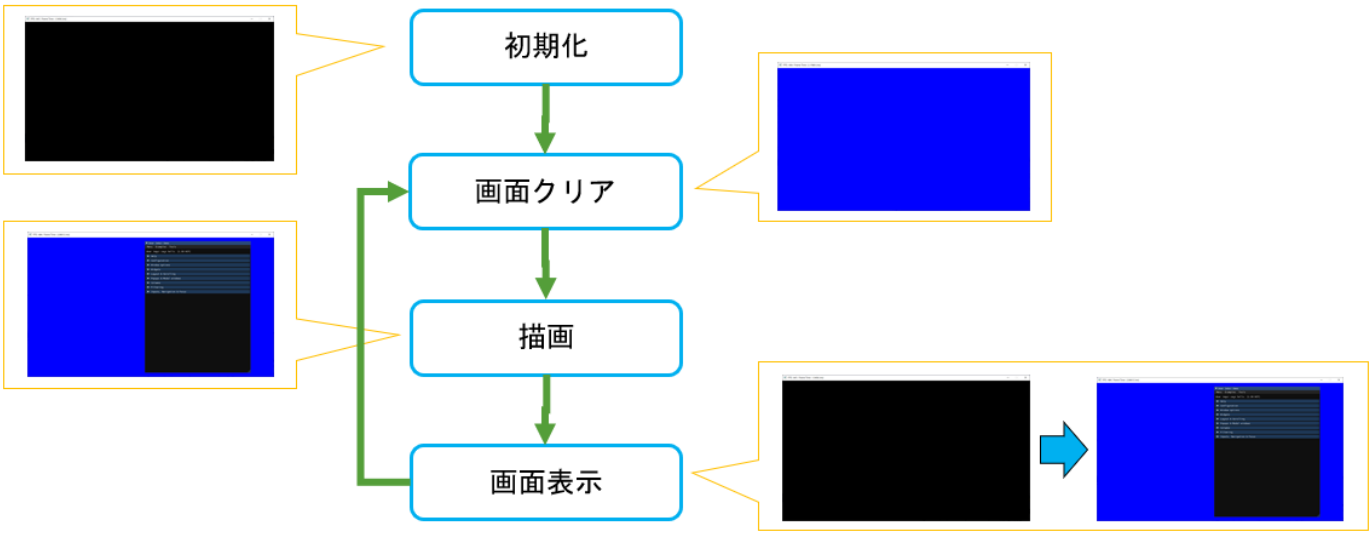
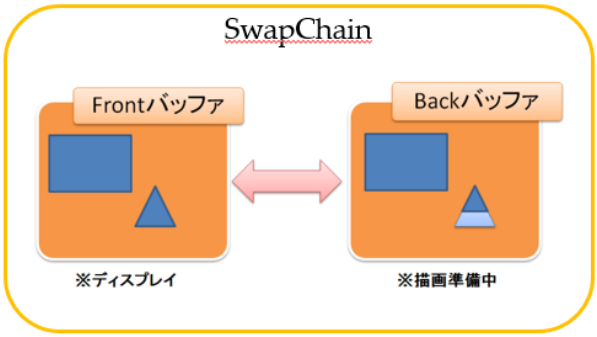
描画エンジン開発 EX

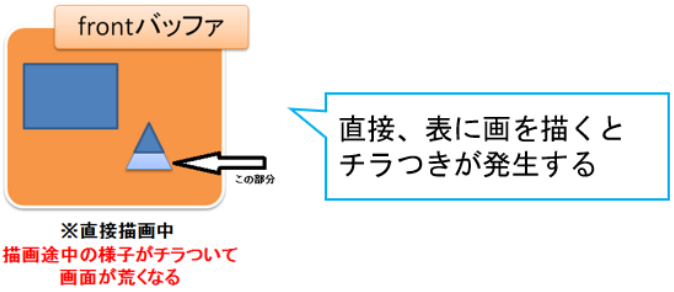
○概要

DirectX11 を利用して CG を画面に表示する。
初期化から画面の表示までは下図のような流れで処理します。



画面の表示は下図のような仕組みになっています。
２枚の画面領域が存在し、表の画面を表示している間に裏の画面に CG を描きます。
裏の画面の描画が完了したら表と裏の画面を入れ替えて裏を表示、表に CG を描くという流れです。





○初期状態の確認

初期状態のプロジェクトを実行し、画面を確認しよう。
下図のように真っ白のウインドウが表示されれば OK です。
この白い部分にプログラムで CG を描いていきます。

○DirectX の初期化

DirectX11 は D3D11CreateDeviceAndSwapChain()関数を呼び出すことによって DirectX11 で最低限必要なオブジェクトが生成されます。

オブジェクト	説明
ID3D11Device	GPU で扱うリソースを生成する
ID3D11DeviceContext	GPU にリソースを渡したり、描画したりする
IDXGISwapChain	描画した内容を画面に表示する

ではグラフィックス関連を統括する Graphics クラスを作成し、DirectX11 を初期化します。
Source フォルダ以下に Graphics.cpp と Graphics.h を作成しましょう。

Graphics.h

```
#pragma once

#include <d3d11.h>
#include <wrl.h>

// グラフィックス
class Graphics
{
```

```
private:
    Graphics() = default;
    ~Graphics() = default;

public:
    // インスタンス取得
    static Graphics& Instance()
    {
        static Graphics instance;
        return instance;
    }

    // 初期化
    void Initialize(HWND hWnd);

    // 画面表示
    void Present(UINT syncInterval);

    // デバイス取得
    ID3D11Device* GetDevice() { return device.Get(); }

    // デバイスコンテキスト取得
    ID3D11DeviceContext* GetDeviceContext() { return immediateContext.Get(); }

    // スクリーン幅取得
    float GetScreenWidth() const { return screenWidth; }

    // スクリーン高さ取得
    float GetScreenHeight() const { return screenHeight; }

private:
    Microsoft::WRL::ComPtr<ID3D11Device>            device;
    Microsoft::WRL::ComPtr<ID3D11DeviceContext>     immediateContext;
    Microsoft::WRL::ComPtr<IDXGISwapChain>          swapchain;

    float    screenWidth;
    float    screenHeight;
};
```

扱いやすいようにシングルトンで取得できるようにする。

ComPtr で DirectX のオブジェクトをスマートポインタとして扱う。

Graphics.cpp

```
#include "Misc.h"
#include "Graphics.h"

// 初期化
void Graphics::Initialize(HWND hWnd)
{
    // 画面のサイズを取得する。
    RECT rc;
    GetClientRect(hWnd, &rc);
    UINT screenWidth = rc.right - rc.left;
    UINT screenHeight = rc.bottom - rc.top;

    this->screenWidth = static_cast<float>(screenWidth);
    this->screenHeight = static_cast<float>(screenHeight);
```

```
    HRESULT hr = S_OK;

    // デバイス＆スワップチェーンの生成
    {
        UINT createDeviceFlags = 0;
#if defined(DEBUG) || defined(_DEBUG)
        createDeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;
#endif
        D3D_FEATURE_LEVEL featureLevels[] =
        {
            D3D_FEATURE_LEVEL_11_0,
            D3D_FEATURE_LEVEL_10_1,
            D3D_FEATURE_LEVEL_10_0,
            D3D_FEATURE_LEVEL_9_3,
            D3D_FEATURE_LEVEL_9_2,
            D3D_FEATURE_LEVEL_9_1,
        };

        // スワップチェーンを作成するための設定オプション
        DXGI_SWAP_CHAIN_DESC swapchainDesc;
        {
            swapchainDesc.BufferDesc.Width = screenWidth;
            swapchainDesc.BufferDesc.Height = screenHeight;
            swapchainDesc.BufferDesc.RefreshRate.Numerator = 60;
            swapchainDesc.BufferDesc.RefreshRate.Denominator = 1;
            swapchainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
            swapchainDesc.BufferDesc.ScanlineOrdering = DXGI_MODE_SCANLINE_ORDER_UNSPECIFIED;
            swapchainDesc.BufferDesc.Scaling = DXGI_MODE_SCALING_UNSPECIFIED;
            swapchainDesc.SampleDesc.Count = 1;
            swapchainDesc.SampleDesc.Quality = 0;
            swapchainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
            swapchainDesc.BufferCount = 1;
            swapchainDesc.OutputWindow = hWnd;
            swapchainDesc.Windowed = TRUE;
            swapchainDesc.SwapEffect = DXGI_SWAP_EFFECT_DISCARD;
            swapchainDesc.Flags = 0;
        }

        D3D_FEATURE_LEVEL featureLevel;

        // デバイス＆スワップチェーンの生成
        hr = D3D11CreateDeviceAndSwapChain(
            nullptr,
            D3D_DRIVER_TYPE_HARDWARE,
            nullptr,
            createDeviceFlags,
            featureLevels,
            ARRAYSIZE(featureLevels),
            D3D11_SDK_VERSION,
            &swapchainDesc,
            swapchain.GetAddressOf(),
            device.GetAddressOf(),
            &featureLevel,
            immediateContext.GetAddressOf()
        );
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
```

```
    }
}

// 画面表示
void Graphics::Present(UINT syncInterval)
{
    swapchain->Present(syncInterval, 0);
}
```

バックバッファの内容を画面に表示する

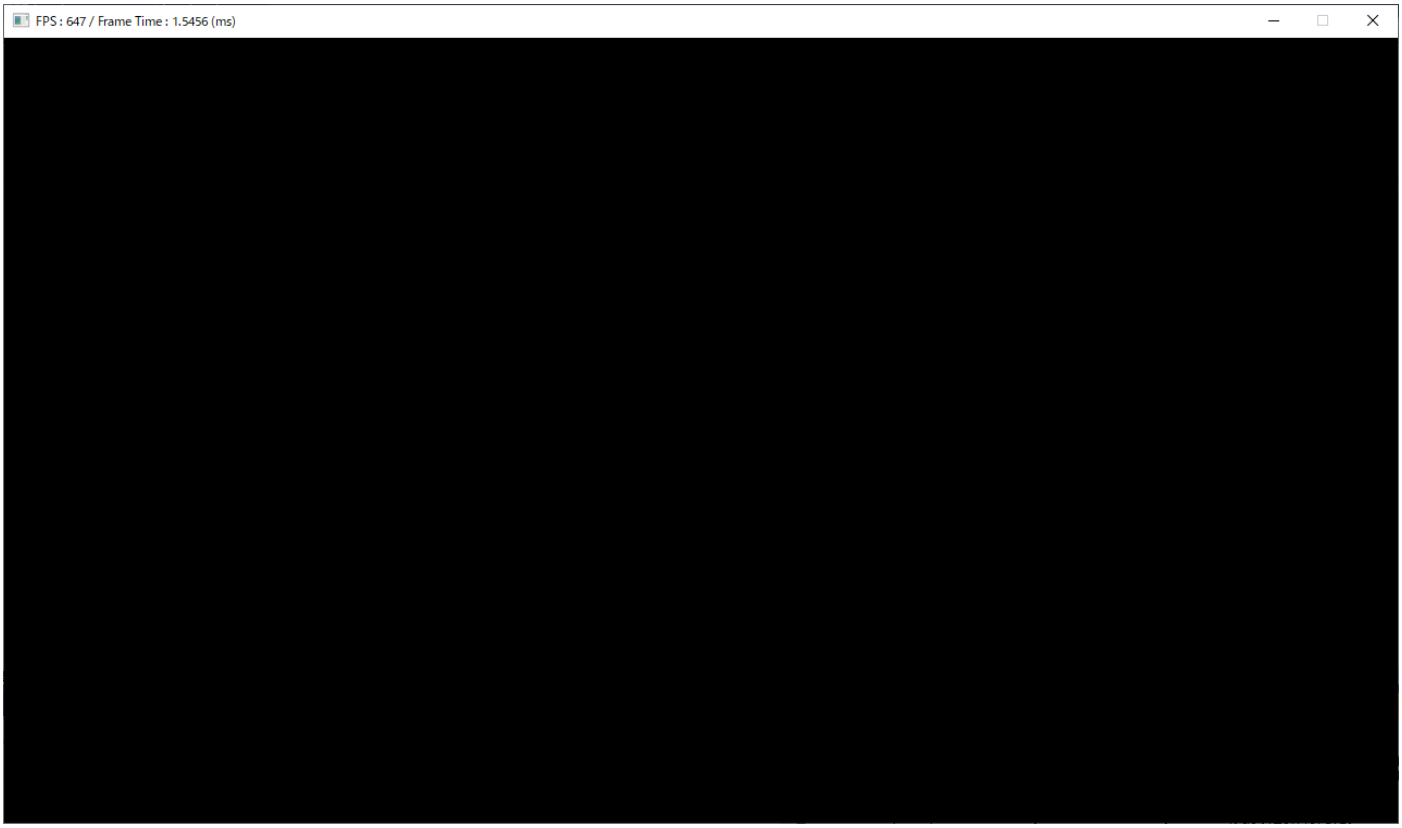
Framework.cpp でグラフィックス初期化プログラムを呼び出しましょう。

Framework.cpp

```
---省略---

#include "Graphics.h"

// 垂直同期間隔設定
static const int syncInterval = 0;

// コンストラクタ
Framework::Framework(HWND hWnd)
    : hWnd(hWnd)
{
    // グラフィックス初期化
    Graphics::Instance().Initialize(hWnd);
}

// 描画処理
void Framework::Render(float elapsedTime)
{
    // 画面表示
    Graphics::Instance().Present(syncInterval);
}
```

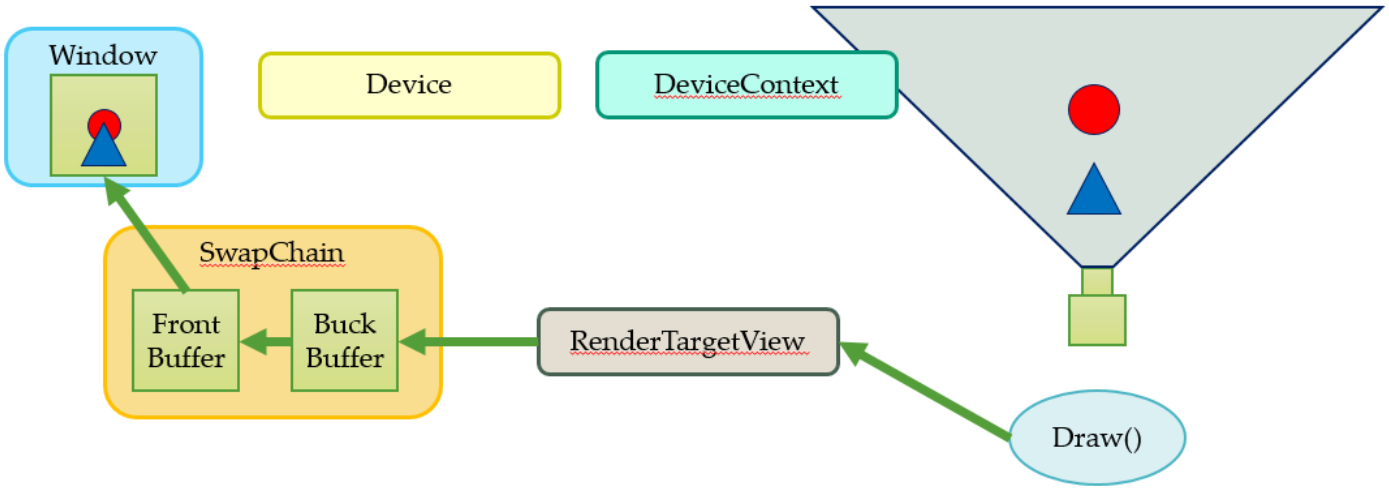
画面を表示する際にフレームの同期をとる
0:可変フレームレート
1:60FPS
2:30FPS
3:20FPS
4:15FPS

実行してみてエラーがでずに下図のように真っ黒なウインドウが表示されれば OK です。



○描画パイプライン

CG を描くためには多くの手順を踏んで画面に表示します。
この画面に表示するまでの描画の流れを描画パイプラインと呼びます。



○画面クリア

グラフィックス用語で画面領域のことフレームバッファと呼びます。
この画面領域を管理する FrameBuffer クラスを作成し、青色にしてみましょう。
FrameBuffer.cpp と FrameBuffer.h を作成しましょう。

FrameBuffer.h

```h
#pragma once

#include <wrl.h>
#include <d3d11.h>

class FrameBuffer
{
public:
    FrameBuffer(ID3D11Device* device, IDXGISwapChain* swapchain);

    // クリア
    void Clear(ID3D11DeviceContext* dc, float r, float g, float b, float a);

private:
    Microsoft::WRL::ComPtr<ID3D11RenderTargetView>      renderTargetView;
    D3D11_VIEWPORT                                      viewport;
};
```

FrameBuffer.cpp

```cpp
#include "Misc.h"
#include "FrameBuffer.h"

// コンストラクタ
FrameBuffer::FrameBuffer(ID3D11Device* device, IDXGISwapChain* swapchain)
{
    HRESULT hr = S_OK;

    UINT width, height;

    // レンダーターゲットビューの生成
    {
        // スワップチェーンからバックバッファテクスチャを取得する。
        // ※スワップチェーンに内包されているバックバッファテクスチャは'色'を書き込むテクスチャ。
        Microsoft::WRL::ComPtr<ID3D11Texture2D> texture2d;
        hr = swapchain->GetBuffer(
            0,
            __uuidof(ID3D11Texture2D),
            reinterpret_cast<void**>(texture2d.GetAddressOf()));
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // バックバッファテクスチャへの書き込みの窓口となるレンダーターゲットビューを生成する。
        hr = device->CreateRenderTargetView(texture2d.Get(), nullptr, renderTargetView.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // バックバッファテクスチャからサイズ情報を取得1
        D3D11_TEXTURE2D_DESC texture2dDesc;
        texture2d->GetDesc(&texture2dDesc);

        width = texture2dDesc.Width;
        height = texture2dDesc.Height;
    }
```

テクスチャに直接
書き込みはできないので
ビューと呼ばれる
窓口を通して描く必要がある

```
    // ビューポート
    {
        viewport.Width = static_cast<float>(width);
        viewport.Height = static_cast<float>(height);
        viewport.MinDepth = 0.0f;
        viewport.MaxDepth = 1.0f;
        viewport.TopLeftX = 0.0f;
        viewport.TopLeftY = 0.0f;
    }
}

// クリア
void FrameBuffer::Clear(ID3D11DeviceContext* dc, float r, float g, float b, float a)
{
    float color[4]{ r, g, b, a };
    dc->ClearRenderTargetView(renderTargetView.Get(), color);
}
```

レンダーターゲットビューを通して
バックバッファの色をクリアする

Graphics.h

```
---省略---

#include <memory>
#include "FrameBuffer.h"

// グラフィックス
class Graphics
{
    ---省略---
public:
    ---省略---

    // フレームバッファ取得
    FrameBuffer* GetFrameBuffer() { return frameBuffer.get(); }

private:
    ---省略---

    std::unique_ptr<FrameBuffer>   frameBuffer;
};
```

Graphics.cpp

```
---省略---

// 初期化
void Graphics::Initialize(HWND hWnd)
{
    ---省略---

    // フレームバッファ作成
    frameBuffer = std::make_unique<FrameBuffer>(device.Get(), swapchain.Get());
}
```

Framework.cpp

```
---省略---

// 描画処理
void Framework::Render(float elapsedTime)
{
    ID3D11DeviceContext* dc = Graphics::Instance().GetDeviceContext();

    // 画面クリア
    Graphics::Instance().GetFrameBuffer()->Clear(dc, 0, 0, 1, 1);

    // 画面表示
    Graphics::Instance().Present(syncInterval);
}
```

実行して画面を確認してみましょう。
下図のように画面が青色になっていれば OK です。

○デバッグメニュー描画

フレームバッファに CG を描画するにはレンダーターゲットを設定する必要があります。
今回はレンダーターゲットの設定を行い、デバッグメニューを表示します。

デバッグメニューの描画にはIMGUIという高品質なデバッグメニューライブラリを利用します。このプロジェクトではIMGUIを導入するプログラムをあらかじめ用意しているので使ってみましょう。

FrameBuffer.h

```
---省略---

class FrameBuffer
{
public:
	---省略---

	// レンダーターゲット設定
	void SetRenderTargets(ID3D11DeviceContext* dc);

};
```

FrameBuffer.cpp

```
---省略---

// レンダーターゲット設定
void FrameBuffer::SetRenderTargets(ID3D11DeviceContext* dc)
{
	dc->RSSetViewports(1, &viewport);
	dc->OMSetRenderTargets(1, renderTargetView.GetAddressOf(), nullptr);
}
```

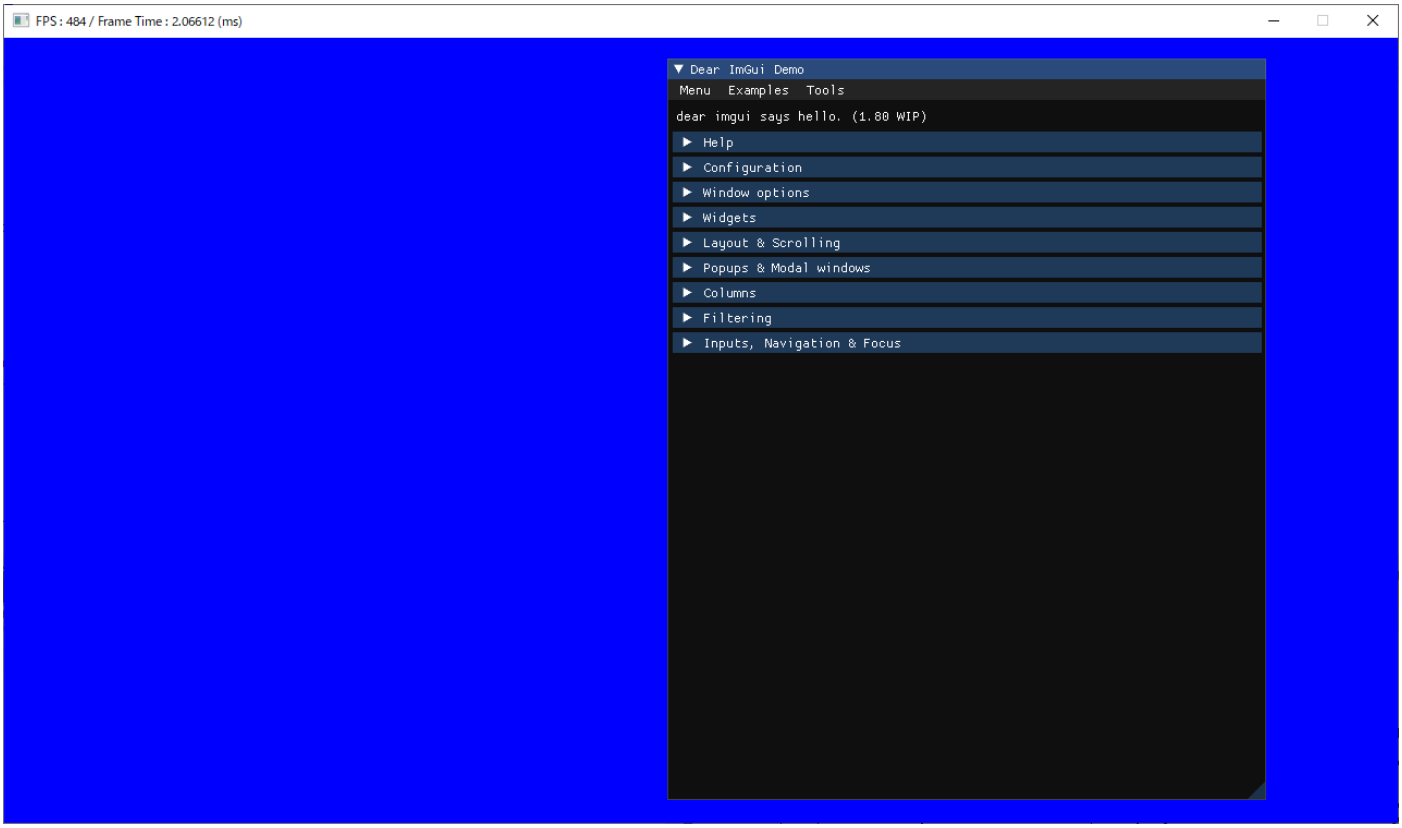
レンダーターゲットビューを通して
バックバッファに CG を描く

Framework.cpp

```
---省略---
#include <imgui.h>
#include "ImGuiRenderer.h"

// コンストラクタ
Framework::Framework(HWND hWnd)
	: hWnd(hWnd)
{
	---省略---

	// IMGUI初期化
	ImGuiRenderer::Initialize(hWnd,Graphics::Instance().GetDevice(),Graphics::Instance().GetDeviceContext());

}

// デストラクタ
Framework::~Framework()
{
	// IMGUI終了化
	ImGuiRenderer::Finalize();
```

```
}

// 描画処理
void Framework::Render(float elapsedTime)
{
    ID3D11DeviceContext* dc = Graphics::Instance().GetDeviceContext();

    // IMGUIフレーム開始処理
    ImGuiRenderer::NewFrame();

    // 画面クリア
    Graphics::Instance().GetFrameBuffer()->Clear(dc, 0, 0, 1, 1);

    // レンダーターゲット設定
    Graphics::Instance().GetFrameBuffer()->SetRenderTargets(dc);
#if 1
    // IMGUIデモウインドウ描画（IMGUI機能テスト用）
    ImGui::ShowDemoWindow();
#endif
    // IMGUI描画
    ImGuiRenderer::Render(dc);

    // 画面表示
    Graphics::Instance().Present(syncInterval);
}

---省略---

// メッセージハンドラ
LRESULT CALLBACK Framework::HandleMessage(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    if (ImGuiRenderer::HandleMessage(hWnd, msg, wParam, lParam))
        return true;

    ---省略---
}
```

実行確認してみましょう。
下図のようにデバッグメニューが表示されていれば OK です。



ImGui::ShowDemoWindow()を呼び出すことで IMGUI で表現できる様々なメニューやグラフを確認することができます。
今後のゲーム開発に役立てるためにも一通り覗いてみましょう。