```
// render target view
ComPtr<ID3D11Texture2D> renderT
// get pointer to back buffer
m_pSwapChain->GetBuffer(0, __u
// create the target view wit
hr = m_pDevice->CreateRenderT
renderTarget->Release();

if (FAILED(hr)) {
    MessageBoxW(hWnd, L"Fai
    exit(0);
}

// depth/stencil buffer d
ComPtr<ID3D11Texture2D> d
D3D11_TEXTURE2D_DESC dep
ZeroMemory(&depthBuffer,

depthBuffer.Width = wid
depthBuffer.Height = h
depthBuffer.MipLevels
depthBuffer.ArraySize
depthBuffer.Format =
depthBuffer.SampleDes
depthBuffer.Usage =
depthBuffer.BindFlag
// create depth/st
if (FAILED(hr)) {
    MessageBoxW(
    exit(0);
}
```

PORTFOLIO

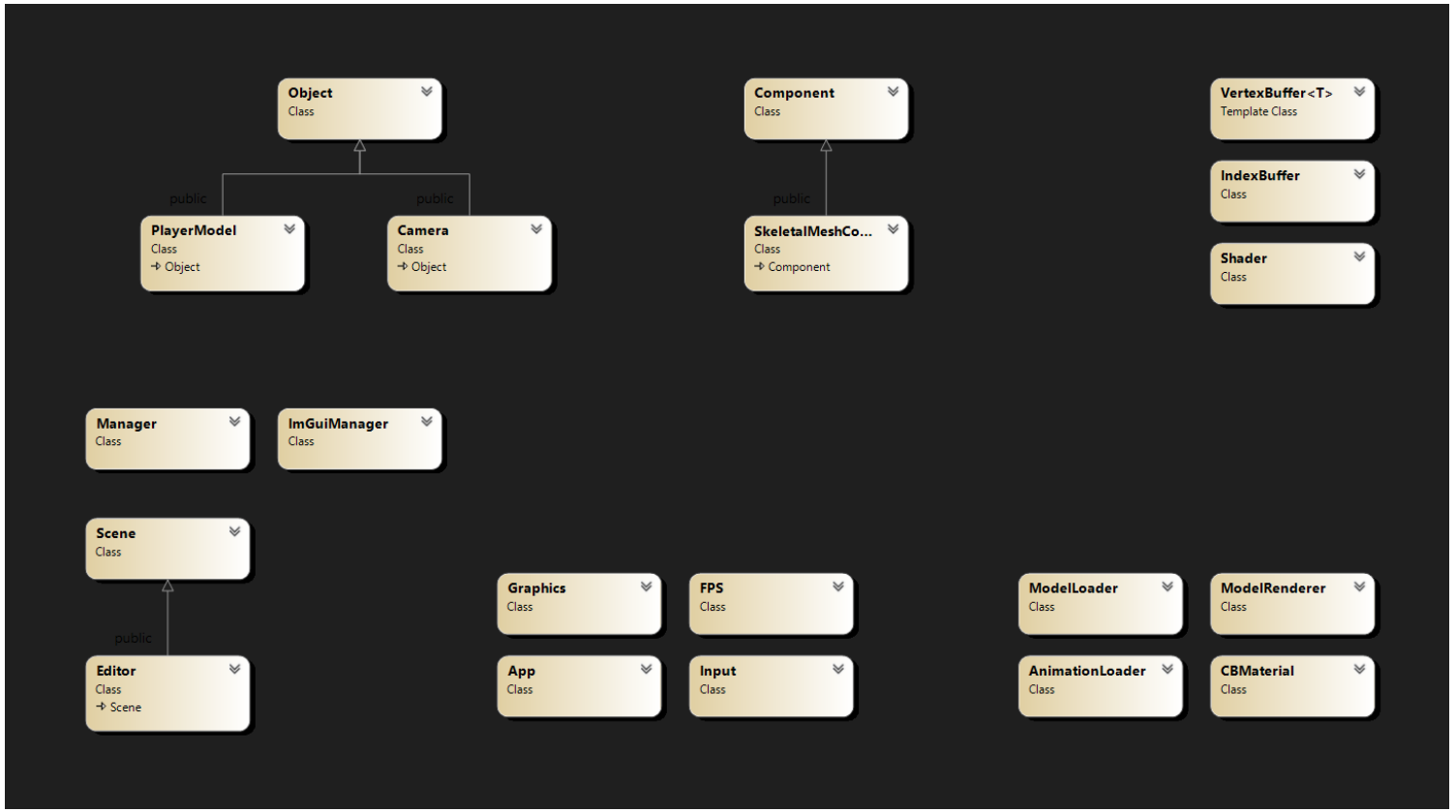
By: ELVIN CUANDRA

作成きっかけ

目的

アニメーションによる理数をより深めたい

工夫点



一から設計したフレームワーク

工夫点

```
class Object {
public:
void InitBase() {
  for (auto& comp : m_component) {
    comp.second->Init();
  }

  Init();
}

protected:
// オブジェクトの純粋仮想関数メソッド
virtual void Init() = 0;

private:
std::map<std::string, std::shared_ptr<Component>> m_component;
}
```

```
class Component {
public:
// オブジェクトの純粋仮想関数メソッド
virtual void Init() = 0;
}
```

オブジェクト＆コンポネントクラス

工夫点

コードの可読性を向上させるために、二つのクラスに切り分け、シーン上のオブジェクトとそのオブジェクトが持ってる各コンポネントの処理を毎フレームに更新

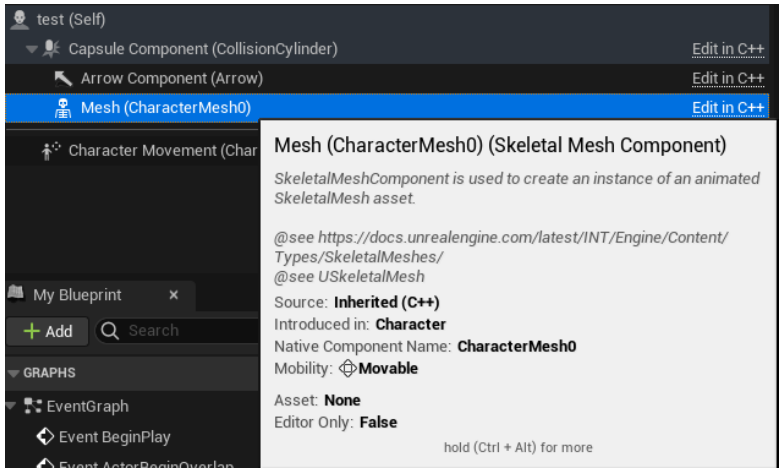
オブジェクト＆コンポネントクラス

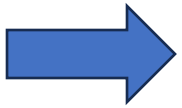
工夫点

```
class PlayerModel: public Object {
public:
virtual void Init() override {
  auto skeletalMeshComponent = CreateComponent<SkeletalMeshComponent>("SkeletalMeshComponent");
  skeletalMeshComponent->LoadModel("assets/model/kachujin.fbx");
  skeletalMeshComponent->LoadAnimation("assets/animations/Run.fbx", "Run");
  }
}
```

PlayerModelクラス

工夫点





```
class SkeletalMeshComponent : public Component
{
private:
std::unique_ptr<ModelLoader> m_pModel;
std::unique_ptr<ModelRenderer> m_pRenderer;
std::unique_ptr<AnimationLoader> m_pAnimation;
std::unique_ptr<Shader> m_pShader;
}
```

Unreal EngineのCharacterクラスを真似し、自分のSkeletal Mesh Component を作成

Skeletal Mesh Component クラス

工夫点

PlayerModelの処理とバックエンドの処理を巻き込ませたくない

Skeletal Mesh Component クラス

工夫点

```
class ModelLoader
```

Assimpでモデルロード、ボーン情報の取得、マテリアルの読み込み処理を担当するクラス

```
class ModelRenderer
```

DirectX11のパイプラインにModelLoaderが用意するデータをGPUに渡す(ボーン行列・マテリアル定数バッファ)

```
class AnimationLoader
```

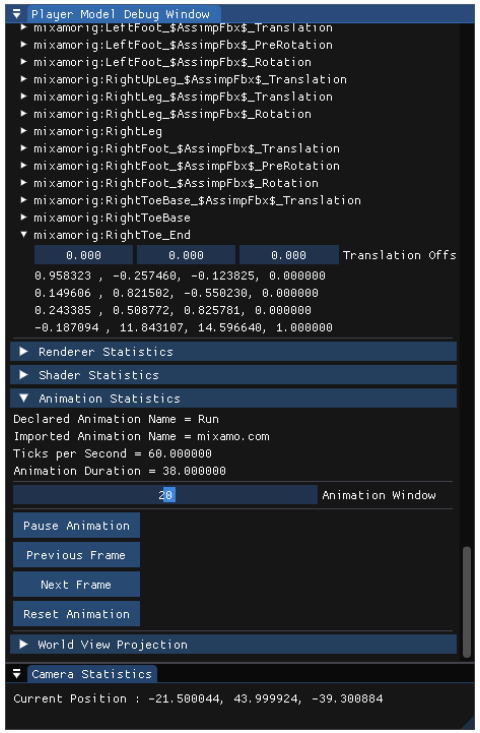
アニメーションデータを読み込んで、ModelLoaderが持ってるデータを更新

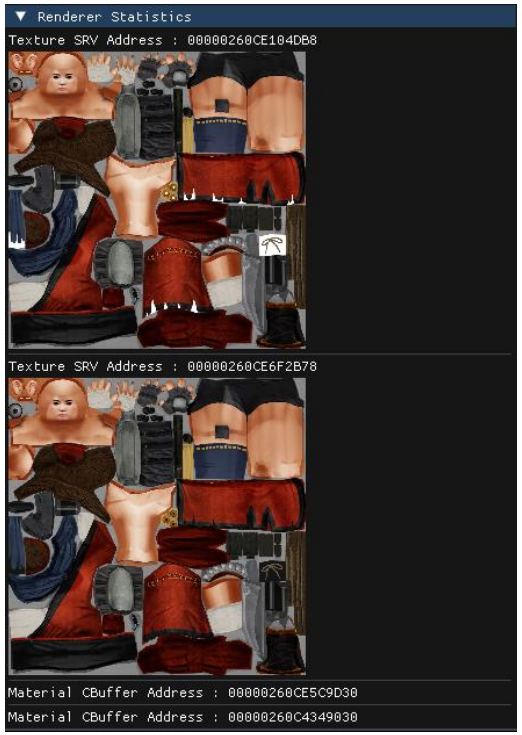
```
class Shader
```

ModelLoaderのVertex3D構造体をもとにIEDを作成し、頂点＆ピックセルシェーダに送る

バックエンド

工夫点





ImGuiを利用し、デバッグしやすい環境を整う（メッシュ情報）

ImGui