ESライブラリ＆＆
ゲームプログラミング

○　３Ｄ編 - 第２回　ワールド変換行列　○

ワールド変換行列

・3Dグラフィクスは、モデルを3D空間に配置し、仮想世界を構築する
・「3D空間にモデルの配置を行う」には、モデルの座標をワールド座標に変換しなければならない
・ワールド変換行列は、モデルの座標を変換し、「3D空間にモデルの配置を行う」ための行列
・頂点座標の変換には４行４列の行列が使われる

仮想世界

モデルは、頂点情報が定義された数値の集まりです。これを演算によって画像に変換し、画面に表示するのが3Dグラフィクスです。モデルを定義するための頂点情報は、モデルの原点から「どのぐらい離れているか」という相対情報です。各モデルごとに独自の中心点を持ち、そこからの距離を定義しています。3Dグラフィクスでは、これらの独立して定義されている各モデルの頂点を「共通の空間」に配置し、仮想世界を構築していきます。

ここで、2Dグラフィクスを考えてみます。2Dグラフィクスで画面を構築するには、さまざまな画像を用意しておき、それを「共通の空間」であるバックバッファ(ディスプレイの裏画面)に転送することによって行います。物理的に存在するディスプレイまたはビットマップの座標を直感的に指定して画像を転送することができます。
これに対し3Dグラフィクスは、頂点は単なる数値で、またディスプレイのような物理的な座標系もありません。「共通の空間」は、存在はするものの、あくまでも概念的な仮想空間となります。そのため、仮想世界にモデルを配置するには、各モデルが独立して持つ頂点の座標系を「共通の空間」である仮想空間の座標系へと座標変換しなければなりません。そのために必要となるのが「ワールド変換行列」をはじめとする４行４列の変換行列です。

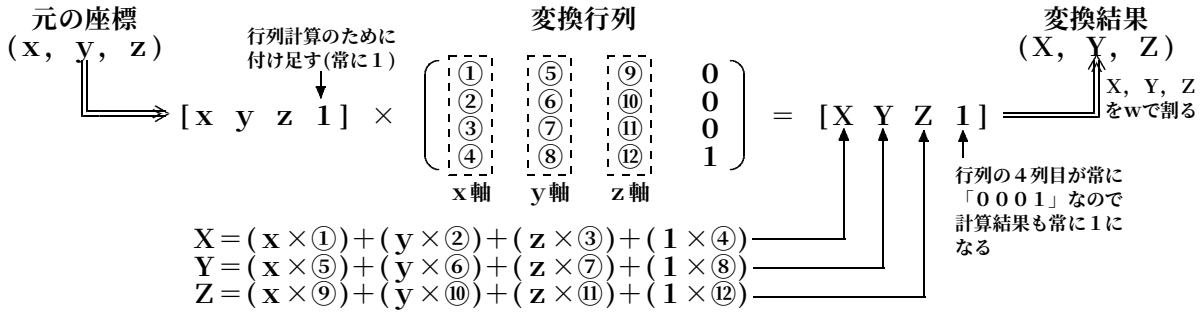
ワールド変換行列

「共通の空間」の座標系をワールド座標系と呼びます。モデルは、モデルを中心とした独自の座標(モデル座標系)で定義されています。これをワールド座標系に変換することにより、「共通の空間」へ配置することができます。
「共通の空間」は概念的な仮想空間であるため、ワールド座標に直接モデルを配置することはできません。どこに、どのように配置するかを格納した「行列」を作成し、座標変換をして配置します。

座標変換は、変換行列によって行われます。モデル座標系をワールド座標系に変換する行列のことをワールド変換行列(ワールドトランスフォーム行列)と呼びます。ワールド変換行列には、回転、スケーリング、平行移動などがあります。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。複数の変換行列を用いる場合は、行列同士を乗算し、合成します。

座標変換では、頂点を示すベクトル(位置ベクトル…座標は位置ベクトルという扱いになります)に変換行列が掛けられ、新しい座標系に変換されます。3Dで座標変換に使用する行列は、同次座標を用いるため４×４の行列でなければなりません。3Dベクトルと４×４の行列を掛けることはできないので、3Dベクトルに４つ目の成分wを１としたものを加え、4Dベクトルに変換してから処理が行われます。

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ ◀座標変換で使用される４×４の行列



Direct3Dでは４×４の行列をD3DMATRIXおよびD3DXMATRIX構造体、ESライブラリではMatrix構造体で定義しています。

座標変換行列

座標変換に用いる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だけ移動する変換行列は、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。

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

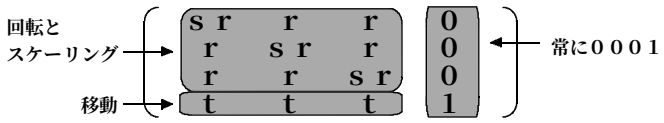
ｙ軸回りの回転は、ｚｘ平面に対して影響を与えるので、以下のような行列になります。

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

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。

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

行列の各成分は、以下のように移動、スケーリング、回転と対応しています。



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。これらは、必要な情報を引数に指定するだけで目的の行列を作成することができるので、ベクトルや行列の知識が無くても扱うことができます。

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

行列の合成

複数の行列を適用したい場合は、行列を乗算して合成します。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とでは、行列の性質上、行列そのものが異なるものになります。

ワールド座標の原点をもとに計算が行われるので、(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ります。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るので、状況にあった順に掛けます。

```
// ワールド変換行列の作成
// ｘ軸回転(０度回転)
D3DXMATRIX   rot_x;
D3DXMatrixRotationX(&rot_x, D3DXToRadian(0.0f));

// ｙ軸回転(４５度回転)
D3DXMATRIX   rot_y;
D3DXMatrixRotationY(&rot_y, D3DXToRadian(45.0f));

// ｚ軸回転(０度回転)
D3DXMATRIX   rot_z;
D3DXMatrixRotationZ(&rot_z, D3DXToRadian(0.0f));

// 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);

// 平行移動(ワールド座標(0.0, 0.0, 3.0)に移動)
D3DXMATRIX   trans;
D3DXMatrixTranslation(&trans, 0.0f, 0.0f, 3.0f);

// 各行列を合成し、ワールド変換行列にする
D3DXMATRIX   world = rot_y * rot_x * rot_z * scale * trans;
```

ワールド変換行列の設定

ワールド変換行列を作成できたら、IDirect3DDevice9::SetTransformメソッドを呼び出してシステムに設定します。以後、すべてのオブジェクトのレンダリング時に、設定したワールド変換行列が適用されます。

```
// ワールド変換行列設定(pD3DDeviceは初期化済みのDirect3DDevice9オブジェクト)
pD3DDevice->SetTransform(D3DTS_WORLD, &world);

// 描画(pMeshは初期化済みのID3DXMeshオブジェクト)
// ワールド変換行列が適用され、ｙ軸を回転軸として45度回転し、1.5倍に拡大された
// メッシュがワールド座標(0.0, 0.0, 3.0)に配置される
pMesh->DrawSubset(0);
```

ESライブラリでは、モデル描画のたびに、座標・スケーリング・回転角の情報をもとにワールド変換行列を作成してシステムに適用し、描画を行っています("Model.cppのDraw関数")。

ESライブラリでの対応

ESライブラリでは、変換行列の生成およびデバイスへの適用を簡単に行えるように、頂点バッファ、モデル、アニメーションモデルの各クラスに座標、姿勢(回転角)、拡大縮小率を設定できる関数が備わっています。それぞれの値を設定しておくだけで、レンダリング前に変換行列が作成され、デバイスへ適用後、描画されるようになっています。

モデル座標系(ローカル座標系)

モデル座標系(ローカル座標系)は、モデル(3Dオブジェクト)が独自に持っている座標空間です。モデルは、頂点を指定して表現されています。これらの点はモデル座標系、いわゆるモデル独自の空間に定義されています。

この座標系の原点はオブジェクトの中心になります。中心は、必ずしも重心などの幾何学上の中心とは限りませんが、回転したりスケーリングしたりする場合の基準となる点となります。モデルの座標は、原点からどのぐらい離れているかという、相対座標で表現されています。

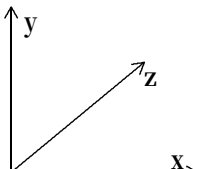
ワールド座標系

ワールド座標系は、デバイス内に表現される仮想的な世界が持つ座標系です。この世界は、モデルを配置するための空間です。ゲームなどの独自の世界は、モデル座標系で定義したオブジェクトをワールド座標系、つまりモデルを仮想世界に配置して独自の世界感を作成しています。このとき、モデル座標系からワールド座標系へ座標変換が行われます。この座標変換をワールドトランスフォーム(ワールド変換)と呼びます。

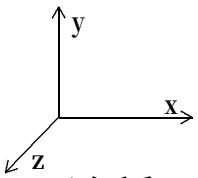
ワールド座標系は、3D空間における絶対座標系です。モデル座標系がモデルごとにあるのに対し、ワールド座標系は、デバイスに対し１つだけです。

左手座標

3D空間では、ｘ軸、ｙ軸、ｚ軸によって構成される座標系が用いられます。このとき、軸の方向付けによって２つの座標系が考えられます。ｘ軸とｙ軸で作られる平面が目の前にあると仮定したとき、ｘ-ｙ平面に対してｚ軸の方向が２種類あることが考えられます。つまり、平面からｚ軸が向こうを指している場合と、平面の手前を指している場合の２種類です。それらは、親指、人差し指、中指をそれぞれｙ軸、ｚ軸、ｘ軸に見立てたとき、左手で表されるか右手で表されるかによって、左手系あるいは右手系と呼ばれます。



▲左手系



▲右手系

DirectX Graphicsでは、ゲームと相性の良い左手系が使われています。一般的なモデリングツールでは、右手系が使われています。

頂点とベクトル

3D空間中のモデルは、頂点によって構成されます。この頂点は、位置ベクトルによって空間のある位置を表します。その位置ベクトルは、通常は原点からの相対的な位置を表すものです。ベクトルは、どのような次元も考えられますが、ゲームでは２次元か３次元が使われます。それらの要素には、慣例としてｘ，ｙ，ｚという名前が使われます(ゲームでは単なる座標で問題ないですが、慣例として位置ベクトルという扱いになっています)。

ESライブラリでは、行ベクトルが採用され、ベクトルをVector2，3，4構造体で表現します。OpenGLなど他の3Dライブラリでは、列ベクトルが採用されているものもあります。その場合には、変換行列の並びが変わります。

同次座標

ベクトルは平行移動をサポートしていません。そもそもベクトルは始点を持たない概念です。大きさと方向だけを持っていて、その始点は原点に固定されています。そのため、方向を保ったまま平行移動することができません。つまり、あるベクトルに、平行移動を表す別のベクトルを加えると、方向も変わってしまいます。これを回避するために、同次座標を導入する必要があります。

3Dベクトルの平行移動は、3D空間では不可能なので、いったん4D空間に持っていきます。4D空間内で平行移動を行い、その4D空間の断面を3D空間に変換するのです。

4D空間での点は、ｘ，ｙ，ｚに加えｗという仮想的な第４軸を基準とする第４の要素を持ちます。4Dベクトルの各要素をｗで割ったものが、3Dでの１点に対応します。このような4Dの座標系を同次座標系といいます。これは、ｗ＝１となる点群を4D空間の断面として、3D空間で認識しているということになり、このことを２つの空間で同じ同次頂点を指していると表現します。なお、ｗが０のときは除算することができないので、3D空間の無限遠にある点となります。

課　　題

(１)SetPosition関数を使い、ティーポットをワールド座標(0.0，0.0，1.0)に移動させましょう。

(２)SetRotation関数でティーポットの姿勢(向き)を変更してみましょう。

(３)SetScale関数でティーポットの大きさを変更してみましょう。

(４)キーボードの入力とMove関数を使い、ティーポットが向いている方向に進むプログラムを作成しましょう。