

14-1. コンピュータグラフィックスで使われる複合曲線

14-1-1. パラメトリック曲線と複合曲線

パラメトリック曲線とは、媒介変数のパラメータ（ t とすると）、 $C=F(t)$ で表される曲線のことを示します。また、複合曲線とは、適当なパラメータ区間によって1つの曲線を表し、それらの曲線を複数繋ぐことで、1本の曲線表現するようなものを指します。複合曲線の、部分区間にある、それぞれの曲線をセグメントと呼ぶことがあります。曲線の次数とは、曲線を媒介変数の多項式で表すときの、多項式の次数を示します。次数が高くなるほど、より豊かな表現ができます。このようなパラメトリック曲線としては、次の曲線がよく利用されています。

- ・エルミート（Hermite）曲線
- ・ベジエ（Bézier）曲線
- ・Bスプライン（Beta-Spline）曲線
- ・有理ベジエ（Rational Bézier）曲線
- ・NURBS（Non Uniform Rational Beta-Spline：非一様有理Bスプライン）曲線

ここでは、ベジエ曲線とBスプライン曲線を取り上げます。

★有理複合曲線

有理ベジエ曲線や、NURBS（有理Bスプライン）曲線は、それぞれベジエ曲線の制御点、あるいはBスプライン曲線の各制御点やノットに重み $\{w_i\}$ を追加したのになっています。表現の幅が広がります。NURBS曲線をベースとした、NURBS（ナーブズ）曲面は、3次元CGなどで広く用いられています。有理ベジエ曲線や、NURBS曲線の場合は、円や楕円、放物線を正確に再現できるという特性があります。

★ベジエ曲線とスプライン曲線の違い

ベジエ曲線では、美しい曲線が得られるが、複合曲線を形成するときに、セグメントである曲線と曲線の繋がりがうまくいかない場合もあります。スプライン曲線は、直線の間の補完として考えられているので、曲線同士の繋がりは良いでしょう。また、ノットの個数によって、次数が変わらないので、ノットを追加すれば微妙なカーブを表現できます。

14-1-2. ベジエ曲線

制御点（Control-Point）と呼ばれる複数の点に基づく多項式で定義され、パラメータ t を0から1まで変化させる。ベジエ曲線 $P(t)$ は、制御点の重みが連続的に変化して構成されます。ベジエ曲線の多項式は、3次までがよく用いられ、3次のベジエ曲線をバーンスタイン曲線と呼ぶことがあります。一般に、 n 次のベジエ曲線を表すには、 $n+1$ 個の制御点が必要である。3次の場合は、4個必要となります。

3次のベジエ曲線の場合の方程式と、各頂点（制御点）について以下のような式で計算します。

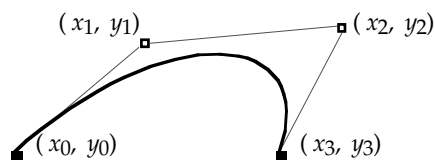


図14-2 ベジエ曲線と4つの頂点

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

この式は、バーンスタイン関数と呼ばれています。それぞれの P_i には、 x_i または y_i が入ります。 x 座標とも、 y 座標とも同じ形の式で計算できるのです。これを使って、配列に格納されている4つの座標値を用いて、ベジエ曲線を描画するようなアプレットを作ってみました。paintメソッドの部分だけを示しています。1-tをいちいち記述するのは面倒なので、ct（complement for t の意）という名前の変数に入れています。座標は、

ウィンドウの200ピクセルの位置をy座標の原点としてあります。x座標は、そのまま計算しています。以前に説明した、折れ線で近似する方式で描くことにしています。

```
public void paint (Graphics g) {
    int    x[] = { 30, 70, 130, 200 };           // x座標の配列
    int    y[] = { 30, 150, 180, 35 };           // y座標の配列

    // ガイドとなる折れ線を引いています
    for ( int i=0; i<x.length-1; i++ ) {
        g.drawLine( x[ i ], 200-y[ i ], x[ i+1 ], 200-y[ i+1 ] );
    }
    int    n, x1, y1, x2, y2;
    double t, ct, delta;
    g.setColor( Color.red );
    n = 100;                                     // 100本の直線で近似
    delta = 1.0/n;
    t = 0;
    x1 = x[ 0 ];
    y1 = 200 - y[ 0 ];
    for ( int i=1; i <= n; i++ ) {
        t = t + delta;
        ct = 1 - t;
        x2 = (int)(ct*ct*ct*x[0] + 3*t*ct*ct*x[1] + 3*t*t*ct*x[2] + t*t*t*x[3]);
        y2 = 200 - (int)( ct*ct*ct*y[0] + 3*t*ct*ct*y[1] + 3*t*t*ct*y[2] + t*t*t*y[3] );
        g.drawLine( x1, y1, x2, y2 );
        x1 = x2; y1 = y2;
    }
}
```

14-1-3. Bスプライン曲線

Bスプライン曲線にも、n次の多項式で表されます。2次か3次が用いられることが多く、TrueTypeフォントは、2次の曲線が使われています。制御点に対する曲線上の頂点のことをノット（結び目という意味）と呼びます。ノットの間隔を一定にしたものを一様Bスプライン曲線と呼びます。一定でないものは、非一様Bスプライン曲線と呼びます。

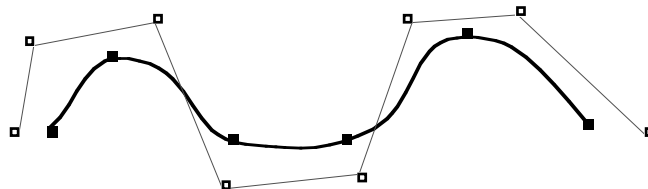


図14-3 スプライン曲線

ここでは、3次のBスプライン曲線を描いてみましょう。座標を求める関数は、以下のようになります。1本のセグメントを描くためには、3次の場合は、4つの頂点を必要とします。

$$B(t) = 1/6 * (-P_0 + 3P_1 - 3P_2 + P_3)t^3 + 1/2 * (P_0 - 2P_1 + P_2)t^2 + 1/2 * (-P_0 + P_2)t + 1/6 (P_0 + 4P_1 + P_2);$$

式が長いですね。ここでも、B(t)は、x座標を求めるのにも、y座標を求めるのにも使われます。P_iが、x座標の場合は、B(t)は、x座標が求まります。P_iがy座標の場合は、y座標が求まります。ベジェ曲線のときと同じように、これを描くpaintメソッドを記述してみましょう。同じように、配列にx座標、y座標を入れておき、100個の折れ線で近似して描くようにしています。y座標も200ピクセルの位置を原点としています。各座標を求めるのに上記の関数が使われているのを確認してください。

```

public void paint (Graphics g) {
    int    x[] = { 30, 70, 130, 200 };
    int    y[] = { 30, 150, 180, 35 };

    for ( int i=0; i<x.length-1; i++ ) {
        g.drawLine( x[ i ], 200-y[ i ], x[ i+1 ], 200-y[ i+1 ] );
    }
    int    n, x1, y1, x2, y2;
    double t, delta;
    x1 = (int)( (x[ 0 ]+4*x[1]+x[2]) / 6.0 );
    y1 = 200 -(int)( (y[ 0 ]+4*y[1]+y[2]) / 6.0 );
    for ( int i=1; i <= n; i++ ) {
        t = t + delta;
        x2 = (int)((-x[0]+3*x[1]-3*x[2]+x[3])*t*t*t/6.0 + (x[0] - 2*x[1] + x[2])*t*t/2.0
            + (-x[0]+x[2])*t/2.0 + (x[ 0 ]+4*x[1]+x[2])/6.0);
        y2 = 200 -(int)((-y[0]+3*y[1]-3*y[2]+y[3])*t*t*t/6.0 + (y[0] - 2*y[1] + y[2])*t*t/2.0
            + (-y[0]+y[2])*t/2.0 + (y[ 0 ]+4*y[1]+y[2])/6.0);
        g.drawLine( x1, y1, x2, y2 );
        x1 = x2; y1 = y2;
    }
}

```

14-2. 行列とアフィン変換

14-2-1. 行列による標記

★行列の表記と行列の積

行列は、行と列に整理して、複数の数値情報をまとめて扱うのに用いる記法です。Javaでは、2次元配列がこれに対応します。特に、積という形を用いて、ある情報に変換を行なう記述に使われます。ここでは、積を中心に、簡単に行列の計算方法に触れていきます。

たとえば、2行2列や、2行1列の行列は、次のように記述されます。

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{2行2列の行列}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \quad \text{2行1列の行列}$$

行や列のどちらかが、1になっているものは、ベクトルとも呼ばれます。ベクトルは、Javaでは1次元配列に対応します。1つの行列をAやBで表すと、掛け算に関しては、以下のような法則が成り立っています。

$$A \times B \times C = (A \times B) \times C = A \times (B \times C) \quad \text{結合則}$$

注意しなければならないのは、行列では交換則が成り立たないということです。 $A \times B \neq B \times A$ なのです。掛け算の順番には、注意しなければなりません。

行列を掛け算するとき、n列の行列は、n列の行列だけと掛け算することができまるとい規則もあります。 $A \times B$ を計算するときは、Aの行列の列の数と、Bの行列の行の数を等しくしておく必要があります。Aの行列が、m行n列であり、Bの行列がn行p列であったとすると、結果の行列は、m行p列になります。

例えば、2行2列の行列と、2行1列の行列を掛け算すると、2行1列の行列になります。以下は、掛け算記号がついていますが、通常は省略します。

$$\begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

このときは、それぞれの行列の要素は、以下のように計算されます。

$$\begin{aligned}x' &= a x + b y \\y' &= c x + d y\end{aligned}$$

A × B の場合には、行列 A の列方向の要素と、行列 B の行方向の要素が掛け合わさると覚えておくといでしょう。また、2 行 2 列の行列同士を掛け合わせると、計算結果の行列も 2 行 2 列となります。

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

このときは、以下のように行列の要素が計算されます。これも、列方向と行方向の要素が掛け合わされていると考えることができますでしょう。

$$\begin{aligned}x' &= a x + b y & w' &= a w + b z \\y' &= c x + d y & z' &= c w + d z\end{aligned}$$

2 行 2 列のような行列のことを行と列のサイズが等しいので、正方行列と呼ぶことがあります。正方行列を掛け合わせていくと、計算結果も正方行列となります。

★回転

原点周りに θ 度回転させた座標は、次のように計算することができます。

$$\begin{aligned}x' &= x \cos \theta - y \sin \theta \\y' &= x \sin \theta + y \cos \theta\end{aligned}$$

どうして、このように求められるのでしょうか？これは、元々、 $x = r \cos \alpha$ で、 $y = r \sin \alpha$ で表わされるとすると（これは、極座標形式で表わされたと思ってください）、原点周りに回転させると以下のようなになるからです。

$$\begin{aligned}x' &= r \cos(\theta + \alpha) \\&= r (\cos \theta \cos \alpha - \sin \theta \sin \alpha) && // \text{三角関数の加法定理より} \\&= x \cos \theta - y \sin \theta \\y' &= r \sin(\theta + \alpha) \\&= r (\sin \theta \cos \alpha + \cos \theta \sin \alpha) && // \text{三角関数の加法定理より} \\&= x \sin \theta + y \cos \theta\end{aligned}$$

これを行列で表示すると以下ようになります。行列の掛け算であることに注意してください。

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

さて、この変換を使って、多角形を回転させてみましょう。以下の paint メソッドは、矢印のような多角形を 10 度ずつ回転させたものです。ただし、画面上の x, y 座標を直接使っていますので、回転の方向は、時計回りになっています（通常は逆時計回りです）。座標値を持つ配列の要素の値を直接書き換えていますので、一時的な変数 (oldx と oldy) に、書き換える前の座標の値を保存しており、新しい座標値の計算には、その値を使うようにしています。

```
public void paint (Graphics g) {
    int    x[] = { 30, 90, 90, 120, 90, 90, 30 };
```

```

int    y[] = { 40, 40, 10, 55, 100, 70, 70 };
g.setColor(Color.blue);
g.drawPolygon( x, y, x.length );
double rad = Math.toRadians( 10 );
for ( int i = 1; i <= 9; i++ ) {
    for ( int j=0; j<x.length; j++ ) {
        int oldx = x[ j ], oldy = y[ j ];
        x[ j ] = (int) (oldx * Math.cos( rad ) - oldy * Math.sin( rad ));
        y[ j ] = (int) (oldx * Math.sin( rad ) + oldy * Math.cos( rad ));
    }
    g.setColor( new Color( i * 30, 0, 250-i*30 ) );
    g.drawPolygon( x, y, x.length );
}
}

```

※このpaintメソッドを、通常の座標系の形で書き直してみなさい。

14-2-2. アフィン変換

回転は、うまく2次の行列で表わせました。しかし、移動を含めると、2次の行列では、表わせなくなります。移動は、 $x' = x + dx$ 、 $y' = y + dy$ となるからです。ここでdxとdyはそれぞれの座標における移動量を示しています。回転なども含めて、一般に、変換後の座標を求める式としては、以下のようになります。

$$\begin{aligned}
 x' &= a_1 x + a_2 y + a_3 \\
 y' &= b_1 x + b_2 y + b_3
 \end{aligned}$$

たとえば、 $a_1 = \cos \theta$ 、 $a_2 = -\sin \theta$ 、 $a_3 = 0$ とおき、 $b_1 = \sin \theta$ 、 $b_2 = \cos \theta$ 、 $b_3 = 0$ とおけば、これは回転を示しているのが解ると思います。

この式で表わされる座標の変換を一般的にアフィン変換 (Affine Transform) と呼んでいます。アフィン変換を行列で表わすために、もう1つ次数を増やして、3行×3列、すなわち3次の行列として表わすことに致します。このようにすると、変換された後の新しい座標を以下のように表わすことができます。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

ここで、変換後の座標(x' , y') の下にある1は名目的に用いられています。この行列の形で表わすと、いろいろな座標変換をうまく組み合わせることができるのです。以下に、アフィン変換で用いられる座標変換の例を行列の形で表わしていきます。

★移動

移動は、元のx座標とy座標の値に、移動量(dx, dy) を足したものになりますので、以下のような形で記述されることになります。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

★回転

回転は、今まで見たとおりです。アフィン変換の行列で、3次で表わすと以下ようになります。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

★拡大・縮小

拡大・縮小は、 $x' = s_x x$, $y' = s_y y$ として計算されますので、以下のような行列で表わせます。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

★鏡像

左右対称の場合と、上下対称の場合が、あります。左右対称の場合は、 $x' = -x$, $y' = y$ となりますし、上下対称の場合は、 $x' = x$, $y' = -y$ となります。それぞれを記述すると以下のようになります。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ 左右対称} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ 上下対称}$$

★せん断（シアー）

せん断は、 $x' = x + ay$ となり、 $y' = bx + y$ となるような変形です。相手方の座標に依存した部分が入ってきます。これは、平行四辺形的な変形をするときに用いられます。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

1 4 - 2 - 3. アフィン変換の組合わせ

行列の積の形で組み合わせることによって、各変換を組み合わせることができます。ここでは、良く使う典型的な例を2つだけ説明します。

★任意の点を回転の中心とする回転

点P (x_p , y_p) を中心で回転させるとしましょう。このときに、まず、原点をその点まで移動させます。そして、原点を中心に回転させ、その後で、移動した分だけ原点を戻してやります。これを、行列の積の形で記述すると次のようになります。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_p \\ 0 & 1 & y_p \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_p \\ 0 & 1 & -y_p \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

これを展開して、通常の方程式の形で記述すると次のようになります。

$$\begin{aligned} x' &= (x - x_p) \cos \theta - (y - y_p) \sin \theta + x_p \\ y' &= (x - x_p) \sin \theta + (y - y_p) \cos \theta + y_p \end{aligned}$$

この式を用いて、先程の多角形を、その重心を中心に回転させてみましょう。まずは、重心の位置を計算します。すべての座標を足し合わせてから、平均を計算します。

```

public void paint (Graphics g) {
    int    x[] = { 30, 90, 90, 120, 90, 90, 30 };
    int    y[] = { 40, 40, 10, 55, 100, 70, 70 };
    g.setColor(Color.blue);
    g.drawPolygon( x, y, x.length );

    int    xp = 0, yp = 0;                // 重心をxp, ypに求める
    for ( int i = 0; i < x.length; i++ ) {
        xp += x[i];        yp += y[i];    // すべてを足し合わせる
    }
    xp /= x.length;        yp /= y.length; // 個数で割って平均を取る

    double rad = Math.toRadians( 10 );
    for ( int i = 1; i <= 9; i++ ) {
        for ( int j = 0; j < x.length; j++ ) {
            int oldx = x[j], oldy = y[j];
            x[j] = (int) ((oldx-xp) * Math.cos( rad ) - (oldy - yp) * Math.sin( rad )) + xp;
            y[j] = (int) ((oldx-xp) * Math.sin( rad ) + (oldy - yp) * Math.cos( rad )) + yp;
        }
        g.setColor( new Color( i * 30, 0, 250-i*30 ) );
        g.drawPolygon( x, y, x.length );
    }
}

```

★任意の直線 $ax + by + c = 0$ に関する鏡像

これは、次のように分解することができます。まず、この直線は垂直線ではないとします。その場合、 y 軸と交差しますから、交差する点の位置まで、原点を移動させます。そして、この直線の傾きの分だけ、回転させてやります。その後に、上下の鏡像を求めます。そして、先程回転させた分を戻し、原点の移動分だけ戻すという操作になります。これを行列の積の形で記述すると次のようになります。ただし、以下の行列の式において、 $r = \sqrt{a^2 + b^2}$ です。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & b/r & a/r & 0 & 0 \\ c/b & a/r & b/r & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

行列の積の一番外側の2つの行列は、交差する点までの移動とそれを戻す移動を意味しています。内側の2つの行列は、直線の傾き分だけ、座標を回転させること（およびその回転を戻すこと）を意味しています。真ん中の行列は、上下の鏡像を求めています。これを展開して、通常の方程式の形で記述すると次のようになります。

$$\begin{aligned}
 x' &= ((b^2 - a^2) x - 2aby - 2ac) / (a^2 + b^2) \\
 y' &= (-2abx + (a^2 - b^2) y - 2bc) / (a^2 + b^2)
 \end{aligned}$$

14-3. 課題

14-1

アフィン変換を使って、星形を、色を変えながら、移動させ、回転させて表示しなさい。クラス名は、StarRotateにて。

ヒント：星の形は、2つの半径を持ちます。外側の円にある各頂点と、内側にある各頂点を角度を変えながら、計算によって求めていきます。

14-2

ベジェ曲線で、リボンとなるように、4つのx座標、y座標の位置を調整してみてください。

14-3

スプライン曲線を、更に多くの点を加えて、複合曲線として描くようにしてみてください。クラス名は、CompoundSplineにて。

ヒント：配列にx,y座標を持っておきます。そして、現在のセグメントが始まりを示す変数を作ります。配列の要素を計算するときに、スプライン曲線のアプレットの記述の例では、0から3までの数が持ちいられていますが、それぞれにこの変数と足し合わせるような形にします。そして、外側にさらにfor文を追加して、このセグメントの始まりを示す変数を0から、座標の配列のサイズ-4まで変化させるようにします。このようにすると、節点（ノット）をずらしながら、曲線を描くことができます。