ESライブラリ＆＆
ゲームプログラミング

○　３Ｄ編 - 第３回　カメラ　○

カメラ

・モデルを3D空間に配置しただけでは画面に描画できない
・3D空間の「どこから」「どの方向を」「どのように」見るのか、視点と視界(＝カメラ)を設定する
・カメラの設定は、広大な仮想世界の「切り取り方」を設定するものといえる
・カメラの位置、姿勢を設定するのがビュー行列、カメラからの見え方を設定するのが射影行列

カメラ

3Dグラフィクスでは、はじめに「共通の空間」(ワールド座標系)にオブジェクトを配置して仮想世界を作成します。次に行うのは、仮想世界の「どこから」「どの方向」を「どのように」見るのか、という「視点」と「視界」の設定です。
「共通の空間」は、float型の範囲の座標を持つ広大な領域です。そのまま画面に描画すると、ほぼすべてのオブジェクトが１ピクセル以下の見えないものになってしまいます。そこで、広大な空間の中から、一部の空間を切り取って画面に描画するようにします。この「切り取り方」を設定するのが、カメラ＝「視点」と「視界」です。

視点と視界は、ワールド座標の「任意の位置」から「任意の方向」の「任意の空間」をとらえることができます。Direct3Dでは、これらのことをカメラまたはビューと呼びます。カメラも頂点データを画面へ描画するための一連の座標変換の一部です。よって、行列で設定を行います。カメラの座標と方向を表す行列と、どのように見えるのかを表す行列を作成し、デバイスに適用します。その後、モデルの描画を行うと、カメラから見た仮想世界の画像が出力されるというわけです。

ビュー変換行列

カメラから見る画像は、「ワールド座標に配置された3Dオブジェクトの頂点」を「カメラを基準とした座標」に変換したものといえます。このカメラを基準とした座標をビュー座標系と呼びます。ワールド座標系からビュー座標系への変換は、ビュー変換行列と呼ばれる行列を用います。

3Dオブジェクトの頂点にワールド変換行列が掛けられることにより、ワールド座標系への変換が行われますが、さらにビュー変換行列を掛けると、カメラから見た世界、つまりカメラを原点とした座標系＝ビュー座標系へと変換されます。

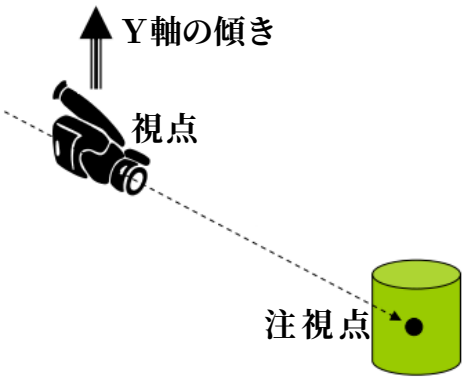
ビュー変換行列は、カメラのｘ軸を示すベクトルをｒ、カメラのｙ軸を示すベクトルをｕ、カメラのｚ軸を示すベクトルをｖ、カメラの位置を示すベクトルをｐとしたとき、以下のような行列になります。

$$\begin{pmatrix} r_x & u_x & v_x & 0 \\ r_y & u_y & v_y & 0 \\ r_z & u_z & v_z & 0 \\ -(R\cdot p) & -(U\cdot p) & -(V\cdot p) & 1 \end{pmatrix}$$

ビュー変換行列は、カメラの位置と方向を定義したものといえます。

LookAt方式

カメラを制御する場合、「視点の位置」「視点の方向」の２つの要素を用いますが、これは、ハンディカメラを扱うような感覚でカメラを制御することになります。この方式のほかに、テレビカメラのように、ある位置からある位置をとらえるLookAtという方式があります。



LookAt方式とは、カメラが捉える対象物(注視点)をもとにした考え方で、直感的にカメラを扱うための方式です。Direct3D以外でも、さまざまな3Dライブラリで使われています。

LookAt方式では、カメラの制御に「視点の位置」「注視点の位置」「Ｙ軸の傾きを制御するベクトル」の３つの要素を用います。

LookAt方式でビュー変換行列を作成するには、まず視点の位置から注視点の位置を引き、正規化します。これは、カメラの視点方向(ｚ軸)を示すベクトル(ｖ)となります。次に、このベクトルとＹ軸の傾きを制御するベクトルの外積を求めます。すると、カメラの右方向(ｘ軸)を示すベクトル(ｒ)となります。さらに、ｒとｖの外積をとると、カメラの上方向(ｙ軸)を示すベクトルをｕが求まります。これで、カメラの(直交する)３つの軸が求まります。あとは、前述のビュー変換行列に代入し、位置と方向ベクトルの内積を計算するだけです。

ビュー変換行列の作成と設定

Direct3DXは、LookAt方式でビュー変換行列を作成するD3DXMatrixLookAtLH関数を提供していますが、LookAt方式以外でビュー変換行列を作成する関数は提供していません。

```
// ビュー変換行列作成([0.0, 1.0, -5.0]から[0.0, 0.0, 0.0]を注視するカメラ)
D3DXMATRIX   view
D3DXMatrixLookAtLH(&view, &D3DXVECTOR3(0.0f, 1.0f, -5.0f),
                   &D3DXVECTOR3(0.0f, 0.0f, 0.0f), &D3DXVECTOR3(0.0f, 1.0f, 0.0f));
```

ビュー変換行列が作成できたら、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、設定したビュー変換行列が適用されます。

```
// ビュー変換行列設定(pD3DDeviceは初期化済みのDirect3DDevice9オブジェクト)
pD3DDevice->SetTransform(D3DTS_VIEW, &view);
```

ESライブラリでは、カメラを簡単に扱えるように、カメラの機能をCCameraクラスにまとめています。ビュー行列については、ハンディカメラのように座標・位置の設定ができるSetView関数と、LookAt方式で設定できるSetLookAt関数を用意しています。(ソースコードは"Camera.hpp", "Camera.cpp")

射影変換(プロジェクション変換)

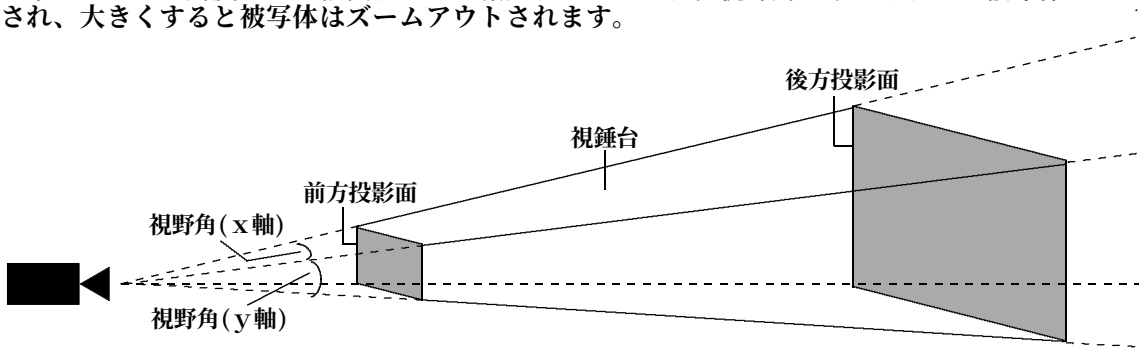
ビュー変換の次に行うのが、「カメラでどのようにとらえるか」という設定です。これを射影変換(プロジェクション変換または投影変換)と呼びます。これは、カメラのレンズを設定する作業といえます。射影変換を行うと、「カメラをとおして見た空間(カメラが捉えた映像)」に変換されます。

射影変換には、おもに透視投影(パースペクティブ射影：Perspective Projection)と正投影(正射影：Orthogonal Projection)が用いられます。

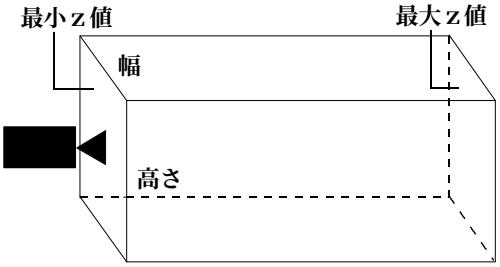
透視投影とは、「パースペクティブ：遠近」という言葉どおり、視点に近ければ近いほど拡大され、遠くのものほど小さく見えるという見え方のことです。

透視投影では、投影面という概念を導入します。投影面とは、スクリーンのように世界を写し取る平面のことです。この投影面を２つ導入します。カメラに近いところと遠いところ、それぞれ前方投影面と後方投影面と呼びます。前方投影面より近い範囲は見えない、後方投影面より遠い範囲も見えない、というようにすると、見える範囲がピラミッドの上部を切り取ったような形に区切られることになります。この形を視錘台と呼びます。視錘台から完全にはずれているオブジェクトは描画されません。このようなオブジェクトのレンダリング処理を省くことにより、描画速度を向上させることができます。

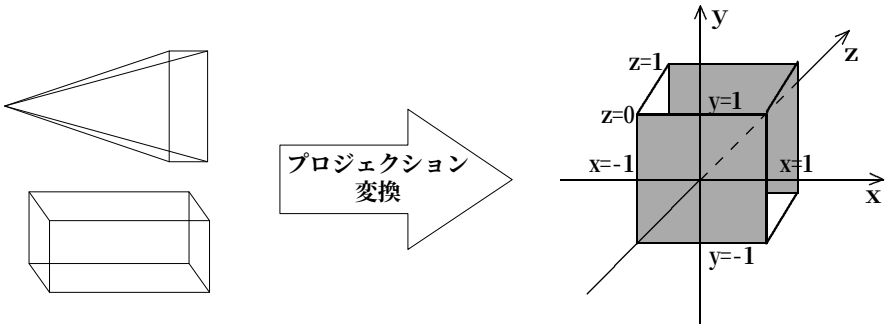
さらに、視野角(Filed Of View)という概念を導入します。これは、見える範囲の角度のことで、視線の中心からどの角度までを描画するかを指定するものです。視野角を小さくすると被写体はズームインされ、大きくすると被写体はズームアウトされます。



正投影とは、オブジェクトを平面的に見るという方法です。遠近感がないので、遠くにあるものが小さく表示されたりしません。正投影は、CADのような図形を正確に表示しなければならない分野で使われています。



射影変換では、カメラがとらえた空間(ビューボリューム)を「X＝－１～＋１」「Y＝－１～＋１」「Z＝０～＋１」の領域に収まるように変換します。この空間を射影空間と呼びます。
透視投影では、ビューボリュームがカメラの原点を頂点とした錐になっているため、前方投影面は引き延ばされ、後方投影面は縮小されます。視点から遠いものほど縮小されるため、遠近感がでます。正投影では、ビューボリュームが立方体なので拡大も縮小も起こりません。



射影変換行列

ビュー座標系に変換された頂点の座標を射影空間の座標(射影座標系)に変換するには、射影変換行列(プロジェクション変換行列)と呼ばれる変換行列を用います。ビュー座標系の頂点に射影変換行列を掛けることにより、カメラを原点とした座標から、実際にカメラから見た座標へ変換されます。

射影変換行列は、投影方法により異なる行列となります。通常、パースペクティブ射影行列の場合は、以下のように視野をもとに作成します。

$$\begin{pmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & zf/(zf-zn) & 1 \\ 0 & 0 & -zn*zf/(zf-zn) & 0 \end{pmatrix}$$

※h = cot(Y軸の視野角 / 2)
w = h / 投影面の横÷縦[＝アスペクト比]
zn = 視点から前方投影面までの距離
zf = 視点から後方投影面までの距離

正射影行列の場合は、以下のように作成します。

$$\begin{pmatrix} 2/w & 0 & 0 & 0 \\ 0 & 2/h & 0 & 0 \\ 0 & 0 & 1/(zf-zn) & 0 \\ 0 & 0 & zn(zn-zf) & 1 \end{pmatrix}$$

※h = ビューボリュームの高さ
w = ビューボリュームの幅
zn = ビューボリュームの最小ｚ値
zf = ビューボリュームの最大ｚ値

射影変換行列の作成と設定

Direct3DXは、視野をもとにパースペクティブ射影行列を作成するD3DXMatrixPerspectiveFovLH関数や正射影行列を作成するD3DXMatrixOrthoLH関数など、いくつかの射影行列作成関数を提供しています。

```
// パースペクティブ射影行列の作成
D3DXMATRIX    projection;
::D3DXMatrixPerspectiveFovLH(&projection, ::D3DXToRadian(45.0f), 4.0f / 3.0f,
                             0.1f, 1000.0f);
```

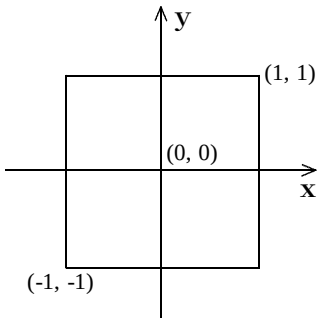
プロジェクション行列が生成できたら、IDirect3DDevice9::SetTransformメソッドを呼び出してシステムに設定します。以後、すべてのオブジェクトのレンダリング時に、設定したプロジェクション変換行列が適用されます。

```
// プロジェクション変換行列設定
pD3DDevice->SetTransform(D3DTS_PROJECTION, &projection);
```

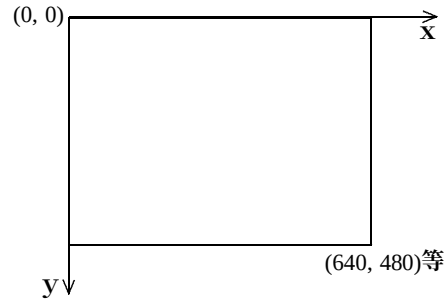
ESライブラリでは、カメラクラス(CCamera)に射影行列を作成し、デバイスに設定するSetProjectionPerspective関数、SetProjectionOrtho関数、SetProjectionOrthoOffCenter関数を用意しています。

スクリーン座標系

3Dグラフィクスを描画する際、最後に行われるのが射影座標系からスクリーン座標系への変換です。スクリーン座標系とは、画面のピクセルに１対１に対応する座標系のことです。射影座標系からスクリーン座標系への変換は、ビューポートスケーリング行列またはビューポート行列と呼ばれる変換行列が用いられます。
射影座標系とスクリーン座標系の違いは、中心の座標です。射影座標系では中央にあり、スクリーン座標系では左上にあるということです。また、ｙ軸の向きが逆になっていることも大きな違いです。



▲射影座標系



▲スクリーン座標系

ビューポートスケーリング行列は、画面の幅をWidth、画面の高さをHeightとしたとき、以下のようになります。

$$\begin{pmatrix} Width/2 & 0 & 0 & 0 \\ 0 & -Height/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Width/2 & Height/2 & 0 & 1 \end{pmatrix}$$

Direct3Dでは、ビューポートスケーリング行列は、IDirect3DDevice9::SetViewportメソッドでビューポート(レンダリング結果を対象面のどこに描画するのか)の設定を行うことにより自動的に生成され、デバイスに適用されます。

最後に、モデル座標系からスクリーン座標系への変換をまとめると、

スクリーン座標 ＝ モデル座標 × ワールド行列 × ビュー行列 × 射影行列 × ビューポート行列となります。

課　　題

カメラを設定し、モデルが正しく描画されるようにしましょう。

(１)カメラの初期設定を行いましょう。

カメラの初期設定を行う前に、画面(ビューポート)の情報を取得します。以下のプログラムをInitialize関数に追加しましょう。

```
// ビューポート情報取得
Viewport   view = GraphicsDevice.GetViewport();
```

(２)カメラの初期設定を行います。

カメラのビュー行列はSetLookAt関数、射影行列はSetPerspectiveFieldOfView関数で行います。カメラが座標(0.0, 2.0, -5.0)から(0.0, 0.0, 1.0)を注視するように、以下のプログラムを完成させInitialize関数に追加しましょう。

```
// カメラ設定
Camera->SetLookAt(Vector3(※ここは各自考えましょう※),  // 座標
                  Vector3(※ここは各自考えましょう※),  // 注視点
                  Vector3_Up);                        // 上方ベクトル
Camera->SetPerspectiveFieldOfView(45.0f, (float)view.Width, (float)view.Height,
                                  1.0f, 1000.0f);
```