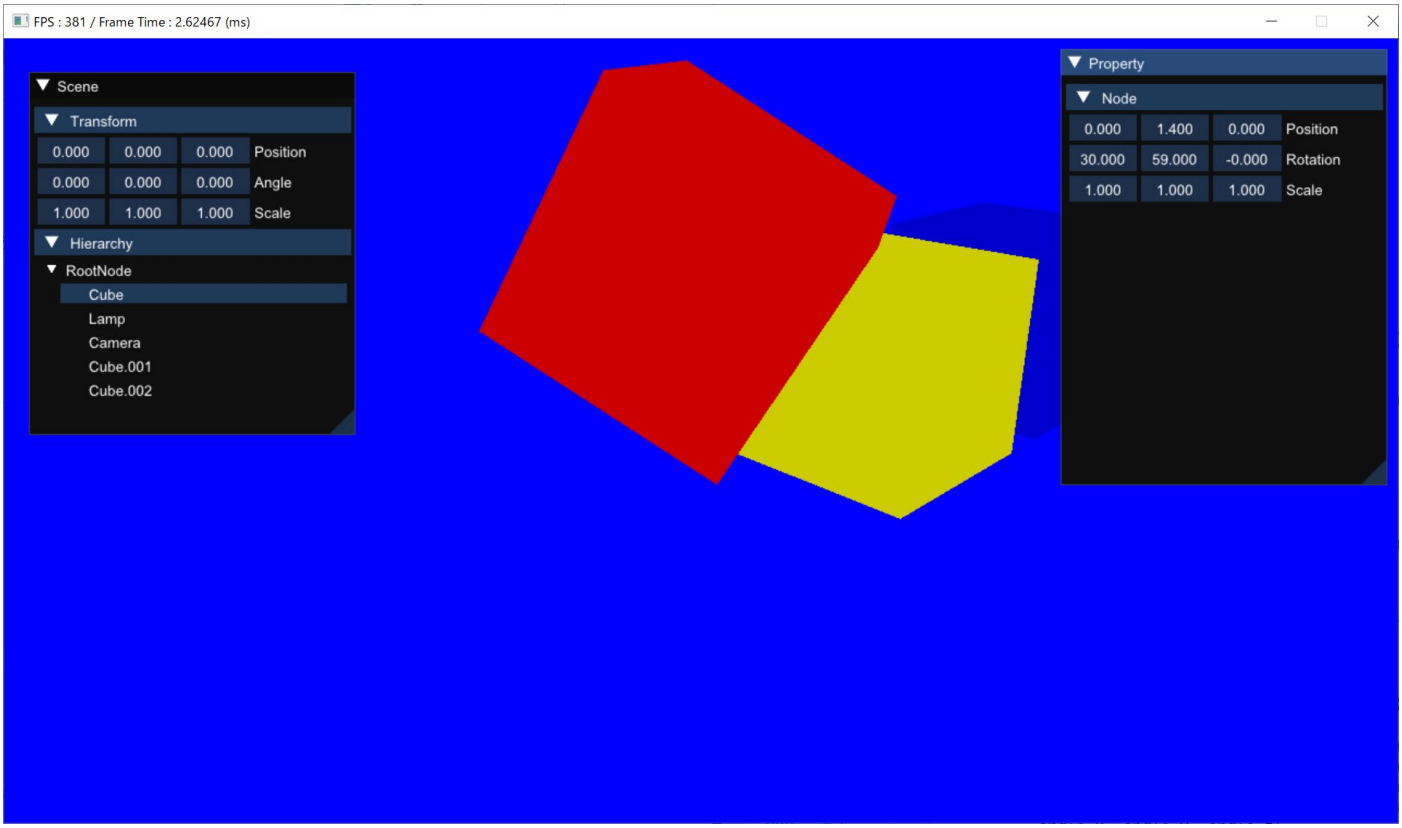
描画エンジン開発 EX

○概要

IMGUI ライブラリでデバッグウインドウを表示し、シーンに表示されているモデルのデータを編集できるようにする。



○トランスフォームデータの編集

シーンに表示されているモデルの位置、回転、スケールデータを編集できるようにします。

Scene.h

```
---省略---

// モデルテストシーン
class ModelTestScene : public Scene
{
    ---省略---

private:
    // シーンGUI描画
    void DrawSceneGUI();

    ---省略---
};
```

Scene.cpp

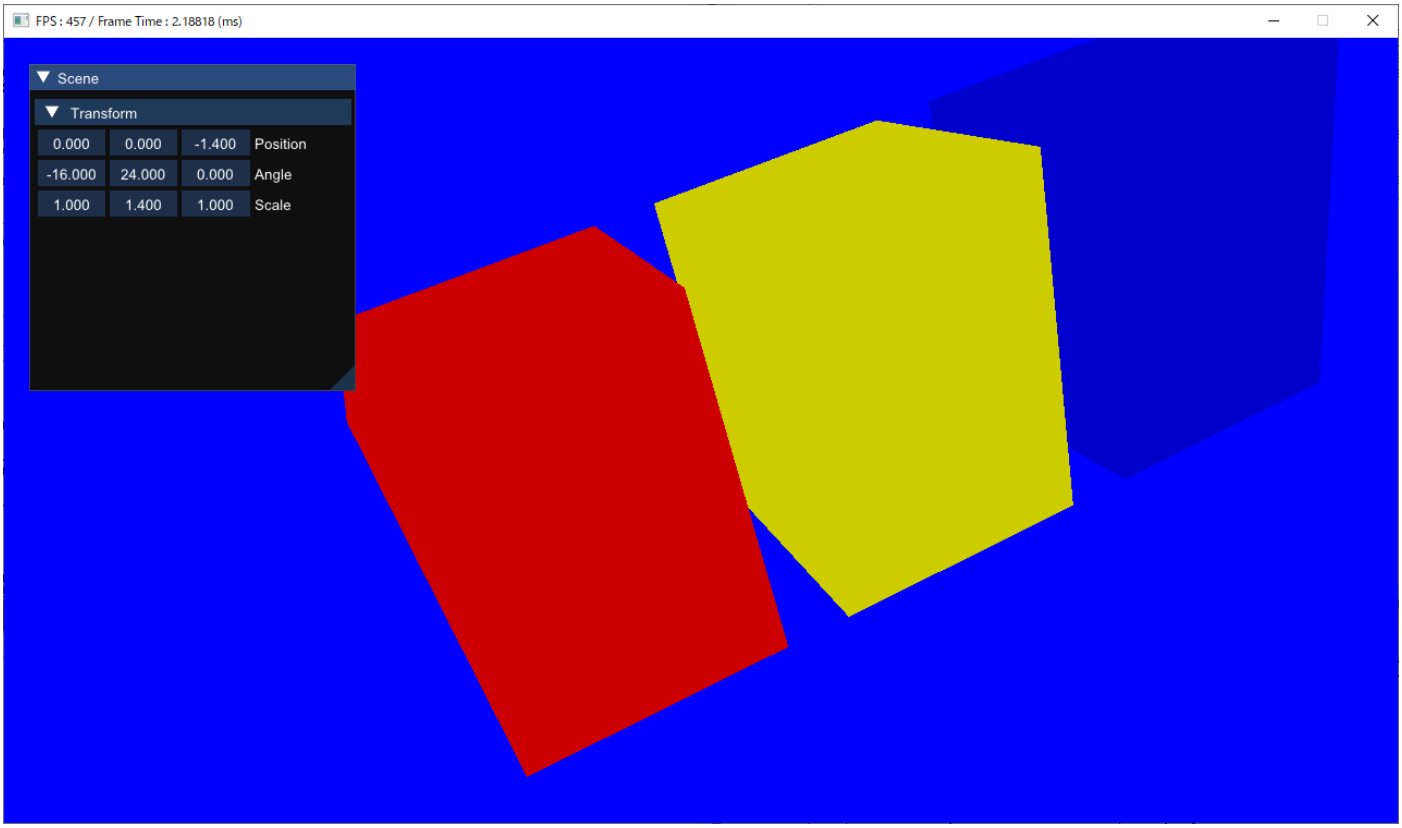
```
#include <imgui.h>
---省略---

// 描画処理
void ModelTestScene::Render(float elapsedTime)
{
    ---省略---

    // デバッグメニュー描画
    DrawSceneGUI();
}

// シーンGUI描画
void ModelTestScene::DrawSceneGUI()
{
    ImVec2 pos = ImGui::GetMainViewport()->GetWorkPos();
    ImGui::SetNextWindowPos(ImVec2(pos.x + 10, pos.y + 10), ImGuiCond_FirstUseEver);
    ImGui::SetNextWindowSize(ImVec2(300, 300), ImGuiCond_FirstUseEver);

    if (ImGui::Begin("Scene", nullptr, ImGuiWindowFlags_None))
    {
        if (ImGui::CollapsingHeader("Transform", ImGuiTreeNodeFlags_DefaultOpen))
        {
            // 位置
            ImGui::DragFloat3("Position", &position.x, 0.1f);

            // 回転
            DirectX::XMFLOAT3 a;
            a.x = DirectX::XMConvertToDegrees(angle.x);
            a.y = DirectX::XMConvertToDegrees(angle.y);
            a.z = DirectX::XMConvertToDegrees(angle.z);
            ImGui::DragFloat3("Angle", &a.x, 1.0f);
            angle.x = DirectX::XMConvertToRadians(a.x);
            angle.y = DirectX::XMConvertToRadians(a.y);
            angle.z = DirectX::XMConvertToRadians(a.z);

            // スケール
            ImGui::DragFloat3("Scale", &scale.x, 0.01f);
        }
        ImGui::End();
    }
}
```

デバッグウインドウを表示する位置とサイズを指定する

ImGui::Begin()からImGui::End()の間にメニューを描く

回転値は見やすいようにラジアンから角度へ変換して表示する

実行確認してみましょう。
デバッグウインドウが表示され、各データを編集してモデルの位置や回転が制御できれば OK です。



○ノードツリー表示

ノードの親子情報をツリー状に表示します。

Model.h

```
---省略---

class Model
{
public:
    ---省略---

    // ルートノード取得
    Node* GetRootNode() { return nodes.data(); }

    ---省略---
};
```

ノード配列の先頭のデータを取得。
Assimp からデータを取得した際、
0 番目のノードがルートになっている

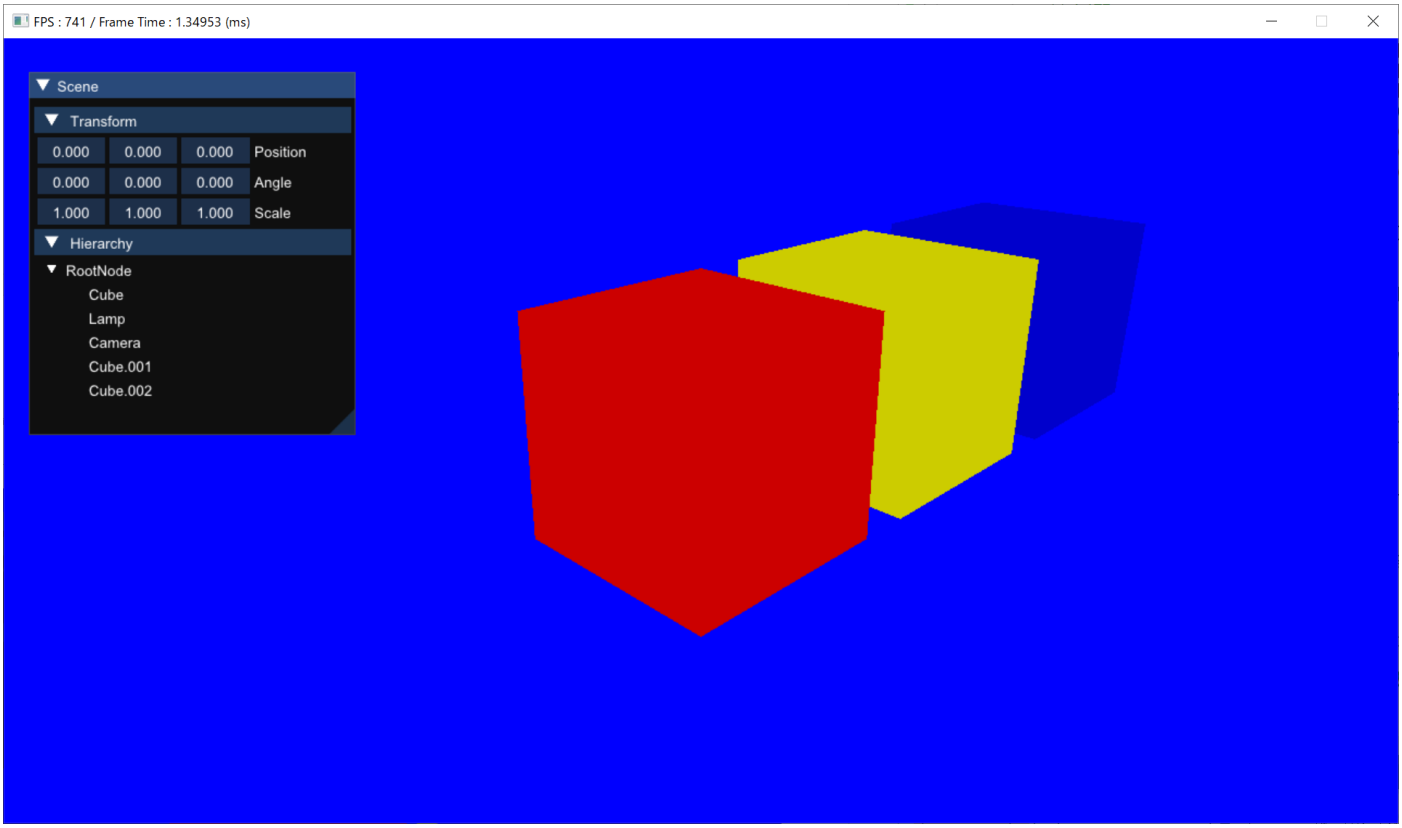
Scene.cpp

```
#include <functional>
---省略---

// シーンGUI描画
void ModelTestScene::DrawSceneGUI()
{
```

```
    ---省略---

    if (ImGui::Begin("Scene", nullptr, ImGuiWindowFlags_None))
    {
        ---省略---

        if (ImGui::CollapsingHeader("Hierarchy", ImGuiTreeNodeFlags_DefaultOpen))
        {
            // ノードツリーを再帰的に描画する関数
            std::function<void(Model::Node*)> drawNodeTree = [&](Model::Node* node)
            {
                // 矢印をクリック、またはノードをダブルクリックで階層を開く
                ImGuiTreeNodeFlags nodeFlags = ImGuiTreeNodeFlags_OpenOnArrow
                    | ImGuiTreeNodeFlags_OpenOnDoubleClick;

                // 子がいない場合は矢印をつけない
                size_t childCount = node->children.size();
                if (childCount == 0)
                {
                    nodeFlags |= ImGuiTreeNodeFlags_Leaf | ImGuiTreeNodeFlags_NoTreePushOnOpen;
                }

                // ツリーノードを表示
                bool opened = ImGui::TreeNodeEx(node, nodeFlags, node->name.c_str());

                // 開かれている場合、子階層も同じ処理を行う
                if (opened && childCount > 0)
                {
                    for (Model::Node* child : node->children)
                    {
                        drawNodeTree(child);
                    }
                    ImGui::TreePop();
                }
            };
            // 再帰的にノードを描画
            drawNodeTree(model->GetRootNode());
        }

        ImGui::End();
    }
}
```

ラムダ式で再帰的に処理する

実行確認してみましょう。
ノードウインドウが表示され、ツリー状にノードが表示されていれば OK です。



○プロパティウインドウ表示

ノードウインドウで選択したノードのパラメータを表示できるようにします。
選択したノードのパラメータをプロパティウインドウに表示し、編集できるようにします。

また、ノードのパラメータの中の回転値はクォータニオンで表現されており、数値だけみてもどういう回転をしているのか想像できないので、メニューに表示するときにはわかりやすいオイラー角に変換して表示します。

トランスフォーム関連の便利関数をまとめる TransformUtils クラスを作成します。
TransformUtils.cpp と TransformUtils.h を作成しましょう。

TransformUtils.h

```
#pragma once

#include <DirectXMath.h>

class TransformUtils
{
public:
    // 行列からヨー、ピッチ、ロールを行列を計算する。
    static bool MatrixToRollPitchYaw(const DirectX::XMFLOAT4X4& m, float& pitch, float& yaw, float& roll);
```

クォータニオンをオイラー角に
変換する関数をつくる

```
    // クォータニオンからヨー、ピッチ、ロールを行列を計算する。
    static bool QuaternionToRollPitchYaw(const DirectX::XMFLOAT4& q, float& pitch, float& yaw, float& roll);
};
```

TransformUtils.cpp

```
#include "TransformUtils.h"

// 行列からヨー、ピッチ、ロールを行列を計算する。
bool TransformUtils::MatrixToRollPitchYaw(const DirectX::XMFLOAT4X4& m, float& pitch, float& yaw, float& roll)
{
    float xRadian = asinf(-m._32);
    pitch = xRadian;
    if (xRadian < DirectX::XM_PI / 2.0f)
    {
        if (xRadian > -DirectX::XM_PI / 2.0f)
        {
            roll = atan2f(m._12, m._22);
            yaw = atan2f(m._31, m._33);
            return true;
        }
        else
        {
            roll = (float)-atan2f(m._13, m._11);
            yaw = 0.0f;
            return false;
        }
    }
    else
    {
        roll = (float)atan2f(m._13, m._11);
        yaw = 0.0f;
        return false;
    }
}

// クォータニオンからヨー、ピッチ、ロールを行列を計算する。
bool TransformUtils::QuaternionToRollPitchYaw(const DirectX::XMFLOAT4& q, float& pitch, float& yaw, float& roll)
{
    DirectX::XMVECTOR Q = DirectX::XMLoadFloat4(&q);
    DirectX::XMMATRIX M = DirectX::XMMatrixRotationQuaternion(Q);
    DirectX::XMFLOAT4X4 m;
    DirectX::XMStoreFloat4x4(&m, M);
    return MatrixToRollPitchYaw(m, pitch, yaw, roll);
}
```

Scene.h

```
---省略---

// モデルテストシーン
class ModelTestScene : public Scene
{
    ---省略---
```

```
private:
    ---省略---

    // プロパティGUI描画
    void DrawPropertyGUI();

    ---省略---
    Model::Node*      selectionNode = nullptr;
};
```

選択したノードのポインタを保持する変数

Scene.cpp

```
---省略---
#include "TransformUtils.h"

---省略---

// 描画処理
void ModelTestScene::Render(float elapsedTime)
{
    ---省略---

    // デバッグメニュー描画
    ---省略---
    DrawPropertyGUI();
}

// シーンGUI描画
void ModelTestScene::DrawSceneGUI()
{
    ---省略---

    if (ImGui::Begin("Scene", nullptr, ImGuiWindowFlags_None))
    {
        ---省略---

        if (ImGui::CollapsingHeader("Hierarchy", ImGuiTreeNodeFlags_DefaultOpen))
        {
            // ノードツリーを再帰的に描画する関数
            std::function<void(Model::Node*)> drawNodeTree = [&](Model::Node* node)
            {
                ---省略---

                // 選択フラグ
                if (selectionNode == node)
                {
                    nodeFlags |= ImGuiTreeNodeFlags_Selected;
                }

                // ツリーノードを表示
                bool opened = ImGui::TreeNodeEx(node, nodeFlags, node->name.c_str());

                // フォーカスされたノードを選択する
                if (ImGui::IsItemFocused())
```

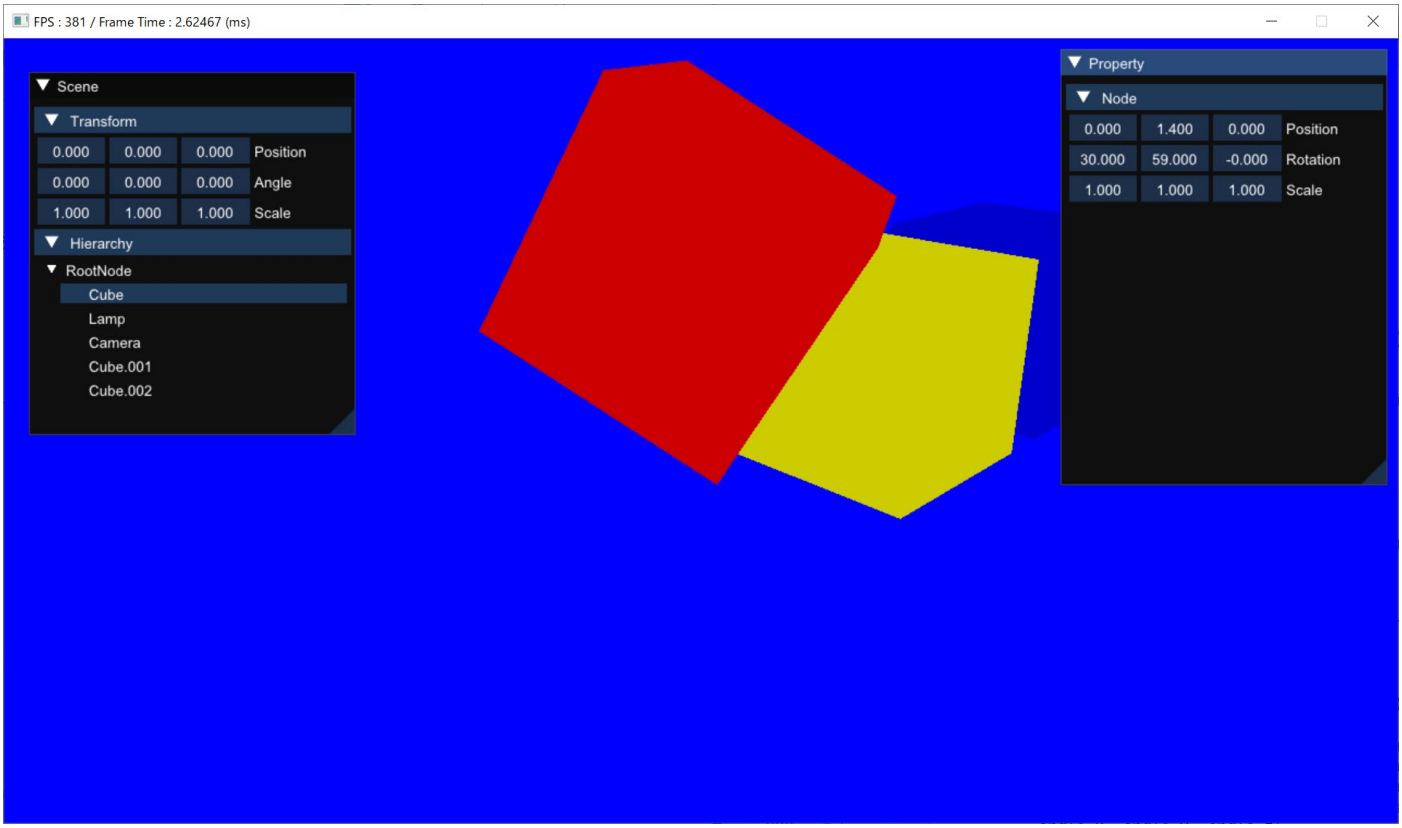
```
                    {
                        selectionNode = node;
                    }

                    ---省略---
                };
                ---省略---
            }
            ---省略---
        }
}

// プロパティGUI描画
void ModelTestScene::DrawPropertyGUI()
{
    ImVec2 pos = ImGui::GetMainViewport()->GetWorkPos();
    ImGui::SetNextWindowPos(ImVec2(pos.x + 970, pos.y + 10), ImGuiCond_FirstUseEver);
    ImGui::SetNextWindowSize(ImVec2(300, 300), ImGuiCond_FirstUseEver);

    ImGui::Begin("Property", nullptr, ImGuiWindowFlags_None);

    if (selectionNode != nullptr)
    {
        if (ImGui::CollapsingHeader("Node", ImGuiTreeNodeFlags_DefaultOpen))
        {
            // 位置
            ImGui::DragFloat3("Position", &selectionNode->position.x, 0.1f);

            // 回転
            DirectX::XMFLOAT3 angle;
            TransformUtils::QuaternionToRollPitchYaw(selectionNode->rotation, angle.x, angle.y, angle.z);
            angle.x = DirectX::XMConvertToDegrees(angle.x);
            angle.y = DirectX::XMConvertToDegrees(angle.y);
            angle.z = DirectX::XMConvertToDegrees(angle.z);
            if (ImGui::DragFloat3("Rotation", &angle.x, 1.0f))
            {
                angle.x = DirectX::XMConvertToRadians(angle.x);
                angle.y = DirectX::XMConvertToRadians(angle.y);
                angle.z = DirectX::XMConvertToRadians(angle.z);
                DirectX::XMVECTOR Rotation = DirectX::XMQuaternionRotationRollPitchYaw(
                                                    angle.x, angle.y, angle.z);
                DirectX::XMStoreFloat4(&selectionNode->rotation, Rotation);
            }

            // スケール
            ImGui::DragFloat3("Scale", &selectionNode->scale.x, 0.01f);
        }
    }

    ImGui::End();
}
```

クォータニオンをオイラー角に変換

ラジアンを角度に変換

編集された場合に
オイラー角をクォータニオンに
戻して設定

実行確認してみましょう。
ノードを選択し、プロパティウインドウでデータを編集できれば OK です。



○フリーカメラ操作

マウス操作でカメラを自由に動かせるようにします。
カメラの動きを制御する FreeCameraController クラスを作成します。
FreeCameraController.cpp と FreeCameraController.h を作成しましょう。

FreeCameraController.h

```
#pragma once
#include "Camera.h"

class FreeCameraController
{
public:
    // カメラからコントローラーへパラメータを同期する
    void SyncCameraToController(const Camera& camera);

    // コントローラーからカメラへパラメータを同期する
    void SyncControllerToCamera(Camera& camera);

    // 更新処理
    void Update();

private:
    DirectX::XMFLOAT3    eye;
    DirectX::XMFLOAT3    focus;
    DirectX::XMFLOAT3    up;
```

```
    DirectX::XMFLOAT3     right;
    float                 distance;

    float                 angleX;
    float                 angleY;
};
```

FreeCameraController.cpp

```
#include <imgui.h>
#include "FreeCameraController.h"

// カメラからコントローラーへパラメータを同期する
void FreeCameraController::SyncCameraToController(const Camera& camera)
{
    eye = camera.GetEye();
    focus = camera.GetFocus();
    up = camera.GetUp();
    right = camera.GetRight();

    // 視点から注視点までの距離を算出
    DirectX::XMVECTOR Eye = DirectX::XMLoadFloat3(&eye);
    DirectX::XMVECTOR Focus = DirectX::XMLoadFloat3(&focus);
    DirectX::XMVECTOR Vec = DirectX::XMVectorSubtract(Focus, Eye);
    DirectX::XMVECTOR Distance = DirectX::XMVector3Length(Vec);
    DirectX::XMStoreFloat(&distance, Distance);

    // 回転角度を算出
    const DirectX::XMFLOAT3& front = camera.GetFront();
    angleX = ::asinf(-front.y);
    if (up.y < 0)
    {
        if (front.y > 0)
        {
            angleX = -DirectX::XM_PI - angleX;
        }
        else
        {
            angleX = DirectX::XM_PI - angleX;
        }
        angleY = ::atan2f(front.x, front.z);
    }
    else
    {
        angleY = ::atan2f(-front.x, -front.z);
    }

}

// コントローラーからカメラへパラメータを同期する
void FreeCameraController::SyncControllerToCamera(Camera& camera)
{
    camera.SetLookAt(eye, focus, up);
}
```

```
// 更新処理
void FreeCameraController::Update()
{
    // デバッグウインドウ操作中は処理しない
    if (ImGui::IsWindowFocused(ImGuiFocusedFlags_AnyWindow))
    {
        return;
    }

    // IMGUIのマウス入力値を使ってカメラ操作する
    ImGuiIO io = ImGui::GetIO();

    // マウスカーソルの移動量を求める
    float moveX = io.MouseDelta.x * 0.02f;
    float moveY = io.MouseDelta.y * 0.02f;

    // マウス左ボタン押下中
    if (io.MouseDown[ImGuiMouseButton_Left])
    {
        // Y軸回転
        angleY += moveX * 0.5f;
        if (angleY > DirectX::XM_PI)
        {
            angleY -= DirectX::XM_2PI;
        }
        else if (angleY < -DirectX::XM_PI)
        {
            angleY += DirectX::XM_2PI;
        }
        // X軸回転
        angleX += moveY * 0.5f;
        if (angleX > DirectX::XM_PI)
        {
            angleX -= DirectX::XM_2PI;
        }
        else if (angleX < -DirectX::XM_PI)
        {
            angleX += DirectX::XM_2PI;
        }
    }
    // マウス中ボタン押下中
    else if (io.MouseDown[ImGuiMouseButton_Middle])
    {
        // 平行移動
        float s = distance * 0.035f;
        float x = moveX * s;
        float y = moveY * s;

        focus.x -= right.x * x;
        focus.y -= right.y * x;
        focus.z -= right.z * x;

        focus.x += up.x * y;
        focus.y += up.y * y;
        focus.z += up.z * y;
    }
    // マウス右ボタン押下中
```

```
    else if (io.MouseDown[ImGuiMouseButton_Right])
    {
        // ズーム
        distance += (-moveY - moveX) * distance * 0.1f;
    }
    // マウスホイール
    else if (io.MouseWheel != 0)
    {
        // ズーム
        distance -= io.MouseWheel * distance * 0.1f;
    }

    float sx = ::sinf(angleX);
    float cx = ::cosf(angleX);
    float sy = ::sinf(angleY);
    float cy = ::cosf(angleY);

    // カメラの方向を算出
    DirectX::XMVECTOR Front = DirectX::XMVectorSet(-cx * sy, -sx, -cx * cy, 0.0f);
    DirectX::XMVECTOR Right = DirectX::XMVectorSet(cy, 0, -sy, 0.0f);
    DirectX::XMVECTOR Up = DirectX::XMVector3Cross(Right, Front);
    // カメラの視点＆注視点を算出
    DirectX::XMVECTOR Focus = DirectX::XMLoadFloat3(&focus);
    DirectX::XMVECTOR Distance = DirectX::XMVectorSet(distance, distance, distance, 0.0f);
    DirectX::XMVECTOR Eye = DirectX::XMVectorSubtract(Focus, DirectX::XMVectorMultiply(Front, Distance));
    // ビュー行列からワールド行列を算出
    DirectX::XMMATRIX View = DirectX::XMMatrixLookAtLH(Eye, Focus, Up);
    DirectX::XMMATRIX World = DirectX::XMMatrixTranspose(View);
    // ワールド行列から方向を算出
    Right = DirectX::XMVector3TransformNormal(DirectX::XMVectorSet(1, 0, 0, 0), World);
    Up = DirectX::XMVector3TransformNormal(DirectX::XMVectorSet(0, 1, 0, 0), World);
    // 結果を格納
    DirectX::XMStoreFloat3(&eye, Eye);
    DirectX::XMStoreFloat3(&up, Up);
    DirectX::XMStoreFloat3(&right, Right);
}
```

Scene.h

```
---省略---
#include "FreeCameraController.h"
---省略---

// モデルテストシーン
class ModelTestScene : public Scene
{
    ---省略---

private:
    ---省略---
    FreeCameraController       cameraController;
};
```

Scene.cpp

```
---省略---

// コンストラクタ
ModelTestScene::ModelTestScene()
{
    ---省略---
    cameraController.SyncCameraToController(camera);
}

// 描画処理
void ModelTestScene::Render(float elapsedTime)
{
    // カメラ更新処理
    cameraController.Update();
    cameraController.SyncControllerToCamera(camera);

    ---省略---
}
```

実行確認してみましょう。
マウスでカメラを移動、回転、ズーム操作ができれば OK です。