 0.0;
                obja[i][j][k] = 0.0;
            }
            colp[i][j] = 1.0 * (double)rand() / (double)RAND_MAX - 0.0;
            colv[i][j] = 0.0;
        }
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutInitWindowSize(600, 400);
    glutInitWindowPosition(0, 0);
    glutCreateWindow(argv[0]);
    glutKeyboardFunc(myKeyboard);
    glutTimerFunc(delay, myTimer, 1);
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```