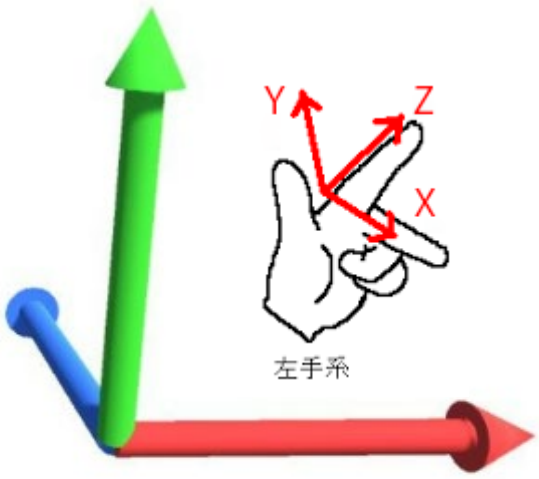
○概要

モデルを右手座標系から左手座標系へ変換する。

○座標系

3D 空間は X Y Z の 3 つの軸で表現されています。
この XYZ 軸をどちらの向きにするか決定する必要があり、このときの表現に右手座標系、左手座標系などと呼ぶことがあります。

右手座標系



左手座標系

DirectX は左手座標系で考えられており、本課題でも左手座標系で描画実装をしています。
しかし扱う 3D モデルが右手座標系で作られていることが多く、Assimp で読み取った 3D モデルのデータも右手座標系になっています。

現状の実装では描画処理は左手座標系、3D モデルデータは右手座標系と食い違っているため、左右が反転した状態で表示されています。

右手座標系

左手座標系

本当はこちらが正しい

座標系が違うと
左右が鏡写しになった状態で
表示されてしまう

今回は座標系の違いを行列計算で辻褄が合うように実装します。

Model.h

```
---省略---

class Model
{
public:
    ---省略---

    struct Node
    {
        ---省略---
        DirectX::XMFLOAT4X4    globalTransform;
        DirectX::XMFLOAT4X4    woorldTransform;
    };
};
```

グローバル行列とは
3D モデルの足元の位置を基準として
どれだけ離れているか、回転しているかを
表現する行列

ワールド行列は
世界の中心を基準として
どれだけ離れているか、回転しているかを
表現する行列

Model.cpp

```
---省略---

// トランスフォーム更新処理
void Model::UpdateTransform(const DirectX::XMFLOAT4X4& worldTransform)
{
    DirectX::XMMATRIX ParentWorldTransform = DirectX::XMLoadFloat4x4(&worldTransform);

    // 右手座標系から左手座標系へ変換する行列
    DirectX::XMMATRIX CoordinateSystemTransform = DirectX::XMMatrixScaling(-1, 1, 1);
```

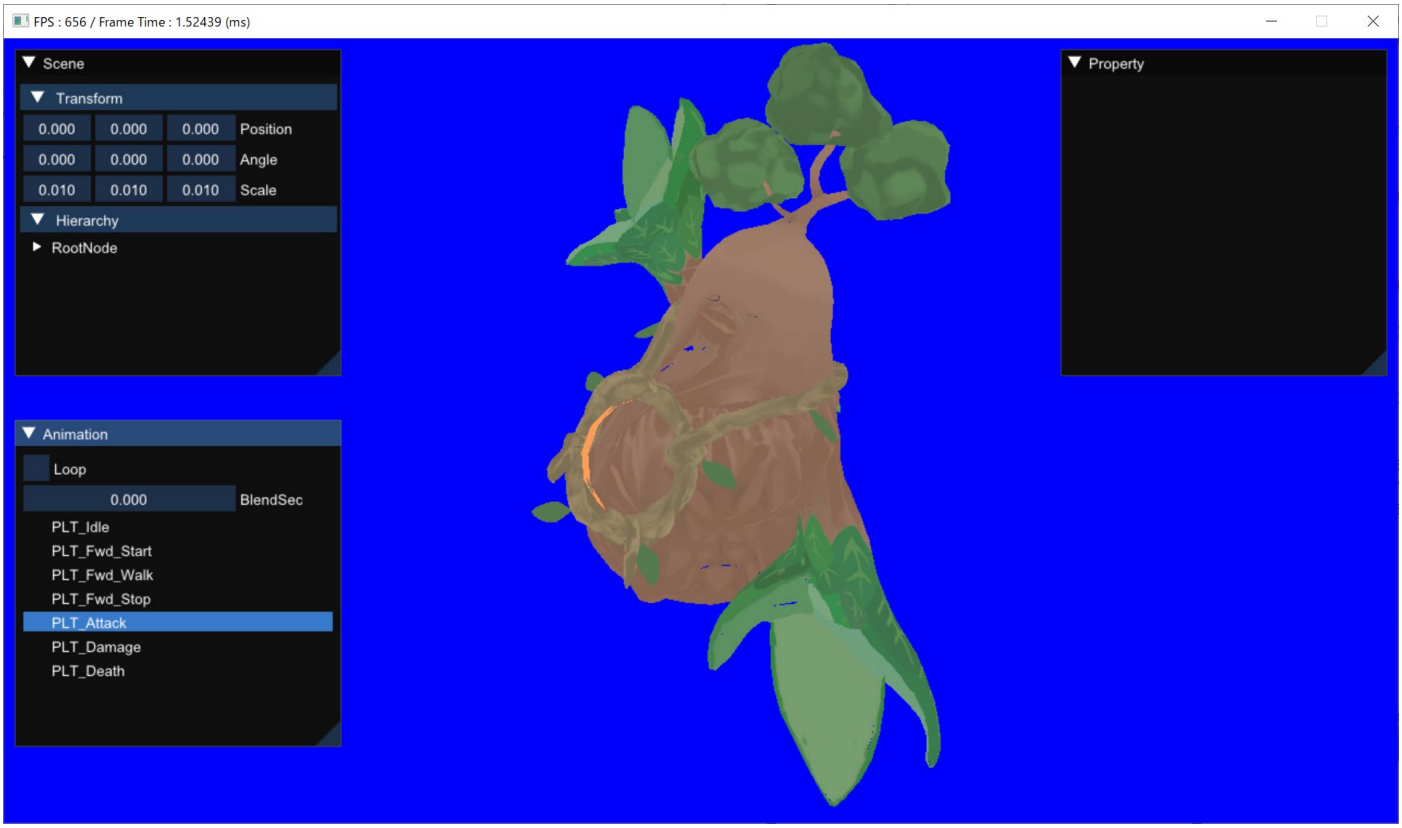
X 軸をマイナスにスケーリングすることで
右手系を左手系に変換する

```
    for (Node& node : nodes)
    {
        // ローカル行列算出
        DirectX::XMMATRIX S = DirectX::XMMatrixScaling(node.scale.x, node.scale.y, node.scale.z);
        DirectX::XMMATRIX R = DirectX::XMMatrixRotationQuaternion(DirectX::XMLoadFloat4(&node.rotation));
        DirectX::XMMATRIX T = DirectX::XMMatrixTranslation(node.position.x, node.position.y, node.position.z);
        DirectX::XMMATRIX LocalTransform = S * R * T;

        // グローバル行列算出
        DirectX::XMMATRIX ParentGlobalTransform;
        if (node.parent != nullptr)
        {
            ParentGlobalTransform = DirectX::XMLoadFloat4x4(&node.parent->globalTransform);
        }
        else
        {
            ParentGlobalTransform = DirectX::XMMatrixIdentity();
        }
        DirectX::XMMATRIX GlobalTransform = LocalTransform * ParentGlobalTransform;



        // ワールド行列算出
        DirectX::XMMATRIX WorldTransform = GlobalTransform * CoordinateSystemTransform * ParentWorldTransform;

        // 計算結果を格納
        DirectX::XMStoreFloat4x4(&node.localTransform, LocalTransform);
        DirectX::XMStoreFloat4x4(&node.globalTransform, GlobalTransform);
        DirectX::XMStoreFloat4x4(&node.worldTransform, WorldTransform);
    }
}
```

いきなりワールド行列を計算するのではなく、
右手座標系のグローバル行列を計算する。

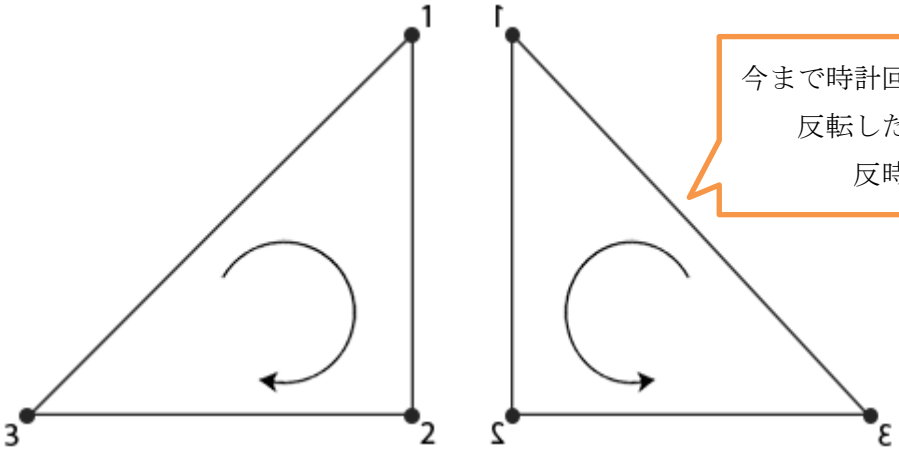
右手座標系のグローバル行列を
左手に変換し、ワールド行列を計算する。

実行確認してみましょう。
表示がおかしくなっていますが、右手で攻撃アニメーションができていれば OK です。
表示がおかしくなっている原因は表示するポリゴンが裏面になってしまっているからです。



○表示の修正

現在は頂点を時計回りに結ぶことでポリゴンを構成しています。
しかし座標系を変換するためにX軸を反転してしまったため、ポリゴンが裏返ってしまったため、表示を戻すには頂点を反時計回りにしてポリゴンを構成する必要があります。



今まで時計回りでポリゴンが表示されていたが、
反転した状態でポリゴンを表示するには
反時計回りにする必要がある。

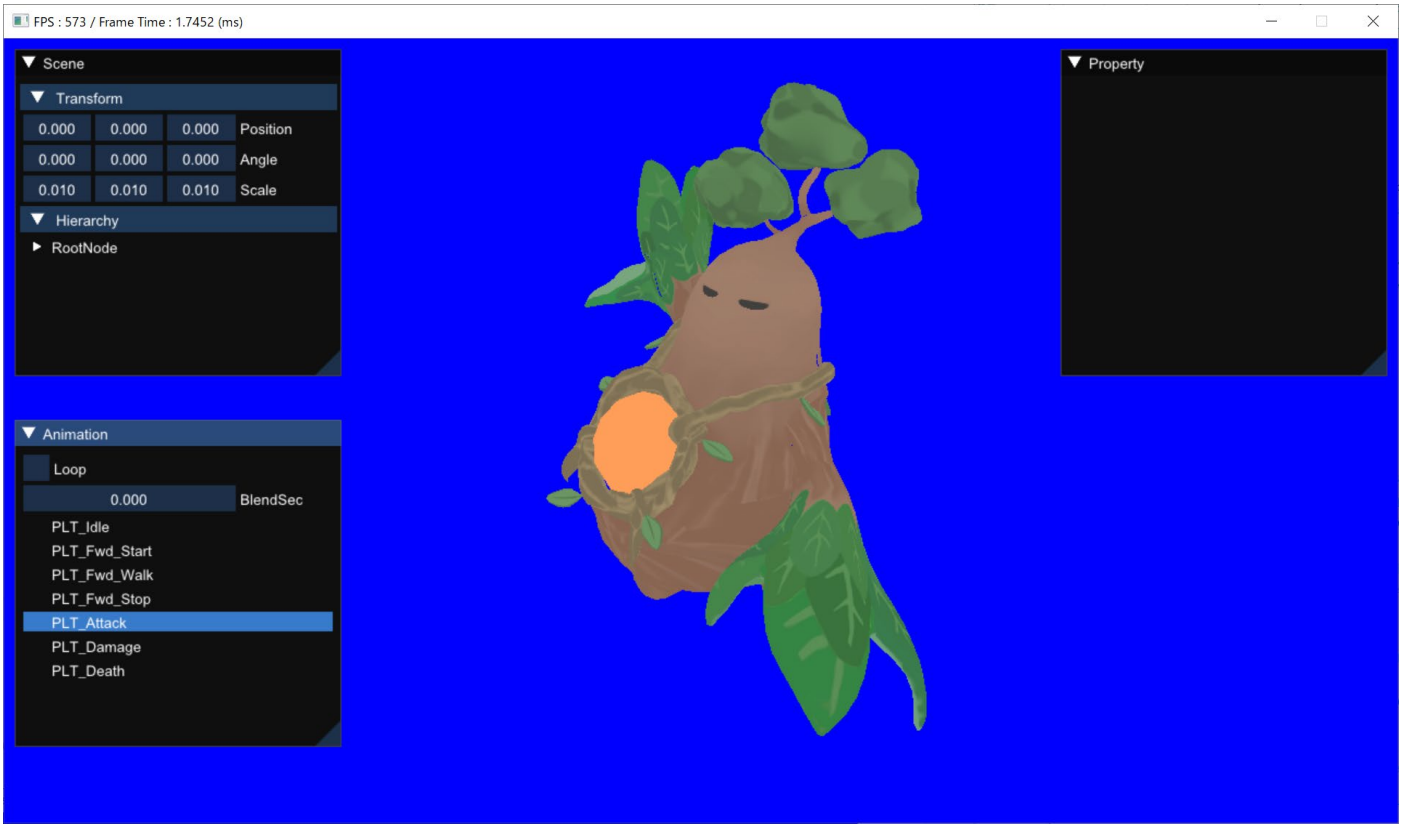
この問題を解決するにはラスタライザーステートの設定で調整するか、インデックスバッファの並び順を変えるかで解決できます。
今回はインデックスバッファの並び替えで解決します。

AssimpImporter.cpp

```
---省略---

// メッシュデータを読み込み
void AssimpImporter::LoadMeshes(MeshList& meshes, const NodeList& nodes, const aiNode* aNode,
                                                                          std::string nodePath)
{
    ---省略---

    // メッシュデータ読み取り
    for (uint32_t aMeshIndex = 0; aMeshIndex < aNode->mNumMeshes; ++aMeshIndex)
    {
        ---省略---

        // インデックスデータ
        for (uint32_t aFaceIndex = 0; aFaceIndex < aMesh->mNumFaces; ++aFaceIndex)
        {
            const aiFace& aFace = aMesh->mFaces[aFaceIndex];
            uint32_t index = aFaceIndex * 3;
            mesh.indices[index + 0] = aFace.mIndices[2];
            mesh.indices[index + 1] = aFace.mIndices[1];
            mesh.indices[index + 2] = aFace.mIndices[0];
        }
        ---省略---
    }
    ---省略---
}
```

三角形を構成する
頂点の順番を変える

実行確認してみましょう。
正しく表示されていれば OK です。



ちなみにラスタライザーステートで解決する場合は以下のパラメータを設定することで解決できます。

```
D3D11_RASTERIZER_DESC desc{};
desc.FrontCounterClockwise = true;
```

今回はここを設定してしまうとスプライトなど全ての描画に影響が出てしまうためインデックスバッファの並び替えで解決しました。

○スケーリング

3D モデルは制作するアーティストや環境によってサイズが異なることがあります。
例えばステージモデルは 1.0 が 1 メートル単位で制作され、キャラクターモデルは 1.0 が 1 センチメートル単位で制作されていることがあります。
この違いを座標系の変換と同時に吸収するプログラムを実装します。

Model.h

```
---省略---

class Model
{
public:
```

```
    Model(ID3D11Device* device, const char* filename);
    Model(ID3D11Device* device, const char* filename, float scaling = 1.0f);

    ---省略---

private:
    ---省略---
    float    scaling = 1.0f;
};
```

Model.cpp

```
---省略---

// コンストラクタ
Model::Model(ID3D11Device* device, const char* filename, float scaling)
    : scaling(scaling)
{
    ---省略---
}

---省略---

// トランスフォーム更新処理
void Model::UpdateTransform(const DirectX::XMFLOAT4X4& worldTransform)
{
    ---省略---

    // 右手座標系から左手座標系へ変換する行列
    DirectX::XMMATRIX CoordinateSystemTransform = DirectX::XMMatrixScaling(-1, 1, 1);
    DirectX::XMMATRIX CoordinateSystemTransform = DirectX::XMMatrixScaling(-scaling, scaling, scaling);

    ---省略---
}
```

Scene.cpp

```
---省略---

// コンストラクタ
ModelTestScene::ModelTestScene()
{
    ---省略---

    // モデル作成
    model = std::make_unique<Model>(device, "Data/Model/Plantune/plantune.fbx");
    model = std::make_unique<Model>(device, "Data/Model/Plantune/plantune.fbx", 0.01f);
    ---省略---
    scale.x = scale.y = scale.z = 0.01f;
}
```

実行確認してみましょう。

意図通りのサイズで 3D モデルが表示されていれば OK です。