コンピュータグラフィクスの基礎理論 第 9 回資料

渡辺大地

自分の仕事に強固な理論的基礎を与えるべきときが来た。
— サム・モルガン

9.1 行列と 1 次変換

前回までは、線分、領域、曲線の生成に関する議論を進めてきました。これにより、それぞれの図形要素をどのようにプログラムで表現すればよいのかが明らかになりました。今回からは、それらの図形に対して移動や拡大縮小を行うための理論を述べていきます。その理論の名前は「**1 次変換**」と言います。この 1 次変換という理論を用いることで、図形の平行移動や回転移動、あるいは拡大や縮小という操作を体系的に実現することが可能となります。

1 次変換は、行列と密接に関連しています。行列は、数学の様々な分野の中でも極めて抽象的な概念であるため、一見すると何の役に立つのかわかりづらいものです。実際のところ、行列の利用用途は 2 種類に大別できます。1 つは、前半で取り上げたように連立方程式の解を求めるための道具としてのものです。そしてもう 1 つが、これから紹介する 1 次変換ということになります。

9.2 同次座標

これまで、平面上のベクトルは (x, y) の 2 つの数値 (成分) で表されるということを述べました。しかし、1 次変換を効果的に用いるためにもう 1 つ、「**同次座標**」という数値 (成分) をその中に加えることにします。

$$(x, y) \qquad \longrightarrow \qquad (x, y, 1)$$

同次座標は、必ず 1 になります。 例えば、$(5, 3)$ は $(5, 3, 1)$ に、$(-4, 2)$ は $(-4, 2, 1)$ になります。必ず 1 が来るのであれば、まったく意味がないように思えるかもしれません。確かに、位置や方向の表現にはまったく意味がありません。しかし、この同次座標を加えておくと、後で行列を利用して平行移動を表現できるのです。とにかく、「同次座標は 必ず 1 である。」とだけ覚えておいて下さい。

9.3 3 行 3 列の行列

行列も、今回から 3 行 3 列まで拡張します。つまり、

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{pmatrix}$$

のような行列を今後扱っていきます。これまでは 2 行 2 列の場合だけを扱ったのですが、今回は 3 行 3 列の行列の場合の演算を改めて定義します。

3 行 3 列行列の和と差:

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{pmatrix} \pm \begin{pmatrix} p_0 & p_1 & p_2 \\ q_0 & q_1 & q_2 \\ r_0 & r_1 & r_2 \end{pmatrix} = \begin{pmatrix} a_0 \pm p_0 & a_1 \pm p_1 & a_2 \pm p_2 \\ b_0 \pm q_0 & b_1 \pm q_1 & b_2 \pm q_2 \\ c_0 \pm r_0 & c_1 \pm r_1 & c_2 \pm r_2 \end{pmatrix} \tag{9.1}$$

3 行 3 列の場合の行列でも、やはり和や差が各要素を足したり引いたりすることは同様です。ただ、この科目中では今後行列の和と差を用いることはありません。

3 行 3 列行列とベクトルとの積:

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{pmatrix} \begin{pmatrix} x \\ y \\ t \end{pmatrix} = \begin{pmatrix} a_0 x + a_1 y + a_2 t \\ b_0 x + b_1 y + b_2 t \\ c_0 x + c_1 y + c_2 t \end{pmatrix} \tag{9.2}$$

3 行 3 列の行列は、2 次元のベクトルに同次座標が加わったものと掛け算をすることができます。結果は上のとおりです。同次座標の解説でも述べた通り、t は常に 1 となります。この式は、これから最も頻繁に出てきます。

3 行 3 列行列同士の積:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

のとき、

$$c_{ij} = \sum_{k=0}^{2} a_{ik} b_{kj} \tag{9.3}$$

実際にやっていることは掛け算と足し算だけなのですが、総和記号を用いているため、やや複雑に見えます。プログラム中では、TFK_Vector や TFK_Matrix を用いる場合には自分でこの演算式を記述する必要はありません。とりあえず、あくまで定義として眺めて下さい。

9.4 行列による平行移動

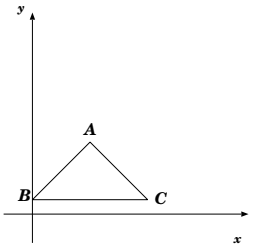

図 45: 平面上の 3 角形

ここで、図 45 にある頂点 A, B, C の位置座標を仮に $(A_x, A_y), (B_x, B_y), (C_x, C_y)$ とします。さて、この 3 角形を $(4, 2)$ だけ平行移動してみましょう。移動した様子が次の図 46 です。

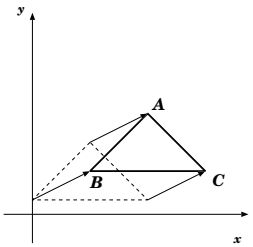

図 46: 3 角形を平行移動した様子

このとき、頂点 A, B, C の座標値はそれぞれ

$$(A_x + 4,\ A_y + 2),$$
$$(B_x + 4,\ B_y + 2),$$
$$(C_x + 4,\ C_y + 2)$$

になります。つまり、各頂点の位置ベクトルにベクトル $(4, 2)$ を足していることになります。

一般に、位置座標が (x, y) である点を (p, q) だけ平行移動した場合、平行移動先の座標は $(x + p,\ y + q)$ になります。この計算は行列で表すことができて、以下のようなものです。

平行移動行列:

任意の頂点座標 (x, y) を (p, q) 平行移動させるには、次のような行列に位置ベクトル $(x, y, 1)$ を掛ければよい。

$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \tag{9.4}$$

実際に掛けてみると、

$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + p \\ y + q \\ 1 \end{pmatrix} \tag{9.5}$$

となりますから、先ほどの解説と一致することがわかります。

9.5 行列による回転移動

まず、一般的なベクトル $\mathbf{V} = (p, q)$ を考えます。(図 47)

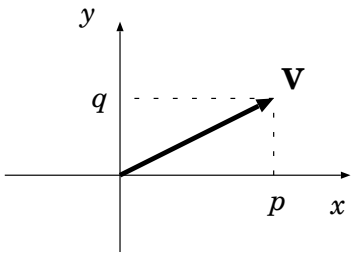

図 47: $\mathbf{V} = (p, q)$

このベクトルを、x 軸上のベクトル $\mathbf{E}_0$ と y 軸上のベクトル $\mathbf{E}_1$ が $\mathbf{V} = \mathbf{E}_0 + \mathbf{E}_1$ となるように 2 つのベクトルに分解してみます。すると、

$$\begin{aligned} \mathbf{E}_0 &= (p, 0) \\ \mathbf{E}_1 &= (0, q) \end{aligned} \tag{9.6}$$

となります。(図 48)

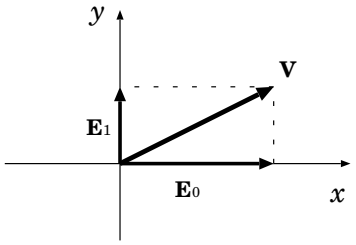

図 48: ベクトルの分解の様子

さて、$\mathbf{V}, \mathbf{E}_0, \mathbf{E}_1$ の 3 個のベクトルを全て角度 θ だけ回転させてみましょう。そうすると、図 49 のようになります。(図 49 では $\theta = 30$ 度 $= \frac{\pi}{6}$ ラジアン です。)

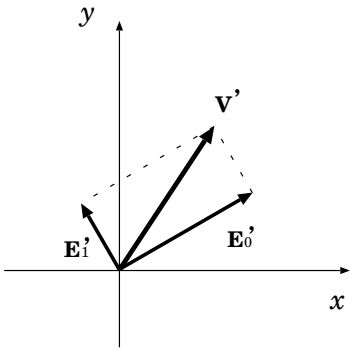

図 49: ベクトルの回転

回転後のベクトルをそれぞれ $\mathbf{V}', \mathbf{E}_0', \mathbf{E}_1'$ としたとき、回転させた後でも $\mathbf{V}' = \mathbf{E}_0' + \mathbf{E}_1'$ が成り立つことが図 49 を見ると直感的にわかると思います。また、$\mathbf{E}_0', \mathbf{E}_1'$ の x, y 各成分は 図 50 を考えてみると、次のようになることがわかります。

$$\begin{cases} \mathbf{E}_0' & = (p\cos\theta,\ \ p\sin\theta) \\ \mathbf{E}_1' & = (-q\sin\theta,\ \ q\cos\theta) \end{cases} \tag{9.7}$$

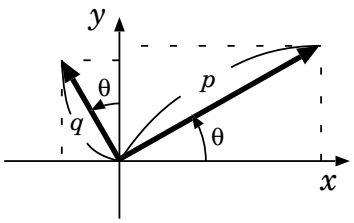

図 50: ベクトルの回転

従って、

$$\begin{aligned} \mathbf{V}' &= \mathbf{E}_0' + \mathbf{E}_1' \\ &= (p\cos\theta - q\sin\theta,\ \ p\sin\theta + q\cos\theta) \end{aligned} \tag{9.8}$$

ということになります。これを行列を使って表現すると、

$$\begin{pmatrix} p' \\ q' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ q \\ 1 \end{pmatrix} \tag{9.9}$$

ということになるわけです。

たとえば、$(2, 3)$ を原点を中心に $\frac{\pi}{6}$ ラジアン回転させた位置座標を求めるには、$\sin\frac{\pi}{6} = \frac{1}{2}$, $\cos\frac{\pi}{6} = \frac{\sqrt{3}}{2}$ ですから次のようになります。

$$\begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{3} - \frac{3}{2} \\ \frac{3}{2} + \frac{3\sqrt{3}}{2} \\ 1 \end{pmatrix} \tag{9.10}$$

9.6 サンプルプログラム

Linear_Samp.cpp

```
 1: #include <TinyFK/TinyFK.h>
 2:
 3: // 3 角形生成用関数
 4: void makeTriangle(TFK_Vector pos[3], int colorID)
 5: {
 6:     TFK_Line        line;
 7:
 8:     line.create(pos[0], pos[1]);
 9:     line.setPalette(colorID);
10:     line.create(pos[1], pos[2]);
11:     line.setPalette(colorID);
12:     line.create(pos[2], pos[0]);
13:     line.setPalette(colorID);
14:
15:     return;
16: }
17:
18: int main(int argc, char *argv[])
19: {
20:     TFK_Window      win;
21:     TFK_Palette     pal;
22:     TFK_Vector      pos[3], nPos[3];
23:     TFK_Matrix      matrix;
24:
25:     win.setPlaneMode(true);
26:     win.setBaseSize(520, 520);
27:     win.setCanvas(10, 10, 500, 500);
28:     win.open();
29:
30:     pal.setColor(1, 1.0, 0.0, 0.0);
31:     pal.setColor(2, 0.0, 0.0, 1.0);
32:
33:     // 座標設定 (含む同次座標)
34:     pos[0].set(0.0, 0.0, 1.0);
35:     pos[1].set(5.0, 5.0, 1.0);
36:     pos[2].set(-5.0, 5.0, 1.0);
37:
38:     // 3 角形生成
39:     makeTriangle(pos, 1);
40:
41:     // (0.5, -0.2) 平行移動行列生成
42:     matrix[0][2] = 0.5;
43:     matrix[1][2] = -0.2;
44:
45:     // 1 次変換
46:     nPos[0] = matrix * pos[0];
47:     nPos[1] = matrix * pos[1];
48:     nPos[2] = matrix * pos[2];
49:
50:     // 移動後の 3 角形生成
51:     makeTriangle(nPos, 2);
52:
53:     while(win.wait() == true) {
54:         win.draw();
55:     }
56:
57:     return 0;
58: }
```

解説を以下に述べます。

- 22 行目では、「**配列**」というものを生成しています。配列とは、簡単に言えば変数が複数集まったものです。例えば 22 行目の pos の場合、pos[0], pos[1], pos[2] の 3 個の変数を生成したのと同じことになります。

  配列の生成するには、変数を定義するのとほぼ同様で、名前の後ろにカギ括弧で配列の長さを指定します。また、サンプルプログラムのようにカンマで続けて別の配列を定義することも可能です。

- 4 〜 16 行目にある makeTriangle は、3 角形を生成するための関数です。main 関数の前にこれを記述しておくことによって、main の中で makeTriangle を利用することが可能になります。引数として 3 角形の頂点座標を表す TFK_Vector 型の配列と、色番号を表す整数をとっています。C++ で配列の中身全てを関数に受け渡す場合は、このサンプルプログラムのように記述します。

- 22 行目の pos, nPos 配列は、それぞれ位置を表わすベクトルを保存するための変数です。また、23 行目は 1 次変換を用いて平行移動先を計算するための行列を生成しています。TFK_Matrix 型は、これまで 2 行 2 列の行列として扱ってきましたが、実は 3 行 3 列の行列として扱うことができます。生成時は以下のような行列となっています。

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

  これは、単位行列 (恒等行列) の 3 行 3 列版です。

- 25 行目にある「setPlaneMode」という関数は、同次座標を TinyFK 中で用いる場合に必要となるものです[4]。

- 34〜36 行目で 3 角形の各頂点に対して位置座標を指定していますが、これまでと違い 3 番目の引数があります。ここが同次座標になります。1 次変換をプログラムで用いたい場合は、同次座標も指定していく必要があります。

- 42,43 行目で、行列に値を設定しています。このサンプルでは $(0.5, -0.2)$ だけ平行移動させるための行列を作成しています。まず、42 行目で 1 行目 3 列目に 0.5 を、43 行目で 2 行目 3 列目に -0.2 を設定しています。そして 46 〜 48 行目で、行列とベクトルの積を算出しています。それぞれの結果が平行移動先ということになります。

9.7 今回の課題

以下の問題中、「原点」とはウィンドウの中心を指すこととします。

⋆.....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

[4]TinyFK で 3 次元による表示を行う際には、この設定で「false」を引数に入力します。

問題 1: ⋆

Linear_Samp.cpp を元に、以下の図 51 のように元の 3 角形を原点を中心に $18° = \pi/10$ ラジアン 回転したものを描画するプログラムを作成せよ。

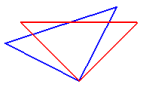

図 51: 問題 1 の実行例

問題 2: ⋆⋆

Linear_Samp.cpp を元に、以下の図 52 のように元の 3 角形を原点を中心に $18° = \pi/10$ ラジアン ずつ回転したものを 1 回転分 (つまり元図形とさらに 19 個の 3 角形) を描画するプログラムを作成せよ。

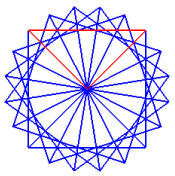

図 52: 問題 2 の実行例

問題 3: ⋆⋆⋆

ウィンドウ中を 4 回クリックすると、その 4 点を制御点とする Bézier 曲線を描き、さらに図 53 のようにその曲線を原点を中心に $18° = \pi/10$ ラジアン ずつ回転したものを 1 回転分を描画するプログラムを作成せよ。

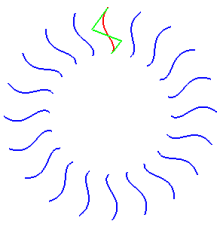

図 53: 問題 3 の実行例

締め切りは次回授業開始時です。