オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics 2D編 － 第４回　スプライトの変形 ◀

スプライトの変形

Direct3DXでは、スプライトの拡大縮小や回転といった変形を行う場合、どのように変形するのかを格納した「行列」を指定する必要があります。
ID3DXSprite::Drawメソッドが呼び出されると、スプライトの座標と行列から、スクリーン座標が計算されます。行列には、回転、スケーリング、平行移動、せん断などがあります。回転しながら移動するといった場合は、回転のための行列と移動のための行列が必要となります。複数の行列を同時に用いる場合は、行列を乗算し、連結します。

いろいろな行列

DirectX Graphicsでは、行列を格納する構造体としてD3DMATRIX構造体、D3DXMATRIX構造体、D3DXMATRIXA16構造体が定義されています。いずれも４行４列の２次元配列として定義されるfloat型のメンバを持っていますが、D3DXMATRIX構造体およびD3DXMATRIXA16構造体は、四則演算子がオーバーロードされていたり、行列を作成するD3DXMatrix系の関数に使用できるといった利点があります。

代表的な行列を以下に示します。ある点をｘ，ｙ，ｚのそれぞれについてt_x，t_y,t_zだけ移動する行列は、以下のようになります。

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}$$

オブジェクトを拡大縮小させる操作をスケーリングといいます。ｘ，ｙ，ｚのそれぞれのスケール(倍率)をs_x，s_y，s_zとすると、スケーリング行列は以下のようになります。

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

回転では、軸ごとに行列を使います。ｘ軸回りの回転は、ｙｚ平面に対して影響を与えるので、以下のような行列になります。

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

ｙ軸回りの回転は、ｚｘ平面に対して影響を与えるので、以下のような行列になります。

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

ｚ軸回りの回転は、ｘｙ平面に対して影響を与えるので、以下のような行列になります。

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

行列を作成する関数

Direct3DXには、行列を作成するための関数が多数用意されています。これらを使用すると、行列の知識が無くても、簡単に目的の行列を作成することができます。

D3DXMatrixIdentity関数	単位行列を作成します
D3DXMatrixInverse関数	逆行列を作成します
D3DXMatrixTranspose関数	転置行列を作成します
D3DXMatrixRotationX関数	ｘ軸を回転軸とした回転行列を作成します
D3DXMatrixRotationY関数	ｙ軸を回転軸とした回転行列を作成します
D3DXMatrixRotationZ関数	ｚ軸を回転軸とした回転行列を作成します
D3DXMatrixRotationYawPitchRoll関数	ｙ，ｘ，ｚ軸を回転軸とした回転行列を作成します
D3DXMatrixRotationAxis関数	任意の軸を回転軸とした回転行列を作成します
D3DXMatrixScaling関数	スケーリング行列を作成します
D3DXMatrixTranslation関数	平行移動行列を作成します
D3DXMatrixAffineTransformation関数	アフィン変換行列を作成します
D3DXMatrixTransformation関数	座標変換行列を作成します
D3DXMatrixShadow関数	頂点を平面に射影する行列を作成します
D3DXMatrixReflect関数	平面の座標系を反映した行列を作成します

複数の行列を適用したい場合は、行列を乗算して結合します。このとき、掛けた順に適用されます。回転行列に移動行列を掛けるのと、移動行列に回転行列を掛けるのでは、行列の性質上、行列そのものが異なるものになります。実世界と同じように、回転してから移動するのと、移動してから回転するのとでは、まったく別の状態となるのです。通常は回転行列、スケーリング行列、平行移動行列またはスケーリング行列、回転行列、平行移動行列の順に掛けます。

このとき問題になるのが、回転行列を掛ける順番です。回転行列は、ｘ，ｙ，ｚ軸で個別の行列を用います。ｘ，ｙ，ｚ軸の順で掛けた行列と、ｚ，ｙ，ｘの順で掛けた行列は、別の行列になります。この順番は、そのときの状況により正解が変わりますが、2Dではｚ，ｘ，ｙの順に掛けます。

変換行列の設定

変換行列が作成できたら、ID3DXSprite::SetTransformメソッドを呼び出してシステムに設定します。以後、すべてのスプライトの描画時に、設定した変換行列が適用されます。

```
// スプライト描画開始(pD3DXSpriteは、スプライトインタフェース)
pD3DXSprite->Begin(D3DXSPRITE_ALPHABLEND | D3DXSPRITE_SORT_DEPTH_FRONTTOBACK);

// 行列の作成
// ｚ軸回転(４５度回転)
D3DXMATRIX   rot_z;
D3DXMatrixRotationZ(&rot_z, D3DXToRadian(45.0f));

// スケーリング(ｘ，ｙ，ｚ方向それぞれ1.5倍に拡大)
D3DXMATRIX   scale;
D3DXMatrixScaling(&scale, 1.5f, 1.5f, 1.5f);

// 行列を連結する
D3DXMATRIX   mat = rot_z * scale;

// 座標変換行列の設定(pD3DDeviceは初期化済みのDirect3DDevice9オブジェクト)
pD3DXSprite->SetTransform(&mat);

// スプライト描画(pD3DTextureは、イメージが格納されたDirect3DTexture9オブジェクト)
RECT         src = {0, 0, 120, 120};       // 転送元領域
D3DXVECTOR3  dest(320.0f, 240.0f, 0.0f);   // 転送先座標
pD3DXSprite->Draw(pD3DTexture, &src, NULL, &dest, 0xffffffff);

// スプライト描画終了
pD3DXSprite->End();
```

課　　題

スプライトクラスに、スプライトを変形して描画する機能を追加しましょう。

(１)スプライトを変形して描画するDrawEx関数を作成します。以下のプロトタイプをISpriteクラス定義の適切な場所に追加しましょう。

```
virtual void DrawEx(const RECT* pSrc, const D3DXVECTOR3& inPos,
                    const D3DXVECTOR3& inScale, const D3DXVECTOR3& inRotation,
                    const D3DXVECTOR3* pCenter = NULL, const float inAlpha = 1.0f,
                    const D3DCOLOR inColor = 0x00ffffff) = 0;
```

DrawEx関数の仕様は、以下のとおりです。

DrawEx　　描画

スプライトを変形して描画します(実際は、描画スプライトリストへ登録するだけです)。

試
```
void DrawEx(const RECT* pSrc, const D3DXVECTOR3& inPos,
            const D3DXVECTOR3& inScale, const D3DXVECTOR3& inRotation,
            const D3DXVECTOR3* pCenter = NULL, const float inAlpha = 1.0f,
            const D3DCOLOR inColor = 0x00ffffff);
```

pSrc	転送元の領域を格納したRECT構造体のアドレス。NULLを指定すると、イメージ全体が描画される
inPos	スプライトの描画位置を格納したD3DXVECTOR3構造体のアドレス
inScale	スプライトのスケーリング倍率を格納したD3DXVECTOR3構造体のアドレス
inRotation	スプライトの回転角を格納したD3DXVECTOR3構造体のアドレス。ｘ，ｙ，ｚ軸それぞれについて回転できるが、実際にはｚ軸回転しか正しく描画されない
pCenter	描画や変形の基準点となる、スプライトの中心を示すD3DXVECTOR3構造体のアドレス。NULLの場合は、座標(0,0,0)が使われる
inAlpha	透明度。範囲は0.0f(完全に透明)から1.0f(完全に不透明)
inColor	D3DCOLOR_XRGBマクロで作成した色値。この値とイメージの色値が乗算されます。0xffffffffを指定すると、元の色が維持される

(２)CSpriteクラスにDrawEx関数を追加します。この関数のプロトタイプを(１)をもとに作成し、適切な場所に追加しましょう。

(３)DrawEx関数を実装します。以下のプログラムを完成させ、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          描　画                                                   */
/*****************************************************************************************************/
void CSprite::DrawEx(const RECT* pSrc, const D3DXVECTOR3& inPos, const D3DXVECTOR3& inScale, const D3DXVEC
TOR3& inRotation,
                     const D3DXVECTOR3* pCenter, const float inAlpha, const D3DCOLOR inColor)
{
    // 色設定
    const D3DCOLOR   COLOR = ((DWORD)(inAlpha * 255.0f) << 24) | (inColor & 0x00ffffff);

    // 回転ベクトルをクォータニオンに変換
    D3DXQUATERNION   qt;
    ::D3DXQuaternionRotationYawPitchRoll(&qt, D3DXToRadian(inRotation.y),
                                             D3DXToRadian(inRotation.x),
                                             D3DXToRadian(inRotation.z));

    // 行列設定
    D3DXMATRIX   mat;
    ::D3DXMatrixTransformation(&mat, NULL, NULL, &inScale, NULL, &qt, &inPos);
    m_pD3DXSprite->???????????(&mat);

    // 描画
    m_pD3DXSprite->????(m_pD3DTexture, pSrc, pCenter, NULL, COLOR);
}
```

(４)CNullSpriteクラスに「何もしない」DrawEx関数を実装しましょう。