アニメーション

前回までに入力装置を利用したプログラムについて学んだ。本日は以下の項目について学ぶ。

・コンピュータグラフィックスを用いたアニメーションの作成方法
・簡単なアニメーションの作成

アニメーションの原理

画像が少しずつ変化しながら連続的に速い速度で再生されると、人間の目には動いているように見える（図１参照)。これを残像効果という。この画像の再生回数のことをフレームレートといい、単位は１秒間に再生されるフレーム数という意味で fps（Frames Par Second）が用いられる。人間は動画が滑らかであると感じるには最低でも 8～10fps が必要であるとされており、例えば映画やテレビの画面更新間隔は十分に滑らかに見えるよう、それぞれ 24fps や 30fps で作成されている。

コンピュータでアニメーションを作成する場合には、フレーム毎の座標をデータとして指定することは非常に手間がかかり、データ量も増える。そのため、指定する座標を計算して求めることが多い。

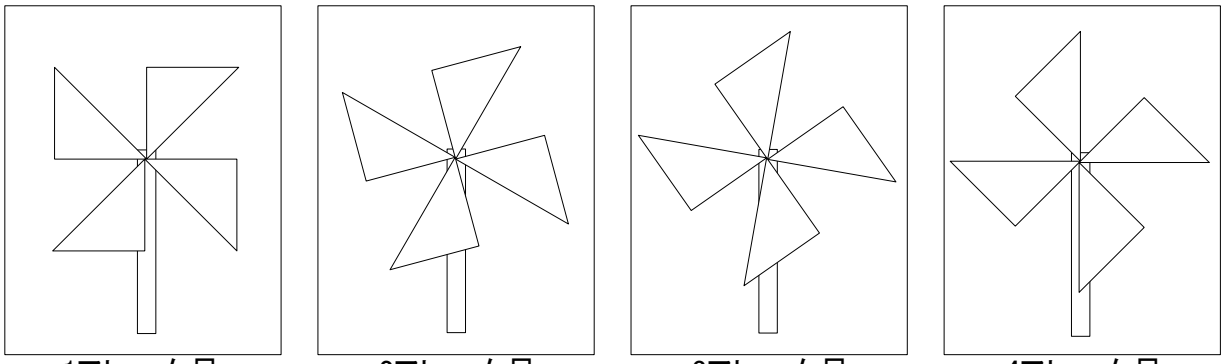

図１　アニメーションの原理：風車の羽を少しずつ回転させながら描画するとアニメーションになる。

アニメーションの例

それでは、実際にアニメーションのプログラムを見ていこう。新しく学ぶ部分については太字で示してある。

プログラム例１

```
#define _USE_MATH_DEFINES          // math.hの円周率M_PIなどのマクロ定義を使うために必要

#include <stdio.h>
#include <math.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
void animation(int value);         // コールバック関数 glutTimerFunc()用

double theta = 0.0;                // 羽の角度, 初期値
```

```
double degree = M_PI / 180;        // 度をラジアンに変換するのに使う

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| GLUT_DOUBLE);         // 表示モードの指定
        glutCreateWindow("2D oekaki");                       // ウィンドウを生成
        glutDisplayFunc(display);                            // 描画イベント処理
        glutReshapeFunc(resize);                        // ウィンドウサイズ変更時のイベント処理
        glutTimerFunc(1, animation, 0);                      // 1ms経過ごとに実行される

        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);          // ウィンドウをクリアする色を白に設定
}

void display(void)
{
        int i;
        double r1 = 0.7, r2;

        r2 = r1 / M_SQRT2;                                   // M_SQRT2は√2

        glClear(GL_COLOR_BUFFER_BIT);

        // 棒
        glColor3f(0.0, 1.0, 0.0);                            // 色をRGBで指定　この場合は緑
        glBegin(GL_QUADS);                                   // 開始　四角形を描く
        glVertex2f( 0.05, -0.9);                             // 頂点を指定
        glVertex2f(-0.05, -0.9);
        glVertex2f(-0.05,  0.0);
        glVertex2f( 0.05,  0.0);
        glEnd();

        // 羽
        glColor3f(1.0, 0.0, 0.0);                            // 色をRGBで指定　この場合は赤

        glBegin(GL_TRIANGLES);
        for(i = 0; i < 4; i++)
        {
                glVertex2f(0.0, 0.0);
                glVertex2f(r1 * cos(theta * degree + i * M_PI / 2),
                           r1 * sin(theta * degree + i * M_PI / 2));
                glVertex2f(r2 * cos(theta * degree + M_PI / 4 + i * M_PI / 2),
                           r2 * sin(theta * degree + M_PI / 4 + i * M_PI / 2));
```

```
        }
        glEnd();

        glutSwapBuffers();                              // 描画
}

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
}

void animation(int value)
{
        theta += 0.5;
        if(theta >= 360) theta -= 360;                  // 360度を超えた場合の処理
        glutPostRedisplay();
        glutTimerFunc(1, animation, 0);                 // 1ms経過ごとに実行される
}
```

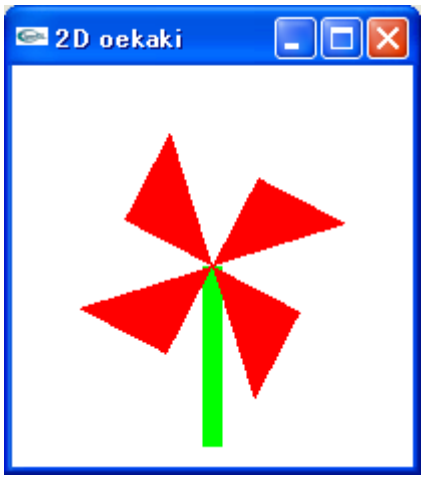

図２　実行結果：風車の羽が回る

このプログラムでは風車の羽が回るアニメーションが表示される。アニメーションに関して必要な処理は以下である。

```
void glutTimerFunc(unsigned int ms, void (*func)(int value), value);
```

一定時間が経過するとコールバック関数を呼び出す。この時間の設定は ms で行い、ミリ秒単位で指定できる。コールバック関数は func で設定する。value はコールバック関数に渡される値である。タイマーは複数同時に使うことができ、value はコールバック関数でタイマーを区別するために利用する。

```
void glutPostRedisplay(void);
```

ウィンドウの再描画を指示する。イベントキューに glutDisplayFunc()で登録したコールバック関数 display()の実行が追加される。

void glutSwapBuffers(void);

ダブルバッファを利用した描画において、裏バッファと表バッファを交換する。ダブルバッファを有効にするには、表示モードを「glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);」などに変更しなくてはならない。

glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)についての補足

GLUT_RGBA は RGBA（赤、緑、青、不透明度）を利用した表示を行う指定であり、GLUT_DOUBLE はダブルバッファを有効にする。「|」はビット毎の論理和である。ダブルバッファはアニメーションのちらつきを防ぐために用いる。描画内容はバッファというメモリに貯えられる。前回までのプログラムではウィンドウに表示を行うバッファに直接書き込んでいた。この状態でアニメーションを行うと、ウィンドウをクリアして、描画するという処理を連続で行うため、ちらつきが発生してしまう。今回の表示モードでは別のバッファメモリに描画し、描画が完了したら、表示するバッファをを切り替えて表示する。つまり、裏で新しい絵を作成して、完了したら表に出すという処理を行っている。この切り替えの処理は display()の最後に記述した「glutSwapBuffers();」を呼び出すことで実行する。

風車の羽の座標データの指定について

風車の羽は４枚の三角形で表現する。三角形の描画は glBegin(GL_TRIANGLES);で行う。この関数は glEnd();との間に記述する３つの頂点を１セットとして、三角形の描画を行う。三角形の頂点座標は図３のように指定できる。

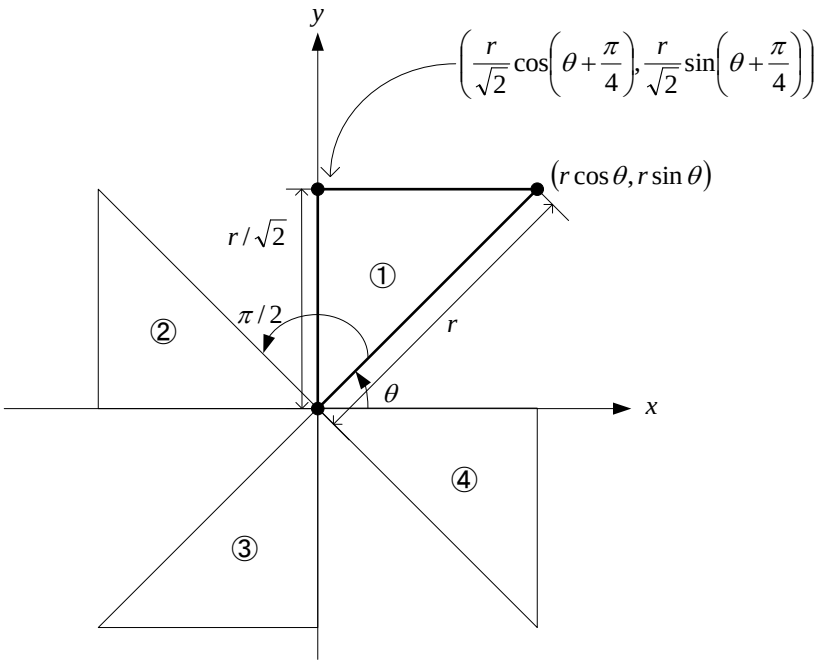

図３　風車の羽の座標データの指定について

頂点の座標は直角三角形と三角関数の知識を用いれば決定することができる。直角三角形の斜辺の長さをrとし、羽の回転角度を θ とすると、三角形の各頂点の座標は以下のようになる。三角形の頂点の順番は原点を起点として、左回りに示した。

①

$$(0,0),(r\cos\theta,r\sin\theta),\ \left(r\cos(\theta+\pi/4)/\sqrt{2},r\sin(\theta+\pi/4)/\sqrt{2}\right)$$

②（①の三角形を $\pi/2$ [rad]回転させたもの）

$$(0,0),(r\cos(\theta+\pi/2),r\sin(\theta+\pi/2)),\ \left(r\cos(\theta+\pi/4+\pi/2)/\sqrt{2},r\sin(\theta+\pi/4+\pi/2)/\sqrt{2}\right)$$

③（①の三角形を π [rad]回転させたもの）

$$(0,0),\ (r\cos(\theta+\pi),r\sin(\theta+\pi)),\ \left(r\cos(\theta+\pi/4+\pi)/\sqrt{2},r\sin(\theta+\pi/4+\pi)/\sqrt{2}\right)$$

④（①の三角形を $3\pi/2$ [rad]回転させたもの）

$$(0,0),(r\cos(\theta+3\pi/2),r\sin(\theta+3\pi/2)),\ \left(r\cos(\theta+\pi/4+3\pi/2)/\sqrt{2},r\sin(\theta+\pi/4+3\pi/2)/\sqrt{2}\right)$$

演習

演習１　プログラム例１作成し、動作を確認しなさい。

演習２　プログラム例１に以下の変更を加え、プログラムを理解しなさい。

・風車の羽の回転の変化の量を変更しなさい。

・風車の羽の数を６枚に変更しなさい。羽はバランス良く配置すること。

課題１

簡単なアニメーションを行うプログラムを作成しなさい。複数枚の画像を付けたり、文章で説明したりすることで、どういったアニメーションを作ったのか説明すること。レポートにはプログラムも付けること。