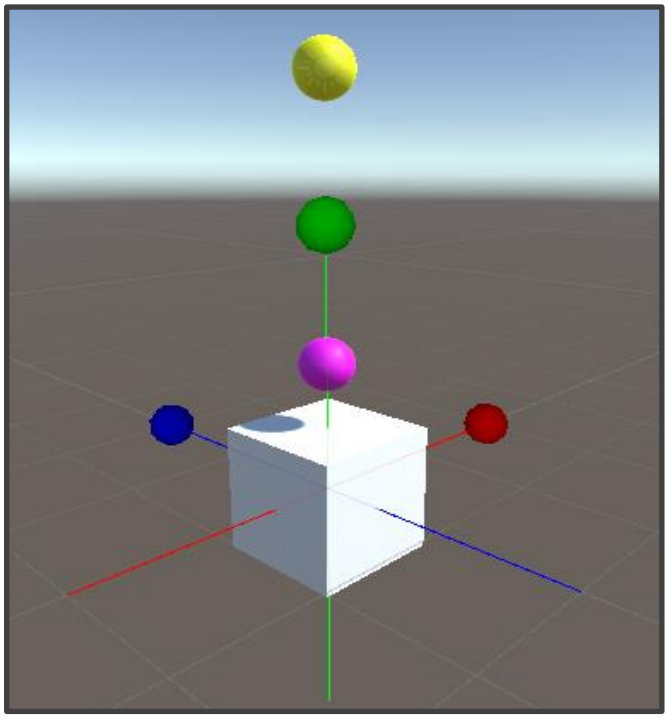
もう１度回転の整理

回転による移動イメージと、プログラムを整理していきます。



白い四角形キューブの上に黄色の球体があります。
キューブからYのプラス3 {0.0f, 3.0f, 0.0f} に位置しています。

紫の球体は、キューブからYのプラス1 {0.0f, 1.0f, 0.0f} に位置しています。

白いキューブのグローバル座標が、　　{0.0f, 0.0f, 0.0f} の場合、
　紫　　のグローバル座標は、　　{0.0f, 3.0f, 0.0f}
　黄色　　のグローバル座標は、　　{0.0f, 1.0f, 0.0f}

白いキューブのグローバル座標が、　　{10.0f, 10.0f, 10.0f} の場合、
　紫　　のグローバル座標は、　　{10.0f, 13.0f, 10.0f}
　黄色　　のグローバル座標は、　　{10.0f, 11.0f, 10.0f}

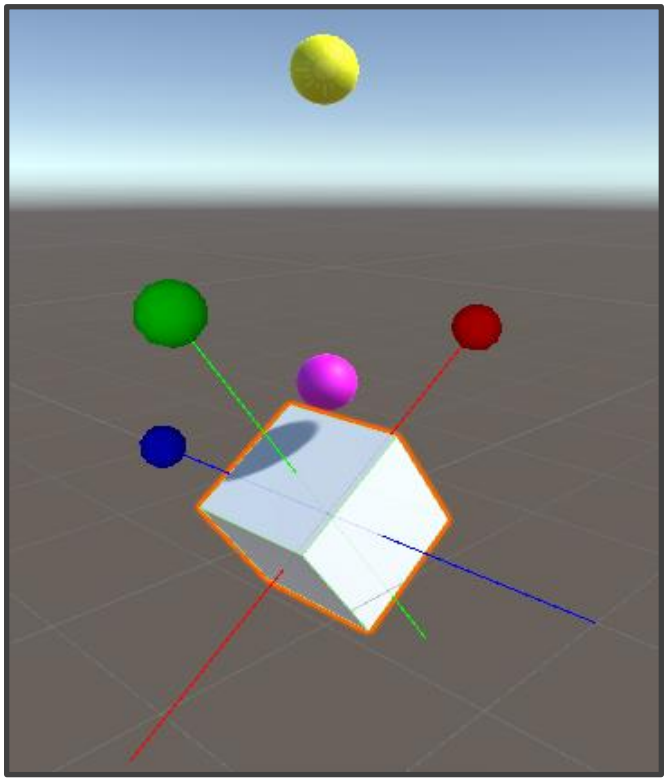
となります。
黄色の球体のグローバル座標の求め方をプログラムで書くと、

```
VECTOR yellowPos = VAdd(whitePos, yellowLocalPos);
```

となります。

yellowLocalPosが、{0.0f, 3.0f, 0.0f}となりますが、
この座標のことを、白いキューブに対するローカル座標、相対座標といいます。

白いキューブが無回転(Zの正方向を向いている)場合は、
このような単純な計算で、ローカル座標からグローバル座標に変換できますが、
回転がかかってしまうと、もうひと手間かかってしまいます。



白いキューブの位置は変えず、Zを正の35度回転をかけると、
上図のようになります。

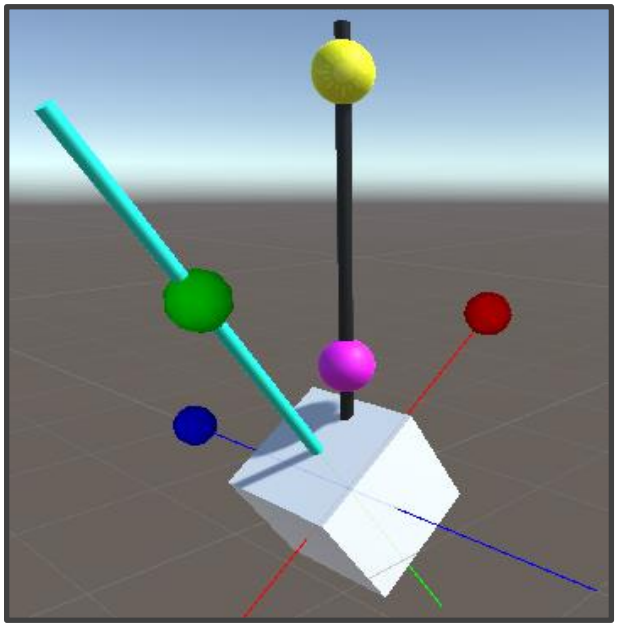
白いキューブの上方向が変わっていますが、
先ほどの単純な計算のままだと、黄色と紫が白いキューブの回転に
対応できていないことがわかります。

回転移動の前に、方向の復習になりますが、
黄色の紫の球体も、白いキューブから見た時、同じ方向に位置しています。
なぜ、同じ方向になるかというと、
単位ベクトル(力や長さを1にしたベクトル)が一致しているからです。
単位ベクトルを求めるプログラムは、DxLibの関数を使用すると、

```
VNorm({0.0f, 3.0f, 0.0f})        →      {0.0f, 1.0f, 0.0f}
VNorm({0.0f, 1.0f, 0.0f})        →      {0.0f, 1.0f, 0.0f}
```

上記のようになり、一致していますので、同じ方向と言えます。
VNormは、ベクトルを正規化して、単位ベクトルに変換する関数です。

黄色や紫の球体にとって、白いキューブは親にあたりますので、
親の回転が変わったら、子も回転させないといけません。
回転とは、このベクトルを変更することだとイメージしましょう。



黒線のベクトル
{0.0f, 1.0f, 0.0f}から、
水色のベクトルに【変換】
する必要があります。

このベクトル変換を行うプログラムを、行列計算で行うと、

```
// 白いキューブの回転行列
MATRIX whiteMat = mWhite->GetTransform()->matRot;
// 白いキューブの回転行列を元に第一引数のベクトルを変換する
VECTOR yellowLocalPos = VTransform({0.0f, 3.0f, 0.0f}, whiteMat);
// 白いキューブのグローバル座標に相対座標を加え、
// 黄色の球体のグローバル座標を求める
VECTOR yellowPos = VAdd(whitePos, yellowLocalPos);
```

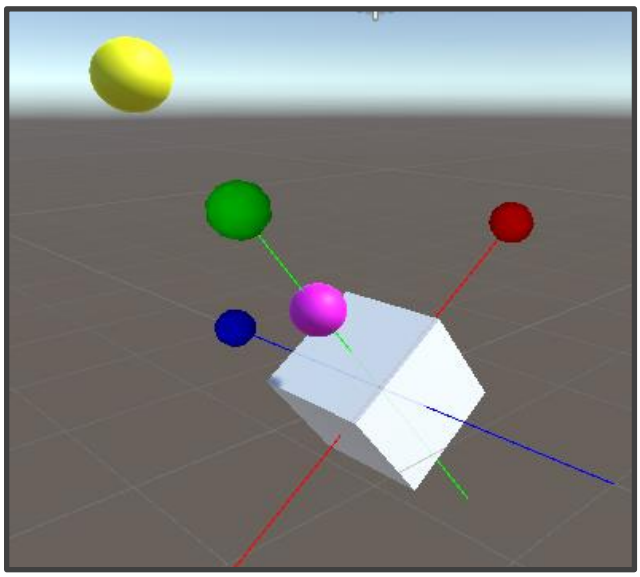
クォータニオン計算で行うと、

```
// pos' = q・pos・q(-1)
Quaternion tmp;
tmp = tmp.Mult(mWhite->GetTransform()->quaRot);
tmp = tmp.Mult(Quaternion(0.0f, 0.0f, 3.0f, 0.0f));
tmp = tmp.Mult(mWhite->GetTransform()->quaRot.Inverse());
VECTOR yellowLocalPos = { (float)tmp.x, (float)tmp.y, (float)tmp.z };
VECTOR yellowPos = VAdd(whitePos, yellowLocalPos);
```

計算が長いので、クォータニオン計算を一部関数化して、

```
// 白いキューブの回転クォータニオン
Quaternion whiteQua = mWhite->GetTransform()->quaRot;
// pos' = q・pos・q(-1)を一通り計算し、
// 引数で渡された相対座標にクォータニオン回転を加えて
// 新しい相対座標を取得
VECTOR yellowLocalPos = whiteQua.PosAxis({0.0f, 3.0f, 0.0f});
VECTOR yellowPos = VAdd(whitePos, yellowLocalPos);
```

このように計算していくと、



どのやり方でも、黄色と紫の球体が、
白いキューブの上方向に回転移動します。

以上、回転のおさらいとなります。

※ 方向に関する補足ですが、

```
VECTOR dir = VNorm(VTransform({0.0f, 0.0f, 1.0f}, mat));
```

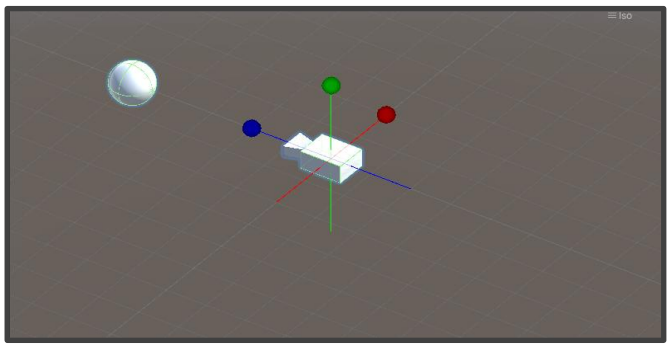
時々、このようにして、行列から、方向(単位ベクトル)を取得する
式をコーディングしているかと思います。
{1.0f, 0.0f, 0.0f}や{0.0f, 1.0f, 0.0f}は、力や長さが1の単位ベクトル
になりますので、それを回転移動させると、対象物にとっての、
前方方向、右方向、上方向などが取得できます。
VNorm(正規化)をする必要はないのですが、癖付けのために記述しています。

そして、クォータニオンによる、回転移動の演習として、
カメラにFREEモード(カメラを自由に動かせる)を追加していきます。

移動においては、WASDキーを押下すると、
カメラの向いている方向に対して、
前方、後方、右方向、左方向に移動するようにしてください。

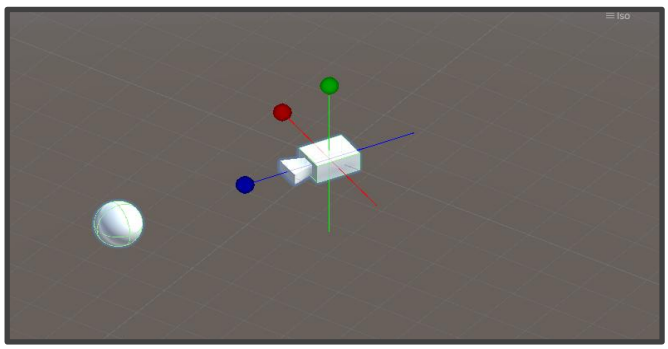
カメラの回転については、
現在のカメラ位置を中心として、注視点を回転移動させるようにしてください。
キーは、矢印キーとします。

←キーを押下すると



こんな感じに
注視点を移動。

この状態から、
↑キーを押下すると、



こんな感じに
注視点を移動。

イベントシーンの
カメラワークなどに
使用すると楽しいです。