# コンピュータグラフィクスの基礎理論
# 第 10 回資料

**渡辺大地**

月から見ても、
すべての星は月を中心に回っているように見える。
— A. ビーン

## 10.1 行列による拡大・縮小

前回、平行移動と回転移動を行列で表現する方法を紹介しましたが、行列を使って図形の拡大や縮小も扱うことが可能です。

**拡大・縮小行列:**

任意の頂点座標 $(x, y)$ を、原点を中心に $x$ 方向に $\alpha$ 倍、$y$ 方向に $\beta$ 倍拡大 (縮小) した位置に移動させるには、次のような行列に位置ベクトル $(x, y, 1)$ を掛ければよい。

$$\begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{10.1}$$

例えば、$(2, 2)$ を原点を中心に $x$ 方向に 2 倍、$y$ 方向に 3 倍拡大した座標値は、以下のようにして求められます。

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 1 \end{pmatrix} \tag{10.2}$$

## 10.2 任意位置での図形回転

前回、図形を原点を中心に回転させる行列は次のようなものであると解説しました。

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{10.3}$$

しかし、この行列はあくまで原点を中心に回転させるだけのものです。

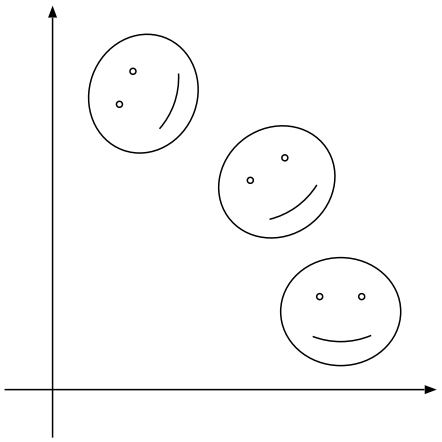

図 54: 原点を中心とした図形の回転の様子

実際に図形をコンピュータ上で扱うとき、図形の回転の中心は原点ではなく図形の真ん中にしたいことが普通でしょう。

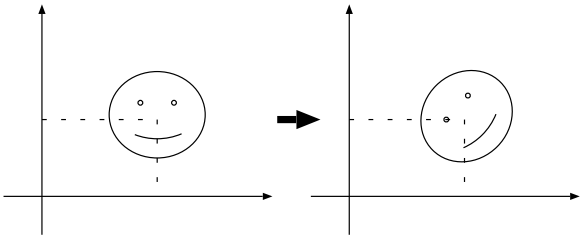

図 55: 図形の真ん中を中心とした回転

では、どうすればこれを実現することができるのでしょうか？ それは、次の 3 ステップを使うことで可能となります。

1. まず、図形の真ん中が原点に来るように、図形を平行移動させます。

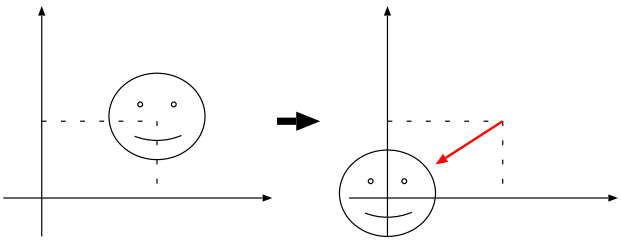

図 56: 図形の回転方法 (1)

2. 次に、平行移動させた図形を原点を中心に回転させます。

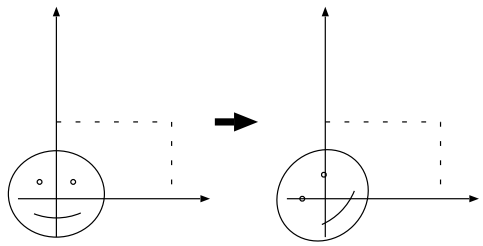

図 57: 図形の回転方法 (2)

3. 最後に、回転させた図形を元の位置に平行移動させます。

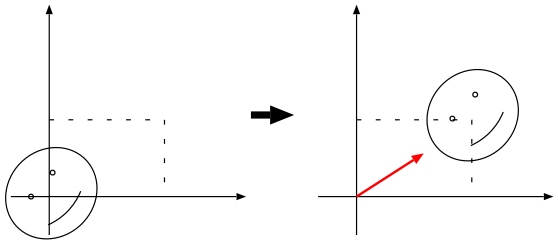

図 58: 図形の回転方法 (3)

イメージはこれで掴めると思いますので、次節では実際にこれを実現する手法を述べます。

## 10.3 行列による任意位置図形回転

ここでは、前節のステップを行列を用いて求めてみます。まず、前提として図形の情報を次のように定めます。

| | |
|---|---|
| $(p, q)$ | ..... 図形の中心位置 |
| $\theta$ | ..... 回転角度 |
| $\mathbf{P}$ | ..... 図形上の頂点 (の位置ベクトル) |

さて、まず最初のステップであった「図形を原点へ移動させる」というものですが、これは言い換えると「図形上の頂点を $(-p, -q)$ 平行移動する」ということになります。ですから、移動後の頂点位置座標値を $\mathbf{P}'$ とおくと

$$\mathbf{P}' = \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & -q \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P} \tag{10.4}$$

となります。

次に、移動した点を原点を中心に $\theta$ だけ回転させます。回転後の頂点位置座標値を $\mathbf{P}''$ とおくと、

$$\mathbf{P}'' = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}' \tag{10.5}$$

となります。

最後に、回転された図形を元の位置に戻します。これは、「図形を $(p, q)$ 平行移動する」ということになりますから、移動後の頂点位置座標値を $\mathbf{P}'''$ とおくと、

$$\mathbf{P}''' = \begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}'' \tag{10.6}$$

という計算によって、最終的な位置が決定できます。合わせて 3 回の行列とベクトルの積を利用するので中々大変ですが、一応求めることが可能となりました。

## 10.4 合成変換

前節での手順を用いて、図形の真ん中を中心とした回転することができることを示しました。しかし、そのためには $\mathbf{P}', \mathbf{P}'', \mathbf{P}'''$ という 3 個の頂点位置を順番に求めていかなければなりません。この節では、数学のトリックを使って、$\mathbf{P}'$ と $\mathbf{P}''$ を求めずに、直接 $\mathbf{P}$ から $\mathbf{P}'''$ を求める方法を記述します。それには、「**合成変換**」という手法を用います。

まず、前節で使った 3 個の行列に対してそれぞれ $\mathbf{T}_1, \mathbf{R}, \mathbf{T}_2$ という名前をつけます。つまり、

$$\mathbf{T}_1 = \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & -q \\ 0 & 0 & 1 \end{pmatrix} \tag{10.7}$$

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{10.8}$$

$$\mathbf{T}_2 = \begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \tag{10.9}$$

とします。すると、$\mathbf{P}, \mathbf{P}', \mathbf{P}'', \mathbf{P}'''$ の関係は、次のようにシンプルに記述できます。

$$\mathbf{P}' = \mathbf{T}_1\mathbf{P} \tag{10.10}$$
$$\mathbf{P}'' = \mathbf{R}\mathbf{P}' \tag{10.11}$$
$$\mathbf{P}''' = \mathbf{T}_2\mathbf{P}'' \tag{10.12}$$

これは、ただ単に式 (10.4) 〜 (10.6) を書き直しただけにすぎません。さて、ここで式の (10.10) と (10.11) に注目して下さい。両方に $\mathbf{P}'$ が出てきますから、これをまとめてしまうと次のようになります。

$$\begin{aligned}\mathbf{P}'' &= \mathbf{R}\mathbf{P}' \\ &= \mathbf{R}(\mathbf{T}_1\mathbf{P}) \\ &= (\mathbf{R}\mathbf{T}_1)\mathbf{P}\end{aligned} \tag{10.13}$$

これを、さらに式 (10.12) とまとめてしまえば、

$$\begin{aligned}\mathbf{P}''' &= \mathbf{T}_2\mathbf{P}'' \\ &= \mathbf{T}_2(\mathbf{R}\mathbf{T}_1\mathbf{P}) \\ &= (\mathbf{T}_2\mathbf{R}\mathbf{T}_1)\mathbf{P}\end{aligned} \tag{10.14}$$

となります。つまり 3 つの行列の積 $\mathbf{T}_2\mathbf{R}\mathbf{T}_1$、すなわち

$$\begin{pmatrix}1 & 0 & p\\ 0 & 1 & q\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0 & -p\\ 0 & 1 & -q\\ 0 & 0 & 1\end{pmatrix} \tag{10.15}$$

の結果である行列に $\mathbf{P}$ を掛けると、そのまま $\mathbf{P}'''$ が計算できます。

これが何を意味するのかと言いますと、(10.15) 式は図形の真ん中を中心に回転させる行列の求め方を示しているということになるわけです。行列の積の計算は随分面倒なものですが、TFK_Vector と TFK_Matrix を用いれば簡単にプログラムで実現することができます。例えば、回転の中心位置が $(5, 3)$ で、回転角度が 15 度であるような行列は、次のようにして作成することができます。(もちろん、このリストは作成の方法を示しているだけなので、そのままでは動作しません。)

合成変換の作成例

```
// (-5, -3) 平行移動する行列
TFK_Matrix T1;

// 15 度回転する行列
TFK_Matrix R;

// (5, 3) 平行移動する行列
TFK_Matrix T2;

// T2 * R * T1 を代入する行列
TFK_Matrix C;

double sinValue, cosValue;

// T1 の設定
T1[0][2] = -5.0;
T1[1][2] = -3.0;

// R の設定
sinValue = sin(15.0 * TFK_PI/180.0);
cosValue = cos(15.0 * TFK_PI/180.0);
R[0][0] = cosValue;
R[0][1] = -sinValue;
R[1][0] = sinValue;
R[1][1] = cosValue;

// T2 の設定
T2[0][2] = 5.0;
T2[1][2] = 3.0;

// T2 * R * T1 を C に代入
C = T2 * R * T1;
```

最後の行で C という変数に 3 つの行列の積を代入しています。TFK_Matrix は、上記のように「*(アスタリスク)」記号を用いて行列同士の積をそのまま求めて代入することができます。ここで注意しなければならないのは、行列の積は交換法則が成り立たないということです。通常の実数の場合、積の順番を入れ替えても結果は同じですが、行列の場合は順番が異なると結果も異なります。ですから、最後の積演算は順番に注意して下さい。

## 10.5 サンプルプログラム

今日の課題の基本となるプログラムリストです。

Rot_Samp.cpp

```
 1: #include <TinyFK/TinyFK.h>
 2:
 3: // 4 角形生成用関数
 4: void makeRectangle(TFK_Vector pos[4], int colorID)
 5: {
 6:     TFK_Line        line;
 7:     int             i;
 8:
 9:     for(i = 0; i < 3; i++) {
10:         line.create(pos[i], pos[i+1]);
11:         line.setPalette(colorID);
12:     }
13:     line.create(pos[3], pos[0]);
14:     line.setPalette(colorID);
15:
16:     return;
17: }
18:
19: int main(int argc, char *argv[])
20: {
21:     TFK_Window      win;
22:     TFK_Palette     pal;
23:     TFK_Vector      pos[4], nPos[4];
24:     TFK_Matrix      matrix;
25:     int             i;
26:
27:     win.setPlaneMode(true);
28:     win.setBaseSize(520, 520);
29:     win.setCanvas(10, 10, 500, 500);
30:     win.open();
31:
32:     pal.setColor(1, 1.0, 0.0, 0.0);
33:     pal.setColor(2, 0.0, 0.0, 1.0);
34:
35:     // 座標設定 (含同時座標)
36:     pos[0].set(2.0, 2.0, 1.0);
37:     pos[1].set(2.0, 8.0, 1.0);
38:     pos[2].set(8.0, 8.0, 1.0);
39:     pos[3].set(8.0, 2.0, 1.0);
40:
41:     // 正方形生成
42:     makeRectangle(pos, 1);
43:
44:     // (-3, -1) 平行移動行列生成
45:     matrix[0][2] = -3.0;
46:     matrix[1][2] = -1.0;
47:
48:     // 1 次変換
49:     for(i = 0; i < 4; i++) {
50:         nPos[i] = matrix * pos[i];
51:     }
52:
53:     // 移動後の正方形生成
54:     makeRectangle(nPos, 2);
55:
56:     while(win.wait() == true) {
57:         win.draw();
58:     }
59:
60:     return 0;
61: }
```

前回のサンプルとほとんど同様ですが、23 行目の配列サイズが「4」となっていることに注意して下さい。

## 10.6 今回の課題

⋆.....必ずやるべき基本的な課題
⋆⋆.....理解度を見るための標準的な問題
⋆⋆⋆...余裕のある人のみチャレンジしてほしい難問

**問題 1:** ⋆

Rot_Samp.cpp を元に、原点 (ウィンドウの中心) を中心に $\frac{\pi}{18}$ ラジアン $= 10°$ 回転させた正方形を描画するプログラムを作成せよ。

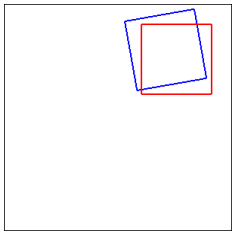

図 59: 問題 1 の出力画面

**問題 2:** ⋆⋆

Rot_Samp.cpp を元に、$(5, 5)$ を中心に $\frac{\pi}{18}$ ラジアン回転させた正方形を描画するプログラムを作成せよ。

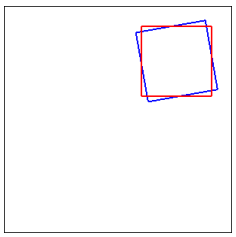

図 60: 問題 2 の出力画面

**問題 3:** ⋆⋆⋆

図 61 のように、問題 2 に加えてさらに $\frac{2}{18}\pi$, $\frac{3}{18}\pi$, ... , $\frac{8}{18}\pi$ ラジアン回転させたそれぞれの正方形を描画するプログラムを作成せよ。

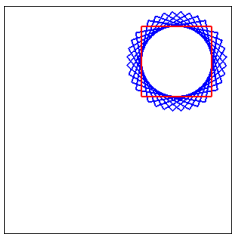

図 61: 問題 3 の出力画面

締め切りは次回授業開始時です。