３次元座標変換のメモ書き

mebiusbox software

2019 年 6 月 2 日

目次

1．はじめに

入社してきた新入社員が期待と不安とやる気に満ちあふれているところで，まずは研修から始めるところが多いかと思います．研修なんて受けたことがないので，きちんと基礎を叩き込んでくれるところで研修受けたい．そう思うこの頃です．
今回は３次元座標変換について，巷にはそういった情報は沢山あるのですが，自分なりの整理も含めて少し書くことにしました．新人向けに書いていますので，最初は基本的なところから入っていきます．

2．基本的な座標変換

2.1 座標系（coordinate system）

３次元空間は実世界と同じであり，上下・左右・前後の３つの次元です．この上下・左右・前後はそれぞれ直交関係にあります．そして，例えば上下というのは，私から見た上下と，あなたから見た上下は必ずしも一致するとは限りません．つまり，上下・左右・前後というのはどこか基点があるということです．この場合，私やあなたが基点であり，向いている方向に依存しています．次に，例えば前方に５歩進むとしたら人によって進む距離に違いがあります．もし，前方に１メートル進む場合は全員同じ距離になりますが，今度は１メートル進むための歩数が変わってきます．このように，単位も重要になってきます．よって，基点と向きと単位を決めれば，その３次元空間での座標が特定できることになります．この基点を原点（origin）とし，向き（axis），単位（unit）を決めたものを**座標系**といいます．ここでは上下を Y，左右を X，前後を Z とします．
３次元空間では座標系を決めるときに２つの種類があります．**左手座標系**と**右手座標系**です．左手座標系では Z 方向が奥に向かって正であり，DirectX や UnrealEngine，Unity などが左手座標系です．右手座標系は Z 方向が手前に向かって正であり，OpenGL や WebGL などが右手座標系です．座標系が同じだとしても，上下方向が Y や Z だったりするので，座標系と軸を確認するようにしましょう．
ここでは右手座標系で説明します．

2.2 ベクトル（vector）

座標系が決まれば，位置を表現することができます．３次元空間での位置はベクトルで表されます．まず，点 p の位置を XYZ の各軸で測った値をそれぞれ x,y,z とすると

$$p = (x, y, z)$$

で位置を表現できます．ベクトルは方向と大きさを表していて，例えば点 q から点 p に向かうベクトル v は

$$q = (0,30,0), \quad p = (0,50,0), \quad \vec{v} = \vec{qp} = (0,20,0)$$

となります．これにはベクトルの始点，終点の位置，大きさ，方向がありますね．ここで，ベクトル $\vec{v}$ の始点を原点 $O = (0,0,0)$ とすれば，

$$\vec{OV} = \vec{v} = (0,20,0)$$

となって，大きさと方向のみを表すことになります．このときの x, y, z のことをそれぞれ X 成分，Y 成分，Z 成分といい，この表記を**成分表示**といいます．
ベクトルの成分表示が出来たので，ベクトルから大きさと方向をそれぞれ取り出してみましょう．まず，大きさですがこれは始点から終点までの距離になります．始点を原点とすればピタゴラスの定理を使ってベクトルの大きさ $|\vec{v}|$ は

$$|\vec{v}| = \sqrt{x^2 + y^2 + z^2}$$

で得られます．３次元ベクトルが３つの値を持っていることに対して，ベクトルの大きさは１つの量を表す値です．この値をベクトルに対して**スカラ**（scalar）といいます．

次に方向は，大きさが１のベクトルで表します．このようなベクトルを**単位ベクトル**（unit vector）といいます．単位ベクトルはベクトルの各成分を大きさで割ることで求められます．

$$\frac{\vec{v}}{|\vec{v}|} = \left(\frac{x}{|\vec{v}|}, \frac{y}{|\vec{v}|}, \frac{z}{|\vec{v}|}\right)$$

これを**正規化**（normalize）といいます．つまり，ベクトルを正規化すればそれは方向を表すベクトルということになります．

※ベクトルや行列の演算について，これから出てきますが詳しいことはここでは解説しません．ベクトルや行列などの線形代数については他を参照したり，もしくは私が書いた「CG のための線形代数入門シリーズ」を参照してください．

2.3 基底（basis）

ベクトルの成分表示で XYZ で測った値を表記することがわかりました．もちろん，この XYZ というのはそのベクトルを測った座標系に基づいています．この座標系とベクトルの関係を見ていきましょう．

XYZ というのは座標系における軸のことです．これまで，上下・左右・前後という曖昧な表現でしたが，方向を表すベクトル，つまり単位ベクトルを使うと，XYZ の向きをベクトルで表すことが出来ます．一般的な座標系として，左右を X，上下を Y，前後を Z としたとき，それぞれの向き（単位ベクトル）$\vec{e}_x, \vec{e}_y, \vec{e}_z$ は

$$\vec{e}_x = (1,0,0), \quad \vec{e}_y = (0,1,0), \quad \vec{e}_z = (0,0,1)$$

となります．これを使うとベクトルの成分表示は

$$\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = p_x \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + p_y \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + p_z \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = p_x\vec{e}_x + p_y\vec{e}_y + p_z\vec{e}_z$$

となります．この $\vec{e}_x, \vec{e}_y, \vec{e}_z$ はお互い直交関係になっています．ベクトルの内積が０のとき，直交であることから

$$\vec{e}_x \cdot \vec{e}_y = \vec{e}_y \cdot \vec{e}_z = \vec{e}_z \cdot \vec{e}_x = 0$$

の関係であることがわかります．これらの単位ベクトルによって座標系の向きを決めることができます．この XYZ の向きベクトルの組を**基底**といい，それぞれのベクトルのことを**基底ベクトル**といいます．ベクトルの内積は片方が単位ベクトルのとき，もう片方のベクトルのその単位ベクトル方向の大きさを求められます．これを使って，ある座標系の基底ベクトルと，位置を表すベクトルから

$$p_x = \vec{p} \cdot \vec{e}_x$$

$$p_y = \vec{p} \cdot \vec{e}_y$$

$$p_z = \vec{p} \cdot \vec{e}_z$$

と各成分が得られます．さらに行列を使って，$\vec{e}_x = i, \vec{e}_y = j, \vec{e}_z = k$ とおくと

$$M = \begin{pmatrix} i_x & i_y & i_z \\ j_x & j_y & j_z \\ k_x & k_y & k_z \end{pmatrix}$$

$$\vec{p}' = M\vec{p}$$

$$= \vec{p}\cdot i + \vec{p}\cdot j + \vec{p}\cdot k$$

となります．この行列 M は XYZ の基底ベクトルで構成されていることから，基底を表していることが想像できると思います．行列とベクトルの内積の性質から，基底を表す行列 M に，任意のベクトル $\vec{p}$ をかけると（一次変換），行列 M で表す基底での成分に変換することができ，この値は一意に決まります．よって，この行列 M はベクトル $\vec{p}$ から $\vec{p}'$ の写像を表しています．
基底ベクトルというのは常に単位ベクトルというわけでもなく，また，３次元での基底は直交していなくても定義することができます．このような座標系は斜交座標系といったりします．ここでは，右手座標系ですので，基底ベクトルは直交関係であり，また，基底ベクトルが単位ベクトルであるとします．このような基底を**正規直交基底**といいます．

2.4 基底変換（basis transformation）

いよいよ座標変換に入っていきます．座標変換といっても色々種類がありますので，１つずつ見ていきましょう．まずは基底変換です．基底というのはすでに見てきたように，基底ベクトルの組であり，基底ベクトルは単位ベクトルで互いに直交になっています．座標系で出てきた「原点」「向き」「単位」のうち，「原点」は $O = (0,0,0)$ のままでどの基底でも同じです．また，単位も基底ベクトルは常に単位ベクトルですので，単位も変わりません．結局，基底は向きのみを表しているといえます．そうすると基底が変わるということはどういうことでしょうか．原点が固定で，大きさも変わらないとなれば，それは向きが「回転」すると考えられます．つまり，基底変換は回転させる座標変換といえます．
ここで少し三角関数について復習しておきましょう．まずは三角比から．

$$\sin\theta = \frac{a}{c}, \quad \cos\theta = \frac{b}{c}, \quad \tan\theta = \frac{a}{b}$$

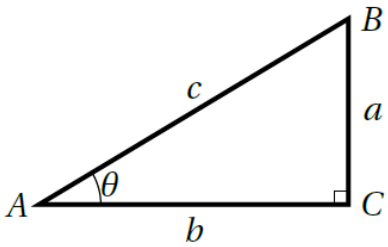

Fig.1: 三角比

単位円で三角関数を見てみると

$$x = \cos\theta, \quad y = \sin\theta$$

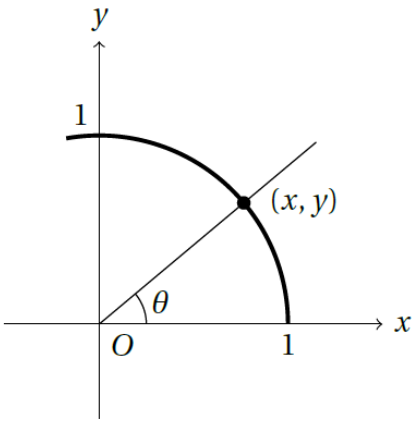

Fig.2: 単位円の三角関数

となります．また，ピタゴラスの定理から

$$\sin\theta^2 + \cos\theta^2 = 1$$

が得られます．三角関数の逆関数にはそれぞれ arcsin, arccos, arctan があります．これらの関係は

$$\theta = \arcsin(\sin\theta) \quad \left(-\frac{\pi}{2} \leqq \theta \leqq \frac{\pi}{2}\right),$$

$$\theta = \arccos(\cos\theta) \quad \left(0 \leqq \theta \leqq \pi\right),$$

$$\theta = \arctan(\tan\theta) \quad \left(-\frac{\pi}{2} < \theta < \frac{\pi}{2}\right)$$

となります．次に加法定理です．次の図を見てください．

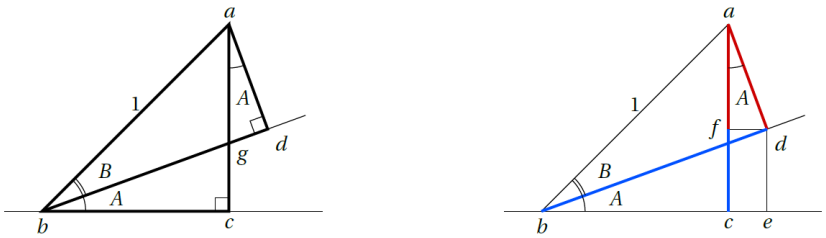

Fig.3: 加法定理

Δabc において，斜辺の長さが 1 から

$$\overline{ac} = \sin(A+B), \quad \overline{bc} = \cos(A+B)$$

Δabd において，

$$\overline{ad} = \sin B, \quad \overline{bd} = \cos B$$

次に

$$\overline{af} = \overline{ad}\cos A = \cos A \sin B$$

$$\overline{fc} = \overline{de} = \overline{bd}\sin A = \sin A \cos B$$

$\overline{ac} = \overline{af} + \overline{fc}$ の関係から

$$\sin(A+B) = \sin A\cos B + \cos A\sin B$$

$$\cos(A+B) = \cos A\cos B - \sin A\sin B$$

$$\sin(A-B) = \sin A\cos B - \cos A\sin B$$

$$\cos(A-B) = \cos A\cos B + \sin A\sin B$$

となります．

それでは回転を見ていきましょう．次の図のように点 p を点 p' に回転する場合を考えます．ここでは $z=0$ とします．

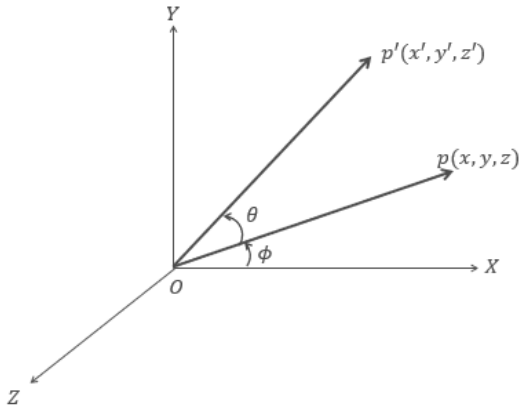

Fig.4: 点の回転

すると点 p の各成分は

$$x = r\cos\phi, \quad y = r\sin\phi$$

となります．回転後の点 p' の各成分は加法定理を使って

$$x' = r\cos(\theta+\phi) = r\cos\theta\cos\phi - r\sin\theta\sin\phi$$

$$y' = r\sin(\theta+\phi) = r\sin\theta\cos\phi + r\cos\theta\sin\phi$$

となります．ここで

$$r = \frac{x}{\cos\phi}, \quad r = \frac{y}{\sin\phi}$$

を代入して，整理すると

$$x' = \frac{x\cos\theta\cos\phi}{\cos\phi} - \frac{y\sin\theta\sin\phi}{\sin\phi} = x\cos\theta - y\sin\theta$$

$$y' = \frac{x\sin\theta\cos\phi}{\cos\phi} + \frac{y\cos\theta\sin\phi}{\sin\phi} = x\sin\theta + y\cos\theta$$

$$z' = z$$

となって，行列で書くと

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

となります．この行列の基底ベクトルに注目してみると，回転させたい方向とは逆の方向に回転したベクトルになっています．つまり，反対方向に回転させた基底を使って変換を行うと，回転した座標になるということになります．

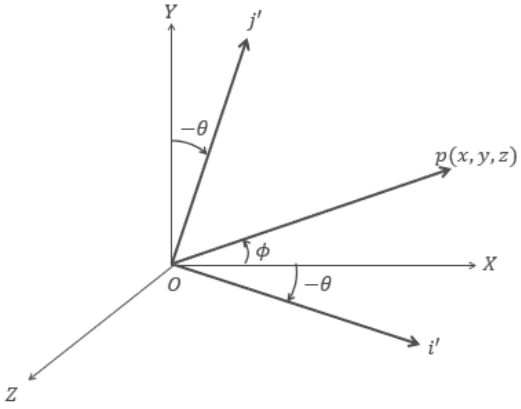

Fig.5: 逆方向に回転

この回転行列は Z 軸を基準に回転を表していることになります．この Z 軸回転の行列を R_z とおくと

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

となります．同じように X 軸，Y 軸の回転行列 R_x, R_y は

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

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

となります．ここで各回転行列の行ベクトル，および列ベクトルは

$$\sin\theta^2 + \cos\theta^2 = 1$$

から，すべて単位ベクトルであり，列ベクトルにおいても互いに内積がゼロなので直交していることがわかります．
行列は掛け合わせて合成できるので，例えば XYZ 回転の行列 R_{xyz} は

$$R_{xyz} = R_z R_y R_x$$

となります．ここでは取り上げませんが，任意の軸を基準に回転する行列の式も簡単に見つかると思います．これらの回転行列を組み合わせて基底変換の行列をつくることが出来ます．

2.5 剛体変換（rigid-body transformation）

基底変換では座標系のうち，向きしか変換できませんでした．ここで，原点を平行移動させる変換を考えます．このような大きさが変わらず，向きと位置が変わる変換を**剛体変換**といいます．剛体（rigid body）は力を加えても形状を変えない物体のことで物理シミュレーションでよく使います．基底変換では向きしか変換できなかったので，これに加えて平行移動する変換をしなければなりません．基底変換ではある点 p を点 p' に変換する行列 M から

$$\vec{p}' = M\vec{p}$$

と表せました．ここで，この変換に $\vec{t}$ の平行移動を追加するには

$$\vec{p}' = M\vec{p} + \vec{t}$$

とすればよいことになります．これは１次関数の形になっていますね．しかし，このままでは３行３列の行列で表すことができず，扱いづらいものになっています．そこで，４次元の**同次座標系**（homogeneous coordinate）を使います（同次座標系は斉次座標系とも呼ばれます）．４次元の同次座標系を使うと４次元空間の断面と３次元空間の座標を同じ（写像）扱いにすることができます．４次元空間の位置は w が追加された x, y, z, w で表し，３次元空間との関係は

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

となります．通常は $w = 1$ として

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

とします．ちなみに $w = 0$ とすると，無限遠点となり大きさが無限となります．ただし，無限遠点でも向きは存在しているので，$w = 0$ の場合は向きだけを表しているとも考えることができます．

４次元の同次座標系を使うことで４行４列の行列を扱えるようになりました．平行移動を行列で表記すると

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

となります．基底変換の行列は回転を表す行列なので，回転行列を R とおくと，剛体変換は次のようになります．

$$\vec{p}' = M\vec{p} + \vec{t} = TR\vec{p} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

2.6 アフィン変換（affine transformation）

基底変換の式

$$\vec{p}' = M\vec{p}$$

は一次変換でした．次に，剛体変換の式は基底変換に平行移動を追加した式

$$\vec{p}' = M\vec{p} + \vec{t}$$

は１次関数と同じ形になっていましたね．もちろん，これは写像を表しています．この１次関数をベクトル空間で一般化し，一次変換と平行移動を合わせた変換を**アフィン変換**といいます．よって，剛体変換も基底変換もアフィン変換の特殊な形となります．アフィン変換では回転，平行移動の他に，拡大・縮小，鏡映，せん断があります．

先に鏡映とせん断についてです．鏡映（reflection）は基準となる軸に対して対称となる位置に移動する変換です．例えば，右手座標系と左手座標系を相互に変換する場合は次のような行列になります．

$$M_{reflection} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

せん断（shearing）はスキュー（skew）変換とも呼ばれ，歪ませる変換です．これは別軸の成分の値が他の軸の成分に影響させることで歪ませます．例えば，Y 軸方向の値分，X 軸方向にずらす場合は

$$M_{skew} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

となります．

次は拡大・縮小です．この行列 S は次のようになります．

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

拡大・縮小行列で変換してみると

$$\vec{p}' = S\vec{p} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \end{pmatrix}$$

となります．この拡大・縮小と回転，平行移動の３つを組み合わせて座標変換行列をつくるのが基本的なやり方となります．式で表すと

$$\vec{p}' = TRS\vec{p}$$

$$= \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x R_{11} & s_y R_{12} & s_z R_{13} & t_x \\ s_x R_{21} & s_y R_{22} & s_z R_{23} & t_y \\ s_x R_{31} & s_y R_{32} & s_z R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

となります．また，TRS の変換行列において４行目のベクトルは固定値になっているため

$$TRS = \begin{pmatrix} s_x R_{11} & s_y R_{12} & s_z R_{13} & t_x \\ s_x R_{21} & s_y R_{22} & s_z R_{23} & t_y \\ s_x R_{31} & s_y R_{32} & s_z R_{33} & t_z \end{pmatrix}$$

と３行４列の部分だけあればよいことになります．これは変換行列のデータ量を小さくするときに利用します．

TRS の変換行列が得られたので，今度は TRS から拡大・縮小，回転，平行移動に分解することを考えてみましょう．まず，平行移動は t_x, t_y, t_z をそのまま取り出すことができます．問題は拡大・縮小と回転です．これを抽出するためには回転行列の性質を利用します．

ここからは少しややこしくなりますが，１つ１つ整理しながら読み進んでもらえるといいかなと思います．

基底変換のところで回転行列について扱いました．θ 回転させる場合，反対方向に回転，つまり $-\theta$ 回転した基底を使えば θ 回転することになるということでしたね．さらにいうと，回転した座標を回転した基底で変換すると，回転前と同じ座標になります．ここで注意なのが，回転した基底は回転行列とは異なるということです．基底を表す行列 M，点を p として，回転前の行列 M_1，回転前の点 p_1，回転後の行列 M_2，回転後の点 p_2 とすれば

$$M_1 p_1 = M_2 p_2$$

の関係が成り立ちます．また，逆行列を使えば

$$p_1 = M_1^{-1} M_2 p_2, \quad p_2 = M_2^{-1} M_1 p_1$$

となります．θ 回転する基底を作ると，それは $-\theta$ 回転した基底ベクトルということでした．この行列の逆行列を考えてみると，θ 回転した基底ベクトルになっているということになります．これはまさに回転後の基底です．ここまでをまとめると，

- 基底を表す行列の逆行列は，回転後の基底である
- 基底を表す行列は，回転後の基底における回転前の基底である

となります．次に回転行列は**直交行列**の性質を持っています．直交行列はその転置行列を自身にかけると単位行列となるものです．つまり，直交行列の転置行列は逆行列ということになります．

$$M^T M = M M^T = I, \quad M M^{-1} = M^{-1} M = I, \quad \therefore M^T = M^{-1}$$

また，直交行列の行列式の性質も確認してみましょう．行列 A, B の行列式の関係は

$$|AB| = |A||B|$$

です．これに $A = M, B = M^T$ を代入すると

$$|M M^T| = |M||M^T|$$

行列式の性質として行列とその行列の転置行列の行列式は同じです．

$$|M| = |M^T|$$

よって

$$|M M^T| = |M||M^T| = |M||M| = |M|^2$$

となります．一方で

$$M M^T = I$$

および，単位行列 I の行列式は $|I| = 1$ ということがわかっているので

$$|M M^T| = |I| = 1$$

となり，

$$|M M^T| = |I| = |M|^2 = 1$$

となります．よって，二乗して 1 となる値が直交行列の行列式となるので 1 と −1 ということになります．また，回転行列のように行ベクトル，列ベクトルがすべて単位ベクトルであり，互いに直交である場合は行列式の値が 1 になります．
これまで直交行列の性質をみてきました．回転行列は直交行列ですので，直交行列の性質から回転行列の逆行列は転置行列と等しくなります．このことを踏まえて

- 基底を表す行列の逆行列は，回転後の基底である

を考えてみると，逆行列は転置行列だったので，基底を表す行列の列ベクトルが回転後の基底ベクトルとなります．結局

- 基底を表す行列の列ベクトルは，回転後の基底ベクトルである
- 基底を表す行列は，回転後の基底における回転前の基底である

となります．これで TRS の座標変換行列から回転と拡大・縮小を抽出する準備ができました．もう一度変換行列を確認してみると

$$\vec{p}' = TRS\vec{p} = \begin{pmatrix} s_xR_{11} & s_yR_{12} & s_zR_{13} & t_x \\ s_xR_{21} & s_yR_{22} & s_zR_{23} & t_y \\ s_xR_{31} & s_yR_{32} & s_zR_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

ここで列ベクトルに注目し，それぞれ $\vec{v}_x, \vec{v}_y, \vec{v}_z$ とおくと

$$\vec{v}_x = \begin{pmatrix} s_xR_{11} \\ s_xR_{21} \\ s_xR_{31} \end{pmatrix}, \quad \vec{v}_y = \begin{pmatrix} s_yR_{12} \\ s_yR_{22} \\ s_yR_{32} \end{pmatrix}, \quad \vec{v}_z = \begin{pmatrix} s_zR_{13} \\ s_zR_{23} \\ s_zR_{33} \end{pmatrix}$$

となります．各ベクトルの大きさを求めると

$$|\vec{v}_x| = \sqrt{(s_xR_{11})^2 + (s_xR_{21})^2 + (s_xR_{31})^2}$$

$$|\vec{v}_y| = \sqrt{(s_yR_{12})^2 + (s_yR_{22})^2 + (s_yR_{32})^2}$$

$$|\vec{v}_z| = \sqrt{(s_zR_{13})^2 + (s_zR_{23})^2 + (s_zR_{33})^2}$$

となります．さらに二乗して平方根をはずして整理すると

$$|\vec{v}_x|^2 = s_x^2R_{11}^2 + s_x^2R_{21}^2 + s_x^2R_{31}^2 = s_x^2(R_{11}^2 + R_{21}^2 + R_{31}^2)$$

$$|\vec{v}_y|^2 = s_y^2R_{12}^2 + s_y^2R_{22}^2 + s_y^2R_{32}^2 = s_y^2(R_{12}^2 + R_{22}^2 + R_{32}^2)$$

$$|\vec{v}_z|^2 = s_z^2R_{13}^2 + s_z^2R_{23}^2 + s_z^2R_{33}^2 = s_z^2(R_{13}^2 + R_{23}^2 + R_{33}^2)$$

となります．ここで回転行列の列ベクトルは回転後の基底ベクトルであり，単位ベクトルなので，

$$\sqrt{R_{11}^2 + R_{21}^2 + R_{31}^2} = 1 \quad \therefore R_{11}^2 + R_{21}^2 + R_{31}^2 = 1^2 = 1$$

$$\sqrt{R_{12}^2 + R_{22}^2 + R_{32}^2} = 1 \quad \therefore R_{12}^2 + R_{22}^2 + R_{32}^2 = 1^2 = 1$$

$$\sqrt{R_{13}^2 + R_{23}^2 + R_{33}^2} = 1 \quad \therefore R_{13}^2 + R_{23}^2 + R_{33}^2 = 1^2 = 1$$

となります．これを代入すると

$$|\vec{v}_x|^2 = s_x^2, \quad |\vec{v}_y|^2 = s_y^2, \quad |\vec{v}_z|^2 = s_z^2 \qquad \therefore |\vec{v}_x| = s_x, \quad |\vec{v}_y| = s_y, \quad |\vec{v}_z| = s_z$$

となって，列ベクトルの大きさが拡大・縮小の値と一致することがわかります．s_x, s_y, s_z がわかれば，回転は

$$\frac{1}{s_x}\vec{v}_x, \quad \frac{1}{s_y}\vec{v}_y, \quad \frac{1}{s_z}\vec{v}_z$$

で各成分が得られます．まとめると，TRS の合成行列を M とおくと

$$\vec{p}' = TRS\vec{p} = \begin{pmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

これから TRS を求めると

$$\vec{t} = \begin{pmatrix} M_{14} \\ M_{24} \\ M_{34} \end{pmatrix}, \quad \vec{s} = \begin{pmatrix} \sqrt{M_{11}^2 + M_{21}^2 + M_{31}^2} \\ \sqrt{M_{12}^2 + M_{22}^2 + M_{32}^2} \\ \sqrt{M_{13}^2 + M_{23}^2 + M_{33}^2} \end{pmatrix}, \quad R = \begin{pmatrix} M_{11}/s_x & M_{12}/s_y & M_{13}/s_z \\ M_{21}/s_x & M_{22}/s_y & M_{23}/s_z \\ M_{31}/s_x & M_{32}/s_y & M_{33}/s_z \end{pmatrix}$$

となります．

3．レンダリングパイプライン

コンピュータグラフィックスでは，レンダリングするために入力データ（頂点データやテクスチャなど）から画面に表示するまでの工程があり，その工程順番を決めたものをレンダリングパイプラインといいます．座標変換はそのレンダリングパイプラインの工程で行われます．具体的にはモデル変換，ビュー変換，投影変換，ビューポート変換です．

3.1 モデル変換（modeling transformation）

座標変換ができれば座標系がいくつもあっても構わないのですが，一般的に**ローカル座標系**（local coordinate system）と**ワールド座標系**（world coordinate system）が使われます．頂点データはローカル座標系で作成し，それをワールド座標系に変換します．このワールド座標系への変換を**モデル変換**といいます．頂点データはローカル座標系で作成しますが，普通は親子階層を持っていて，親の座標系の下にさらに子の座標系，さらにその子の座標系・・・というようになっています．座標変換は一方向に伝搬するので，ワールド←親←子←孫・・・と座標変換できます．逆行列を使えば逆方向に伝搬することも出来ますね．モデル変換や親子での座標変換も基本的に平行移動・回転・拡大・縮小の TRS で座標変換されます．

3.2 ビュー変換（viewing transformation）

レンダリングではカメラを導入して，カメラから見えるシーンを描画します．モデル変換によってワールド座標系にすべて変換されますので，ワールド空間上にカメラを配置します．そして，ワールド座標系の座標をカメラから見た座標系（カメラ座標系またはビュー座標系）に変換します．これを**ビュー変換**といいます．ビュー変換には一般的に 2 種類あります．

3.2.1 LookAt 方式

まず，LookAt 方式です．これはカメラの位置（C）と対象の位置（P），カメラの傾きを表す方向（U）の 3 つで指定します．まず，カメラの位置が原点となりますので，平行移動が必要です．

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

次に，カメラから見て前後方向（Z 軸）を決定します．それは対象の位置からカメラの位置に向かうベクトルです．

$$\vec{z} = \frac{\vec{C} - \vec{P}}{|\vec{C} - \vec{P}|}$$

次に，カメラの上方向を表すベクトルと求めた Z 方向との外積を求めます．これは X 方向になります．

$$\vec{x} = \frac{\vec{U} \times \vec{z}}{|\vec{U} \times \vec{z}|}$$

最後に，Z 方向と X 方向の外積を求めます．これは Y 方向になります．

$$\vec{y} = \frac{\vec{z} \times \vec{x}}{|\vec{z} \times \vec{x}|}$$

これで基底ベクトルが揃ったので，行列にすると

$$M = \begin{pmatrix} x_x & x_y & x_z & 0 \\ y_x & y_y & y_z & 0 \\ z_x & z_y & z_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x_x & x_y & x_z & -\vec{x} \cdot \vec{t} \\ y_x & y_y & y_z & -\vec{y} \cdot \vec{t} \\ z_x & z_y & z_z & -\vec{z} \cdot \vec{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

となります．

3.2.2 位置と回転を指定する方式

カメラの位置と回転を表す行列をそれぞれ T, R とすれば，

$$\vec{p}' = TR\vec{p}$$

はカメラ座標系からワールド座標系に変換する式です．ここで逆行列を使えば

$$\vec{p} = R^{-1}T^{-1}\vec{p}' = (RT)^{-1}\vec{p}'$$

となって，ワールド座標系からカメラ座標系に変換する式となります．さらに，回転行列の逆行列は転置行列，平行移動の逆行列は符号を反転したもの

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

ですから，

$$R^{-1}T^{-1} = \begin{pmatrix} R_{11} & R_{21} & R_{31} & 0 \\ R_{12} & R_{22} & R_{32} & 0 \\ R_{13} & R_{23} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

と書くことも出来ます．

3.3 投影変換（projective transformation）

投影変換はとても重要な変換であり，ここは環境に依存する部分でもあります．投影変換は射影変換とも言われます．投影変換には近くのものが大きく，遠いものが小さくなる**透視投影変換**（perspective projection）と，奥行きによって変換しない**正投影変換**（orthogonal projection）があります．

投影変換の手順としては，まず，カメラから見える範囲を決めます．具体的には**前方クリッピング面**（near clipping plane）と**後方クリッピング面**（far clipping plane）というのを導入し，その空間内に入っているものだけが投影されるようにします．この領域を**ビューボリューム**（view volume）といい，透視投影変換では視錐台（view frustum），正投影変換では長方体の形になります．

次に，このビューボリュームの後方クリッピング面が $z=1$ で，このときのビューボリュームの x, y が $-1 \leq x \leq 1, -1 \leq y \leq 1$ となるように正規化します．このビューボリュームを**正規化ビューボリューム**といいます．

この時点でビューボリュームも正規化ビューボリュームもまだカメラ空間にありますが，正規化ビューボリュームを $-1 \leq z \leq 1$ の空間に変換します．この空間を**クリップ空間**（clip space）といい，この空間でのビューボリュームは**標準視体積**（canonical view volume）といいます．このとき，Z は奥の方向が正となって左手座標系になります．このクリップ空間は 4 次元の同次座標系になっていて，x, y, z, w で表されます．投影変換はカメラ空間からクリップ空間に変換することをいいます．

> DirectX などは Z の範囲が [0,1] になっているので注意が必要です．

クリップ空間の座標で x, y, z を w で除算することを**透視除算**といい，$x/w, y/w, z/w$ となります．この透視除算した座標を**正規化デバイス座標**（normalized device coordinate）といいます．

3.3.1 透視投影変換

まずは透視投影変換から見ていきましょう．透視投影では近いほど大きく，遠いほど小さく見えます．次の図を見てください．

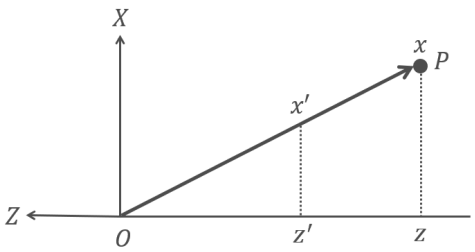

Fig.6: 透視投影

原点 O の位置から点 P を見たとき，点 P を $z'=-1$ の平面に投影すると，三角形 xoz と $x'oz'$ は相似の関係なので

$$x' : x = z' : z = -1 : z, \quad \therefore x' = \frac{x}{-z}$$

となります．これは Y についても同じです．よって

$$x' = \frac{x}{-z}, \quad y' = \frac{y}{-z}$$

となります．次に投影面というのを考えます．これは先程の投影される平面のことで，$z'=-1$ です．

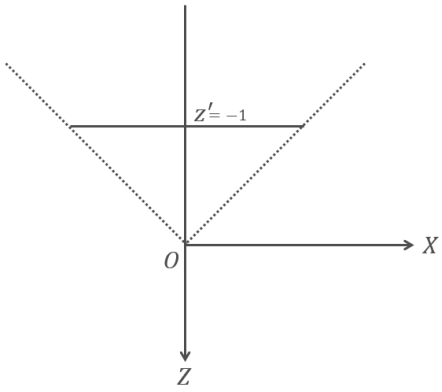

Fig.7: 投影面

また視野角を 90° とします．そうすると z' における X と Y は $[-1, 1]$ になります．

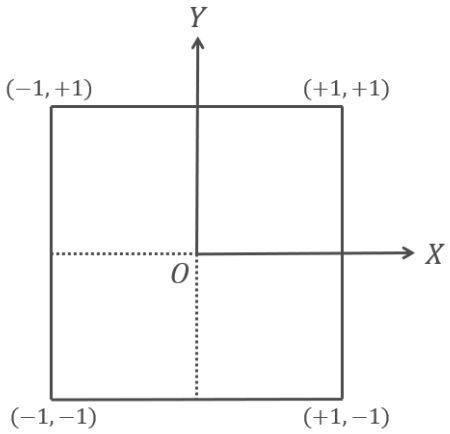

Fig.8: 視野角

ここで，最初の透視変換の式と，投影面を考慮して行列で表すと次のようになります．

$$M_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

この行列で頂点 p を座標変換すると

$$p' = M_p \cdot p = M_p \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -z \end{pmatrix}$$

となります．p'_w で p'_{xyz} の各成分を除算すれば

$$\begin{pmatrix} x/-z \\ y/-z \\ -1 \end{pmatrix}$$

となります．次に正規化ビューボリュームの範囲 $[-1, 1]$ に値を変換する必要があります．まずは Z 値からですが，次のような 1 次関数を考えます．

$$p'_z = az + b$$

この行列は次のようになります．

$$M_w = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad M_w \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ za + b \\ -z \end{pmatrix}$$

クリップ座標系は同次座標系ですので，w で各成分を割ることになります．透視投影ではこれを透視除算といい，透視除算した座標を正規化デバイス座標といいます．ここで

$$p'_w = -z$$

ですから，

$$p'_z = az + b = -a \cdot p'_w + b$$

という関係になります．また，正規化デバイス座標の z_{ndc} は

$$z_{ndc} = \frac{p'_z}{p'_w} = \frac{-a \cdot p'_w + b}{p'_w} = -a + \frac{b}{p'_w} = -a + \frac{b}{-z}$$

となります．ここで，前方クリッピング面と後方クリッピング面を導入します．

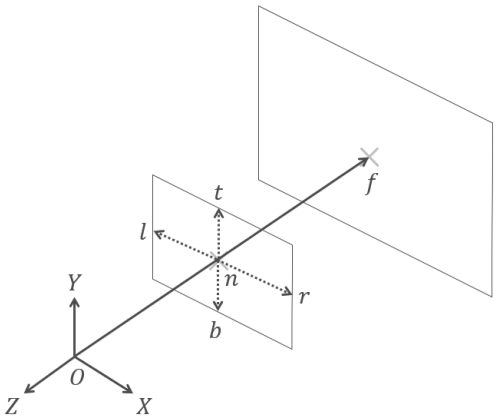

Fig.9: 視野角錐

この前方クリッピング面（$z = n$）のとき $z_{ndc} = -1$，後方クリッピング面（$z = f$）のとき $z_{ndc} = 1$ となるように，a と b を求めます．

$$-a + \frac{b}{-n} = -1, \quad -a + \frac{b}{-f} = 1, \quad \therefore b = -f(1 + a)$$

……

$$-a+\frac{b}{-n}=-1$$

$$a=\frac{b}{-n}+1=\frac{-f(1+a)}{-n}+1=\frac{-f-af-n}{-n}=\frac{f+af+n}{n}$$

$$an=f+af+n$$

$$an-af=a(n-f)=f+n$$

$$a=\frac{n+f}{n-f}$$

$$b=-f(1+a)=-f-af$$

$$=-f-f\left(\frac{n+f}{n-f}\right)$$

$$=-\frac{nf-f^2}{n-f}-\frac{nf+f^2}{n-f}$$

$$=\frac{nf+f^2-nf-f^2}{n-f}$$

$$=\frac{-2nf}{n-f}$$

よって

$$\therefore a=\frac{n+f}{n-f},\quad b=\frac{-2nf}{n-f}$$

また,

$$p'_z=az+b=\frac{n+f}{n-f}z+\frac{-2nf}{n-f}$$

行列に書き直せば

$$M_w=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&\frac{n+f}{n-f}&\frac{-2nf}{n-f}\\0&0&-1&0\end{pmatrix}\quad M_w\begin{pmatrix}x\\y\\z\\1\end{pmatrix}=\begin{pmatrix}x\\y\\z\frac{n+f}{n-f}+\frac{-2nf}{n-f}\\-z\end{pmatrix}$$

となります．ここで，$n=-1, f=-5$ としたときの図を以下に示します.

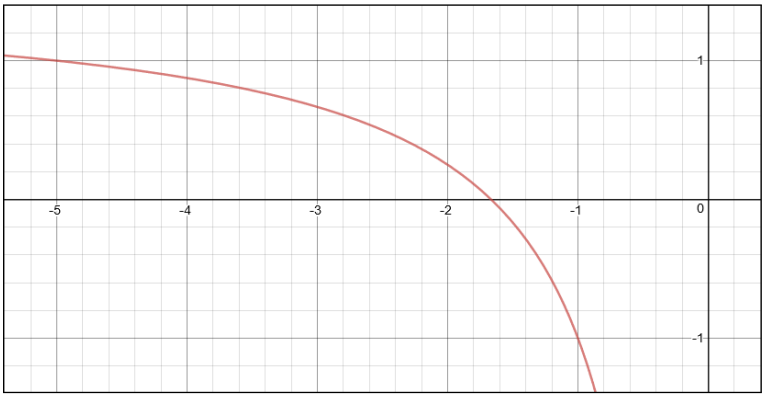

Fig.10: 透視投影のクリップ空間の Z

この図では横軸が z，縦軸が z_{ndc} です．$z=-1$ のとき $z_{ndc}=-1$，$z=-5$ のとき $z_{ndc}=1$ になっているのがわかります．

次に l,r,t,b を $[-1,1]$ に収まるように拡大・縮小します．$z=-1$ のとき $-1,1$，$z=-n$ のとき l,r または t,b ですから，相似の関係から

$$\frac{1-(-1)}{-1}/\frac{r-l}{n}=\frac{-2n}{r-l},\quad \frac{1-(-1)}{-1}/\frac{t-b}{n}=\frac{-2n}{t-b}$$

となります．これを行列表記すると

$$M_s=\begin{pmatrix}\frac{-2n}{r-l} & 0 & 0 & 0\\ 0 & \frac{-2n}{t-b} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

となります．あとは，これまで Z 軸がクリッピング面の中心を通っていることを想定していました．これがずれている場合に対処します．そのためにせん断を使います．前方クリッピング面での中心位置分を Z 軸の値によって変化させてずらします．この行列は次のようになります．

$$M_{sh}=\begin{pmatrix}1 & 0 & -\frac{r+l}{2}\frac{1}{n} & 0\\ 0 & 1 & -\frac{t+b}{2}\frac{1}{n} & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\quad M_{sh}\begin{pmatrix}x\\ y\\ z\\ 1\end{pmatrix}=\begin{pmatrix}x-\frac{r+l}{2}\frac{z}{n}\\ y-\frac{t+b}{2}\frac{z}{n}\\ z\\ 1\end{pmatrix}$$

最終的に透視投影変換は

$$M_p=M_wM_sM_{sh}=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & \frac{n+f}{n-f} & \frac{-2nf}{n-f}\\ 0 & 0 & -1 & 0\end{pmatrix}\begin{pmatrix}\frac{-2n}{r-l} & 0 & 0 & 0\\ 0 & \frac{-2n}{t-b} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0 & -\frac{r+l}{2}\frac{1}{n} & 0\\ 0 & 1 & -\frac{t+b}{2}\frac{1}{n} & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$=\begin{pmatrix}\frac{-2n}{r-l} & 0 & \frac{r+l}{r-l} & 0\\ 0 & \frac{-2n}{t-b} & \frac{t+b}{t-b} & 0\\ 0 & 0 & \frac{n+f}{n-f} & \frac{-2nf}{n-f}\\ 0 & 0 & -1 & 0\end{pmatrix}$$

となります．

この方法は汎用性が高いものの扱いづらいため，実際には画角とアスペクト比を使った方法がよく使われます．画角を fovy，アスペクト比を aspect(=幅/高さ) とすれば

$$M_p=\begin{pmatrix}\frac{f}{aspect} & 0 & 0 & 0\\ 0 & f & 0 & 0\\ 0 & 0 & \frac{n+f}{n-f} & \frac{-2nf}{n-f}\\ 0 & 0 & -1 & 0\end{pmatrix}$$

ここで

$$f=\frac{1}{\tan\left(\frac{fovy}{2}\right)}=\cot\left(\frac{fovy}{2}\right)$$

となっています．

3.3.2 正投影変換

透視投影変換のときは奥行きによって見え方が変わらないので，w による透視除算がありません．正投影変換では，ビューボリュームの中心を原点に持ってきて，大きさを正規化するだけです．まずはビューボリュームの中心を原点にする平行移動を求めます．これは次のようになります．

$$M_t = \begin{pmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

各軸での中心を求めて移動しています．

次に正規化です．各軸がそれぞれ $[-1,1]$ の範囲になるようにします．これは次のようになります．

$$M_s = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

よって，正投影変換の行列は

$$M_o = M_s M_t = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

となります．次の図は正投影変換において $n=-1, f=-5$ のときのグラフです．

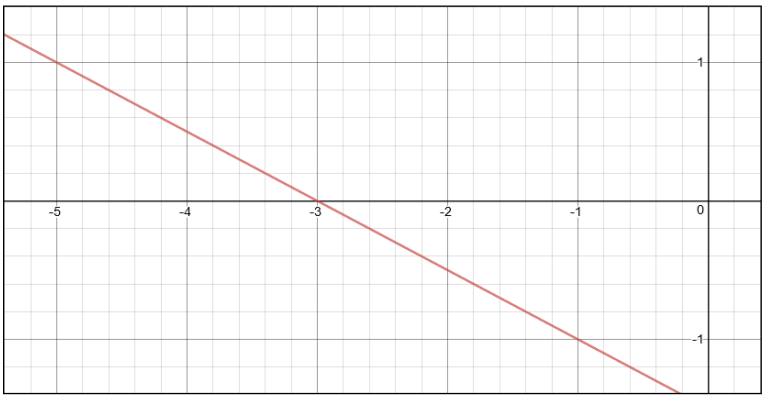

Fig.11: 正投影のクリップ空間の Z

OpenGL の投影関数

OpenGL には投影変換行列を設定してくれる関数があります．`glFrustum` 関数は前方クリッピング面の左右上下の座標 l,r,t,b と Z 軸の距離 n，および遠方クリッピング面の Z 軸の距離 f を指定して，透視投影変換行列を設定します．`gluPerspective` 関数は視野角 (fovy) とアスペクト比 (aspect)，前方・後方クリッピング面の Z 軸の距離 n,f を指定して，透視投影変換行列を設定します．`glOrtho` 関数はクリッピング面の左右上下の座標 l,r,t,b と前方・後方クリッピング面の Z 軸の距離 n,f を指定して正投影行列を設定します．これらの関数のドキュメントには行列の各要素が記載されています．その行列と，これまで導出してきた行列と比較してみるといくつか符号が違っているところがあります．これは，OpenGL の関数では n,f を正の値，導出した行列の n,f は負の値だからです．そのため，導出した行列を OpenGL の関数の行列と同じにするには n,f の符号を反転すればよいことになります．よって，それぞれの行列は

$$M_{frustum} = \begin{pmatrix} \frac{-2(-n)}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{-2(-n)}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{(-n)+(-f)}{(-n)-(-f)} & \frac{-2(-n)(-f)}{(-n)-(-f)} \\ 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$M_{perspective} = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{(-n)+(-f)}{(-n)-(-f)} & \frac{-2(-n)(-f)}{(-n)-(-f)} \\ 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$M_{ortho} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{(-f)-(-n)} & -\frac{(-f)+(-n)}{(-f)-(-n)} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.4 ビューポート変換（viewport transformation）

投影変換によってクリップ空間に変換された座標は透視除算することで正規化デバイス座標に変換されます．

$$\vec{p}_{ndc} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

この正規化デバイス座標系は XYZ ともに −1 から 1 の範囲になっています．これを画面に表示するために**ビューポート変換**をします．画面のことをウィンドウとかスクリーンというので，ビューポート変換後をウィンドウ座標系やスクリーン座標系ともいいます．また，ビューポート変換をスクリーン座標変換ということもあります．

ビューポート変換ではスクリーンに表示する領域と位置を指定します．ここで，スクリーンの幅を w，高さを h，始点を ox,oy とおくとビューポート変換は

$$M_v = \begin{pmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2}+ox \\ 0 & -\frac{h}{2} & 0 & \frac{h}{2}+oy \\ 0 & 0 & \frac{f-n}{2} & \frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_v\vec{p}_{ndc} = \begin{pmatrix} x\frac{w}{2}+\frac{w}{2}+ox \\ -y\frac{h}{2}+\frac{h}{2}+oy \\ z\frac{f-n}{2}+\frac{f+n}{2} \\ 1 \end{pmatrix}$$

となります．Y が反転しているのは正規化デバイス座標系では Y 軸の上が正に対して，スクリーンでは左上が原点であり Y 軸は下が正だからです．Z は $[n,f]$ の範囲になるようにしています．通常は $n=0, f=1$ です．

3.5 一般的な頂点データのパイプライン

レンダリングパイプラインの主要な座標変換を一通り見てきました．一般的に頂点データがどのような座標変換が適用されて画面に表示されるかを図にしてみました．

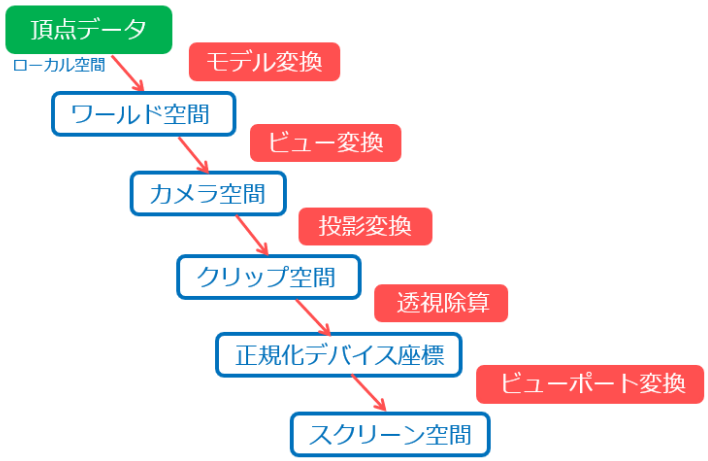

Fig.12: 一般的な頂点データのパイプライン

実際は環境によって変わる場合があると思いますので，この辺り触れる機会があれば，その環境のドキュメントを確認してください．

4． さいごに

簡単にまとめるつもりが，なんか思っていた以上に長くなってしまった気がしますね．また，説明がまだ上手く出来ていないところが多々あるので反省．意見や感想，間違いの指摘など受け付けていますので，なにかあればよろしくおねがいします．また，少しでも参考になれば幸いです．

5． 付録 A：Three.js の深度の関数

Three.js のシェーダコードには深度と Z 値を変換する以下の関数があります．

- viewZToOrthographicDepth
- orthographicDepthToViewZ
- viewZToPerspectiveDepth
- perspectiveDepthToViewZ

例えば，深度の値からカメラ座標系での Z 値を求めれば，スクリーンの座標からカメラ座標系での座標 (XYZ) を求めることができます．また，透視変換の場合は一般的に深度の値が線形になっていないため，線形に変換して使用することがあります．これらの関数のコードは次のようになっています．

```
float viewZToOrthographicDepth(float viewZ, float near, float far)
{
  return (viewZ + near) / (near - far);
}
```

```
float orthographicDepthToViewZ(float linearClipZ, float near, float far)
{
    return linearClipZ * (near - far) - near;
}

float viewZToPerspectiveDepth(float viewZ, float near, float far)
{
    return ((near + viewZ) * far) / ((far - near) * viewZ);
}

float perspectiveDepthToViewZ(float invClipZ, float near, float far)
{
  return (near * far) / ((far - near) * invClipZ - far);
}
```

最初に，perspectiveDepthToViewZ から見ていきましょう．これを数式で表すと

$$viewZ = \frac{nf}{(f-n) \times invClipZ - f} \tag{1}$$

となります．まず，これらの関数の **n** と **f** は正の値を指定します．よって，透視投影の式は

$$p'_z = az + b = \frac{n+f}{n-f}z + \frac{-2nf}{f-n}$$

$$a = \frac{n+f}{n-f}$$

$$b = \frac{-2nf}{f-n}$$

です．また，正規化デバイス座標の z_{ndc} は次のようになります．

$$z_{ndc} = \frac{p'_z}{p'_w} = \frac{1}{-z}\left(\frac{n+f}{n-f}z + \frac{-2nf}{f-n}\right) \tag{2}$$

正規化デバイス座標では $[-1,1]$ の範囲になりますが，一般的に深度バッファに格納される深度値 $depth$ は $[0,1]$ の範囲にビューポート変換されます．よって

$$depth = \frac{z_{ndc}}{2} + \frac{1}{2}, \qquad \therefore z_{ndc} = 2 \cdot depth - 1$$

これを式（2）に代入すると

$$2 \cdot depth - 1 = \frac{1}{-z}\left(\frac{n+f}{n-f}z + \frac{-2nf}{f-n}\right)$$

となります．z について解くと

$$
\begin{aligned}
2\cdot depth-1 &= \frac{1}{-z}\left(\frac{n+f}{n-f}z+\frac{-2nf}{f-n}\right)\\
&= -\frac{n+f}{n-f}+\frac{-2nf}{f-n}(-z)\\
2\cdot depth &= \frac{f+n}{f-n}+\frac{2nf}{(f-n)z}+1\\
&= \frac{(f+n)z}{(f-n)z}+\frac{2nf}{(f-n)z}+\frac{(f-n)z}{(f-n)z}\\
&= \frac{fz+nz+2nf+fz-nz}{(f-n)z}\\
&= \frac{2fz+2nf}{(f-z)n}\\
depth &= \frac{fz+nf}{(f-n)z} \qquad (3)\\
&= \frac{fz}{(f-n)z}+\frac{fn}{(f-n)z} = \frac{f}{f-n}+\frac{fn}{(f-n)z}\\
\frac{nf}{(f-n)z} &= depth-\frac{f}{f-n}\\
\frac{nf}{(f-n)z} &= \frac{(f-n)\cdot depth}{f-n}-\frac{f}{f-n} = \frac{(f-n)\cdot depth-f}{f-n}\\
\frac{(f-n)z}{nf} &= \frac{f-n}{(f-n)\cdot depth-f}\\
(f-n)z &= \frac{(f-n)nf}{(f-n)\cdot depth-f}\\
z &= \frac{nf}{(f-n)\cdot depth-f} \qquad (4)
\end{aligned}
$$

ここで式（1）と比べると

$$viewZ = \frac{nf}{(f-n)\times invClipZ-f} \qquad (5)$$

一致していますね．よって，**invClipZ** には **depth** を渡せばいいことになります．そして，**perspectiveDepthToViewZ** の逆関数 **viewZToPerspectiveDepth** は **depth** について解けば式が得られます．式（3）から

$$depth = \frac{fz+nf}{(f-n)z} = \frac{(n+z)f}{(f-n)z}$$

となって，**viewZToPerspectiveDepth** の実装と同じ式になっていることがわかります．今度は **orthographicDepthToViewZ** を見てみると

$$viewZ = linearClipZ*(n-f)-n \qquad (6)$$

となっています．こちらも n と f は正の値なので

$$p'_z = az + b = \frac{2}{n-f}z + \frac{n+f}{n-f}$$

$$a = \frac{2}{n-f}$$

$$b = \frac{n+f}{n-f}$$

正規化デバイス座標の z_{ndc} は

$$z_{ndc} = \frac{p'_z}{p'_w} = \frac{p'_z}{1} = p'_z \tag{7}$$

深度値 $depth$ は

$$depth = \frac{z_{ndc}}{2} + \frac{1}{2}, \qquad \therefore z_{ndc} = 2 \cdot depth - 1$$

これらの式から

$$2 \cdot depth - 1 = \frac{2}{n-f}z + \frac{n+f}{n-f}$$

$$\frac{2}{n-f}z = 2 \cdot depth - 1 - \frac{n+f}{n-f}$$

$$= \frac{2 \cdot depth \cdot (n-f)}{n-f} - \frac{n-f}{n-f} - \frac{n+f}{n-f}$$

$$= \frac{2 \cdot depth \cdot (n-f) - n + f - n - f}{n-f} = \frac{2 \cdot depth \cdot (n-f) - 2n}{n-f}$$

$$2z = 2 \cdot depth \cdot (n-f) - 2n$$

$$z = depth \cdot (n-f) - n \tag{8}$$

となります．これは式（6）と一致します．また，**orthographicDepthToViewZ** の逆関数 **viewZToOrthographicDepth** は

$$depth \cdot (n-f) - n = z$$

$$depth \cdot (n-f) = z + n$$

$$depth = \frac{z+n}{n-f} \tag{9}$$

となります．

5.1 線形（リニア）深度

一般的に透視投影の深度は線形ではありません．これだと扱いづらいので，線形深度に変換して使用します．透視投影では深度が線形ではありませんが，正投影では線形深度になっています．

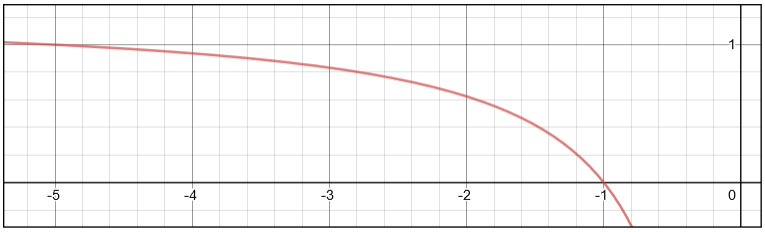

Fig.13: 透視投影の深度

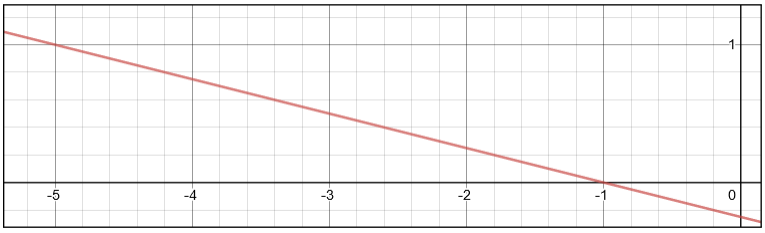

Fig.14: 正投影の深度

Three.js では **perspectiveDepthToViewZ** と **viewZToOrthographicDepth** を使って次のように線形深度に変換することができます.

```
float getLinearDepth(float invClipZ, float near, float far)
{
  float viewZ = perspectiveDepthToViewZ(invClipZ, near, far);
  return viewZToOrthographicDepth(viewZ);
}
```

また perspectiveDepth を depth，orthographicDepth を linearDepth とおくと

$$linearDepth = \frac{z+n}{n-f}, \qquad z = \frac{nf}{(f-n)\cdot depth - f}$$

linearDepth について解くと

$$
\begin{aligned}
linearDepth &= \frac{z+n}{n-f} \\
(n-f)\cdot linearDepth &= z+n \\
&= \frac{nf}{(f-n)\cdot depth - f} + n \\
&= \frac{nf}{(f-n)\cdot depth - f} + \frac{\{(f-n)\cdot depth - f\}\cdot n}{(f-n)\cdot depth - f} \\
&= \frac{nf + n(f-n)\cdot depth - nf}{(f-n)\cdot depth - f} \\
&= \frac{n(f-n)\cdot depth}{(f-n)\cdot depth - f} \\
-(f-n)\cdot linearDepth &= \frac{n(f-n)\cdot depth}{(f-n)\cdot depth - f} \\
linearDepth &= \frac{-n\cdot depth}{(f-n)\cdot depth - f} \\
&= \frac{-n\cdot depth}{f\cdot depth - n\cdot depth - f} \\
&= \frac{-n\cdot depth}{f(depth-1) - n\cdot depth}
\end{aligned}
$$

となります．よって，**getLinearDepth** 関数は次のように書きかえることができます．

```
float getLinearDepth(float depth, float near, float far)
{
  float nz = near * depth;
  return -nz / (far * (depth-1.0) - nz);
}
```

参考文献

[1] CG-ARTS 協会「コンピュータグラフィックス」CG-ARTS 協会，2015

[2] 久富木隆一「ゲームアプリの数学」SB クリエイティブ株式会社，2015

[3] Yuki Ozawa「深度バッファのレンジと精度を最大化する」https://qpp.bitbucket.io/translation/maximizing_depth_buffer_range_and/

[4] Brano Kemen「Maximizing Depth Buffer Range and Precision」http://outerra.blogspot.com.ar/2012/11/maximizing-depth-buffer-range-and.html

[5]「Three.js」https://threejs.org/