Chapter 2　SkinModel クラス

2.1 冗長性の除去

　Chapter 1 では、DirectX::ModelPtr クラスを使って、複数のモデルを表示していましたが、下記のように少々冗長なコードが多くなってしまいました。

初期化処理で冗長な部分

```
DirectX::EffectFactory effectFactory(g_graphicsEngine->GetD3DDevice());
//テクスチャがあるフォルダを設定する。
effectFactory.SetDirectory(L"Assets/modelData");
//CMOファイルからモデルを作成する関数の、CreateFromCMOを実行する。
g_teapotModel = DirectX::Model::CreateFromCMO(
   g_graphicsEngine->GetD3DDevice(),   //第一引数はD3Dデバイス。
   L"Assets/modelData/teapot.cmo",     //第二引数は読み込むCMOファイルのファイルパス。
   effectFactory,                      //第三引数はエフェクトファクトリ。
   false                              //第四引数はCullモード。気にしなくてよい。
);

//Hands-On 2 ユニティちゃんを表示するためにDirectX::ModelPtrのインスタンスを作成する。
g_unityChanModel = DirectX::Model::CreateFromCMO(
   g_graphicsEngine->GetD3DDevice(),   //第一引数はD3Dデバイス。
   L"Assets/modelData/unityChan.cmo",  //第二引数は読み込むCMOファイルのファイルパス。
   effectFactory,                      //第三引数はエフェクトファクトリ。
   false                              //第四引数はCullモード。気にしなくてよい。
);

g_starM
odel = DirectX::Model::CreateFromCMO(
   g_graphicsEngine->GetD3DDevice(),   //第一引数はD3Dデバイス。
   L"Assets/modelData/star.cmo",       //第二引数は読み込むCMOファイルのファイルパス。
   effectFactory,                      //第三引数はエフェクトファクトリ。
   false
);
```

描画処理で冗長な部分

```
g_teapotModel->Draw(
    g_graphicsEngine->GetD3DDeviceContext(),//D3Dデバイスコンテキスト。
    state,                  //レンダリングステート。今は気にしなくてよい。
    g_teapotWorldMatrix,     //ワールド行列。
    g_viewMatrix,           //ビュー行列。
    g_projMatrix            //プロジェクション行列。
  );

    .
    .
    .

//Hands-On 3 ユニティちゃんを表示するためにDirectX::ModelPtrのDraw関数を呼び出す。
g_unityChanModel->Draw(
  g_graphicsEngine->GetD3DDeviceContext(),//D3Dデバイスコンテキスト。
  state,                  //レンダリングステート。今は気にしなくてよい。
  g_unityChanWorldMatrix,     //ワールド行列。
  g_viewMatrix,           //ビュー行列。
  g_projMatrix            //プロジェクション行列。
);

    .
    .
    .

//星の描画
g_starModel->Draw(
  g_graphicsEngine->GetD3DDeviceContext(),//D3Dデバイスコンテキスト。
  state,                  //レンダリングステート。今は気にしなくてよい。
  g_starWorldMatrix,       //ワールド行列。
  g_viewMatrix,           //ビュー行列。
  g_projMatrix            //プロジェクション行列。
);
```

黄色に着色された箇所は冗長になっており、同じことを繰り返ししています。

このチャプターでは、DirectX::ModelPtr をもっと簡単に使えるようにするため、薄くラップした SkinModel クラスを作成しています。では、さっそく SkinModel クラスの利用の仕方を見ていきましょう。

2.2 SkinModel クラスの使い方

では、SkinModel クラスを利用する方法を見ていきましょう。

main.cpp のモデル表示処理で SkinModel クラスを使うように書き換えています。まず、SkinModel クラスのインスタンスを定義する必要があります。

Lesson_02/main.cpp(4 行目)

```
//////////////////////////////////////////////////////////////
// グローバル変数。
//////////////////////////////////////////////////////////////
HWND     g_hWnd = NULL;         //ウィンドウハンドル。
GraphicsEngine* g_graphicsEngine = NULL;  //グラフィックスエンジン。

SkinModel g_teapotModel;            //ティーポットモデル。

CMatrix g_viewMatrix = CMatrix::Identity();    //ビュー行列。
CMatrix g_projMatrix = CMatrix::Identity();    //プロジェクション行列。
CMatrix g_worldMatrix = CMatrix::Identity();//ワールド行列。
```

グローバル変数に g_teapotModel が追加されていることを確認してください。

続いて、モデルの読み込み処理です。

Lesson_02/main.cpp(137 行目)

```
//////////////////////////////////////////////////////
//ティーポットモデルの初期化。
g_teapotModel.Init(L"Resource/modelData/teapot.cmo");
```

読み込む cmo ファイルのパスを指定しています。

最後に描画処理です。

main.cpp(92 行目)

```
//////////////////////////////////////////
//ここからモデル表示のプログラム。
//3Dモデルを描画する。
g_teapotModel.Draw(
  g_viewMatrix,              //ビュー行列。
  g_projMatrix               //プロジェクション行列。
);
```

これで teapot モデルが表示されます。

2.3 Hands-On ユニティちゃんを表示してみる。

Step-1. Assets/modelData/UnityChan.FBX を Visual Studio に追加する。

Step-2. ユニティちゃんを表示するための変数を追加。

```
//Hands-On 1 ユニティちゃんを表示するための変数を追加。
SkinModel g_unityChanModel;
```

Step-3. cmo ファイルをロードする。

```
//Hands-On 2 cmoファイルをロードする。
g_unityChanModel.Init(L"Assets/modelData/unityChan.cmo");
```

Step-4 ユニティちゃんを表示するために、SkinModel の Draw 関数を呼び出す。

```
//Hands-On 3 ユニティちゃんを表示するために、SkinModelのDraw関数を呼び出す。
g_unityChanModel.Draw(
  g_viewMatrix,
  g_projMatrix
);
```

Step- 5. ユニティちゃんの表示位置を変更する。

```
//Hands-On 5 ユニティちゃんを表示するために、SkinModelのDraw関数を呼び出す。
CQuaternion unityRot;
CVector3 unityScale = { 1.0f, 1.0f, 1.0f };

CVector3 unityPos;
unityPos.x = 200.0f;  //左に動かす。
unityPos.y = 0.0f;
unityPos.z = 0.0f;

g_unityChanModel.UpdateWorldMatrix(unityPos, unityRot, unityScale);
```

Step-6. ユニティちゃんを回転させる。

```
CQuaternion unityRot;

unityRot.SetRotationDegX(-90.0f);
```

2.4 SkinModel のクラスの実装

では、SkinModel クラスの実装を見ていきましょう。

Lesson02/SkinModel.h

```
/*!
*@brief スキンモデルクラス。
*/
class SkinModel
{
public:
  /*!
  *@brief 初期化。
  *@param[in]  filePath     ロードするcmoファイルのファイルパス。
  */
  void Init(const wchar_t* filePath);
  /*!
  *@brief モデルをワールド座標系に変換するためのワールド行列を更新する。
  *@param[in]  position  モデルの座標。
  *@param[in]  rotation  モデルの回転。
  *@param[in]  scale   モデルの拡大率。
  */
  void UpdateWorldMatrix(CVector3 position, CQuaternion rotation, CVector3 scale);
  /*!
  *@brief モデルを描画。
  *@param[in]  viewMatrix   カメラ行列。
  *  ワールド座標系の3Dモデルをカメラ座標系に変換する行列です。
  *@param[in]  projMatrix   プロジェクション行列。
  *  カメラ座標系の3Dモデルをスクリーン座標系に変換する行列です。
  */
  void Draw( CMatrix viewMatrix, CMatrix projMatrix );
private:
  CMatrix                  m_worldMatrix;  //!<ワールド行列。
  DirectX::ModelPtr        m_modelDx;    //!<DirectXTKが提供するモデルクラス。
};
```

メンバ関数として、cmo ファイルをロードして SkinModel を初期化する Init 関数、ワールド行列を更新する UpdateWorldMatrix 関数、モデルを描画する Draw 関数が宣言されています。では、続いて各メンバ関数の定義を見ていきましょう。

2.5 SkinModel::Init 関数の定義

Lesson_02/SkinModel.cpp(4 行目)

```
void SkinModel::Init(const wchar_t* filePath)
{
  //エフェクトファクトリ。
  DirectX::EffectFactory effectFactory(g_graphicsEngine->GetD3DDevice());
  //テクスチャがあるフォルダを設定する。
  effectFactory.SetDirectory(L"Resource/modelData");
  //CMOファイルのロード。
    //CMOファイルからモデルを作成する関数のCreateFromCMOを実行する。
  m_modelDx = DirectX::Model::CreateFromCMO(
    g_graphicsEngine->GetD3DDevice(), //第一引数はD3Dデバイス。
    filePath,              //第二引数は読み込むCMOファイルのファイルパス。
    effectFactory,           //第三引数はエフェクトファクトリ。
    false              //第四引数はCullモード。今は気にしなくてよい。
  );
}
```

引数で受け取ったファイルパスを使って、cmo ファイルをロードしています。

2.6 SkinModel::UpdateWorldMatrix の定義

Lesson_02/SkinModel.cpp(18 行目)

```
void SkinModel::UpdateWorldMatrix(CVector3 position, CQuaternion rotation, CVector3 scale)
{
  CMatrix transMatrix, rotMatrix, scaleMatrix;
  //平行移動行列を作成する。
  transMatrix.MakeTranslation( position );
  //回転行列を作成する。
  rotMatrix.MakeRotationFromQuaternion( rotation );
  //拡大行列を作成する。
  scaleMatrix.MakeScaling(scale);
  //ワールド行列を作成する。
  //拡大×回転×平行移動の順番で乗算するように！
  //順番を間違えたら結果が変わるよ。
  m_worldMatrix.Mul(scaleMatrix, rotMatrix);
  m_worldMatrix.Mul(m_worldMatrix, transMatrix);
}
```

座標、回転クォータニオン、拡大率からワールド行列を計算しています。行列の乗算順番に注意してください。

2.7 SkinModel::Draw 関数の定義

Lesson_02/SkinModel.cpp(33 行目)

```
void SkinModel::Draw(CMatrix viewMatrix, CMatrix projMatrix)
{
  DirectX::CommonStates state(g_graphicsEngine->GetD3DDevice());
  m_modelDx->Draw(
    g_graphicsEngine->GetD3DDeviceContext(),
    state,
    m_worldMatrix,
    viewMatrix,
    projMatrix
  );
}
```

ワールド行列、ビュー行列、プロジェクション行列を渡して、3D モデルを描画しています。DirectX::CommonStates は気にしなくて構いません。

2.8 章末テスト

下記の URL のテストを行いなさい。

URL

https://goo.gl/forms/bcBKr0UbGprcvsUl2