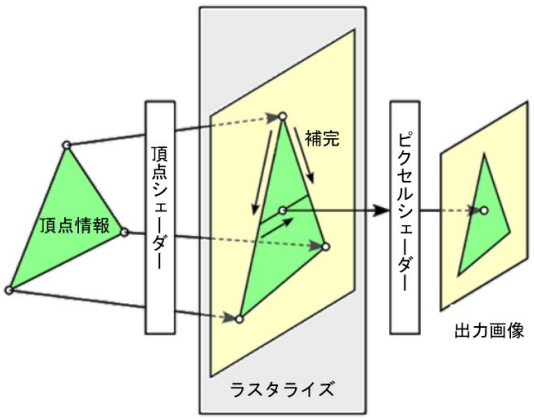
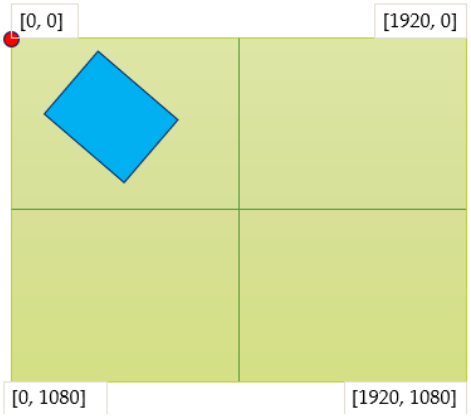
描画エンジン開発 EX

○概要

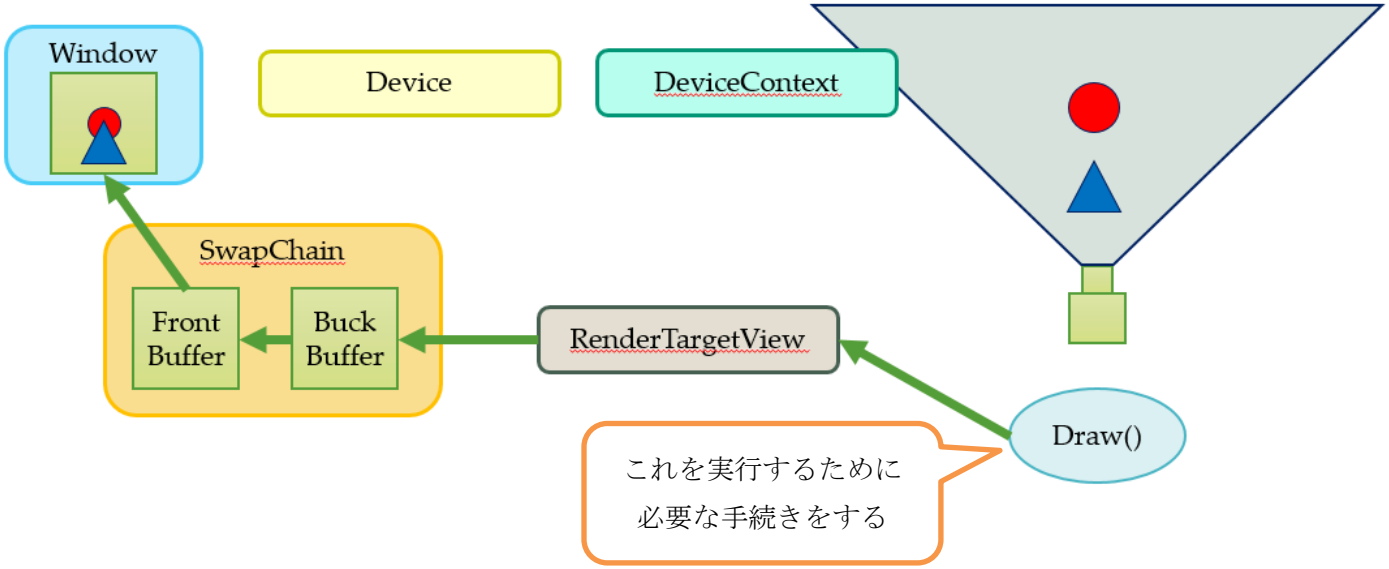
描画パイプラインを把握し、四角形ポリゴンを描画する。
四角形ポリゴンをスクリーン空間での座標指定で描画できるようにする。
四角形ポリゴンを四角形の中心を軸に回転できるようにする。

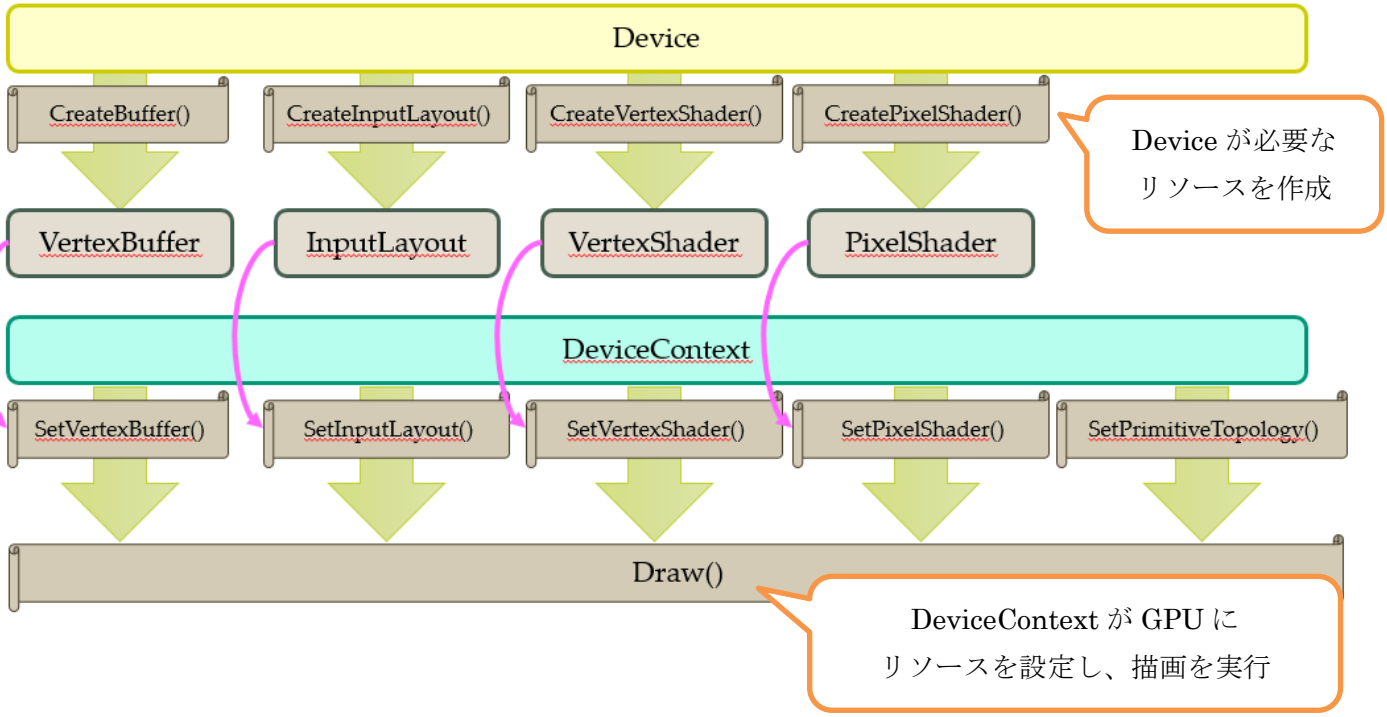




○描画パイプライン

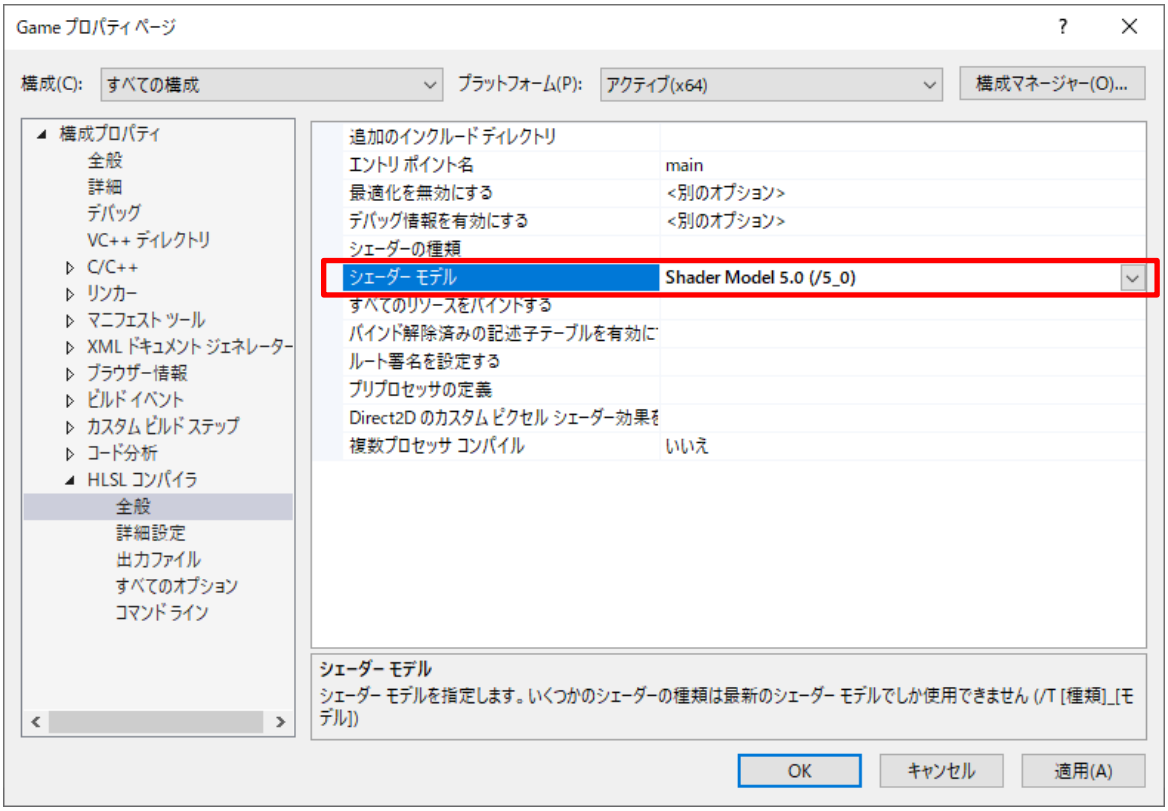
描画するまでの流れは下図のような流れになっている。

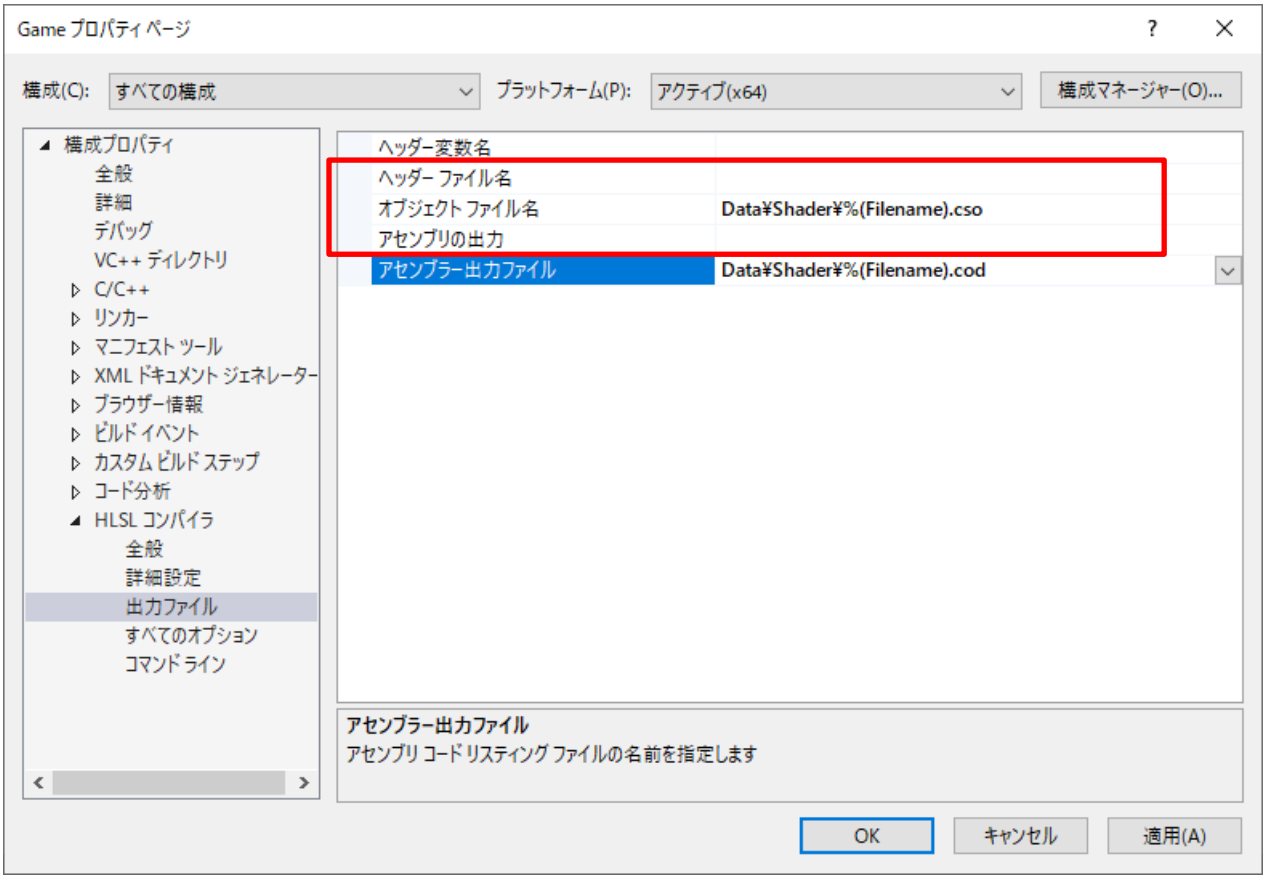




○シェーダー

シェーダーとは描画するポリゴンの頂点位置や色を計算するプログラムのことです。
Shader フォルダ以下に Sprite.hlsli と SpriteVS.hlsl と SpritePS.hlsl を作成しましょう。
作成した後、Visual Studio のプロジェクトの設定で下図のようになっているか確認しましょう。





オブジェクトファイル名：Data¥Shader¥%(Filename).cso
アセンブラー出力ファイル：Data¥Shader¥%(Filename).cod

Sprite.hlsli

```
// 頂点シェーダー出力データ
struct VS_OUT
{
    float4 position : SV_POSITION;
    float4 color    : COLOR;
};
```

SpriteVS.hlsl

```
#include "Sprite.hlsli"

// 頂点シェーダーエントリポイント
VS_OUT main(float4 position : POSITION, float4 color : COLOR)
{
    VS_OUT vout;
    vout.position = position;
    vout.color = color;

    return vout;
}
```

後で実装するcpp側の記述と対応する

```
D3D11_INPUT_ELEMENT_DESC inputElementDesc[] =
{
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, ～    },
    { "COLOR",    0, DXGI_FORMAT_R32G32B32A32_FLOAT,～ },
};
```

SpritePS.hlsl

```
#include "sprite.hlsli"

// ピクセルシェーダーエントリポイント
float4 main(VS_OUT pin) : SV_TARGET
{
    return pin.color;
}
```

プログラムをビルドしてみましょう。
コンパイルしてエラーがなければ Data¥Shader フォルダ以下に SpriteVS.cso と SpritePS.cso ファイルが出力されていることを確認しましょう。

○スプライト

スプライトとは 2D ポリゴンのことです。
2D ポリゴンを描画する Sprite クラスを作成します。
Sprite.cpp と Sprite.h を作成しましょう。

Sprite.h

```
#pragma once

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

// スプライト
class Sprite
{
public:
    Sprite(ID3D11Device* device);

    // 頂点データ
    struct Vertex
    {
        DirectX::XMFLOAT3 position;
        DirectX::XMFLOAT4 color;
    };

    // 描画実行
    void Render(ID3D11DeviceContext* dc) const;

private:
    Microsoft::WRL::ComPtr<ID3D11VertexShader>          vertexShader;
    Microsoft::WRL::ComPtr<ID3D11PixelShader>           pixelShader;
    Microsoft::WRL::ComPtr<ID3D11InputLayout>           inputLayout;

    Microsoft::WRL::ComPtr<ID3D11Buffer>                vertexBuffer;
};
```

Sprite.cpp

```cpp
#include <fstream>
#include "Sprite.h"
#include "Misc.h"

// コンストラクタ
Sprite::Sprite(ID3D11Device* device)
{
    HRESULT hr = S_OK;

    // 頂点バッファの生成
    {
        // 頂点バッファを作成するための設定オプション
        D3D11_BUFFER_DESC buffer_desc = {};
        buffer_desc.ByteWidth = sizeof(Vertex) * 4;
        buffer_desc.Usage = D3D11_USAGE_DYNAMIC;
        buffer_desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
        buffer_desc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
        buffer_desc.MiscFlags = 0;
        buffer_desc.StructureByteStride = 0;
        // 頂点バッファオブジェクトの生成
        hr = device->CreateBuffer(&buffer_desc, nullptr, vertexBuffer.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }

    // 頂点シェーダー
    {
        std::ifstream fin("Data/Shader/SpriteVS.cso", std::ios::in | std::ios::binary);

        // ファイルサイズを求める
        fin.seekg(0, std::ios_base::end);
        long long size = fin.tellg();
        fin.seekg(0, std::ios_base::beg);

        // 読み込み
        std::unique_ptr<char[]> data = std::make_unique<char[]>(size);
        fin.read(data.get(), size);

        // 頂点シェーダー生成
        HRESULT hr = device->CreateVertexShader(data.get(), size, nullptr, vertexShader.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

        // 入力レイアウト
        D3D11_INPUT_ELEMENT_DESC inputElementDesc[] =
        {
            { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT,    0, D3D11_APPEND_ALIGNED_ELEMENT,
                                                                D3D11_INPUT_PER_VERTEX_DATA, 0 },
            { "COLOR",    0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT,
                                                                D3D11_INPUT_PER_VERTEX_DATA, 0 },
        };
```

4 頂点分の頂点バッファを生成。
D3D11_USAGE_DYNAMIC,
D3D11_CPU_ACCESS_WRITE
を指定することで毎フレーム
頂点を編集できるようになる。

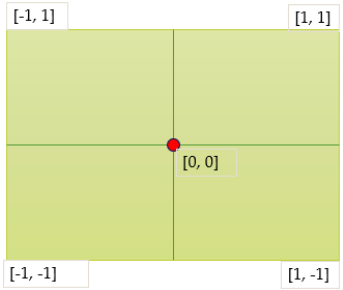
RGB の 3 つの要素を Float で構成する。
つまり FLOAT3

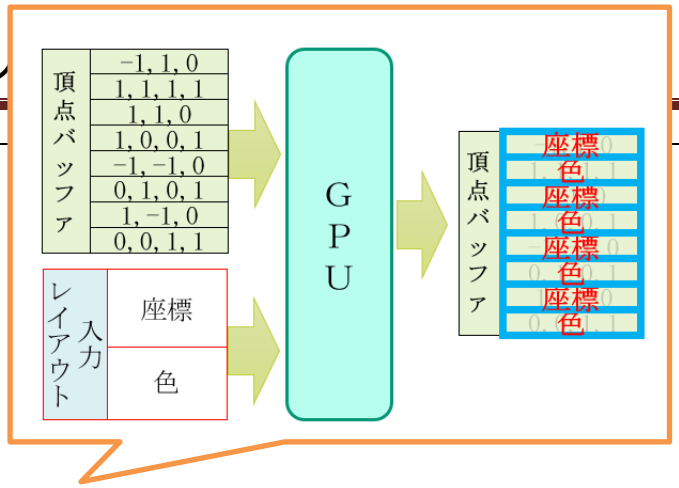
頂点データの内容と対応させる。
シェーダーに渡す頂点データの内容を教えるための設定

```cpp
struct Vertex
{
    DirectX::XMFLOAT3 position;
    DirectX::XMFLOAT4 color;
};
```

```
        hr = device->CreateInputLayout(inputElementDesc, ARRAYSIZE(inputElementDesc),
                                       data.get(), size, inputLayout.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }

    // ピクセルシェーダー
    {
        std::ifstream fin("Data/Shader/SpritePS.cso",
                          std::ios::in | std::ios::binary);

        // ファイルサイズを求める
        fin.seekg(0, std::ios_base::end);
        long long size = fin.tellg();
        fin.seekg(0, std::ios_base::beg);

        // 読み込み
        std::unique_ptr<char[]> data = std::make_unique<char[]>(size);
        fin.read(data.get(), size);

        // ピクセルシェーダー生成
        HRESULT hr = device->CreatePixelShader(data.get(), size, nullptr, pixelShader.GetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));
    }
}

// 描画実行
void Sprite::Render(ID3D11DeviceContext* dc) const
{
    // 頂点座標
    DirectX::XMFLOAT2 positions[] = {
        DirectX::XMFLOAT2(-0.5f, +0.5f),        // 左上
        DirectX::XMFLOAT2(+0.5f, +0.5f),        // 右上
        DirectX::XMFLOAT2(-0.5f, -0.5f),        // 左下
        DirectX::XMFLOAT2(+0.5f, -0.5f),        // 右下
    };
    // 頂点カラー
    DirectX::XMFLOAT4 colors[] = {
        DirectX::XMFLOAT4(1, 0, 0, 1),          // 左上
        DirectX::XMFLOAT4(0, 1, 0, 1),          // 右上
        DirectX::XMFLOAT4(0, 0, 1, 1),          // 左下
        DirectX::XMFLOAT4(1, 1, 1, 1),          // 右下
    };

    // 頂点バッファの内容の編集を開始する。
    D3D11_MAPPED_SUBRESOURCE mappedSubresource;
    HRESULT hr = dc->Map(vertexBuffer.Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedSubresource);
    _ASSERT_EXPR(SUCCEEDED(hr), HRTrace(hr));

    // 頂点バッファの内容を編集
    Vertex* v = static_cast<Vertex*>(mappedSubresource.pData);
    for (int i = 0; i < 4; ++i)
    {
        v[i].position.x = positions[i].x;
        v[i].position.y = positions[i].y;
        v[i].position.z = 0.0f;

        v[i].color.x = colors[i].x;
```

画面に描画するときは -1.0～1.0 の範囲内で指定する必要がある。
この空間を NDC 空間と呼ぶ






```
        v[i].color.y = colors[i].y;
        v[i].color.z = colors[i].z;
        v[i].color.w = colors[i].w;
    }

    // 頂点バッファの内容の編集を終了する。
    dc->Unmap(vertexBuffer.Get(), 0);

    // GPUに描画するためのデータを渡す
    UINT stride = sizeof(Vertex);
    UINT offset = 0;
    dc->IASetVertexBuffers(0, 1, vertexBuffer.GetAddressOf(), &stride, &offset);
    dc->IASetInputLayout(inputLayout.Get());
    dc->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    dc->VSSetShader(vertexShader.Get(), nullptr, 0);
    dc->PSSetShader(pixelShader.Get(), nullptr, 0);

    // 描画
    dc->Draw(4, 0);
}
```

今後、様々な描画テストをするためのシーンを作成します。
Scene.cpp と Scene.h を作成しましょう。

Scene.h

```
#pragma once

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

// シーン基底
class Scene
{
public:
    Scene() = default;
    virtual ~Scene() = default;

    // 更新処理
    virtual void Update(float elapsedTime) {}

    // 描画処理
    virtual void Render(float elapsedTime) {}
};

// スプライトテストシーン
class SpriteTestScene : public Scene
{
public:
    SpriteTestScene();
    ~SpriteTestScene() override = default;

    // 描画処理
    void Render(float elapsedTime) override;
```

```
private:
    std::unique_ptr<Sprite>    sprites[8];
};
```

Scene.cpp

```
#include "Scene.h"
#include "Graphics.h"

// コンストラクタ
SpriteTestScene::SpriteTestScene()
{
    ID3D11Device* device = Graphics::Instance().GetDevice();

    sprites[0] = std::make_unique<Sprite>(device);
}

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

    sprites[0]->Render(dc);
}
```

作成したシーンを Framework で作成し、呼び出しましょう。

Framework.h

```
---省略---
#include "Scene.h"

class Framework
{
public:
    ---省略---

private:
    ---省略---
    std::unique_ptr<Scene>scene;
};
```

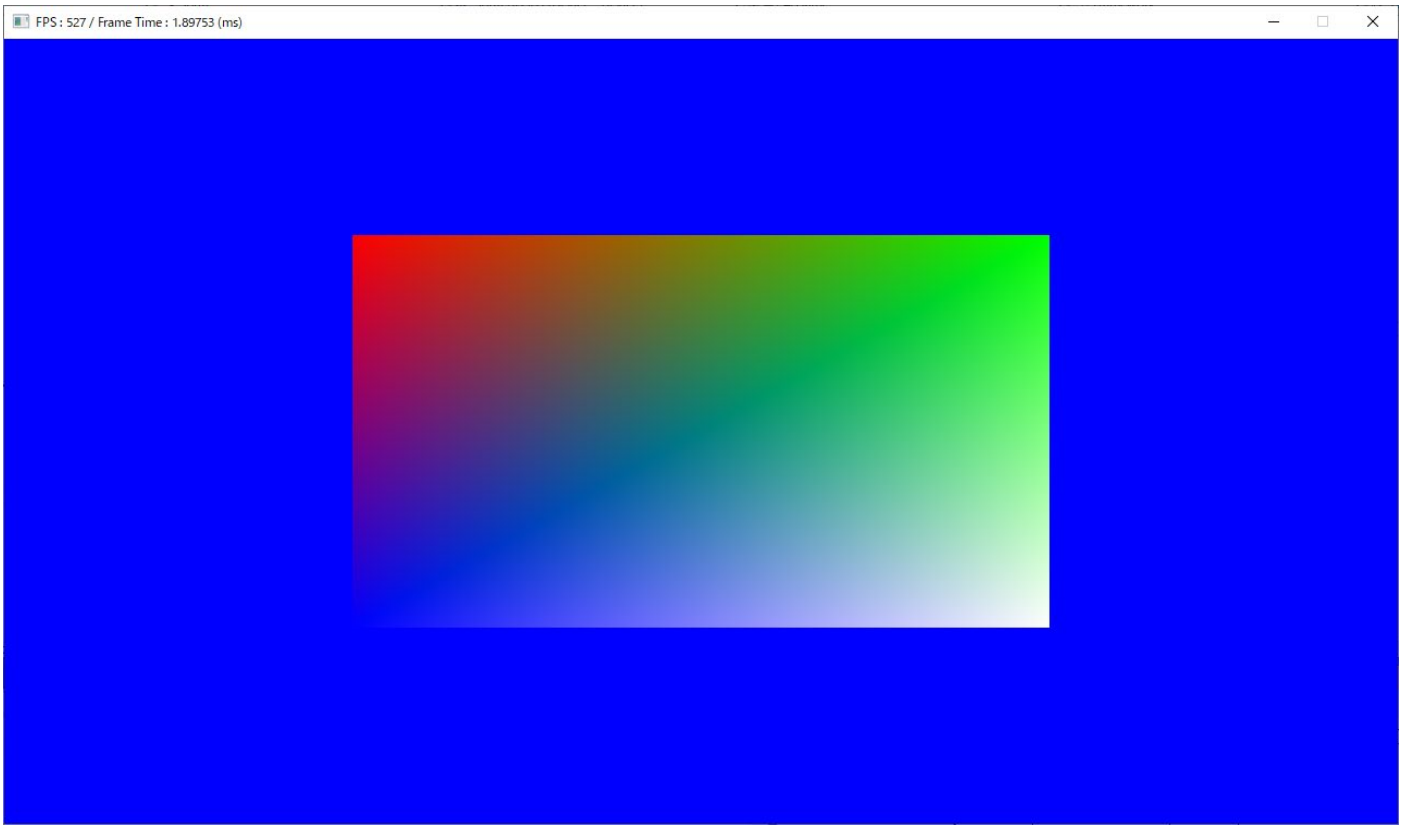
Framework.cpp

```
---省略---

// コンストラクタ
Framework::Framework(HWND hWnd)
{
    ---省略---
    // シーン初期化
```

```
    scene = std::make_unique<SpriteTestScene>();
}

// 更新処理
void Framework::Update(float elapsedTime)
{
    // シーン更新処理
    scene->Update(elapsedTime);
}

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

    // シーン描画処理
    scene->Render(elapsedTime);

#if 0
    // IMGUIデモウインドウ描画（IMGUI機能テスト用）
    ImGui::ShowDemoWindow();
#endif
    ---省略---
}
```

デモウインドウは邪魔なので
非表示にしておく

実行確認してみましょう。
下図のように色がグラデーションされた四角形が表示されていれば OK です。



○スクリーン空間

スプライトをスクリーン空間で描画できるようにします。

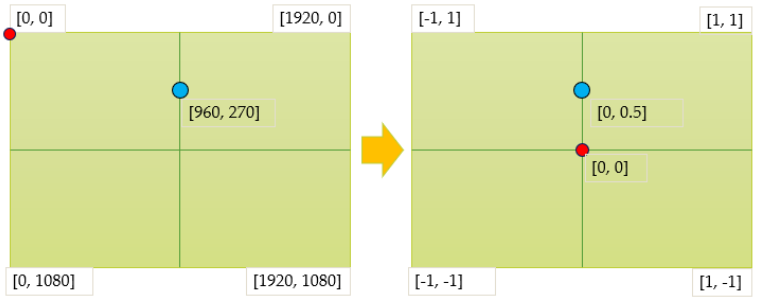
Sprite.h

```
---省略---

// スプライト
class Sprite
{
public:
    ---省略---

    // 描画実行
    void Render(ID3D11DeviceContext* dc) const;
    void Render(ID3D11DeviceContext* dc,
        float dx, float dy,                 // 左上位置
        float dw, float dh,                 // 幅、高さ
        float r, float g, float b, float a // 色
    ) const;

    ---省略---
};
```

Sprite.cpp

```
---省略---

// 描画実行
void Sprite::Render(ID3D11DeviceContext* dc,
    float dx, float dy,                 // 左上位置
    float dw, float dh,                 // 幅、高さ
    float r, float g, float b, float a // 色
) const
{
    // 頂点座標
    DirectX::XMFLOAT2 positions[] = {
        DirectX::XMFLOAT2(-0.5f, +0.5f),        // 左上
        DirectX::XMFLOAT2(+0.5f, +0.5f),        // 右上
        DirectX::XMFLOAT2(-0.5f, -0.5f),        // 左下
        DirectX::XMFLOAT2(+0.5f, -0.5f),        // 右下

        DirectX::XMFLOAT2(dx,      dy),         // 左上
        DirectX::XMFLOAT2(dx + dw, dy),         // 右上
        DirectX::XMFLOAT2(dx,      dy + dh),    // 左下
        DirectX::XMFLOAT2(dx + dw, dy + dh),    // 右下
    };
    // 頂点カラー
    DirectX::XMFLOAT4 colors[] = {
        DirectX::XMFLOAT4(1, 0, 0, 1),          // 左上
        DirectX::XMFLOAT4(0, 1, 0, 1),          // 右上
        DirectX::XMFLOAT4(0, 0, 1, 1),          // 左下
        DirectX::XMFLOAT4(1, 1, 1, 1),          // 右下
```

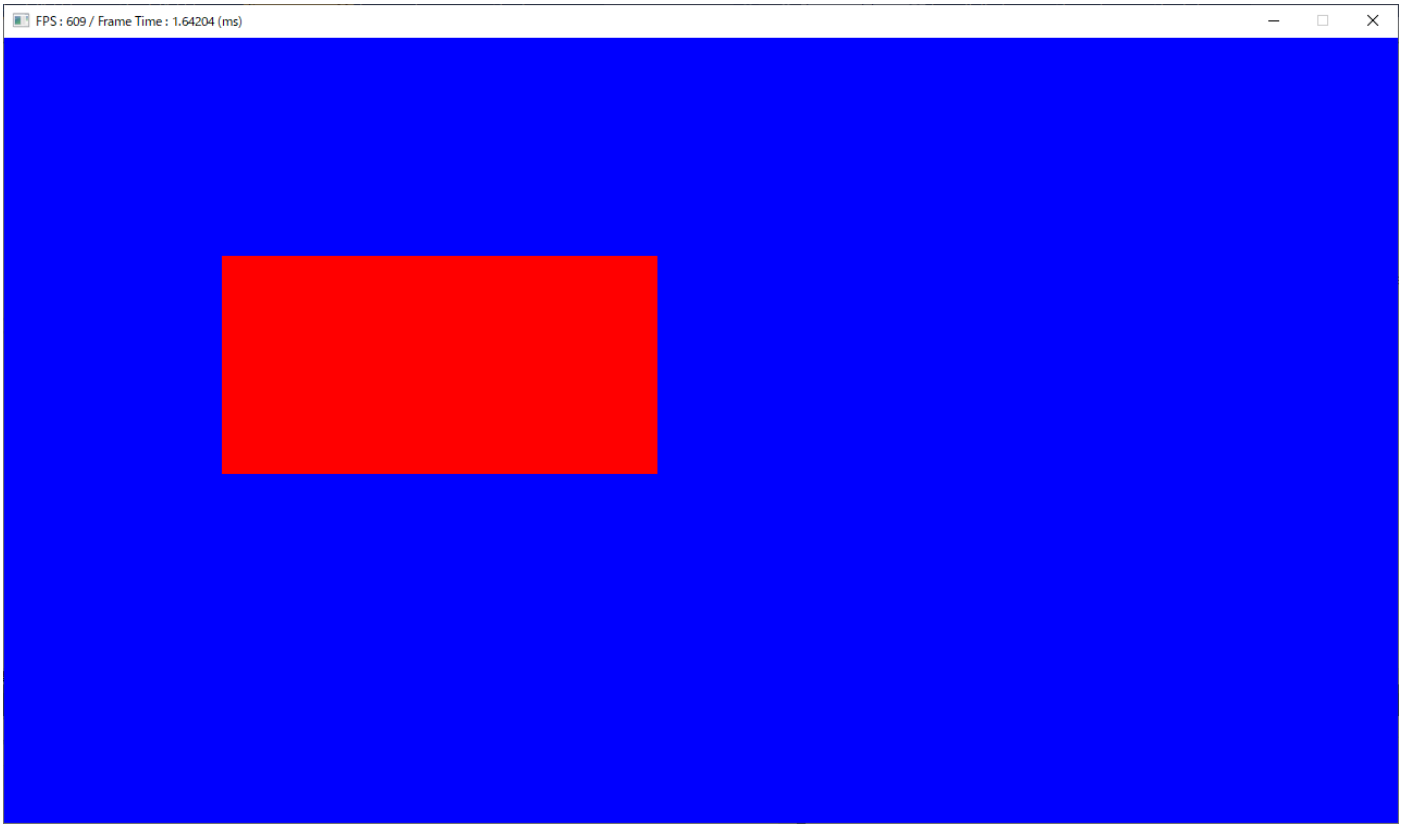
```
    };

    // 現在設定されているビューポートからスクリーンサイズを取得する。
    D3D11_VIEWPORT viewport;
    UINT numViewports = 1;
    dc->RSGetViewports(&numViewports, &viewport);
    float screenWidth = viewport.Width;
    float screenHeight = viewport.Height;

    // スクリーン座標系からNDC座標系へ変換する。
    for (DirectX::XMFLOAT2& p : positions)
    {
        p.x = 2.0f * p.x / screenWidth - 1.0f;
        p.y = 1.0f - 2.0f * p.y / screenHeight;
    }

    // 頂点バッファの内容の編集を開始する。
    ---省略---

    // 頂点バッファの内容を編集
    Vertex* v = static_cast<Vertex*>(mappedSubresource.pData);
    for (int i = 0; i < 4; ++i)
    {
        ---省略---
        v[i].color.x = colors[i].x;
        v[i].color.y = colors[i].y;
        v[i].color.z = colors[i].z;
        v[i].color.w = colors[i].w;

        v[i].color.x = r;
        v[i].color.y = g;
        v[i].color.z = b;
        v[i].color.w = a;
    }

    ---省略---
}
```



Scene.cpp

```
---省略---

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

    sprites[0]->Render(dc, 200, 200, 400, 200, 1, 0, 0, 1);
}
```

実行確認してみましょう。
下図のようにスクリーン空間で指定した位置とサイズのスプライトが表示されていれば OK です。



○回転

スプライトの中心を軸に回転できるようにします。

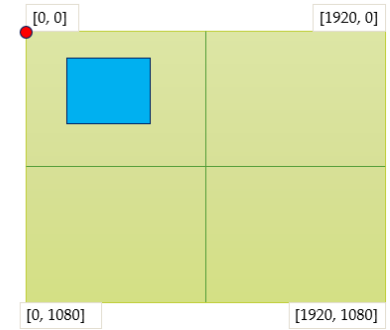
Sprite.h

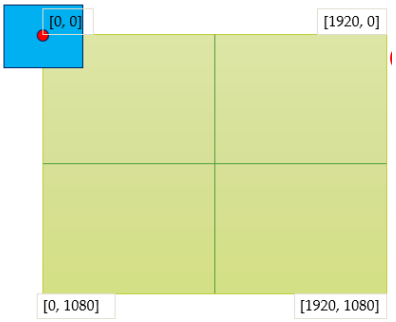
```
---省略---

// スプライト
class Sprite
{
public:
    ---省略---

    // 描画実行
    void Render(ID3D11DeviceContext* dc,
        float dx, float dy,                 // 左上位置
        float dw, float dh,                 // 幅、高さ
        float angle,                        // 角度
        float r, float g, float b, float a // 色
    ) const;

    ---省略---
};
```

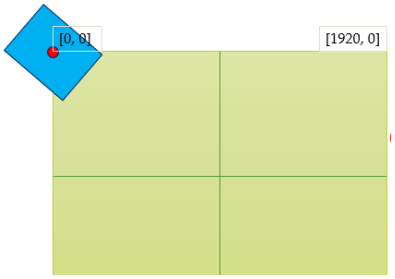
Render 関数を改造

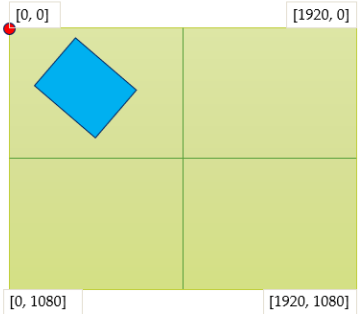
Sprite.cpp

```
// 描画実行
void Sprite::Render(ID3D11DeviceContext* dc,
    float dx, float dy,                 // 左上位置
    float dw, float dh,                 // 幅、高さ
    float angle,                        // 角度
    float r, float g, float b, float a // 色
) const
{
    // 頂点座標
    ---省略---

    // スプライトの中心で回転させるために４頂点の中心位置が
    // 原点(0, 0)になるように一旦頂点を移動させる。
    float mx = dx + dw * 0.5f;
    float my = dy + dh * 0.5f;
    for (auto& p : positions)
    {
        p.x -= mx;
        p.y -= my;
    }

    // 頂点を回転させる
    float theta = DirectX::XMConvertToRadians(angle);
    float c = cosf(theta);
    float s = sinf(theta);
    for (auto& p : positions)
    {
        DirectX::XMFLOAT2 r = p;
        p.x = c * r.x + -s * r.y;
        p.y = s * r.x + c * r.y;
    }

    // 回転のために移動させた頂点を元の位置に戻す
    for (auto& p : positions)
    {
        p.x += mx;
        p.y += my;
    }

    // 現在設定されているビューポートからスクリーンサイズを取得する。
    ---省略---
}
```









Scene.cpp

```
---省略---

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

    sprites[0]->Render(dc, 200, 200, 400, 200, 45, 1, 0, 0, 1);
}
```

実行確認してみましょう。
下図のように 45 度回転したスプライトが表示されていれば OK です。