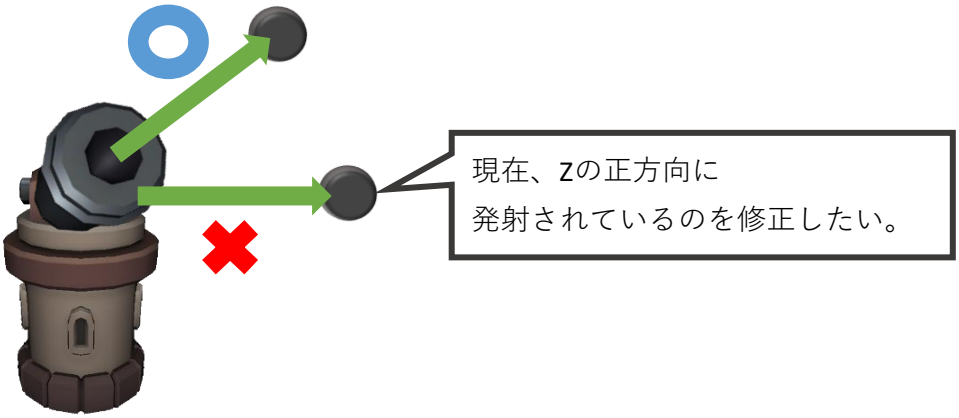
行列による座標の回転

次に、生成した弾を砲身の角度に合わせて移動させていきたいと思います。



繰り返しになりますが、移動処理とは、座標 + 移動量。
移動量 = 方向 × スピード で求められます。
弾のスピードは、適当に調整するとして、方向を何とかして求める必要があります。

砲身が持っている角度は、Zの正方向({0.0f, 0.0f, 1.0f})を向いている状態を
0度として、どれだけX軸、Y軸、Z軸に回転させたのか、というラジアン値になります。

ということは、Zの正方向({0.0f, 0.0f, 1.0f})を砲身の角度分回転させることで、
砲身が向いている方向を取得することができます。

方向といっても、{0.0f, 0.0f, 1.0f} ただの座標と変わりませんので、
座標を回転させればよい、というイメージでも大丈夫です。

回転の軸が１種類のみでしたら、２Ｄのように三角関数を用いて、
簡単に計算できそうですが、この砲身は、X軸回転とY軸回転が
混在した回転になっていますので、軸ごとに回転させる必要があります。

このように３Ｄでは、複数軸でオブジェクトを回転させることが
当たり前になってきますので、少しでも計算が楽になるように、
数学の行列が登場してきます。

数学で習った(習うであろう)行列は、右手系になります。

【右手系】X軸回転

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

【右手系】Y軸回転

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

【右手系】Z軸回転

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

DxLibは左手系になりますので、数学の行列を転置する必要があります。

左手系はこちら。

【左手系】X軸回転

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

【左手系】Y軸回転

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

【左手系】Z軸回転

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

インターネットで行列で検索すると数学系(右手系)で解説されている
サイトが多いですので、混乱しないように気を付けてください。

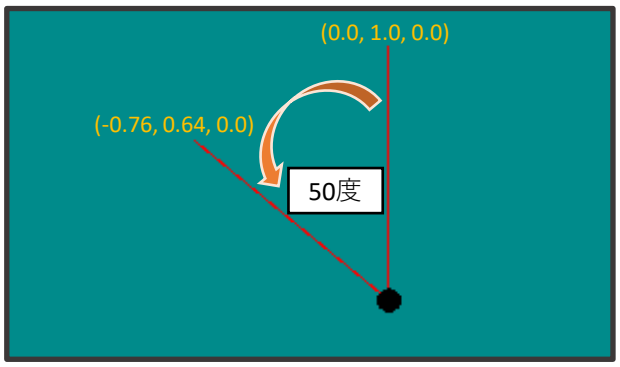
試しに３ＤのＺ回転を計算してみましょう。
↑方向(0.0f, 1.0f, 0.0f)を50度(=0.872..ラジアン)回転させます。

sin(0.872..) = 0.766..
cos(0.872..) = 0.642..

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0.642 & 0.766 & 0 & 0 \\ -0.766 & 0.642 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

横×縦で計算を行っていきますので、

```
(0.0 * 0.642) + (1.0 * -0.766) + (0.0 * 0.0) + (0.0 * 0.0)    =  -0.766..
(0.0 * 0.766) + (1.0 *  0.642) + (0.0 * 0.0) + (0.0 * 0.0)    =   0.642..
(0.0 *   0.0) + (1.0 *    0.0) + (0.0 * 1.0) + (0.0 * 0.0)    =   0.0
(0.0 *   0.0) + (1.0 *    0.0) + (0.0 * 0.0) + (0.0 * 1.0)    =   0.0
```



正面がZの正方向になりますので、正方向を見た時の、
時計の反対回りが、正の回転になります。しっかり向きが回転しました。

この計算をそれぞれの軸で行っていけば良いのですが、
それでも複雑になりがちで、計算量も多くなりやすいですので、
DxLibでは以下の手順で、回転の制御や、座標(方向)の回転を行っていきます。

①回転行列を作る

```
// 単位行列(無回転の状態)
MATRIX matRot = MGetIdent();
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ 単位行列

```
// それぞれの軸の行列を作り、更に行列を合成していく
matRot = MMult(matRot, MGetRotX(barrelRot_.x));
matRot = MMult(matRot, MGetRotY(barrelRot_.y));
matRot = MMult(matRot, MGetRotZ(barrelRot_.z));
```

MATRIX

DxLibで定義されているfloat4x4の構造体。(float m[4][4])

MATRIX MGetRotX(float XAxisRotate)

引数で指定された回転値分だけＸ軸回転する回転行列を戻り値として返す。
引数の回転値の単位は、ラジアン。MGetRotY、MGetRotZも同様。
先ほどのZ回転50度の例でいくと、MGetRotZ(0.872..)の結果は、以下となる。

$$\begin{pmatrix} 0.642 & 0.766 & 0 & 0 \\ -0.766 & 0.642 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

MATRIX MMult(MATRIX In1, MATRIX In2)

二つの行列の乗算を行う。行列の合成などともいいます。
内部式は長いのでリファレンスを参照してください。
https://dxlib.xsrv.jp/function/dxfunc_3d_math.html#R11N17

乗算を行う順番によって、結果が変わってきますので、
しばらくは、XYZの順番で合成していきます。

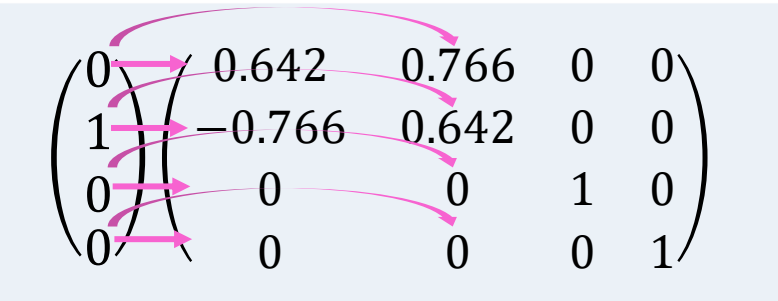
②方向を回転させる

```
// 回転行列を使用して、ベクトルを回転させる
VECTOR dir = VTransform({ 0.0f, 0.0f, 1.0f }, matRot);
```

```
-------------------------------------------------------------
```

```
VECTOR    VTransform( const VECTOR &InV, const MATRIX &InM )
{
    VECTOR Result ;
    Result.x = InV.x * InM.m[0][0] + InV.y * InM.m[1][0]
                                   + InV.z * InM.m[2][0] + InM.m[3][0] ;
    Result.y = InV.x * InM.m[0][1] + InV.y * InM.m[1][1]
                                   + InV.z * InM.m[2][1] + InM.m[3][1] ;
    Result.z = InV.x * InM.m[0][2] + InV.y * InM.m[1][2]
                                   + InV.z * InM.m[2][2] + InM.m[3][2] ;
    return Result ;
}
```

行列を使ったベクトルの変換、とリファレンスには記載がありますが、
回転だけではなく、ベクトルの拡大・縮小、平行移動なども行いますので、
「変換」というワードが使用されています。
今回は、方向(ベクトル)を回転させる、という意味合いで、捉えて貰って
大丈夫です。
やっていることとしては、Z回転でやった方向と行列の計算です。



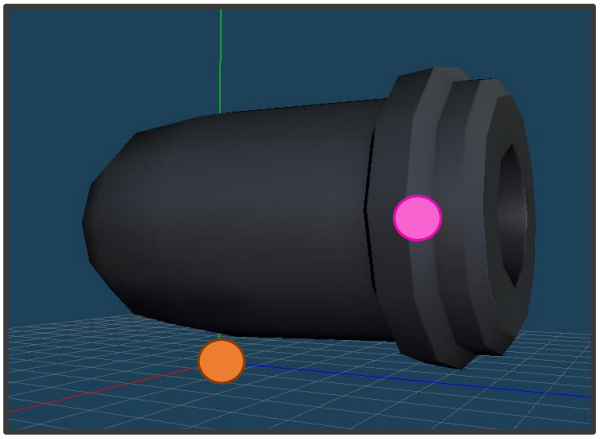
③回転させた方向(砲身の向き)に球を発射する

```
// 弾を指定位置から、指定方向に発射させる
shot->CreateShot(barrelPos_, dir);
```

回転、難しいですね。。。
ですが、これも慣れです。復習として、もう一度同じようなことを
やっていきます。

実装が上手くいっていれば、砲身の向きに球が移動している状態かと思います。

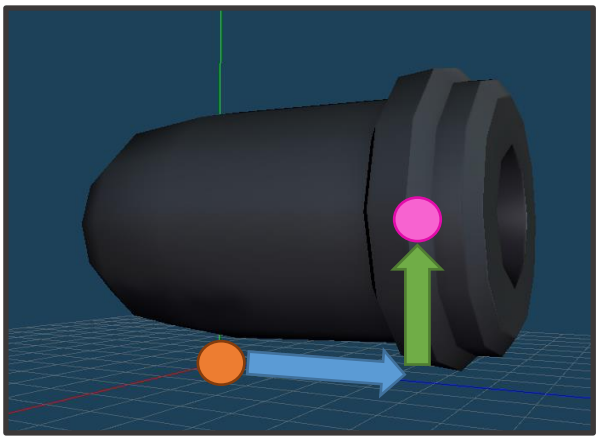
しかし、弾の発射位置が砲身の少し下からになっていますので、
ズレて発射されているようになってしまっています。
これは、砲身の３Ｄモデルの原点が、下図のように砲身の下に作られているからです。
（ ３Ｄモデル上の原点 ＝ モデルの座標 ）



回転も、この原点を中心に
回るような動きになりますので、
もし、調整したい場合は、
Blender等で
FBXファイルから修正する
ことができます。

３Ｄモデルの原点自体は、変更しませんが、
弾の発射位置は、ピンクあたりの位置に調整していきたいと思います。

原点からの相対座標を使いましょう。



Zのプラス(正方向)に、
Yのプラス(正方向)。

前方側の上ですね。

仮で、相対座標、
{ 0.0f, 25.0f, 30.0f }
としましょう。

位置の確認用にDraw関数内で球体を描画してみましょう。

Cannon.cpp

```
void Cannon::Draw(void)
{
    // 弾の発射位置
    VECTOR pos = barrelPos_;

    // 砲身からの相対座標
    VECTOR localPos = { 0.0f, 25.0f, 30.0f };

    // ローカル座標からワールド座標へ変換
    pos = VAdd(pos, localPos);

    // 弾の発射位置目安
    DrawSphere3D(pos, 10.0f, 10, 0x00ff00, 0x00ff00, true);

    // 砲台のモデル描画
    MV1DrawModel(standModelId_);

    // 砲身のモデル描画(一時的に半透明にする)
    MV1SetMaterialDifColor(barrelModelId_, 0, GetColorF(1.0f, 1.0f, 1.0f, 0.5f));
    MV1DrawModel(barrelModelId_);
}
```

砲台・砲身が前方を向いている時は問題無いのですが、
横や後ろを向いた時、発射させたい位置と異なっていることがわかります。

これは、相対座標が、【前方側の上】になっているままだからです。

合ってる

ずれている

先ほどの発射方向と同じように、
この相対座標も、砲身の向きに応じて、回転させる必要があるのです。
同じ手順でやってみましょう。

①回転行列を作る

```
// 砲身の回転行列
MATRIX matRot = MGetIdent();
matRot = MMult(matRot, MGetRotX(barrelRot_.x));
matRot = MMult(matRot, MGetRotY(barrelRot_.y));
matRot = MMult(matRot, MGetRotZ(barrelRot_.z));
```

②相対座標を回転させる

```
// 方向と同じ要領で、相対座標を回転
VECTOR localPosRot = VTransform(localPos, matRot);
```

③相対(ローカル)座標から絶対(ワールド)座標へ変換

```
pos = VAdd(pos, localPosRot);
```

合ってる！

合ってる！

Y軸回転、X軸回転を行っても、綺麗に発射位置が確保できていると思います。

それでは、ProcessShot関数に戻って、弾の発射処理を完成させていきましょう。