親子回転とタレット

タレットの座標など、微調整する時間が惜しいですので、
必要な情報は、下記を参照ください。

ボス情報

スケール	2.0f
ローカル回転	Yの180度

タレット情報

スケール	80.0f

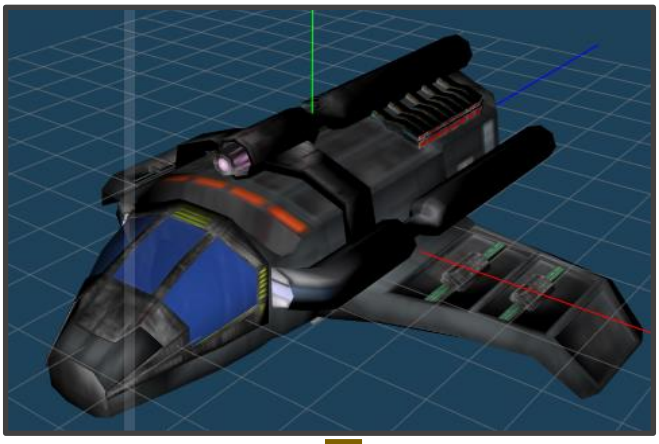
ボス座標を親とした場合の相対座標と相対回転

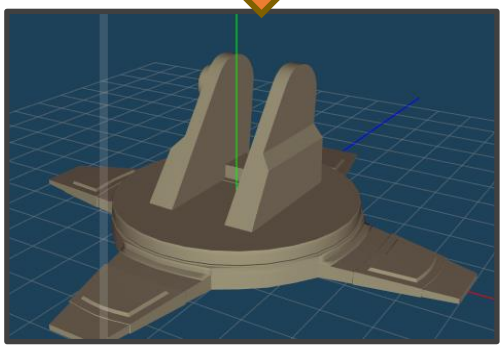
前方①座標	{ 4.5f, 5.5f, 7.8f }
前方①回転	{ 0.0f, 0.0f, AsoUtility::Deg2RadF(-18.0f) }
前方②座標	{ -4.5f, 5.5f, 7.8f }
前方②回転	{ 0.0f, 0.0f, AsoUtility::Deg2RadF(18.0f) }
横①座標	{ 4.5f, 5.5f, 0.0f }
横①回転	{ AsoUtility::Deg2RadF(20.0f), AsoUtility::Deg2RadF(90.0f), 0.0f }
横②座標	{ -4.5f, 5.5f, 0.0f }
横②回転	{ AsoUtility::Deg2RadF(20.0f), AsoUtility::Deg2RadF(-90.0f), 0.0f }
後方①座標	{ 3.5f, 5.0f, -17.8f }
後方①回転	{ 0.0f, AsoUtility::Deg2RadF(180.0f), AsoUtility::Deg2RadF(18.0f) }
後方②座標	{ -3.5f, 5.0f, -17.8f }
後方②回転	{ 0.0f, AsoUtility::Deg2RadF(180.0f), AsoUtility::Deg2RadF(-18.0f) }

砲台は、Y軸回転を常に行い、-30～30度を稼働範囲としている。
砲身は、X軸回転を常に行い、-10～20度を稼働範囲としている。

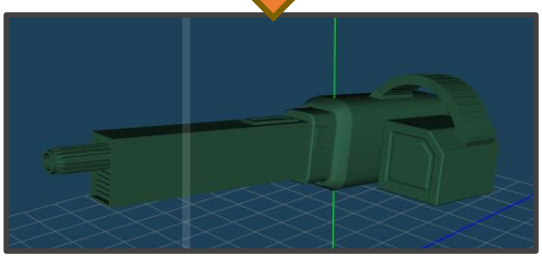
親子関係

上記で上げた、タレットの相対座標ですが、
ボスがZの正方向を向いている時の相対位置になりますので、
ボスが回転して、姿勢が変わると、その回転を位置にも反映する
必要があります。
位置は、それだけで良いですが、回転の方は、更に、
タレットの砲台部分と砲身部分が独立して、
Y軸回転、X軸回転を行っています。
親子関係でいくと、
（親）ボス → 砲台 → 砲身（子）となりますので、
それぞれの順番で回転を引き継ぎ、合成していく必要があります。





砲台と砲身は別々のモデルと
なっており、
それぞれ制御する。



砲身は、砲台の回転に影響
を受けるので、
ボス→砲台→砲身といった
具合に、孫関係にあたる。
位置は砲台と一緒で良い。

親子の相対位置

```
// タレットの回転をボス(親)の回転と同じにする
transform->quaRot = mTransformParent->quaRot;
// 取得してきた回転を使って、回転に応じた相対座標を取得する
localPos = Quaternion::PosAxis(transform->quaRot, mLocalPos);
// 親の座標に計算した相対座標を加える
transform->pos = VAdd(mTransformParent->pos, VScale(localPos, SCALE));
```

親子の相対回転

```
// 回転させたい回転量を計算していく
Quaternion localRot;

// まずは親からの相対回転
axis = Quaternion::AngleAxis(mLocalAddAxis.y, AsoUtility::AXIS_Y);
localRot = localRot.Mult(axis);

axis = Quaternion::AngleAxis(mLocalAddAxis.x, AsoUtility::AXIS_X);
localRot = localRot.Mult(axis);

axis = Quaternion::AngleAxis(mLocalAddAxis.z, AsoUtility::AXIS_Z);
localRot = localRot.Mult(axis);

// 次に今回のフレームで回転させたい回転量(砲台だとY軸、砲身だとX軸)
axis = Quaternion::AngleAxis(addAxis.y, AsoUtility::AXIS_Y);
localRot = localRot.Mult(axis);

axis = Quaternion::AngleAxis(addAxis.x, AsoUtility::AXIS_X);
localRot = localRot.Mult(axis);

axis = Quaternion::AngleAxis(addAxis.z, AsoUtility::AXIS_Z);
localRot = localRot.Mult(axis);

// 親の回転が子の回転に代入されている前提で、今回計算した回転を加える
transform->quaRot = transform->quaRot.Mult(localRot);
```

上記の親子の相対回転で、まずは砲台のモデル制御を行った後に、
砲台の回転を親として、その子供の砲身のモデル制御を行っていく。

後は、砲身の前方方向に球を発射して、
物足りなければ、一定時間経ったら弾が分裂して更に発射されるようにすると、
サンプルのタレットのような挙動になります。