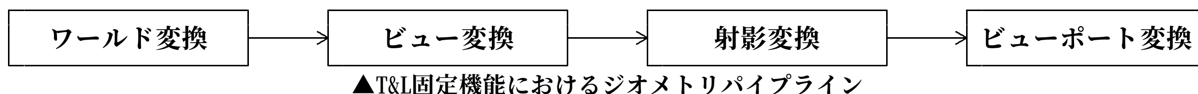
オブジェクト指向と
ゲームプログラミング

▶ DirectX Graphics 3D編 ― 第４回　ワールド変換 ◀

ジオメトリ処理(T&L固定機能)のアーキテクチャ

ジオメトリ処理とは、3Dオブジェクトの頂点データに対して、座標変換および照明演算を行い、実際に描画できるデータに変換を行う処理のことです。DirectX Graphicsのジオメトリ処理は、T&L固定機能とバーテックスシェーダがあり、どちらかを選んで使います。
T&L固定機能では、頂点データをジオメトリパイプラインで処理します。ジオメトリパイプラインとは、一連の座標変換処理のことです。3Dオブジェクトは、ワールド、ビュー、射影、ビューポートの４つの座標変換を経てピクセルに変換されます。各座標変換には変換行列と呼ばれる行列を用います。

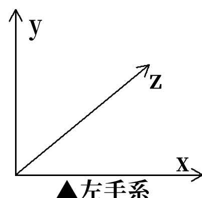


▲T&L固定機能におけるジオメトリパイプライン

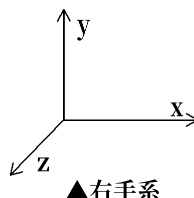
固定機能の動作は、ステートの設定によって制御します。DirectX Graphicsは、ステートマシンとして設計されています。変換行列、マテリアル、テクスチャ、レンダリングステートといった状態は、一度設定すると変更されない限り維持され続けます(ただし、IDirect3DDevice9::Resetメソッドを呼び出した場合は消去されます)。
これに対してバーテックスシェーダとは、必要な変換処理を簡単なプログラムで記述し、ビデオカードに直接命令を送る方法です。将来的には、バーテックスシェーダが主流になると言われています。

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はどちらでも扱えますが、通常は左手系を使います。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。

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、3Dオブジェクトが独自に持っている座標空間です。3Dオブジェクトは、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です。

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、3Dオブジェクトを配置するための空間です。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、つまり仮想世界に配置してシーンを作成します。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3Dオブジェクトごとにあるのに対し、ワールド座標系は１つだけです。

頂点とベクトル

　3D空間中のオブジェクトは、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。
　DirectX Graphicsでは、ベクトルをD3DVECTOR構造体およびD3DXVECTOR2，3，4構造体で表現します。

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。

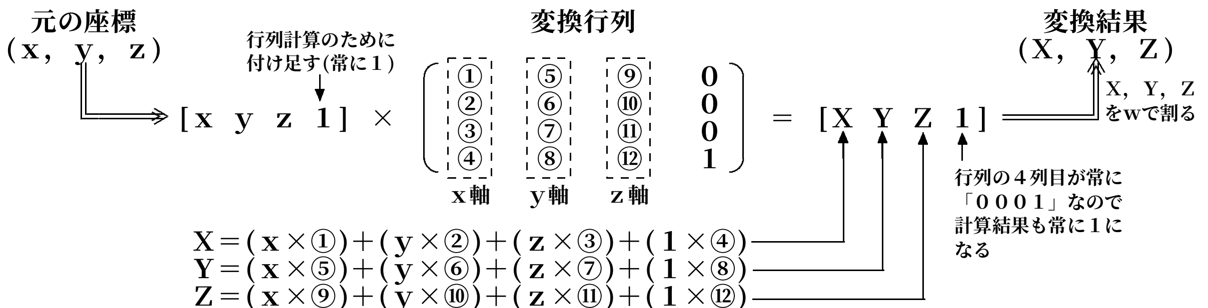
ワールド変換行列

　3Dオブジェクトをワールド座標系に配置する場合、ワールド座標の値を直接指定するのではなく、どこに、どのように配置するのかを格納した「行列」を指定します。
　DirectX Graphicsでは、3Dオブジェクトをレンダリングするためのメソッドが呼び出されると、モデル座標で定義されるオブジェクトをスクリーン座標に変換するために、一連の座標変換またはバーテックスシェーダが実行されます。
　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、せん断などが用いられます。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、行列を乗算し、合成します。

　頂点を示すベクトルに変換行列が掛けられ、新しい点に変換されます。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



DirectX Graphicsでは、座標変換に用いる４×４の行列は、D3DMATRIX構造体、D3DXMATRIX構造体、D3DXMATRIXA16構造体として定義されています。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。

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。

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

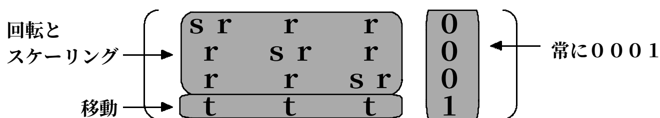
ｙ軸回りの回転は、ｚｘ平面に対して影響を与えるので、以下のような行列になります。

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

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。これらを使用すると、ベクトルや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。

関数	説明
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
```

これらの関数は、必要な情報を引数で指定するだけで目的の行列を作成することができます。作成された行列は、D3DXMATRIX構造体に格納されます。

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演算が行われるので、(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、状況に合った順に掛けます。

```
// 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);
```

課　題

頂点バッファクラスとメッシュクラスにワールド変換行列を設定する機能を追加し、オブジェクトをワールド座標に自由に配置できるようにしましょう。

(１)頂点バッファクラスに、ワールド座標の位置を指定するSetPosition関数、スケーリングを指定するSetScale関数、回転角を指定するSetRotation関数を追加します。
以下のプロトタイプをIVertexBufferクラス定義の適切な場所に追加しましょう。

```
virtual void SetPosition(const float x, const float y, const float z) = 0;
virtual void SetScale   (const float x, const float y, const float z) = 0;
virtual void SetRotation(const float x, const float y, const float z) = 0;
```

(２)CVertexBufferクラスに位置、拡大率、回転角を保持するメンバ変数を追加します。以下のプログラムを適切な場所に追加しましょう。

```
D3DXVECTOR3   m_Position;       // 位置
D3DXVECTOR3   m_Scale;          // 拡大率
D3DXVECTOR3   m_Rotation;       // 回転角
```

(３)CVertexBufferクラスのインスタンス生成時に、(２)で追加したメンバに初期値が設定されるようにします。以下の初期化子をコンストラクタに追加しましょう。

```
m_Position(0.0f, 0.0f, 0.0f), m_Scale(1.0f, 1.0f, 1.0f), m_Rotation(0.0f, 0.0f, 0.0f)
```

(４)CVertexBufferクラスにSetPosition関数、SetScale関数、SetRotation関数を追加します。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
virtual void SetPosition(const float x, const float y, const float z)
{ m_Position.x = x; m_Position.y = y; m_Position.z = z; }

virtual void SetScale   (const float x, const float y, const float z)
{ ※この関数の内容は、各自考えましょう }

virtual void SetRotation(const float x, const float y, const float z)
{ ※この関数の内容は、各自考えましょう }
```

(５)CNullVertexBufferクラスに「何もしない」SetPosition関数、SetScale関数、SetRotation関数を追加しましょう。

(６)頂点バッファクラスに、保持している位置を返すSetPosition関数、スケーリングを返すSetScale関数、回転角を返すSetRotation関数を追加します。
以下のプロトタイプをIVertexBufferクラス定義の適切な場所に追加しましょう。

```
virtual D3DXVECTOR3 GetPosition() const = 0;
virtual D3DXVECTOR3 GetScale   () const = 0;
virtual D3DXVECTOR3 GetRotation() const = 0;
```

(７)CVertexBufferクラスに位置、スケーリング、回転角を返すメンバ関数を追加します。以下のプログラムを完成させ、適切な場所に追加しましょう。

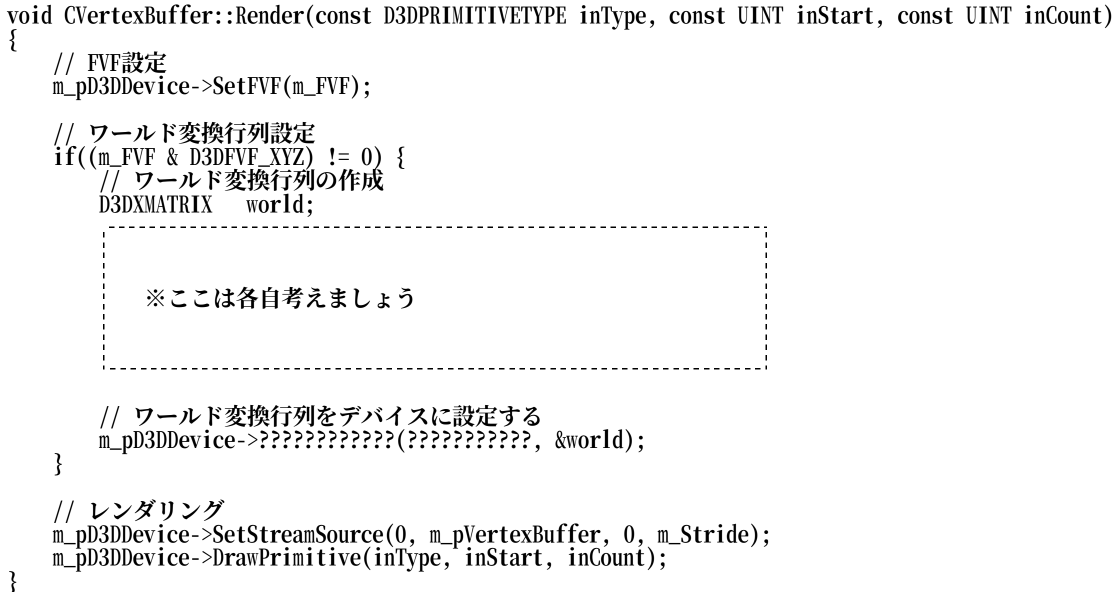
```
virtual D3DXVECTOR3 GetPosition() const { return m_Position; }
virtual D3DXVECTOR3 GetScale   () const { return ???????;    }
virtual D3DXVECTOR3 GetRotation() const { return ??????????; }
```

(８)CNullVertexBufferクラスにGetPosition関数、GetScale関数、GetRotation関数を実装します。次のプログラムを適切な場所に追加しましょう。

```
virtual D3DXVECTOR3 GetPosition() const { return D3DXVECTOR3(0.0f, 0.0f, 0.0f); }
virtual D3DXVECTOR3 GetScale   () const { return D3DXVECTOR3(0.0f, 0.0f, 0.0f); }
virtual D3DXVECTOR3 GetRotation() const { return D3DXVECTOR3(0.0f, 0.0f, 0.0f); }
```

(９)頂点バッファのレンダリング前に、ワールド変換行列を設定するようにします。

　位置、拡大率、回転角をもとに、ワールド変換行列を作成してデバイスに設定する処理をCVertexBuffer::Render関数に追加します。CVertexBuffer::Render関数を以下のように変更しましょう。

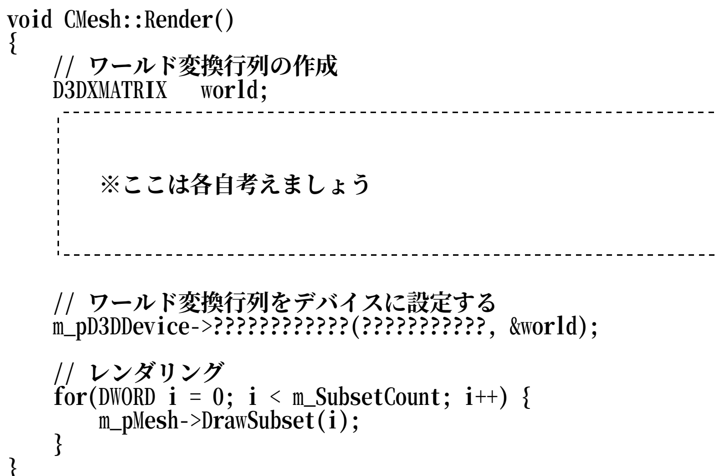


```
void CVertexBuffer::Render(const D3DPRIMITIVETYPE inType, const UINT inStart, const UINT inCount)
{
    // FVF設定
    m_pD3DDevice->SetFVF(m_FVF);

    // ワールド変換行列設定
    if((m_FVF & D3DFVF_XYZ) != 0) {
        // ワールド変換行列の作成
        D3DXMATRIX   world;

          ※ここは各自考えましょう

        // ワールド変換行列をデバイスに設定する
        m_pD3DDevice->????????????(???????????, &world);
    }

    // レンダリング
    m_pD3DDevice->SetStreamSource(0, m_pVertexBuffer, 0, m_Stride);
    m_pD3DDevice->DrawPrimitive(inType, inStart, inCount);
}
```

(10)メッシュクラスに、ワールド座標の位置、拡大率、回転角を指定する関数と、それらを返す関数を追加します。頂点バッファクラスの例(１)～(８)を参考に、IMeshクラス、CMeshクラス、CNullMeshクラスに変更を加えましょう。

(11)メッシュのレンダリング前に、ワールド変換行列を設定するようにします。

　位置、拡大率、回転角をもとに、ワールド変換行列を作成してデバイスに設定する処理をCMesh::Render関数に追加します。CMesh::Render関数を以下のように変更しましょう。



```
void CMesh::Render()
{
    // ワールド変換行列の作成
    D3DXMATRIX   world;

      ※ここは各自考えましょう

    // ワールド変換行列をデバイスに設定する
    m_pD3DDevice->????????????(???????????, &world);

    // レンダリング
    for(DWORD i = 0; i < m_SubsetCount; i++) {
        m_pMesh->DrawSubset(i);
    }
}
```

(12)ワールド変換行列が正しく適用されるかを確認します。次のプログラムを「ティーポット生成(またはシンプルシェイプ生成)」のあとに追加しましょう。

```
    m_pMesh->SetPosition(  0.0f, 0.0f, 1.0f);    // 位置
    m_pMesh->SetScale   (  0.5f, 0.5f, 0.5f);    // 拡大率
    m_pMesh->SetRotation(-15.0f, 0.0f, 0.0f);    // 回転角
```