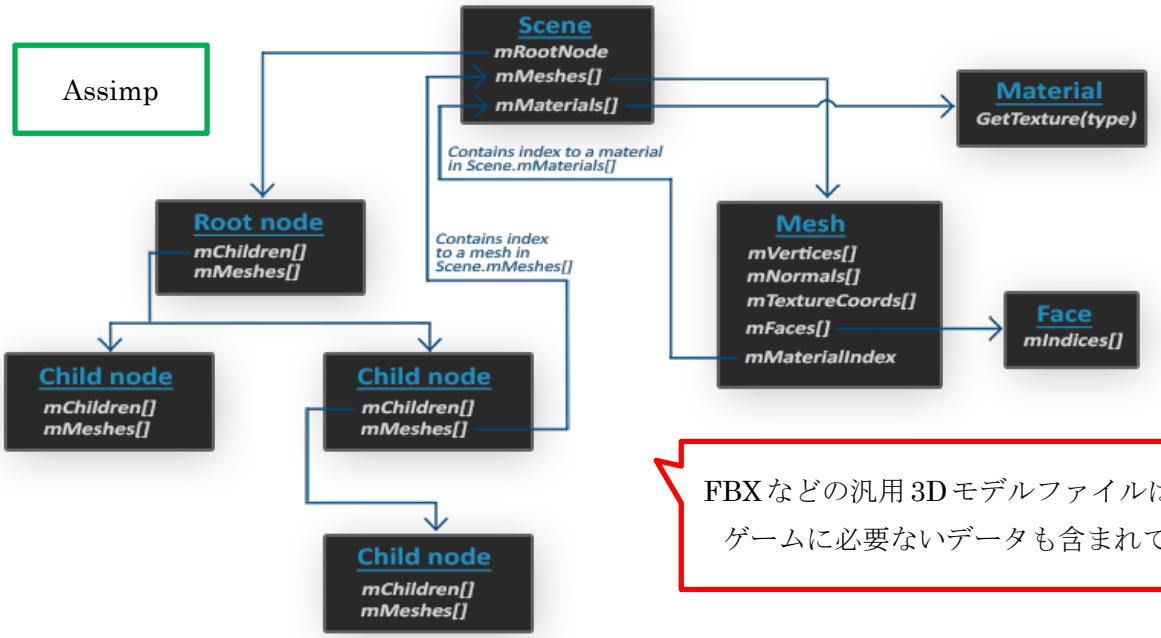
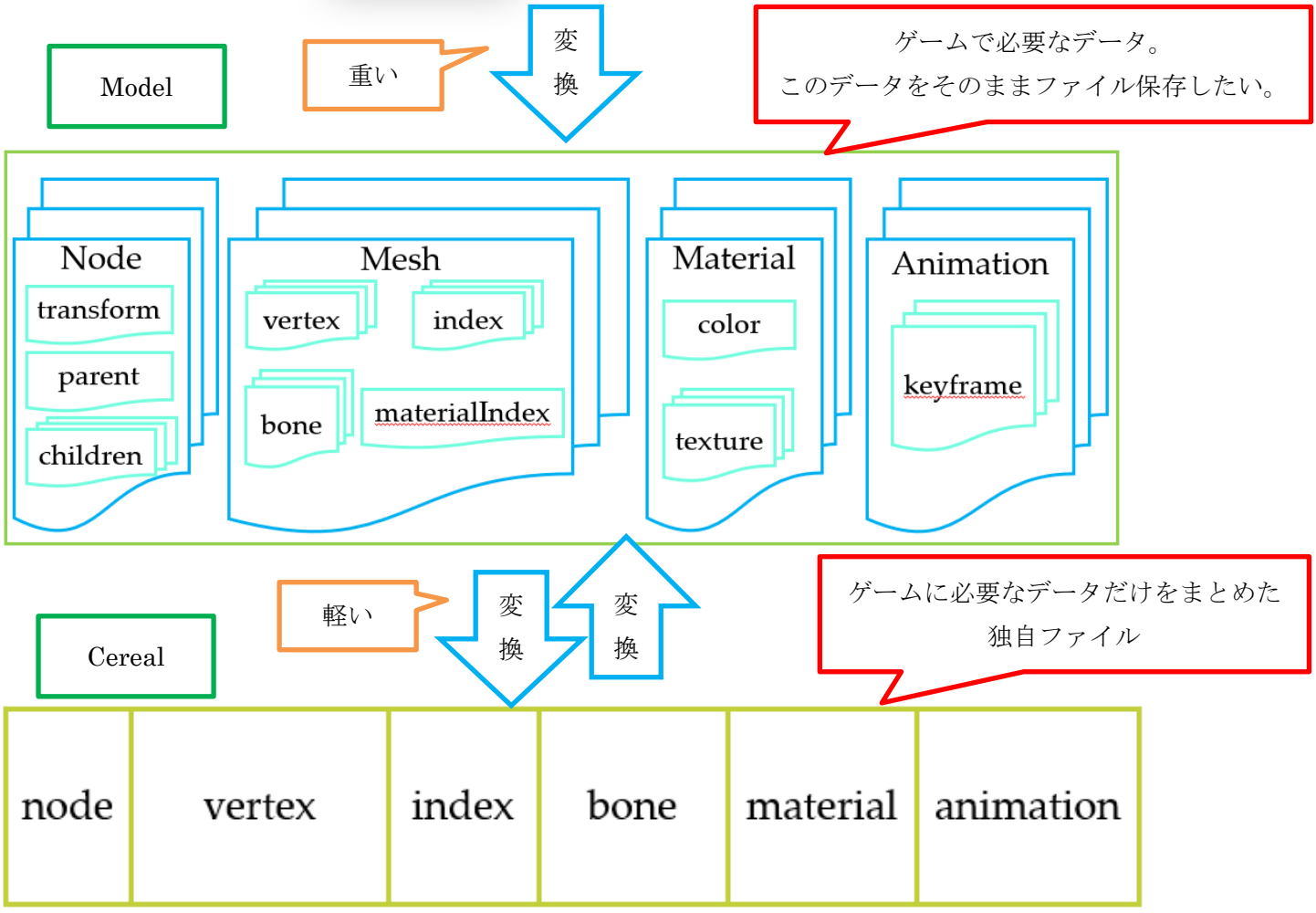
描画エンジン開発 EX

○概要

独自形式の 3D モデルデータファイルを作成し、ファイル読み込みを高速化する。



FBX などの汎用 3D モデルファイルは読み込みが遅く、
ゲームに必要ないデータも含まれてしまっている。



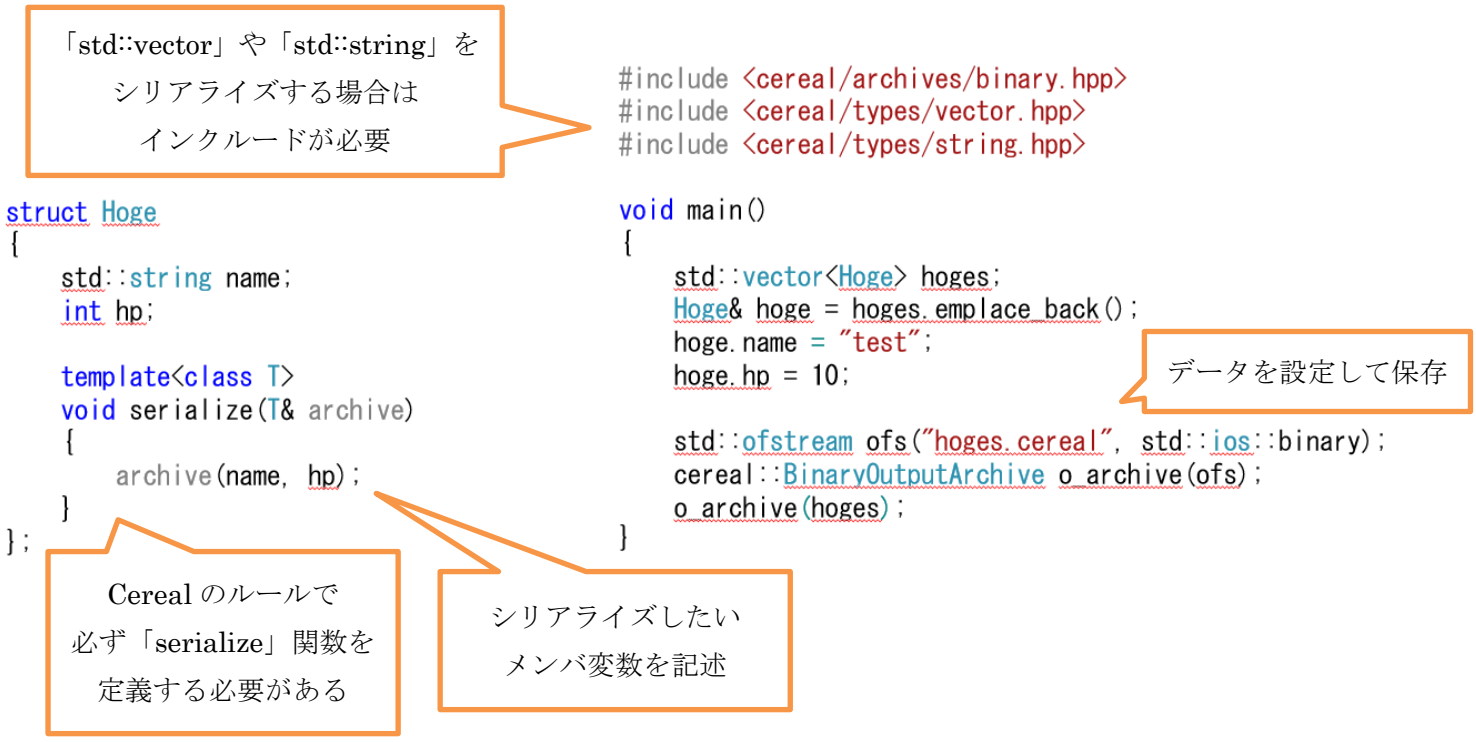
○シリアライズ

複雑なデータを単純化し保存することをシリアライズと呼びます。
また、シリアライズされたデータを復元することをデシリアライズと呼びます。

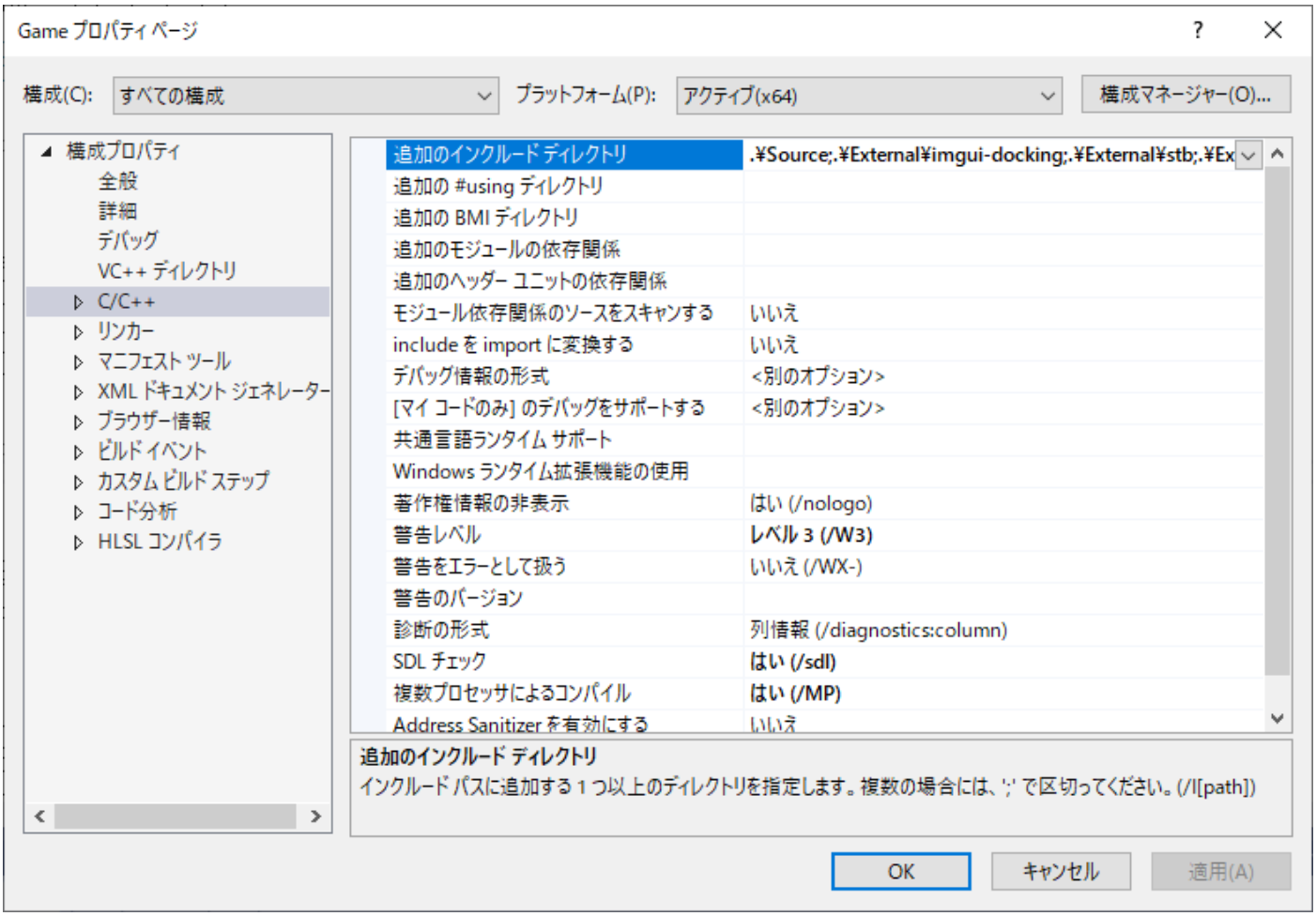
○Cereal

C++11 用のシリアライズライブラリ。
比較的簡単に導入でき、処理速度や扱いやすさが優れているため今回はこのライブラリを使用してシリアライズ処理を実装します。

Cereal ライブラリを利用してシリアライズする流れ。



VisualStudio のプロジェクト設定で左上の「構成」を「すべての構成」に設定し、「C/C++」を選択し、「追加のインクルードディレクトリを編集しましょう。



以下のライブラリパスを追加します。

.¥External¥cereal¥include

○モデルデータのシリアライズ

Model クラスの保存したいデータ型に serialize 関数を追加します。

Model.h

```
---省略---

class Model
{
public:
    ---省略---

    struct Node
    {
        ---省略---

        template<class Archive>
        void serialize(Archive& archive);
    };
```

Cereal のルールでシリアライズしたい場合は serialize という関数名でこのように記述する

```
struct Material
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct Vertex
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct Bone
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct Mesh
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct VectorKeyframe
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct QuaternionKeyframe
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};

struct NodeAnim
{
    ---省略---

    template<class Archive>
    void serialize(Archive& archive);
};
struct Animation
```

```
    {
        ---省略---

        template<class Archive>
        void serialize(Archive& archive);
    };

    ---省略---

private:
    ---省略---

    // シリアライズ
    void Serialize(const char* filename);

    // デシリアライズ
    void Deserialize(const char* filename);

    ---省略---
};
```

Model.cpp

```
---省略---
#include <fstream>
#include <cereal/cereal.hpp>
#include <cereal/archives/binary.hpp>
#include <cereal/types/string.hpp>
#include <cereal/types/vector.hpp>
---省略---

namespace DirectX
{
    template<class Archive>
    void serialize(Archive& archive, XMUINT4& v)
    {
        archive(
            cereal::make_nvp("x", v.x),
            cereal::make_nvp("y", v.y),
            cereal::make_nvp("z", v.z),
            cereal::make_nvp("w", v.w)
        );
    }

    template<class Archive>
    void serialize(Archive& archive, XMFLOAT2& v)
    {
        archive(
            cereal::make_nvp("x", v.x),
            cereal::make_nvp("y", v.y)
        );
    }

    template<class Archive>
    void serialize(Archive& archive, XMFLOAT3& v)
```

XMFLOAT3 などの構造体に
serialize 関数を定義できないので
Cereal のルールとしてこのように定義する

```
    {
        archive(
            cereal::make_nvp("x", v.x),
            cereal::make_nvp("y", v.y),
            cereal::make_nvp("z", v.z)
        );
    }

    template<class Archive>
    void serialize(Archive& archive, XMFLOAT4& v)
    {
        archive(
            cereal::make_nvp("x", v.x),
            cereal::make_nvp("y", v.y),
            cereal::make_nvp("z", v.z),
            cereal::make_nvp("w", v.w)
        );
    }

    template<class Archive>
    void serialize(Archive& archive, XMFLOAT4X4& m)
    {
        archive(
            cereal::make_nvp("_11", m._11),
            cereal::make_nvp("_12", m._12),
            cereal::make_nvp("_13", m._13),
            cereal::make_nvp("_14", m._14),
            cereal::make_nvp("_21", m._21),
            cereal::make_nvp("_22", m._22),
            cereal::make_nvp("_23", m._23),
            cereal::make_nvp("_24", m._24),
            cereal::make_nvp("_31", m._31),
            cereal::make_nvp("_32", m._32),
            cereal::make_nvp("_33", m._33),
            cereal::make_nvp("_34", m._34),
            cereal::make_nvp("_41", m._41),
            cereal::make_nvp("_42", m._42),
            cereal::make_nvp("_43", m._43),
            cereal::make_nvp("_44", m._44)
        );
    }
}

template<class Archive>
void Model::Node::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(name),
        CEREAL_NVP(path),
        CEREAL_NVP(parentIndex),
        CEREAL_NVP(position),
        CEREAL_NVP(rotation),
        CEREAL_NVP(scale)
    );
}

template<class Archive>
```

保存する必要のあるデータだけ記述する。
行列データなどはこれらのデータから
計算されるので保存する必要なし。

```
void Model::Material::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(name),
        CEREAL_NVP(diffuseTextureFileName),
        CEREAL_NVP(normalTextureFileName),
        CEREAL_NVP(color)
    );
}

template<class Archive>
void Model::Vertex::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(position),
        CEREAL_NVP(boneWeight),
        CEREAL_NVP(boneIndex),
        CEREAL_NVP(texcoord),
        CEREAL_NVP(normal),
        CEREAL_NVP(tangent)
    );
}

template<class Archive>
void Model::Bone::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(nodeIndex),
        CEREAL_NVP(offsetTransform)
    );
}

template<class Archive>
void Model::Mesh::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(vertices),
        CEREAL_NVP(indices),
        CEREAL_NVP(bones),
        CEREAL_NVP(nodeIndex),
        CEREAL_NVP(materialIndex)
    );
}

template<class Archive>
void Model::VectorKeyframe::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(seconds),
        CEREAL_NVP(value)
    );
}

template<class Archive>
void Model::QuaternionKeyframe::serialize(Archive& archive)
{
    archive(
```

```
        CEREAL_NVP(seconds),
        CEREAL_NVP(value)
    );
}

template<class Archive>
void Model::NodeAnim::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(positionKeyframes),
        CEREAL_NVP(rotationKeyframes),
        CEREAL_NVP(scaleKeyframes)
    );
}

template<class Archive>
void Model::Animation::serialize(Archive& archive)
{
    archive(
        CEREAL_NVP(name),
        CEREAL_NVP(secondsLength),
        CEREAL_NVP(nodeAnims)
    );
}

// シリアライズ
void Model::Serialize(const char* filename)
{
    std::ofstream ostream(filename, std::ios::binary);
    if (ostream.is_open())
    {
        cereal::BinaryOutputArchive archive(ostream);

        try
        {
            archive(
                CEREAL_NVP(nodes),
                CEREAL_NVP(materials),
                CEREAL_NVP(meshes),
                CEREAL_NVP(animations)
            );
        }
        catch (...)
        {
            _ASSERT_EXPR_A(false, "Model serialize failed.");
        }
    }
}

// デシリアライズ
void Model::Deserialize(const char* filename)
{
    std::ifstream istream(filename, std::ios::binary);
    if (istream.is_open())
    {
        cereal::BinaryInputArchive archive(istream);
```

データ保存処理

```
        try
        {
            archive(
                CEREAL_NVP(nodes),
                CEREAL_NVP(materials),
                CEREAL_NVP(meshes),
                CEREAL_NVP(animations)
            );
        }
        catch (...)
        {
            _ASSERT_EXPR_A(false, "Model deserialize failed.");
        }
    }
    else
    {
        _ASSERT_EXPR_A(false, "Model File not found.");
    }
}
```

データ読み込み処理

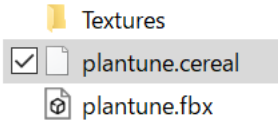
Model.cpp

```
---省略---

// コンストラクタ
Model::Model(ID3D11Device* device, const char* filename)
{
    std::filesystem::path filepath(filename);
    std::filesystem::path dirpath(filepath.parent_path());

    // 独自形式のモデルファイルの存在確認
    filepath.replace_extension(".cereal");
    if (std::filesystem::exists(filepath))
    {
        // 独自形式のモデルファイルの読み込み
        Deserialize(filepath.string().c_str());
    }
    else
    {
        // 汎用モデルファイルの読み込み
        AssimpImporter importer(filename);

        // マテリアルデータ読み取り
        importer.LoadMaterials(materials);

        // ノードデータ読み取り
        importer.LoadNodes(nodes);

        // メッシュデータ読み取り
        importer.LoadMeshes(meshes, nodes);

        // アニメーションデータ読み取り
        importer.LoadAnimations(animations, nodes);

        // 独自形式のモデルファイルを保存
```

```
        Serialize(filepath.string().c_str());
    }

    ---省略---
}
```

次回以降は独自形式ファイルで
読み込めるように保存

実行確認してみましょう。
初回に実行すると FBX ファイルを読み込み後に独自形式ファイル（.cereal）が出力されていれば OK です。



もう一度実行確認してみましょう。
二回目以降は独自形式ファイル(.cereal)を読み込むことになるので、初回と比べて読み込み時間が短縮されたことが体感できるはずです。

今後、データを拡張してデータ構造が変わった場合などは古いファイルは使えなくなるので、トラブルが起きた場合は古いファイルを消して新しくファイルを作り直すようにしましょう。

```
    struct Material
    {
        std::string        name;
        std::string        diffuseTextureFileName;
        std::string        normalTextureFileName;
        std::string        emissionTextureFileName;
        DirectX::XMFLOAT4 color = { 1, 1, 1, 1 };

        Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> diffuseMap;
        Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> normalMap;
        Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> emissionMap;

        template<class Archive>
        void serialize(Archive& archive);
    };
```

新しくデータが増えた場合は
古いファイルは使えない

削除して作り直す